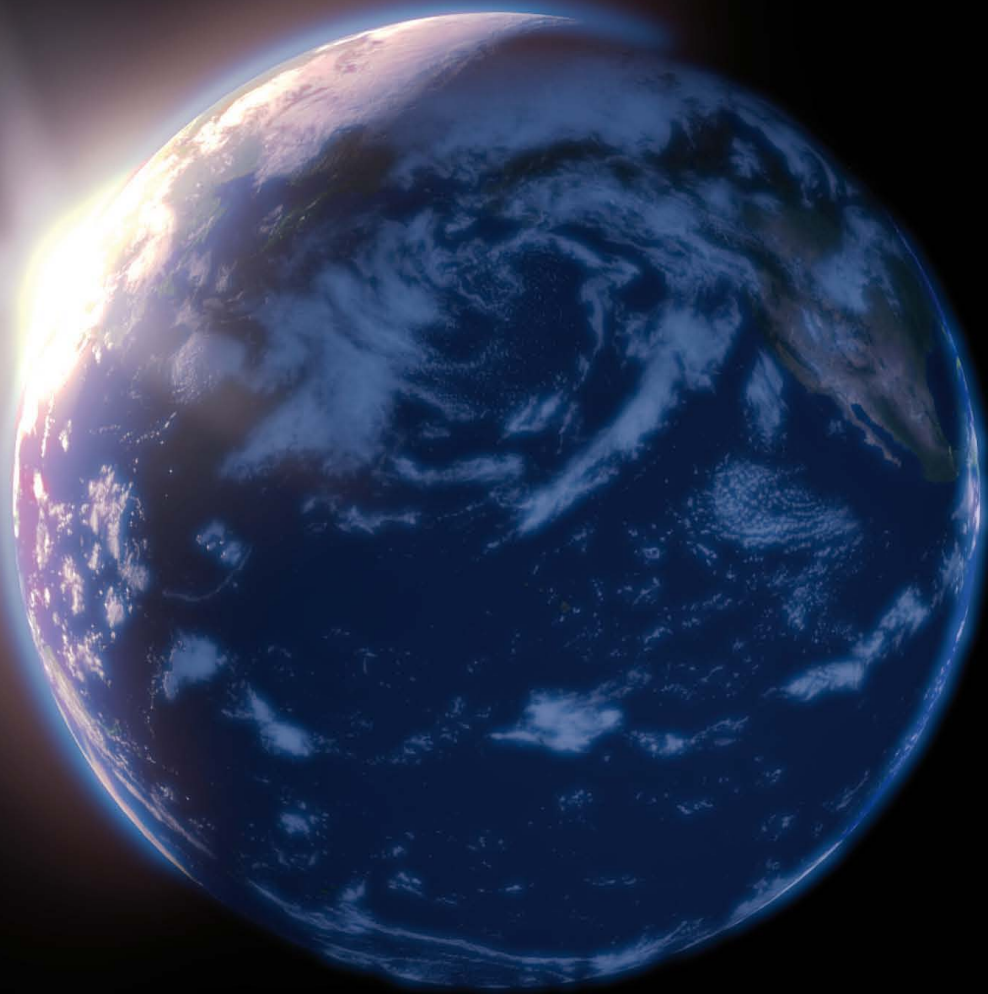




Cool projects that will push your skills to the limit

Blender 2.5

Challenging and fun projects that will push your
Blender skills to the limit

HOTSHOT

John E. Herreño

[PACKT] open source*
PUBLISHING community experience distilled

Blender 2.5 HOTSHOT

Challenging and fun projects that will push your
Blender skills to the limit

John E. Herreño

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

Blender 2.5 HOTSHOT

Copyright © 2011 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: June 2011

Production Reference: 2100611

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-849513-10-4

www.packtpub.com

Cover Image by John E. Herreño (jedihe@gmail.com)

Credits

Author

John E. Herreño

Project Coordinator

Shubhanjan Chatterjee

Reviewers

Rosario Azzarello

Joost van Beek

Ken Beyer

Marian Heddesheimer

Proofreader

Chris Smith

Indexers

Tejal Daruwale

Monica Ajmera Mehta

Acquisition Editor

Steven Wilding

Production Coordinator

Adline Swetha Jesuthas

Development Editor

Susmita Panda

Cover Work

Adline Swetha Jesuthas

Technical Editors

Aaron Rosario

Pooja Pande Malik

Copy Editor

Kriti Sharma

About the Author

John E. Herreño V. is a 27-year-old who developed a deep curiosity for the "magical" things that modern digital computers can do, at the early age of 13. He recently graduated as an Electronics Engineer from the National University of Colombia, which helped him understand how computers are built. He himself studied some principles of software development to get to know how to get the most out of computers.

He became interested in Blender 3D after finding version 1.72 bundled in a magazine CD bought for a very different reason, and since then has been learning from the awesome community of users on Internet forums and available tutorials.

Today, he's highly convinced of the power that open source software and the business models around it have to improve the general quality of life in developing countries.

But, above all, he's a just a human being who wants to know and serve Jesus Christ, the one who died for our sins and was buried and rose again the third day.

Currently he spends most of his time working from home, doing Drupal web development and Blender training, along with some custom software development. He also maintains a personal website at <http://jeh3.net> and can be contacted at jedihe@gmail.com.

I want to express my gratitude to my parents Ramiro and Lucela, my brother Wbeimar, and my sister Dayana for being such a great blessing from God and the greatest encouragement I have found. I would also like to thank the team at Packt: Steven Wilding for starting this project and providing me with some good guidance to outline the contents, Rebecca Sawant and Shubhanjan Chatterjee for their patience on my permanent missing of deadlines, and Susmita Panda for the very good and clear feedback on the produced material. Finally, I would like to thank my current employers, García Iguarán Asociados S.A.S., for their continued support and encouragement on this project.

About the Reviewers

Rosario Azzarello (born February 19, 1980 in Lipari, Italy) is a famous Swiss game designer who specializes in Serious Games and Augmented Reality. He uses Blender and other open source software every day for all the projects that he gets to finish for the society that he is working for.

The most recent serious game that he has produced is "Circolino": A bundle of five Serious Games and self-built input devices to play games while making a range of different physiotherapy exercises. This Serious Game was made for the Children's Rehabilitation Center of Affoltern (Switzerland) and is adapted to children below 12 years of age who have difficulties with motor functions.

To create Circolino he used the Blender Game Engine for the game and the 3D assets (environment, characters, and items), Python for scripting it, Gimp for the textures, Inkscape for the 2D game elements, Audacity for the sounds and the music, and Scribus for the documentation.

If you want know more about Rosario Azzarello and the different projects made with open source software, please visit his website at <http://www.gamedesign.ch>.

Thanks to Dr. Med. Silvia Miranda Azzarello (my wife), Arion Azzarello (my son) and Erik Azzarello (my brother) for their patience and loving support.

Joost van Beek is a talented software engineer and animator based in the Netherlands; Mr. van Beek received his Bachelor's degree in Computer Science in 1999. He has extensive ICT knowledge with a special interest in 3D design and animation. During his years of study Mr. van Beek was already exploring the world of 3D rendering with the first editions of 3D Studio / Blender. Other areas of intensive knowledge are photo enhancement and vector-oriented design; this makes Mr. van Beek a principal in the area of web design, web development, and animations.

Mr. van Beek has recently launched his company "De VeranderFabriek". Within this company, Mr. Van Beek is active in the areas of social media, marketing in combination with ICT, and animations. His in-depth knowledge about social media and his clear vision on the topic makes him a successful leader in this field of work.

[Edward van der Kust]

At "De VeranderFabriek" Joost van Beek has a thorough knowledge of ICT solutions and, more important, understands the impact of using them into actual business scenario's, within and outside the social media realm. Apart from that he is on top of new developments in and around the ICT domain.

I enjoy working with Joost, because of his knowledge, the quality of the work he offers in short time, and his nice personality.

Through KatsBits.com, **Ken Beyer** has been providing Blender and general editing tutorials and resources for over ten years, educating gamers to take the next step up from playing games to making them, learning how to make simple static model replacements, to levels and modifications, to fully independent game production.

He has also long been sought as the 'go-to-guy' for game developers seeking to commercially exploit their ideas, where his highly regarded technical expertise helps companies implementing Blender 3D and other creative processes into their production pipelines. He also provides trouble-shooting expertise to start-up and established companies alike, developing or clarifying 'best practice' strategies as they relate to Blender, creativity, and commercial game production.

Marian Heddeshimer is a web developer and online teacher with over ten years of experience in developing PHP and MySQL database applications for the Internet. He does web development for living, such as creating custom themes and plugins for WordPress.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- ▶ Fully searchable across every book published by Packt
- ▶ Copy and paste, print and bookmark content
- ▶ On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Project One: Render a Starship Shield Impact	7
Mission Briefing	7
Modeling the Mother Ship	11
Modeling the Space Fighter	19
Shading the Space Fighter	26
Modeling and Shading the Planet	31
Creating the Nebula Background	33
Assembling and Organizing the final Scene	42
Compositing the Final Scene	52
Mission Accomplished	61
You Ready to go Gung HO? A Hotshot Challenge	62
Project Two: Fly a Fighter Aircraft Through a Storm	63
Mission Briefing	63
Modeling the Fighter Aircraft	66
Shading the Fighter Aircraft	86
Modeling and Shading the Storm Cloud	90
Assembling the Final Scene	96
Animating the Fighter Aircraft	99
Animating a Simple Lightning Effect	103
Mission Accomplished	104
You Ready to go Gung HO? A Hotshot Challenge	105
Project Three: Construct a Car Using Modeling and Shading	107
Mission Briefing	107
Creating the Initial Patches for the Model	111
Creating the Doors	125
Joining the Patches	131

Table of Contents

Creating the Wheels	137
Shading	140
Simple Render Setup	147
Mission Accomplished	148
You Ready to go Gung HO? A Hotshot Challenge	149
Project Four: Create a Professional Looking Demo Reel	151
Mission Briefing	151
Creating the Intro and Closing Image	154
Creating the Studio Setup	155
Creating the Individual Scenes	162
Composing the Final Video	167
Mission Accomplished	171
Are You Ready To Go Gung HO? A Hotshot Challenge	172
Project Five: Build an Interactive Walkthrough Using the Game Engine	173
Mission Briefing	173
Modeling the Building	176
Modeling the Environment	183
Shading the Environment	187
Shading the Building	200
Setting Up the Objects for the Game Engine	205
Scripting	211
Mission Accomplished	216
You Ready To Go Gung HO? A Hotshot Challenge	217
Project Six: Detailed Render of the Earth from Space	219
Mission Briefing	219
Preprocessing the Textures	221
Object Setup	225
Lighting Setup	229
Compositing Preparation	232
Compositing	236
Mission Accomplished	246
You Ready To Go Gung HO? A Hotshot Challenge	247
Project Seven: Animate a Humanoid Character	249
Mission Briefing	249
Creating the Armature	251
Armature Setup	258
Skinning	262
Animation Keys	264
Animation Breakdowns	268

Animation Tweaks	270
Mission Accomplished	272
You Ready To Go Gung HO?A Hotshot Challenge	273
Project Eight: Create a Snail	275
Mission Briefing	275
Modeling the Shell	277
Shading the Shell	283
Modeling the Body	290
Shading the Body	298
Scene Setup	302
Compositing	305
Mission Accomplished	306
You Ready To Go Gung HO?A Hotshot Challenge	307
Project Nine: Render a Kitchen	
This chapter is not present in the book but is available as a free download from: http://www.packtpub.com/sites/default/files/downloads/3104_Chapter9.pdf	
Project Ten: Model a Basic Humanoid Character	
This chapter is not present in the book but is available as a free download from: http://www.packtpub.com/sites/default/files/downloads/3104_Chapter10.pdf	
Index	309

Preface

With *Blender 2.5 Hotshot* you will take a tour around Blender 3D tools and discover the tight integration between them by working on projects covering many aspects of computer image generation, modeling, lighting, compositing, animation, and the game engine.

What this book covers

Project 1, Render a Starship Shield Impact: This project will take us into the exciting task of creating a space scene from scratch. We will be modeling spaceships and planets, texturing them by using both procedural and image textures, creating a simple nebula background by using a particles system and, finally, compositing it all together.

Project 2, Fly a Fighter Aircraft Through a Storm: We'll be creating a somewhat detailed fighter aircraft by applying a simple but useful modeling methodology, including some hints on topology. We'll also create a very nice shader for the aircraft by using both procedural and image textures. For the environment, we will create a very nice stormy cloud and apply a very cool volumetric shader to it, fully controlled by a particle system. Finally, we will create a simple animation of the aircraft flying through the stormy cloud.

Project 3, Construct a Car using Modeling and Shading: With this project we will learn to work by using blueprints. We will take some time to discuss another modeling methodology, useful for sketching out 3D shapes quickly, while deferring most of the topological issues for a second, finishing pass. This will also require us to work on creating a good car paint shader by using the material nodes system.

Project 4, Create a Professional Looking Demo Reel: Here we will be using the models created in previous projects to create a simple but stylish demo reel video. This will let us get acquainted with the internal video sequence editor integrated in Blender 3D. We will also learn to work with the commonly used workflow of image sequences, along with tricks for materials and data reuse.

Project 5, Build an Interactive Walkthrough Using the Game Engine: This project will guide us on the basics for working with the Blender Game Engine (BGE), including the creation of some nice Python scripts. On the modeling and shading front, we will learn how to create materials that play nicely with the BGE and some tricks for getting a good-looking environment.

Project 6, Detailed Render of the Earth from Space: For the creation of a space scene with the earth, we will learn some useful methods for pre-processing image textures and using them effectively in Blender 3D. The project will allow us to learn to control every aspect of a material by using the appropriate maps. Finally, we will do some post-processing to get the rendered image looking much better than what the render engine produces alone.

Project 7, Animate a Humanoid Character: In this project, we will take the character modeled in Project 10, *Modeling a Basic Humanoid Character*, and add a simple armature to it by using some tools offered by Blender 3D. We will also add a couple constraints to get it working in a somewhat automated way. We will then animate a basic walk-cycle scene, highlighting the steps for creating the poses and polishing them.

Project 8, Create a Snail: With this project we will have the opportunity to try out some nice tricks. We will model the snail's shell by using a simple parametric method, then the body will be modeled by using an iterative process for keeping the job simple and using the very useful sculpt tools. With the modeled snail, we will then practice creating some very cool shaders, by using some nice tricks with procedural textures. Finally, we will compose the rendered snail into live footage.

Project 9, Render a Kitchen: This project will give us the opportunity to learn some commonly useful tasks when doing architectural rendering, both in the texturing and lighting aspects. For the modeling part, we will be creating a basic scene using the simple but effective measurement system offered by Blender 3D. On the texturing front, we will be working with some useful settings for controlling textures, orientation, and a few material properties that are very handy for this type of project. On the lighting part, we will be using a very simple setup that will give us some very nice results with no post-processing at all.

Project 10, Model a Basic Humanoid Character: Here we take a look at yet another methodology for modeling, starting with a basic box-modeling procedure and then improving it by means of the sculpt tools. The chapter will be quite short, but it will be very useful to learn how to combine different tools improve our workflow.



Project 9, Render a Kitchen and *Project 10, Model a Basic Humanoid Character* are not present in the book but are available as free downloads from the following links:

http://www.packtpub.com/sites/default/files/downloads/3104_Chapter9.pdf and http://www.packtpub.com/sites/default/files/downloads/3104_Chapter10.pdf

What you need for this book

- ▶ You will need some general knowledge of how to perform tasks in Blender 3D, such as making simple transformations, adding different types of objects, as well as basic use of modeling tools such as extrude.
- ▶ Good skills for navigating the 3D Viewport can be very helpful when comprehending some of the explanations.
- ▶ General acquaintance with Blender 3D's structure and workflow is necessary: knowing how to navigate between screens and scenes, locating the different editors available, and so on.
- ▶ Some familiarity with image editing software such as Photoshop or Gimp can be very helpful, though it is not required.

Who this book is for

If you are already able to perform some basic tasks with Blender 3D and want to go from the simple and boring renders of monkey-headed characters living in box-like scenes to creating more detailed, visually rich, and polished results then this book is for you. It will give you a good idea of the amount of complexity required for real-world projects and help you in deciding what area you are going to specialize in later on.

This book is also useful if you are an advanced user of any other 3D package wanting to get up and running fast with Blender 3D.

Conventions

In this book, you will find several headings appearing frequently.

To give clear instructions on how to complete a procedure or task, we use:

Mission Briefing

This section explains what you will build, usually with a screenshot of the completed project.

Why Is It Awesome?

This section explains why the project is cool, unique, exciting, and interesting. It describes what advantage the project will give you.

Your Hotshot Objectives

This section explains the major tasks required to complete your project.

- ▶ Task 1
- ▶ Task 2
- ▶ Task 3
- ▶ Task 4
- ▶ Task 5
- ▶ Task 6
- ▶ Task 7
- ▶ Task 8

Mission Checklist

This section explains any pre-requisites for the project, such as resources or libraries that need to be downloaded, and so on.

Task 1

This section explains the task that you will perform.

Prepare for Lift Off

This section explains any preliminary work that you may need to do before beginning work on the task.

Engage Thrusters

This section lists the steps required in order to complete the task.

Objective Complete - Mini Debriefing

This section explains how the steps performed in the previous section allow us to complete the task.

Classified Intel

Extra information that will give deeper understanding of some technical aspects related to the project

You will also find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
fd = cont.sensors['fd']
bk = cont.sensors['bk']
lt = cont.sensors['lt']
rt = cont.sensors['rt']
up = cont.sensors['up']
down = cont.sensors['down']
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".



Warnings or important notes appear in a box like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

Project 1

Render a Starship Shield Impact

One of the most common scenes to create using Computer Graphics is a space scene. Many well-known space movie scenes come to mind when thinking about it and, of course, the very mood by itself is an interesting motivation to accomplish the task. In this chapter we're going to make a space scene, mixing some techniques to get a believable environment and create models that give the illusion of massive objects, a spaceship in this case.

Mission Briefing

Our job is to create a scene in space where a few fighters are attacking a mother ship. To give the right mood to the scene we'll add a simple background comprised of a big planet and a nebula background.

Let's make a simple sketch to plan the scene beforehand, so that the actual modeling, texturing, lighting, and compositing work are clearly understood; that way it is possible to have a clear picture of the whole project from the very beginning, which helps a lot in focusing the effort on what really needs it.



The previous sketch shows the elements that will be needed to create our nice render of a ship being attacked by a few fighters, accompanied by a planet and a simple stars and nebula background.

We'll be working on producing a believable mother ship, though we'll avoid modeling detailed elements—Since the camera is going to be far enough away, details won't be distinguishable. The actual shader for the mother ship will be the default one, because we're going to take care of its look by using compositing techniques. Space fighters will be modeled after a basic style distinct from the mother ship, and will include a couple of simple shaders. The finishing touches of the scene will be done by modeling and texturing the planet, creating a simple background with stars and nebula, and adding an energy shield to the mother ship—Even if there are just a few space fighters, it's better to protect the ship! Finally, we'll take all the elements and composite them to get a more polished look than the one produced by the render engine alone.

What Does It Do?

In this project we will create two basic models by using common modeling techniques and applying some very useful tools that help in reducing the time required and complexity for some tasks.

The background for the scene is going to be composed of two elements: A planet and a nebula. The planet itself doesn't pose a problem at all, but the nebula will give us the opportunity to play a bit with a particles system and perform some magic with the Node Editor.

For the energy shield, we're going to use a moderately complex shader, which includes some ramps and a few textures. Since this shield is the most attractive element in the whole scene, we're going to learn how to have a total control on the impacts that it receives, using a group of lamps.

Finally, we'll be working on the compositing of the Final Scene, separating objects in **RenderLayers** and creating a moderately complex setup that will add some effects such as glow and perform certain color manipulations to give a better look to the render.

Why Is It Awesome?

This project will help us learn some basic, but very cool techniques that can add more visual richness to our work; we'll also learn to effectively use some tools available in Blender 3D that are not so well known.

In the modeling area, we're going to learn how to use a very nice script for populating procedurally generated geometry on an object; this is very useful for giving visual richness to objects that must look complex, but will not be seen closely by the viewer. On the other part, we'll look at a way of effectively using the **Proportional Edit**, which will help us to form a nice, curvy shape for one of our models, without having to manually tweak every single vertex.

For the shading part, we'll be creating a basic rusty-metal shader, using just procedural textures. We'll also apply a very common technique to get nice looking reflections by using a texture.

Last, but not least, the compositing techniques that we'll work with are quite a few: creating a nebula look from just a few dots, adding surrounding glow to an object, and enhancing colors, contrast, and brightness in an image. These techniques are quite useful since they allow us to drastically affect the look of a rendered image without having to re-render it; they're also useful to perform some types of corrections in a very friendly and controlled way.

Your Hotshot Objectives

Our work can be summarized in seven activities, as follows:

1. Modeling the mother ship.
2. Modeling the space fighter.
3. Shading the space fighter.
4. Modeling and shading the planet.
5. Creating the nebula background.
6. Assembling and organizing the final scene.
7. Compositing the final scene.

Mission Checklist

In this project we'll need Blender 2.49b for using a special script that hasn't been ported yet—as of writing this book—to Blender 2.50. If you haven't installed Blender 2.49b, now is a good time to do so.

We also need to download two textures: A cold cloudy sky to be used as a reflection map in a couple shaders and the diffuse map for the planet in the background.

A good place to find free textures on the Internet is www.cgtextures.com. The licensing of their textures is open to both private and commercial use, under some restrictions (see license page at <http://www.cgtextures.com/content.php?action=license>). Let's go there and download the image called Skies0263 (<http://www.cgtextures.com/texview.php?id=18926>) from the Skies Partial section; this image has a strong stormy look. A good guide to find it is looking at the URL of the link and checking for the ID **18926**. This image was chosen because it has blue spots, cloudy spots, strong contrast, and a cold mood; that way it will look nicer when acting as reflection map.

To get the texture for the planet, let's go to www.nasa.gov and download one of the free planet maps (http://www.nasa.gov/zip/272246main_ven0aaa2.jpg.zip). As of writing this book, the images can be found by selecting **Multimedia** → **3D Resources** in the homepage, then choosing **Images and Textures** in the list of categories; make sure to download a projected map, not a normal photo of the planet; also pay attention to the resolution and quality of the image, since the planet is going to be quite big in the final render, resolution is very important. For the explanation we'll be using the Venus texture map, which has a width of 1440 pixels and a height of 720 pixels, good enough for our purposes.

Before starting work on it, let's discuss how we are going to organize our project. Splitting the project into smaller parts is very convenient to avoid having to work with a cluttered viewport at some point. For the sake of simplicity, we are going to organize our project in the following manner:

- ▶ One file containing everything
- ▶ Separated scenes for the mother ship, the space fighters, the planet, and the nebula
- ▶ One main scene to bring together everything (except for the nebula scene) and composite the final render

It is suggested to have a folder exclusively for this project's files. A good file name could be `space-scene.blend`. Do not forget to save as often as possible while working on the project and use incremental saving, by using the plus button on the right-hand side of the filename text entry in the file browser, this way we have an easy way to revert when errors occur.

Now that we have a clear idea of what we're going to create, along with the way to organize our objects, it's time to fire up Blender 2.50, save the file with the proper file name and start the ride.

Modeling the mother ship

What we'll be doing first is create the model for the mother ship. This model will need some work to define its general shape; however, the hard part of it will be taken care of by a very useful script bundled with Blender 2.49b. We'll start working with Blender 2.50, and indicate explicitly when to switch to Blender 2.49b.

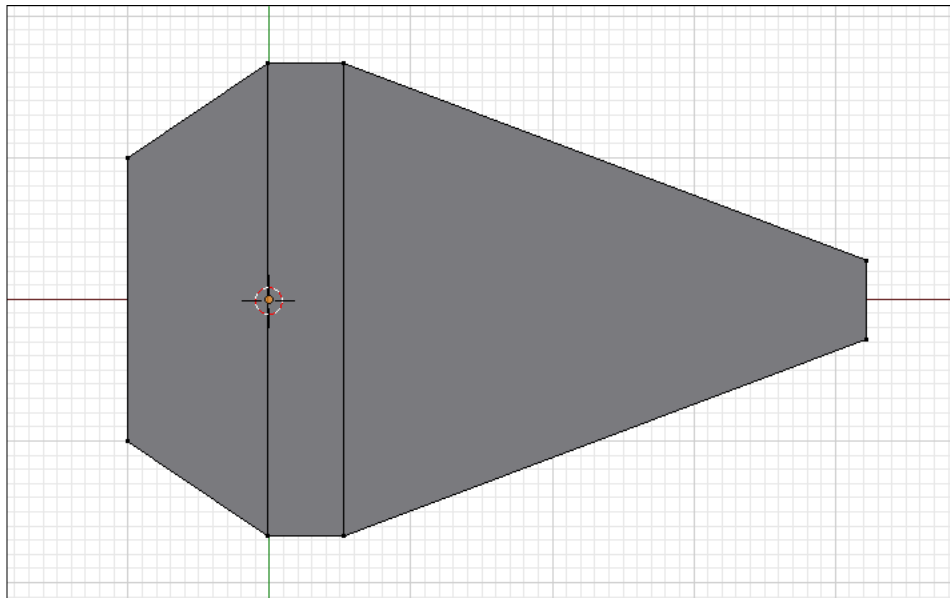
Prepare for Lift Off

The modeling of the actual ship is going to be done by free modeling, without using any blueprint on the viewport; that said, it's important to emphasize that it doesn't mean we should work without having a clear idea of what to model, so it's best to have a sketch of the model:

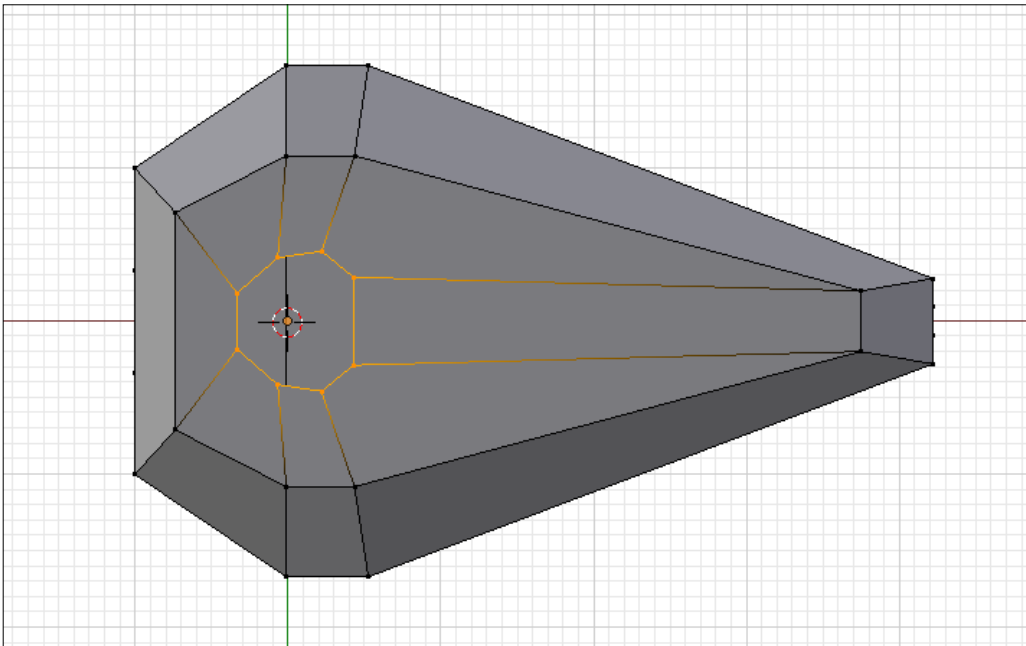


Engage Thrusters

1. Let's change the name for the default scene to **Mother Ship**, select the default cube and delete it (X key or the *Delete* key); the camera and light can be left in, so we can perform some render tests if needed. Remember the menu entries indicated are from the menu located in the 3D View header (horizontal bar at the bottom).
2. Set the 3D viewport to **Top View** by going to **View** → **Top** (*Numpad 7*), orthographic mode (**View** → **View Persp/Ortho** or *Numpad 5*), and make sure the 3D Cursor is located at **(0,0,0)**; if it isn't, go to **View** → **Align View** → **Center Cursor and View All** (**Shift C**).
3. Now add a plane and enter **Edit Mode**. Select the two vertices forming the rightmost edge (vertical edge) and move it **-1** unit along the **X axis** (G key, X key, then type "-1"); then scale it by a factor of **1.6** (S key, then type "1.6") to create a trapezoid shape and continue by extruding and moving the extruded edge straight along the positive **X axis** by **0.5** units (*E* key, *Esc* key, G key, X key, then type "0.5").
4. Finish the basic shape by extruding the rightmost edge, moving the extruded edge along the positive **X axis** by **3.6** units (*E* key, *Esc* key, G key, X key, then type "3.6") and scaling it by a factor of **0.16** (S key, then type "0.16") to get a short edge. The important thing to pay attention to here is not the exact measurements, but the shape of the object and how the proportions work to give some sense of form. Go ahead and play with the proportions until they look nice; be careful to keep the edges correctly aligned. Here is a screenshot to show what the model should look like by now:



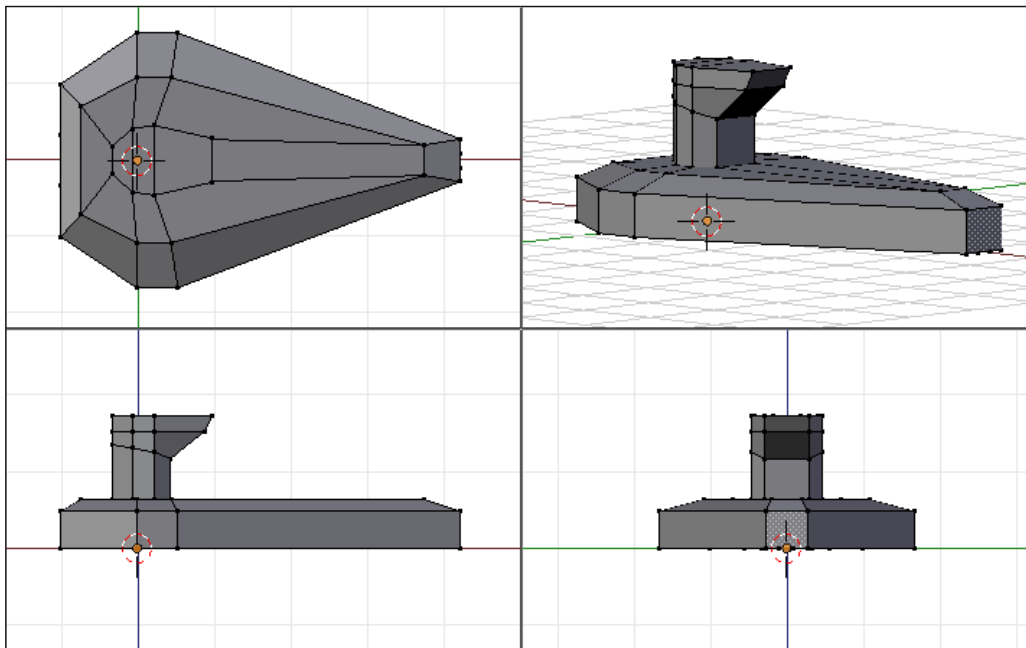
5. With the basic shape in place, we can now switch to **Edge Select Mode** (*Ctrl + Tab*) and select only the edges surrounding the shape.
6. Switch to **Front View** (*Numpad 1*) and extrude the selected edges upwards; moving the extruded edges **0.5** units along the positive **Z axis** (G key, Z key, then type "0.5") should be okay if you didn't change the size of the model too much from the suggested measures; in this case we don't want the extrusion to be too short because it would make the ship look too flat, but should also avoid moving the extrusion too much, since that will cause the 3D shape to not look like a ship. If the proportions are causing any kind of doubt or trouble, a good thing to do is to play a bit, moving the extruded edges along the **Z axis** and rotating the view around the model; that way we can get a feel for how well the shape looks in each case and choose the best one.
7. With the last extruded edges selected, we switch to **Top View** (*Numpad 7*) and extrude once again (*E* key); this time don't move the extrusion, just cancel the translate operation (*Esc* key, right after the *E* key) and follow it by scaling down by a factor of **0.7** (*S* key, then type "0.7"). Now let's take a look at a screenshot to avoid unintelligible written explanations for the next few steps:



To get to the state shown in the screenshot, use the following steps:

8. Tweak the last extrusion from **Top View** to resemble the shape shown close to the border, then switch to **Front View** and move the entire tweaked ring around 0.16 units along the Z Axis (G key, Z key, then type "0.16"); continue by switching to **Top View**, then perform another extrusion, scale it down and tweak it to get the round shape shown for the inner ring.
9. At this point we begin modeling the ship's bridge by selecting the ring from the last extrusion, switching to **Front View** and extruding it **0.7** units upwards; immediately after, extrude again but this time move the extrusion **0.35** units upwards.
10. Finally add an edge loop by using the **Loop Cut and Slide** tool (located at the **Tool Shelf, View** → **Tool Shelf**, or using *Ctrl + R*) and clicking on the edges that run along the last extrusion—when entering the slide part of the command, cancel it by pressing *Escape* to get the edge loop right at the middle.

Right now we have the geometry needed to shape the bridge, but there is some tweaking needed to get the final shape that we really want; this screenshot shows the final shape of the bridge after moving some vertices around and creating three faces to close the top of it:



The basic shape of our mother ship is finished but there are some additional things we need to do to get it finalized.

11. Let's add more detail to the long part of the ship; locate the longest edges of the mesh (running from the side of the tip into the side and back of the ship), then start the **Loop Cut and Slide** tool (*Ctrl + R*) and create three more edge loops that run across those long edges (to increment the number of edge loops to add, scroll the mouse wheel while using the tool); these new edge loops will not only run across the side of the ship, but also up and across the bridge. This is caused by the particular way the faces of our model are organized. Name the object as `motherShip` and save the file.

Now it's time to open good old Blender 2.49b and use it for populating some geometry on the surface of our ship, but without having to create all that geometry by hand.

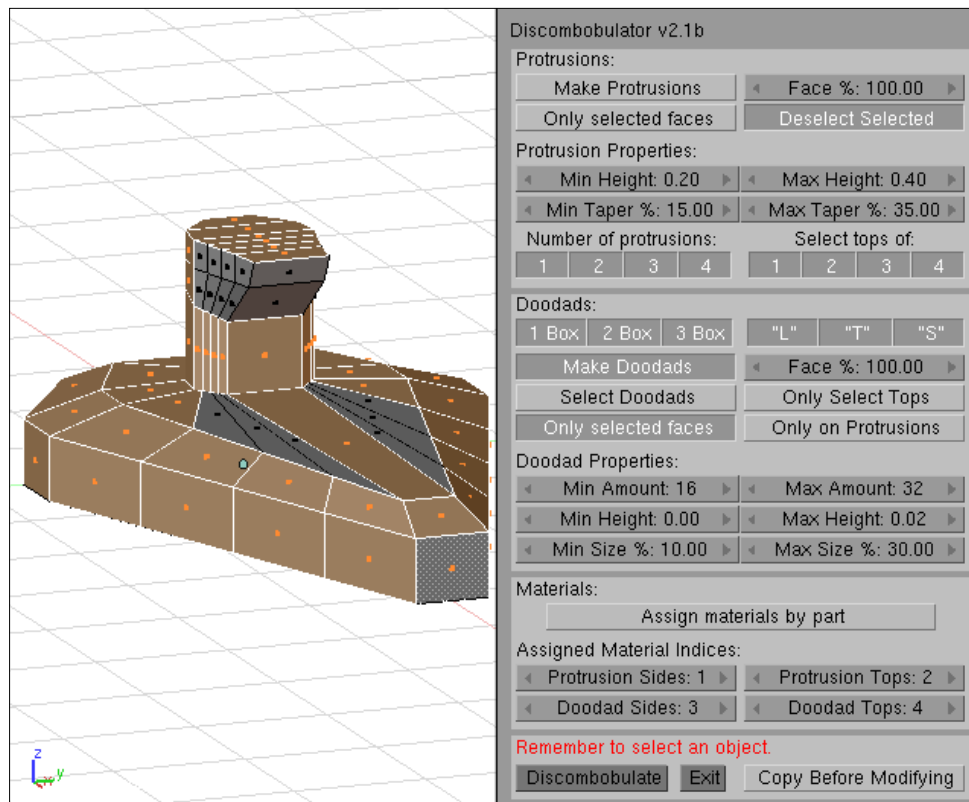
Start by deleting the default cube, and continue by appending the `motherShip` object from the `space-scene.blend` file.

12. To do that go to **File** → **Append** or **Link**, locate the `space-scene.blend` file on your system and click on it. Now Blender will show all the data that the file is holding, organized in folders according to its type. Let's click on the **Objects** folder and then on the `motherShip` object name; to perform the append make sure that **Append** is selected in the header at the bottom of the window and click on the **Load Library** button. A copy of the object will be created at the current scene. If the appended object doesn't appear, look for it in the layers that are turned off.
13. Now we need to do a vertical split of the 3D View by right-clicking on the horizontal line that separates the 3D View from the **Buttons** window, selecting **Split Area** and then clicking inside the 3D View to perform the split.
14. Change the 3D View at the right-hand side to a **Scripts Window** (not the **Text Editor**) and load the **Discombobulator** script, located at **Scripts** → **Mesh** → **Discombobulator**; the script interface will appear on the **Scripts Window**, ready to be configured and applied on the object that is selected in the 3D View on the left-hand side.

Before applying the Discombobulator we're going to select the faces that we want to be affected by the operation.
15. In **Edit Mode**, switch to **Face Select Mode**, select all faces and then deselect the faces on the front and side of the bridge (not the whole extrusion, just the upper part) since these faces are supposed to hold the window to see outside. It is also recommended to deselect the faces with a shape that looks too much like a triangle, since those could cause the Discombobulator script to generate some not-so-nice geometry.
16. The configuration for the Discombobulator is as follows:
 - ▶ Deselect **Make Protrusions** (upper left-hand side), check **Only Selected Faces** (middle left-hand side)

- ▶ On the **Doodad Properties** section (middle) set:
 - ❑ **Min Amount** to **16**
 - ❑ **Max Amount** to **32**
 - ❑ **Min Height** to **0**
 - ❑ **Max Height** to **0.02**
 - ❑ **Min Size %** to **10**
 - ❑ **Max Size %** to **30**
- ▶ Finally, make sure to deselect the **Copy Before Modifying** option (bottom right-hand side).

Before issuing the command, let's take a look at a screenshot, showing the faces that must be deselected and the configuration for the Discombobulator script:



We can now click on **Discombobulate**, at the bottom left-hand side.



If Discombobulator seems to be adding no geometry to faces that are selected in the object, it is because the normals of those faces are inverted. To resolve it simply go into **Edit Mode**, select the faces with the problem and flip the normals by going to **Mesh** → **Normals** → **Flip (W → Flip Normals)**. Notice that Discombobulator can reset its values when undoing the previous modification, so it may be necessary to re-enter the right values.

This will give us a nice, rich geometry addition to the surface of our ship, which seen from far enough can appear like a very elaborated model.

It's quite likely that there are some fragments of the mesh added by Discombobulator floating outside the face.

17. To solve that we just go into **Edit Mode** and select and delete those fragments; a very useful selection command to use when having so much geometry is the **Select Linked** command (hover your mouse pointer over the fragment and hit the *L* key).

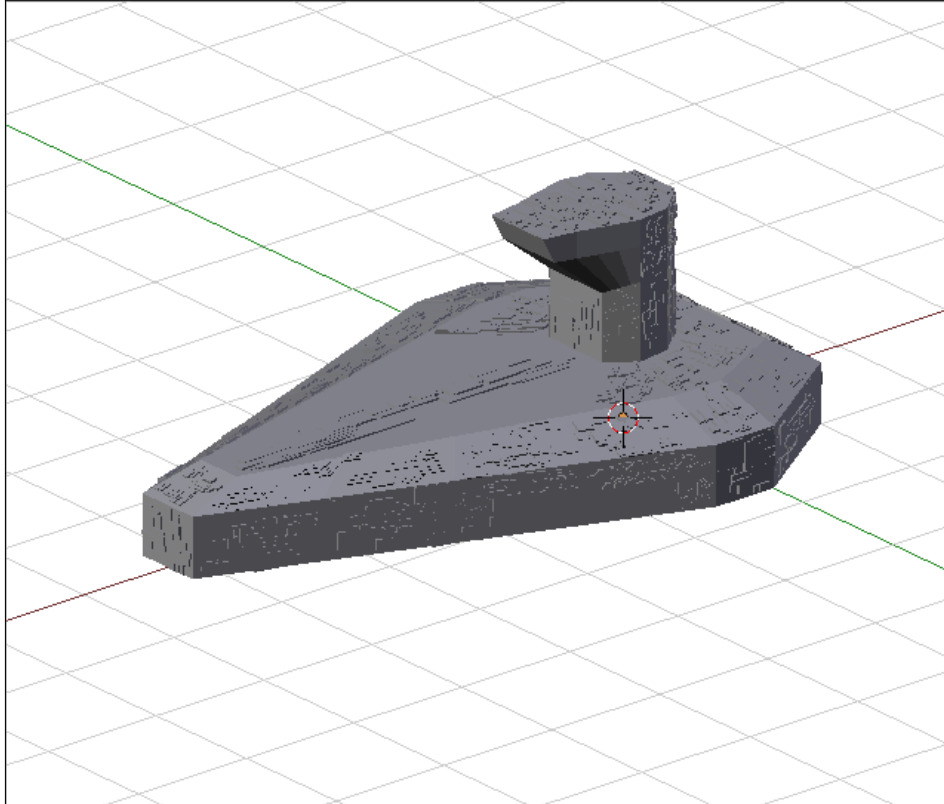
To get this detailed model back into Blender 2.50 we need to perform the same trick of saving the file and then appending the object from the other instance of **Blender**.

18. Save the file in Blender 2.49b as `motherShip-detailed.blend`, go to Blender 2.50, choose **File** → **Append**, and perform the same operation as we did before. Be advised that this time there can be more than one object in the file saved by Blender 2.49b, so we must look beforehand for the name of the mesh with all the details added. As before, make sure that you are appending the object, not linking it—make sure the **Link** checkbox is disabled in the left-hand side region of the **File Browser** in Blender 2.50.



The possibility of moving data so easily between different versions of Blender (even so much different as 2.50 vs. 2.4x series) is one of the nicest things the application has. It is allowed by the file format of Blender using a technology that kind of self describes the file content and structure; that way any Blender version can know what data from the file it can load and just ignore the parts that it doesn't understand. In this case, mesh data is 100% understandable between these versions, so we can move it freely without worrying at all. Note, though, that there are types of data that are not 100% compatible between versions.

Back in Blender 2.50 we finally have a nice-looking model for our mother ship. Remember to check if the basic model is still there and delete it (or hide it; layer 20 can be a good place to hide objects that could be of use later). Let's take a look at the final model for the mother ship:



We can now save the file and go to the next part.

Objective Complete - Mini Debriefing

Modeling the mother ship was quite straightforward; we started by defining the overall shape for our mother ship, mostly paying attention to the shape and proportions of it. The finishing touches for our model were added very easily with the use of the Discombobulator script available in Blender 2.49b.

Modeling the space fighter

The model for the space fighter will be created starting with a sphere, then tweaking it with the nice proportional editing tool available in Blender and creating the cabin with some extrusions; the next part will be working on the wings and adding the cannon compartment and the cannon itself.

Prepare for Lift Off

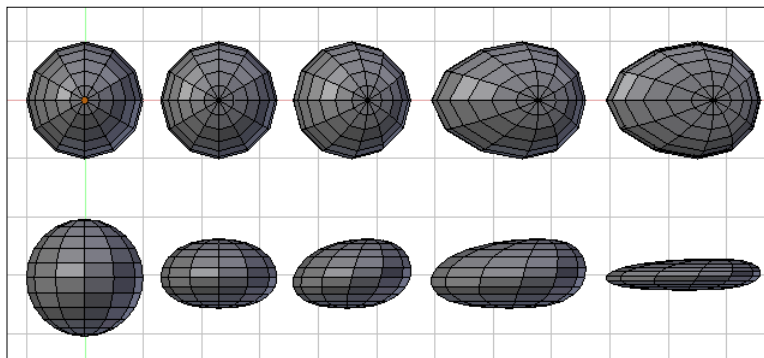
Here you can see the sketch for the fighter that we're going to create:



Engage Thrusters

1. Start by creating a new scene and rename it a **Space Fighter**; this scene will hold the model for the fighter that will be attacking the mother ship.
2. We need a sphere for the body of the fighter, so create a new **UV Sphere** and set its segments and rings to **12**, then switch to **Edit Mode** and **Front View**.

3. Select the vertex at the south pole of the sphere, activate proportional editing (**Mesh** → **Proportional Editing** → **Enable**) and set its type to smooth, then move the selected vertex upwards; pay attention to the radius of influence of the proportional editing and resize it as needed by using the mouse scroll wheel. What we are trying to accomplish now is to get the squashing effect for the body of the fighter, so that it looks more like an egg than a sphere.
4. The upper part of the body of the ship also needs to be squashed the same way as we did with the bottom part. This time, though, follow the squashing by a translation of the selected vertex along the positive direction of the **X axis** (keep the proportional editing enabled); moving it around **0.8 units** should be okay; since this time the operation depends on the size of the influence, the actual values may change. At this point the body has a nicer shape than a squashed sphere, all thanks to the very handy proportional editing.
5. To give the finishing touch to the shape of the body, switch to **Top View**, select the leftmost vertex and move it along the negative direction of the **X axis**; around **-0.8 units** of translation should be okay.
6. We can now turn off proportional editing and make the body look more aerodynamic by selecting all the vertices, switching to **Front View** and scaling it by a factor of **0.45 on the Z Axis**. The whole purpose of this scaling is to make the body look more slim; we can now disable the proportional editing tool, since we don't need it for the next steps. A screenshot showing the progression of the previous steps can be very helpful:



7. Now we can create the cabin of the fighter. Switch to **Top View** and select the north pole of the shape and the vertices forming the ring immediately surrounding it. Delete the selected vertices.

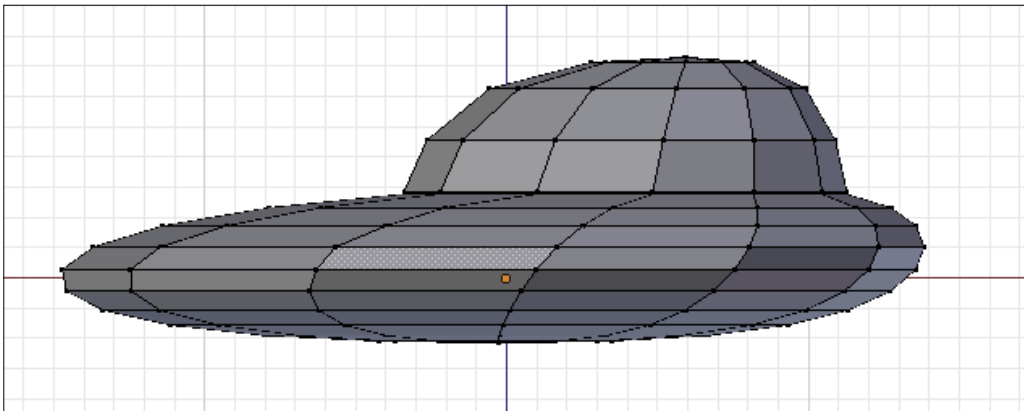
8. Then switch to **Edge Select Mode** (*Ctrl + Tab*), select one of the edges of the border of the hole we just created, and go to **Mesh** → **Edges** → **Edge Loop** (*Ctrl + E* → **Edge Loop** or *Alt + Right-click*). That way we get all the edges selected easily. Now switch to **Front View** and go to **Mesh** → **Add Duplicate** (*Shift + D*), to create a duplicate of the selected ring, press *Escape* to avoid moving it. What we just did was create a ring that is located exactly where the original selection was, but completely disconnected from it; this will be the contact point between the cabin and the body.



If the duplicated ring gets automatically merged back, disable the **AutoMerge Editing** option in the **Mesh** menu and repeat the duplication step.

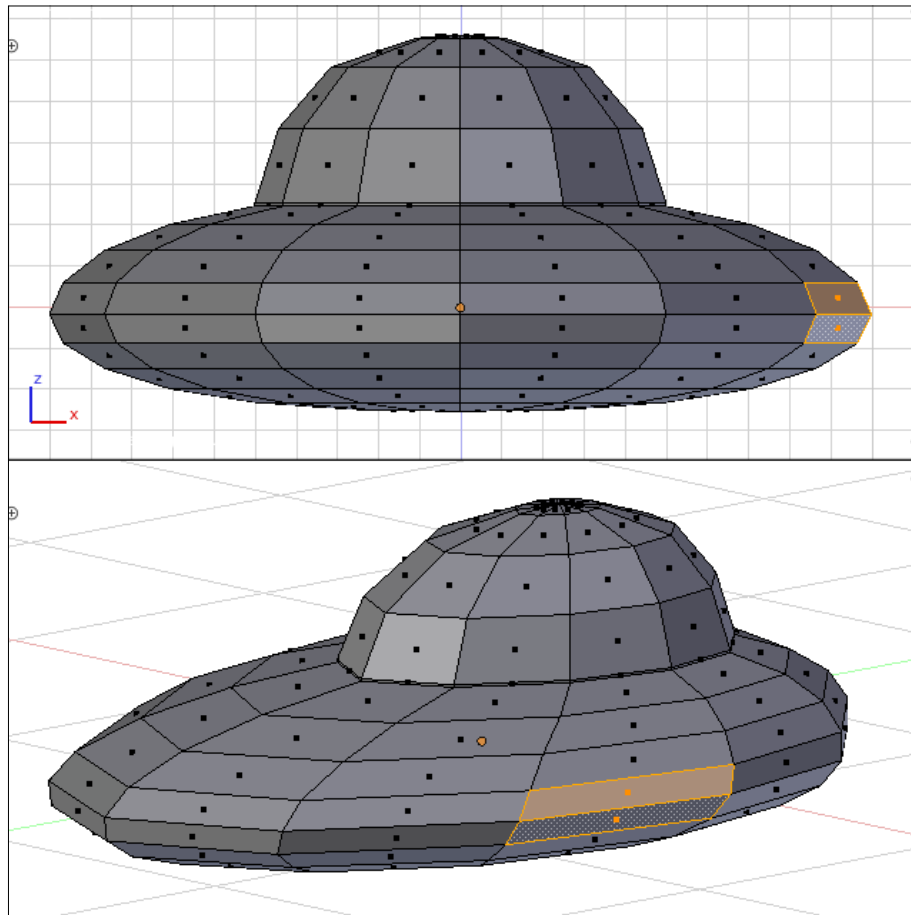
9. Having the first ring for the cabin, let's extrude it five times and move the extrusions straight up; the first extrusion should be moved just slightly (the ring will be very close to the initial one), the other ones can be moved more. Each extrusion must be followed by a scaling operation; what we want to get here is an egg-like shape that will be the basis for the cabin, its height should be comparable to the height of the body. The scaling of the last extrusion should leave us with a very tiny ring (it's a good idea to not scale it completely to zero, to avoid merging the vertices and getting ugly artifacts when using the **Subdivision Surfaces** modifier).
10. As a finishing touch you can select one of the vertices of the cabin and move it along the **X axis** with proportional editing enabled, to break the rigid symmetry that the scaling operations left us with; be careful to preserve some symmetry since this is a mechanical model.

At this point it's a good idea to have a look at how the model should look:



Since our model is using a low number of polygons, it doesn't look really nice.

11. To have a nicer looking ship we can just add a Subdivision Surface modifier: Go to the **Properties Editor**, locate the **Modifiers** tab and in the drop-down menu labeled **Add Modifier** select **Subdivision Surface**; setting the subdivisions to 2 (both for **View** and **Render**) should make it look nicely curved.
12. Up until now we've been modeling our space fighter aligned with the **X axis**; to ease the rest of the modeling work, let's switch to **Object Mode** and rotate it **90 degrees** around the **Z axis** (**R** key, **Z** key, then type "90"); that way the fighter will be aligned with the **Y axis**; remember to switch back to **Edit Mode** and check that proportional editing isn't enabled.
13. The next step is creating the wings of the space fighter. To do that switch to **Face Select Mode** and **Front View**, then locate the line where the top part and the bottom part of the ship encounter ("equatorial" edge loop) and select the two rightmost faces sharing an edge on the equatorial line. The following screenshot shows clearly the faces to be selected:





To be able to perform this selection easily, it is better to have the visualization of the 3D View set to **Solid**; that way we can locate easily the indicated faces. You can switch the visualization from **Solid** to **Wireframe** by using the **Z** key.

14. Now switch to **Top View** and perform a first extrusion that will be left unmoved, exactly where it starts (press the *Escape* key after extruding), then perform a series of three extrude-scale-rotate operations. The whole purpose of it is getting the shape of the wing resembling a horn going right and up (from **Top View**). Play a bit with the tweaking until the shape looks good enough.

While modeling the first wing it's quite normal to start thinking about how we are going to model the other side to be exactly symmetrical. Luckily we have a modifier that will save us all that hassle!

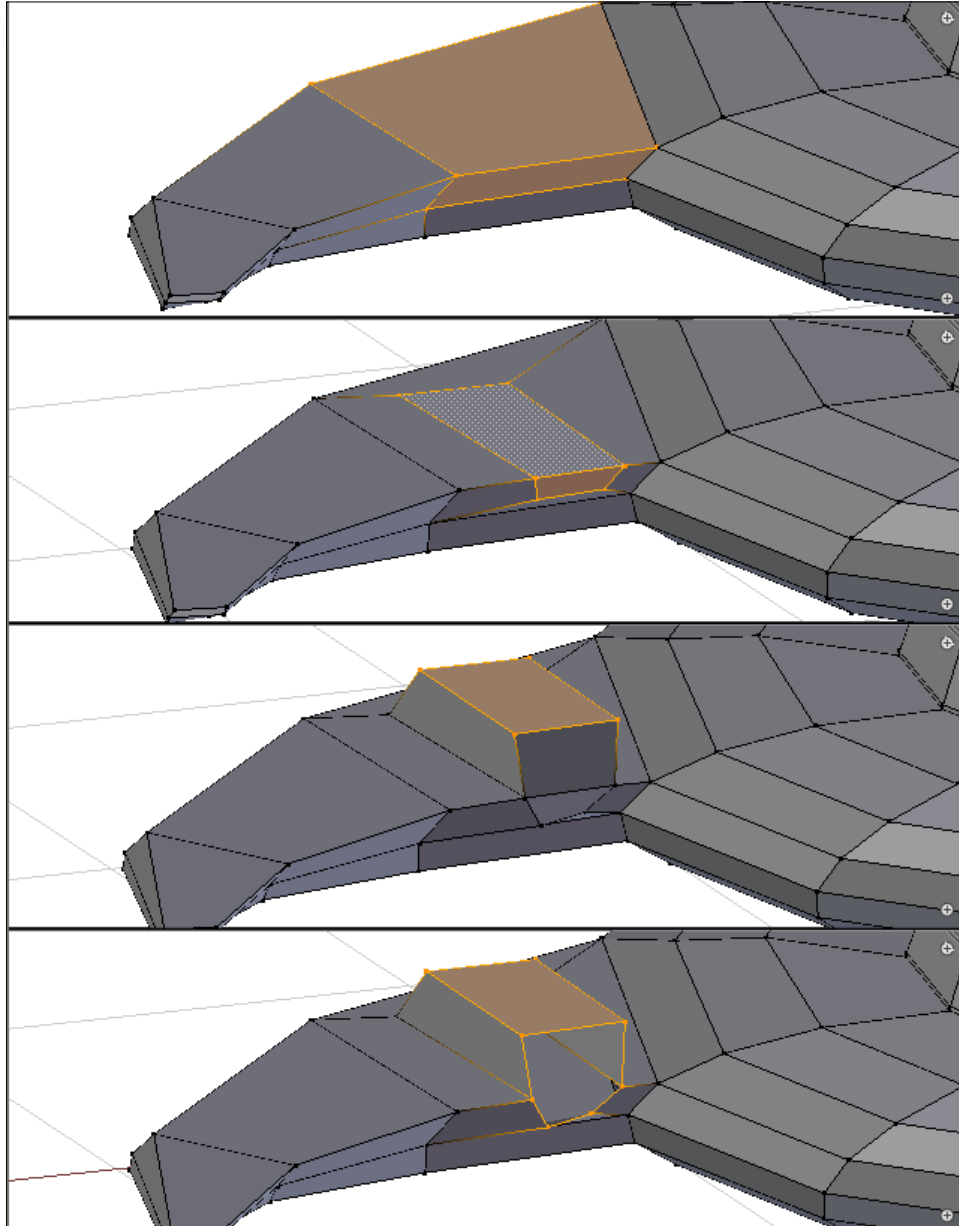
15. Go into **Top View** and set the **3D View** mode to **Wireframe** (press the **Z** key once or twice). Deselect all the vertices (press the **A** key once or twice) and start the **Border Select** tool (**Select** → **Border Select**); now draw a rectangle containing all the vertices in the left-hand side part of the mesh; be careful to not include any of the vertices lying on the green line that represents the **Y axis**. Delete the selected vertices, go to the **Properties Editor**, select the **Modifiers** tab and add a **Mirror** modifier from the **Add Modifier** menu. In the **Mirror** modifier panel that appears, set **Axis to Y (disable X)** and enable **Clipping**. If you had preserved the **Subdivision Surface** modifier that we added previously, make sure it gets located below the **Mirror** modifier we just created by clicking on the triangle pointing down, right below the name of the modifier.



If the settings given for the **Mirror Modifier** don't work for you, make sure that you have been modeling the fighter in the center of the 3D world and that the object's origin is also located in the center of the 3D world.

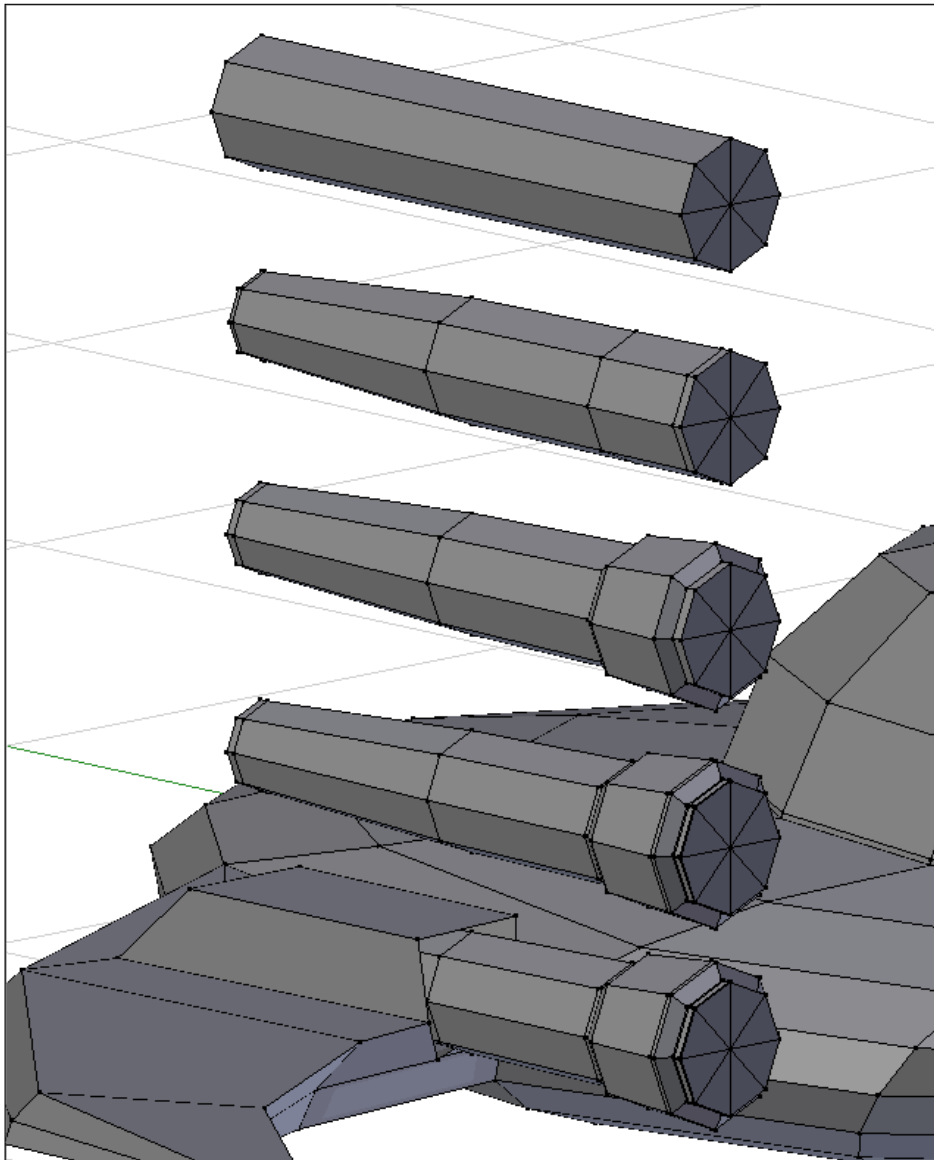
16. As a small tweak for the wings we can go to **Top View**, set the **3D View** mode to **Wireframe**, use the **Border Select** tool to select the vertices that form the join of the wing at the bottom-right part of the model and move them a bit to the back of the fighter; that way the wings have a nicer shape and the whole fighter's proportions look better.
17. Another possible improvement for the model could be selecting the vertices forming the wings (but not the ones in the joint with the body), locating the **3D Cursor** in the joint of the wing and the body, switching to **Top View**, setting the pivot to be the 3D Cursor (press the *Period* key) and then rotating the selected vertices at will. Perform some more tweaking if you want to and rotate the view around the model to get a better sense of its shape.

As a final task to add some detail to the wings, let's disable the **Subdivision Surface** modifier (the eye icon in the modifier settings) and perform an extrusion to hold a cannon in each wing. Since the details of this part are hard to describe in words, let's look at a series of screenshots showing the process:



18. The extrusion must be done with the two faces shown selected in the first step; after extruding, let's scale it down on the **X axis** and then do some manual tweaking to organize it as show in the second step. For the third step just extrude the upper face and move the extrusion upwards. Finally delete the front faces for the whole shape we created (only the faces) and then perform some manual tweaking to finish it.

Of course, we also need a cannon to go with it, so here is a screenshot showing the process to create the cannon:



19. The first step is just creating a cylinder and scaling it appropriately. Steps 2, 3, and 4 are achieved by simply adding more edge loops (*Ctrl + R*) and scaling when necessary. For the final shape shown in step 4, add a few edge loops close to the ones defining the general shape; these loops will help to create a nice crease effect when applying the subdivision surface modifier. Step 5 is just the mesh from step 4 located at the final position on the wing.

Objective Complete - Mini Debriefing

This model required more detailed work. We started with a basic sphere shape and transformed it into a nice shaped space fighter by using very basic transformations and also the very useful **Proportional Edit**.

The cabin and wings were created by just performing some simple extrusions and tweaking as needed. The final details in the wings and the cannon demanded more hands-on tweaking in order to get a nice looking shape and enough detail for our space fighter.

Classified Intel

One of the tools we used for modeling the space fighter is the Mirror modifier. Let's know more about its settings: The axis setting in the **Mirror Modifier** indicates along which axis the object will be mirrored, the actual point of symmetry for the mirror is indicated by the orange dot called **Object Center**. The clipping option allows us to perform an auto-merge, which is very useful to avoid creases at the mirror plane, specially when using a Subsurf modifier. **Merge Limit** allows us to set how far apart any two vertices can be to still be included in the merge operation.

Shading the space fighter

With all the elements in the model ready, we must go into shading it. The materials that we're going to create are not too complex, because we expect most of the final look of the scene to come from the compositing setup.

Prepare for Lift Off

We will be creating four shaders, one for each part:

- ▶ For the body, called `sf_shipBody`
- ▶ For the cabin, called `sf_cristalCabin`
- ▶ For the wings, called `sf_shipWings`
- ▶ For the cannons, called `sf_metalCannon`

Let's start by creating the materials and assigning them to the correct geometric parts of the fighter.

1. Go to the **Properties Editor** and locate the **Material** tab, locate the list of material slots and click on the button with the plus sign, on the right-hand side of it, until you get a total of four slots. The slots will have no names, which means that no material is assigned.
2. Since we need a different material for each slot, we must select each slot and click on the button with the plus sign that is below the **Select** button (below the material slots list; remember we are in **Edit Mode**). As of now we must have four slots, each one with a different material name.
3. Now select each material and put a name to it by modifying the text in the **text** field that is now located where the button used to add a material to each slot initially was; pick the name from the list of shader names given earlier.
4. To assign each material to the corresponding mesh faces, make sure to have **Face Select Mode** active, deselect all faces, and then select the cabin by using the **Select Linked** tool (hover the mouse pointer over that part of the mesh and press the **L** key), then go to the **Properties Editor**, select the `sf_cristalCabin` material, and click on the **Assign** button, located just below the material slots list. Do the same for the corresponding mesh parts for the other three materials. Since the body and wings are not separated mesh patches, it will be necessary to work on selecting each one by hand (**Select** → **Border** can be of help).



An easy way to check if each material is assigned to the correct mesh parts is to choose a different strong diffuse color for each one and check in the 3D View that each material is assigned to the correct faces in the mesh. The diffuse color is located right below the title of the diffuse panel of the material that is currently selected.

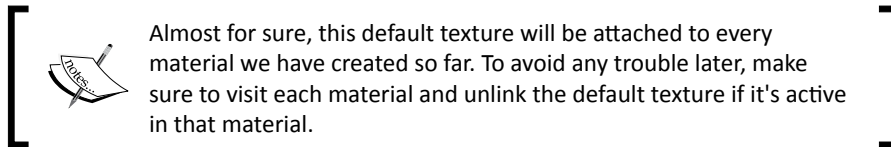
5. Let's now list the values for the different properties of the material `sf_cristalCabin`:
 - ▶ Diffuse color: 0.64 Red, 0.74 Green, 0.80 Blue
 - ▶ Specular panel:
 - Specular Intensity: 0.92
 - Specular Hardness 13
 - Ambient: 0.4 (Shading panel)

Let's explain what these settings are for:
 - ▶ Diffuse color is for giving a metal bluish color to the cabin.

- ▶ Specularity settings are for getting a strong, kind of metallic specular reflection.
- ▶ Ambient setting is for lowering the effect of the ambient color of the world on the material. The ambient color is like a default filling color that will affect every shader that has the Ambient setting value higher than 0.

The material for the cabin needs to have a texture to create the illusion of reflection.

6. Click on the **Texture tab** in the **Properties Editor**. If there is a texture called "Tex" make sure to unlink it by clicking on the **X** button that is below the texture slots list.



7. We can now select any free slot for the texture and click on the **+ New** button that is right below the texture slots list. After adding the new texture, name it `sf_cabin_reflMap` and set its type to **Image or Movie**; below the black preview locate the **Image** panel and click on the **Open** button. In the file browser that Blender shows, browse your system for the sky image downloaded previously.
8. With the proper texture loaded, go down to the **Mapping** panel and select **Reflection** on the drop-down menu labeled with **Coordinates**. Finally go to the **Influence** panel and set the **Diffuse Color** factor to **0.65** (it must be checked). That will get this material finished.
9. The next material to tweak is `sf_shipBody`. The main settings are as follows:
 - ▶ Diffuse panel:
 - ❑ Color: 0.01 Red, 0.03 Green, 0.01 Blue
 - ❑ Intensity: 0.88
 - ▶ Specular panel:
 - ❑ Color: 0.92 Red, 0.98 Green, 1 Blue
 - ❑ Specular Shader Model: Blinn (drop-down menu on the right-hand side of the color)
 - ❑ Intensity: 0.3
 - ❑ Ramp: Enabled
 - ❑ Hardness: 10
 - ❑ IOR: 9
 - ❑ Ambient: 0.4 (Shading panel)

The ramp (Specular panel) must have two colors, with these settings:

- ▶ First color stop:
 - ❑ Position: 0.083
 - ❑ Color: 1 Red, 0.86 Green, 0 Blue
 - ❑ Alpha: 0.2
- ▶ Second color stop:
 - ❑ Position: 0.292
 - ❑ Color: 0.69 Red, 1 Green, 0.90 Blue
 - ❑ Alpha: 1

Make sure the interpolation method for the ramp is set to **Linear** (the default one) and the **Input** field is set to **Shader**.

Now it's time to go and create a texture that will give a nice rusty look to the material.

10. Go to the **Texture tab** in the **Properties Editor**, delete any default texture that may be linked to the material and add a new one; name it `sf_body_specNoise`. As the name suggests, this texture will just add noise to the specular component of the shader; the diffuse component will also be affected, but just slightly.

The texture `sf_body_specNoise` must be of type **Musgrave** and have these settings:

- ▶ Musgrave panel:
 - ❑ Type: fBM
 - ❑ Dimension: 0.1
 - ❑ Lacunarity: 4.2
 - ❑ Octaves: 8
 - ❑ Basis: Voronoi F1
 - ❑ Size: 0.4
- ▶ Influence panel:
 - ❑ Diffuse Intensity: Enabled, -0.376
 - ❑ Diffuse Color: Disabled
 - ❑ Specular Intensity: Enabled, 1
 - ❑ Blend: Add (drop-down menu located at the bottom left of the panel)

Now it's a good time to check the material preview and find a nice-looking rusty material.

11. The next material to work on is `sf_shipWings`. Go to the **Material** tab in the **Properties Editor** and select it from the material slots list. Then set it as follows:
 - ▶ Diffuse color: 0.26 Red, 0.22 Green, 0.20 Blue
 - ▶ Specular panel:
 - ❑ Color: 0.77 Red, 0.52 Green, 0.55 Blue
 - ❑ Intensity: 0.92
 - ❑ Specular Shader Model: Blinn
 - ❑ Hardness: 7
 - ❑ IOR: 7.2
 - ❑ Ambient: 0.4 (Shading panel)
12. To refine this material we need to go to the **Texture** tab in the **Properties Editor** and add a couple of textures. The first texture will be the same `sf_body_specNoise` we used in the `sf_shipBody` material. To reuse an existing texture, just select an empty texture slot, click on the drop-down menu on the left-hand side of the + **New** button, and select the texture from the list.

The settings at the panel **Musgrave** must be left untouched. Just go to the **Influence** panel and make sure that the only enabled one is **Specular Intensity**, with a value of **0.86**; **Blend** must be set to **Add** and **Stencil** must be enabled (bottom right-hand side corner of the panel).



The stencil option causes the next textures in the texture list to be masked by the texture that acts as a stencil. That way it's possible to control the regions that get affected by a certain texture by masking properly.

13. The second texture to add to the `sf_shipWings` material is, again, one that we created earlier called `sf_cabin_reflMap`. Just select an empty texture slot, and then link the texture to this material by selecting it from the drop-down menu of available textures. The only setting that must be changed is in the **Influence** panel: Lower the **Diffuse Color** down to **0.35**.
14. The last material that we have to define is `sf_metalCannon`. This one will be very much like the `sf_cristalCabin` material, since in this scene we just want both elements to look shiny and reflective.

These are the settings for the `sf_metalCannon` material:

 - ▶ Diffuse color: 0.63 Red, 0.76 Green, 0.8 Blue
 - ▶ Specular Hardness: 13
 - ▶ Ambient: 0.4 (Shading panel)

15. The only texture that this material will use is `sf_cabin_reflMap`, with a couple of modifications in the **Influence** panel:

- ▶ Diffuse Color: 0.6
- ▶ Blend: Value

We chose **Value** as the blending type because it will make the effect look like a grayscale image, which reinforces the idea of the material being a cold metal.

16. To finish the space fighter make sure the object's name is `spaceFighter` and save the file.

Objective Complete - Mini Debriefing

Shading the space fighter was done by applying two kinds of materials: A very basic rusty metal and a reflective crystal (for the cabin). We tweaked the diffuse and specular options according to the needs of each kind of material and then added a finishing touch with the textures in each case. For the reflection, we used a simple technique that yields very nice results and with almost zero impact on render time.

Classified Intel

One of the tricks used in this part was the one used to get faked reflections. Reflections are a must when creating a polished material, such as metal or crystal. Since our scene is located in an empty space, activating the calculation for reflections will be of no use because there is nothing to reflect. In such cases, the reflection mapping trick comes quite handy to get a reflection effect easily, by just using an image and the appropriate settings for the influence values of the texture.

A good question that can arise while working on this part is: Why are we using the Blinn shader for the specular component? The technical definition of the word **shader** is related to a computation that affects the appearance of an object, so using the word to describe either the full material or only certain components of it is commonly accepted (for example, diffuse shader and specular shader, wood shader, and so on). Since shaders are a computation to simulate certain real-world effects, they are susceptible of being optimized for certain use cases. In this case, we need the Blinn shader for the specular part of the material because of the IOR factor that it includes in the computation and that allows controlling the size of the specular area on the surface of the object.

Modeling and shading the planet

This one will be very simple, since we're going to use a sphere for the planet; the important part of the work here is the texturing, that's why we downloaded a projected map directly from `www.nasa.gov` to work with it.

Engage Thrusters

1. Create a new empty scene and name it `Planet`, add a **UV Sphere**, set both its segments and rings to 12, and set its size to 8 (all these are settings in the **Tool Shelf**, immediately available after creating the UV Sphere). Then go to the upper part of the Tool Shelf and click on the **Smooth** button (remember we are working in Object Mode).
2. Then go to the **Modifiers** tab in the **Properties Editor** and add a **Subdivision Surface** modifier; set it to two subdivisions both on **View** and on **Render**.
3. The next step is going to the **Material** tab on the **Properties Editor** and adding a new material to the Sphere (it should have no material by default). Name the material as "planet", and adjust the settings as follows:
 - ▶ Diffuse panel:
 - Color: 1 Red, 1 Green, 1 Blue
 - Intensity: 1
 - ▶ Specular Intensity: 0
 - ▶ Ambient: 0.05 (Shading panel)
4. In the **Texture** tab select an empty texture slot and click on **+ New**, name the new texture as `planetDiff` (*Diff* stands for diffuse), then set its type to **Image or Movie** and click on **Open** to load the planet map from the hard disk.

Since the map of the planet has been created using a mapping technique to put on 2D a texture that exists in 3D, we need to set the right options to get the inverse effect: Taking the 2D map and projecting it onto the sphere properly. To do that go to the **Mapping** panel and select **Sphere** from the drop-down menu labeled as **Projection**. Finally make sure that the **Influence** panel has **Diffuse Color** enabled and its value set to **1**; no other option in the influence panel should be enabled.

Now we have a nicely textured planet to use in the background of our scene. Thankfully it is very easy to find good planet maps on the Internet, so this part of the project becomes quite trivial.

Objective Complete - Mini Debriefing

The planet is very easy, since it depends mostly on a good texture, which is quite easy to find on the Internet. The trick to get the texture correctly applied on the sphere is just to use the correct projection method, which in this case is very intuitive: Sphere.

Classified Intel

One of the options used in this part was the projection option for the texture; it's an option used to project an image into the mesh automatically. In this case, the planet texture is a projected map. Projected maps are a planar representation of a 3D element, in this case a planet. An example of a projected map is a map of the whole world; since the earth is sphere-like in shape, representing it on a planar surface requires some projection technique. The reason for having to use a projected map instead of a simple photo is because the map will apply extremely easily to the sphere and have no visible artifacts or seams that could make it look bad.

Creating the nebula background

Our space scene is in need of something else: A good background to have a believable environment and to reinforce the mood. For this background we're going to create a simple nebula effect, using Blender's particle system and some basic compositing. As we'll see, the main part of the job will be on compositing since the actual 3D scene is very simple to create.

Engage Thrusters

1. Start by adding a new empty scene, and rename it to `Nebula`. Since this scene is going to have just a few objects and be used only to get a basic image, which we will use to composite later, the objects' names won't be important at all.
2. Make sure the 3D Cursor is located at the origin of the 3D world (**View** → **Align View** → **Center Cursor** and **View All** or *Shift + C*) and add a camera, move it **-18** units along the **Y axis**, and rotate it **90** degrees on the **X axis**.
3. Next add a cube and then scale it by a factor of **6.2**. In order to be able to look inside the cube, let's go into the **Properties Editor**, click on the **Object** tab and in the **Display** panel select **Wire** below the **Type** setting. Continue by adding a circle, setting its **Vertices** parameter to **8** and enabling the **Fill** option.
4. To light the particles in the scene we're going to need a light, so add a point light; once added, go to the **Object Data** tab in the **Properties Editor** (the one with the light throwing four rays around) and set these options in the **Lamp** panel:
 - ▶ Energy: 4
 - ▶ Distance: 3
 - ▶ Sphere: enabled

The energy value is to get the lit objects looking very bright, the distance and sphere options are to limit the effect of the lamp to a certain distance around it; limiting the distance at which the lamp affects the objects is important to get control on the final nebula render.

Before continuing make sure that:

- ▶ The cube, the light, and the circle are located at the same place.
- ▶ The camera is viewing directly at the center of the cube and the sphere showing the limit of the lamp doesn't go out of the camera frame (**View** → **Camera** or *Numpad 0*).

Once you're done with checking the objects and camera view frame, let's go enjoy playing with particles.

Particle's Party

The particle system we are going to add will be created inside the volume of the cube so that the lamp can light the particles that fall within its radius of influence.

5. Select the cube, go to the **Properties Editor** and select the **Particles** tab. Notice the empty list of particle systems, meaning that this object doesn't have any particle system attached to it. Add a new particle system by clicking on the button with the plus sign and Blender will immediately show all the parameters available for the particle dynamics, render, and settings.

Now it's time to apply these settings:

- ▶ Emission panel:
 - ❑ Amount: 10000
 - ❑ End: 1
 - ❑ Lifetime: 300
 - ❑ Emit From: Volume
 - ❑ Even Distribution: disabled
- ▶ Enable **Random** between the **Jittered** and **Grid** options.

First we set the **Amount** to a high enough value so we get many particles in the **3D View**; setting the **End** parameter to 1 means all the particles will be emitted at the same time (**Start** is 1 by default); the **Lifetime** parameter controls for how many frames each particle will live. Finally, the emission options work together to get the particles in a random fashion and occupying the whole volume of the cube; disabling even distribution helps to accentuate the randomness of the spatial distribution of the particles.

Continuing with the particle system settings we have:

- ▶ Velocity panel:
 - ❑ Normal: 0
- ▶ Rotation panel:
 - ❑ Initial Rotation Random: 0.45

- ❑ Angular Velocity: Random
- ❑ Angular Velocity Value: 6 (right below the options None, Spin, and Random)

Setting the normal component of the velocity to **0** prevents the particles from moving around, since we need them to stay quiet in front of the camera. The options in the rotation panel are all set to get a random orientation of each particle and to get each particle to rotate randomly. Random rotation is important, since it will produce the variation that we need in the rendered particles.

At this point you could be very curious about how the particles are behaving, so decide to hit *Alt + A* but all the particles fall down! The reason for the falling of the particles is that the scene has a global gravity value set and this specific particle system is configured to be affected by that force; to correct that we go to the **Field Weights** panel and put the **Gravity** value all way down to **0**. One more test for the particles' dynamics shows us that they are staying quiet in space, which is exactly what we want.

Now we need to set how the particles will be drawn and rendered; to do that go to the **Render** panel and set these options as indicated:

- ▶ Emitter: Disabled
- ▶ Select "Object" from the row of seven options.
- ▶ In the "Dupli Object" field that appears, click and select the name of the circle from the list.



Since we haven't been paying attention to the naming of objects, it's quite likely that the names will be generic. Thankfully we just have a few objects, so it will take only a couple seconds to try them one by one, by hand. Of course, using good naming conventions is always recommended.

By disabling the emitter option we tell Blender not to render the cube, just the particles it is generating; selecting object as the option to render particles we get the option to put the mesh object we want in place of each particle of the system. This object, replicated and randomized by size, rotation, and speed, is what finally gives us the base image to composite on the nebula background.

Our particles are still lacking one of the randomized settings: Size. Go to the **Physics** panel and set the **Size** to **0.09** and the **Random Size** to **1**. At this point we can launch a test render to see how the particle system looks when rendered.

6. If the background color of the rendered image is not black, we must set it to black; for that, go to the **World** tab in the **Properties Editor** and set the **Horizon Color** to black. Now we have the basic image that we need to composite our nebula background. Don't forget: Now is a good time to save our file.

Compositing the Nebula

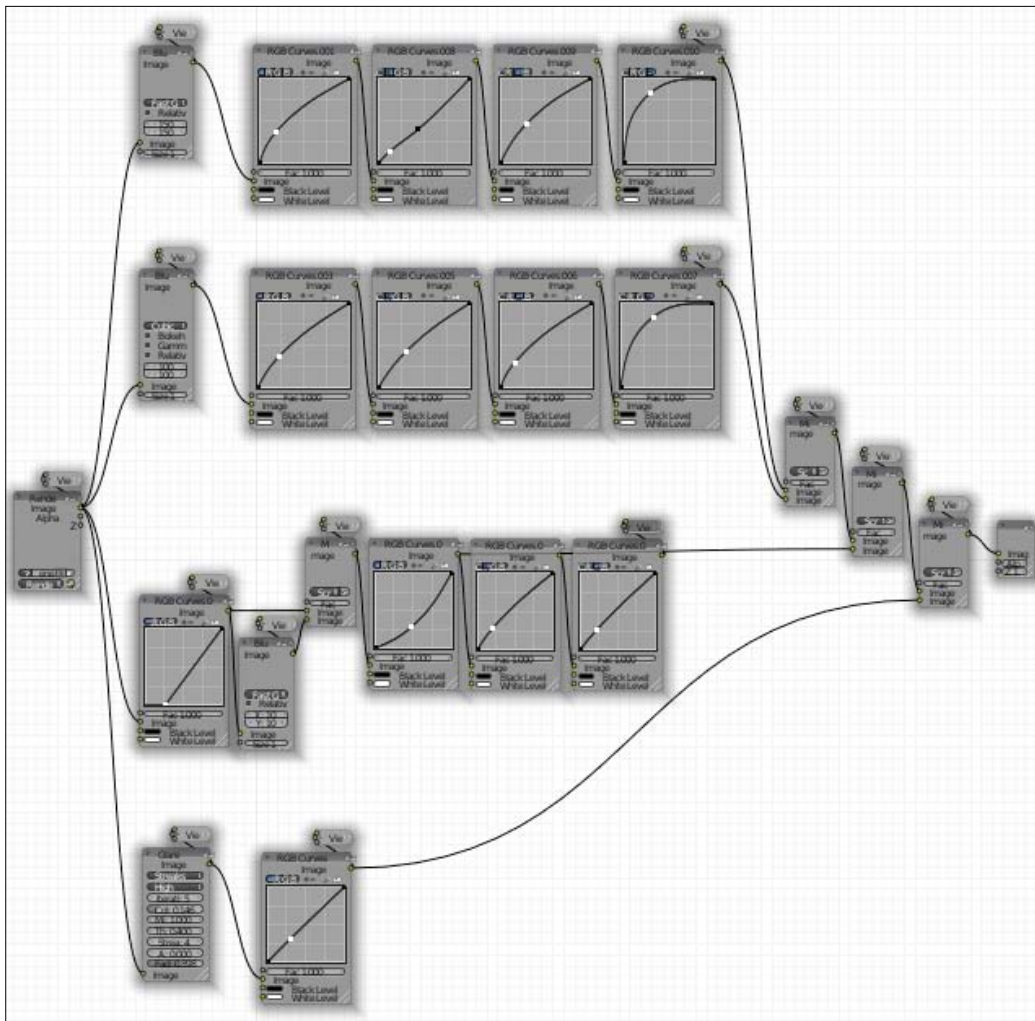
To composite the nebula we will be working on the same scene "Nebula".

7. The first thing we need to do is switch to the **Compositing** screen. Locate the **Node Editor** (biggest editor, in the upper left-hand part of the program window). To enable the compositing pipeline we must select the Compositing nodes option (icon of two pictures) and then check the option **Use Nodes** in the header of this editor.



Don't let the word 'header' make you think it is located at the top part of the editor. By default, in most cases, Blender Editors have their headers at the bottom; the very prominent exception to this is the **Properties Editor**, which comes with its header located at the top. If you prefer headers located on the top side of the editor, just click on the header background with the right mouse button and select **Flip to Top**.

We're going to be running through several screenshots to describe the process. Since the composite setup is not-so-simple, we'll start by taking a look at the setup as a whole. Let's inspect this screenshot:



The general setup can be described this way:

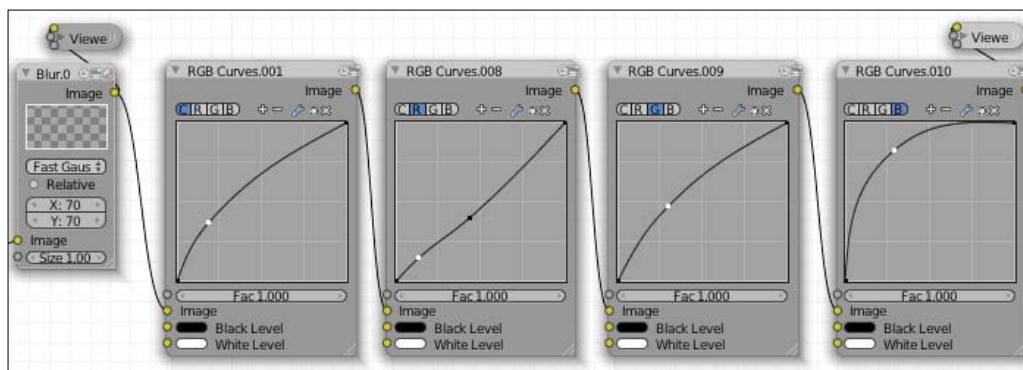
- There is only one input node (renderlayer node, at the far left-hand side).
- There is only one output node (the composite node is usually the only output).
- In the middle we have four rows of processing and a final series of three nodes for mixing the results of the four rows of processing. Let's label these processing rows with the letters A, B, C, and D (top to bottom).



Before starting the creation of the composite setup make sure to click on the button with the camera that is at the bottom right-hand side corner of the renderlayer node; that way the image buffer will be created and the compositing nodes will have the information they need to work with.

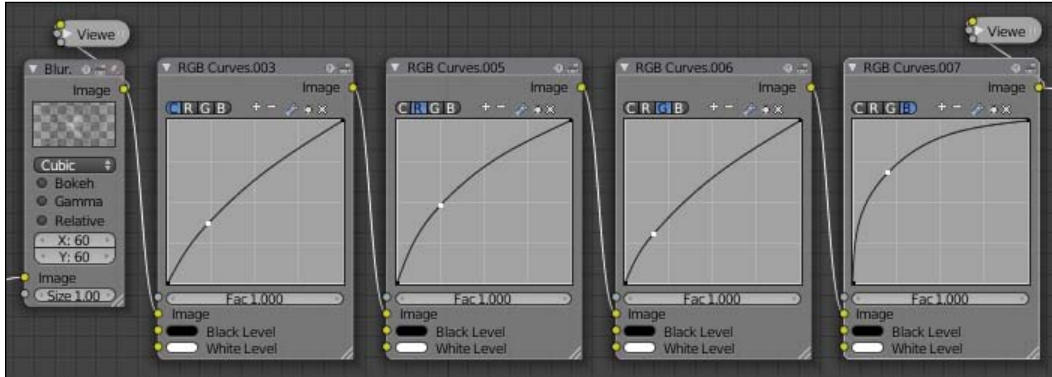
Notice that the small nodes put above some processing nodes are just **Viewer** nodes, used to inspect the result of the processing at that point. A **Viewer** node can be added by going to **Add** → **Output** → **Viewer** and it can be collapsed by clicking on the triangle on its top left.

8. Now that we have a general idea of how the setup must be constructed, let's start with row A:



This row serves the purpose of performing a wide blur of the image from the renderlayer. The first node is a Blur node (**Add** → **Filter** → **Blur**) that receives the output from the image slot in the renderlayer node; it performs a strong blurring on that image using the Fast Gaussian method and using the value of 70 both for the X and Y size parameters; we can then see a series of **RGB Curves** nodes (**Add** → **Color** → **RGB Curves**), which in reality are just one single **RGB Curves** node, but separated here to show side by side the curve for each channel. You will see that the color and green channels are being pushed to be more bright; the red channel is being brightened up a bit only in the shadows; finally, the blue channel is being strongly pushed to become brighter. The connection between these nodes is the obvious: Output slot "image" of a node goes into input slot "image" of the next node in the row, from left to right.

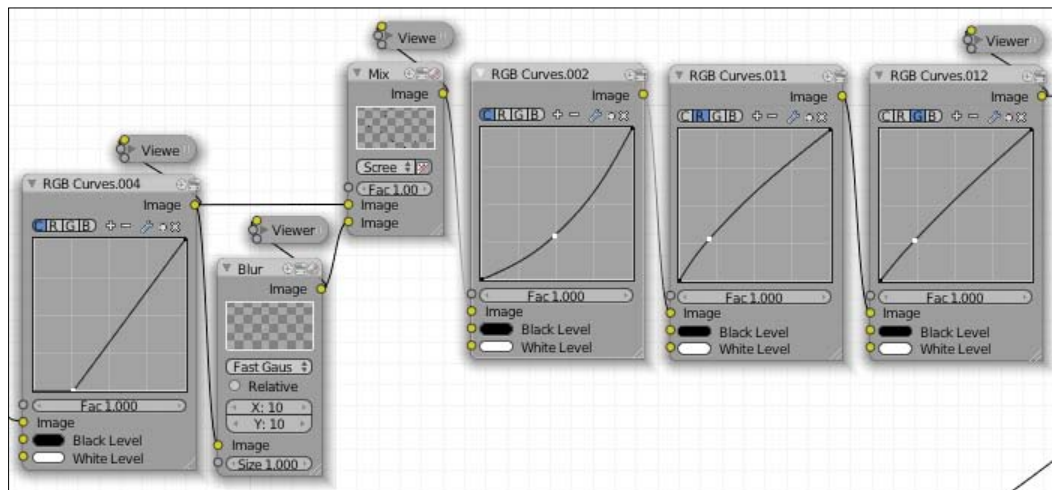
9. Row B appears as seen in the following screenshot:



In this row we find the same structure as in row A, but this time the blurring is not as strong (X and Y sizes are set to 60), and the blurring method is set to Cubic. The series of **RGB Curves** showing each channel for a single node of its type show that all the channels were brightened up, being the blue the one that received the strongest lift. Connection between nodes is again as trivial as for row A.

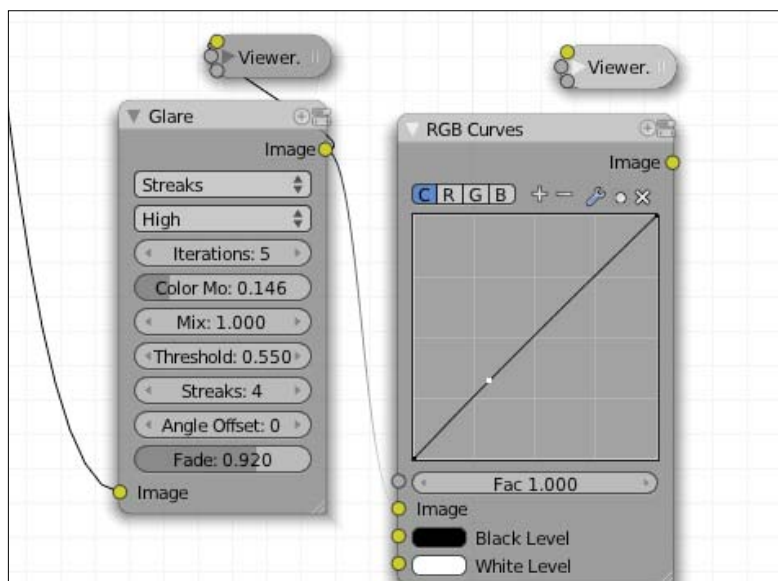
Rows A and B serve the purpose of creating the nebula look; a blueish nebula can be deduced from the strong pushing we made to that channel in both rows. The reason for using two rows of nodes with almost the same setup is because it introduces variation both in color and shape that's needed to break with the simple look that only one of these setups would produce. Though this nebula still looks quite basic, the stars we are going to add later will help it to look much better.

10. Examining row C we find a more complex setup:



The first node of this row is **RGB Curves** (**Add** → **Color** → **RGB Curves**), which performs a clamping on the color channel. With this clamping the dark gray colors below 70% of brightness become totally black; the other shades of gray are just evenly distributed from black to white. Next we find a couple of nodes performing a small blurring on the result from the clamping RGB Curves node and then mixing the clamped image with the blurred one. Notice that the mix node is set to use a **Fac** of 1.0 and the **Screen** method, which is generally used to add highlights. At the output of the mix node we can put a Viewer node (**Add** → **Output** → **Viewer**) and check how the blurring adds immediately the sense of glowing to the points that are going to be the stars of the nebula. Finally, the output of the mixing node goes through an RGB Curves node, here shown separated into three to show how the Color, Red and Green channels look, which performs some lifting of the brightness of the image by increasing the contrast of the shadows (stronger slope, in the bottom left-hand side) and also reduces the contrast to the bright parts a bit (softer slope, upper right-hand side). The lifting for the red and green channels causes the image buffer to become yellow tinted.

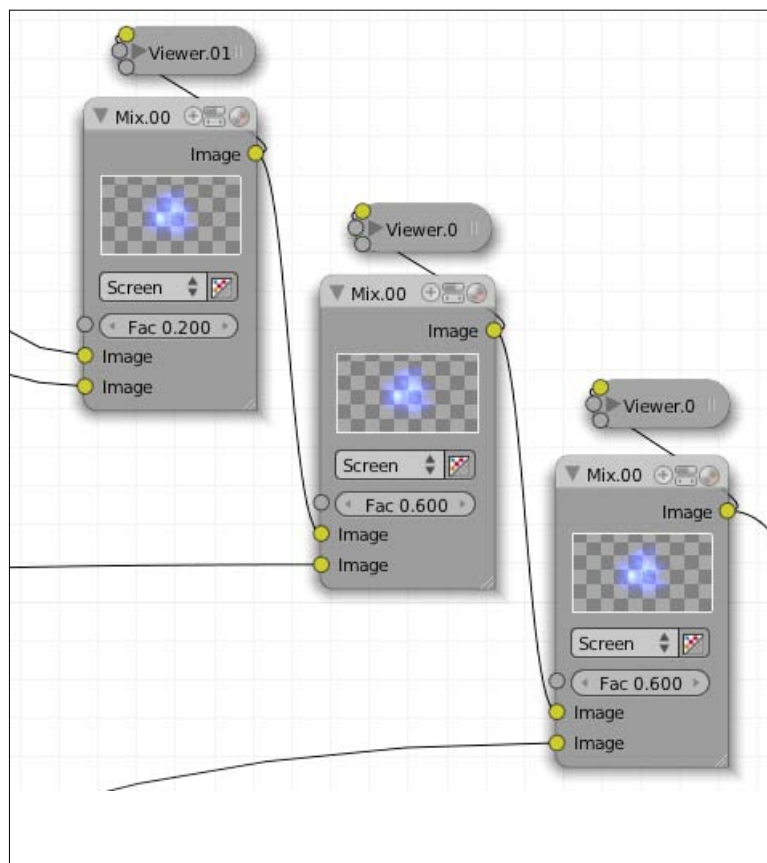
11. The last row of processing looks like this:



This row is the simplest one in this setup. It just performs a nice processing by the Glare node (**Add** → **Filter** → **Glare**) that receives the image from the renderlayer node and then does a very subtle lifting of the color channel with an RGB Curves node. But don't let the apparent simplicity of this row make you think it is irrelevant, because the glare node does the very important task of generating the streaks to get the nice looking effect of stars.

The glare node is a very powerful node, which performs various tasks; in this setup we just use it in the streaks mode, which does the exact job of generating those nice fading lines that we usually relate to stars. Since we want very polished streaks we select the quality to be high and set iterations to **5**. The color modulation adds some variation to the streaks that resembles chromatic aberration; the mix and threshold values determine how much of the result will be combined with the input image buffer and which color values to take into account when generating the streaks. Finally we select how many streaks to generate, how much to rotate the streaks, and the fading factor for the streaks. It is highly recommended to put a viewer node on the output of this node, play a bit with the settings, and get some practice on how it works. To see the full size image buffer getting into a viewer node, enable **Backdrop** in the header of the Node Editor; the backdrop will show the image coming into any viewer node that gets selected.

12. Finally we need to mix all these rows of processing to get the nebula mixed with the stars. For that, we perform the mixing with three mix nodes, as seen in the next screenshot:



The mixing is very straightforward, the links coming from the left-hand side and getting into the mix nodes are in the order A, B, C, D, from top to bottom. Notice that all three nodes use the screen method to mix the image buffers coming in, and don't forget to set the right values for the mixing factor in each node (**Fac** parameter). The last mixing node connects directly into the composite node (the node receiving the image buffer we want to use as the final result of the composite).

13. To finish the creation of the nebula we need to save a file with the final result. An easy way to do that is to add a viewer node to the last mix node, go to the **Image Editor** that is at the bottom left-hand side of the Compositing screen we choose from the beginning and choose the **Viewer Node** entry from the list of available images (drop-down menu on the side of the **Image** menu); once the image is visible, click on **Image** → **Save As** and save the file in the folder of the project. It's important to save in OpenEXR format, since that will let us perform some final tweaks on the compositing of the main scene if needed. To select a different file format, look for the **File Type** drop-down menu, in the left-hand side column of the **File Browser**. Let's also save the blender file.

Objective Complete – Mini Debriefing

This part of the project gives a first look at the power of the compositor. We started by creating a very simple particles setup to get a few dots rendered. The real magic happened in the compositor, where we took these dots and converted them into a basic nebula, including some nice stars within it.

Classified Intel

In this part of the project we saved an image in an exotic format called OpenEXR. The **OpenEXR** format is an image format that saves images using float values for each color channel of each pixel. Common image formats such as JPG or PNG only use a limited range of 0-255 for each channel of each pixel. Since float values are able to represent a much bigger range of different values than the 256 allowed by common image formats, a render saved in OpenEXR format better preserves all the finer details that the render engine produces when performing the calculations. Of course, OpenEXR images produce quite big file sizes compared to its low color-depth counterparts.

Assembling and Organizing the Final Scene

Since we have been working on separate scenes for each part of the project, for the final render we need to put everything together, that way it's easy to work on the compositing part of the project.

Engage Thrusters

1. Start by adding a new scene to our file and name it `Main`.
2. Add a camera to the scene and name it `mainCamera`, go to the **Object** tab in the **Properties Editor**, and set the values in the **Transform** panel as follows:
 - ▶ Location: 6.49 X, 10.71 Y, 3.96 Z
 - ▶ Rotation: 73.38° X, 0° Y, 165.39° Z
3. Now that the camera is placed, go to the `Planet` scene, select the planet sphere, and go to **Object** → **Make Links** → **Objects to Scene** → **Main**. Go back to the main scene, select the planet sphere and set its properties (**Object** tab, in the **Properties Editor**) as follows:
 - ▶ Location: 10.31 X, -10.89 Y, 3.30 Z (Transform panel)

The rotation for the Z axis can be changed at will.

4. The next object to bring into the main scene is the main ship; go to the `Mother Ship` scene, select the object of the detailed mother ship and link the object to the main scene by going to **Object** → **Make Links** → **Objects to Scene** → **Main**. Now is a good time to bring up the **Properties** sidebar (**View** → **Properties**) and make sure the dimensions of the model correspond to these: **5.26 X, 3.38 Y, 1.77 Z**.
5. If the dimensions of the mother ship are very different from the given values, scale it until it gets values closer to the indicated ones. The location and rotation of the mother ship must be left unchanged (they must be zero).
6. We still need one more object in our main scene: the space fighter. Let's go to the space fighter scene, select the object, and hit **Ctrl + L** → **Objects to Scene** → **Main**. Now go back to the main scene, select the space fighter and apply these values in the **Transform** panel (**Object** tab of the **Properties Editor**):
 - ▶ Location: 6.57 X, 7.53 Y, 2.55 Z
 - ▶ Rotation: 14.21° X, 50.65° Y, 150.75° Z
 - ▶ Scale: 0.089 for all X, Y, and Z

If we take a look at what the camera is seeing right now, we should see something resembling the original sketch that we started with.

At this point you must be wondering: Where is the energy shield for the mother ship? We left the shield for later because the shader for it needed to be tweaked according to what the compositing nodes allowed (and helped) us to do.



Certainly, the work on the energy shield is similar to what we did while creating the mother ship, because the final result we're reproducing here doesn't show the different exploration paths and tests that can be done when actually working on the project from scratch. Anyway, talking about how some things depend on others and the way it affects the production flow is a very good thing to highlight, since that helps have a better idea of how the original process went.

Energy Shield to the Rescue

Thankfully, the shield isn't hard to create.

7. Let's add a UV Sphere and set its rings and segments to 16. Then go to the **Object** tab in the **Properties Editor** and set the **Transform** properties as follows:
 - ▶ Location: 0.9 X, 0 Y, 0.53 Z
 - ▶ Rotation: 0° X, 90° Y, 0° Z
 - ▶ Scale: 1.82 X, 2.8 Y, 4.5 Z

Name the object as `shield` and go to the **Material** tab in the **Properties Editor** to start working on the material that will simulate the energy shield.

8. Add a new material (the slot will be created automatically) and name it `energyShield`. Set its properties as follows:
 - ▶ Diffuse panel:
 - ❑ Color: 0 Red, 0.1 Green, 1 Blue
 - ❑ Intensity: 0
 - ▶ Specular panel:
 - ❑ Specular Shader Model: Blinn
 - ❑ Intensity: 0
 - ❑ Ramp: enabled
 - ❑ Hardness: 20
 - ❑ IOR: 4.1

The diffuse color will just give a strong blue as the basis for the shader; its intensity is set to 0 since we're going to rely on transparency to achieve the effect of seeing the border of the shield. The settings for the specular are just the common settings that we've been working with: Blinn shader model, to have more control on the specular spot and a low hardness to let the spot grow. The specular intensity is put to 0 because we're going to control it by using a texture.

Now for the ramp (Specular panel):

- ▶ Interpolation method: Ease (just below the IOR value)
- ▶ First color stop:
 - ❑ Position: 0
 - ❑ Color: 1 Red, 0 Green, 0.37 Blue
 - ❑ Alpha: 0
- ▶ Second color stop:
 - ❑ Position: 0.106
 - ❑ Color: 1 Red, 0 Green, 0.37 Blue
 - ❑ Alpha: 1
- ▶ Third color stop (click on the **Add** button above the ramp):
 - ❑ Position: 0.84
 - ❑ Color: 0.1 Red, 0.16 Green, 0.52 Blue
 - ❑ Alpha: 1
- ▶ Input: Result
- ▶ Blend: Multiply

What we are controlling here is the color that will be shown at each point of the specular "disc"; the interpolation method works great to get nice blending for strong colors like the ones used in the ramp color stops. The three ramp color stops are just indicating that the very center of the specular disc will be blue, then will blend into a strong pink color and then fade completely as indicated by the alpha of the first color stop. The input setting is necessary for the ramp to mold correctly to the form of the specularity, no matter the angle between the camera and the light causing it.

Let's continue adjusting the shader settings:

- ▶ Shading panel:
 - ❑ Emit: 0.2
 - ❑ Ambient: 0
- ▶ Transparency panel: Enabled
 - ❑ Method: Raytrace
 - ❑ Alpha: 0.345
 - ❑ Fresnel: 3.6
 - ❑ Blend: 1.23
 - ❑ IOR: 1.05

At this step we're just enabling the transparency, so that the camera will be able to see the mother ship; the `raytrace` method is needed to have refraction, which we use just slightly (IOR bigger than 1.0). The alpha value works together with the Fresnel and Blend values to provide the border effect for the shield. Showing the border of the energy shield is important in this scene, because that gives a visual clue that what is surrounding the ship is the energy shield that protects it.

Having done the basic setup for the shader, it's time to go and add the textures that will give the rich look to the shield.

9. Let's go to the **Texture** tab of the **Properties Editor** and add two textures.

The first texture is called `shieldDistort` and must be set as follows:

- ▶ Type: Clouds
- ▶ Clouds panel:
 - ❑ Color mode: Greyscale
 - ❑ Noise: Soft
 - ❑ Basis: Blender Original
 - ❑ Size: 0.15
 - ❑ Depth: 6
- ▶ Influence panel:
 - ❑ Diffuse Color: Disabled
 - ❑ Geometry: Enable Warp
 - ❑ Warp: 0.02

These settings just create a simple clouds texture in grayscale and small in size, which will be used to apply some random distortion of the shape of the next texture. To activate the distortion effect we enable **Warp** and set its value to 0.02 to get a very subtle effect.

10. The second texture is called `shieldFX` and corresponds to the main texture for the shield look; its settings are as follows:

- ▶ Type: Musgrave
- ▶ Musgrave Panel:
 - ❑ Type: Multifractal
 - ❑ Lacunarity: 0.2
 - ❑ Octaves: 0.6
 - ❑ Basis: Voronoi F2-F1
 - ❑ Size: 0.20

- ▶ Contrast: 1.4 (Colors panel)
- ▶ Mapping panel:
 - ❑ Size: 8 for all X, Y, and Z
- ▶ Influence panel:
 - ❑ Diffuse Color: Disabled
 - ❑ Specular Intensity: Enabled, 1.0
 - ❑ Blend: Add

These settings are mostly results of a trial and error process; the important idea behind trial and error is using the Voronoi type of function, since it creates a nice-looking structured texture. The scaling (Size of 8) is needed to get the individual "tiles" of the texture to be of the right size for the dimensions of the shield object. Finally in the influence panel we activate the influence of this texture on the intensity value of the specularity, and set its influence mode to **Add**. Since the material's intensity value for the specularity is 0, this means the texture has absolute control of the specular intensity.

Now is a good time to separate the objects into different layers so that the compositing part can process them independently; after that we'll add the lighting to each layer, and get ready to composite the final result.

11. The objects must be put into layers as follows (select each object and set its layer by going to the **Relations** panel, in the **Object** tab of the **Properties Editor**):
 - ▶ Planet: layer 1
 - ▶ Mother Ship: layer 2
 - ▶ Space Fighter: layer 3
 - ▶ Energy Shield: layer 4

The lighting rig will be a bit complex, but not difficult to create. Let's start adding lights in layer 1.

12. Go to layer 1 (then type "1") and add three lamps, with these settings:
 - ▶ Lamp 1:
 - ❑ Type: Point
 - ❑ Energy: 2
 - ❑ Color: 1.0 Red, 0.9 Green, 0.75 Blue
 - ❑ Location: 24.45 X, 15.1 Y, 13.9 Z

- ▶ Lamp 2:
 - ❑ Type: Point
 - ❑ Energy: 0.2
 - ❑ Color: 0.75 Red, 0.85 Green, 1.0 Blue
 - ❑ Location: 13.35 X, 17.5 Y, 12.3 Z
- ▶ Lamp 3:
 - ❑ Type: Hemi
 - ❑ Energy: 0.5
 - ❑ Color: 0.01 Red, 0.02 Green, 0.04 Blue
- ▶ Rotation: 111.15° X, 1.19° Y, -128.87° Z

Lamp 1 will serve the purpose of being the main source of lighting for the scene; its yellowish color tries to mimic the kind of lighting coming from the sun. Lamp 2 is just a simple fill light with a very low energy value; its blueish tint serves as an indication of cold light coming from all around the space. Finally, we have lamp 3, which will act as a very uniform fill light (Hemi lights produce very flat lighting) and set it to a blueish tint. Since point lights don't take into account its rotation when lighting the scene, its values are not provided. Hemi lights don't take into account its location (thus not provided here), but they do care about rotation values, and so they are provided.

13. In layer 2 we will add two lamps:

- ▶ Lamp 4:
 - ❑ Type: Area (Lamp panel)
 - ❑ Color: 0.1 Red, 0.04 Green, 0.01 Blue
 - ❑ Energy: 0.3
 - ❑ Distance: 25
 - ❑ Gamma: 0.45
 - ❑ This Layer Only: Enabled
 - ❑ Area Shape: Square (Area Shape panel)
 - ❑ Area Size: 10
 - ❑ Location: 6.85 X, -3.35 Y, 1.63 Z
 - ❑ Rotation: 72.5° X, 30.25° Y, 44.05° Z

- ▶ Lamp 5:
 - ❑ Type: Hemi
 - ❑ Color: 0.74 Red, 0.9 Green, 1 Blue
 - ❑ Energy: 0.3
 - ❑ This Layer Only: Enabled
 - ❑ Location: 4.06 X, -9.05 Y, 5.15 Z
 - ❑ Rotation: 61.35° X, 16.26° Y, 15.45° Z

These two lamps are set to only light objects living in the same layer as they are, so they will provide lighting only for the mother ship (this doesn't mean that the mother ship will only be lit by these lights!). Lamp 4 provides a dim orange tint to the ship, to convey the idea of the planet's reflected light affecting the ship; the settings for the area of the lamp cause its lighting to be distributed, which will give a nice soft distribution of lighting; the gamma setting is used as a trick to lower the contrast of the lighting provided by this lamp. Lamp 5 is used as a simple fill light for the backside of the ship, again with a blueish tint, like all the fill lights in the scene. Notice that both lamps do care about the direction to which they are pointing, so their rotation values are provided.

Layer 3 will have no lights added and for layer 4 we have to work on creating the lights that will cause the energy shield to be shaded using the nice shader we created for it.

14. Start by adding a lamp with these settings **Object Data** tab of the **Properties Editor**):

- ▶ Lamp panel:
 - ❑ Type: Point
 - ❑ Color: 1 Red, 1 Green, 1 Blue
 - ❑ Energy: 3
 - ❑ Distance: 2.23
 - ❑ Sphere: Enabled
 - ❑ This Layer Only: Enabled
 - ❑ Location: 1.31 X, 2.76 Y, 2.45 Z

This light will only light the sphere shape around it, until the distance value is reached. Since this effect is only needed for the energy shield, we enabled the option to affect only objects in this layer. Now let's duplicate this light a couple times and put its duplicates at some other places near the shield object, to add some nice variation to the shield impact effect for each lamp. Make sure to change the distance setting, the energy, and to put the lights at different distances from the surface of the shield object. Strong changes in the energy value will also cause strong differences in the shield impact effect of each lamp.

For the left-hand side of the shield (looking from the camera) we want to add a bigger shield impact effect.

15. For that we need a different lamp than the ones we've been creating until now; these are the settings:

- ▶ Type: Sun
- ▶ Color: 1 Red, 1 Green, 1 Blue
- ▶ Energy: 5
- ▶ This Layer Only: Enabled
- ▶ Location: 7.74 X, -0.14 Y, 0.42 Z
- ▶ Rotation: 81.24° X, 74.47° Y, 23.02° Z

The reason for using a sun lamp is because it will cause the specularity on the shield to be bigger, making the effect bigger. The main reason for wanting to do that is to get a shield impact spot where it clearly shows the shape of the shield.

There is one last little step to do to make sure the shield is going to be affected only by these lamps.

16. Select all the lamps that we created to cause the shield impacts and go to **Object** → **Groups** → **Create New Group**. Now go to the **Object** tab in the **Properties Editor**, locate the **Groups** panel, and change the name of the group to `shieldImpact`.
17. Next, select the shield object, go to the **Material** tab in the **Properties Editor**, locate the **Options** panel, click on the **Light Group** field (shows an icon with a sphere and a cube) and select the `shieldImpact` group; also enable the **Exclusive** option, so as not to let any other object be lit up by lamps in this group.



There's quite some redundancy here, since the lamps are controlled by the layer and distance + sphere options; but it's not a bad idea to make entirely sure that things will work as we need them to.

As a last step we're going to prepare some settings that will only be used for the Mother Ship, when compositing the scene.

18. Let's go to the **Properties Editor**, locate the **World** tab and set these options:

- ▶ Ambient Occlusion panel: Disabled (it will be enabled later)
 - ❑ Factor: 1
 - ❑ Blend Mode: Multiply (drop-down menu)
- ▶ Gather panel:
 - ❑ Gather Method: Raytrace (first row of options)
 - ❑ Attenuation Distance: 1
 - ❑ Attenuation Falloff: Enabled
 - ❑ Attenuation Falloff Strength: 0.01
 - ❑ Sampling Method: Adaptive QMC
 - ❑ Samples: 16

What we are doing here is preparing the settings for Ambient Occlusion, which is a nice method to add shadowing to the objects in the scene. Setting the **Blend Mode** to **Multiply** causes the shadowed parts to become quite dark, which is what we need. The **Gather Attenuation** options are for controlling how wide the shadows will be; since this effect will only be applied to the mother ship, we use a small distance (the details of the ship are quite small). The Raytrace method is the one with the best results in terms of shape of the shadows; the method and samples are set to get soft shadows (a low samples number will likely cause noise in the rendered shadows). Now, why leave these settings disabled? (**Ambient Occlusion** panel was left disabled), well, that's because this effect will only need to be applied for one layer of the scene, not for the whole scene, so we will come back and turn it on temporarily only when needed.

That's it for the scene creation; now it's time to play with the compositor.

Objective Complete - Mini Debriefing

We left for the very end the creation of the energy shield for the mother ship. This task requires no modeling, since we use a sphere with some scale values; the actual shield effect is achieved by means of the shaders. In this case we applied Raytraced transparency (to be able to use some slight refraction) and worked only on the specular shader for the material. The trick is about textures and ramps, and getting the effect of the textures affecting the right channels of the material. The lighting works together with the shading, and we just created a basic lighting setup to control where the impacts on the shield will be located.

We also prepared a simple **Ambient Occlusion** setup, in order to be ready for the compositing part of the final scene.

Compositing the final scene

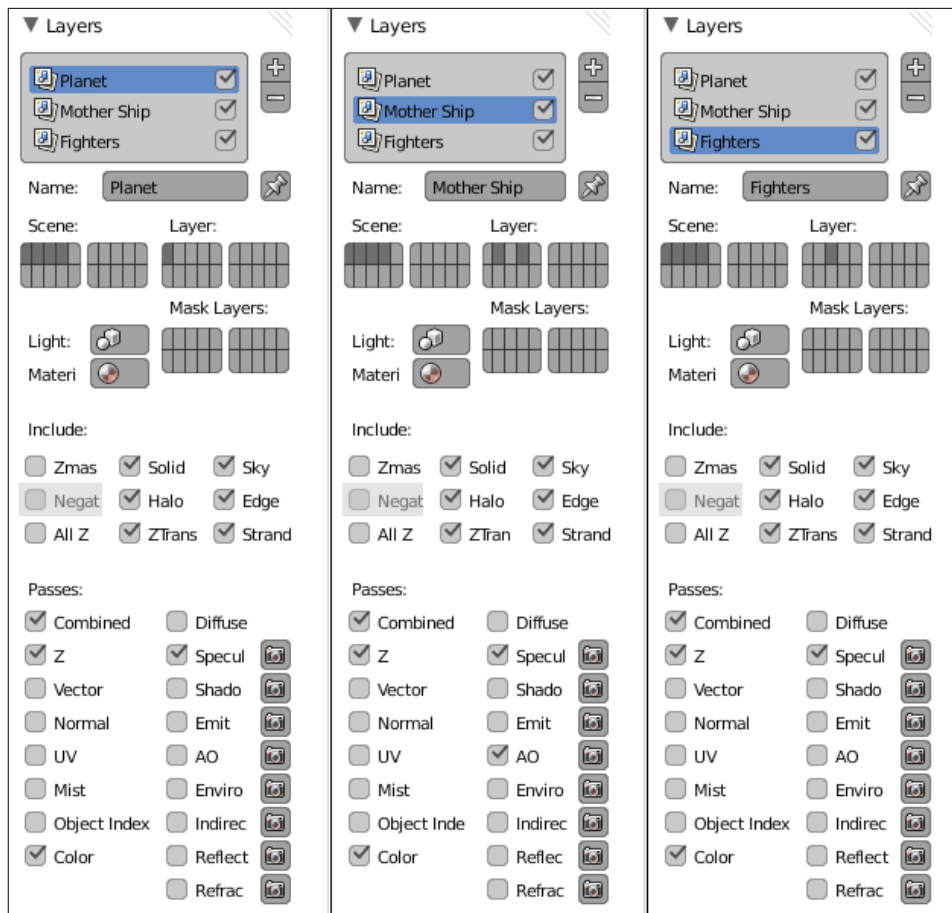
The compositing will be done in the same scene that we are working in, which is "Main". Before we do any node manipulation, we must separate the parts of the scene using `RenderLayers`; then we can go to the **Node Editor** and create the node setup to get our nice final render.

Prepare for Lift Off

Start by going to the **Render** tab of the **Properties Editor** and locating the **Layers** panel. Change the name of the default renderlayer to `Planet`, locate the `Layer` buttons (**Scene** and **Layer** appear as two grids of buttons, as seen in the next screenshot) and enable layer 1; in the **Passes** section below, make sure to have **Combined**, **Z**, **Color**, and **Specular** enabled.

Add a second renderlayer, name it `Mother Ship`, enable layers 2 and 4 in the `Layer` buttons, and in the `Passes` section make sure to have **Combined**, **Z**, **Color**, **Specular**, and **AO** enabled.

The last renderlayer must be named `Fighters`, the `Layer` buttons must have only layer 3 enabled, and the enabled passes must be **Combined**, **Z**, **Color**, and **Specular**. This screenshot shows the setup for the three renderlayers:



The passes other than **Combined** and **Z** won't be used explicitly in the compositing setup; the reason to enable them is just to have the possibility to inspect more closely a certain component of the image at any moment while working on the compositing setup.

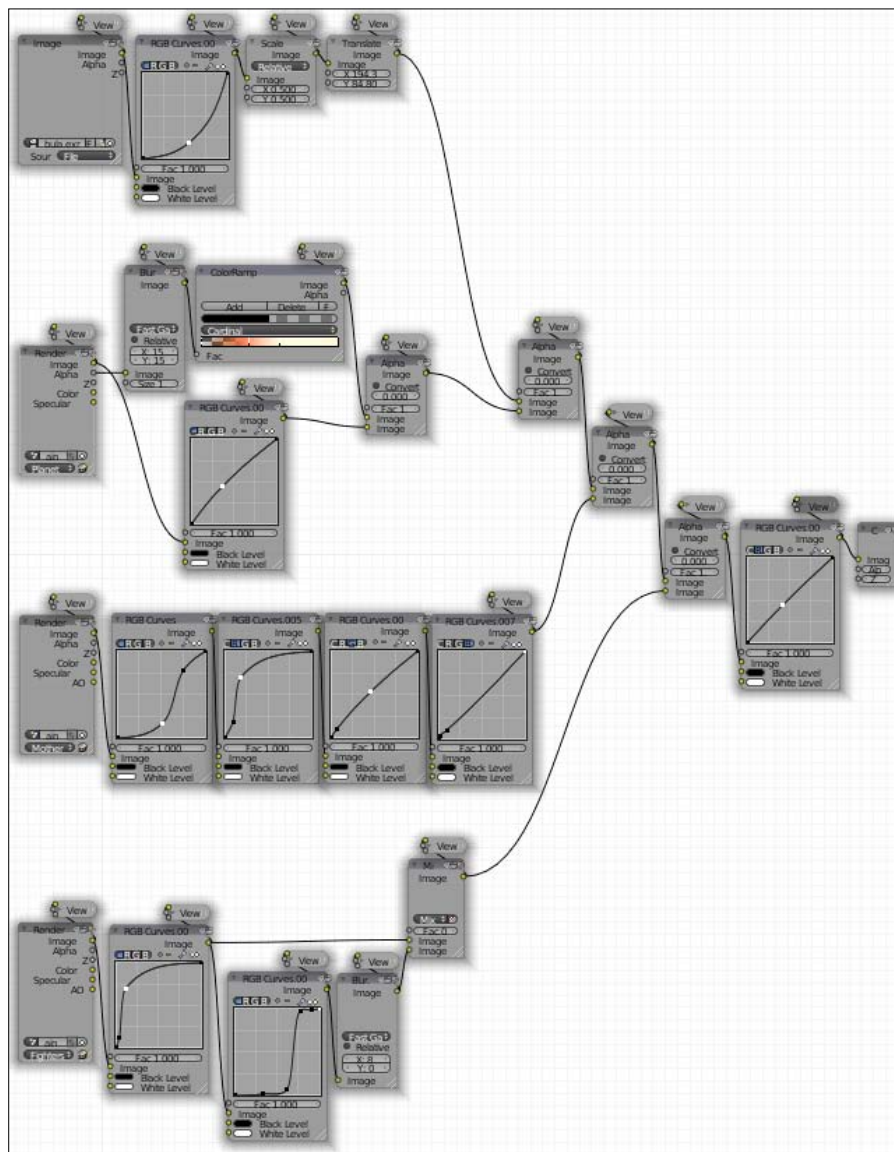
To finish the renderlayers setup, make sure that layers 1, 2, 3, and 4 are enabled in the **Scene** buttons grid (to the left-hand side of the **Layer** buttons) in any of the renderlayers—it's a shared setting between all the renderlayers.

Getting on with It

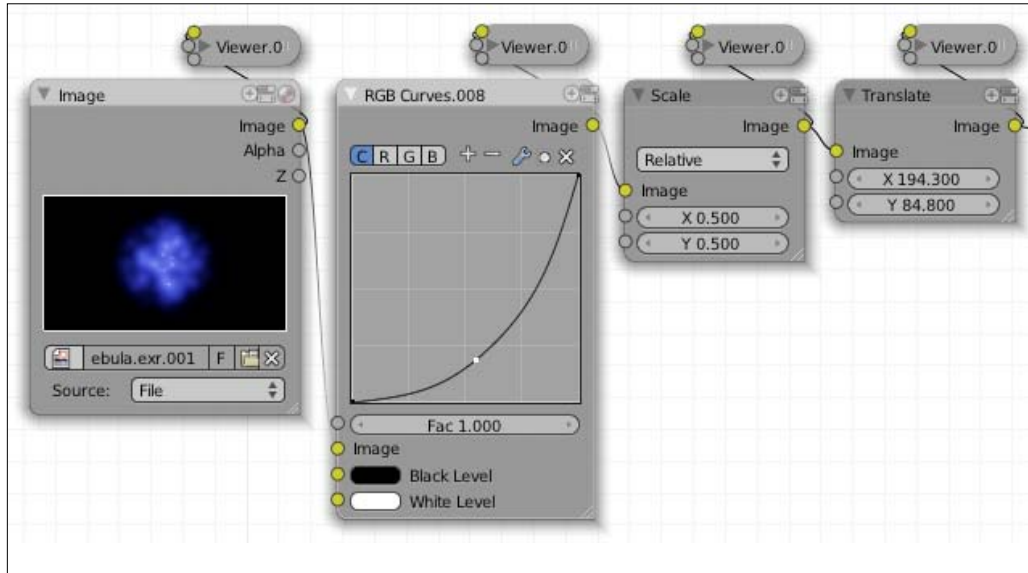
With all the renderlayers settings in place, it's time to do our magic with the **Node Editor**.

1. Go to the **Compositing screen**, choose the **Compositing** nodes option, and enable **Use Nodes** in the header.

This setup will be composed of four rows of processing nodes and a group of three mix nodes to join the separate results. The general view of the setup is as follows:

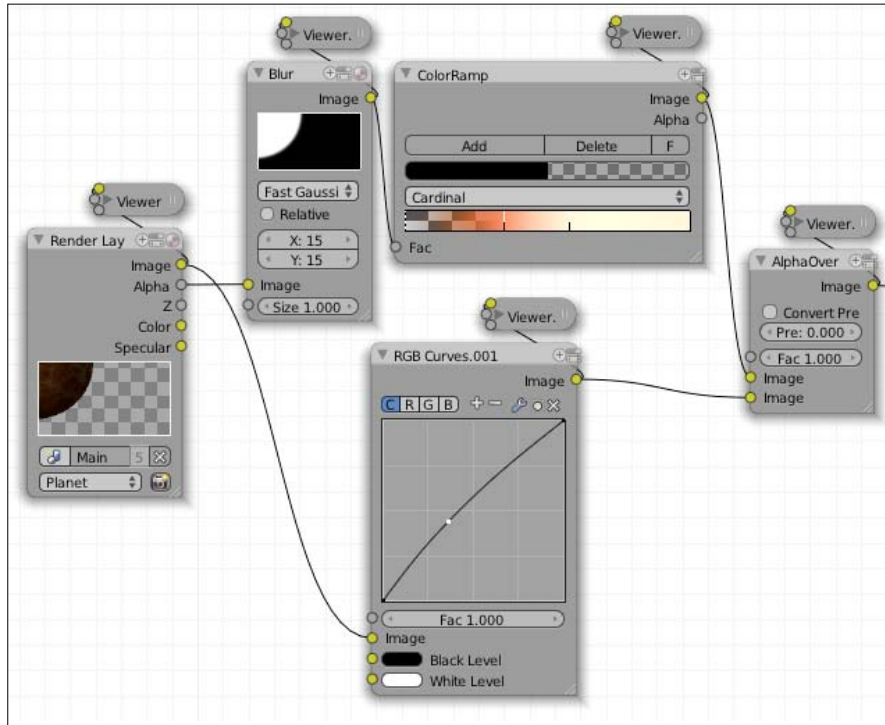


2. Let's call the processing rows A, B, C, and D (from top to bottom), and start with the close-up for row A:



We start by creating an **Image** node (**Add** → **Input** → **Image**) and loading the `.exr` image rendered previously in the Nebula scene. Then we proceed to add an **RGB Curves** node (**Add** → **Color** → **RGB Curves**) node and perform a standard lowering of brightness; the **Scale** node is only needed if the render size for the composite is smaller than the `Nebula.exr` image. Finally the translate node allows us to move the nebula image anywhere in the background.

3. Now let's see the setup for row B:



First we add a **Render Layers** (Add → Input → **Render Layers**) node, and pick the Planet renderlayer; to create the image buffer for this renderlayer, let's click on the camera icon in the bottom right-hand side of the node. In this screenshot we can see two processes being performed. The **Blur** and **ColorRamp** at the top are used to create a glowing effect for the border of the planet; notice that the **Blur** node input is not the image from the render layer, but its Alpha channel; the blurring creates a soft region of transition between the planet and the empty space around it, then the **ColorRamp** takes that grayscale image and maps it into a gradient.

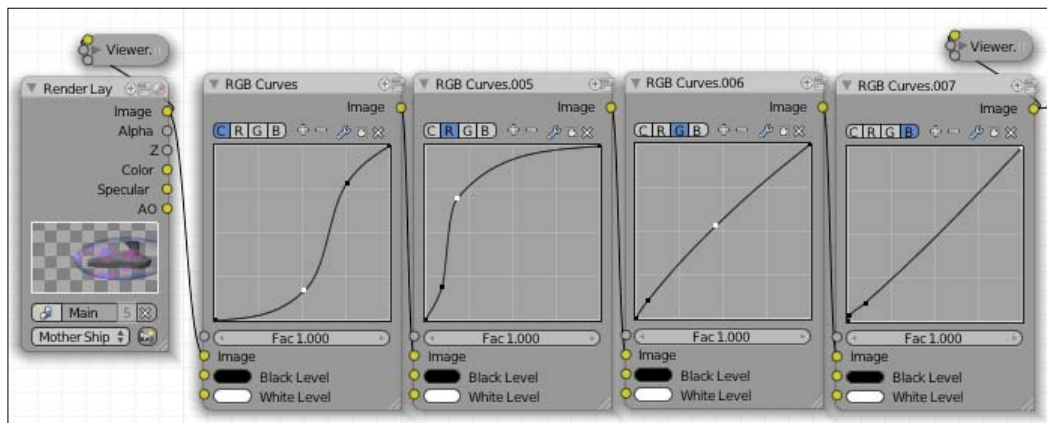
4. The color stops for the **ColorRamp** are as follows:
- ▶ Color stop 1:
 - ❑ Color: 0 Red, 0 Green, 0 Blue
 - ❑ Position: Completely to the left
 - ❑ Alpha: 0

- ▶ Color stop 2:
 - ❑ Color: 1 Red, 0.24 Green, 0.11 Blue
 - ❑ Position: Around 1/3 from left to right (there is no way to specify the position by a number)
 - ❑ Alpha: 1
- ▶ Color stop 3:
 - ❑ Color: 1 Red, 0.93 Green, 0.69 Blue
 - ❑ Position: Around 60% from left to right
 - ❑ Alpha: 1

The **ColorRamp** works this way: For every pixel in the input image buffer, map it to the gradient, if the input pixel color is black, take the leftmost color and alpha from the gradient but if the input pixel color is white, take the rightmost color from the gradient. Any gray color in between gets mapped to the appropriate point within the gradient. Checking the result of the gradient with a **Viewer** node helps to visualize this process and understand it. Don't forget to set the **Blend** type to **Cardinal**, since it's important to get a smooth transition between colors.

The RGB Curves node in the bottom just performs a simple lifting of brightness (Color channel), to make the planet more visible and the **AlphaOver** node in the left just puts the glow behind the planet.

5. Row C must be set up as shown here:

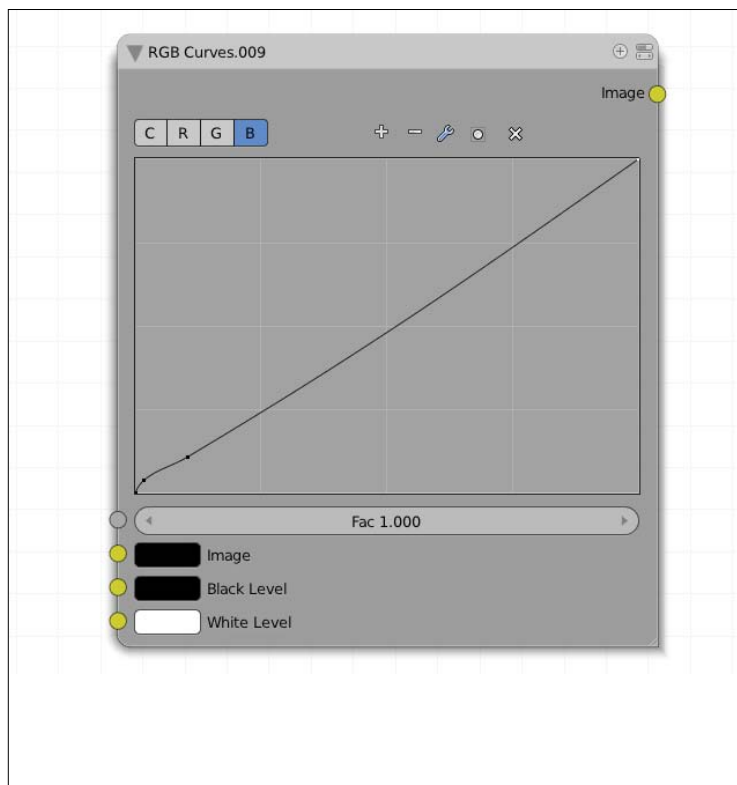


This is just a simple setup where we put a **Render Layer** node as input, selecting the *Mother Ship* renderlayer and then perform a simple processing with an **RGB Curves** node, shown in the previous screenshot separated into four nodes, one for each channel.



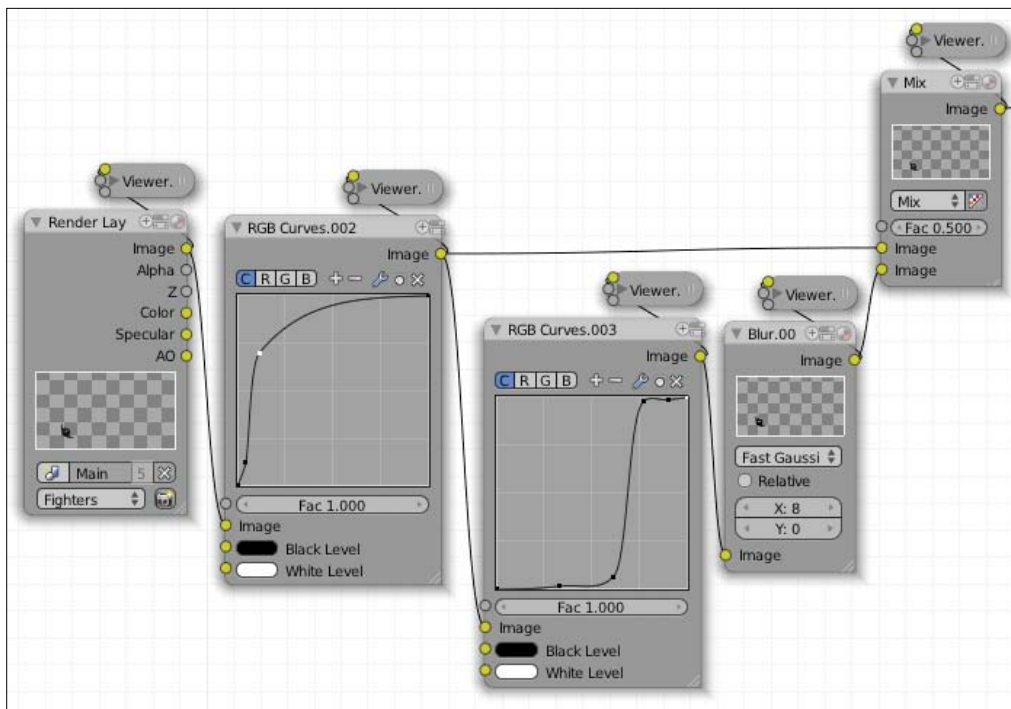
Before clicking on the camera icon of the **Render Layer** node, be sure to enable **Ambient Occlusion** in the **World** tab of the **Properties Editor**. After the image buffer is created, remember to revert it to the disabled state.

The **RGB Curves** nodes perform these tasks: Enhance the contrast of the middle tones for all channels, increase the brightness of the dark tones of red, lift slightly the green channel, and finally, increase the contrast in a very small zone of the dark tones of blue. Since the last channel curve is too small to be seen clearly, let's have a closer look to better see what its shape is:



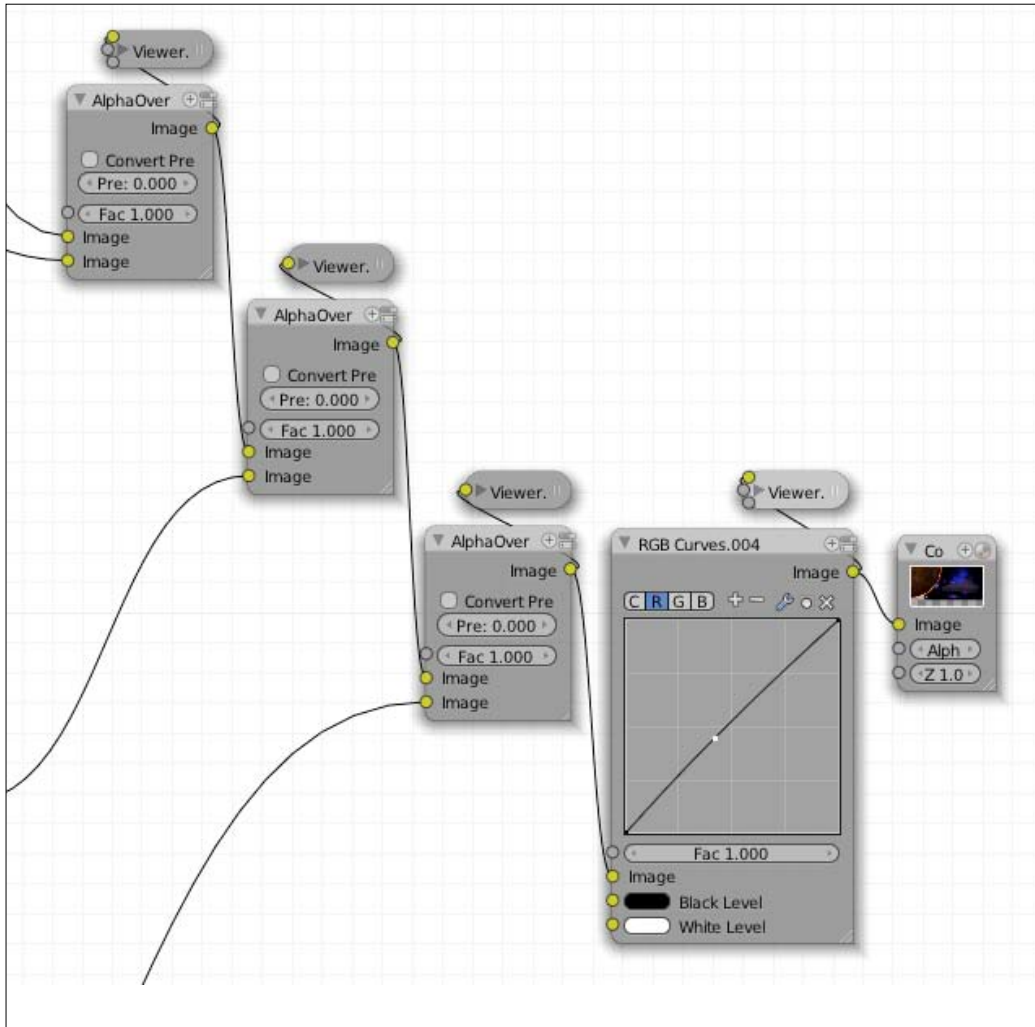
Notice the subtle shape of the curve at the bottom left-hand side.

- The setup for row D is as follows:



The row starts with a **Render Layer** node, using the `Fighters` renderlayer. We then proceed to do a strong lifting of brightness in the first **RGB Curves** node in this row. A side processing of the image buffer performs a clamping of colors (both to the left and right) and then blurs the resulting clamped image. Finally we mix the result of the first **RGB Curves** node of this row with the result of the **Blur** node. The purpose here is to start brightening up the original image buffer (it's very dark) and then create a very high-contrast image to blur, to get nice blurred highlights in the space fighter. The final **Mix** node (**Mix** mode and 0.5 **Fac**) is just the standard way of mixing images when the only important thing is to mix colors.

7. For the final mixing of the four rows of processing, we perform the mixing process:



Note that this time the mixing is performed using **AlphaOver** nodes. The reason for that is because we need to mix separate image buffers that have alpha channel information. There's also a small tweak to the overall brightness of the Red channel of the whole image buffer, using an **RGB Curves** node that feeds directly the **Composite** result node.

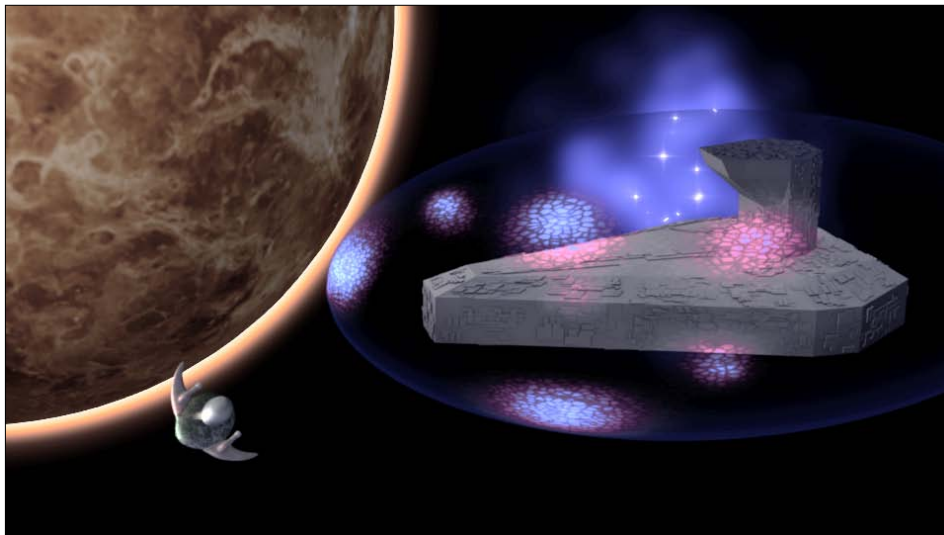
Objective Complete - Mini Debriefing

We separated the render process into renderlayers by associating one or more layers of the scene (**Layer** buttons grid) with a given renderlayer. The passes section determines which of the different kinds of calculations the render engine performs we want available as separate slots in the RenderLayer compositing nodes, when choosing a specific renderlayer. Activating the four layers in the **Scene** buttons grid is necessary, since the renderlayers are going to need to pick one or more of the layers selected there; it's like a two-step filtering: The **Scene** buttons grid decides which layers from the scene to feed into the render engine, while the **Layer** buttons grid picks from those layers the ones that will be used by a specific renderlayer.

Mission Accomplished

And that's it! Our finished scene was created by modeling a mother ship and a space fighter, adding a bit of work in the shaders for the space fighter, then creating a background composed of a planet and a nebula background for which we did a bit of work with a particle system and the compositor. The final touches for the scene were done by adding the energy shield and performing the final composite on the full scene.

Let's take some time to look at our finished image:



You Ready to go Gung HO? A Hotshot Challenge

Our scene looks quite good, but it can definitely use some improvements, let's check a few of them:

- ▶ How about giving the nebula a nicer look? Adding some texture to it will help a lot in enhancing the background.
- ▶ What if there are more space fighters in the scene? Let's work on creating some duplicates of the space fighter and spread them all around the scene. Playing with their distances from the camera can help a lot in adding depth to the scene.
- ▶ How about checking an image from a space movie and getting some info about the background color? That will give you some ideas on how to improve our very simple black background.
- ▶ Adding detail to the mother ship could be of help to enhance its look: Big cannons and antennas are possible enhancements for it.
- ▶ Our mother ship could be enriched very much by adding some effects that mimic light coming from it: Small windows, space ports on the sides, and the bridge could be created by adding a few simple 3D objects and playing on the compositor.

Project 2

Fly a Fighter Aircraft Through a Storm

Fighter aircraft, those hi-tech machines that fly extremely fast. Wouldn't it be cool to model one? How about creating a scene of it flying through a storm cloud? Yeah, that would be very nice. The powerful lightning bolts and the energy being released in a storm as it flies through will add to the awe of the powerful man-made war machine. This scene should be a beautiful blend of nature's power and human prowess.

Mission Briefing

Modeling a fighter aircraft offers quite some challenges, since the details of its shape require very fine hand-tweaking. For this project we're going to need to concentrate on the model; even though it is not going to be visible on a close-up shot we definitely want to have a model that looks good enough to be used in some other projects. The main part of the job will be building the top part of the body (the one that the camera will see). We will also avoid spending too much time on the very minute details of what could be a realistic model, for the sake of keeping it simple.

Our fighter aircraft will need some shading, so we need to create a material for it. Thankfully, a clever use of cloud textures and ramps will give us some very nice military-looking material. We'll also be working on a simple fake reflection material for the cabin, to give it a better look.

Having finished the fighter, we'll pass on to the modeling and shading of the storm clouds, using the new and very cool volumetric shader available in Blender 2.50. For this part we are going to use a technique that lets us define the shape of the cloud very precisely.

The finishing parts of the scene will be about creating a basic animation for the fighter to pass flying close to the storm cloud; we'll also add a simple lightning effect inside the cloud.

What Does It Do?

We will create a short animation in which a fighter aircraft flies inside a giant storm cloud, including a simple lightning effect inside the cloud.

Our main tasks will be modeling the aircraft and creating the giant storm cloud. The modeling part will require some very detailed work but the final product is going to make it worth the effort. We'll be applying some basic *topology* concepts to avoid the situation of having to reconstruct some parts of the mesh if it doesn't conform well to the desired shape. On the clouds side, we are going to use the volumetric material available in Blender 2.50, which will prove how easy it is to create nice looking clouds, even dark ones. Since we require a very precise shape for the cloud, we are going to use a very useful type of texture available for volumetric materials, which will require us to set up a fast particle system to get the ultimate control we want.

Finally, we are going to put all the objects in a scene and perform a couple of animation tasks to get a renderable animation. For these animation tasks, we'll be applying a very simple methodology to keep things nice, clean, and simple.

Why Is It Awesome?

This project will let us get acquainted with some very important things when working in CG. First of all we'll see how useful and helpful it is to create a good mesh structure when modeling a certain object, planning ahead the shape it will have in the end. We are also going to see how easy it can be to set up a nice material using just procedural textures and standard material settings.

The best part of this project is creating the storm cloud. This will let us see how powerful and easy it is to set up the volumetric material available in Blender. When combined with the technique for shaping the volumetric density using a particle system, we'll see how much control we can get for objects of this kind. The best part of it all is that by mixing various independent tools within Blender our job will be quite easy to accomplish, without having to compromise on the quality of things.

We'll also learn a simple but effective approach to create basic animations, and avoid falling into an overcomplicated workflow for them. Learning to keep things simple is very important since that will give us more space to be creative rather than having to do repetitive tasks over and over.

Your Hotshot Objectives

This project will be dedicated mostly to the fighter aircraft. The storm cloud and the final animation will be quite straightforward. Here's a summary of the steps involved in this project:

1. Modeling the fighter aircraft.
2. Shading the fighter aircraft.
3. Modeling and shading the storm cloud.
4. Assembling the final scene.
5. Animating the fighter aircraft.
6. Animating a simple lightning effect.

Mission Checklist

For this project, we're going to use a reference image showing the top view of the fighter. For simplicity's sake, we provide you with a render of the finished model, as seen in the following image; this way things are easier to set up and follow along:



We also need to download an image texture. Let's go to www.cgtextures.com and download the image named **BronzeCopper0030** (ID 11583, visible on the link address) from the **Metal** section, and **Bronze And Copper** subsection. This image will be used to create the fake reflection effect for the canopy of the fighter.

We'll be working on a single file, called `fighter-storm.blend`. Inside this file, we're going to work on separate scenes for the fighter and the clouds. Finally, we're going to assemble everything into a single scene for animating the aircraft and add the finishing touches.

Modeling the Fighter Aircraft

We are going to model our fighter by following the reference image provided. The first part of the modeling will consist of defining the *topology*, which will later allow us to perform some tweaking on the shape, rather than on the mesh structure itself. The top part of the fighter will require quite some work from us whereas the bottom part will be modeled after a very simple shape, since it isn't going to be visible to the camera. The final touches will require us to create the flaperons and stabilators for the aircraft. The engine intake, the exhaust, and all the internal parts will be ignored.

Prepare for Lift Off

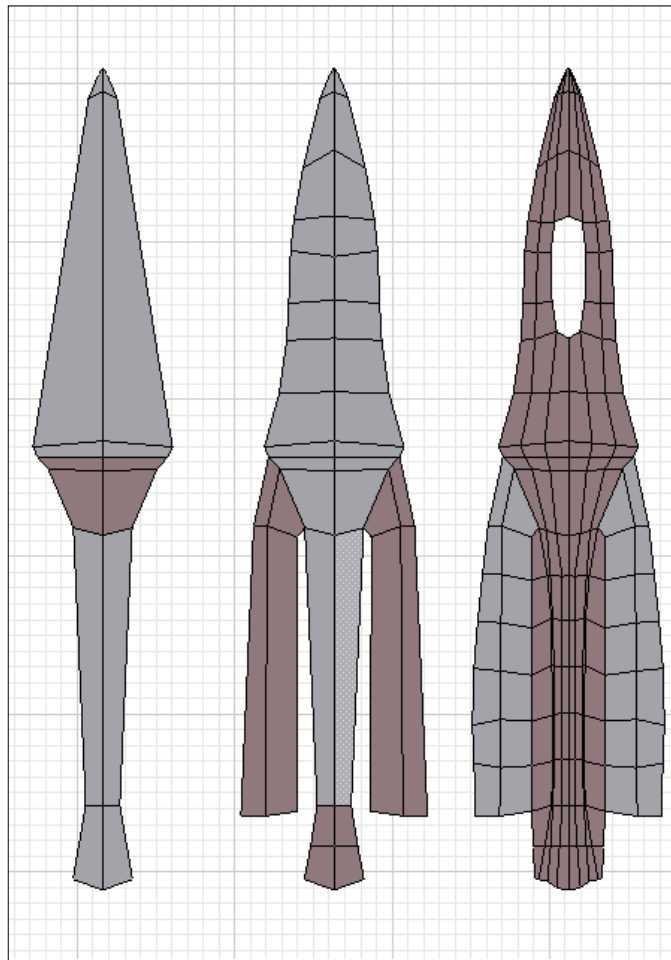
1. Create a new scene and name it `fighter`. Delete any object that could possibly be in it, to get it clear.
2. Open the properties panel (**View** → **Properties**) and locate the **Background images** section. Enable it and click on **Add Image**. Select **Top** for the **Axis** setting and click on the triangle pointing to the right-hand side, immediately below the **Add Image** button.
3. Now click on the **Open** button that appears below the **Axis** setting. In the **File Browser** window that appears, locate the render of the finished fighter and click on **Open**.
4. Set the **Size** parameter for the background image to 2.6. Having the image ready, it is time to switch to **Top View** (**View** → **Top**) and **Orthographic Projection** (**View** → **View Persp/Ortho**, once or twice). The image should appear at the center of the **3D View**.

Engage Thrusters

1. From **Top View**, add a new plane (**Add** → **Mesh** → **Plane**) and clear its location (**Object** → **Clear** → **Location**), just to make sure that it is at the center of the 3D world. Name the object `fighterBody` (**Item** panel, Properties Sidebar) and enter **Edit Mode**.

2. Select the leftmost edge of the plane and move it **1 unit** in the positive direction of the **X Axis**, this way it will be positioned right on the line showing the **Y Axis** (the green line).
3. Next, go to the **Modifiers** tab in the **Properties Editor** and add a **Mirror** modifier to the object; make sure to enable the **Clipping** option for the **Mirror Modifier**.

With the basic setup in place, it's time to start working on constructing the basic shape. For that let's take a look at the first three stages of the modeling. These will help us to understand the process better, as you can see in the next screenshot:

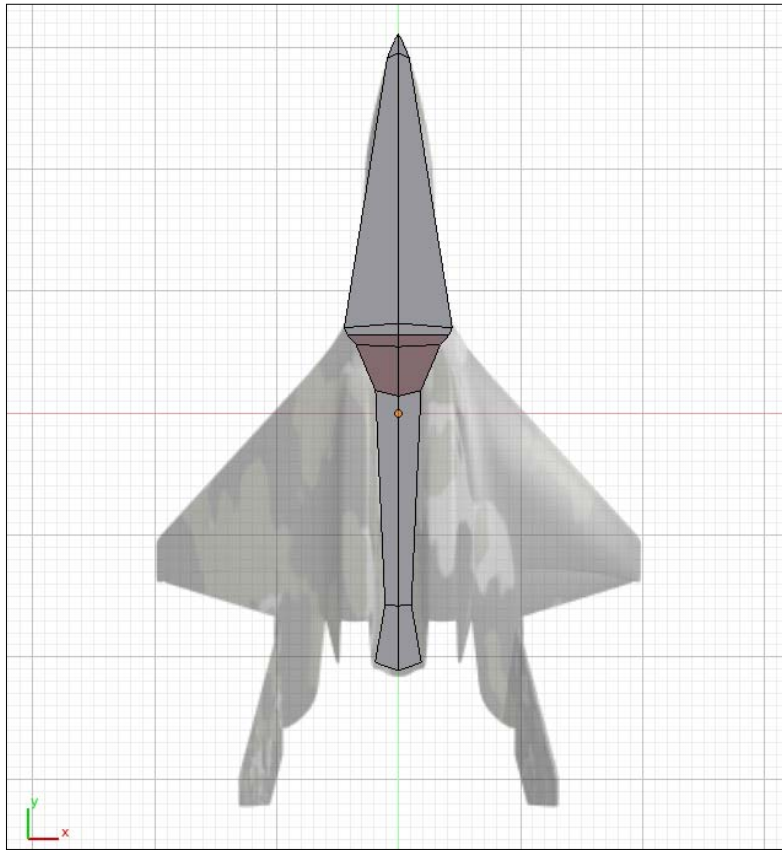


The previous screenshot shows multiple meshes together in a single image. In the 3D Viewport you should be working on a single mesh that is symmetrical on the **Y Axis**.

Since the shape at the left-hand side doesn't clearly show where everything should be located, take a peek at the one on the right-hand side just to clarify what part of the mesh must be over what part of the background image. Remember we are working in **Top View** all the time.

First stage

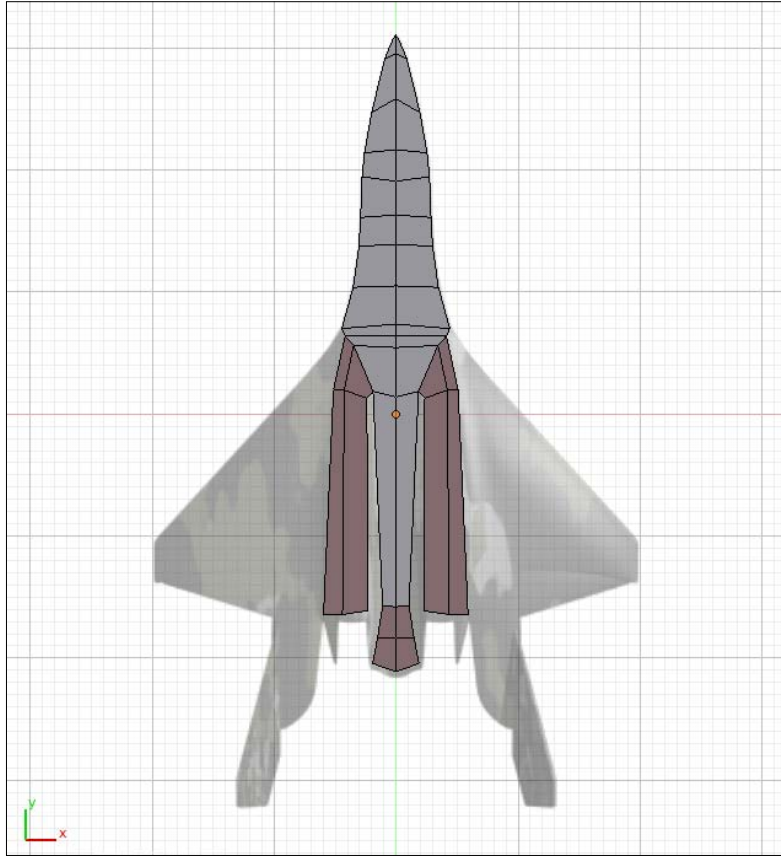
4. Select the quad and move it over to the neck of the aircraft (the place where the wings meet the rest of the body), highlighted in the screenshot showing the first three stages. Select the vertex in the right-down part of the mesh and move it to the left and up side. Next, select the upper edge and extrude it along the positive direction of the **Y Axis**, move it just a bit to get the right proportions. To get everything as expected, remember to place the mesh over the correct parts of the provided render.
5. Continue by extruding again and moving slightly in the positive direction of the **Y Axis** and then by performing a long extrusion in the same direction. Finish this part by a small extrusion, into the very tip of the nose.
6. Of course, the extrusions will give just a plain shape, so it's time to tweak the vertices by hand, one by one, so that the finished shape properly surrounds the border of the reference image. At this time, the shape of the mesh is very coarse, so we shouldn't be worried about it not following the curvy shape of the background image; the details are going to be added later.
7. The first stage is completed by selecting the edge at the very bottom and extruding it twice along the negative direction of the **Y Axis**; shape it by following the reference image to get the result shown in the following screenshot:



Second stage

8. This stage is about creating two "arms" extruded from the rightmost edges of the first two polygons created for the model. The first extrusion must stay quite close (since it's used to re-direct the next extrusion) and be tweaked a bit to conform to the reference image. The second extrusion is a long one and must go on until it's positioned at the same place on the **Y Axis** as the last extrusion from stage one. Notice the slight widening of this last extrusion with respect to the previous one.
9. Next, we need to add some detail to the "head" of the mesh. Add six edges using the **Loop Cut and Slide** tool (**Ctrl + R**) running horizontally (the number of loops can be increased by rotating the scroll wheel). Take some time to tweak the mesh so that it fits the reference image better.

10. We finish this stage by adding a small edge loop to the "tail" that we created in the first stage. All these changes are highlighted in the following screenshot:

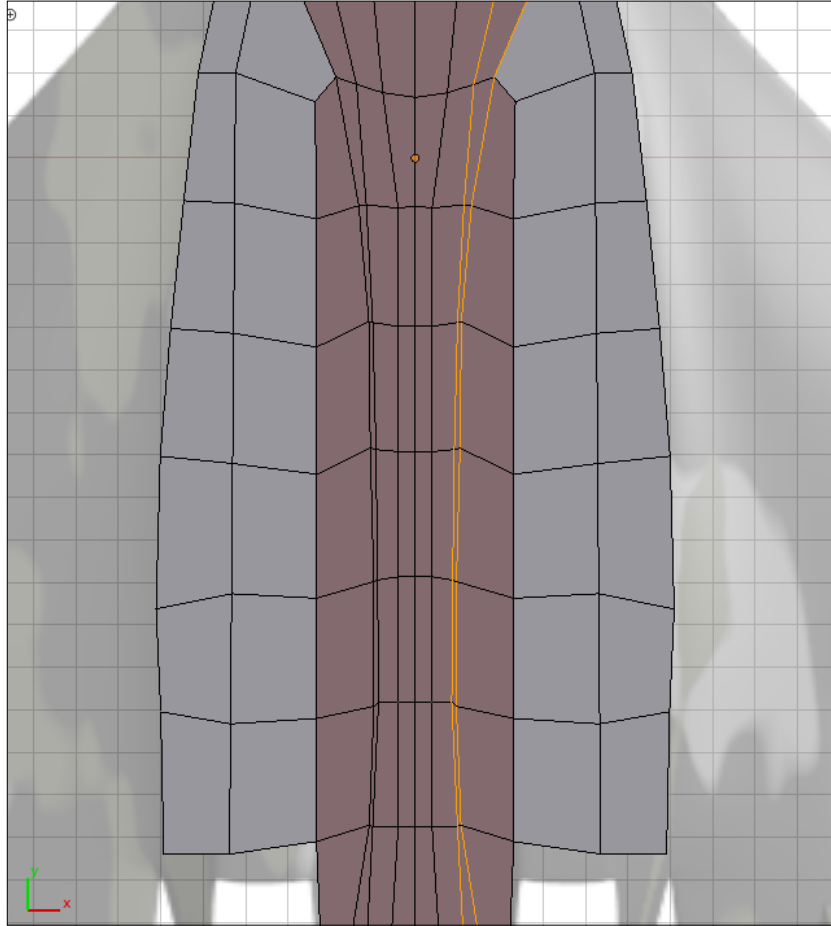


Third stage

In this stage we start adding details to the mesh:

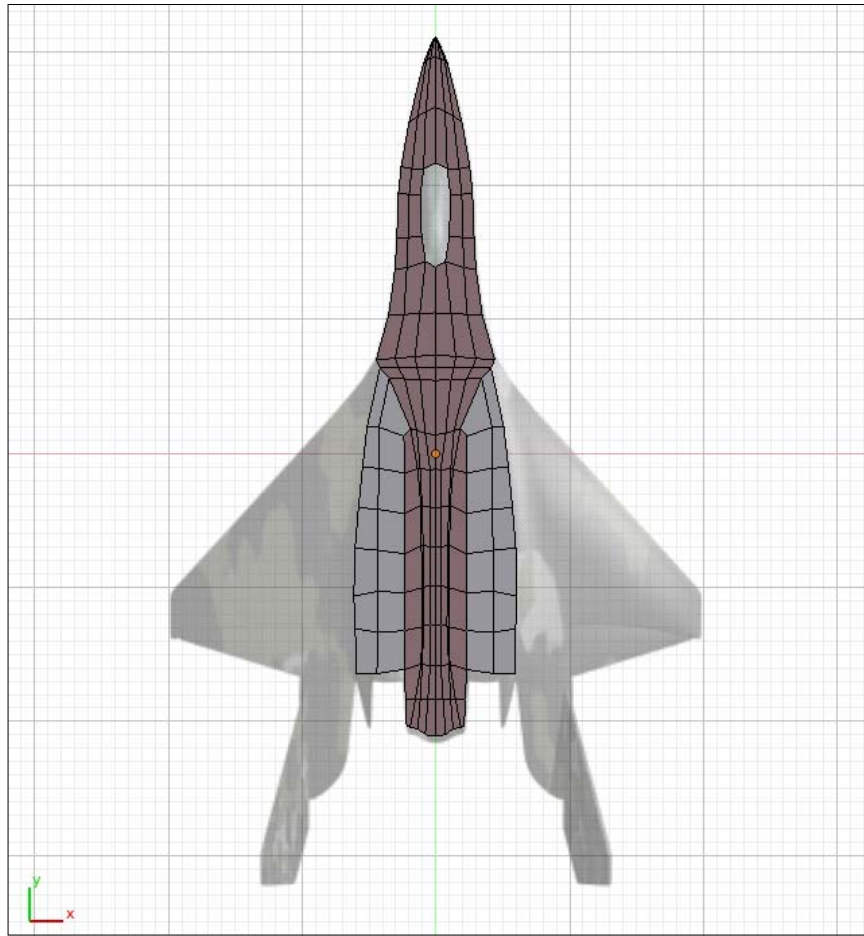
11. First we fill the space within the "arms" created in the second stage with faces (select opposing edges and press *F*) and then we add five horizontal edge loops by using the **Loop Cut and Slide** tool (*Ctrl + R*) and rotating the scroll wheel of the mouse.
12. Before moving ahead, let's tweak the new edge loops. Move the vertices and position them according to the curve that they must follow in the reference image.
13. Don't forget to add the two new faces attached to the "tail" of the model. To create these, just extrude them, vertex by vertex, on the right-hand side of the "tail" and then fill in the faces using **Mesh** → **Faces** → **Make Edge/Face**.

14. More detail needs to be added running along the **Y Axis**. Start the **Loop Cut and Slide** tool (*Ctrl + R*) and add two more edge loops running vertically, through the inner faces of the mesh. Then we perform some more tweaking with the newly created edge loops. Pay special attention to the newly created edges that were positioned quite close to the outer edges of the face loop on which the new edge loops were added. A screenshot can help us see it more clearly:



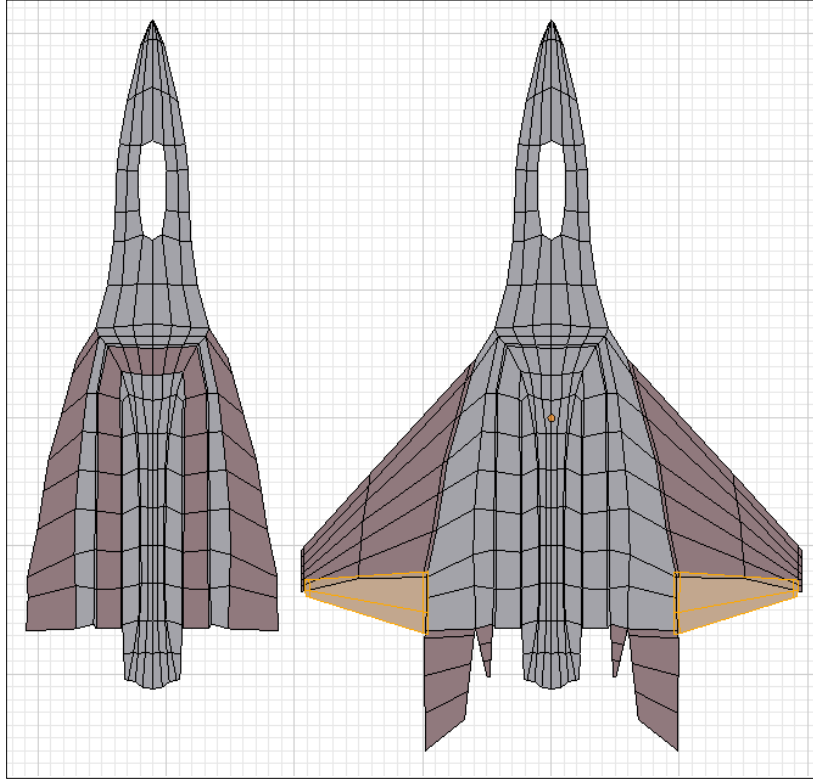
15. Now it's time to create the cockpit by deleting two vertices in the upper part of the mesh where we want it to be located and then giving the ring around the hole an oval shape.

16. Now that we have quite some detail to play with in our mesh, it's a very good idea to do some tweaking wherever necessary. It's important to do it now, since leaving the tweaking for later will certainly force us to work on a very dense mesh, which will makes things more difficult than they should be. Make sure that the mesh follows the shape of the reference image, as shown in the next screenshot:



At this point, we have quite a detailed mesh for the fighter aircraft. We can now start "feeling" how it will be tweaked into the final shape, which is very good for letting us get immersed in the task of modeling.

Now let's take a look at the two final stages for creating the top part of the model:

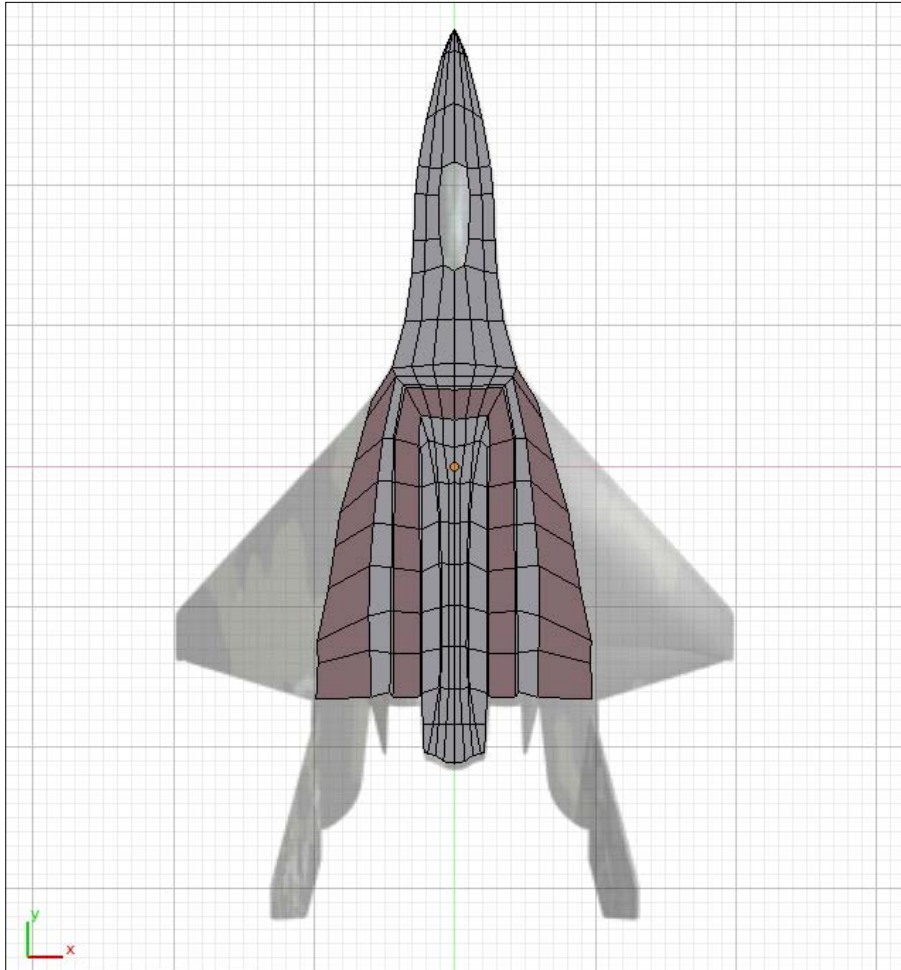


As before, the screenshot shows the new mesh parts highlighted to ease the reproduction of the process.

Fourth stage

17. For this part, we need to add two edge loops running across the inverted U shape highlighted in the screenshot. After adding the edge loops, it is necessary to use the **Edge Slide** tool (*Ctrl + E*) to move some edges closer to the "borders" of the inverted U shape. Take a close look at the screenshot to find which edges were put very close to their neighbor edge loop; this is important to get the mesh correctly prepared for some tweaking later on.

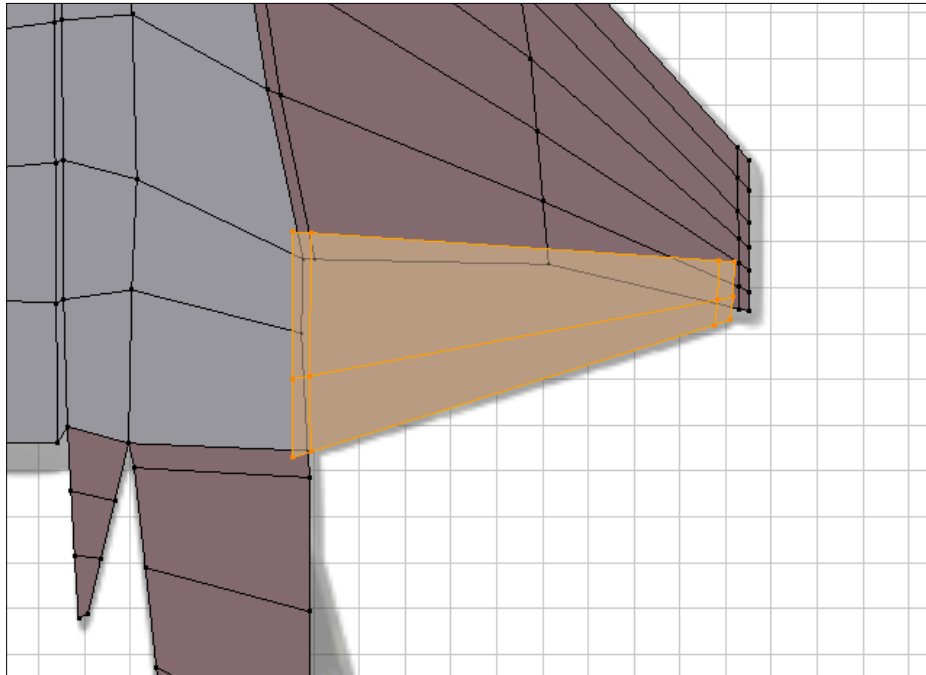
18. Another extrusion must be done on the eight edges running back from the "neck" to get the base shape from which to extrude the actual wings. The following screenshot shows the mesh's state after performing these operations:



Fifth stage

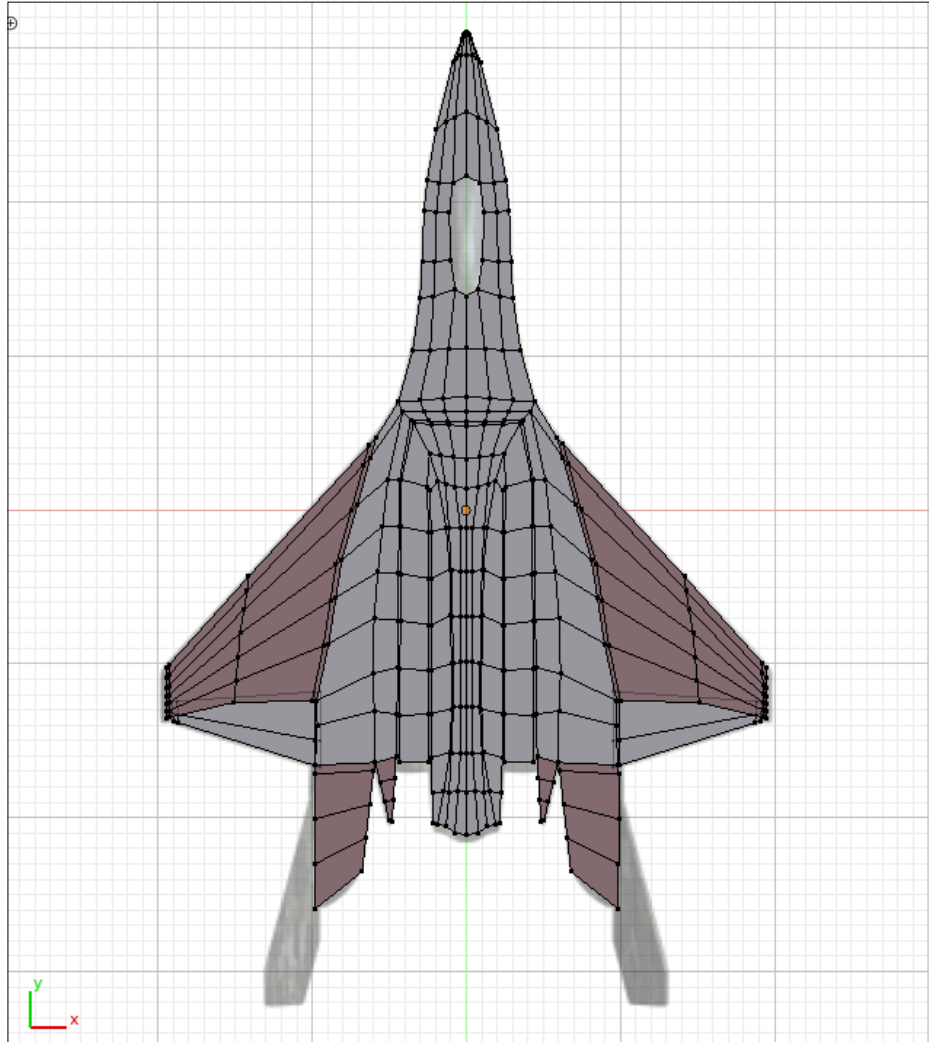
19. In this part, we just perform some extrusions on the back part of the aircraft and tweak them by following the reference image.
20. Then we perform two successive extrusions on the side of the mesh for the wing(s) and shape them accordingly.

21. After the wings' shape is finished, let's add two edge loops, running very close to the tip and the origin of the first extrusion. Take a close look at the screenshot to find the edge loops that run very close to their "parallel" edge loops.
To complete this stage, we only need to model the flaperons (the ones just behind each wing).
22. To do that, click on a free spot in the 3D View, away from the Y axis and add a Plane. Then position it close to where the right flaperon is and give it the long trapezoid shape it should have.
23. To finish, use the **Loop Cut and Slide** tool to add an edge loop running along the longest dimension of the polygon and perform the usual addition of the edge loops very close to the rightmost and leftmost border using the **Loop Cut and Slide** tool. In the following screenshot the flaperons are shown selected to ease their identification:

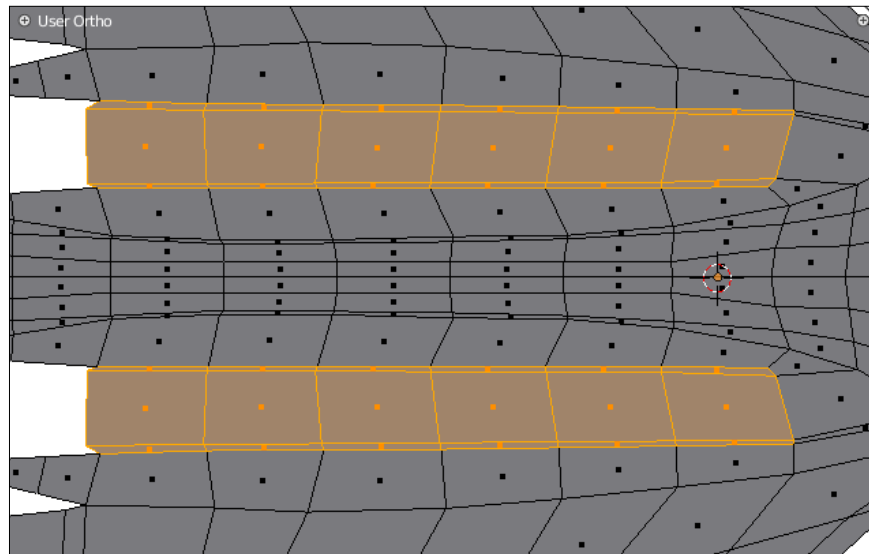


Don't forget to get this basic flaperon shape overlapping the mesh of the wing (though not physically correct). This is one of the advantages of working with CG: Using simple things to fake complex ones.

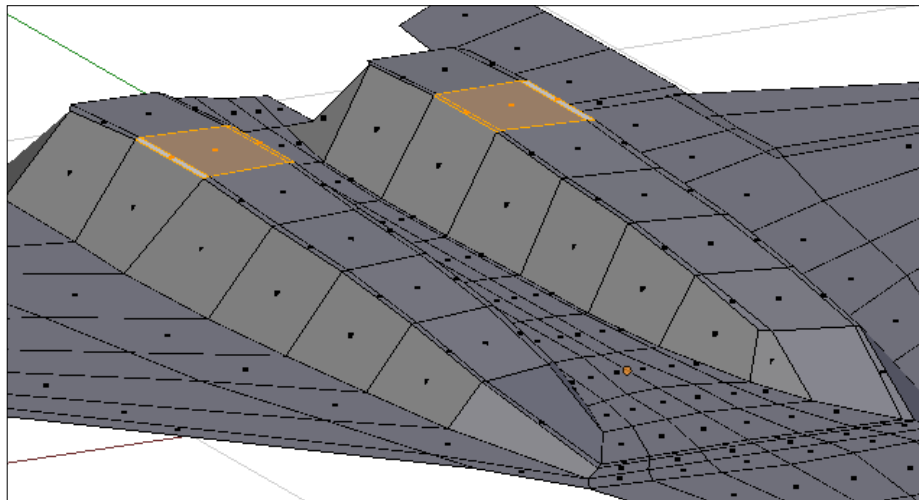
Let's check the final result of the fifth stage, as shown in the following screenshot:



After all that work, we have the plain structure from which we are going to get our nice fighter aircraft. The good news is that all the detailed work we have been doing up till now had a purpose: Allowing us to get the desired 3D shape easily by just moving around the existing polygons. So, let's start giving some volume to it by following the next screenshot:

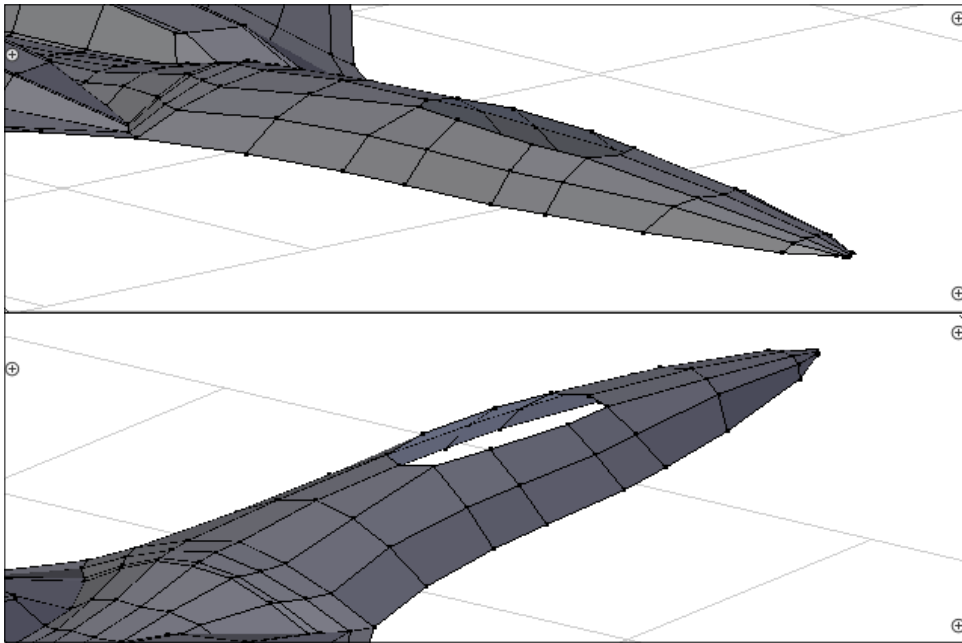


24. We'll start by selecting the faces shown in the previous image (remember that every dot represents a face in the mesh).
25. Then switch to **Right View (View → Right)** and perform a repetitive move upwards—deselect the three rightmost faces—move upwards until the shape from the side looks curvy; be careful to keep the curve looking soft. This part is the housing containing the engine and the exhaust of a real aircraft. A screenshot showing the result of these steps is very helpful:



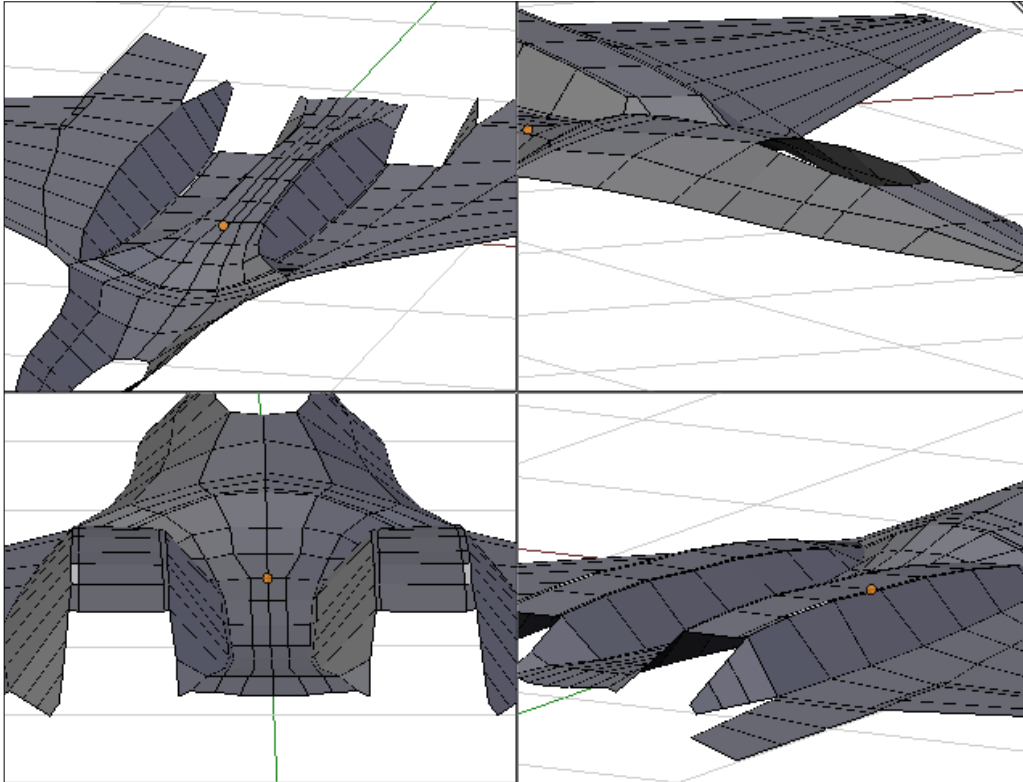
26. After giving shape to the place where the exhaust is located, we go on to tweak the mesh from the nose to the "neck", making sure that it gets a nice round shape. A good idea is to start selecting the ring defining the hole of the cockpit and moving it upwards, then tweaking vertices around to get them to form a soft and nice shape. It is important to work on the nose carefully, since it has quite a few edge loops finishing there; that way we avoid getting ugly deformations later.

It is highly recommended to rotate the view a lot while doing these tweaks, to make entirely sure that the shape that we want is the one we are getting. This is a very useful habit when working in CG; since the computer screen can only show us two dimensions, there's no other way to really know what the shape is; only rotating the view around and paying close attention to the details helps. A good way to have a clear idea on what is the shape that we are aiming for is to look at the following reference image:



27. The next thing to do is to shape the space between the two protrusions that we raised first. In this case, to give a good variation to the shape, let's put some concave curvature into it. Concave curvatures tend to be problematic because, sometimes, a small change breaks the softness of the curve. A good way to work around this problem is to look at the coin from the other side (literally), which means just rotating the view to look at the model from the downside, that way we can work more easily on it and avoid the complications of tweaking a part of the mesh that is between two protrusions.

Now is a good time to check some screenshots showing the result of all this hand tweaking:

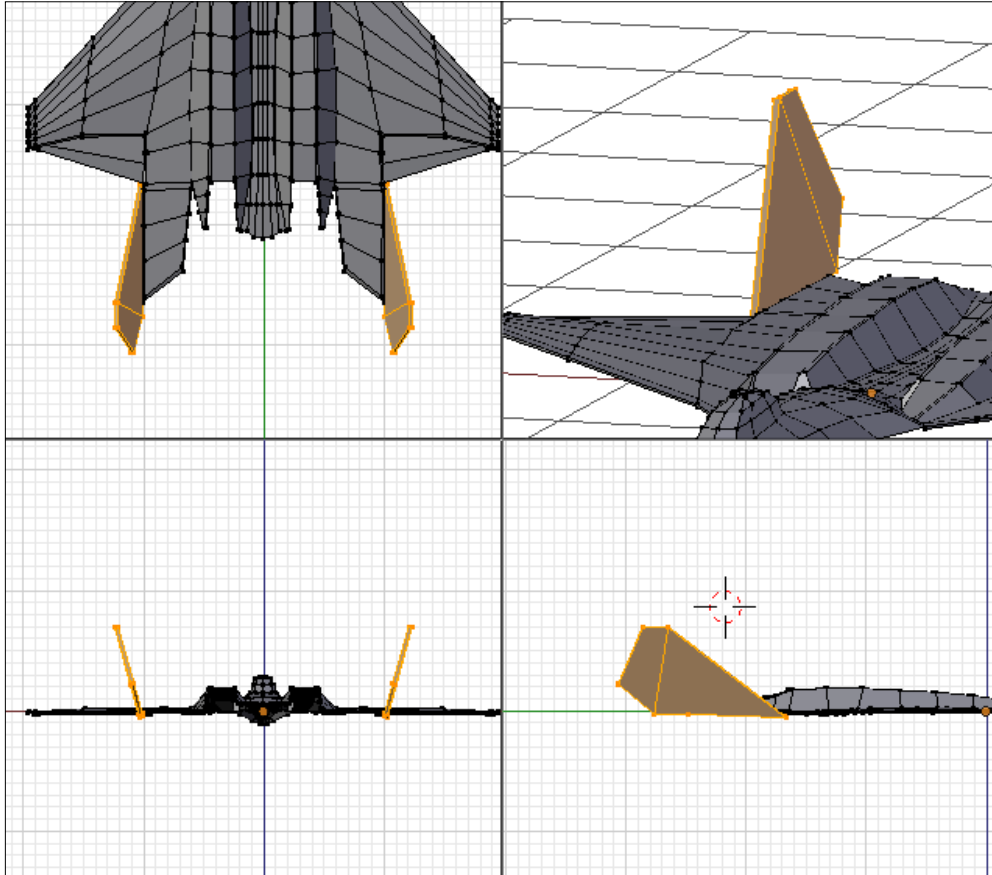


In the previous screenshot, the top left-frame shows the mesh upside down; the other three are just different views showing the shape of the various parts that we've been working on.

After all this hand tweaking, there is more coming!

28. Let's create the canopy (cabin) for our fighter aircraft. The technique is rather simple but effective: Select the ring forming the hole of the cockpit, duplicate it with *Shift + D* and do not move it from there. Then start making four successive extrudes—move upwards—scaling operations to give it a first oval shape. When that's done, perform some hand tweaking on it to refine the shape to make it look nice from any angle. Remember to not scale down the last extrusion completely to zero, just leave it small enough so as not to be seen by the camera when looking at the whole model. This is something that helps to avoid triangles, which could cause ugly deformations when we use the **Subdivision Surface** modifier.

29. To finish the top part of the model, let's create the stabilators, those two pieces that go in the tail of the model. For that, let's check a series of screenshots:



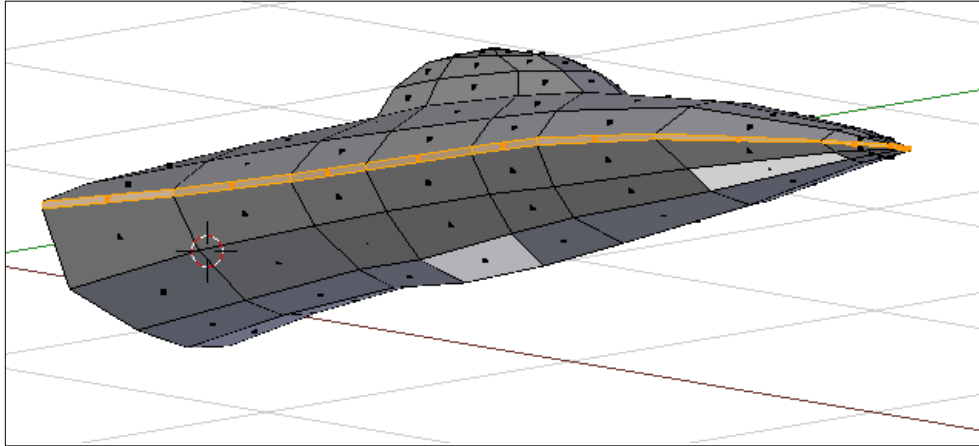
For this part, just create the shape from the right view (**View** → **Right**) starting from a simple plane, then extrude it by around 0.03 units; finally go to the front view (**View** → **Front**) and rotate it by 15 degrees, making sure that it gets oriented as shown in the previous screenshot.

30. If the stabilators were created in **Edit Mode**, then they belong to the same object as the rest of the mesh; if that's the case, select them (selecting the one on the right should also select the mirror on the left) and go to **Mesh** → **Vertices** → **Separate**, to get them into a different object.

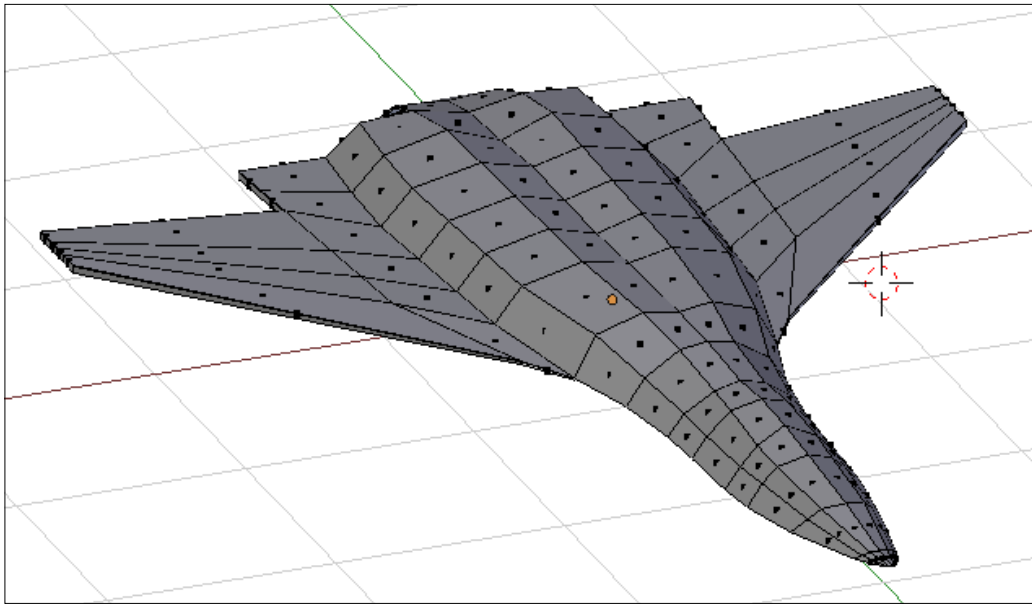
31. Go to **Object Mode**, select the new object and finish it by going to the **Modifiers** tab in the **Properties Editor** option and adding two separate Subdivision Surface modifiers. Switch the first modifier to Simple mode and set its subdivisions to 3, both in View and Render. The second Subdivision Surface modifier must be left in Catmull-Clark mode and set to two subdivisions. This trick of alternating modes for the modifiers gives the corners of the solid a nice rounded bevel.
32. Rename the object containing the stabilators to `fighterStabilators`, to keep the scene organized.

The top part of the aircraft is finished, but we still need to create the lower part of it. This time we're not going to invest so much time, because this part won't be seen in the final scene. It's enough to give a rough shape to the whole model to have a good looking appearance.
33. Let's go to **Object Mode** (if in **Edit Mode**), select the `fighterBody` object, then switch to **Edit Mode**, select the part of the mesh from the "neck" of the aircraft straight to the nose tip (except for the canopy), duplicate it (**Mesh** → **Add Duplicate**) and mirror it along the global **Z Axis** (**Mesh** → **Mirror** → **Z Global**); this will give us an inverted version of this part of the mesh.
34. Now move the mesh down in the **Z Axis** to position it near the top part of the mesh. Make sure to leave a space of around **0.02** units between these matching parts and start filling in faces between them by selecting four vertices each time and pressing the *F* key; that way we get them fully connected.
35. Let's also fill the hole that is just below the canopy by extruding the middle edge into the symmetry axis for the Mirror modifier and then add faces manually with the *F* key.

36. Finish this part by selecting the vertices in the tip of the nose, enabling **Proportional Editing** and moving it down, while controlling the radius of effect of **Proportional Editing** with the scroll wheel. The idea is to give it a shape resembling the beak of a bird; that way we break the symmetry of the model and give it a better look. The next screenshot shows the final result that we get after performing all these operations:



- Pay close attention to the shape of the bottom part, since what we need to do is perform some more tweaking on it to break the symmetrical look it has with the top part. The selection is the strip of faces created between the two manually mirrored parts of the mesh.
37. Next we need to create the missing parts of the bottom side of the model. To do this, we start by extruding the edges in the "neck" (three edges, on each side of the mirrored mesh) to **2.3** units in the negative direction of the **Y Axis**.
38. Then we use the **Loop Cut and Slide** tool to add five edge loops running across the last extrusion and perform some tweaking to get a shape resembling the shape of the upper part of the body but without all the detail used there.
39. The bottom part of the wings can be created by following the same procedure used for the top part: Perform a first extrusion on the six edges of the last extrusion and do an extrusion to the side; do some tweaking to get it to fit the shape correctly.
40. Then select the four edges that are located on the most positive part of the **Y Axis**, extrude again, and tweak the last part of the wing. To get a better sense of how this part of the mesh must look, let's see a screenshot of it:



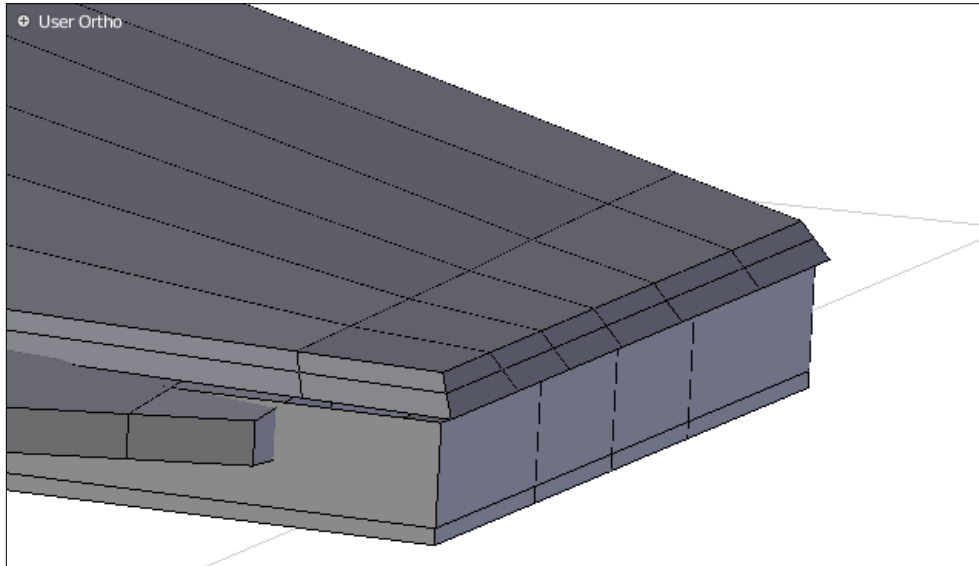
Now it's time to create the borders for the wings and get them to look solid.

41. For that, make sure that the top and bottom parts of the mesh are separated by around **0.045** units. Then select the border of the wings, all around from the "neck" of the aircraft back to the tail, do an extrusion and move it down by **0.015** units. Now we need to give this last extrusion a good, soft looking shape (when the **Subdivision Surface** modifier gets applied); for this we must get these new edges "scaled" proportionally around each of its originating edges. Since this operation is not available by default in Blender, we must go to the corners and move the vertices created by the last extrusion locking the **Z Axis** value, so that the last extrusion doesn't look straight vertical, but has a bevel of about 45° .
42. After adjusting all these edges this way, we need to add an edge loop running along the faces that form this bevel. This extra edge loop will force the subdivision to give us a nice round shape on the mesh when the **Subdivision Surface** modifier gets applied.
43. For the bottom part of the `fighterBody`, we must perform the same process: Selecting the faces around the wings, from the "neck" to the tail, extruding them upwards by **0.035** units, and finalizing the process by adding an edge loop running across all the faces created by the extrusion, but this time making sure to put the edge loop closer to the bottom part of the mesh.

After doing this there's a last fragment of mesh to give some thickness: The flaperons.

44. Select the flaperons and extrude them by **0.015** units on the **Z Axis**. Make sure they are located behind the border of the top part of the wings; don't worry if it crosses the geometry for the border of the bottom part of the wings, since we don't want a model for a close up in that place.

Explaining this in words isn't, by any means, easy; let's look at a screenshot showing what we need to do with the borders of the wings:



The mesh here shows the border of the top part of the model, notice that it isn't straight vertical, but instead it has a bevel. The bottom part of the wing has an extrusion running vertically and the flaperon is shown on the left-hand side. Perfection is not needed here, so we allow for some small errors, such as not closing the top part of the mesh with the bottom part or allowing certain faces to pass through some others. We must take this into account when performing the tweaking needed in the place where these extrusions meet the "neck" of the aircraft, thus avoiding some unnecessary headaches.

45. To add some richness to the shape of the wings (they are flat right now) we can go and raise some parts of them, tweaking carefully to produce a subtle curvature on the surface. This improves the shape of the overall object.

Objective Complete - Mini Debriefing

In this task, we used a reference image to aid us in the creation of the basic geometry from which we could shape the fighter aircraft. We started by creating an entirely flat mesh, but made sure the geometry had the correct topology that would be useful later when shaping the object.

This time we used a mixed technique: The reference image was used to copy the shape of the object seen from above, but we just performed free modeling when shaping the 3D volume of our model. This way of doing things has the advantage of helping us to get a very precise starting shape, but avoiding the repetitive process of just copying all the time from a blueprint; it's a good mix between the two ways of doing things.

If there's a word to define this task, it is... *tweaking*. If you (like me) are too lazy to do too much fiddling around and moving vertices one by one, maybe the word will scare you. But the truth is once your mind starts playing freely with the mesh, the trial and error process involved in tweaking is just a way to explore interesting options and be creative. Once immersed in the modeling process, time goes by and we don't even notice it.

Classified Intel

The modeling part for this project required quite some work from us, mostly in performing lots of small tweaks, but also on ensuring a proper *topology* for the resulting mesh.

A word on topology

Topology is just a fancy word to mean the way vertices, edges, and faces are *connected* in a mesh. A good way to understand it is by knowing that if we take the mesh that took so much time to create and *only* move the vertices around in space to form a random shape, the topology of the mesh isn't changed. Topology is about the structure of the mesh, not about its shape. Be careful, though, since a given topology determines how well the mesh will look when shaped in a certain way; so we can say that a good topology allows us to have optimal control over the shape.

Topology is a very important subject in mesh modeling. It helps a lot with high-level concepts, allowing us to decouple from the atomic elements of the mesh (vertices, edges, and faces) and start thinking in terms of how certain sets of polygons in the mesh will give us a very good control on the types of shapes that we want to create at a certain point.

To avoid becoming too extensive in this discussion on topology, let's just define some concepts the easy way: The set of faces that the **Loop Cut and Slide** tool *cuts through* is called a *face loop*; the set of edges added after using the **Loop Cut and Slide** tool, as the set of edges selected by *Alt + Left Clicking* on an edge, is called an *edge loop*; on the vertex level, a vertex connected to a number of edges different than four is called a *pole*, and performs the task of *redirecting* face loops. Take some time to analyze the mesh that we just modeled and see how the face loops, edge loops, and poles (vertices connected to three or five edges) were put in the right place to help get the desired shape.

Shading the Fighter Aircraft

With the mesh finished and ready, we just need to add some basic shading to it. Thankfully, the shader is very easy to create and looks very good on the model.

Engage Thrusters

1. Select the **fighterBody** object and switch to **Edit Mode**.
2. Then go to the **Material** tab in the **Properties Editor** and add two material slots (the button with the plus icon, on the right of the list of material slots), along with two distinct materials (select each slot and click on the plus icon next to the material name, below the list of material slots). Name the first material **fighterBody** and the second one **fighterCabin**.
3. The **fighterBody** material must be set as follows:
 - ▶ Diffuse color: 0.19 Red, 0.19 Green, 0.19 Blue
 - ▶ Specular panel:
 - Intensity: 0.3
 - Hardness: 10
 - ▶ Shading panel:
 - Ambient: 0.0
 - Cubic Interpolation: Enabled

Here we set a quite dark diffuse color. Add some specularity to it and disable the ambient lighting for this material (it controls the influence of the **Ambient** color in the **World** tab of the **Properties Editor**). Finally, we enable cubic interpolation, to give the material more contrast between the lit and shadowed parts.

4. Next, go to the **Texture** tab in the **Properties Editor** and add two textures; name the first one **lightgrayBlots** and set it as follows:
 - ▶ Texture Type: Clouds

- ▶ Clouds panel:
 - ❑ Noise: Soft
 - ❑ Basis: Blender Original
 - ❑ Size: 0.50
 - ❑ Depth: 2
 - ▶ Influence panel:
 - ❑ Diffuse color: 1.0
 - ❑ Blend: Mix
5. In the **Colors** panel, enable **Ramp** and set its color stops indicated as follows:
- ▶ First color stop:
 - ❑ Color: 0.74 Red, 0.74 Green, 0.74 Blue
 - ❑ Position: 0.60
 - ❑ Alpha: 0
 - ▶ Second color stop:
 - ❑ Color: 0.74 Red, 0.74 Green, 0.74 Blue
 - ❑ Position: 0.63
 - ❑ Alpha: 1
6. Now select the second texture slot, name it **greenBlots**, and set it as follows:
- ▶ Texture type: Clouds
 - ▶ Clouds panel:
 - ❑ Noise: Soft
 - ❑ Basis: Blender Original
 - ❑ Size: 0.35
 - ❑ Depth: 2
 - ▶ Influence panel:
 - ❑ Diffuse color: 1.0
 - ❑ Blend: Mix
7. This texture also requires a ramp, so let's enable it in the **Colors** panel and set it this way:
- ▶ First color stop:
 - ❑ Color: 0.37 Red, 0.38 Green, 0.28 Blue

- ❑ Position: 0.54
- ❑ Alpha: 0
- ▶ Second color stop:
 - ❑ Color: 0.37 Red, 0.38 Green, 0.28 Blue
 - ❑ Position: 0.56
 - ❑ Alpha: 1

These two textures are just standard cloud textures that get mapped to strong blots by using the very useful ramp option. In this case, we choose the colors to convey the idea of military camouflage.

Now let's go to the **Material** tab in the **Properties Editor** and take a look at the finished material. Quite good for just a couple of textures, don't you think?

8. The settings for the material **fighterCabin** are as follows:
 - ▶ Diffuse color: 0.66 Red, 0.72 Green, 0.76 Blue (Diffuse panel)
 - ▶ Specular panel:
 - ❑ Color: 0.71 Red, 0.83 Green, 1.0 Blue
 - ❑ Specular Shader Model: Blinn
 - ❑ Intensity: 1.0
 - ❑ Hardness: 30
 - ❑ IOR: 7.0
 - ▶ Ambient: 0.0 (Shading panel)

The settings for the material just define a bluish tone for the base material and add a strong specularity to it. We disable the "default" lighting again, by setting the **Ambient** parameter to **0**.

9. Now we must go to the **Texture** tab in the **Properties Editor**, create a new texture, and name it `fakeRefl`. Let's set it as follows:
 - ▶ Type: Image or Movie
 - ▶ Image panel: Click on the folder icon and load the texture we downloaded at the beginning of the project
 - ▶ Image mapping panel:
 - ❑ Extension: Repeat
 - ❑ Repeat: 4 X, 4 Y
 - ▶ Contrast: 1.8 (Colors panel)
 - ▶ Coordinates: Reflection (Mapping panel)

- ▶ Influence panel:
 - Diffuse color: 0.4
 - Blend: Value

For this texture, we just used an image texture with some strong dots in it to give the illusion of being reflected from the canopy. The good old trick of using the Reflection coordinates is very handy for this kind of use. Setting the Blend to Value makes the texture affect the material only by how bright or dark the color is, and not by the color hue itself.

10. Now that we are done with creating the materials, the only thing left to do is assigning the `fighterBody` material to the `fighterStabliators` object. To do this, select the `fighterStabliators` object and add a new material slot in the **Material** tab of the **Properties Editor**. Next click on the **red icon** to the left-hand side of the name of the material (below the list of the material slots) and select the material **fighterBody**. This way we use the same material in the stablators, which are a separate object from the body of the aircraft. Now is also a good time to save our file.

Objective Complete - Mini Debriefing

We created two simple materials for the fighter aircraft. The first one, `fighterBody`, was created by making some good use of ramps for the two texture channels involved and choosing the right colors, both in the diffuse property of the material and in the color stops of the ramps, to end up with a very cool military look for the body of the fighter.

The canopy was quite easy to do, by using a bluish base color to convey the idea of coldness and applying a simple texture image in reflection coordinates to fake reflections on the surface of it.

Finally, we just shared the `fighterBody` material with the `fighterStabilators` object by assigning it to the correct material slot for the object.

Classified Intel

The two materials that we created have the ambient setting disabled. This setting is used to apply a general, flat, lighting to the object; when no light reaches a certain part of the object, it is "lit" only by the ambient color defined in the **World** tab of the **Properties Editor**. In places where the light is affecting the object, the final look gets a tinted look from the ambient color; the ambient setting of the material controls how much effect this general lighting has on the material it is set for. In this case, we don't want this behavior for the material, so we just turn it off.

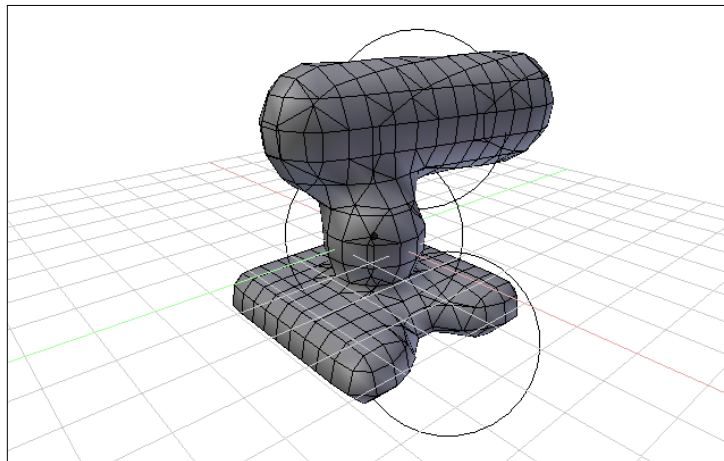
Modeling and Shading the Storm Cloud

This part of the project is quite nice. We are going to create a stormy cloud, having full control of its shape, and creating a volumetric material for it. The volumetric materials in Blender allow some very nice effects, and this cloud is going to look cold, very cold. The best part of all this is that the techniques used will allow us to have it done in a matter of minutes, which definitely helps a lot in enjoying the ride.

Prepare for Lift Off

Let's create a new scene in our file and name it `env`, which stands for environment. If the scene has some default object, let's delete it and leave only the camera.

This time we're going to use a very neat type of object in Blender, called **Metaballs**. **Metaballs** come in different shapes and act like mercury, because they tend to "melt" when located close enough. Let's take a look at an object composed of metaballs:



In this example object, we can see a meta plane (lower part), a meta ball (over the plane, touching it), and a meta capsule (above, touching the ball). We can see how the three elements get melted when they are in contact. There's a small catch with the image: There is a ball that is not visible explicitly, which is acting in negative mode to subtract a part of the plane (below and to the right). It is highly advisable to work on metaballs by first creating an element, going into Edit Mode for it, and then continuing by adding the other elements needed. Also, there's a common problem when trying to select these, so it's recommended to switch the **Viewport Shading** to **Wireframe** to perform the selection of a specific element of the whole set.

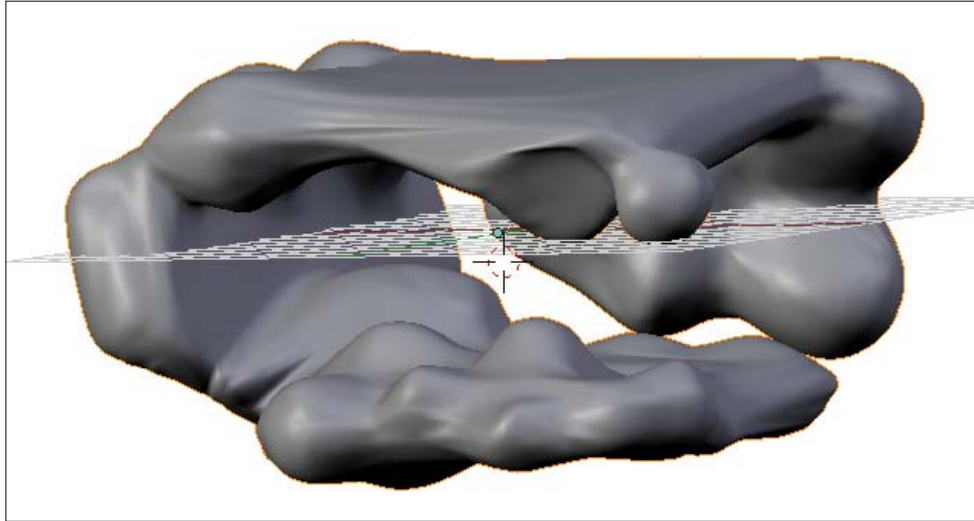
Engage Thrusters

1. Make sure that the 3D Cursor is located at the origin of the 3D World; in case it isn't, go to **View** → **Align View** → **Center Cursor** and **View All**.
2. Then add a **Meta Plane** (**Add** → **Metaball** → **Meta Plane**), select it, switch to **Edit Mode**, select it by clicking on the green circle in its center, and scale it by a factor of **5** locking the **Z Axis** (S, *Shift* + Z, then type "5").
3. Duplicate it (**Metaball** → **Duplicate**) and move it upwards by **6** units.
4. Then start working freely by adding metaballs and giving them different scales, and positioning them to create a connection between the upper meta plane and the lower one. Around 15 metaballs should be enough to play with until the shape of the whole object resembles the one that we could expect of a storm cloud. The only important thing now is to give it a general shape, since fine tweaking will be added next.
5. Now, switch to **Object Mode** and convert the metaball object to a mesh (**Object** → **Convert** → **Mesh from Curve/Meta/Surf/Text**).
6. Next, go to the **Modifiers** tab in the **Properties Editor** and add a **Decimate** modifier, set its **Ratio** to **0.1** and click on **Apply**. Now we have a low resolution mesh that resembles the shape of our clouds.



If the decimate modifier shows a message complaining about the mesh being "Non-manifold", the solution is to switch to **Edit Mode**, enable **Vertex Selection Mode**, deselect all vertices, press the *Space* bar and type "Manifold", then click on the entry that says "Select Non Manifold". The next thing to do is locate the vertices that are now selected and delete them. This should fix the problem in case it occurs.

7. To give some more randomness to the shape of our cloud, let's switch to **Sculpt Mode**. Go to the **Tool Shelf (View → Tool Shelf)**, select the **Grab** brush from the list (click on the brush preset image), and play a bit with the brush to create irregular deformations on the mesh (use the *F* key to change the size of the brush if needed). Hopefully, we'll have a very nice and random mesh that will be our storm cloud. Let's refer to the following screenshot of a suggested shape for it:



The screenshot is of a subdivided version of the cloud, just to get it looking nice. The actual mesh to be used for it shouldn't be subdivided, since that will only create an unnecessary overhead for Blender to process and render it. Notice that this mesh occupies the whole default grid of the Blender scene. Let's make sure that this is the actual scale of our storm cloud mesh, to keep it in accord with the indications.

8. Now is a good time to name the object as `cloudShape` and also to save an incremental version of the file.
9. The next step is to add a particle system to the mesh. Go to the **Particles** tab in the **Properties Editor** and add a new particle system to the object. Name it **cloudDensity** and set it as follows:
 - ▶ Emission panel:
 - Amount: 100000
 - Emit From: Faces
 - Distribution: Jittered
 - ▶ Normal: 0.0 (Velocity panel)
 - ▶ Physics panel:

- ❑ Size: 0.3
- ❑ Random size: 0.38
- ▶ Render panel:
 - ❑ Emitter: Disabled
 - ❑ Unborn: Enabled
 - ❑ Render: None (options row below Unborn)
- ▶ Display panel: "None" option
- ▶ Gravity: 0.0 (Field Weights panel)

What we just did was create the set of particles that will define the fine details of our cloud. We choose to emit the particles from the faces (surface) of the mesh, distributing them with the Jittered option. We also made sure that they won't move (velocity 0 and field weights 0) and set the size of the particles and the randomness for it. Finally, we disable the standard render options for particles, turn the emitter invisible and make the unborn particles appear (these three options are in the **Render** panel).

At this point, we have the particles that will supply the density information to the volumetric shader for our cloud, but we don't yet have an object to be rendered as the cloud itself (using the particle system to modulate its shape).

10. Let's create a cube and switch to **Edit Mode**; now start scaling it until the `cloudShape` object is entirely contained within it.
11. To avoid difficulties with the navigation of the scene, switch to **Object Mode**, go to the **Object** tab in the **Properties Editor**, locate the **Display** section, and set the type to **Wire**. This way the 3D Viewport won't get cluttered by a boring giant cube.
Now it's time to create the volumetric material that will complete the recipe to get our cloud.
12. Go to the **Material** tab and add a new material slot. To make sure this material is only used by this object, click on the button with the plus icon to the right-hand side of the name of the material (below the material slot list).
13. Name the material `cloud`, choose "Volume" from the row of options just below the material name and set it as follows:
 - ▶ Density panel:
 - ❑ Density: 0.0
 - ❑ Density scale: 3.0
 - ▶ Shading panel:
 - ❑ Scattering: 5.0
 - ❑ Emission: 0.3

- ☐ Emission color: 0.08 Red, 0.11 Green, 0.25 Blue
- ☐ Reflection: 1.0
- ☐ Reflection color: 0.29 Red, 0.29 Green, 0.29 Blue
- ☐ Transmission color: 1.0 red, 1.0 Green, 1.0 Blue
- ▶ Lighting panel:
 - ☐ Method: Shaded
 - ☐ External shadows: Enabled
 - ☐ Light cache: Enabled
 - ☐ Resolution: 20
- ▶ Transparency panel: Raytrace
- ▶ Integration panel:
 - ☐ Method: Randomized
 - ☐ Step Size: 0.05
 - ☐ Depth Cutoff: 0.01

We start by putting the default value at 0 for the density of this material and increase the scale to 3, to have a stronger density when the texture drives it to 1. In the shading panel, we set the options to give the cloud the cold look: The reflection color gives a general gray tint to the shader, the emission parameter indicates how much of the emission color will be emitted by the cloud, and the scattering value makes the light scatter very fast in the shader (thus giving the idea of thickness). Finally, we set the Lighting to shaded, which simulates self shadowing of the volumetric material, and set the resolution to 20 in order to avoid very long render times. The Raytrace option for the transparency gives the best results and the Step Size for the Integration was chosen to get a good balance between the quality and the render time.

To connect the information from the particle system with the volumetric material, we need to use a special kind of texture.

14. Go to the **Texture** tab, create a new texture slot, and add a new texture of **Point Density** type; name it `cloudPoints`, and set it as follows:

- ▶ Point Density panel:
 - ☐ Source: Particle System
 - ☐ Object: cloudShape
 - ☐ System: cloudDensity
 - ☐ Cache: Emit Object Space
 - ☐ Radius: 0.3

- ☐ Falloff: Soft
- ☐ Softness: 2.0
- ▶ Turbulence panel: Enabled
 - ☐ Influence: Static
 - ☐ Noise Basis: Voronoi F2-F1
 - ☐ Size: 1.0
 - ☐ Depth: 4
 - ☐ Strength: 2.0
- ▶ Influence panel:
 - ☐ Density: Enabled, value 1.0
 - ☐ Emission Color: Disabled
 - ☐ Blend: Add
 - ☐ RGB to Intensity: Enabled

We started by defining the source for the Point Density information to be a particle system then selected the object to use and its associated particle system. We also set the Falloff to soft in order to get a nice falloff for the density information of each particle. The turbulence panel allows us to enable the most important part of this setup: Variation in the density. What we set here is just a simplified texture modifying the density information of each particle providing it. Finally, we just add the result from this texture to the density parameter of the material (remember the base material has a density of 0) and make sure that the emission color is not affected by our texture.

Objective Complete - Mini Debriefing

What we needed was a cloud that we could control, both in shape and in appearance.

Since we just need a mesh to give the general shape of the cloud, the easy way to accomplish it was to use the nice "melting" property of metaball objects, model a basic shape, and just convert it to a mesh to be able to reduce the number of polygons by using the Decimate modifier. Once that was done, we added some randomness to the shape of the mesh by hand, by using the very useful sculpting tools in Blender. With the mesh ready, the only thing needed was create a particle system with a high number of particles and set its physics properties so that the particles would be visible and stay still all the time.

Finally, we just added a cube that would be the actual rendered object, using a volumetric material, and created a setup for controlling the density information for it. For that we used a very neat method that requires the existence of a particle system to modulate the value of the density for a given volumetric material. The final touches are provided by the turbulence option in the **Point Density** texture type.

Classified Intel

Volumetric materials are special because the simplifications applied to develop the standard shading models only apply for objects with a defined surface. Given this situation, there was a need to develop techniques to render volumetric materials in CG, and thankfully, we now have such an implementation available in Blender.

The main reason for volumetric materials being much more heavy to render than a standard material is that the renderer must simulate the actual process of light traveling inside the volume, and its scattering inside and outside, taking into account the variations in density, scattering, and so on. Common surface materials are rendered much faster because the renderer only needs to calculate the interaction of light with the surface.

Assembling the Final Scene

With all the pieces ready, it's now time for us to link all the objects to a single scene, and set up the final elements for render. Since we have all the objects in the same Blender file, getting them into the same scene will be very easy.

Engage Thrusters

1. Start by creating a new scene and naming it **main**.
2. Then go to the fighter scene, select both the **fighterBody** and the **fighterStabilators** objects, and go to **Object** → **Make Links...** → **Objects to Scene...** → **main**.
3. Now go to the **env** scene, select both the **cloudShape** and the giant cube objects and perform the same action to link them into the main scene. Next, go to the main scene.

Comparing the relative sizes of the cloud and the aircraft, it is clear that they are not correct.
4. To fix this, let's create a new **Empty** object (**Add** → **Empty**), clear its location (**Object** → **Clear** → **Location**), and its rotation (**Object** → **Clear** → **Rotation**).
5. Then make sure that the **fighterBody** object is located at the center of the 3D World, select both this and the **fighterStabilators** object, and then select the recently created **Empty** object, now parent the objects to the **Empty** object by going to **Object** → **Parent** → **Set**.
6. Name the empty object **fighterParent** and set its scale to **0.3**.



Almost for sure, the cloud object will make it hard to select the other objects in the scene, so it's recommended to select objects with the *Alt + Left Click* combo (*Alt + Shift + Left Click* to accumulate); this shortcut shows a list of objects near the place we clicked, to be able to select exactly what we want.

Next, we need to separate the objects in different layers, to be able to separate the lighting for the storm cloud from that of the aircraft.

7. Make sure that the `fighterParent` object (and its children) are associated only with layer 1 of the scene (select the objects, press *M* and click only on the desired layer), and that the `cloudShape` object and the cube surrounding it are associated only with layer 2 of the scene.



Remember that the visibility of layers in the 3D Viewport is controlled by the double row of buttons in the header of the 3D View. To enable multiple layers, click on them while holding down the *Shift* key.

If we take a look at our scene, there's something important missing: Lighting!

8. Go to layer 1 of the scene (make sure it's the only enabled layer in the 3D Viewport layer buttons), add a point lamp, and set its properties as follows:
 - ▶ Energy: 0.3
 - ▶ Distance: 30
 - ▶ This Layer Only: Enabled
 - ▶ Shadow: No Shadow
 - ▶ Location: 7.3 X, -3 Y, -0.8 Z (suggested; it really depends on the cloud shape)

We just put this lamp close to a corner of the grid, planning to get the aircraft flying around. Its low energy value is caused by the need of having a very dim lighting on the fighter while flying through the storm.

9. Now select the **fighterParent** object and set its properties (**Transform** panel, Properties Sidebar) as follows:
 - ▶ Location: 6.4 X, 4.3 Y, -2.0 Z
 - ▶ Rotation: 3.43° X, -19.2° Y, 117.2° Z

The purpose of getting the aircraft located and rotated like this is to prepare it for the animation that we're going to create later. The aircraft is going to fly from the right-hand side of the shot to the left. We'll be setting up the camera in the next few steps.

10. Now switch to **layer 2** and add two sun lamps with these settings:

- ▶ Lamp 1:
 - ❑ Rotation: 100.8° X, -57.2° Y, 37.4° Z (**Transform** panel, Properties Sidebar)
 - ❑ Energy: 2.0
 - ❑ Color: 0.07 Red, 0.11 Green, 0.19 Blue
 - ❑ This Layer Only: Enabled
- ▶ Lamp 2:
 - ❑ Rotation: 74.9° X, 0° Y, 135.6° Z (**Transform** panel, Properties Sidebar)
 - ❑ Energy: 4.0
 - ❑ Color: 0.6 Red, 0.8 Green, 1.0 Blue
 - ❑ This Layer Only: Enabled

These two lamps serve the purpose of adding some general lighting to the storm cloud. The colors of the lights are chosen to give some nice contrast (warm versus cold) and the energy values to modulate their effect on the final look of the storm cloud.

11. We also need to add a camera and set it to a good place for getting a good shot; recommended settings are as follows:

- ▶ Location: 8 X, -9.6 Y, 1.6 Z
- ▶ Rotation: 70.2° X, 0° Y, 42.8° Z

12. One last thing left to do is just setting the world options to have a good blue background for our scene. To do this, go to the **World** tab in the **Properties Editor** and adjust the settings as indicated:

- ▶ World panel:
 - ❑ Blend sky: Enabled
 - ❑ Horizon Color: 0.0 Red, 0.08 Green, 0.18 Blue
 - ❑ Zenith Color: 0.01 Red, 0.02 Green, 0.06 Blue
- ▶ Ambient Occlusion panel: Enabled
 - ❑ Factor: 0.10
 - ❑ Mode: Add

These settings for the world will just give us a nice blue gradient in the background to give some color to our dull scene.

13. Let's also play a bit with the `c1oudShape` object and its surrounding giant cube to position them correctly for the camera. This is a good time to save an incremental version of the file and launch a render job to see how the scene is coming along.

Objective Complete - Mini Debriefing

This part was just about getting the objects in a common scene, correctly scaled and located for an initial render. The more complex part of it was organizing the objects in separate layers and adding the corresponding lights to each layer.

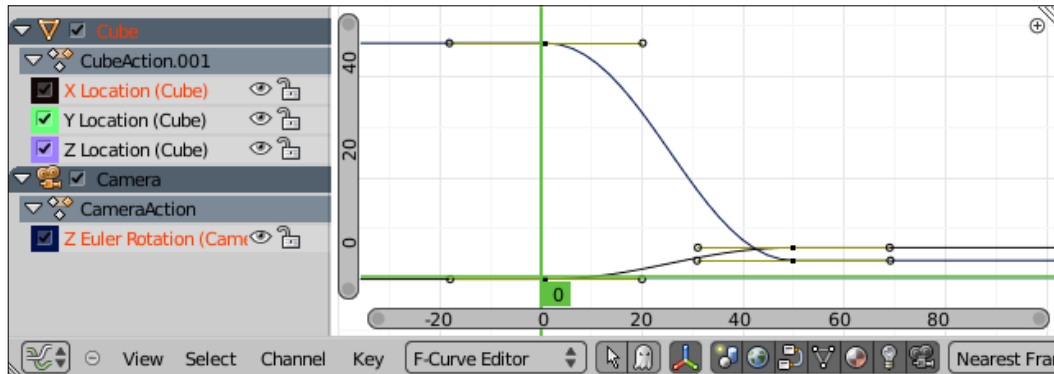
Animating the Fighter Aircraft

At this point, we have the scene ready to be animated, which we will do right now. For simplicity's sake we're only going to animate the aircraft and take the time to understand the steps involved in it. The purpose is to have the aircraft describing a soft arc while passing around the storm cloud. The animation of the aircraft is going to be done on the `fighterParent` object.

Prepare for Lift Off

To get the 3D Viewport clean for work, it is recommended to turn off layer 2, and leave only layer 1 enabled. We also need to split the 3D View to get a new window and set it to the **Graph Editor**, to be able to edit the animation curves that we are going to create. It's also very useful to keep at hand the **Properties** for the object (**View** → **Properties**). Finally, set the Start for the animation to 1 and the End to 51 (Timeline editor, the one with the play and record buttons); this way, we can easily move to the beginning and the end, when needed. Don't forget we are going to animate only the **fighterParent** empty object, and not the actual mesh objects.

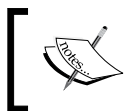
Before getting on with the task, let's take a couple of minutes to know the bare minimum about the Graph Editor. First, let's look at a screenshot:



The Graph Editor is composed of two regions: On the left-hand side there's the animation channels' list (hierarchically organized, grouped by the object they belong to) and on the right-hand side we have the region where the curves are shown and edited. The idea is this: Each property of an object (X Location, Z Scale) that is animated will have a separate channel assigned, and each channel has a curve that controls the changes on its corresponding property, over time. The Y axis (vertical) in the right-hand side region represents the value of a given property and the X axis represents time (in frames).

There is a special kind of cursor in this editor, represented by the crossing lines with the attached label showing the frame number. To know the value of a property at a certain frame, we just put the center of this cursor on the point of the curve that we want to inspect, and the horizontal line will let us easily read the value on the Y axis of the region. The curves themselves are composed of special points, called control points, which represent *keyframes* of the animation. Each control point has two handlers attached to it, and we can select and edit either the entire control point or just one of its handlers.

Of course, this editor has some special commands to work with, but for the sake of our project, we're only going to perform some basic editing, which is as intuitive as using the same methods to edit objects in the 3D View (right-click for selecting handlers, G key for moving control points or handlers, and so on). As for the creation of the animation channels (and their curves), it will be done easily from the widgets on the **Transform** panel of the **Properties** Sidebar.



Notice that the selection of the animation channels must be done by left-clicking on the channel name, and not by right-clicking. If you've defined a different set of keyboard shortcuts, this can be different.

Engage Thrusters

1. Make sure that you are at frame 1 of the animation (value of the field to the right-hand side of the End setting of the Timeline) and find the Transform properties of the object (Properties Sidebar).
2. Then right-click on one of the buttons of the Location values and select **Insert Keyframes**.
3. Now let's change the frame to 51. Move the empty object along its local Y axis by around 12 units (G key once, Y key twice, then type "12") and insert another set of keyframes at this position in time. To test the first keyframe, click on the play button in the **Timeline** window; the aircraft must be moving forward in a straight direction, perfectly aligned with its nose.

We now need to give some curvature to the path that the aircraft is describing. For this, there is a very handy tool that will help us.

4. Go to the **Object** tab in the **Properties Editor** and find the **Motion Paths** panel, set **Start** to **1**, **End** to **51** and **Step** to **3**, then click on **Calculate Paths**. Now we have a nice visual representation of the path for the `fighterParent` object, to add the desired curvature to the trajectory.
5. Let's go to the **Graph Editor** and find the curves for the X Location and Y Location channels (select the channel to highlight its curve on the right-hand side). Once found, select the rightmost handler of the first keyframe for each curve and move it to the right and then down; it is recommended to move them one by one.
6. Now it's time to perform some trial and error changes: Go to the **Motion Paths** panel, click on the **Clear Paths** button, and then on **Calculate Paths** to let Blender show us the new shape of the trajectory. Then edit the handlers some more for the curves and redraw the paths using the **Motion Paths** controls. We must do this until the shape of the trajectory suits our needs (it should look concave, seen from the camera); remember not to give it a strong curvature, since that doesn't make sense for an aircraft flying so fast.

Once we have got a good shape for the trajectory, we can see that the aircraft isn't rotating following the shape of the curve.

7. To solve that, let's go to frame 1 and make sure that the nose of the aircraft is correctly aligned with the path (edit only the **local** X and Z rotations, not the local Y rotation), and insert a set of keyframes on the **Rotation** property of the `fighterParent` object.
8. Next, go to **frame 51** and rotate the object only on its local **Z Axis** (R key once, Z key twice) until it gets correctly aligned with the last part of the path, and insert another set of keyframes for the **Rotation** property.

Now we can play the animation to see how well the nose of the plane follows the shape of the path, and we find that in the middle of the trajectory, the nose will, almost for sure, be misaligned.

9. To correct that, stop the playing animation, go to **frame 30**, find the animation curve for the **Z Euler Rotation** property of the object and move the rightmost handler of the first keyframe to the right-hand side until the object in the **3D Viewport** gets correctly aligned. At this point, our animation has the look that we want for it.
10. As a last task, make sure that the aircraft orientation looks okay when seen from one of its sides while going along the path. In case it is not correctly aligned, go to frames 1 and 51, rotate it only on its **local X axis** (*R* key, *X* key twice) and insert a rotation keyframe for that channel, to overwrite the current keyframe value in each case.

That's it; with a couple of keyframes and making use of the curve handlers for the animation curves, we got a simple but nice trajectory for our aircraft.

11. At this point, it's very likely that the `cloudShape` object isn't located in the best place for the fighter to pass besides it; a few tweaks on the position of the `cloudShape` object and its surrounding giant cube will correct the problem. Also, make sure that the camera is shooting at an interesting angle and do not show the aircraft standing still in the beginning or in the end. Saving the file now would be quite useful too.

Objective Complete - Mini Debriefing

This part of the project was only about creating the small animation for the aircraft, to get it moving around in the shot. This task required us to work with the Graph Editor for manipulating the animation curves.

The workflow that we followed has the advantage of guaranteeing that the motion and rotation will be smooth, since we used just two keyframes for the start and end of the animation. If we had started by creating three or more keyframes for this animation, the task would have certainly become more complicated than what is really needed in this case. A general rule of thumb is to keep the number of keyframes low, thus avoiding making something more complex than what is really needed.

The trial and error we applied when working on the X Location and Y Location curves is made possible by the very nice Motion Paths calculation that Blender offers. This tool is very useful for this kind of workflow since it lets us work more intuitively, and not just try to guess how the curves will interact to define the final 3D shape of the trajectory.

Animating a Simple Lightning Effect

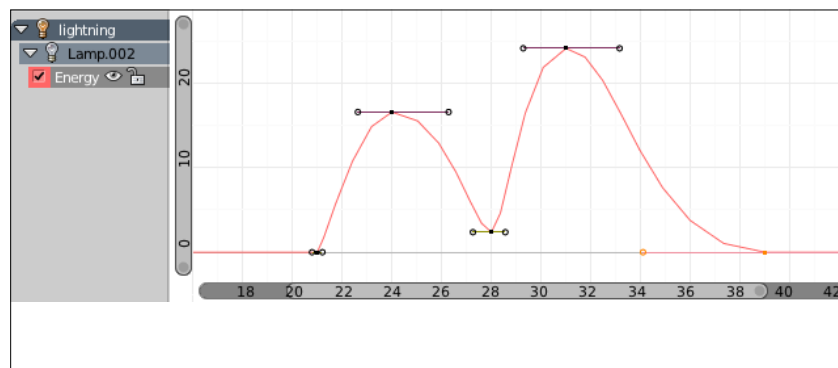
To finish the look of our nice scene, let's add a simple effect of lightning shining inside the cloud. For this, we will just need to work with a single lamp and add a simple animation for its energy property.

Engage Thrusters

1. Go to **layer 2** of the main scene and add a point lamp, setting it up as follows:
 - ▶ Color: 0.73 Red, 0.84 Green, 1.0 Blue
 - ▶ Energy: 0.0
 - ▶ Distance: 3.0
 - ▶ Sphere: Enabled
 - ▶ This Layer Only: Enabled
 - ▶ Shadow: No Shadow
2. Now go to **frame 21** and add a **keyframe** on the Energy property for the lamp (Right-click and select **Insert Keyframe**). Next go to **frame 35** and perform the same operation.

Let's go to the Graph Editor and find the curve for the Energy channel of the lamp; we will find a horizontal curve with only two keyframes.

3. We just need to add three new keyframes by selecting one of the existing control points, duplicating it (**Shift + D**), and then moving it accordingly. The desired shape is shown in the next screenshot:



Notice that the peak value for the curve is **20**, which is quite a high energy value. This is because the strong scattering of the cloud dims the light too fast, so a high energy value is needed.

4. We can take some time and make some test renders to see how well the lightning effect inside the clouds looks; almost for sure, we will need to move the lamp so that it is located inside the cloud, but not too far from the cloud surface.

Objective Complete - Mini Debriefing

This very short task allowed us to add a very basic lightning effect to the storm cloud, using just a simple lamp, and animating its energy value.

The workflow for animating the energy value was simply creating the first and last keyframes of the animation and then adding three more keyframes by hand directly in the Graph Editor. This is an easy and fast way to do it, since adding keyframes from the Blender UI would have also required some more tweaking later.

Mission Accomplished

The results of a few simple techniques are quite nice to behold. Except for the modeling part, this project is quite easy to accomplish and the results are very good.

In the modeling task, we worked hard to get a very good shape on the top part of the aircraft and applied some topology concepts to make sure that the final result would be the expected one. This task showed us the importance of always looking at our models from different angles and inspecting the shape at all times to get good results.

The shading part went by very fast, but that doesn't mean that our shaders are bad; in fact, the **fighterBody** shader looks quite good and it only required a couple of minutes to set up. The good thing about shading is that when the procedural textures are enough to accomplish certain tasks and we know how to use them for the task at hand, the time required to set them up is almost negligible.

The volumetric material settings went by quite fast too, though it required quite a few parameters to get it working the way we need. Anyway, the possibility of setting up this nice storm cloud with a few parameters for the shader is really cool. We also made sure that the cloud had a nice random look by applying some turbulence to it. The cool trick in this part was using a particle system attached to the faces of a mesh to control the density of the volumetric material; this gives us almost total control on both the shape of the cloud and the final render appearance.

Finally we just added a couple of finishing touches by animating the trajectory for the aircraft and putting a simple lightning effect inside the storm cloud. In this part, we just applied some simple workflows to avoid overcomplicating the task.

And here it is, our project in all its glory:



You Ready to go Gung HO? A Hotshot Challenge

There are quite a few ways in which our animation can be improved, let's take a look at some of them:

- ▶ Adding lightning striking from one part of the cloud to another. This would require adding compositing to the animation.
- ▶ Adding a second aircraft and creating a path for it to move between the clouds.
- ▶ Animating the camera to make the shot look more natural.
- ▶ Adding an exhaust plume left by the aircraft while flying, using a second particle system.
- ▶ Giving some dynamics to the cloud by playing with the physics properties of the particle system to get the particles moving around just slightly and slowly.

Project 3

Construct a Car Using Modeling and Shading

The Internet is full of 3D car models with varied looks and styles. Quite a few users are fans of this type of art and have big collections of models that they themselves have created. If we think about it, we can see that modeling a car is a very good opportunity to get a first look at some basic ideas of topology, as well as get a nice model to work with at the same time; so, let's go pimp our own ride!

Mission Briefing

The project is about modeling a car without getting into much detail. We will pay special attention to the topology (structure) of the mesh, which is very important in case we want to add fine details to the model later.

We will take this as an opportunity to try a method of modeling that can be very handy when doing an iterative process, since we will be creating separated patches (groups of connected polygons, not to be confused with patches from **NURBS** modeling) for the different parts of the mesh. Our main concern at the beginning will not be handling a single, complex patch of polygons; instead, we will be doing some very easy modeling with small patches and keep the polygon count low most of the time. This way, we can concentrate on what's most important in the beginning: Topology and form.

The final part of the modeling process is just learning how to stitch the different patches correctly and get nice, continuous surfaces when the subdivision surfaces modifier is applied on the mesh.

Finally, we'll be working on creating a few materials, one of which is a material built using nodes.

What Does It Do?

With this project, we will be able to do some very nice poly to poly modeling, which allows us to do some hands-on work, which is very rewarding when the model starts to take form. It definitely feels like molding with our hands.

The first purpose of this project is to get acquainted with the basic analysis of topology for a simple form. The selected car has some features that make it easy to model using a mesh modeler, such as Blender. This aspect is very important, because if we try to model a very complex form at the same time that we are learning the basics of topology, things may become too complicated. Instead, we decide to work on a model that has a basic shape, but looks very nice when we build it.

In the shading part, we will be creating a moderately complex shader using the material's node system. The material will allow us to learn some basic methods of mixing simple materials in order to get visually richer results.

Why Is It Awesome?

Topology is definitely the coolest part of this project. The key will be to keep the poly count low for as long as possible, so that a good topology will help us out when adding a subdivision surfaces modifier, which will create a very nice, high-resolution mesh by using our low-poly mesh as the input.

When correctly applying topology in a model from the beginning, the real complexity of the job is left to the computer. The idea is to have a simple mesh that the subdivision surfaces modifier can use to produce a high resolution mesh, with all the features that we want. That's the key, and mastering this is important in order to really enjoy mesh modeling.

Our model is mechanical, yet the knowledge that we will acquire with this project is very useful for modeling organic shapes too. So this project really gives us a very nice opportunity to improve our modeling skills.

On the shading side, we will produce a few shaders using some basic concepts to create them. However, the really neat part is creating the shader for the car paint, where we will work with the node editor to mix few simple materials into a complex and visually rich material. Being able to compose complex materials from basic ones is a very powerful tool, since the level of control that can be achieved for the final shader is quite high and the process to create the shader isn't that complicated. It's all about *layering* the components and using the right mixing methods to get the desired effect.

Your Hotshot Objectives

Our project will be divided into six parts; the first three will be the longest and the last three won't take as much time to do. These steps are as follows:

1. Creating the initial patches for the model.
2. Creating the doors.
3. Joining the patches.
4. Creating the wheels.
5. Shading.
6. Simple render setup.

Mission Checklist

For this project, we will need two images: One to be used as the background in the final render, and the other to be used as the blueprint for the actual modeling. These images and the other files from the project should have their own folder, to keep things simple.

The image for the background must be downloaded from www.cgtextures.com. You can go there and get the **FullSkiesDusk0001** texture (in the **Dusk** subsection of the **Skies 360** section of the site). This time, we need a full sky map because it can give us much more information for the purpose of our render, though we are not going to strive for a perfectly mapped sky (we don't really need it). When the available layouts for the image are shown, choose the normal sky map and not the cube map. The image ID (readable in the URL to which the thumbnail must be pointing) is **23746**; this way it's easier to get the correct image.

To download the blueprint, go to www.the-blueprints.com, visit the cars blueprints section, and in the section for Volkswagen, look for a blueprint of the **Volkswagen Golf 5** (page 2).

To ease the search, the URL for Volkswagen car blueprints can be used: <http://www.the-blueprints.com/blueprints/cars/vw>. Make sure that the downloaded blueprint has at least the top, front, and right views of the car. Save the file as `vw-golf-5.jpg`.

To avoid having problems when setting up the blueprint in Blender's 3D Viewport, let's perform a small tweaking and verification on the downloaded image. Download Gimp and open `vw-golf-5.jpg`; Gimp is suggested, but any program capable of working with layers should be enough for our needs.

The purpose of working with Gimp is twofold. First, we need to make entirely sure that the different views of the car have the same size. Secondly, we also perform a bit of organization to be able to use only one image as the blueprint in Blender's 3D Viewport. Start by identifying the parts of the original image that correspond to the top, front, and side views of the vehicle. Then duplicate the layer three times and cut each individual view on its own layer.

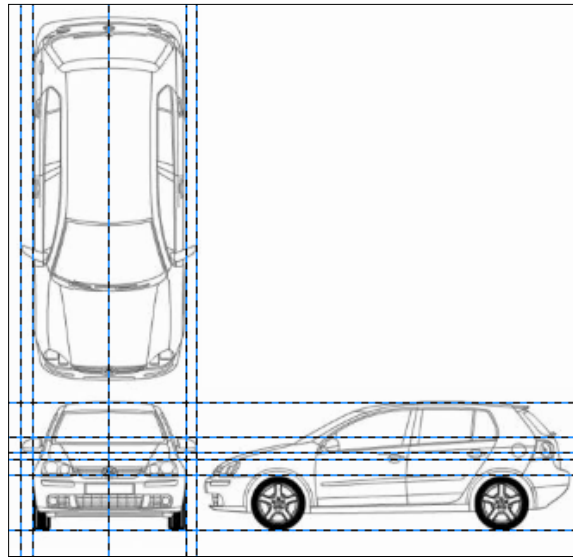
The cut in Gimp can be performed as follows:

1. Using the **Rectangular Selection Tool** (*R* key), select the layer in the **Layers** window (*Ctrl + L*) and draw a rectangle encompassing the part to be preserved.
The selection in Gimp does a masking to indicate which pixels of the image are going to be affected. In this case, we used rectangular selection since it gets the job done correctly and quickly.
2. Invert the selection (*Ctrl + I*), and cut it (*Ctrl + X*). (In case the cut portion becomes white and not transparent, simply add the alpha channel to the layer before cutting by right-clicking on the layer's thumbnail in the **Layers** window and selecting the entry from the menu.)

By inverting the selection, we tell Gimp that instead of affecting the pixels originally selected, it must protect that region and affect the previously protected ones. This way we can later cut the pixels outside the rectangle.

3. Having the three pieces ready, hide the original image layer and resize the image canvas to 200 percent by going to **Image** → **Canvas Size**, and setting the width and height to twice the original value; make sure to not resize the layers' images.
Resizing the canvas is needed just to get more space to work, so there's no need to resize the actual layers' images.
4. Now pick each one of the layers created previously and rotate or flip them so that, when they are put forming an L shape (the top view above, the front view in the corner, and then the right view), they are coherent. Imagine folding the arms of the L into the computer screen to form a three dimensional corner such that the actual object lies inside that "box".
In this case, we assume the L shape has two "arms" (up and right) and a corner, each being one of the views previously cut from the original blueprint. The need to check for coherency is because the side view of the original blueprint must be flipped horizontally to get it to appear correctly as the right-hand side view of the car.
5. The last step is to just align the images correctly using line guides, so that the top and front view are aligned by vertical lines, and the front and right views are aligned by horizontal lines. The guides in Gimp are created by clicking and dragging from one of the rulers (horizontal or vertical) into the image.

The next image shows the final result of aligning the images correctly using line guides; notice that the line guides let us do both jobs, verifying alignment and scaling or sizing simultaneously.



Save the image as `vw-golf-5-prepared.png`.

Creating the Initial Patches for the Model

We will create this model by using a simple but helpful technique, especially useful when using blueprints. We will be creating separated, low poly mesh patches and forming the basic shape of the model with them. By using this technique, we reduce the probability of having to rework big parts of the mesh if we make any mistakes while defining the basic structure of the mesh. It also helps in keeping the number of polygons in the finished model as low as possible, something important when we start working on polishing some corners.

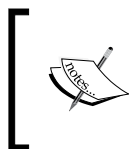
Prepare for Lift Off

This first task of the project needs a preparation step: Let's put the `vw-golf-5-prepared.png` image as the background for our 3D Viewport.

1. Start Blender and save the file as `car.blend`, in the corresponding project folder.
2. Now activate the quad view mode by opening the **Properties** sidebar (**View** → **Properties** or the **N** key) and clicking on the **Toggle Quad View** button located in the **Display** panel. Make sure to enable both the **Lock** and **Box** options, to have the three main views nicely aligned. Notice that these options for the quad view mode will block the rotation of the three main views (not the camera/user view) so we can only pan and zoom those views around.

3. Now locate and enable the **Background Images** panel. Click on the **Add Image** button and then click on the triangle pointing to the right-hand side that appears in the rectangle representing these specific background image settings.
4. On the Axis drop-down menu, select **Top** and click on the button below that has the **Open** label.
5. Select the image `vw-golf-5-prepared.png` from the hard disk. Just after selecting the image, only the top view must show the background image. Now it's time to move it to get the Y axis passing along the line of symmetry of the drawing. To do that, modify the X parameter for this background image, until it gets correctly positioned.
6. To set up the background image for the front view, we must follow the same steps, except that we just need to pick the image already loaded (button on the left-hand side of the **Open** button) and choose **Front** in the Axis drop-down menu. This image also requires changing the X value to get it symmetrically aligned, this time with respect to the Z axis (blue line in the 3D View). A good way to check that things are being set up correctly is to pan the front view upwards and the top view downwards, and check if the background images are aligning correctly.

The last background image must be set for the right view, using the same image and positioning it as needed for the three images to be aligned *simultaneously*. This last part of simultaneous alignment of the three images may sound weird, so let's just use a simple method to get it done: After making sure that the top and front views have their backgrounds aligned vertically, we need to locate a point that is easily identifiable in the three views of the car. Click on the top view and then the front view until the 3D Cursor gets correctly located on both the background images. Once that's done, it's just a matter of adjusting the X and Y offsets for the background image in the right view to get the image located in such a way that the 3D Cursor is marking the exact same point in the three views.



Here's a helpful trick to get the images aligned fast: The settings for the top and front images should have the same (or very close) X value, and the settings for the front and right images should have the same (or very close) Y value.

Yeah, quite a bit of fiddling to just get a few images positioned correctly, but without this step it is not possible to make real use of the background images for copying the shape into 3D. Remember that in the modeling tasks we will be working almost all the time using the quad view mode and that all the geometry we are going to create will be part of the same object, so we must be in Edit Mode when adding a new geometry. Saving an incremental version of the file now would be a good idea.

Engage Thrusters

We'll start the modeling by creating the basic patches from which to work out the car; remember the idea is to create a sketch first (using simple patches of mesh) so that we can create the initial shape very fast and later work out the joining and detailing. Let's begin as follows:

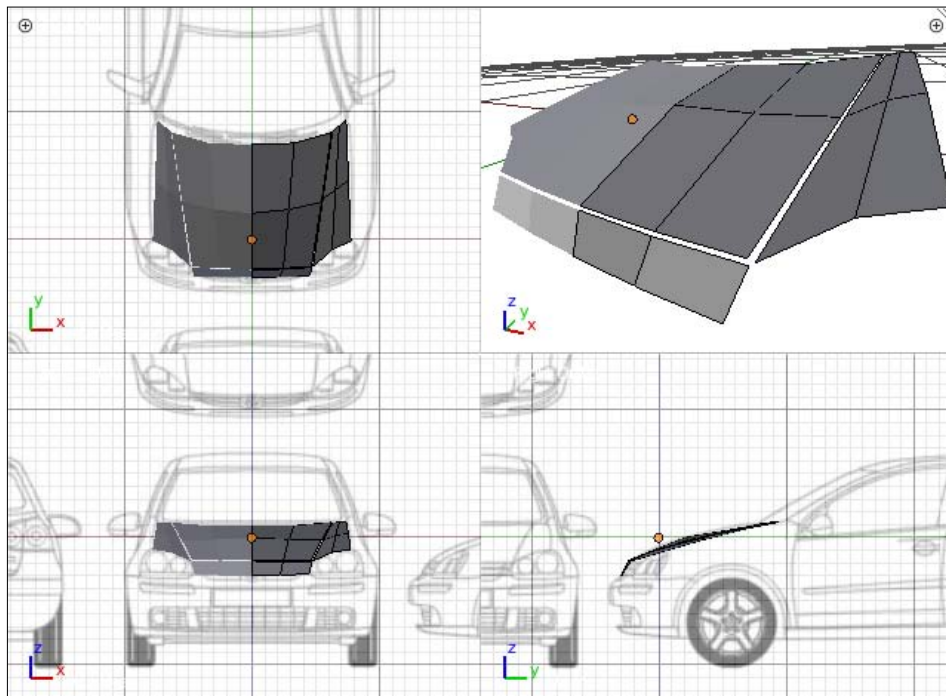
1. Reset the position of the **3D Cursor** (**View** → **Align View** → **Center Cursor and View All** or press *Shift + C*) and add a **Plane** (**Add** → **Mesh** → **Plane**). Then go into **Edit Mode** and move the cursor until it's positioned entirely on the right-hand side of the Y axis (green line).
2. Now let's add a **Mirror Modifier** by going to the **Modifiers** tab of the **Properties Editor**; the Axis for mirroring must be set to X and the **Clipping** option must be enabled.
3. Next, let's work on **Top View** and move the two leftmost vertices of the plane that we added, one by one, to be located where the Y axis intersects the hood of the car; the other two vertices must be located on the corners of the hood.
4. Since we performed these operations in **Top View**, we cannot guarantee that the Z axis value for each vertex is the correct one; to solve this we go on to work on the **Front View** region, moving the vertices only along the Z axis so that they follow the background image correctly.
5. Next, we need to add a small bit of detail to this very basic hood shape by cutting two edge loops using the **Loop Cut and Slide** tool (*Ctrl + R*), located in the **Tool Shelf** sidebar (**View** → **Tool Shelf**). The first edge loop must be cut along the Y axis (vertically) and the second along the X axis (horizontally), since the operation must be performed in the **Top View** region). To finish this patch, just tweak the position of the last vertices added from the Right View to give them slight curvature, according to the blueprint.
6. We continue by adding a second patch, this time located on the side of the hood. The process is similar to what we did for the first patch: Position the 3D Cursor close to where the patch will be and add a plane (remember we are working in Edit Mode), lower its size and then position each vertex to the correct spot according to the Top View background image. Follow that by moving the vertices only along the Z axis (Front View) to get the final form of the patch. Continue by adding a couple of edge loops (one vertical and one horizontal, in the Top View) with the **Loop Cut and Slide** tool and finish by tweaking the new vertices so that the patch gets a nice shape.



Since we are using the Mirror Modifier, it's possible that at some point we have vertices "stuck" on the Y axis. Since that "stickiness" is caused by the **Clipping** option of the modifier, it can be circumvented by temporarily disabling this option. It is important to not disable it permanently, since it helps a lot in keeping the symmetrical parts of the mesh connected.

7. The third patch will be a simple plane, shaped to cover the front region of the car that has the Volkswagen logo. This time the plane can be added in the Front View and we only need to add one edge loop, running vertically (Front View). Let's not forget to tweak the vertices using multiple views, to make entirely sure that the patch has the correct shape and is positioned at the correct spot in the 3D world.

Let's take a look at how these first three patches look. Pay special attention to how they have some empty space in between them; this is needed because at this point we are just creating a "sketch" of our model, so when we need to change things, it can be done easily.

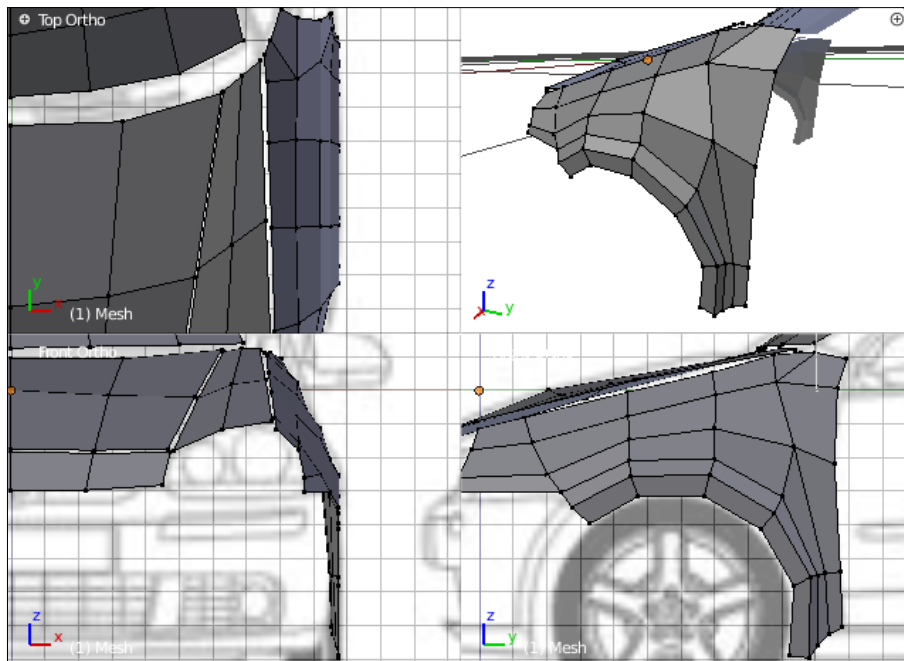


Now we are going to create three more patches for our initial "3D sketch".

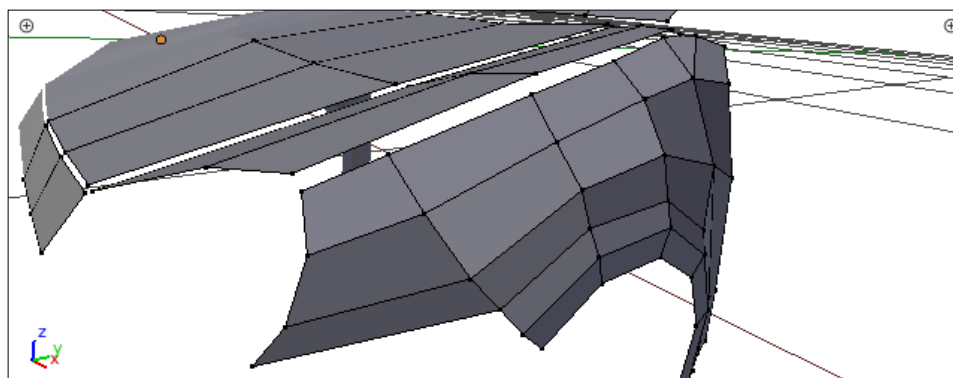
8. Let's add a plane in the Front View and accommodate it to the shape of the windshield, then go to the Top View and finish positioning the vertices correctly. The last thing to do is just add a bit of detail by cutting two edge loops vertically and two horizontally (Front View), and tweaking as necessary.
9. We must then create the patch for the roof of the car. To do this, just add a plane in the top view and tweak the positions of the vertices to cover the whole roof. Then correct the Z coordinate of each vertex by using either the front or the right view and finish by adding two edge loops (**Loop Cut and Slide** tool, or press *Ctrl + R*), one horizontal and one vertical (Top View) and making the appropriate tweaks.

The next patch is going to be the front fender and it requires some detailed work because we need to get its topology correct, so we don't have to redo it later.

We need to add a circle to start shaping the patch, but the circle must be located correctly, so we start by locating the 3D Cursor by using the front wheel.
10. In the Front View region, click on the right-hand side border of the wheel, just guessing where the center will be; then in the right view region, click on the correct position in order to get the 3D Cursor centered according to the wheel.
11. Now add a circle in the Right View, set its number of vertices to 12 and set its size so that it properly surrounds the drawing of the wheel. Since explaining the next few steps in words is quite complicated, let's look at the screenshot of the finished patch, and then continue explaining the process.



12. As can be seen in the previous screenshot, the front fender starts as a circle surrounding the wheel, from which we delete some vertices, to leave only six and shape them to form less than half a circle, taking care of distributing them evenly.
13. When the arc is shaped correctly, we select it, perform one extrusion from the right view and scale it up a bit to form a flat ring.
14. We now extrude again and press the *Escape* key to avoid moving the extruded edges, then scale them out a bit and go to the front view and move the extrusion a bit to the left-hand side.
15. Another extrusion must be made in the right view (with the last extruded vertices selected) and tweaked manually to get the shape shown in the screenshot.
To continue the construction of the front fender patch, we must use more extrusions, but this time let's take a look (in the right-top view of the screenshot) at the two vertices that are connected to five edges (these are called poles); because of these vertices, the extrusions must be separated into parts.
16. Let's select the row of edges until the vertex that will be a pole is reached, extrude the selection and perform some tweaking.
17. We must now select the row of edges following the pole, extrude them, and tweak the positions of the vertices.
18. It's then time to create the polygon filling the space above the pole. Simply extrude one vertex from one of the neighboring faces, move it to the final location and form a new face by selecting it and the other three vertices close to it, and pressing the *F* key.
19. Let's use this process to construct the remaining parts of this patch. Taking a look at another screenshot, from a different angle, can help a lot to make better sense of the shape being constructed:



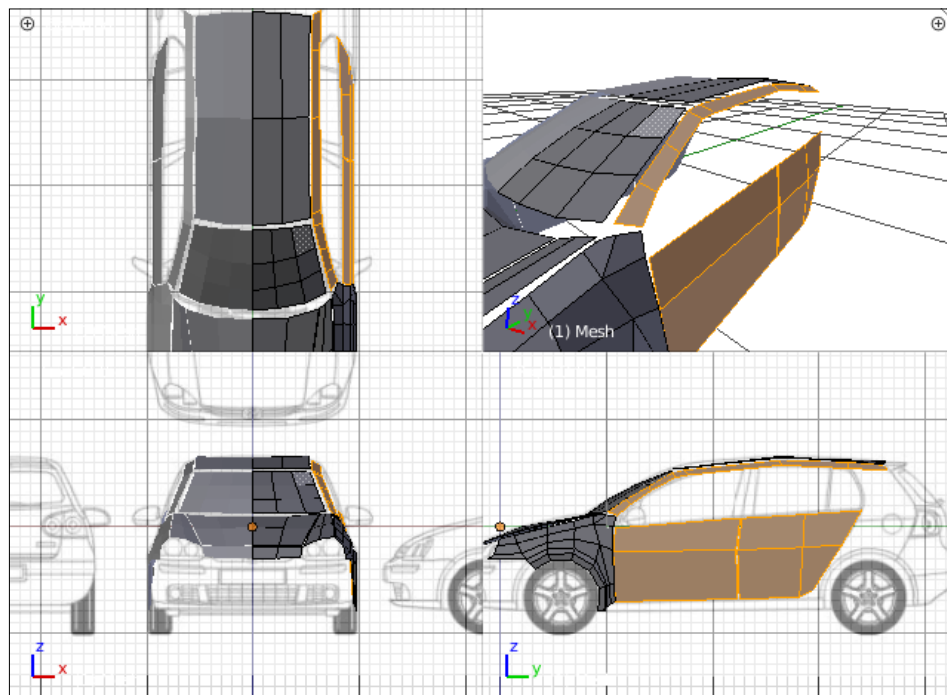
In the previous screenshot, the space between patches can be seen clearly, which is important in order to visualize what the position of this patch is, with respect to the one next to it.

20. As you can see in the center of the image, there is a last, simple extrusion that must be done and tweaked to help shape the place where the headlights will be.

The next step is to model the basic patches for the doors (without the glass panes and their frames) and the thin strip of mesh that runs along the side of the roof, shaping the bigger frame for the individual frames for the windows of the doors.

21. For the doors we will work in the right view. Start by duplicating the three lower edges (running vertically) of the front fender that will be touching the finished doors' patches. Move the duplicate a bit in the positive direction of the Y Axis and then perform one extrusion. Move it to the right until it is located where the first door must end, then duplicate it, and move it a bit to the right (just to separate the two doors).
22. For the second door, we must perform two successive extrusions: The first one must end where the lower part of the door starts running diagonally up and to the right-hand side. The second extrusion's three vertices must be tweaked manually so that the triangle-like shape to finish the door is achieved. Apart from using the blueprint for guidance, it is very useful to use the user view to rotate the model around and check for any errors or whether some fine tweaking is necessary.
23. For the thin strip of faces running from the space between the windshield and the first door's window frame, and up along the side of the roof, let's start by duplicating one of the edges of the front fender and move it up a bit and rotate it to get it aligned with the surface the strip must describe.
24. Continue by performing three extrusions in the right view, placing the last one according to the highest part of the windshield. Take some time to tweak these extrusions carefully, rotating the view around and checking for the shape to be well constructed.

25. Then extrude the upper edge just a bit, to get the extruded edge next to the side of the roof patch and finish the strip by performing three successive extrusions, going in the positive direction of the Y axis (alongside the roof). The first extrusion must get aligned with the edge loop running across the roof around the middle of it, the second must end in the very place where the second door ends, and the third one must end aligned with the last border of the roof patch. Of course, we must take some time to carefully tweak the new geometry, to create a good shape with it. Let's look at a screenshot to get a clear idea of what the final shape of these patches should be:



How about taking a couple of minutes to look around the shape and the geometry that we created? At this point, the shape of the car is evident with just a few patches of geometry, which is very useful because we can see how well our modeling task is going after creating just a few faces organized into simple patches. The idea of the patches is very useful because it allows concentrating on the form at the beginning, and leaving the difficult topological problems for later (the easy ones are solved immediately, such as the poles we worked on in the front fender).

The next patch that we are going to create is the one defining the front bumper of the car and is easy to work on in the right view. The process is very straightforward, even though it (of course) requires some hand tweaking.

26. Start by locating the three edges of the front fender that run between the wheel and the headlight, and are on the border of that patch. Select the edges, duplicate them, and move them down just a bit to get a clear separation between the old patch and the new one.
27. We now need to select the two shorter edges and perform three successive extrusions in order to shape the remaining part of the circular shape of the bumper surrounding the wheel. In this case, we should not distribute the extrusions evenly, but instead get the first one to use half of the space to fill, the second one more than half of the remaining space, and the third one to fill the small remaining space. Tweak a bit to get the curvy shape correctly defined.
28. Then find the vertex that belongs to this patch and is located towards the left-hand side (we are working in the right view) and just perform three successive extrusions going straight down. Put the extruded vertices around the same height as the ones directly to the right-hand side. When you are done with these easy extrusions, just go on creating the fill faces by selecting four of them and using the *F* key.

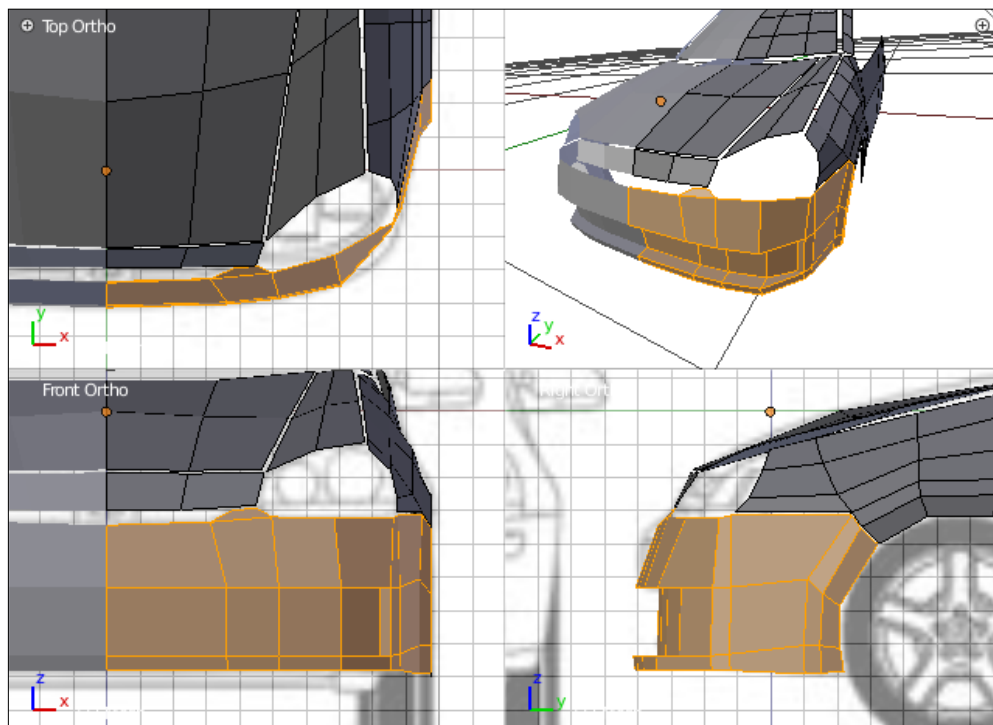


An easier way to fill faces manually is by switching to **Edge Select Mode**, selecting two edges each time, and pressing the *F* key.

It's now time to do some more...extrusions!

29. In the right view, select the three leftmost vertical edges of the front bumper patch, go to the top view, and extrude once. Move the extrusion just a bit to start generating the curvature of the corner of the front bumper.
30. Continue by extruding again from the top view and moving the extrusion in front of and below the headlight, right in the middle of the bumper.
31. It's then time to go to the front view, deselect the upper vertex from the last extrusion, and move the remaining selection in the negative direction of the Y axis (in the right view region). Using the reference blueprint to guide us in shaping the patch is very easy, so let's do it.
32. To finish the front bumper patch, we just need to select the four vertices running vertically below the headlight, perform three successive extrusions in the top view, and keep moving them to the left-hand side. The first extrusion must end touching the leftmost part of the headlight, the second one must be moved to less than half of the remaining space, and the last one can just be moved to the left-hand side until the clipping of the Mirror Modifier sticks it to the YZ plane of the 3D world.

33. A couple of finishing touches can be made by selecting the four faces starting from the symmetry plane, on the second row of faces of these last extrusions, moving them a bit in the negative direction of the Y axis (right view), and then extruding and moving the extrusion into the positive direction of the Y axis (right view). There is also a small extrusion to be done and tweaked to shape the border of the headlight better. Once again, a screenshot will help in making the procedure clearer:

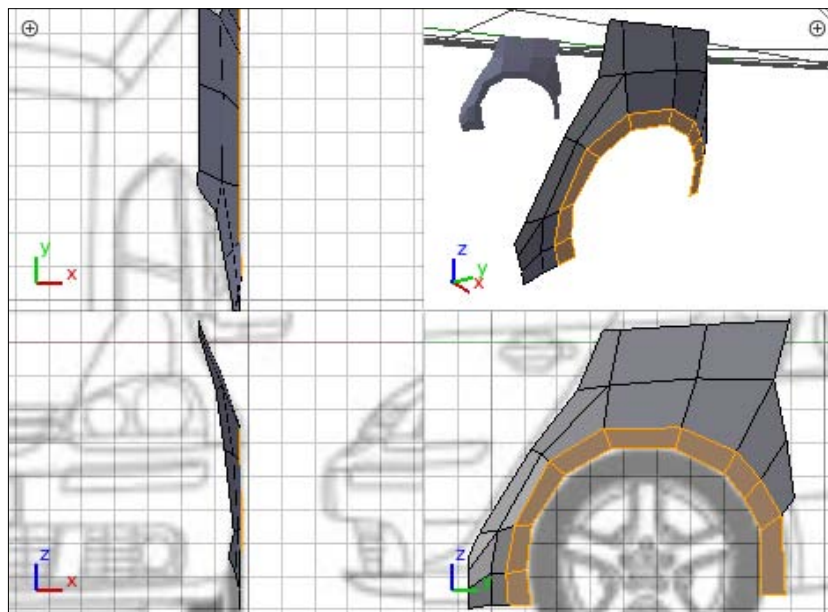


At this point, we have modeled the entire front part of the car—except for a few small elements. It's now time to start working on the back part.

Let's work on modeling the rear fender of the car; thankfully, we have already modeled the front fender and this time the process is going to be almost the same—conceptually speaking.

34. Using the front and right views, position the 3D Cursor at the center of the back wheel, add a circle from the right view and set its vertices parameter to 16, select and delete the lower six vertices, and tweak the remaining 10 to create the shape surrounding the wheel, following the blueprint.

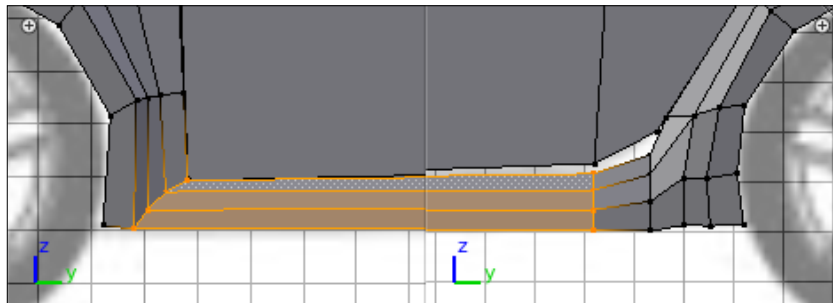
35. The next step is to select the arc that we just created, extrude it, press *Escape* to cancel the grab operation, and then scale it up a bit. Follow this by tweaking these new vertices to create an evenly spaced strip around the arc, remembering to tweak the vertices only from the right view, to have the faces forming a flat surface when viewed from the front right view.
36. It's now time to select all but the first edges of the outer ring (counter clockwise, starting from the right-hand side) and extrude them to have some more geometry. The tweaking at this point must be done taking into account that we are working on a low-poly model for now, so let's be smart and use the small number of polygons effectively.
37. There are also three extrusions to be done, two contiguous edges at a time, and the corresponding tweaking to complete it. This part also includes the creation of two poles with five edges (the reason for us to extrude only two edges at a time). To avoid having to explain these steps by using overly complicated descriptions, let's take a look at a screenshot:



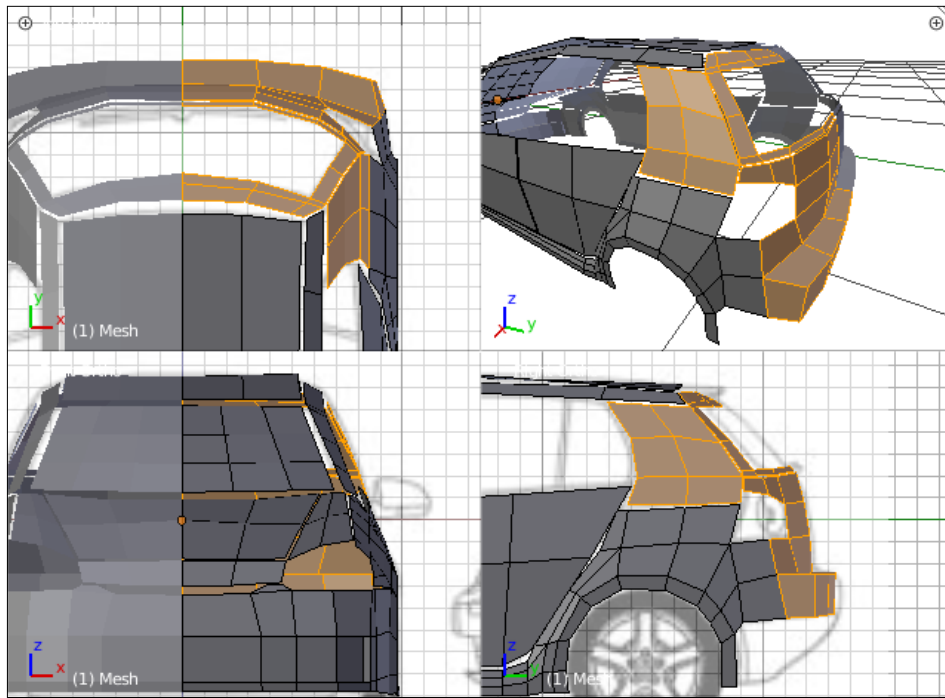
The top, front, and right views show how the shape looks when superimposed on the blueprint; the user view (top-right) shows a free view of the patch, to help visualize the shape that it has.

After we have this patch done, it's time to connect it with the patch of the front fender.

38. To do that, let's just select the three leftmost edges (right view) of the back fender patch, extrude them, press *Escape*, and then move them to align them with the vertical edge of the door. Then extrude again, press *escape*, and move the extrusion in the negative direction of the Y axis, until they get close to the lower part of the front fender, from the right-hand side to the bottom left-hand side of the front door.
39. Now merge the extruded vertices with those of the front fender patch by selecting two each time and using the Merge tool (**Mesh** → **Vertices** → **Merge**), and using the option from the menu that will make the merged vertex be located where the vertex of the front fender is. Don't forget to ensure that this "bridge" has a good shape and doesn't end up being entirely flat. Here we have a small screenshot to help us get it done correctly:

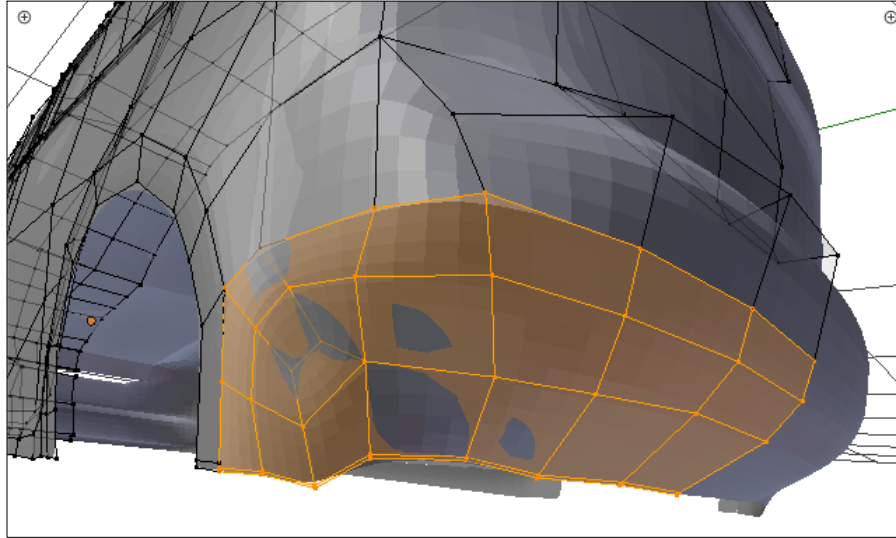


Our next step is to model the remaining part of the back fender, the frame for the rear window, and the small patch that will fill the space in between the frame of the window of the second door and the frame of the rear window. This will need us to add more geometry for the big patch that now has the front and rear fenders, and two new patches for the rear window frame and the patch besides it. Once again, we'll first look at an image to make the description easier to follow:



40. We start by extruding the two edges just below the brake light shown in the blueprint and moving them to the right (right view) until they reach the corner (using the top view as guide).
41. We then continue by extruding only the lower edge of the last extrusion straight into the positive direction of the Y axis to start creating the shape that will match the bumper below.
42. Next, we just perform three successive extrusions from the previous one and move them left (top view), to close the shape in the symmetry axis.
43. We can now finish the protrusion we are shaping by extruding the edge just below the brake light thrice to the left (top view), filling the empty space between it and the protrusion with three faces. Tweak by hand to give it a good shape.
44. We only need to perform a couple of extrusions going upwards, to continue shaping the trunk, and leave space for the brake light. Notice the two small extrusions that finish enclosing the brake light and remember to model that part.
45. The patch filling the space between the window frame of the second door and the rear window frame can be created easily by selecting two edges from the back fender, duplicating them, moving them upwards, extruding twice upwards and then tweaking as needed. Take a close look at the top part of the rear window frame and make sure to create those two rows of faces.

46. Finally, one last patch: The rear bumper. The process for this is very easy; just select all the edges of the lower border of the rear bumper patch, duplicate them, and perform three extrusions downwards; we must be careful, though, since there are a couple of three edge poles that we need to create in the vertical corner of the patch. Since this is easier to understand with an image, let's look at the resulting patch in this screenshot:



The screenshot shows what the subdivided version of the mesh looks like. Let's also add a Subdivision Surfaces modifier to the object and look around for the nice curvature that our mesh exhibits.

Objective Complete- Mini Debriefing

This part of the process was about sketching. Our purpose in doing this is just to get a good basic shape for our model, but avoid the complexity of handling dense meshes. The separation into patches is needed so that we can concentrate on one small part at a time. This lets us plan the local topology and defer the connection between the different parts for later. This will allow us to work a bit more on solving certain topology issues.

Since we are working with a blueprint, the quad view option is very handy to get a good glimpse of the shape, and how the separate parts are being correctly positioned in space.

Classified Intel

Both the front and the back fender that we just created had a particular aspect that no other patch has: Poles. At this point of the modeling task, we need to identify and create the appropriate poles for this model; in this case we only identified a few of them. The reason for thinking about topology so early is because this is the best time for it: Having just a few polygons to work with makes it very easy to work on such technicalities. It's better than waiting until the model has so much geometry that it becomes very difficult to perform such important modifications.

The nice thing about topology is that once it's done correctly, the remaining job is mostly about fine-tuning and exploring the subtle variations of the shape that will enrich the final model. If a bad topology is in place, the final, subdivided version of the model will hardly look nice. Moreover, since the surface we are modeling has certain features such as continuity (the space between doors is a simple example of that) and the subtle way that it bends, we definitely need to exploit the properties of the subdivision surfaces algorithm. Good poles and face loops are the key to get a good shape with just a few polygons, and also to ensure the proper finishing of the subdivided version of the model.

Creating the Doors

Now that we are done with the basic, general shape of the object, it's time to start working on the doors, more precisely, on creating frames and windows, and adding the necessary detail for them to look nicely beveled when subdivided. In case we have a subdivision surfaces modifier active on our object, let's temporarily disable it to do our work without worrying about it for now.

Engage Thrusters

1. Let's start working on the first door. The first step is fairly easy; Just select the upper horizontal edge of the door (user view), extrude it, and move it a bit in the negative direction of the X axis (front or top view).
2. Now let's take a look at the strip going up from the side of the windshield, and then going alongside the roof. From that strip, select the lower vertices, from the first at the left-hand side (right view) through the one that is exactly over the roof post that separates the window of the first door from the window of the second. Once selected, duplicate them and press *Escape* to avoid moving the duplicates (they'll be located at the exact same place where the original vertices are located). Make sure to keep the vertices selected and extrude them down (right view) to cover the thickness of the window frame (we must use the blueprint for guidance on this). Once the extrusion is moved down, we can just work on the individual vertices, tweaking them to form the shape nicely. Make sure to tweak them not only in the right view, but also in other views, so that it has the correct curvature needed.

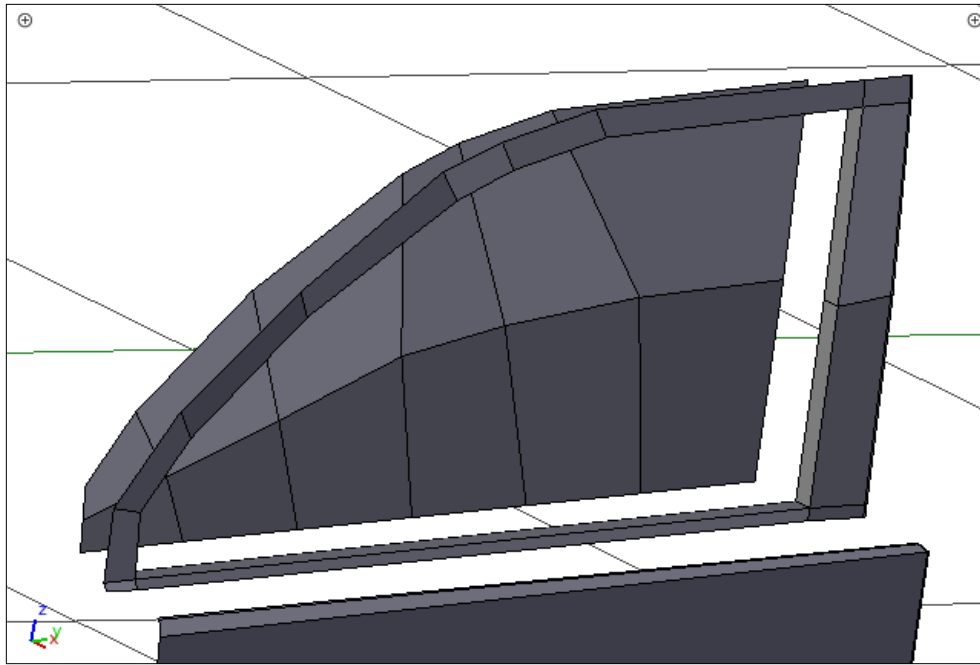
Rotating the view around helps a lot in checking how nice the shape is looking.

3. Since the rightmost face (right view) isn't subdivided to allow for an extrusion to form the roof post, let's add a trivial edge loop by using the **Loop Cut and Slide** tool (**Ctrl + R**) from the Tool Shelf (**View** → **Tool Shelf** or press the **T** key), and move it to the correct place so that we have an appropriate edge to extrude down and create the roof post for the first window; two extrusions are enough to shape the roof post. As always, we must perform the extrusions in a specific view (right view for now) and then make sure to tweak the vertices or edges by using the other views (top view is very helpful for the current task) so that the shape fits appropriately with the blueprint.
4. Then do a third and final extrusion on the roof post and make sure to move it down just slightly, enough to touch the border of the door patch.
5. To complete the frame, we just need to work on the left-hand side of it (right view), extruding down twice (the last extrusion is a small one), and then connecting the two small extrusions on both sides of the frame by creating a face between them (select the edges and press the **F** key).
6. Finally, we just need to select the inner edges of the frame, extrude them, press **Escape**, and then move the extrusion slightly in the negative direction of the X axis (top view) to create the depression in which the window glass will be.



We must be very careful about performing more extrusions than indicated, since that would leave us with garbage polygons that can be quite hard to clean up. A good way to check if our mesh has extra faces is to temporarily switch to the Face Select mode and check if the dots used to indicate the median point of a face appear in the middle of an edge. Then check if this was caused by us performing an unnecessary extrusion. Cleanup can be done by selecting *only* the problematic part of the mesh and invoking the **Remove Doubles** tool (**W** → **Remove Doubles**).

7. With the last extrusion selected, let's duplicate it and move it by 0.2 units in the positive direction of the X axis (top view), to be able to work easily on filling the faces for the window; this can be done according to the next screenshot:



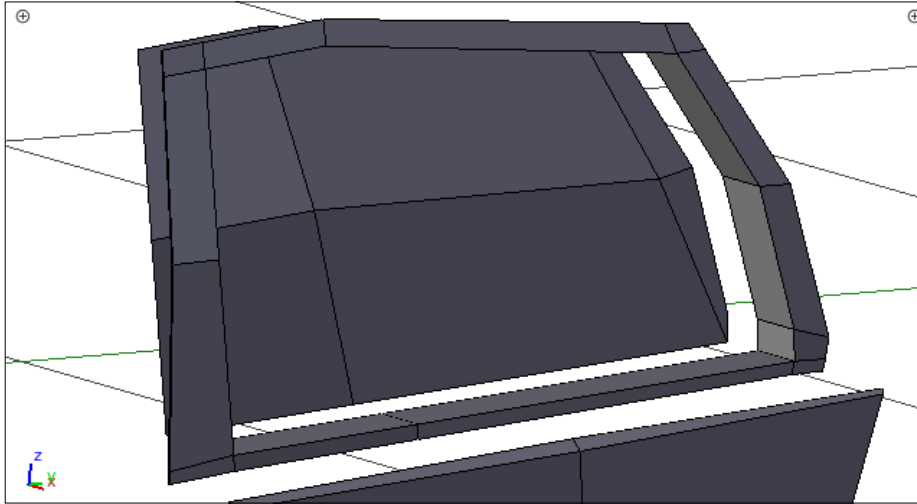
8. If the edge count for the top side of the frame doesn't correspond to the one in the screenshot, it's just a matter of using the **Loop Cut and Slide** tool (**Tool Shelf** sidebar, **View** → **Tool Shelf**) to add it to the frame, to the window, and also to the original strip that runs alongside the windshield; in that case, don't forget to tweak the shapes accordingly.



At this point, we are working with some polygons that are very close to each other. In this case, selections can be complicated, so it is recommended to perform an initial selection in the Face Select and Solid mode for the view, then switch to the Vertex Select mode and the wireframe view, and use the brush selection tool (C key) to deselect the vertices that must not be selected (clicking with the middle mouse button).

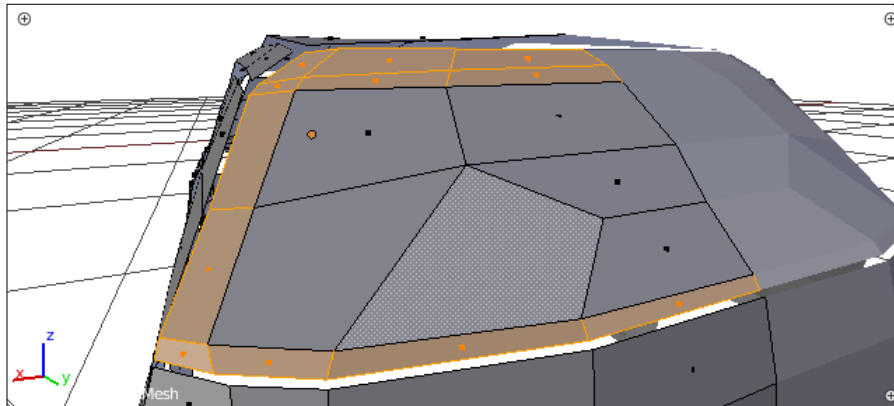
9. The next part to work with is the frame and window for the second door. The process is almost identical to the one used for the first door, except that we don't have all the geometry ready to be extruded from the patches around. We must pay special attention to the lower part of the frame, so that it touches the edges slightly extruded to the left-hand side in the actual door (the part that holds the door handle).

10. The window glass can also be easily created by duplicating the edges of the border of the last extrusion of the frame (the one that created the depression). Here's a screenshot of the finished frame and window glass, to help us with some ideas on which new geometry should be added to what, and where:



Notice that the inward extrusion of the frame is exaggerated on purpose, to show it more clearly in the screenshot. The actual model can have a shorter extrusion.

11. The last window that we are going to add is the rear window. In this case, it is pretty easy, since its frame provides us with the geometry to duplicate and start working with. Since there's a small trick applied in this window to avoid creating more edges in the upper part of its frame, let's look at a screenshot of how it must be done:



Notice the two poles (a 5-edge pole at the center and a 3-edge pole at the bottom right) that are used to redirect the topological flow and help us to keep the top part of the frame with the number of vertices unchanged.

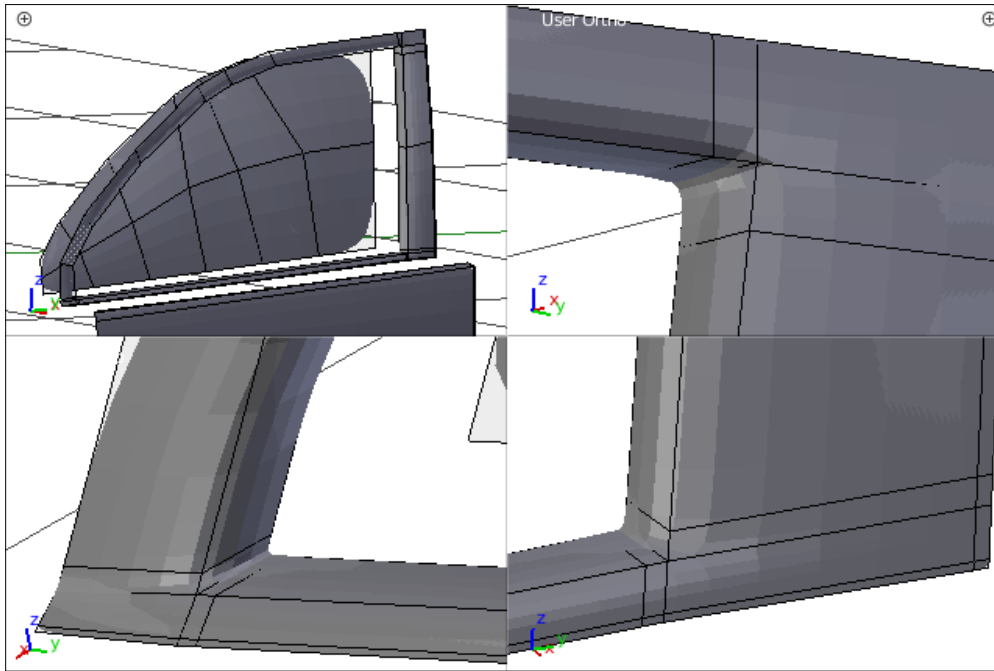
The last thing we must do is explore one method that can be used to create nice beveled corners when using subdivision surfaces.

12. Let's add a Subdivision Surfaces modifier to the object (or enable it if it is disabled) and then take a look at the shape of the doors, frames, and windows. We can see that the modifier has changed these shapes into very rounded ones, but we do not want that. What we want are sharp bevels and defined corners. The problem though is that we need the modifier to get a high resolution mesh with very soft and nice curvature on its surface. The solution is to add more geometry at key places to get what we want: A high resolution mesh with nice curvature and sharp corners with nice beveling.

For the explanation, let's work on the window frame, which has a few corners that need to be corrected now that the Subdivision Surfaces modifier is enabled.

13. Select only the patch of the window (*L* key with the pointer over the patch) and then hide the unselected parts of the mesh (**Mesh** → **Show/Hide** → **Hide Unselected** or use *Shift + H*).
14. Now locate the bottom right-hand side corner of the frame (right view), zoom into that part, invoke the **Loop Cut and Slide** tool (*Ctrl + R*), then move the mouse pointer over the vertical edges that form the roof post, and come down into the corner. When the tool shows an edge loop running horizontally across those vertical edges, click and then move the pointer down to move the new edge loop very close to the corner, but leave enough space to be able to easily select it in case is needed later.

15. Then perform the same task, but this time for the horizontal edges coming from the left-hand side into the corner. When those edge loops are added, the corner of the frame has a nice round bevel, but also looks quite sharp, which is exactly what we need. This screenshot shows the frame after these edge loops have been added close to the corners:



Let's take a closer look at the screenshot and find the other edge loops helping to define the external corners of the frame, created using the **Loop Cut and Slide** tool.

16. Let's now take some time to apply the same technique to the door and window. Make sure that the finished patches of the door and its frame have their borders touching perfectly to get this part of the model finished.

Before beginning our work on the second window, let's talk a bit about an easy way to get the borders of two patches to match exactly. Since we have been modeling using patches that have some free space between them, we now have to join their borders, which is quite complicated to do by hand, vertex by vertex. Thankfully, Blender offers us a very neat option to do some basic snapping, which is exactly what we need now.

17. In the header of the 3D Viewport, locate the magnet icon next to **Proportional Editing**; we just need to enable the magnet and select the **Vertex** option in the Snap Element drop-down (to the right-hand side of the magnet toggle); let's also make sure that the Snap Element (a new drop-down that appears) is set to **Closest**. With the snap option enabled, it's just a matter of selecting the vertex that we want to move, starting the translation (G Key) and then positioning the mouse pointer over the vertex that is located where we want to put the selected one. Don't forget to turn the snap option off when done with this part, since it can be annoying when working out small tweaks on the mesh.
18. Now that we know the easy way to get the borders of two patches touching exactly, it's time to add the edge loops that will help in controlling the shape of the corners of the frame of the second door. Let's take some time to work on it to get it done. Pay attention to the border between the two frames of the doors and make sure they also get in contact with the snap tool and that their corners are correctly defined by adding the appropriate edge loops.

Objective Complete - Mini Debriefing

This part of the project was only about the doors. We took the time to model their frames and windows carefully, since we have now started creating patches that have no empty space separating them. This demanded all our attention in order to avoid mistakes that could cause headaches later.

The most important technique discussed was the one we used to get sharp and nicely beveled corners, first for the frames and then for the other patches of the doors. This technique is quite easy to apply when the topology has been created carefully, mostly by avoiding triangles that appear in the mesh, because the **Loop Cut and Slide** tool can only determine the path for the edge loop to be added as long as it finds quads; it stops when it finds triangles. A very nice property of the **Loop Cut and Slide** tool is that it can be used for learning to "read" the face loops on the mesh, since the cut only runs along a face loop.

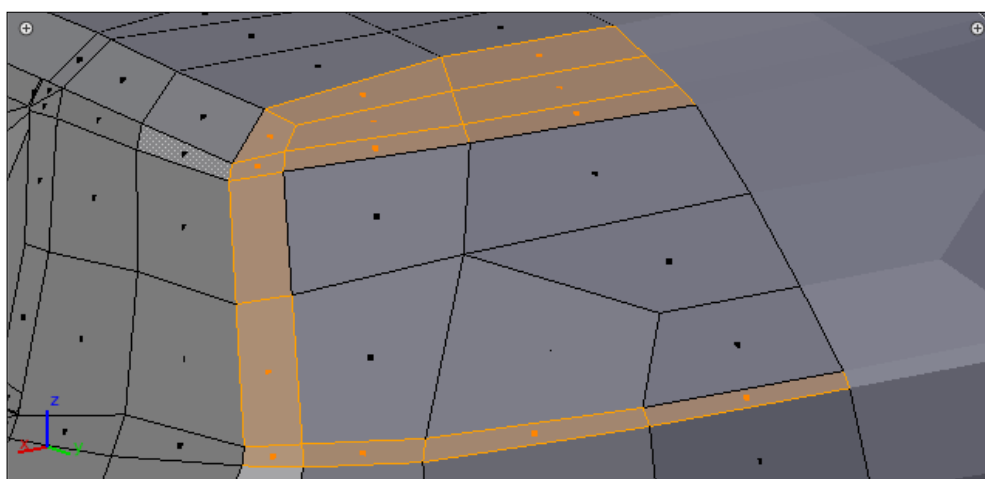
Joining the Patches

After all that hard work, we now have a low poly, "patchy" version of our model. So, the next step is to work on those patches, adding new edge loops, if needed, to get them matching correctly with their neighbors, and also performing some merges that will get two patches joined into one. For this part of the project, the quad view can be disabled, since it doesn't help much and it's better to have only one big view of the model to work on these fine tweaks.

Engage Thrusters

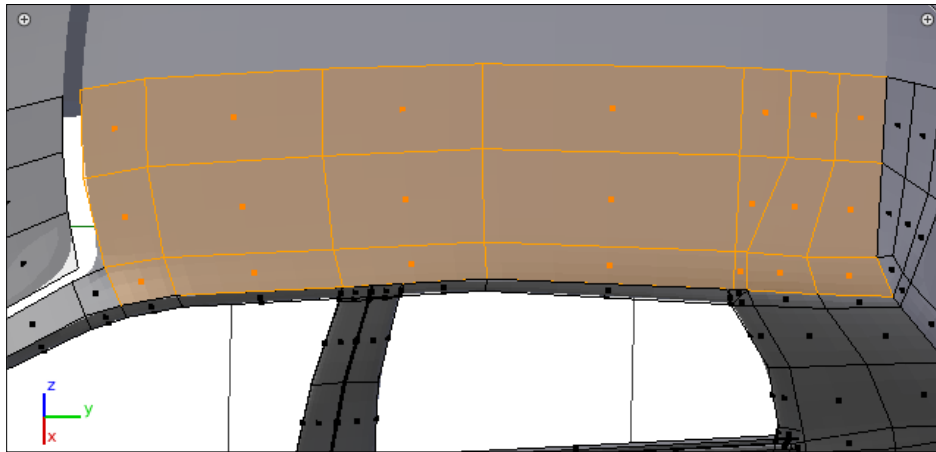
A good idea is to start with an easy spot to work on.

1. Let's go to the frame of the rear window and look at the space between it and the roof. That space must be filled with only two faces (a 3-edge pole is needed to accomplish this fill). Then it's time to enable the snap option and start snapping the vertices in the border of the patch of the frame of the rear window to touch the patches of the roof, the small patch to its side, and the big patch of the back fender below. The following screenshot helps us get a clear picture of what the patch must look like after this:

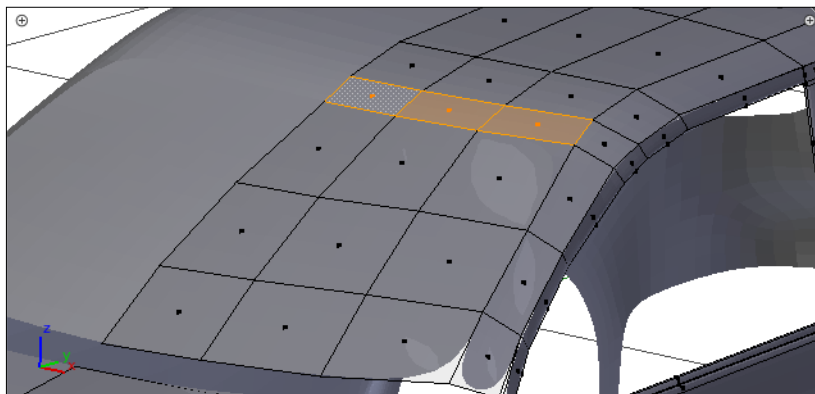


Notice carefully how the small patch to the left-hand side of the frame patch has changed. It is now touching the upper patch (the long strip coming from the side of the windshield) and the lower patch (the back fender patch). It's important to note that the changes in this patch also include merging its borders with the borders of the other patches that it touches. Since the merging can be very difficult and error-prone if done with all the patches being visible, it is recommended to just select those patches whose borders will be merged, and hide the unselected ones (**Mesh** → **Show/Hide** → **Hide Unselected** or press *Shift + H*). Once the view is restricted to only those patches that you are going to work with, the merging can be easily done by using the snap option to snap one vertex to another, then switching to the wireframe shading (*Z* Key), selecting the two vertices that are located on the same coordinates, and then using the **Merge** tool available in the **Tool Shelf** sidebar (**View** → **Tool Shelf**) to get a single vertex out of them. Once done with merging the borders of these three patches, let's get back the hidden patches (**Mesh** → **Show/Hide** → **Show Hidden** or press *Alt + H*).

2. Now let's work on merging the roof and the thin strip running alongside it. The process is quite straightforward, using the tools and methods we have been discussing. A screenshot is useful here to show the kind of tweaks that must be made on the fly, such as adding a few more edge loops to the roof patch, so that it can correctly match the geometry of the other patch, as shown in the following screenshot:

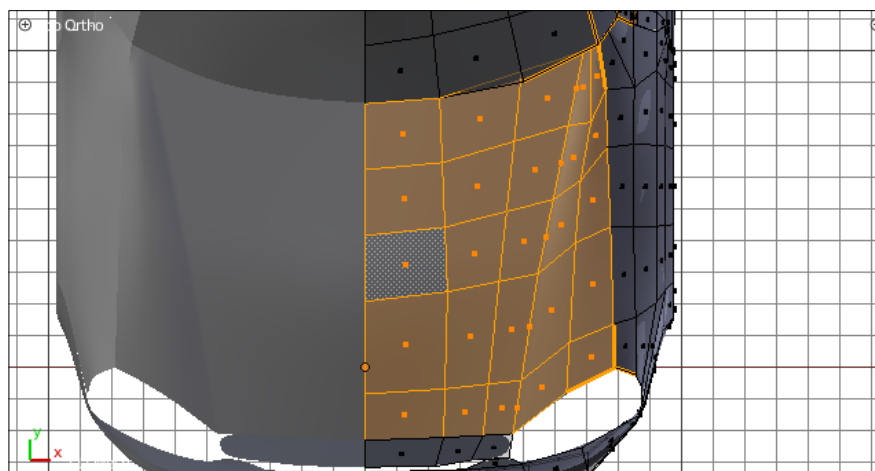


3. After merging those patches, it's time to add a new edge loop to the roof, running vertically (top view), so that the vertices of the windshield patch can be matched correctly with the vertices on the border of the roof.
4. Then it's just a matter of filling the space between the roof and the windshield with an extrusion of the roof (be careful to keep the patches separated and not to merge their borders). Then we just need to use snapping to get the vertices of the windshield put exactly where the vertices of its neighboring patches are. Once more, a screenshot will save us a long explanation:



The next part is merging the border of the hood and the small patch on its side, which is quite trivial using the methods that we have been working with until now.

5. Since the resulting patch will need to create a continuous surface with the front fender, we need to add three more edge loops running horizontally (top view), two in the upper part of the patch, and one in the lower (top view). Let's take some time to tweak the newly added edge loops. Since they will have broken the nice curvy shape for the hood, we must get it back by some manual tweaking.
6. Then it's time to snap the borders of the hood to the surrounding patches: Windshield, front fender, and the small patch in between the hood and the front bumper. The next screenshot should help you decide where to put certain parts of the geometry that could not find a suitable neighboring vertex to snap to.



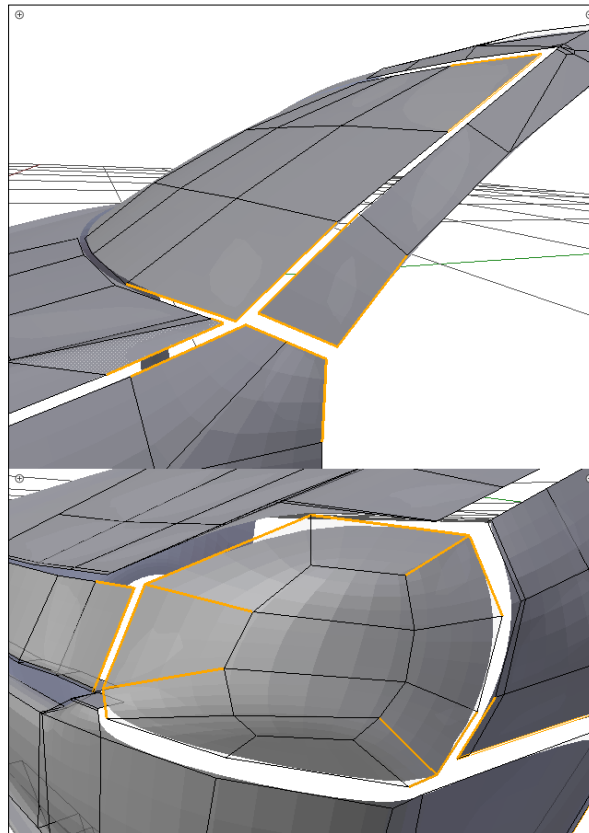
7. Having done quite a few snaps of the patch borders, it should be easy to work on the remaining ones: the front fender with the doors and the strip that surrounds the frames of the doors and the front fender with the front bumper. Let's take some time to work out those snaps, and avoid merging any vertices.
8. At this point, we can create the headlight patch by just selecting the vertices surrounding its place (select only one of those pairs of vertices that are located in the same place). Once the appropriate vertices are selected, let's hide the unselected parts of the mesh (**Mesh → Show/Hide → Hide Unselected**) and create the edges that join the whole ring into a single patch. Then select all the vertices of this curved ring, extrude them, press *Escape*, scale them by a factor of 0.5, and then fill in the hole with new faces that will only create horizontal edges inside the hole. After filling the hole, let's take some time to tweak the vertices until the shape looks nice enough. Since we are not going to work on the internal details of the car (including the internals of the headlights), let's not spend too much time on it. Finally, let's show the hidden parts of the mesh (**Mesh → Show/Hide → Show Hidden**).

9. Using the same method, let's also create the brake light patch, which can be filled quite easily by just creating two faces inside, separated by a vertical edge (front view).

To close this part of the project, let's talk about another method of creating sharp corners when the Subdivision Surface modifier is acting on a mesh.

There is an option for each edge of the mesh that causes the subdivided mesh to stick to the edge with the option activated, it's called "crease". By default, each edge has a crease value of 0, but by increasing it we get a nice behavior of the subdivided mesh, which is closely following the shape of the edge with a crease value of 1. The crease option is another way of getting sharp corners on the mesh. Let's apply it, check how it looks, and then decide which of the two methods for this particular purpose can better fit this particular model. The crease property is located in the **Properties** sidebar (**View** → **Properties**), at the bottom of the **Transform** panel.

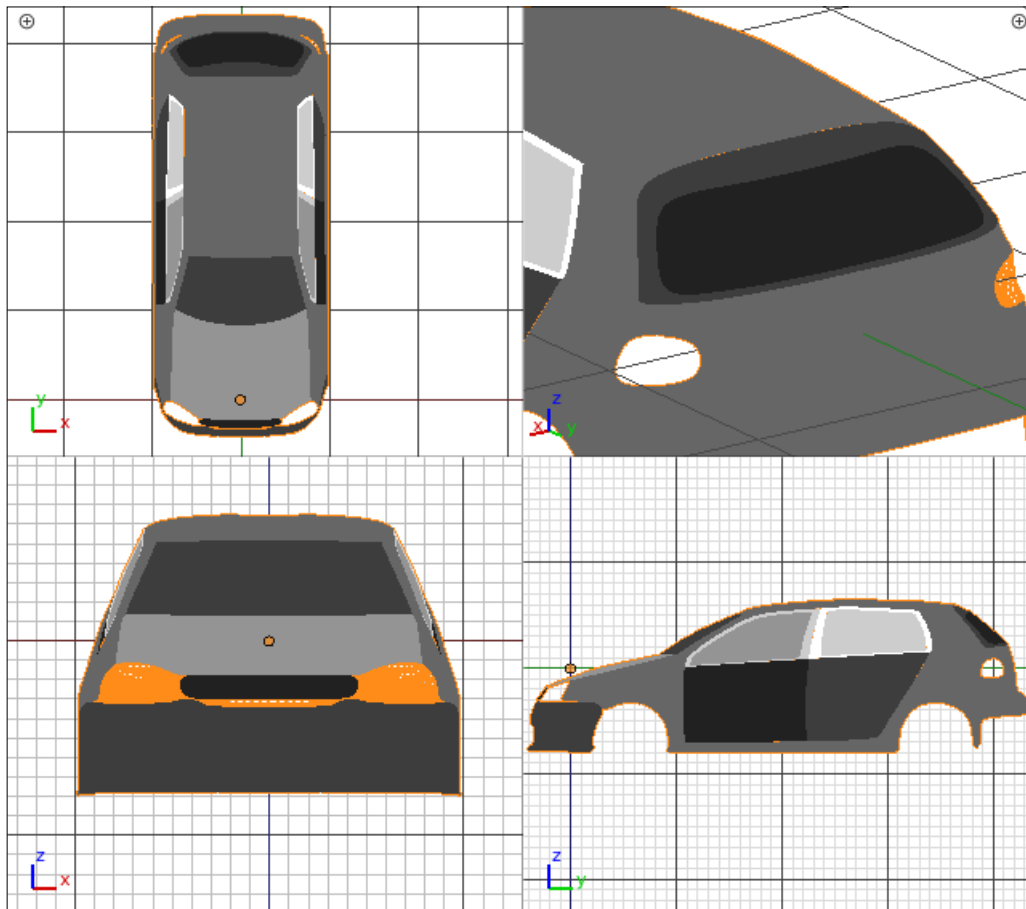
10. Given the complexity of describing the mesh in words, let's take a look at the following screenshot, highlighting the edges that must be selected in order to increase their crease value to 1:



Take a close look at the image to know which edges must be selected in order to set their crease value to **1**. To be able to see the effect in each case, it is recommended to select just a few of them, then set the **Crease** slider to **1**, and then continue selecting other edges and setting the property for them too.

11. A small but necessary task is to just select the lower edges of the front bumper and the patch with the two fenders, and then extrude them inside just enough to get the border looking nice.
12. Also select the circular borders of the two fenders surrounding the wheels and make an extrusion, moving the new geometry inside the car volume.

As a final check on the work we've just done, let's perform some tests on the mesh to know if the separated patches are the correct ones. For this, let's take a look at the following screenshot where each patch is assigned a different color than its neighbors:



The easy way to check that our mesh is separated into the correct patches is by using the **Select Linked** tool (select a vertex and then go to **Select** → **Linked**); the way to use this selection tool is by simply pointing to a vertex in the mesh and hitting the *L* key.

Objective Complete - Mini Debriefing

This task was all about assembling patches into the final shape for our car. The nice thing about this process is that it requires almost no thinking, because the complex part of the job (creating the correct topology) has been already solved.

Two very nice tools that aid a lot in performing this task are the Snap option and the Merge tool. Snap offers a very basic but extremely useful set of options to move parts of the mesh around and position them exactly where we want, very fast. The Merge tool shows how helpful it is in cases such as ours, when we need to connect separated patches of a mesh. The Merge tool is very handy, more so when used just after a snapping operation, which speeds things up a lot.

Creating the Wheels

Until now, we have been modeling the main body of our car. Now that it's ready, it's time to create the wheels, both the tires and the rims. For this, we will use a very simple approach, and get it done very fast. A small adjustment to help with this part is to open the **User Preferences** window (**File** → **User Preferences**), then locate the Editing tab, and change the **Align** option to "**View**" (left column); after changing this setting, the **User Preferences** window can be closed.

Engage Thrusters

1. Let's start by enabling the quad view mode for the **3D Viewport** (**View** → **Toggle Quad View** or press *Ctrl + Alt + Q*), then locate the **3D Cursor** in the center of the front wheel of the blueprint (right view), and in the border of the wheel (front view).
2. With the **3D Cursor** in place, let's add a circle in the right view (press *Shift + A*, then select **Circle**), set its **Vertices** parameter to **10** (press *F6* immediately after creating the circle, to change its settings), and also reduce its radius parameter until the circle correctly shapes the border of the tire. Now go to the front view and make sure that this circle is located in the place where the tire of the blueprint starts curving upside.
3. Continue by extruding the circle to the left-hand side and move it enough to reach the other "corner" of the tire.
4. Then select the circle to the right-hand side (front view), extrude it to the left-hand side just slightly (locking the movement to the X axis), and then scale it down to shape the curve of the tire.

5. Now, let's perform a few more extrusions and scales in order to shape the exterior part of the tire according to the blueprint. The inner part of the tire can be modeled by following the same steps as explained for the exterior.

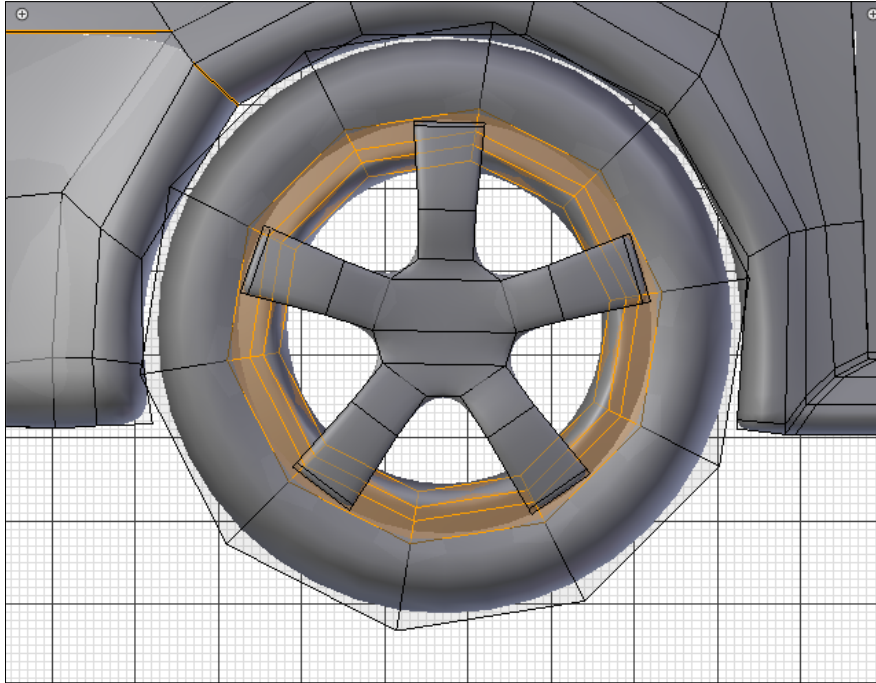
With the tire modeled, we now need to add the rim.

6. For that we just need to select the inner circle of the exterior part of the tire, duplicate it, and press *Escape* to leave it positioned exactly where the border of the tire is. Then perform three extrusions and scales to shape the outer ring of the rim; make sure to finish this "ring" with an extrusion that goes inside the wheel.
7. Now is a good time to check the shape of the tire and the part of the rim that we created and perform any necessary tweaks.

To finish the process, we just need to create the cover of the rim, which has a star shape.

8. For this, we select one of the circles belonging to the ring of the rim, duplicate it, scale it down to be just a bit smaller than the ring of the rim that we already created, and move it in the positive direction of the X axis (in front view is easier).
9. Next, we select all the edges in this duplicated circle, extrude them, press *Escape* and then scale them down by a factor of 0.3.
10. After that, we select every other edge in the outer circle of the patch that we are working in, and then delete them (**Mesh** → **Delete** → **Edges**).
11. The inner part has to be filled. We start filling by selecting two opposite edges on the inner circle and creating a face between them. Then continue creating faces parallel to this first one until the inner part gets completely filled with faces.
If we take a look at the blueprint, we will see that the shape of each arm of this "star" is more even than the one we have now.
12. To get the arms to look more even, let's go selecting and scaling by a factor of 0.5 for each one of the five edges that remain from the outer circle.
13. Once that is done, we continue by adding a trivial edge loop (with the **Loop Cut and Slide** tool) cutting each arm across into two sections; once the five arms have this new edge inside, let's just select all these new edges and move them slightly in the positive direction of the X axis. Now we can see the nice curved shape that appears.
14. To finish the cover of the rim, we only need to sharpen the tip of each arm by adding an edge loop parallel, and very close, to the rim.
15. With the shape completed, we just need to select all the edges that form the entire border of this patch of polygons and then extrude them inwards, just enough to give a good shape to the arms.

16. Let's also take some time to tweak the position of the rim's cover, so that it is located properly over the ring of the rim.
17. To finish this part, let's just select the three patches forming the wheel, duplicate them, and move them in the positive direction of the Y axis (right view) to put it just below the back fender. To wrap it up, here's a screenshot showing the final result of the front wheel:



Notice that the topology for the inner disc of the rim isn't exactly the best one (the shape obtained with the Subdivision Surface modifier doesn't look very nice), but we are not going to spend much time on it, since it's a minor detail.

Objective Complete - Mini Debriefing

The procedure that we followed is quite straightforward. We began everything with a circle having the appropriate number of vertices, and then worked the three patches by extrusions, scales, and moves. The ease with which we constructed this part of the car model is because of the simple radial symmetry that it exhibits.

Shading

After all the detailed work on the mesh, we now have the model ready. Now it's time to assign it some nice shaders. We will be working in a few different materials, so this part will become a bit longer, though the results are worth it. One of the materials will be created using the nodal system for shaders that Blender has, which will let us learn some basic ideas of working with this method of creating materials, which allows you to create very complex setups.

Engage Thrusters

1. Let's switch to Object Mode and add five empty material slots for our car. Select the first slot, add a new material and name it `tire`. This material will be used for the tires and for the frames of the doors; the settings for the material are as follows:

- ▶ Diffuse color: 0.029 Red, 0.029 Green, 0.029 Blue
- ▶ Specular panel:
 - Intensity: 0.036
 - Hardness: 8
- ▶ Ambient: 0.0 (Shading panel)

This material is quite simple, but it provides a believable finish for the tires. The key for this material is in setting the specularity with low values. It gives a soft specularity that is quite appropriate for this kind of a 'plastic' material.

2. Next, select the second material slot, add a new material, and name it `glass`. This material will be used for the glass of the windshield, the windows, and the rear window. The settings are as follows:

- ▶ Diffuse color: 0.012 Red, 0.012 Green, 0.012 Blue
- ▶ Specular panel:
 - Intensity: 0.8
 - Hardness: 20
- ▶ Mirror panel: Enabled
 - Reflectivity: 0.1
 - Fresnel: 2.0
 - Gloss amount: 0.93

We will use a dark color for the windows, a shade that's not transparent, since we haven't modeled the car's interior. We use a strong specularity to enforce the reflective finish; a small touch of imperfection is added to the material by lowering the gloss amount setting from 1, which causes the reflection to become slightly blurred. Adding subtle imperfections to computer generated images is important, since real-world objects are subject to many kinds of imperfections, be it their surface, shape, and so on.

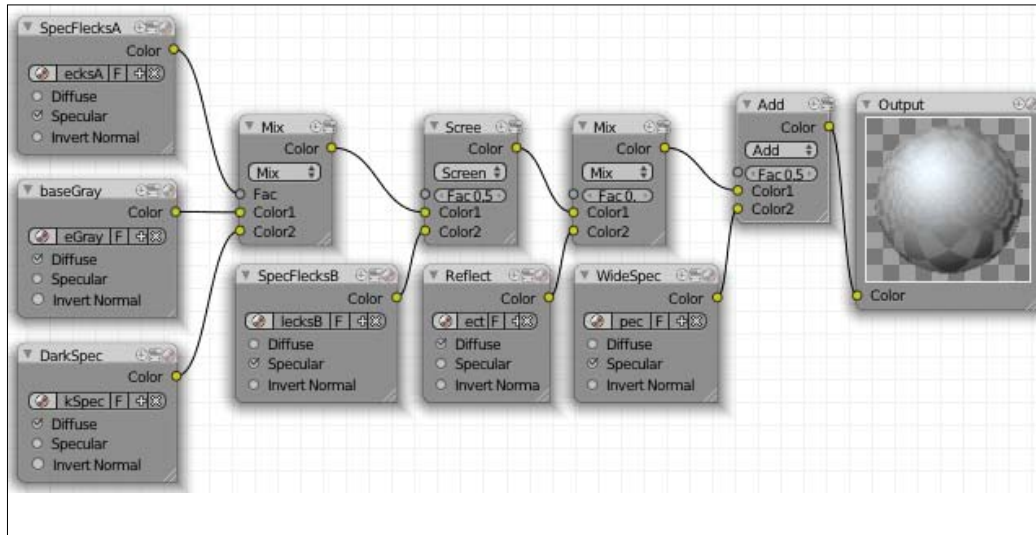
3. The third material slot must hold a new material named `rimMetal`; this material will be used only for the rims of the wheels. Here are the settings for it:
 - ▶ Diffuse color: 0.84 Red, 0.84 Green, 0.84 Blue
 - ▶ Specular panel:
 - Intensity: 0.8
 - Hardness: 10
 - ▶ Mirror panel: Enabled
 - Reflectivity: 0.3
 - Fresnel: 1.4
 - Gloss amount: 0.9

This material has a common setup with a strong specularity and low hardness to let the specular reflection (not to be confused with the **Mirror Reflection**) be wide and also adds a moderately strong reflectivity. To give some visual richness to the material, we set the gloss amount to a low value, which will blur the reflection on the surface.

4. The fourth material slot must hold a new material, and we must set its name as `lights`. It will be used for creating simple headlights and brake lights for our car. The settings for this material are as follows:
 - ▶ Diffuse color: 0.84 Red, 0.90 Green, 1.0 Blue
 - ▶ Specular panel:
 - Intensity: 1.0
 - Hardness: 20
 - ▶ Mirror panel: enabled
 - Reflectivity: 0.5

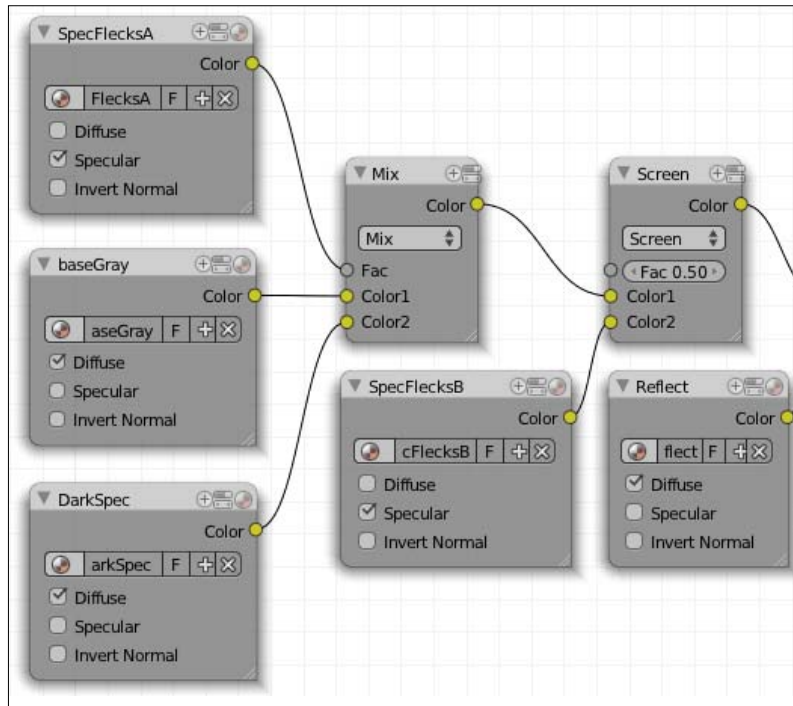
This material is quite basic; it has a strong specular reflection and also a strong reflectivity (mirror reflection); the diffuse color is set to give it a soft bluish tint. A refined shader for the headlights will require much more work than just setting these basic parameters, but that task is not within the scope of this project.

- Now for the final material slot, let's add a new material and name it `carPaint`. This material just requires us to press the button located to the right-hand side of the **X** used to unlink this material from the slot (the button to click is the one with the two small windows connected by a curve).
- After clicking on that button, let's switch to the **Compositing** screen and select the "Material nodes" option from the header of the **Node Editor**. The **Node Editor** must now have a simple setup consisting of a material node and an output node. To have our path clear before creating the actual nodes setup, let's take a look at what the entire setup looks like:



For this node setup, we will be creating six individual material nodes and setting their properties on the **Properties Editor**, just as for any other normal material. However, their names can't be changed in the **Properties Editor**. You can do that by using the **Properties** sidebar in the **Node Editor** (*N* key). Take into account that the material nodes have a name for both the node and the material itself, so make sure to name them the same to avoid any mistake.

The following screenshot shows details for the left part of the node setup, and will guide us along the way:



7. Let's start by disconnecting the original, empty material node from the output and moving the latter to the right-hand side, to have some space for the nodes that we'll be creating.
8. Click on the **Add New** button of the empty material node, and then set both the name of the material and the node to **SpecFlecksA** (**Properties** sidebar, *N* key). Make sure to leave only the **Specular** checkbox enabled in the node before you select it. Now that the node is selected, its settings can be adjusted in the **Properties Editor** as follows:
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue
 - ▶ Specular panel:
 - Intensity: 1.0
 - Hardness: 3

9. Next, go to the **Texture** tab in the **Properties Editor**, add a texture for the **SpecFlecksA** material, name it `flecksA`, and set it as follows:

- ▶ Type: Noise
- ▶ Mapping panel:
 - ❑ Size: 10 X, 10 Y, 10 Z
- ▶ Influence panel:
 - ❑ Diffuse Color: disabled
 - ❑ Specular Intensity: enabled, 1.0
 - ❑ Blend: Multiply

This material is just used to add small flecks on the surface, which is done by setting a very strong specularity and then modulating it with a texture. The Multiply blend mode causes the flecks to only appear where the noise texture is white. Make sure to put this node as the top left one in the setup.

10. Now add a new material node (**Add** → **Input** → **Material**), name it as `baseGray`, leave only the **Diffuse** option enabled, and make sure to have it selected. Go to the **Material** tab in the **Properties Editor** and set it as follows:

- ▶ Diffuse Color: 0.68 Red, 0.69 Green, 0.70 Blue

This is the second node on the left-hand side of the setup.

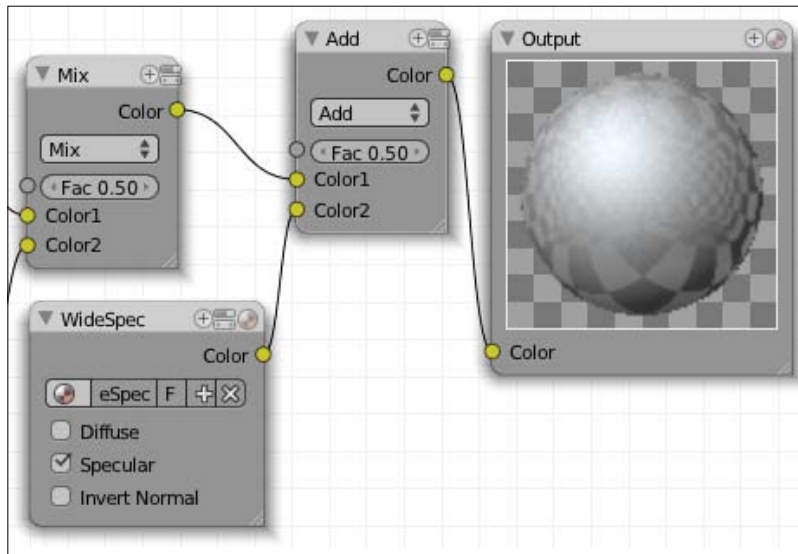
11. Now add the third node, put it below the first two, name it as `DarkSpec`, and leave it enabled only in the **Diffuse** channel. Then set its specular intensity to 1.0 and its hardness to 3 in the **Material** tab of the **Properties Editor**.
12. The next step is to add a **Mix** node (**Add** → **Color** → **Mix**), put it on the right-hand side of the column of three nodes that we just created, and connect the color output of the **speckFlecksA** node into the **Fac** input, the one for the `baseGray` node into the **Color1** input and the one for the `DarkSpec` node into the **Color2** input.
13. This first mix node must be combined with a new node material. Let's add a new material node (**Add** → **Input** → **Material**), create a new material, and name it `SpecFlecksB`. Make sure to only leave the **Specular** channel enabled on this node. Select it, go to the **Material** tab in the **Properties Editor** and set it as follows:

- ▶ Diffuse Color: 0.0 Red, 0.0 Green, 0.0 Blue
- ▶ Specular panel:
 - ❑ Intensity: 0.7
 - ❑ Hardness: 3

14. Since this material also needs a texture, let's go to the **Texture** tab, add a new texture, name it `flecksB`, and set it as follows:
 - ▶ Type: Noise
 - ▶ Mapping size: 1.0 X, 1.0 Y, 1.0 Z
15. Now that we have finished setting up `specFlecksB`, it's time to add another mix node to the setup, set its mixing mode to **Screen**, and connect the first **Mix** node and the latest material node into the Color1 and Color2 inputs, in that order.
16. Add a fifth material node to the setup, create a new material, and name it as **Reflect**. Leave enabled only the option for **Diffuse** on this node and select it to set its properties as follows:
 - ▶ Diffuse panel:
 - ❑ Color: 0.94 Red, 0.92 Green, 0.89 Blue
 - ❑ Intensity: 0.6
 - ▶ Specular panel:
 - ❑ Shader type: Phong
 - ❑ Intensity: 0.5
 - ❑ Hardness: 8
 - ▶ Mirror panel: enabled
 - ❑ Reflectivity: 1.0
 - ❑ Fresnel: 2
 - ❑ Gloss amount: 0.95

This material has a subtle yellowish tint and a low diffuse intensity in order to look darker. The specular settings and mirror reflectivity give it a plain reflective look, as seen in the preview of the material in the material node.

At this point, we need to see the screenshot for the remaining part of the setup:



17. With the **Reflect** material node in place, we need to add a new **Mix** node and connect the **Screen** node to the **Color1** input, and the **Reflect** node into the **Color2** input.
18. Finally, we just need to create one more material node. Create a new material and name it **WideSpec**. This node must have enabled only the Specular channel and its settings are as follows:
 - ▶ Specular panel:
 - Color: 0.65 Red, 0.77 Green, 0.87 Blue
 - Shader type: Blinn
 - Intensity: 1.0
 - Hardness: 8
 - IOR: 10.0

This material uses the specular properties to create a nice, soft, blueish specular highlight.

19. To finish the **carPaint** material, we just need to add a new **Mix** node, set its mode to **Add** and connect the previous mix node into the **Color1** input and the **WideSpec** material node into the **Color2** input. From the **Add** node, we just need to connect the output directly into the **Output** node, and we are done with this setup.

20. With the materials finished, it's just a matter of switching to **Edit Mode** for the object, and selecting the patches that each material will be assigned to. The assignments are as follows:

- ▶ `tire` material: The tires, roof posts, and the rear bumper
- ▶ `glass` material: The windshield, doors' windows, and the rear window
- ▶ `rimMetal` material: The rims for the wheels
- ▶ `lights` material: The headlights and brake lights
- ▶ `carPaint` material: The hood, the front fender (all its patch), and the front bumper

Objective Complete - Mini Debriefing

The shading work didn't give us much trouble; the only complex material is `carPaint`, because it uses six basic materials in a node setup; though the individual materials in the nodes aren't complex at all. It's notable how the final look of the `carPaint` material is constructed by mixing the simpler materials in the node setup. We can now save the file and take some time to play with a few settings of the `carPaint` material to see how the end result is built, step by step, by layering the material nodes in the setup.

Also notice how helpful it was to leave the patches for the mesh separated, since that meant we could easily select them with the **Select Linked** tool when assigning the material to each patch.

Simple Render Setup

To finish our project, we need at the very least, a basic scene to appreciate how our nice car looks when rendered. However, before we do anything, remember that the shaders for the car depend heavily on ray-traced reflections, so our basic scene is not going to get the most out of them, though it will let us have a sense of how the finished car looks.

Engage Thrusters

1. Our render setup will be very basic, we just need to add a circle from top view, and scale it so that it is bigger than the car. Then perform an extrusion, press *Escape*, and scale it down a bit.
2. Follow this with an extrusion and also move the extruded edges a bit downwards. Then scale them down just slightly and finish by extruding and scaling the extruded edges by a factor of 0.05. Finally, select the first circle of this mesh and extrude it down a unit. Now move this simplistic platform and position it just below the car. This platform will be left with the default material.

3. Next, we need to set up the environment to put a bit of richness in the lighting. In the **World** tab of the **Properties Editor**, enable **Ambient Occlusion**, set its factor to 0.4, and its mode to **Add**.
4. Now go to the **Texture** tab in the **Properties Editor** and make sure the text at the top shows the name "World", which means we can create and link textures to the world of our scene. Add a new texture, name it `skymap`, set its type to **Image of Movie**, and load the `FullSkiesDusk0001` image that we downloaded from `cgtextures.com`. Then make sure to set the coordinates (**Mapping** panel) to Sphere and enable the **Blend**, **Horizon**, **Zenith Up**, and **Zenith Down** options in the **Influence** panel.
5. Finally, let's add a point light and leave it with the default settings. The last thing to do is position the camera appropriately to take a good shot of the car. Let's position the camera using the camera **Fly** mode (*Shift + F*, in camera view). In case we forgot to set the smooth shading on the car, let's just reveal the **Tool Shelf** and click on the appropriate button there.

Objective Complete - Mini Debriefing

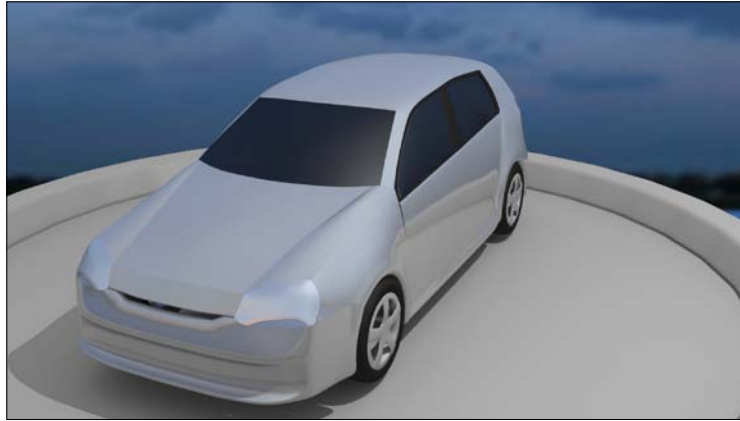
The render setup is very basic, and its only purpose is to have a first glimpse of how our car looks in a render. We just created a very simple platform to put the car in, added a `skymap` as the background to get a bit of environmental lighting that will help in giving the shaders a better look, and added a simple light to the scene.

Mission Accomplished

In this project, we have been working mostly on the modeling of our car, but also put some effort into creating a nice shader for the car paint.

To keep the tutorial of a reasonable length, we didn't work out the very small details of the model, but the good news is that the topology we put in place is good enough to keep working on the model and get it looking very nice. Of course, that will require quite some time of fine tweaking and applying some techniques to get the contact between patches to look as smooth as possible, but with the techniques we learned we have the right toolbox to go for that.

Now, after all that work, let's take a look at the finished render:



You Ready to go Gung HO? A Hotshot Challenge

Our car will be greatly enhanced by working on some finer details. Here are a few options to improve our work so far:

- ▶ Modeling and shading the rear view mirror glasses
- ▶ Adding finer details to the rims
- ▶ Modeling a detailed logo and grill, and shading them
- ▶ Adding a license plate with your name
- ▶ Adding thickness to the border of all the patches; that way it will be visible in the render and the look of the car will be greatly improved
- ▶ Modeling and shading the interior parts of the car, which will allow us to put some transparency to the glass shader and have a more detailed model to render

Project 4

Create a Professional Looking Demo Reel

A very important part of producing CG artwork is having a good demo reel ready for any opportunities that may arise. We can find lots of different types of reels on the Internet, some that show general skills and some that focus on a particular one. In the end, it really depends on both the artist's professional interests and the kind of job he or she is looking for. The latter can bear quite heavily on the decision, since a demo reel is all about showing a profile that matches what a studio could need from the artist.

Our reel will showcase our skills as modelers. Our first priority should be to find a way to get the structure of the mesh rendered over the object so that the viewer can look at both a nice render and a well constructed mesh. To get some homogeneity in the final video, we'll be using a template scene from which to create the specific scenes for each object to be showcased.

Mission Briefing

We'll be producing a short video showcasing the three objects that we created in the previous chapters. It'll have a simple intro and closing image showing the artist's basic info and three clips showing the objects rotating so that the viewer can see them from all angles.

When producing a reel, it is important to keep it short, which means just showing the best work we have done and highlighting the aspects of it in which we are more skilled. Since our reel will have the purpose of showing our modeling skills, the scenes won't have a descriptive text, just renders showing the structure of the meshes since it is the most important aspect to show for a modeler.

Though simple, we will give some small touches to it so that it looks pleasant and modern. This is important to show that, although the reel has a specific purpose, we have enough understanding of CG in general to produce the complete objects from start to finish.

What Does It Do?

In this project, we are going to create a nice video that will serve as a first step towards creating a nice demo reel. Since we have already worked on some objects that mostly required modeling work, we'll be creating a reel showcasing modeling more than anything else.

We'll start by creating a simple render showing some basic info about us, adding a little bit of sophistication by using a custom font. This render will serve as the intro and closing image for the video. The reason to keep it simple is so that the person looking at it gets the information easily and immediately, and also because what we want to showcase is our model, and not our animation skills.

For the objects that represent our modeling skills, we'll be creating a basic studio setup to give them a good-looking environment and to add a dash of quality to the demo reel. The studio setup will be composed of world settings to get a nice soft gradient in the background of the scene and a plane with special settings that will cause it to only be rendered in the pixels where it is shadowed by an object above. Together, these two elements will produce a very clean looking environment for the models. This environment will also require some tweaking to get the objects rendered with lighting that looks good.

For the creation of the scenes for each of the objects, we'll be using Blender's append functionality, which will save us lots of time.

After rendering the individual image sequences for each object, we'll work with Blender's Video Sequence Editor, which will prove how useful it can be to perform basic video editing tasks and also how helpful it is to have all these kinds of different editors available in the same package.

Why Is It Awesome?

This project offers a few nice aspects to learn from. First, we'll be reusing objects produced previously by using Blender's Append and Link functionality; second, we'll be working on creating a studio setup using a very simple, yet effective method; third, we'll be using Blender's sequence editor, which offers a small but very useful suite to edit and produce videos. All of that together can give us a very good idea of how cool is it for Blender to offer such a variety of functionality packed together; the best of it all is how well integrated all the editors are, which certainly helps in reducing the usual time involved in moving the data from the 3D package to the video compositing package, and back, and forth, and so on.

Now, if we start considering the many different editors that Blender has, we definitely start seeing how useful and powerful Blender can be when used to its full capacity. If we think about it, we'll notice that we can start from a single vertex in Blender and end up with a composited video, including post-processing, all done without any extra package. Isn't that cool?

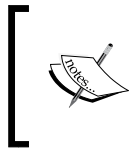
Your Hotshot Objectives

This project will be split into just four tasks; since we won't be producing complex objects now, the complexity of it will lie mostly in organizing the scenes and assembling the final product. Here is the outline of the tasks:

1. Creating the intro and closing image.
2. Creating the studio setup.
3. Creating the individual scenes.
4. Composing the final video.

Mission Checklist

For this project we will need a font file. A good place to find fonts is www.1001freefonts.com; let's browse to that URL and download the font named "SF Electrotome" (located in the **Modern** section). After the package is downloaded, extract its contents into a folder named `font-package` inside the project's main folder.



A very important issue to keep in mind when using resources such as fonts or images, is licensing. This specific font package has an accompanying file stating that it's "freeware", so it should be useable in a reel (which is a commercial work).

Creating the Intro and Closing Image

This first part will be about creating a very simple, yet interesting image to be used as the intro and the closing for our demo reel. The reason for keeping it simple is because what we want to highlight is the actual 3D work to be showcased in our reel and to make it easy for the viewer of the reel to find out who is the author of the artwork showcased—and, hopefully, get in contact.

Prepare for Lift Off

We start by opening up Blender and saving the default file with the name `demoreel.blend`, in a folder created specifically to hold this project's files.

Engage Thrusters

1. Create a new empty scene and name it `intro`. Make sure the 3D Cursor is positioned at the origin of the **3D World** (**View** → **Align View** → **Center Cursor and View All** or press *Shift + C*) and add a new camera to the scene.
2. With the camera selected, clear its rotation (**Object** → **Clear** → **Rotation** or press *Alt + R*) and then move it **14** units in the positive direction of the Z axis.
3. Now add a text object (**Add** → **Text**), then go into the **Edit Mode** for the text object and name it whatever you want.
4. Next, let's switch to **Object Mode**, add another text object, and move it 0.9 units in the negative direction of the Y axis. Edit it (**Edit Mode**) to read `Modeling reel`, then press *Enter* (while in **Edit Mode**) to go to the next line and write your e-mail address.
5. Now switch to **Object Mode**, go to the **Object Data** tab in the **Properties Editor** window (**F icon**), locate the font panel, and set the size parameter to 0.6.

At this point, we have the text elements ready, but the default font doesn't look good enough for our demo reel, so let's use the font that we downloaded previously.

6. Select either of the two text objects in the scene, go to the **Object Data** tab in the **Properties Editor**, locate the **Font** panel, and click on the folder icon to the right-hand side of the **Bfont** text to get the File Browser. Next we need to locate the font, so let's browse to the folder where we extracted the font package, select the `SF Electrotome Condensed.ttf` file and click on the **Open** button (upper right-hand side).

7. For the other text object we just need to select it, go to the **Object Data** tab in the **Properties Editor**, locate the **Font** panel, and select the already loaded font from the drop-down menu that appears after clicking on the icon of a document with an **F** inside.

Since our scene was created empty, there is no light object to "light" our objects.

8. Let's create a point lamp. Open the properties sidebar (**View** → **Properties**), locate the **Transform** panel, and set the location values as X: 3, Y: 2, Z: 5.
9. We can now take a few seconds to render the image and see it done; it's simple, easy to read, and we added a little bit of detail by using a nice font for it.
10. To finish this part, let's save the image. Go to **Image** → **Save As**, locate the folder of the project, and save the image as `intro.tga` (the format must be selected from the drop-down menu on the left sidebar of the **File Browser**).

Objective Complete - Mini Debriefing

We just worked out a very simple scene with a couple of text objects to be used as the intro and closing image for our final video. It is important to use very short texts in this intro to the reel, since we only need to display our name, the kind of material that the reel showcases, and how to get in contact with the artist, in case the viewer likes our artwork and decides to hire us. The reason to use the same image in both the intro and the end of the reel, is to make as easy as possible for the viewer to find that information whenever he/she wants.

Creating the Studio Setup

We are going to showcase our artwork and highlight the modeling part of it, so we can easily create a "template" scene, which will be duplicated later to show each individual piece of artwork separately, but keeping the whole look entirely uniform.

Engage Thrusters

1. Open up a new instance of Blender, rename the default scene as `template`, and save the file as `template.blend` in the project's folder.
2. Delete the default cube in the center of the scene and, after making sure that the 3D Cursor is positioned at the 3D World's origin, add a new Plane (**Add** → **Mesh** → **Plane**).
3. With the plane selected, switch to **Edit Mode**, select all the vertices, and scale it up by a factor of eight. In the **Properties Sidebar** (**View** → **Properties**), locate the **Item** panel and rename the object as `floor`.

Since we want to create a studio setup, we need to use a special trick for the floor's material.

4. With the floor object selected, let's go to the **Material** tab in the **Properties Editor** and add a new material slot. With the new material slot selected, add a new material by clicking on the **New** button with the plus icon. Name the new material as `transp_floor`. Now set it as follows:
 - ▶ Ambient: 0.0 (Shading panel)
 - ▶ Shadows Only: Enabled (Shadow panel)

What we just did was set the material of the floor to just be visible in the pixels where an object has a shadow covering it, which means that the object will have a nice shadow below, but without explicitly showing the object that is receiving the rendered shadow. Disabling the ambient value prevents the shadow from being illuminated too much, which would cause it not to be seen easily.

5. Now let's add a simple sphere object and put it outside the view of the camera. This object will only serve the purpose of holding a material that will be used later, in the scenes with the animations to be included in the final video.
6. Select the sphere, go to the **Item** panel in the **Properties Sidebar**, and name it `wire-sphere`.
7. Next, go to the **Material** tab of the **Properties Editor**, add a new material named `wire`, and set it as follows:
 - ▶ Type: Wire
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (Diffuse panel)
 - ▶ Specular panel:
 - Specular Color: 0.26 Red, 0.003 Green, 0.0 Blue
 - Ramp: Enabled
 - Hardness: 5
 - ▶ Transparency panel: Enabled
 - Method: Z Transparency
 - Z Offset: 0.02 (Options panel)

For the ramp, let's use these settings:

- ▶ First color stop:
 - Position: 0.0
 - Color: 0.0 Red, 0.0 Green, 0.0 Blue
 - Alpha: 0.0

- ▶ Second color stop:
 - ❑ Position: 0.25
 - ❑ Color: 1.0 Red, 0.05 Green, 0.0 Blue
 - ❑ Alpha: 0.5
- ▶ Third color stop:
 - ❑ Position: 1.0
 - ❑ Color: 0.65 Red, 0.20 Green, 0.04 Blue
 - ❑ Alpha: 1.0

This material is set as `wire` type only to render lines showing how the mesh is constructed. The most important part of this material is the Z offset value, since it helps to get the wire lines get drawn closer to the camera, which we will need when applying it later, since we'll be having two copies of the object in the very same location in space. Thus, the Z Offset helps getting the illusion of the wire being drawn over the object. The settings for the specular reflection are just to give the wireframe a better look than just a plain color; the reddish colors were chosen to give it a warm look.

8. Next, we need to add an empty object (**Add** → **Empty**) and name it `controller`; this object will be the parent of each individual object to be showcased, so it will control each one of them.
9. Now locate the current frame slider in the **Timeline** window (the window with the playback buttons, just below the **3D View** in the **Default** screen) and set it to **1**.
10. Make sure that the `controller` object is selected, locate the **Transform** panel in the **Properties Sidebar** (**View** → **Properties**), right-click on the button for the Z component of the rotation and select **Insert single keyframe**.
11. Next, set the current frame to **201** (**Timeline window**), go to the **Transform** panel in the **Properties Sidebar**, set the value for the **Z** component of the rotation to **360**, and insert another single keyframe for it.
12. Finish this part by setting the first and last frames for the animation as **1** and **200** (**Timeline** window, buttons to the left-hand side of the playback buttons).
13. A final tweak for the animation of the empty `controller` must be made by going to the **Graph Editor** (**Shift + F6**), locating the curve for the **Z Rotation** channel, selecting both keyframes (pressing the **L** key while hovering the pointer just over one of them helps to do it) and then going to **Key** → **Extrapolation Mode** → **Linear Extrapolation**. This gives us an even, uniform speed for the rotation of the controller.

The next element to set up is the environment of the scene.

14. Let's go to the **World** tab of the **Properties Editor**, rename the current settings to background, and set it as follows:
 - ▶ World panel:
 - ❑ Paper Sky: Enabled
 - ❑ Blend Sky: Enabled
 - ❑ Horizon Color: 0.80 Red, 0.80 Green, 0.82 Blue
 - ❑ Zenith Color: 0.20 Red, 0.20 Green, 0.22 Blue
 - ▶ Ambient Occlusion panel: Enabled
 - ❑ Factor: 0.2
 - ❑ Mode: Add
 - ▶ Gather panel:
 - ❑ Mode: Raytrace
 - ❑ Sampling: Adaptive QMC
 - ❑ Samples: 16
15. For the lighting of the scene, we will reuse the default lamp. Select it, go to the **Object Data** tab in the **Properties Editor**, and set it as follows:
 - ▶ Type: Sun (Lamp panel)
 - ▶ Shadow Type: No Shadow (Shadow panel)
16. This lamp must be renamed as `mainLamp` (**Item** panel, in the **Properties Sidebar**).
17. Since we need shadows in the scene, let's duplicate the `mainLamp` lamp and change its settings as follows:
 - ▶ Lamp panel:
 - ❑ Type: Point
 - ❑ Falloff: Inverse Linear
 - ▶ Shadow panel:
 - ❑ Type: Ray Shadow
 - ❑ Only Shadow: Enabled
 - ❑ Sampling: Adaptive QMC
 - ❑ Soft Size: 6
 - ❑ Samples: 16

The last object that we need to create is a plane facing straight at the camera; let's work on that.

18. Select the camera, switch to camera view (**View** → **Camera**) and snap the 3D Cursor to the camera (**Object** → **Snap** → **Cursor to Selected** or press *Shift + S, 3*).
19. With the 3D Cursor positioned properly, let's add a plane to the scene (**Add** → **Mesh** → **Plane**) and get it aligned with the view angle (**Align** to view option in the lower region of the **Tool Shelf** sidebar).
20. Now let's move the plane **3.3** units in the negative direction of the **Local Z** axis (**G, Z, Z, -3.3**). Next, we must scale the plane by a factor of **1.6** in the **Local X** Axis (**S, X, X, 1.6**). This plane must be named `alpha_control` (**Item** panel in the **Properties** sidebar).
21. This object must be associated only to layer 2 of the scene. To do this, select it, go to the **Object** tab in the **Properties Editor**, and check only the button for layer 2 in the **Layers** grid located in the Relations panel; alternatively, we can just press *M, 2, Enter* just after selecting the object.

The `alpha_control` plane will be used to manually set the alpha of the rendered image. To do that, we need to add a material and set it appropriately to be used later in the node compositor.

22. With the `alpha_control` plane selected, let's go to the **Material** tab of the **Properties Editor**, add a new material, and name it `alpha_fade`. Now set it as follows:
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (Diffuse panel)
 - ▶ Shadeless: Enabled (Shading panel)
23. Now let's go to the **Texture** tab, add a new texture with the name `blend_fade`, and set it as follows:
 - ▶ Type: Blend
 - ▶ Blend panel:
 - Progression: Linear
 - Orientation: Vertical
 - ▶ Brightness: 2.0 (Colors panel)
 - ▶ Influence panel:
 - Blend: Add
 - RGB to Intensity: Enabled
 - Default color: 1.0 Red, 1.0 Green, 1.0 Blue

These settings will create a linear gradient, running vertically. The high brightness value causes it to be entirely white but if we set its value down to **0**, the resulting gradient will look entirely black. Since we need this change to happen automatically when rendering, let's animate it.

24. Go to frame **190** of the animation (**Timeline**, just below the **3D View**), put the mouse pointer over the brightness control for the `blend_fade` texture, right-click, and select the **Insert Keyframe** entry from the menu.
25. Now go to frame **200** of the animation, set the brightness value to **0** and add a keyframe for it (right-click, **Insert Keyframe**). This way, we get a customized transition effect when we compose the final video.

The final part of the task is the one that is needed to actually perform the control of the alpha channel of the rendered image to get the custom transition that we just created with the Blend texture; for that, we need to create two **Render Layers** and then use the node compositor.

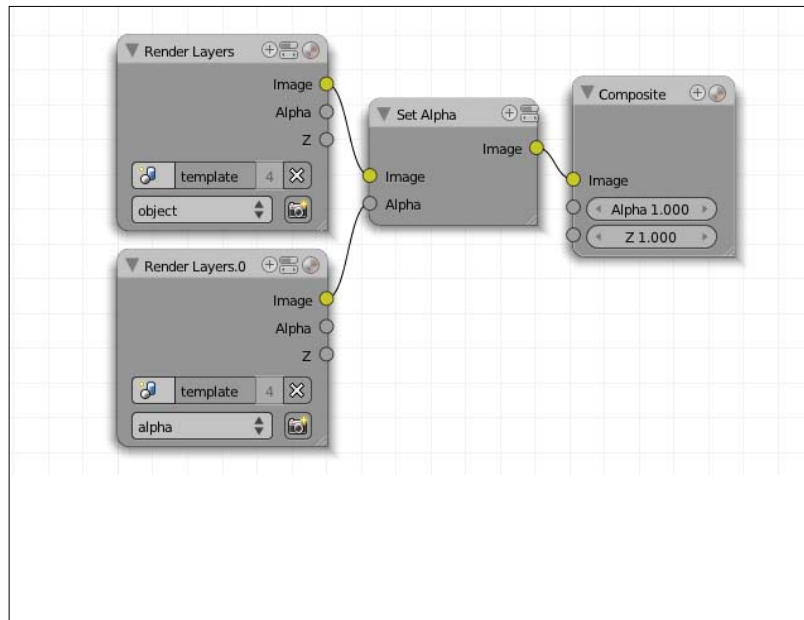
26. First, go to the **Render** tab of the **Properties Editor**, locate the **Layers** panel, select the default render layer and rename it as `object`. In the Scene buttons grid, select layers 1 and 2, and in the Layer buttons grid, select only layer 1.
27. Next, create a new **Render Layer** and name it `alpha`; the Layer buttons grid must have selected only layer 2. It's a good idea to check that all the objects of the scene are associated with layer 1, except for the `alpha_control` object, which must be associated only with layer 2.

Before going to the node compositor, let's take a couple of minutes to set up some useful defaults on this scene.

28. In the **Render** tab of the **Properties Editor**, locate the **Output** panel and set it as follows:
 - ▶ Path: `/tmp/render/seqs/` (Append `C:` at the beginning and change the forward slashes into back slashes if working in the Windows operating system.)
 - ▶ File Format: Targa
 - ▶ Color Mode: RGBA

Setting these options now will save us some typing later, since this scene will be duplicated (thus used as a template).

It's now time to go to the **Node Editor** and create the necessary node setup. Before building it, let's look at a screenshot showing it finished:



As can be seen, the setup is quite simple, consisting of only four nodes and using very basic settings.

29. Let's create it by going to the **Compositing** screen, making sure that the mode of the **Node Editor** is **Compositing Nodes**, and enabling the **Use Nodes** option.
30. Then we just add two **Render Layers** nodes, setting one to use the `object` render layer and the other to use the `alpha` render layer.
31. The mixing is done by using one **Set Alpha** node (**Add** → **Convertor** → **Set Alpha**), connecting the output from the `object` render layer into its **Image** input socket, and connecting the output from the `alpha` render layer into the **Alpha** input socket.
32. The output from the **Set Alpha** node is simply fed into the **Image** input of the **Composite** node. Now is a good time to save our file; we can also close it.

Objective Complete - Mini Debriefing

This task was just about preparing a good "template" scene from which it will be very easy to create the individual object scenes later.

In this scene, we performed the following tasks:

- ▶ We created a nice default background gradient.
- ▶ We prepared a plane to be rendered only as a shadow, to get a cool, studio setup look.

- ▶ We added two lamps to separately control the lighting and the shadows in the scene.
- ▶ We created a placeholder object with a default animation that will make it very easy to rotate each object.
- ▶ We set up a simple customized transition effect, by using an object with the shadeless mode for the material and a blend texture with its brightness animated. The actual control of the alpha channel is done by using the simple node setup added at the end.

Classified Intel

One of the nicest features of the Node Editor is the ability to separate and manipulate each channel of the image as desired. It is perfectly possible to break apart the four channels of the image (Red, Green, Blue, and Alpha), process them one by one, and then merge them into a normal image buffer. This ability opens up quite a few possibilities to play with the rendered image. In our case, we just created a render layer with a gray scale image (the actual pixels store normal RGBA data, but the colors are shades of gray) that would allow us to handle the alpha channel of the image in exactly the way we want. Add to that the many kinds of simulation and animation systems available in Blender, and you get a very powerful mechanism to create custom transitions.

Creating the Individual Scenes

With the template prepared, we just need to start replicating it and add the objects from our previous projects. Then, with a bit of hands-on work, we will get the scenes ready to be rendered into a sequence of images.

Engage Thrusters

We are going to copy the template scene from `template.blend` into new scenes in the `demoreel.blend` file. The steps needed are as follows:

1. Open `demoreel.blend` and go to **File** → **Append**. In the **File Browser** that appears, browse to the folder of our current project and locate the file `template.blend`, and click on it. Now, the **File Browser** shows a list of folders representing the types of data available in the file; click on the folder named **Scene** and then select the **template** scene from the list. To append it to the current file, click on **Link/Append from Library**.
2. Now check the list of scenes on the `demoreel.blend` file and go to the `template` scene.
3. To keep everything organized, let's call the appended scene `item1` to clearly indicate that it's the first sequence of images to be used in the final video.

Now it's time to bring in the object that will be showcased in this scene.

4. Go to **File** → **Append**, locate the folder that we created for project one and locate the file `space-scene.blend`; click on it to get the listing of internal data types available. Now click on the **Object** folder and look for the object named `starFighter`; click on it and then click on the **Link/Append from Library** button (upper right-hand side of the **File Browser**).
5. In case our appended object isn't located at the origin of the 3D World, let's select it in the **Outliner** and then clear its location, rotation, and scale (**Object** → **Clear** → **Location** or press *Alt + G*, **Object** → **Clear** → **Rotation** or press *Alt + R*, and **Object** → **Clear** → **Scale** or press *Alt + S*).
6. Now that we have the object at the origin, let's rotate it around the **Z Axis** so that it "looks" towards the negative direction of the **Y Axis**. Let's also switch to the **Front View** and move the object on the **Z Axis** to get it correctly positioned just above the floor plane.

The original object has its shaders assigned, but let's remember that the purpose of this demo reel is to showcase modeling work, which means that we need to show the mesh structure of the model.

7. To do that, let's select the object and duplicate it (**Object** → **Duplicate** or press *Shift + D* and then press *Escape* to avoid moving the duplicate).
8. Now let's change its name to `starFighter-wire` and go to the **Material** tab in the **Properties Editor**. Delete the material slots for the object, by clicking on the minus button to the right-hand side of the material slot list, until it gets emptied. Then locate the button with the red sphere icon, to the left of the **New** button, click it and then search for the `wire` material. After assigning this material, the duplicate of the object must have the wire material ready.

Since we need the object (and its duplicated `wire` version) to rotate around, following the `controller` object's animation, let's just parent them to it.

9. Make sure to have the current frame set as 1, to avoid having an offset in the rotations between the children and their parent objects; first select the original object, then its wire duplicate, and then finally, the empty scene named `controller`. Set it as the parent for the mesh objects (**Object** → **Parent** → **Set** or press *Ctrl + P*). If we play the animation right now, we can see that both the original and duplicated objects follow the rotation of the empty scene.
10. Let's also do a render of the scene, to make sure that everything is rendering correctly. If the object looks too small, let's just scale up the `controller` empty scene until it has the correct scale. A final check for the object is to ensure that it's associated with only layer 1 of the scene, to get it rendered by the proper render layer.

11. In order to organize the output from the rendering of each scene, let's go to the **Render** tab in the **Properties Editor** and change the output path (Output panel) to `/tmp/render/seqs/item1/item1-`; this will cause the rendered sequence to be written into a separate folder, which will be of help later.
12. Next, we must repeat the same process as before, appending again the template scene from the `template.blend` file. This time we are going to append the objects `fighterParent`, `fighterBody`, and `fighterStabliators` from the `fighter-storm.blend` file.
13. Since they are not a single object, we just need to clear the transformations for the `fighterParent` object by selecting it, clearing its location (*Alt* + *G*), its rotation (*Alt* + *R*), and its scale (*Alt* + *S*).
14. Now we must delete the `fighterParent` object to get rid of the animation curves that could come with it in the appending process.
15. Finally, we select both the `fighterBody` and the `fighterStabliators`, place the objects just over the shadow plane, make sure that the animation frame is set to 1, and duplicate the objects (**Object** → **Duplicate** or press *Shift* + *D* and then press *Escape*). The duplicated objects must be named with the suffix `-wire`; also remember to select all the four child objects and parent them to the `controller` object empty for this scene.
16. The scene must be named `item2`. For the output path of the rendered sequence, let's use `/tmp/render/seqs/item2/item2-` (remember to add *C:* at the beginning and change forward slashes to back slashes if working on a Windows operating system). Don't forget to check for the object and its duplicate to be associated only with layer 1 of the scene.
17. Once again, let's append the template scene from the `template.blend` file. Now name it as `item3` and then append the object datablock of the car that we created in *Project 3, Construct a Car Using Modeling and Shading*.
18. In case it's necessary, perform the usual transform clearing (**Object** → **Clear** menu).
19. Next, let's check whether the object is correctly put over the shadow plane and the animation frame is set to 1. Continue by duplicating it and rename the duplicate by adding the suffix `-wire`.
20. To get the `wire` material associated with the duplicated object, let's delete all the material slots and then select the `wire` material from the drop-down menu with the red sphere icon.
21. After that, we must select both objects, then the `controller` empty object, and parent the objects to it (**Object** → **Parent** → **Set** or press *Ctrl* + *P*). If needed, let's scale the `controller` empty object up or down in order to get the objects sized correctly.
22. Then it's time to make sure that the objects are associated only with layer 1 of the scene, since that's the layer to be used by the appropriate render layer.

23. Finally, let's set the output path setting (**Output** panel at **Render** tab in the **Properties Editor**) to `/tmp/render/seqs/item3/item3-`. Let's also save an incremental version of the file now.
24. To finish this task, we just need to render the corresponding sequences of the `item1`, `item2`, and `item3` scenes. Since we have everything set up and ready, it's just a matter of going to each scene, clicking on the **Animation** button on the **Render** panel (**Render** tab of the **Properties Editor**), and waiting for the image sequence to be created on the hard disk (do it by rendering one entire scene each time, then launching the render process for the next). It's very likely that rendering each sequence will take quite some time. Use this time to brew a pot of delicious Colombian coffee.
25. When the rendering is done, let's take a couple of minutes to inspect the resulting images in the folders at `/tmp/render/seqs/`; we can see that Blender adds a suffix indicating the number of the frame, which is needed to know the correct ordering of the images when assembling them into a video.

Objective Complete - Mini Debriefing

Having prepared the template scene, the current task was just a matter of using Blender's append mechanism to bring copies of it into the main file and then work out some small details to get each scene ready to be rendered into the correct sequence of images. This is the point where we can really see how useful it is to create template scenes from which to start out; though in this case we didn't use the more advanced options of Blender for handling libraries of objects, we certainly got to lower the time needed to create these three scenes.

Classified Intel

The mechanism that Blender uses for sharing or copying data between files is quite simple, but very effective. The idea behind all of it is that Blender organizes data into "packets" called datablocks; the list of folders that appears when browsing inside a blend file is just a list of the types of datablocks available in the file. The simplest datablocks to understand are *Scene* and *Object*; the others may seem a bit weird, but they are just ways of separating data into smaller packets: An *Object* datablock can make use (by referencing, not by containing) of datablocks such as *Mesh* or *Material*. The elegance of it all lies in Blender being smart enough so as to not only append or link the datablock that we select explicitly, but also to know which datablocks it depends upon and append or link them too. It's also worth saying that append and link are just words to denote the creation of a new copy (independent of the original file) or just referencing the datablock in the library file (dependent on the original file), respectively. In this project, we just created independent copies to avoid having to take care of the dependencies between files, which requires some previous planning to avoid headaches later.

A Word on Datablocks

Let's take some time to talk about datablocks; the concept can be quite different from what we are normally used to from other, widely known, software packages but the good thing is that it isn't complicated at all and mastering it is the key to using Blender more effectively.

Let's consider our `starFighter` object, brought from a previous project into the `item1` scene. When we see the object, we think of it as a whole entity; but if we start thinking about how it is composed, we can easily see that it has some separate and clearly distinct components, for example, the shape is different from the material. Taking this basic idea and pushing it further into a more technically oriented (not necessarily obvious) separation of components, Blender's designers decided that the way to handle and store data in Blender would be by packing specific sets of data into datablocks and allowing datablocks to reference other datablocks. Let's take a look at the consequences of that design decision:

- ▶ Some basic information of an object, such as location, rotation, and scale, along with some other data, is stored as a datablock of type `Object`.
- ▶ The set of data to describe a mesh (vertices, edges, and faces) is stored as a separate datablock, of type `Mesh`.
- ▶ The general settings of a Material (the ones editable in the **Material** tab of the **Properties Editor**) are stored as a `Material` datablock.
- ▶ The settings of a texture (available in the **Texture** tab of the **Properties Editor**) are handled as a single datablock of type `Texture`.

With this method, we have the possibility to create the separate components of an object, but how do we get the final assembly that constitutes the full object? That is done by allowing each type of datablock to reference some other types; for the simple object that we are considering, the datablock of `Object` type can reference **one** datablock of `Mesh` type, the `Mesh` datablock itself can reference **multiple** `Material` datablocks, and each `Material` datablock can further reference **multiple** `Texture` datablocks. If we start drilling down from the `Object` datablock into the datablocks referenced from it, we'll be able to build the entire object.

All of that fiddling with small packets of data has a very nice advantage: A single datablock can be referenced from as many datablocks as wanted, allowing us to reuse them easily. If we take a look at any of the objects with the "-wire" suffix in its name and go to the **Material** tab in the **Properties Editor**, we can see that in front of the name of the wire material there is a number; it's the counter of how many explicit "user" datablocks are referencing the wire material datablock. The word "explicit" is important to properly understand the mechanism, since if there were a `Texture` datablock being used by the wire material, it would be referenced explicitly only by one datablock (the wire material), thus having only one "user".

A good exercise to start understanding it all is to use the Duplicate Linked (**Object** → **Duplicate Linked**) command on any mesh object while in Object Mode, then switch to Edit Mode and see that, although the object datablocks are independent (each one can have different location, rotation, and scale values), they share the same **mesh** datablock and thus editing the mesh would cause both the objects to be affected. Let's go ahead and try it; it's going to help us understand this cool feature of Blender and, hopefully, start using it more effectively.

In the end, Blender's individual editors (Node Editor, Material tab of the Properties Editor, UV/Image Editor, Text Editor, and so on) can be seen just as specialized editors for each type of datablock that include an implicit mechanism of linking (referencing) from one datablock to another.

Composing the Final Video

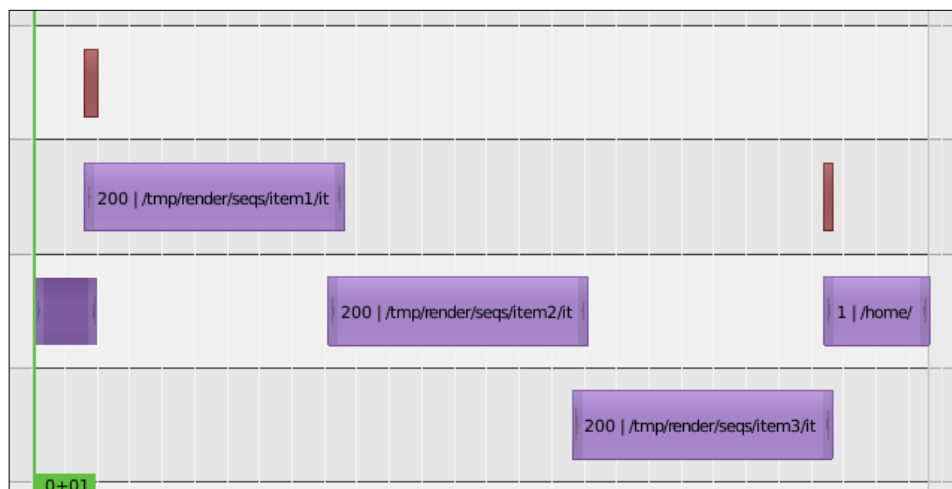
The last task in the project will be about taking the rendered images created in the previous scenes and using the Video Sequence Editor to assemble them into a video. Even though we'll not be using the advanced features of the Video Sequence Editor, this task will give us a good idea of how to work with it effectively.

Prepare for Lift Off

Before doing anything, let's check that we have the `intro.tga` image and the sequence of 200 images for each one of the `item1`, `item2`, and `item3` scenes in the appropriate places in the hard disk.

Engage Thrusters

1. Let's create a new empty scene in the `demoreel.blend` file and set its name to `video`.
2. Then switch to the **Video Editing** screen (drop-down menu beside the **Help** item in the main menu) to start adding the strips. Let's also make sure that the active scene for the **Video Editing** screen is "video". Let's take a look at the final result to have a guide beforehand:



In the previous screenshot we can see two effect strips (the thin ones) and five image strips. The work in the **Video Sequence Editor** is about adding strips from different sources (images, image sequences, videos, and blender scenes) and mixing or transitioning between them by means of effects. It is also possible to perform some special mixing and combining of strips into meta-strips, but we won't use that functionality in this project.

3. The first strip that we are going to add is the `intro.tga` image. To do this, go to the **Video Sequence Editor** and then to **Add** → **Image**. In the File Browser that appears, browse to the folder of the project and look for the `intro.tga` image, select it by clicking with the left mouse button on it, and then click on the **Add Image Strip** button, located at the top-right. Now there must be a strip representing the image in one of the channels (shown as rows) of the editor.
4. Right-click on the center of the strip, then start moving it (G key) and put it into channel 2 (channel numbers are on the left-hand side, aligned with the lower line of the channel). After positioning the strip on channel 2, let's now select the left handle by right-clicking on the small arrow and then move (G key) it until it is located at frame 1; then select the right handle and move it to frame **50**.
5. The second strip that we must add is the strip for the sequence of images rendered from the `item1` scene. To do this, click on **Add** → **Image** and go to `/tmp/render/seqs/item1`. Once the contents of the folder are being displayed, press the A key to get all the files selected, and click on the **Add Image Strip** button.
6. The strip that appears now must be located on channel 3 and moved (the whole strip, not each handle separately) so that its left and right handles are on frames **40** and **239**, respectively.

7. Make sure the strip is selected and press the *N* key to open the right-hand side sidebar of the editor, then look for the **Edit Strip** panel and set the Blend mode to **Alpha Over**, also enable the **Premultiply** option in the **Filter** panel.
8. The third strip to be added must be for the sequence of images that were rendered at `/tmp/render/seqs/item2`; let's not forget to select all the files (*A* key, once or twice) before clicking on **Add Image Strip**.
9. Once the strip is created in the **Video Sequence Editor**, let's select it and put it on channel 2. With the whole strip selected, move it so that the left handle is at frame **228** and the right handle is at frame **427**.
10. This strip must also have the **Blend** mode set to **Alpha Over** (**Edit Strip** panel at the sidebar) and the **Premultiply** option enabled (**Filter** panel).
11. For the fourth strip, we must go to the `/tmp/render/seqs/item3` folder (after clicking on **Add** → **Image**), select all the files (*A* key), and then click on the **Add Image Strip** button.
12. With the entire strip selected, let's move it (*G* key) to channel 1 and locate it so that the left handle is at frame **416** and the right handle is at frame **615**.
13. Once again, the "Blend" mode must be set to **Alpha Over** (**Edit Strip** panel on the sidebar) and the **Premultiply** option, located at the **Filter** panel, must be enabled.
14. To create the fifth (and last) strip, we just need to select the first one, duplicate it (**Strip** → **Duplicate** or press *Shift + D*), and move it to the right (staying on channel 2) until its left handle is located at frame **610**; then select the right handle and move it to frame **700**.

Now that we have all the strips created and positioned at the appropriate places, it's time to create the transitions between them. Since the strips for the image sequences already have the alpha channel modified to get a customized transition from one to another (the **Alpha Over** blend mode and **Premultiply** options were set for the precise purpose of using the customized transition), we only need to add transitions from the first strip to the second and from the fourth to the fifth.

15. The first transition must be created by selecting the first strip (`intro.tga`), then the second one (notice the order), and going to **Add** → **Effect Strip...** → **Cross**. This will add a small strip that starts where the second strip begins and ends where the first strip ends. This strip is positioned automatically on an upper channel, so that it is the one seen in the final render.
16. Transitioning from the fourth to the fifth channel is just a matter of selecting the strip for the image sequence of the `item3` scene, then selecting the last strip (`intro.tga`), and adding a Cross effect strip (**Add** → **Effect Strip...** → **Cross**). Once again, the start and end of the strip are dependent on the strips being used as inputs for the effect and the effect strip is positioned in an higher channel than the two strips being composed.

At this point, we can start scrubbing the video sequence by clicking and dragging with the left mouse button over the **Video Sequence Editor**; let's take a few seconds to move around and see how the sequence editor performs the mixing between the strips. Pay special attention to how the two effect strips affect the output and how the strips for the `item1`, `item2` and `item3` scenes also got the customized transition effect (in the frames that they overlap vertically) that we created by using the alpha channel in each of them.

17. To finish this task, we just need to set the output format for the final video and initiate the rendering process; to do that, go to the Default screen of the video scene (make sure both "Default" and "video" are set before continuing) of our file, then go to the **Render** tab in the **Properties Editor** and set the options as follows:
 - ▶ Dimensions panel:
 - ❑ Start: 1
 - ❑ End: 700
 - ▶ Output panel:
 - ❑ Output path: /tmp/render/demoreel-
 - ❑ File format: MPEG
 - ▶ Encoding panel:
 - ❑ Format: AVI
 - ❑ Codec: H.264
 - ❑ Bitrate: 700 (raise it up if the video quality after rendering isn't good enough)
18. Now we can save the file and start the rendering process by clicking on the "Animation" button (**Render** panel, in the **Render** tab of the **Properties Editor**).
Not much later, we will have our final demo reel ready to conquer the world!

Objective Complete - Mini Debriefing

In this task, we just took the previously rendered images and imported them into the Video Sequence Editor as strips, so that we could use the effects and set some options to get the video composed by transitioning between them appropriately.

For the image sequence strips, we didn't make use of explicit transitioning effects; since we already had a transition "baked" into the alpha channel of the rendered images, it was just a matter of setting the appropriate options for these strips and putting them in the correct vertical, overlapping order (for example, the `item1` strip above the `item2` strip) to get them transitioning automatically.

Finally, we only had to set a few options for Blender's rendering engine to take the composed video from the Video Sequence Editor and render it straight into a nicely compressed AVI file using the H.264 codec, which offers quite a good quality and high compression.

Classified Intel

The Video Sequence Editor works by representing fragments of video as strips; as we have seen before, there are different types of strips (even audio strips), and each individual strip must live in a specific channel. Blender treats channels like layers that are being seen from top to bottom; depending on the particular blend mode of a specific strip, the ones below it could or could not be visible. If we take this into account, it's easy to see that the two cross-effect strips added are the ones that are being rendered by Blender (in the frames where they exist), but they just perform the task of taking information from the two strips involved in the transition and generating the resulting image at each frame. For the custom transition between the image sequences we just used the Alpha Over blend mode, which uses the alpha information of the upper strip to decide how much of the strip below it to let "pass-through". The premultiply option for these strips is just needed for Blender to convert the data format of the individual pixels into the right one, to be used by the Video Sequence Editor correctly.

Mission Accomplished

In this project, we had the opportunity to learn a simple but very effective technique to get a nice studio setup. By using a simple setting in the material options for the floor plane of the template scene, the shadow can be seen without having to see explicitly the object that is receiving it. We also worked with a light setup in which the shadows were separated from the actual lighting, so that we can control them separately and have a greater control.

Since we have the append mechanism to copy information between blend files, it was quite easy to just create a template scene and then copy (append) it into new ones, thus saving time.

As a sneak peak of how much flexibility we have in Blender, we used an animated texture to drive the alpha channel of the template scene by using the Node Editor. Applying the same principle, it's just a matter of using any of the tools available in Blender (animation, physics simulation, particles simulation, and so on) to get a cool, new transition that can be used in the Video Sequence Editor.

One of the most important aspects of our demo reel is showing the mesh structure of the objects, so we just used a material of wire type and applied a simple trick using the Z Offset parameter to get a nicely shaded wire just over the object being rendered.

Working with the Video Sequence Editor of Blender allowed us to see how many tools the package does integrate. Blender not only offers us a very complete set of tools to create 3D objects and animations, it also gives us the possibility to perform post-processing automatically (using the Node Editor). Not only that, we can also take different sources of information (still images, image sequences, Blender scenes) and use them as strips in the Video Sequence Editor to get a finished and compressed video, straight from Blender. All those tasks done in the same package, with the tight coupling between the different parts (scenes can be used as strips) offers a quite powerful environment to play in composing a video, without having a "fight" between separated, heavy packages. All of that in a very small install downloadable for free. Isn't that really cool?

Are You Ready To Go Gung HO? A Hotshot Challenge

Our demo reel looks quite nice, though it could use some improvements, such as:

- ▶ Adding a sound clip to the video would be very nice to give it the final polishing needed.
- ▶ Working on a more elaborate custom transition could enhance the touch of sophistication to it. Be careful though, and avoid doing a transition that looks too complicated, since the demo reel is not about transitions :).
- ▶ Animating the camera to look directly at some special parts of our objects would be very helpful too. Remember to give it a nice slow motion to give the viewer enough time to see the details that you want to show.
- ▶ Adding an overlay image with your contact info over the entire duration of the reel can be very helpful for the viewer.
- ▶ Definitely, the best way to improve your demo reel is to create stunning pieces of CG artwork and awesome real-world projects showing combined skills; of course, doing that requires practicing and challenging yourself, so let's take this a never-ending challenge.

As a final word, it's important to say that the artwork to be included in an actual demo reel shouldn't be taken straight from a tutorial, since the viewer would easily notice. Instead, let's take this project as a mere first exercise to shed some light on how to do it.

Project 5

Build an Interactive Walkthrough Using the Game Engine

We've been doing stuff related to offline rendering only, trying to achieve a certain degree of realism. How about working on an interactive project? Something resembling a video game would be awesome, and Blender has some very nifty tools to do that. So, let's create a simple and interactive walkthrough.

Mission Briefing

Our objective is to create a basic scene that contains a central object and a good background. The objects in the scene will have materials created using techniques that are commonly used to create video game art, and we will also work on adding some cool functionality to move around in the scene, just like a first person shooter game.

The modeling part will be quite simple, since we want to concentrate more on the shading and scripting part. In the shading part, we are going to learn how to effectively use tileable textures and low resolution lightmaps to create shaders for our objects to look very nice when the user navigates around. In the scripting part, we are going to create a simple setup of objects, and set up some configuration and code for the **Blender Game Engine (BGE)**, namely logic bricks and Python scripts; they will allow us to create the interactivity that we aim for in the final product.

What Does It Do?

This project will further demonstrate how powerful Blender by itself can be. Starting with the commonly known functionalities of modeling and shading, we're going to use Blender's bake functionality to get a very beautiful skymap baked into a skydome. We're also going to create some low resolution light-maps to give a nice touch of realism to our scene, while learning how to save computer resources by using the **right** textures, **correctly**.

The most exotic task in this project will be where we use the Blender Game Engine. Even though it isn't a high-end game engine, we will see how useful it can be and how enjoyable it is to have it tightly coupled with Blender itself.

Why Is It Awesome?

This project will teach us some basic ideas on how to achieve good visual quality in a video game, by using some simple, well known techniques. The opportunity to create a product with a good final look will definitely help us get a good grasp on game development. Notice though that we are not going to use sophisticated techniques, just one that, although quite basic, is very effective to get the job done. Once we understand this project completely, it'll be easier to learn some more complex techniques.

This project will also demonstrate Blender's built-in capabilities. We have seen some of this in the previous projects, and this one will help us see it once again, in a different light. Putting it simply: Avoiding the hassle of export or import processes between different file formats and programs is definitely something that we're going to like about Blender, since it saves so much time and avoids so many headaches.

Finally, this project is designed to familiarize us with most components of the Blender Game Engine (BGE): Using logic bricks, adding properties to objects, and writing custom Python scripts. The only part that we're not going to deal with is the state machine, which is not needed for the simple problem that we are going to solve.

Overall, the definitive part of this project will be navigating around in the finished scene and knowing that we built it from scratch, using our favorite 3D suite!

Your Hotshot Objectives

Our interactive walkthrough must be a complete scene, fully modeled, shaded, and will also require some special logic to allow the camera to navigate the scene. Keeping that in mind, we will create it by going through the following stages:

1. Modeling the building.
2. Modeling the environment.
3. Shading the environment.

4. Shading the building.
5. Setting up the objects for the game engine.
6. Scripting.

Mission Checklist

This project is going to use some extra resources, so it's highly recommended that you organize all the files in a separate directory and create a specific folder for the textures. Let's create a folder in our system to hold the project and create a new folder inside it named `tex`, which will be used to hold the texture files.

Now let's download the image textures. Go to <http://blender-archi.tuxfamily.org> and click on the **Textures** link (on the left-hand side). We need to download the following images and save them into the `tex` folder:

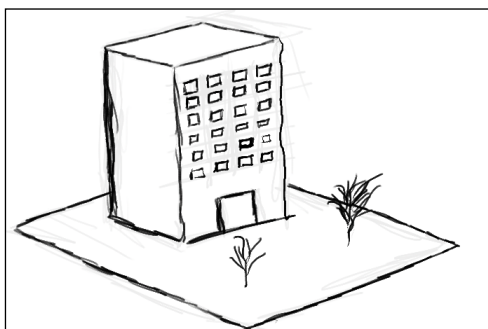
- ▶ Bricks (**Bricks** section)
- ▶ Cement floor (**Concrete** section)
- ▶ Grass (**Vegetation** section)
- ▶ Sky horiz 6 (**Horizontal maps** section)
- ▶ Dypsis (**Plants** section)

We must make sure to download the high-resolution version of each image, and not the thumbnail.

Apart from Blender, we will use Gimp to create some images as we need them through the process and to apply a very useful filter on the baked ones. Be sure to have Gimp installed and ready to use.

Modeling the Building

We will start by modeling the objects in our scene. In this case we are going to create a very simple scene, since our main purpose is to pay attention to the process, and the techniques involved in producing it. Let's take a look at a sketch showing what we are going to model:



Apart from the elements shown in the sketch, we will also be creating a wall of bricks around the base and a skydome surrounding all the other objects, to have a nice background.

In this first task, our objective is to model the building, which will be the main object of our scene and also the most complex one (though not really complex at all).

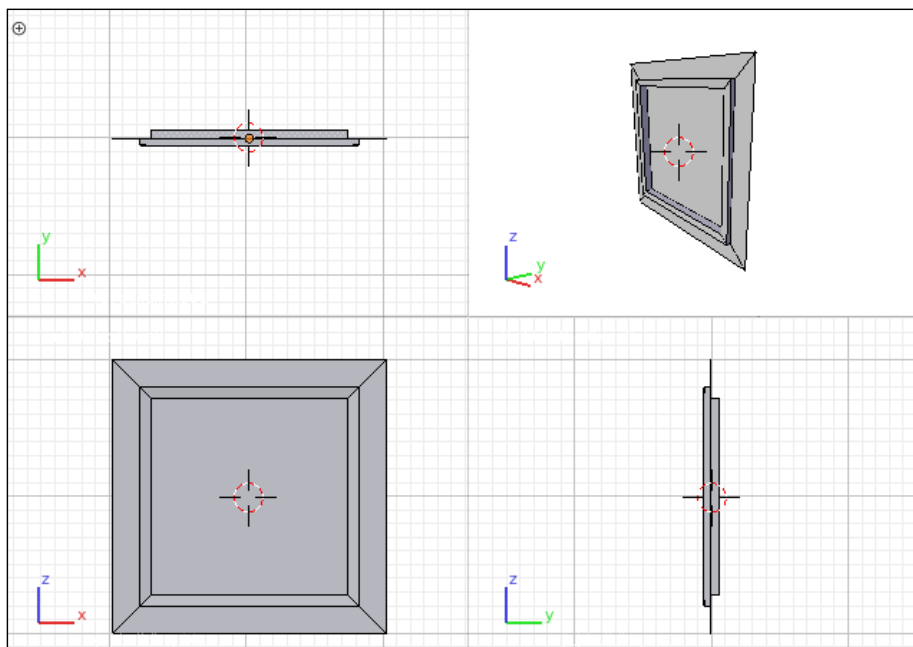
Prepare for Lift Off

Let's open Blender, delete the default cube and rename the default scene from `Scene` to `walkthrough`, then save the scene as `walkthrough.blend` in the project's directory.

Engage Thrusters

1. Switch to Front View (**View** → **Front**) and make sure the 3D Cursor is located at the origin (**View** → **Align View** → **Center Cursor and View All** or press *Shift + C*).
2. Now add a plane (**Add** → **Mesh** → **Plane**); go to the Tool Shelf (**View** → **Tool Shelf**), locate the **Add Plane** panel that appears in the lower part, and check the option for **Align to View**; that way the plane gets oriented towards the front view. Then locate the **Item** panel in the **Properties** sidebar (**View** → **Properties**) and change the name of the object to `building`.
3. Switch to Edit Mode and enable the Face select mode, then select all the faces (the *A* key, once or twice) and go to **Mesh** → **Delete** → **Only Faces**. This way, we are left with only four vertices and the edges connecting them.
4. Now perform an extrusion (the *E* key), press *Escape*, and then scale the extruded geometry by a factor of 0.8 (*S* key, then type `0.8`).

5. With the last extrusion selected, perform another extrusion (*E* key), press *Escape*, and then move it by 0.05 units in the negative direction of the *Y* axis (*G* key, *Y* key, then type -0.05).
6. Now extrude the selected vertices, press *Escape*, and then scale the new extrusion by a factor of 0.9 (*S* key, then type 0.9).
7. Finally, we just need to perform another extrusion. Press *Escape*, move the new geometry 0.1 units in the positive direction of the *Y* axis (*G* key, *Y* key, then type 0.1), and then fill the inner face by pressing the *F* key. The following screenshot shows the progress so far:



At this point, we're going to add the materials to the object. The reason to do this right now is that it will save us some repetitive work later. Notice though, we are just going to create the placeholders for the materials and not set them up completely.

8. Go to the **Material** tab in the Properties Editor and add three material slots. Then select the second and third slots and make sure that they have independent materials assigned to each one of them by clicking on the plus button next to the name (do this only if there is a button with a number to the right-hand side of the material's name).
9. For each material, locate the name text field (just below the **Assign, Select, Deselect** row of buttons); then name the first material *bricks*, the second *frames*, and the third *windows*.

To apply each material to a different part of the mesh, we need to select the faces to which the specific material will be assigned, select the material from the list, and click on **Assign**.

10. Let's select the inner face of the mesh, then select the `windows` material slot from the materials list and click on **Assign**.
11. Now select the faces forming the frame around the window (the protrusion, twelve faces in total), select the `frames` material from the list, and click on **Assign**.
12. Finally, deselect all the faces (the A key), click on the `bricks` material slot, and click on the **Select** button below the materials list. Only the faces created during the first extrusion should be selected in the mesh.

The last thing we must do to the materials is to assign them different colors to easily differentiate each part of the mesh.

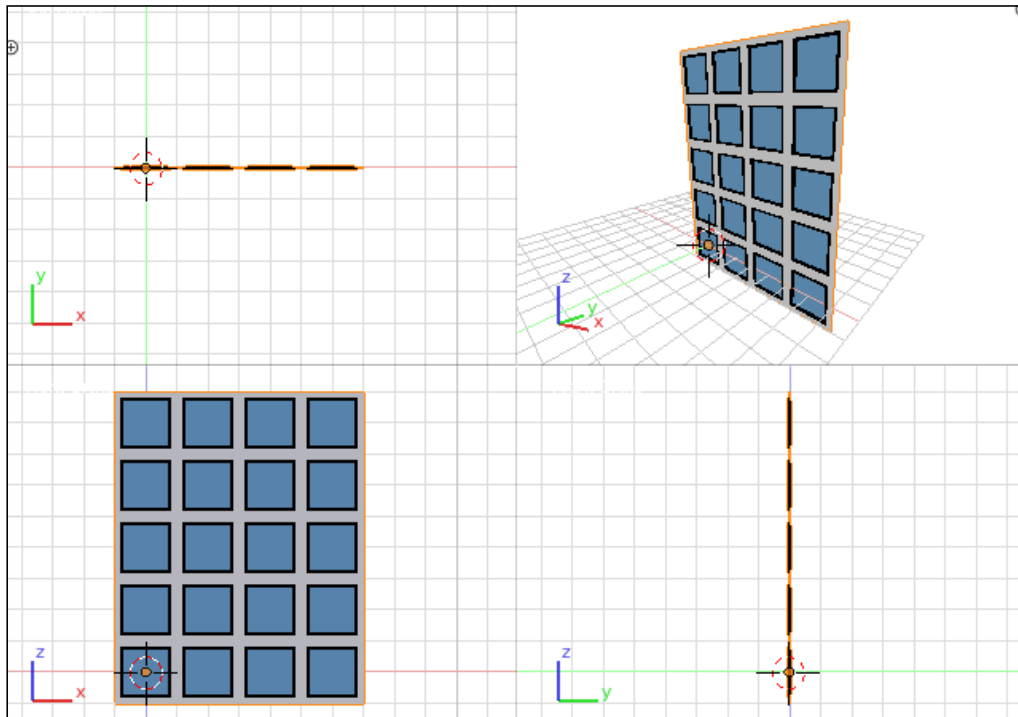
13. Let's select the `frames` material slot from the materials list and set its diffuse color to full black (0.0 Red, 0.0 Green, and 0.0 Blue).
14. For the `windows` material, let's set its diffuse color to 0.19 Red, 0.47 Green, and 0.8 Blue. Let's save the file too.

After creating the basic element for the front of the building, we just need to replicate it using two array modifiers to form the front of the building.

15. Let's switch to Object Mode and go to the **Modifiers** tab of the Properties Editor. Now click on the **Add Modifier** drop-down menu and select the **Array** modifier, then set the modifier options as follows:
 - ▶ Count: 4
 - ▶ Relative Offset: Enabled
 - ▶ X: 1.0
 - ▶ Y and Z: 0.0
 - ▶ Merge: Enabled
16. This array will create a row of windows; after it's created, let's click on the **Apply** button at the top of the **Array settings** panel.
17. Let's add another **Array** modifier, using the following settings:
 - ▶ Count: 5
 - ▶ Relative Offset: Enabled
 - ▶ Y: 1.0
 - ▶ X and Z: 0.0
 - ▶ Merge: Enabled

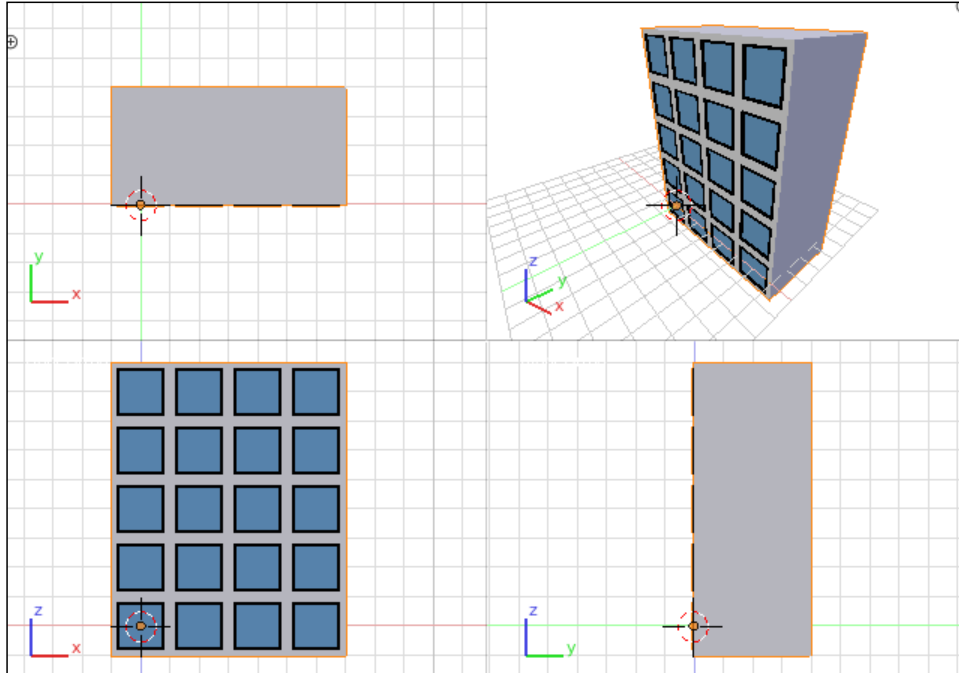
The second modifier must also be applied by clicking on the **Apply** button at the top of the modifier settings panel.

This will give us the full front of the building, with all the windows correctly organized, and the correctly inherited material settings from the base window element that we started with. The current state is shown in the next screenshot:



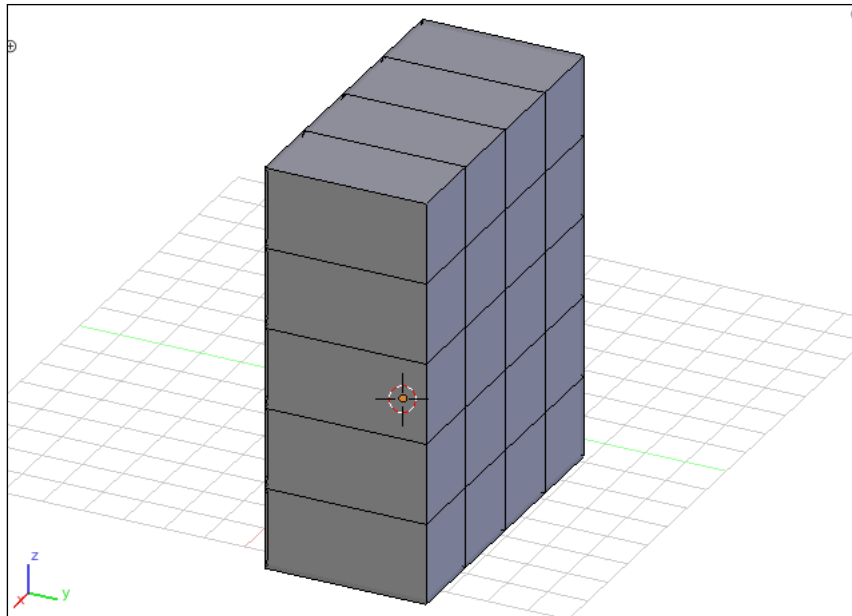
18. Now we need to switch to Edit Mode, enable Vertex select mode, and then select only the four vertices in the corners of the overall mesh we have now. With those vertices selected, let's perform an extrusion. Move them four units in the positive direction of the Y axis (G key, Y key, then 4) and then fill in the face formed by the extruded vertices by pressing the F key.
19. Now let's continue filling faces; this time we must create the faces that form the side, top, and bottom of the building by selecting two vertices from the front and two vertices from behind the building. This will give the building a solid appearance.
20. Since we created the faces by filling them one by one, it's very likely that they will have the shade smooth property enabled. Let's select them and click on the **Flat** button in the **Tool Shelf** sidebar.

21. Then let's assign the `bricks` material to this face by selecting its slot from the materials list in the **Material** tab of the Properties Editor and clicking on the **Assign** button, just below the materials list. The following screenshot shows the progress so far:



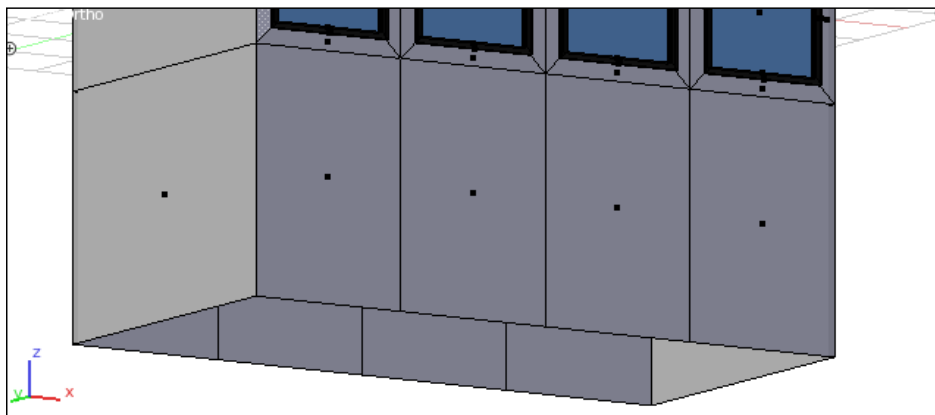
We must then perform two loop cuts (one horizontal and one vertical) to get the geometry of the side and the back of the building matching that of the front.

22. Let's start the **Loop Cut and Slide** tool by clicking on the button located in the **Tool Shelf** sidebar (**View** → **Tool Shelf**) or by pressing `Ctrl + R`. Then go to the mesh and point to one of the vertical edges of the back face so that the cut will run horizontally and scroll the mouse wheel up to increase the number of cuts to four. The loop cut running vertically must be increased to create three loops.
23. Once the new loops are added, it's time to select all the vertices (**Select** → **Select/Deselect All**, once or twice) and click on the **Remove Doubles** button located in the **Tool Shelf** sidebar (or press `W` → **Remove Doubles**). The header must show a message with an *i* icon saying **Removed 14 vertices**, which confirms that the operation went okay. Let's take a look at the next screenshot to check the result of these operations:

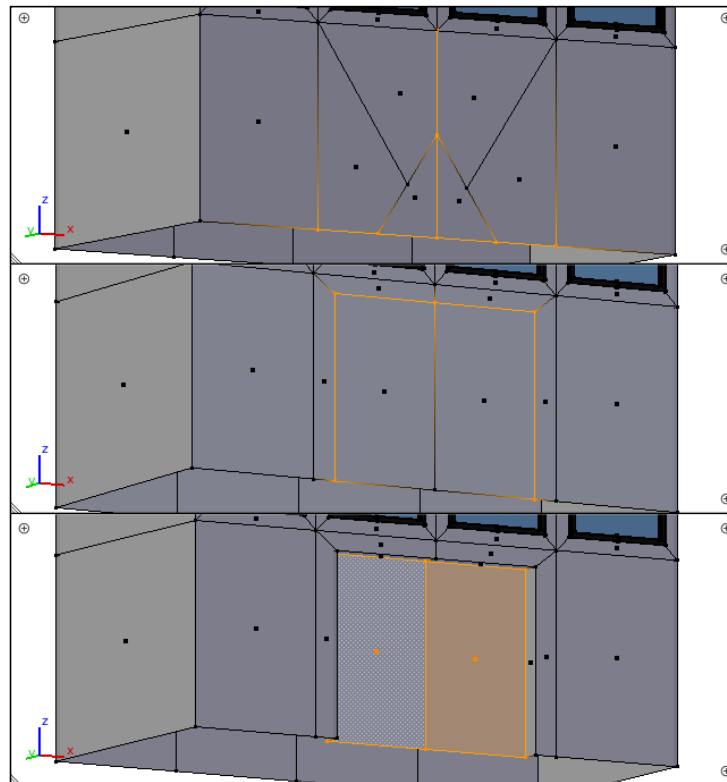


The only part missing from the model of the building is the entrance.

24. To create it, let's switch to the Face select mode, then select the four faces at the bottom of the mesh and extrude them by three units in the negative direction of the Z axis.
25. To clean up the mesh of unneeded geometry, let's delete the four faces forming the bottom; that way we get it open. The following screenshot shows how the bottom of the building should look by now:



26. Now it's time to switch to Edge select mode. Select the three edges connected to the vertex located in the middle of the bottom row of edges (front of the building), and subdivide them (**Mesh** → **Edges** → **Subdivide**). We then need to tweak around the vertices that were just added by the subdivide operation, to shape a frame for the main entrance.
27. Right after that, let's select the two faces inside the shaped frame and extrude them, press *Escape* to avoid moving the extrusion, and finally move it 0.3 units in the positive direction of the Y axis (G key, Y key, and then type 0.3).
28. After the extrusion is moved, let's select the two horizontal faces below the door's extrusion and delete them. Let's take a look at the following screenshot to better understand the process:



29. Let's finish this part by selecting the eight faces that form the frame for the door's entrance. Then go to the **Material** tab of the Properties Editor, select the `frames` material slot from the materials list, and click on the **Assign** button. Then perform the same process to assign the `windows` material to the two faces that form the main door for the entrance (the ones selected in the lower part of the previous screenshot).

30. Having the building model finished, we just need to switch to Object Mode, move it by three units in the negative direction of the X axis (G key, X key, and then type -3), then by two units in the negative direction of the Y axis (G key, Y key, and then type -2), and by four units in the positive direction of the Z axis (G key, Z key, and then type 4). Don't forget to save the file.

Objective Complete - Mini Debriefing

The most complex object that we're going to have is the building itself, though it's very simple. The modeling of the building became very easy by creating a single tile that was then repeated by using Array modifiers to form the front of the building. We then created the front entrance by performing an extrusion and doing some hand tweaking on the geometry. Among the other things we did, it's notable that we assigned the materials to the basic tile and got the remaining tiles with the same materials assigned as expected, thus saving some minutes of repetitive work later.

Modeling the Environment

Even though the main object of the scene will be the one to catch the attention, the environment plays a very important role, since it adds visual richness and reinforces the immersion that the user can experience when using our walkthrough.

Engage Thrusters

Let's start creating the environment closest to the building: The grass.

Grass

We're now going to create the simple mesh for the grass that will be located surrounding the base of the building.

1. Reset the 3D Cursor location (**View** → **Align View** → **Center Cursor and View All** or press *Shift + C*), then switch to Top View (**View** → **Top**) and make sure the view mode is set to orthographic (**View** → **View Persp/Ortho**), and the 3D Viewport shading mode is set to wireframe.
2. Then add a Plane (**Add** → **Mesh** → **Plane**) and change its name to `grass` (**Properties** sidebar, on the **Item** panel). With the plane selected, let's switch to Edit Mode, select all the vertices (**Select** → **Select/Deselect All**) and go to **Mesh** → **Delete** → **Only Faces** (to delete the inner face while preserving the vertices and edges).

3. Now scale the square shape along the X axis by a factor of four (S key, X key, then type 4) to match the borders of the building; then scale it by a factor of two along the Y axis (S key, Y key, then type 2); to correct the hole that is left between the door and the edge of the grass, right below it, select that edge and move it by 0.3 units in the positive direction of the Y axis (G key, Y key, then type 0.3).
4. Now select all the vertices, extrude them, press *Escape* to stop the grab operation, and scale the extrusion by a factor of 1.5 (remember we are in Top View).

We must now perform some tweaking to shape the extrusion correctly.

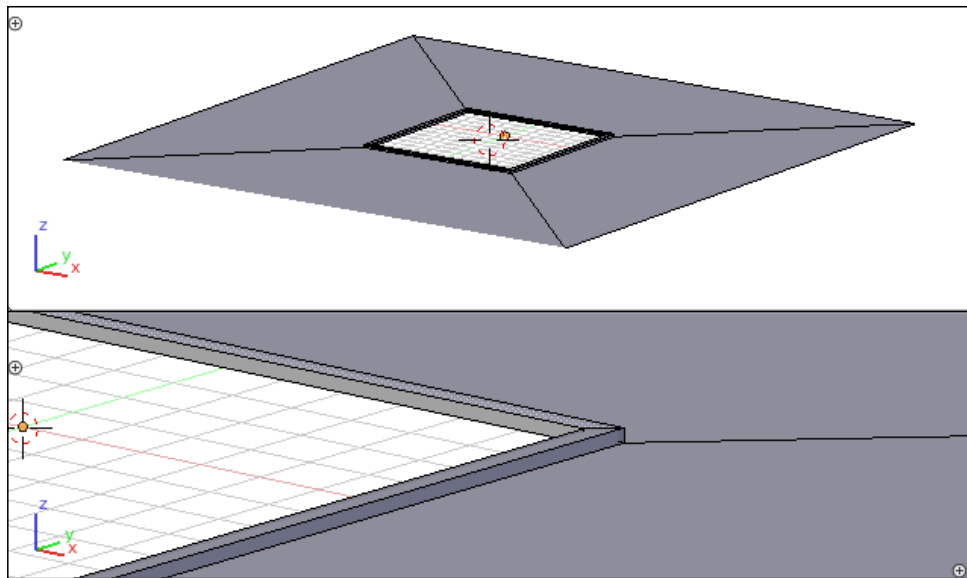
5. Let's select the horizontal edge from the last extrusion that is located at the top of the current view (Top View) and move it by one unit in the positive direction of the Y axis (G key, Y key, then type 1). Then select the horizontal edge that is located in the bottom part of the view, and move it by 5.3 units in the negative direction of the Y Axis (G key, Y key, then type -5.3). Now is also a good time to save the file.



A very common problem when working with meshes the way we are is that, sometimes, one or more faces can appear to be transparent (when viewed with Solid shading). To solve that, select the face and use **W** → **Flip Normals**, to get it showing solid, as expected.

Cement Floor

6. To create the cement floor, we will start by selecting the outer edges of the grass object, then duplicate them (**Mesh** → **Add Duplicate**), and then separate the duplicated geometry into a new object (**Mesh** → **Vertices** → **Separate** → **Selection**). Then switch back to Object Mode, select the new object, and change its name to `floor`. Switch to Edit Mode for this new object and start working on it.
7. Select all the vertices (**Select** → **Select/Deselect All** or press A), extrude them, then press *Escape* to stop moving the extrusion, and scale it by a factor of 1.06 (S key, then type 1.06).
8. Then select the two outer edges running vertically (Top View) and scale them along the X axis by a factor of 1.013 (S key, X key, then type 1.013). That should give the extrusion the same thickness on all four sides.
9. Now let's select the outer edges of the floor object, extrude them, press *Escape* to stop moving the extrusion, and then scale it up by a factor of 3.4.
10. To finish the floor object we just need to switch to the Face select mode. Select the inner four-sided ring of faces and extrude them up, moving the extrusion 0.2 units in the positive direction of the Z axis.



The final shape for the floor is shown in the previous screenshot. Notice the protrusion that must go in the inner part (surrounding the `grass` object).

Fence

11. Once again, we just need to select the outer edges of the floor object, duplicate them (**Mesh** → **Add Duplicate**), and then separate the duplicated edges into a new object (**Mesh** → **Vertices** → **Separate** → **Selection**).
12. Then switch back to Object Mode, select the new object and change its name to `fence` (**Item** panel in the **Properties** sidebar).
13. Switch to Edit mode for the fence object, select all its vertices and then extrude them three units in the positive direction of the Z axis.

Trees

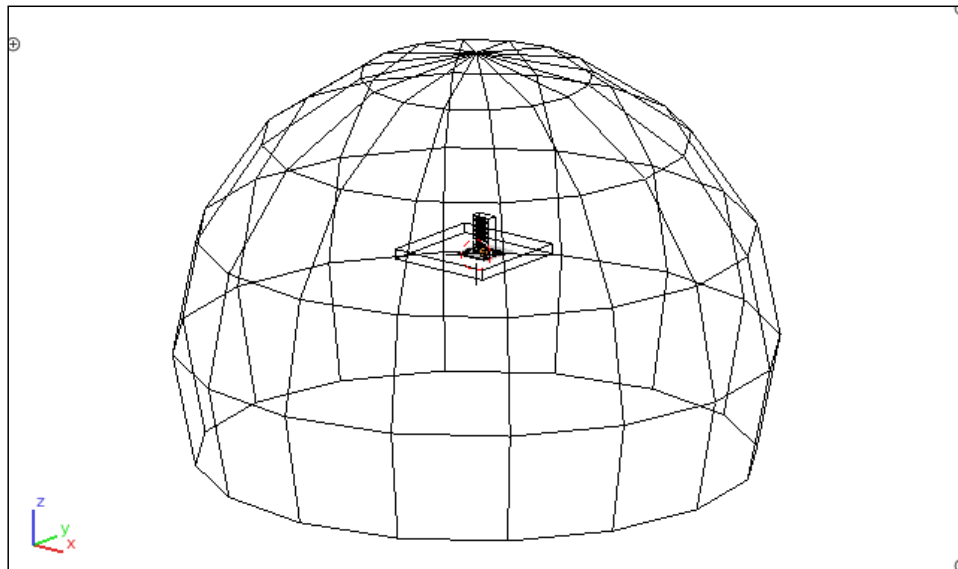
Our decorative trees are just going to be a couple of planes.

14. Start by putting the 3D Cursor in the origin of the 3D world (**Shift + C**), switch to Front View (**View** → **Front**), add a new plane (**Add** → **Mesh** → **Plane**), and enable the **Aligned to View** option in the **Tool Shelf** sidebar (**View** → **Tool Shelf**). This way, the plane gets oriented according to our current view of the scene. This object must be named `tree1` (in the **Item** panel, **Properties** sidebar).
15. Let's move the `tree1` object in the positive direction of the Z axis by one unit. Then duplicate it (**Object** → **Duplicate**) and name it `tree2`. Finally, let's move both `tree1` and `tree2` to a good location just over the `grass` object.

Skydome

To create the skydome we just need a common UV Sphere.

16. Let's make sure the 3D Cursor is positioned at the origin of the 3D World (**View** → **Align View** → **Center Cursor and View All**), then add a UV Sphere (**Add** → **Mesh** → **UV Sphere**), and set its **Segments** to 16 and its **Rings** to 8 (**Tool Shelf** sidebar).
17. Now switch to Edit Mode, deselect all the vertices (**Select** → **Select/Deselect All**). Select the south pole vertex and the two rings above it, and delete them (**Mesh** → **Delete** → **Vertices**).
18. Finally, let's select all the remaining vertices and scale them up by a factor of 120 (S key, then type 120). Don't forget to set the name of the object as `skydome` (in the **Item** panel, **Properties** sidebar). The following screenshot shows the entire scene:



To finish the modeling task, we just need to perform a simple check of the objects that we just created.

19. Let's select the `fence` object, then switch to Edit Mode, enable the display of face normals in the **Mesh Display** panel (in the **Properties** sidebar), and set the normal size to 0.5.

20. Now let's navigate the scene and check that the normal of each face is pointing correctly towards the "outside" of the object. For example, the normals for the fence object must point into the building, since that's the direction of the outside of the **surface** of the fence. In case we find a face with a face normal pointing in the wrong direction, we can correct it by selecting the face and going to **Mesh** → **Faces** → **Flip Normals**. Let's perform the same check and corrections for each object in the scene (except for the skydome), to make sure we won't have any misbehaving objects later in the process.

Now let's save the file and continue with the next task of the project.

Objective Complete - Mini Debriefing

In this project, we divided the process by the type of task, rather than by object. This task, along with the previous one, was just about constructing the geometry that will constitute the scene for our interactive walkthrough. The subsequent tasks will be about shading and generating the logic system needed to get the desired behavior.

What we created was a very simple scene in which a single building is located inside a wall fence; we also added a big skydome that will help us get a nice background.

The environment objects are very simple in shape, and we avoided having to manually match the borders between them by creating one from a duplicate of the border of another (using the Separate command).

Shading the Environment

In the previous tasks we created all the models that we need for our project. This time we are going to apply textures (downloaded from the public repository of the **Blender for Architecture** website) to the objects that constitute the environment of our scene.

We are going to use two different techniques for texturing the objects. Texturing for the skydome will be done using the special spherical map and getting it baked into the skydome object by using a UV mapped image. The other objects in the scene will require the downloaded images only as diffuse maps (defining color) and then baking low resolution lightmaps (defining how luminous each part of the mesh is). The nice thing about all of this is that Blender offers us all the tools we need, except for the Selective Gaussian Blur that Gimp will provide us with.

Engage Thrusters

For this part, let's switch the order we followed when modeling the objects and start with the skydome, which is the only one using a different method of shading (and baking) than the remaining objects.

Skydome

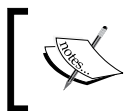
1. Let's start by selecting the `skydome` object and setting its shading mode to smooth (the **Smooth** button in the **Tool Shelf** sidebar).
2. Then let's go to the Properties Editor, click on the **World** tab, and adjust its settings as follows:
 - ▶ World panel:
 - ❑ Real sky: Enabled
 - ❑ Horizon Color: 0.0 Red, 0.0 Green, 0.0 Blue
 - ❑ Zenith Color: 0.0 Red, 0.0 Green, 0.0 Blue
3. After setting the properties for the world background, let's go to the **Texture** tab in the Properties Editor, select the first texture slot, and add a new texture. Change the name of the texture to `skymap` and set it as follows:
 - ▶ Type: Image or Movie
 - ▶ Coordinates: Sphere (Mapping panel)
 - ▶ Influence panel:
 - ❑ Blend: Disabled
 - ❑ Horizon: Enabled
 - ❑ Blend: Add
4. Now let's click on the **Open** button in the **Image** panel and load the image called `Sky_horiz_6.jpg` from the `tex` folder in the project's directory, to get the texture complete.

Note that this texture is related to the world background, and not to any material—take a look at the row of buttons just below the preview and you'll see a button titled **World**. The final result of this process is that the world background will use the `skymap` texture and the mapping will cause it to look like a real sky surrounding our scene. However, since our project is going to run in the Blender Game Engine, we need to bake this texture into the skydome object's material.

Before attempting to bake, it's necessary to set an appropriate material on the `skydome` object, so that it receives every color exactly from the surrounding `skymap`.

5. Let's go to the **Material** tab in the Properties Editor, add a new material for the skydome object, and set the material's name as `perfectGlass`. We then just need to go to the **Mirror** panel, enable it, and then set the **Reflectivity** all the way up to 1.
Since the bake information must be embedded within an image, we need to create a new image that will be used for that purpose.

6. Let's open Gimp and create a new image 512 x 512 pixels in size. It doesn't matter what background color the image has; let's save it as `bake-skymap.tga` in the `tex` folder, within the project's folder, and click on **Save** in the **Save as TGA** dialog. Let's close the image, but keep Gimp open.
7. Back in Blender, let's split the 3D View. Drag the bottom-left corner by moving the pointer. In the lower window that appears, let's open a UV/Image Editor then go to **Image** → **Open** and load the `bake-skymap.tga` image.
8. Now go to the 3D View, switch to Edit Mode for the skydome, select all the vertices, and go to **Mesh** → **UV Unwrap...** → **Unwrap** to get a nice UV map of the mesh of the skydome shaped like a spiderweb.
9. Then go to the UV/Image Editor, select all the vertices of the UV map (**Select** → **Select or Deselect All**) and scale them down to give just a bit of space from the borders of the image.
10. Then let's deselect the outer ring of vertices of the UV map by hitting the **C** key and then "drawing" with the middle mouse button over the vertices that must be deselected. Having deselected the outer vertices, we only need to scale the selected ones up until the outer ring of faces becomes thin (not too thin, though); we do this only to get more pixels of the image being used for the really important part of the bake.
11. Let's also set the name of the UV Layout by going to the **Object Data** tab in the Properties Editor, locating the **UV Texture** panel, and then changing the name from `UVTex` to `skymap`.
12. To perform the bake, we must go to the **Render** tab in the Properties Editor, scroll down to the **Bake** panel, make sure the **Bake Mode** is set to **Full Render**, and then click on the **Bake** button. Now we should see the `bake-skymap.tga` image being filled with the image of the skymap, shaped according to the UV map we created.
13. We can take a look at the bake applied on the skydome by going to the **Display** panel in the **Properties** sidebar of the 3D View, setting the **Shading** method to **Texture Face** or **Multitexture**, and then setting **Viewport Shading** to **Textured** (drop-down menu to the right of the **Object Mode** menu).
14. Let's also perform a final change in the skydome mesh: Switch to **Edit Mode**, select all the vertices (the **A** key, once or twice), and then go to **Mesh** → **Faces** → **Flip Normals**.



The image `bake-skymap.tga` stored in the hard disk hasn't been changed, the bake is now being handled by Blender in memory (notice the asterisk in the **Image** menu of the **UV/Image Editor**).

Since the result of the bake is the one we wanted, let's save it over the original image.

15. In the UV/Image Editor, go to **Image** → **Save As**, then browse to the `tex` folder and select the `bake-skymap.tga` image, and make sure to set the **File Type** to **Targa** and enable **Relative Path (Save As)** panel in the left-hand side sidebar of the **File Browser**. Then click on the **Save As** button in the top right-hand side. After saving successfully, the **Image** menu shouldn't have the asterisk we saw before, meaning the shown image is synchronized with the file in the hard disk.

Taking a closer look at the baked image, you may notice a black background that doesn't play so well with the sky map. Let's go to Gimp to perform a basic fix for that.

16. Open the `bake-skymap.tga` image in Gimp, which now shows the baked result, and choose the **Color Picker** from the toolbox; then pick the color of one of the bluish pixels in the border of the bake, switch to the **Bucket** tool, and perform a click on the black region to fill it with the blue color. This way the bake will look much better than with that black spot. Remember to go to **File** → **Save** and close the image (keep Gimp open).
17. Then we must go back to Blender and go to **Image** → **Reload** in the UV/Image Editor, to refresh the image data from the hard disk.

Finally, we only need to perform a couple of steps to get the bake correctly set up for the Game Engine.

18. With the `skydome` selected, go to the **Material** tab of the Properties Editor and delete the `perfectGlass` material. Then add a new material, change its name to `skyBackground`, and set it as follows:
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (Diffuse panel)
 - ▶ Shadeless: Enabled (Shading panel)
19. After setting the material, let's go to the **Texture** tab of the Properties Editor, add a new texture, and name it as `skyBg`. Then set its **Type** to **Image or Movie**. Having the type of texture set, let's go to the **Image** panel and click on the drop-down menu with the landscape icon and select the `bake-skymap.tga` image from the list, then continue setting the texture as follows:
 - ▶ Mapping panel:
 - Coordinates: UV
 - Layer: `skymap`
 - ▶ Blend: Add (Influence panel)

20. To continue working on the other objects more easily, let's move the `skydome` object to layer two of the scene: Select the `skydome` object and switch to **Object Mode**, then go to the **Object** tab in the Properties Editor and click on the button for `layer2` in the **Relations** panel. This way, all the objects live in `layer1`, except for the `skydome` that lives in `layer2`. Let's save the file too.

Grass

21. Make sure to be in **Object Mode**, then select the `grass` object (just below the building), switch to Top View (**View** → **Top**), and enter Edit Mode.
22. Now go to the UV/Image Editor, and go to **Image** → **Open**, then browse to the `tex` folder, select the `grass.jpg` file, and open it (top right button of the **File Browser**). At this point, the image should be loaded and shown in the UV/Image Editor.

Next, we must get the mesh of the grass object unwrapped to get its UV map over the image.

23. Go back to the 3D View (remember we are in Edit Mode), select all the vertices (the **A** key), and then go to **Mesh** → **UV Unwrap...** → **Unwrap**. This will give us a 2D "mesh" in the UV/Image Editor, resembling almost exactly the shape of the mesh in the 3D View.
24. To scale the grass texture on the object, we must go to the UV/Image Editor, select all the vertices, and then scale them up by a factor of 4.5 (**S** key, then type `4.5`). Since the image is tileable and the UV layout is bigger than the image, it will be repeated nicely on the whole surface of the faces of the object.
25. In order to keep things correctly organized, let's go to the **Object Data** tab in the Properties Editor and change the name of the UV layer to `diff` (**UV Texture** panel).
26. To take a peek at how it looks, let's go to the 3D View and enable the **Multitexture** mode in the **Shading** drop-down of the **Display** panel (**Properties** sidebar); remember we must also set the **Viewport Shading** to **Textured**.

The `grass` object also requires the creation of a bake. This time we are going to bake a lightmap, which is useful to control which parts of the surface of the mesh look bright and which look dark.

27. In the **UV Texture** panel (**Object Data** tab in the Properties Editor), let's add a new UV layer, change its name to `bake`, and leave it selected (click on the item in the list).
28. Then go to Gimp and create a new image of 128 x 128 pixels (yes, quite small) and save it as `bake-grass.tga` (in the `tex` folder; remember also to close the image (not Gimp)).

29. Back in Blender, let's go to the UV/Image Editor and click on **Image** → **Open**, select the **bake-grass.tga** image from the hard disk, and open it.
30. Now it's time to unwrap the mesh again, let's go to the 3D Viewport and switch to Edit Mode (the `grass` object must be active). If the UV/Image Editor switches automatically to the `grass.tga` image, let's go there and switch back to `bake-grass.tga` (click on the drop-down menu, to the left-hand side of the name of the image).
31. Once the correct image is selected and the mesh is in Edit Mode, we just need to go to the 3D View and unwrap the mesh (**Mesh** → **UV Unwrap...** → **Unwrap**). This will give us a new 2D "mesh" above the image in the UV/Image Editor. We just need to make sure it is entirely contained within the image and scale it down a bit to leave some space between the borders of the image and the UV layout.

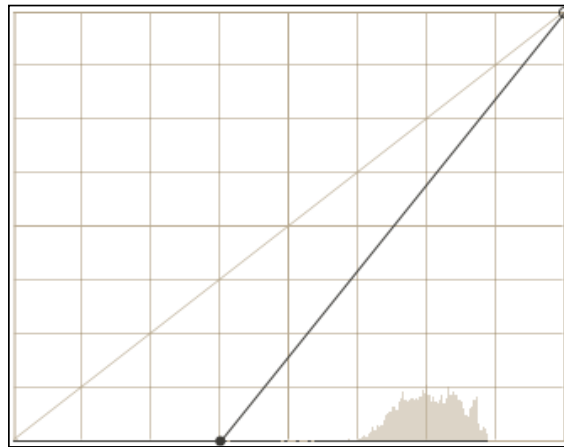
To create the bake of the lightmap, we are going to use the Ambient Occlusion capabilities of Blender's internal render engine, so we need to configure it appropriately.

32. Let's go to the **World** tab in the Properties Editor and set it as follows:
 - ▶ Ambient Occlusion panel: Enabled
 - ❑ Factor: 1.3
 - ❑ Mode: Add (drop-down menu)
 - ▶ Gather panel:
 - ❑ Method: Raytrace
 - ❑ Sampling: Adaptive QMC
33. Then go to the **Texture** tab and let's set the Type of the sky map texture to `None` (thus disabling it).
34. After all that configuration, we are ready to perform the bake; let's make sure that the `grass` object is selected and the bake UV layer is the one that is active (**UV Texture** panel in the **Object Data** tab of the Properties Editor). Then go to the **Render** tab of the Properties Editor, locate the **Bake** panel, set the **Bake Mode** to **Full Render**, and click on the **Bake** button. We should see the `bake-grass.tga` image getting filled in with a grayscale image showing the result of the bake.

The baked image has a small problem, though: It looks noisy.

35. To correct that, let's update the file on disk with the image we see currently in the UV/Image Editor by going to **Image** → **Save As...** and then replacing the original `bake-grass.tga` image (remember to set the **File Type** as **Targa** and enable the **Relative Path** option).

36. Now let's go to Gimp and open the `bake-grass.tga` image. Once it's loaded, let's go to **Filters** → **Blur** → **Selective Gaussian Blur**, set the Blur radius to 10 and the Max. delta to 100, and then click on **OK**.
37. Finally, we just need to apply a color adjustment. To do that, go to **Colors** → **Curves** and set the curve as shown in the following screenshot:



The purpose of the curve is just to increase the contrast of the lightmap, so that the final result will look very nice.

38. Let's now save the image and close it, but remember to keep Gimp open.
39. Having modified the baked image, we need to reload it in Blender. Let's go to Blender, make sure the `bake-grass.tga` image is the one being shown in the UV/Image Editor and then go to **Image** → **Reload** to refresh it from the hard disk.
The ingredients to set up the material are ready (the diffuse and lightmap textures), so let's go to the **Material** tab of the Properties Editor and create the material for the `grass` object.
40. Add a new material, name it `grass`, and set it as follows:
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (Diffuse panel)
 - ▶ Shadeless: Enabled (Shading panel)
41. Then let's go to the **Texture** tab, add a texture, and name it as `grassDiffuse`. Then set its **Type** as **Image or Movie** and select the `grass.jpg` image from the drop-down in the **Image** panel. Other settings are as follows:
 - ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: diff

- ▶ Influence panel:
 - ❑ Color: Enabled
 - ❑ Blend: Add
- 42. A second texture must be added; name it `grassLightmap`, set its **Type** to **Image or Movie**, and select the `bake-grass.tga` image from the drop-down list in the **Image** panel then set it as follows:
 - ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: Bake
 - ▶ Influence panel:
 - ❑ Color: Enabled
 - ❑ Blend: Multiply

Let's explain how the material works: The diffuse channel starts with the black color, the first texture is added, so the result until now is the color that the `grassDiffuse` texture defines. The lightmap is then multiplied on top of the current appearance. Since our lightmap is a grayscale image (not the image format, but the actual colors), the effect is a modulated brightness of the diffuse image. That's the trick for all the materials we are going to create from now on. Notice one thing: These materials are shadeless, which means the lighting of the scene won't affect them.

- 43. To take a look at what the grass material will look like, we can temporarily enable the GLSL Shading (**Display** material in the **Properties** sidebar of the 3D View); let's also save the file.

Floor

- 44. Make sure we are in Object Mode. Select the `floor` object and then switch to Edit Mode.
- 45. Go to the UV/Image Editor and click on **Image** → **Open**, then open the `cement_floor.png` image.
- 46. Back in the 3D View, let's switch to the Edge select mode and select the four edges connecting the flat part of the floor object with the "squared-ring" that we extruded when modeling. After performing that selection, let's add to the selection the edges running diagonally in the raised part of the floor object (three per corner, 12 in total).

47. After selecting all of them, let's go to **Mesh** → **Edges** → **Mark Seam**, to get them tagged appropriately to apply the automatic unwrap that Blender can do.
48. Now let's select all the edges and go to **Mesh** → **UV Unwrap...** → **Unwrap**; the UV/Image Editor will now have the UV layout above the cement floor texture.
49. Let's go to the UV/Image Editor, select all the vertices of the layout, and scale it up by a factor of five. To check how well the texture scale and mapping looks on the model, we can switch to Multi texture shading (**Display** panel of the **Properties** sidebar). If the borders of the extrusion of the floor don't seem to have the texture looking correctly, we can go to the UV/Image Editor, locate the island that corresponds to that part of the mesh (there is a "sync" button in the header for that), and then edit it manually to fix the mapping problem.
50. When the object in the 3D View looks correctly mapped, let's just rename the UV layer to `diff` (**UV Texture** panel in the **Object Data** tab of the Properties Editor), and add a second UV layer with the name `bake`.
51. The lightmap must be mapped apart from the diffuse channel, so let's go to the **Object Data** tab of the Properties Editor and add a new UV layer (**UV Texture** panel), remember to set its name to `bake`.
52. Now let's go to Gimp, create a new image of 256 x 256 pixels and save it as `bake-floor.tga` in the `tex` folder; remember to close the image in Gimp.
53. Back in Blender, let's make sure that the `floor` object is in Edit Mode and the active UV layer is the second one (`bake`), then load the image by going to **Image** → **Open** (UV/Image Editor), browse to the `tex` folder, select the `bake-floor.tga` image, and click on **Open**.

Now we have everything ready to unwrap the mesh to create a UV layout.

54. Let's go to the 3D View, select all the vertices and then go to **Mesh** → **UV Unwrap...** → **Smart Project**. In the **Tool Shelf** sidebar, let's locate the "Island Margin" parameter and set it to `0.12` (that way we get some space between the islands). It's highly recommended to check the thin islands of the UV layout and edit them to get enough thickness (the bigger islands can be resized down to have more space for the smaller ones).

The next step is to perform the bake operation.

55. Let's make sure that the image being shown in the UV/Image Editor is `bake-floor.tga` and the UV layer selected is **bake**; then go to the **Render** tab of the Properties Editor and click on the **Bake** button (**Bake** panel).
56. Once the bake is complete, let's save it over the original image by going to **Image** → **Save As**.

57. After the image is saved, go to Gimp and open it to apply the **Selective Gaussian Blur** filter (**Filters** → **Blur** → **Selective Gaussian Blur**, with a **Blur radius** of 10 and **Max delta** of 100) and the color correction curve shown in the previous screenshot (**Colors** → **Curve**). Let's save the edited image and close it (leaving Gimp open).



It's quite common for the baked image to have some artifacts even after applying the filter. In these cases, it's just a matter of using a brush or the smudge tool to clean it up manually.

58. Back in Blender, let's go to the UV/Image Editor and reload the image (**Image** → **Reload**).

The only remaining steps are the ones required to set the material for the floor.

59. Let's switch to Object Mode and then go to the **Material** tab of the Properties Editor. Add a new material, name it `floor`, and set it as follows:
- ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (Diffuse panel)
 - ▶ Shadeless: Enabled (**Shading** panel)
60. Now go to the **Texture** tab, add a new texture and set its name to `floorDiff`. The texture must be of **Image or Movie** type and we must select `cement_floor.png` from the drop-down menu in the **Image** panel, then set it as follows:
- ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: diff
 - ▶ Influence panel:
 - ❑ Color: Enabled
 - ❑ Blend: Add
61. Once the diffuse texture is in place, we must add a second texture and name it as `floorBake`; the **Type** must be set to **Image or Movie** and we must select the `bake-floor.tga` image from the drop-down menu in the **Image** panel. The settings for this texture are as follows:
- ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: Bake

- Influence panel:
 - Color: Enabled
 - Blend: Multiply

We can go to the 3D View and set the shading method to **GLSL** (**Display** panel in the **Properties** sidebar) to see how the object looks with the material applied. It'd also be good to save our file now.

Fence

62. Let's switch to **Object Mode**, select the `fence` object, and then go into **Edit Mode** for it.
63. Then we must switch to Edge select mode and select the four vertical edges in the corners of the fence. Once selected, let's go to **Mesh** → **Edges** → **Mark Seam**.
64. Now go to the UV/Image Editor and open (**Image** → **Open**) the image `Bricks.jpg`. If the image doesn't show up in the editor, let's go to the drop-down list in the header and pick it from the list.
65. Once the image appears, let's go to the 3D View, select all the vertices of the `fence` object and go to **Mesh** → **UV Unwrap...** → **Unwrap**. The UV layout will appear in the UV/Image Editor with the four faces put one "above" the other.
66. Let's now select all the vertices of the UV layout and scale them up by a factor of 21.
67. Let's also go to the **Object Data** tab in the Properties Editor, change the name of the `UVTex` UV layer to `diff`, and add a new UV layer named `bake`.
Once again, we need a new image for the bake.
68. Let's go to Gimp, create a new image of 128 x 128 pixels in size, and save it as `bake-fence.tga` in the `tex` folder. Don't forget to close the image.
69. Back in Blender, let's load this image in the UV/Image Editor (**Image** → **Open**), then make sure that the active UV layer is the second one (`bake`), and the image shown in the UV/Image Editor is `bake-fence.tga`. Go to the 3D View, select all vertices, and perform a Smart Project unwrap (**Mesh** → **UV Unwrap...** → **Smart Project**), for the **Island Margin** parameter (**Tool Shelf** sidebar), let's use 0.12.
70. Now it's time to go to the **Render** tab in the Properties Editor, locate the **Bake** panel, and click on the **Bake** button.
Now that the render result is baked into our image, let's save it over the original one (**Image** → **Save As**, in the UV/Image Editor) and then go to Gimp to give it some final touches.

71. Let's open the new `bake-fence.tga` image in Gimp and apply the Selective Gaussian Filter (**Filters** → **Blur** → **Selective Gaussian Blur**) by using 10 for the radius and 100 for the Max. delta. Then we must go to **Colors** → **Curve** and apply the curve shown in the previous screenshot. After we are done with this, let's save the edited image, and close it.
72. Back in Blender, we just need to reload the image (**Image** → **Reload**, in the UV/Image Editor) to get it refreshed with the editing done in Gimp.
Finally, let's set up the material for the `fence` object, using the two textures.
73. Switch to **Object Mode** and go to the **Material** tab in the Properties Editor; then add a new material named `fence`, set its diffuse color to black, and enable the **Shadeless** option.
74. For the fence's material, we need to define the usual two textures. Go to the **Texture** tab and start by adding a texture named `fenceDiff` of type **Image or Movie**. Choose the `bricks.jpg` image from the image datablocks list (**Image** panel) and set it as follows:
 - ▶ Mapping panel:
 - Coordinates: UV
 - Layer: diff
 - ▶ Influence panel:
 - Color: Enabled
 - Blend: Add
75. Let's add the second texture; name it `fenceBake`. Change its type to **Image or Movie**, select the `bake-fence.tga` image from the image datablocks list (**Image** panel), and set it as follows:
 - ▶ Mapping panel:
 - Coordinates: UV
 - Layer: Bake
 - ▶ Influence panel:
 - Color: Enabled
 - Blend: Multiply

We can take a close look at the object by using the GLSL shading method (**Display** panel, **Properties** sidebar) and setting the **Viewport Shading** to be "Textured". Let's also save the file now.

Trees

For the trees, we are just going to use a diffuse map without baking a lightmap. The reason is that they are quite small in the scene and don't require really detailed work.

76. Let's select the object named `tree1`, then go to the **Material** tab of the Properties Editor, and add a new material; then change its name to `tree` and set it as follows:
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (Diffuse panel)
 - ▶ Shadeless: Enabled (Shading panel)
 - ▶ Transparency panel: Enabled
 - Alpha: 0.0
77. Then let's go to the **Texture** tab and add a new **Image** texture with the name `treeDiff`. Click on the **Open** button (**Image** panel) and load the image `Dypsis01.png` from the `tex` folder. For this image, we only need to enable both **Color** and **Alpha**, and set the Blend mode to **Add** (all of them in the **Influence** panel).
78. For the `tree2` object, we just need to select it, then go to the **Material** tab in the Properties Editor, and select the `tree` material from the drop-down list. Notice the number **2** to the right-hand side of the name of the material, which means the material datablock is being used by two meshes (`tree1` and `tree2` have independent mesh datablocks).

Objective Complete - Mini Debriefing

This part of the project was about using the tileable textures that we downloaded as diffuse maps and creating some nice bakes directly within Blender to have good lightmaps for the scene.

The main reason to do all this is because we are going to use the Blender Game Engine as the final platform for our walkthrough, which narrows down the shading options, as compared to the normal shading options for an offline render. Another reason is that by shading the objects in this way, the graphics card of a computer can perfectly handle the task of rendering the scene at interactive frame rates. At the time of writing this book, there is not the possibility to use hyperrealistic render engines at interactive frame rates (at least using commercial products, since research is being done heavily on that). To summarize it: All this work is to make processing easier for the graphics card, so that it works nicely for us.

Take into account that the separation of the diffuse texture and the lightmap is very useful, since it allows the diffuse map to be used in other materials, mixed with a different lightmap. Even more, it's possible to mix two or more lightmaps into new materials, which gives way more flexibility and reduces the resource consumption of the final standalone.

Shading the Building

Let's continue applying the shading and baking method that we used in the previous task, this time for shading the building in our scene.

Engage Thrusters

1. Let's select the `building` object, and go straight to add two UV layers (**Object Data** tab in the Properties Editor). The first UV layer must be named `diffBricks` and the second `bake`; make sure to leave the first one selected.
2. Now switch to Edit Mode, then switch to the Face select mode and deselect any face that could be selected now (`A` key).
3. Next, go to the **Material** tab in the Properties Editor, select the `bricks` material slot from the materials list, and click on the **Select** button just below it. That will leave us with the faces on which this material is assigned selected.
4. Now we need to deselect the faces that are part of the sides, the back, and the top of the building. The easy way to do this is to switch to Right View (**View** → **Right**), initiate the Border Select tool (**Select** → **Border Select**), and draw a rectangle enclosing the faces that must be deselected, using the middle mouse button. Then go to the 3D View and use the Region to Loop tool (**Mesh** → **Edges** → **Region to Loop**) to leave selected only the edges on the border of the whole region.
5. Now let's use the Mark Seam tool to tag the edges (**Mesh** → **Edges** → **Mark Seam**).
6. The same series of steps (selecting a "side" of the building, then using the Region to Loop command, and tagging the selected edges) must be applied to each side, the top, and the rear parts of the building.

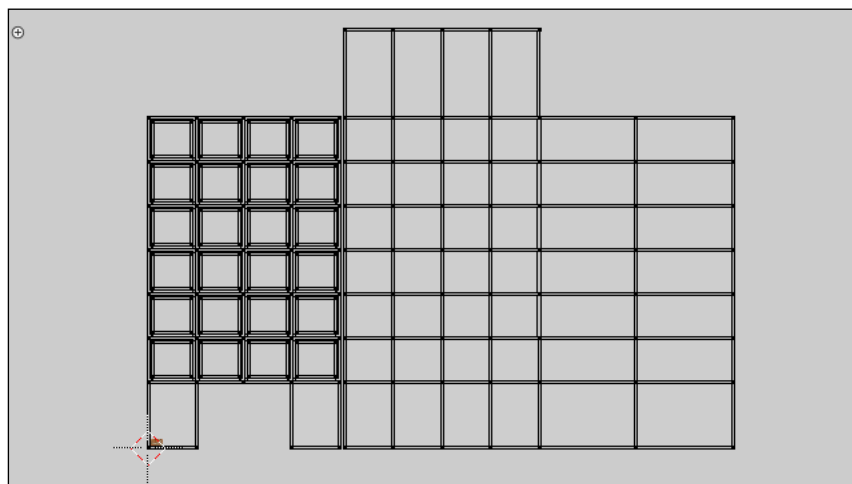
Once the seams for the faces of the building that use the bricks material are done, let's create some more seams that we will need later.
7. Switch to the Face select mode (`Ctrl + Tab`, then press `3`) and deselect all of them (**Select** → **Select/Deselect All**). Then go to the **Material** tab in the Properties Editor, select the frames material, and click on the **Select** button to select only the faces that have this material assigned.
8. Right after getting the faces selected, let's just go to the 3D View and use the Region to Loop command (**Mesh** → **Edges** → **Region to Loop**) to get the borders of the frames selected easily.
9. Finally, let's just tag the edges by going to **Mesh** → **Edges** → **Mark Seam**. Let's also leave everything deselected for now.

10. Now we only need to switch to the Face select mode and select the faces to which the `bricks` material is assigned by selecting the material from the materials list and then clicking on **Select**.
11. Then go to the UV/Image Editor and select the `bricks.jpg` image from the list of available image datablocks. At this point, let's make sure that the active UV layer is the one named `diffBricks`.
12. With the image selected, we just need to go to the 3D View and perform a UV Unwrap (**Mesh** → **UV Unwrap...** → **Unwrap**) to get a UV layout created automatically for us.
13. Now let's go to the UV/Image Editor, select all the vertices of the UV layout, rotate them 90 degrees counterclockwise (*R* key, then type `-90`), and scale them up by a factor of 20 (*S* key, then type `20`).



It's very likely that the island of the UV layout that corresponds to the front of the building will be a bit distorted, causing the mapping to look weird on the model. To correct that, let's just select those faces in the 3D View, switch to Front View (**View** → **Front**), and then perform an unwrap from view (**Mesh** → **UV Unwrap...** → **Project From View**). Then we just need to select the correctly unwrapped island and scale it up to match the scale of the other islands.

The following screenshot shows the finished UV layout for this material:



As with the previous objects, we only need one more texture: The baked result to use as a lightmap.

14. Let's go to Gimp and create a new image with 1024 x 1024 pixel resolution and save it as `bake-building.tga` in the `tex` folder of the project; remember to close the image before going back to Blender.
15. Let's go back to Blender, activate the `bake` UV layer for the building object (**Object Data** tab in the Properties Editor), and load the image that will be used to bake the lightmap for the building by going to the UV/Image Editor; then go to **Image** → **Open**, browse to the `tex` folder, select the image `bake-building.tga`, and click on the **Open** button (top right).
16. Once the image is loaded in the UV/Image Editor viewport, let's go to the 3D View, select all the faces of the building object, and perform an unwrap (**Mesh** → **UV Unwrap...** → **Unwrap**), remember to set the **Island Margin** parameter to `0.06` (**Tool Shelf** sidebar).
17. Now let's take some time to inspect the resulting UV layout, paying special attention to the faces of the layout that are too thin and scaling them up enough to cover some pixels in the image. Take into account that the proportions of a face in the layout don't have to be the same as the proportions of the corresponding face in the 3D View. We can just scale the thin faces of the layout in one direction, X or Y.
18. The only step remaining to get the bake done is going to the **Render** tab of the Properties Editor, then to the **Bake** panel and clicking on the **Bake** button (make sure the **Bake Mode** is set to **Full Render**). This time, the bake could take a few seconds more than before, since the image is the biggest one we have used for baking. Notice that the resulting bake is not only a grayscale image; it now has some blue spots, which correspond to the bake for the windows material.
19. Let's now save the baked image over the original file on disk (**Image** → **Save As** in the UV/Image Editor) and switch to Gimp in order to perform a couple of tweaks to the bake.

We now need to open the baked image in Gimp and then apply the two usual correction steps to get a better bake.
20. First, let's apply the Selective Gaussian Blur (**Filters** → **Blur** → **Selective Gaussian Blur**) with a **Blur radius** of `10` and a **Max delta** of `100`. We must then apply the curves tool (**Color** → **Curves**) using the shape shown earlier in this chapter (while applying color adjustments to the grass, the screenshot showing the settings for this tool). In the previous bakes, these two steps were enough; this time, though, we need to perform an extra correction, since the curves tool made the blue color to look very saturated. For this, let's open the **Hue-Saturation** tool (**Colors** → **Hue-Saturation**), set the **Saturation** value to `-30`, and apply the tool. Now the blue color of the windows looks more natural, which is what we wanted. Save the image in Gimp and also close it (leave Gimp running).

Back in Blender, we only need to set up the materials for the building to finish this part of the project.

21. Let's go to the **Material** tab in the Properties Editor, select the `bricks` material, and set it as follows:
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (Diffuse panel)
 - ▶ Shadeless: Enabled (Shading panel)
22. Then let's go to the **Texture** tab, add a new texture, set its name as `bricksDiff`, its type as **Image or Movie**, and select the `Bricks.jpg` image from the drop-down list in the **Image** panel; then set it as follows:
 - ▶ Mapping panel:
 - Coordinates: UV
 - Layer: `diffBricks`
 - ▶ Influence panel:
 - Color: Enabled
 - Blend: Add
23. Now let's add a second texture datablock, name it `bricksBake`, set its type as **Image or Movie**, and select the `bake-building.tga` image from the drop-down list in the **Image** panel. Set it as follows:
 - ▶ Mapping panel:
 - Coordinates: UV
 - Layer: `bake`
 - ▶ Influence panel:
 - Color: Enabled
 - Blend: Multiply
24. Then go to the **Material** tab, select the `frames` material, and enable the **Shadeless** option for it (Shading panel). This material will not use image textures, since it's entirely black.
25. The last step is setting the `windows` material. Select it from the materials list in the **Material** tab, then set its diffuse color to fully black, and enable the Shadeless option (Shading panel).
26. Then go to the **Texture** tab, add a new texture, change its name to `windowsBake`, set its type to **Image or Movie**, select the `bake-building.tga` image from the drop-down list in the **Image** panel, and set it as follows:
 - ▶ Mapping panel:
 - Coordinates: UV

- ❑ Layer: bake
- ▶ Influence panel:
 - ❑ Color: Enabled
 - ❑ Blend: Add

That's it for the shading part, now we can take a few minutes to navigate our completed scene and look at the nice texturing, including the very beautiful, seamless skydome. Don't forget to save the file.

Objective Complete - Mini Debriefing

This time we worked on the main object of the scene, applying the very same technique as in the previous task of the project. The only new things we found were the added complexity of the UV Layout, the (very likely) situation of having a distorted UV layout island that needed some work to correct it, and having to work with more than one material for a single object.

Classified Intel

Given the number of times that we applied the very same technique in these tasks of the project, it's very likely that you can just close your eyes and repeat the steps from memory without a single mistake. This is typical in cases where the final "deployment medium" isn't the "native" one for the tool that we work with. Let's further explain it to avoid any confusion: The Blender Game Engine is tightly coupled with Blender as a whole, even though there are features of Blender that can't be used in the BGE. That said, the overall point is that the rendering of the scenes, when in the context of a game, is performed by a graphics card, thus becoming a "non-native deployment medium" from the 3D authoring tool (which is primarily intended to be used for offline rendering). Of course, not everything is bad, since a graphics card is able to deliver a full-frame render in a matter of milliseconds!

Finally, let's talk a bit about the different sizes for the images used as canvas to bake onto. The reason can be summarized in just one word: Optimization. If we take a look at the soft shadows that objects have around them in real life, we will notice that those shadows don't have fine details (small variations). Knowing that, we just use small images to save some memory, and the graphics card performs some filtering on them when used to cover a bigger area on screen than the normal size of the image. It all happens without the shadows looking "pixelated" and, even better, without the shadow losing the main element for its identity: Softness. In the end, the trick is twofold: Tileable images to be able to cover big areas with small resource consumption, and small lightmaps to get a good final appearance, also preserving resources as much as possible.

Setting Up the Objects for the Game Engine

Up until now, we've been working exclusively on the commonly known aspects of a 3D project, namely the modeling and shading parts. This project, though, has a new component: We want to create an interactive walkthrough. We decided to use the Blender Game Engine for that. Now, the nice thing of using the BGE is that we don't have to go jumping through hoops to get the models and shaders from the authoring software to the game engine. The part that we definitely cannot get for free is the creation of the logic of the "game", though Blender offers us the very useful abstraction of logic bricks to avoid having to write too much code for it.

Engage Thrusters

In this part we will perform two sub-tasks: Setting up the 3D objects needed for the logic system that we are going to create and defining the logic bricks that will govern the flow of the logic; the scripting will be left for the next task. Let's work on it.

Objects Setup

1. Let's start by switching to Object Mode (if we are not in it), adding an empty object (*Shift + A*), setting its location to 0 X, -18 Y, 0 Z (**Transform** panel, **Properties** sidebar), and changing its name to `base`.
2. With the `base` object selected, let's create a duplicate (**Object** → **Duplicate** or press *Shift + D*) and move it five units in the positive direction of the Z axis (the G key, Z key, then type 5). Then change its name to `derived`.
3. Again select the `base` object, duplicate it, and move it three units in the negative direction of the Y axis (the G key, Y key, then type -3). Let's set the name of this duplicate to `mainControl`.
4. Now let's select the camera of the scene (or add one if there's no camera) and set its transform properties (**Properties** sidebar) as follows:
 - ▶ Location: 0 X, -18 Y, 5 Z
 - ▶ Rotation: 90° X, 0° Y, 0° Z

To finish this part, let's perform the parenting needed to get the object setup done.

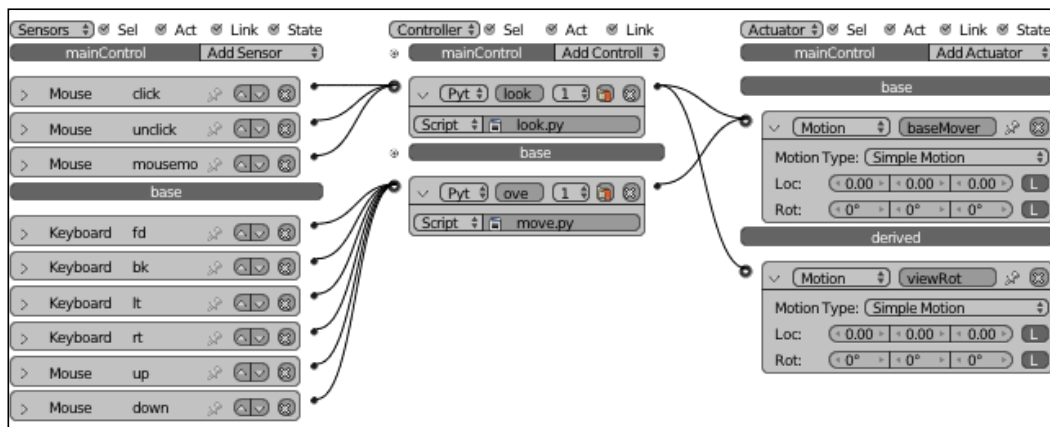
5. Select the camera, then the `derived` object, and then go to **Object** → **Parent** → **Set** (*Ctrl + P*); that way, the `derived` object will be the immediate parent of the camera object.

6. Now, we need to parent the derived object to the base object. To do that, let's select the derived object, then the base object, and then go to **Object** → **Parent** → **Set**. This way we get the parenting hierarchy complete.

The reason to organize these four objects in that specific way is because of the kind of mechanism we are going to create. The base object will serve the purpose of moving around in space and also rotating on its Z axis (thus allowing the camera to look to the left or to the right). The derived object will then inherit the location and Z rotation of the base object, but it will be rotated on its X axis to allow the camera to look up and down. That's the way the mechanism works, by splitting the move and rotate tasks between different objects in the setup. The `mainControl` object was added only to hold certain logic, but it doesn't intervene physically in the mechanism. A good way to visualize the way the mechanism works is by rotating the base object on its Z axis and then the derived object on its X axis, to see how the camera gets oriented around in each case. Remember though, to clear the rotations (*Alt + R*) of the base and derived objects before continuing.

Logic Bricks Setup

7. Let's now go to the UV/Image Editor and change the type of editor to a Logic Editor. But before starting to work on the setup, let's take a look at a screenshot showing it finished; that way we have some idea where we are heading:



What the screenshot shows are the logic bricks organized in three columns. The first column is for the *sensors*, the second for the *controllers*, and the third for the *actuators*. The whole idea of the system is this: Sensors can detect events (for example, a key press) and send activation signals to the controllers they are connected to when the event is detected. A controller is stimulated by the activation signal from one or more of the sensors connected to it and the controller's task is to decide if one or more of the actuators it is connected to must be sent an activation signal or not. Actuators just wait for an activation signal from a controller to perform some task (for example, moving an object, playing a sound, and so on).

-
8. Let's start creating the logic bricks setup for the base object; select it and go to the Logic Editor; then locate the **Add Sensor** drop-down and add six sensors as follows:

- ▶ Sensor 1:
 - ❑ Type: Keyboard
 - ❑ Name: fd
 - ❑ True level triggering: Enabled (second button to the left of "Freq")
 - ❑ Key: W
- ▶ Sensor 2:
 - ❑ Type: Keyboard
 - ❑ Name: bk
 - ❑ True level triggering: Enabled
 - ❑ Key: S
- ▶ Sensor 3:
 - ❑ Type: Keyboard
 - ❑ Name: lt
 - ❑ True level triggering: Enabled
 - ❑ Key: A
- ▶ Sensor 4:
 - ❑ Type: Keyboard
 - ❑ Name: rt
 - ❑ True level triggering: Enabled
 - ❑ Key: D
- ▶ Sensor 5:
 - ❑ Type: Mouse
 - ❑ Name: up
 - ❑ Mouse Event: Wheel Up
- ▶ Sensor 6:
 - ❑ Type: Mouse
 - ❑ Name: down
 - ❑ Mouse Event: Wheel Down

These six sensors will be the ones detecting when the user presses any of the keys used for the horizontal movement (WASD) or scrolls the mouse wheel (to move vertically).

9. Now let's go to the controllers column and add a new controller of type **Python Script**. For this controller, we only set its name for now; let's call it `move`. Remember the name of the controller (upper part of the logic brick) itself is distinct from the text datablock that will contain the actual code (lower part of the controller logic brick).
10. Finally, let's add an actuator (third column) of type **Motion** and change its name to `baseMover`. Let's also connect the output socket of each one of the six sensors of the base object to the input socket of the `move` controller, and connect the output socket of the `move` controller to the input socket of the `baseMover` actuator.

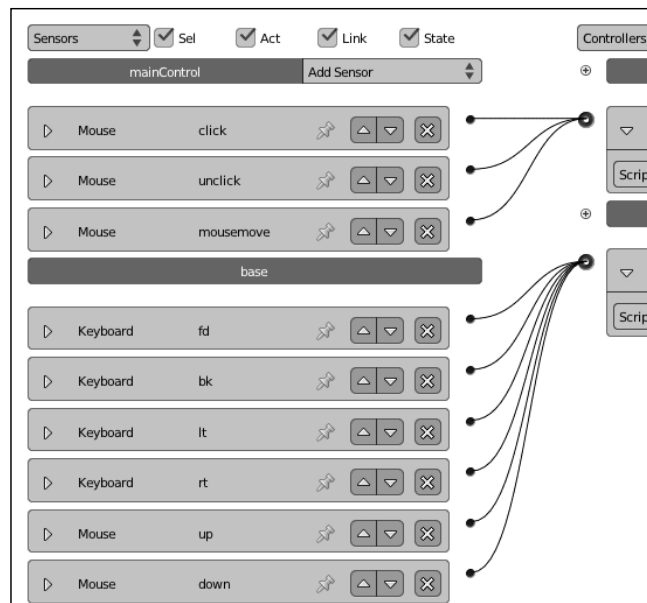
Apart from the logic bricks, we also need to add *properties* to the base object.

11. Let's put the mouse pointer inside the Logic Editor and press the *N* key to show the sidebar; then click on the **Add Game Property** button twice. The first property must be named `walkSpeed`, its type set to `Float` and its value set to `0.1`. The second property must be named `flySpeed`, its type set to `Float`, and its value set to `2.0`. Properties are just a way to give an object the responsibility to hold some data that we can use in the logic. In this case, this object is the one that knows how fast to walk (horizontal movement) and how fast to fly (vertical movement).
12. Now let's select the derived object in the 3D View and only add an actuator to it in the Logic Editor; the actuator must be of type **Motion** and its name must be set as `viewRot`.
13. Finally, let's select the `mainControl` object, then go to the Logic Editor and add three sensors as follows:
 - ▶ Sensor 1:
 - ❑ Type: Mouse
 - ❑ Name: click
 - ❑ Mouse Event: Left Button
 - ▶ Sensor 2:
 - ❑ Type: Mouse
 - ❑ Name: unclick
 - ❑ Invert: Enabled
 - ❑ Mouse Event: Left Button

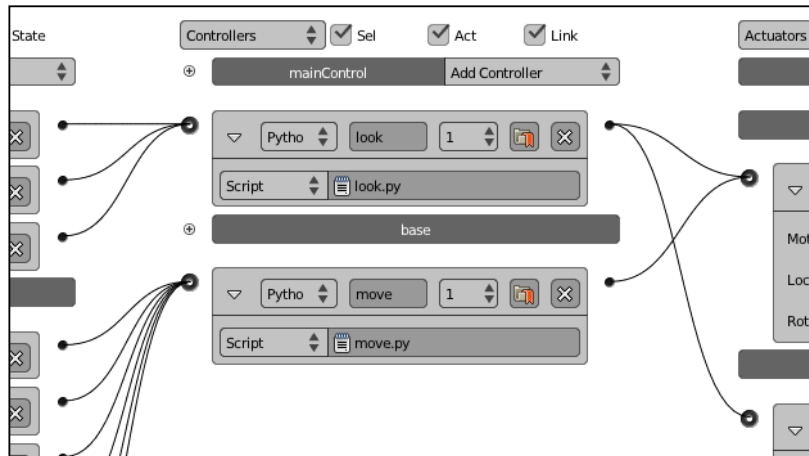
- Sensor 3:
 - ❑ Type: Mouse
 - ❑ Name: mousemove
 - ❑ Mouse Event: Movement

14. Then go to the second column, add a controller of type **Python Script**, and set its name as `look`.
15. The last thing needed to complete the logic bricks setup is to connect the three sensors of the `mainControl` object to the `look` controller and then connect the output of the `look` controller to the `baseMover` and `viewRot` actuators of the `base` and derived objects, respectively. Notice that it is possible to connect logic bricks between different objects if we select them in the 3D View, which is very useful.
16. The `mainControl` object must also have a couple of properties, so let's add two properties to it. The first must be named `hold`, its type set to `Boolean`, and the toggle button left unpressed (meaning `False`). The second property must be named `pointerSpeed`, its type set to `Float`, and its value set to `0.01`. The `hold` property will be used to know if the user has the mouse button clicked when the pointer moves (which means the user wants to look around) and the `pointerSpeed` is just used to control how sensitive the camera must be to mouse movements.

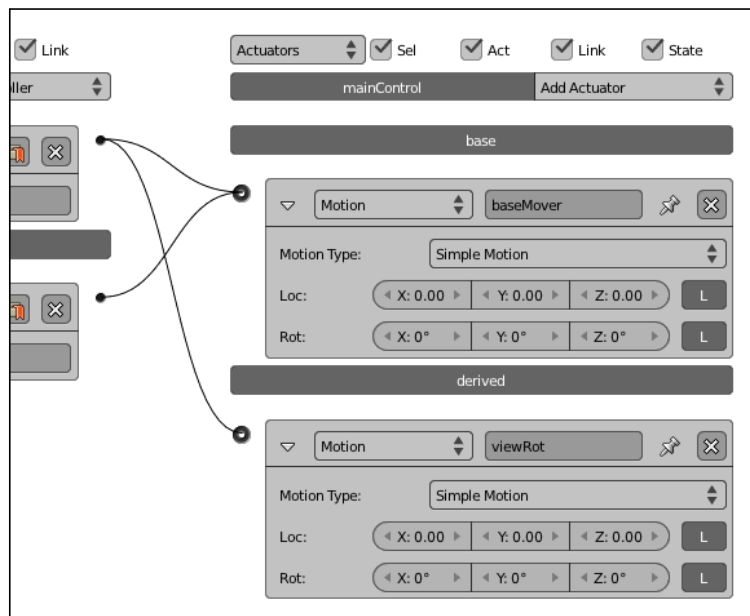
To wrap it up, let's take a look at the next three screenshots, detailing how the final setup must look:



The previous screenshot shows the three sensors added to the `mainControl` object and the six added to the `base` object.



Notice that the previous screenshot shows that both controllers have assigned their corresponding Python scripts, which is not doable yet (the scripts haven't been created). The creation and setup of the scripts will be done in the next task.



The previous screenshot just shows the two motion actuators used by the `base` and `derived` objects.

Objective Complete - Mini Debriefing

This task of the project was entirely dedicated to setting up the objects of our scene for the Blender Game Engine. We started by creating the setup of objects that will serve us as the primary mechanism to perform the navigation of the scene. One thing to highlight here is that by using these objects and the parenting relationship between them we are saving a lot of coding, since that behavior is already automatically handled by the BGE itself. An example of the kind of complexity that this helps us to avoid is that the rotations are very simple degree or radian values, while in reality the calculations that Blender performs internally to handle rotations involve matrices! Sorry for using exotic mathematical terms here, but there is no other way to explain it, really.

Then we created a setup of logic bricks, which helps us even more in hiding the real complexity involved in doing this kind of interactive project. The nice thing about logic bricks is that the BGE has already defined a whole internal mechanism for detection for events and passing of messages between sensors, controllers, and actuators, so that we can just start using all that by clicking around in the interface.

Scripting

The final task of our project is to work on coding two scripts that will significantly enhance the final behavior of the walkthrough. In this coding part, we will just transcribe the code listings provided; let's also pay special attention to the explanatory texts embedded in the code as comments (statements after the # sign).

Engage Thrusters

1. Let's go to the Logic Editor and change it to a Text Editor (*Shift + F11*), then add a new text datablock, name it `move.py`, and enter this code (the comments are optional):



Since the listings of code don't show some things (usage of tabs, spaces, and indentation) required explicitly for the Python interpreter to accept our code, it's highly recommended to use these listings only as a reference. The actual code should be taken from the files provided along with the book.

```
# make the functionality of these libraries available in the scope
of this script. "bge" is used 99.9% of the time in scripts for the
BGE
import bge
import math
import mathutils
```



```
# get a reference to the controller that called this script. This
line of code exists in 99.9% of the scripts for the bge, since the
controller reference is the starting point to access the sensors
and the actuators connected to it.
cont = bge.logic.getCurrentController()

# get a reference to the object that owns the controller that
called this script; in this project, that object will be "base"
obj = cont.owner

# get references to the sensors attached to the current
controller. These references are needed to ask the sensors for
extra information and act accordingly.
fd = cont.sensors['fd']
bk = cont.sensors['bk']
lt = cont.sensors['lt']
rt = cont.sensors['rt']
up = cont.sensors['up']
down = cont.sensors['down']

# get a reference to the actuator called "baseMover"
act = cont.actuators['baseMover']

# define default values for these variables, which will be used to
decide how much to move in each direction
yVal = 0
xVal = 0
zVal = 0

# if the fd sensor is sending the activation pulse...
if fd.positive:
    # ... add the value of the "walkSpeed" property to the yVal
    yVal += obj['walkSpeed']

# if the bk sensor is sending the activation pulse...
if bk.positive:
    # ... subtract the value of the "walkSpeed" property to the yVal
    yVal -= obj['walkSpeed']

# The next 4 ifs have the same logic than the previous 2, but
applied to different variables
if lt.positive:
    xVal -= obj['walkSpeed']

if rt.positive:
    xVal += obj['walkSpeed']

if up.positive:
    zVal += obj['flySpeed']

if down.positive:
    zVal -= obj['flySpeed']
```

```
# finally, set the dLoc vector of the baseMover actuator to the
values calculated in the previous lines of code...
act.dLoc = [xVal, yVal, zVal]
# ... and send the actuator the activation signal, so that it
performs the action of moving the object.
cont.activate(act)
```

2. Then let's add a second text datablock, set its name to `look.py`, and enter this code:

```
# ask for the functionality of these libraries to be available in
this script
import bge
import mathutils
import math

# create a function to improve the readability of the code below
def showMousePointer():
    bge.render.showMouse(True)

# create a function to improve the readability of the code below
def hideMousePointer():
    bge.render.showMouse(False)

# define two variables to be used as constants (all uppercase
is just a way to signify they are constants). These variables
indicate what is the maximum (up) and minimum (down) angle the
camera can rotate. The computation is needed to convert the value
from degrees to radians.
MAX_ANGLE = 60*math.pi/180
MIN_ANGLE = -MAX_ANGLE

# get a reference to the controller that called this script and to
the object owner of that controller
cont = bge.logic.getCurrentController()
obj = cont.owner

# get a reference to the three sensors connected to the controller
that called this script
click = cont.sensors['click']
unclick = cont.sensors['unclick']
move = cont.sensors['mousemove']

# get references to the actuators that this controller is
connected to
viewRot = cont.actuators['viewRot']
baseMover = cont.actuators['baseMover']

# get references to the objects that own the viewRot and baseMover
actuators. These references are needed to access the properties of
those objects
```

```
xRotator = viewRot.owner
zRotator = baseMover.owner

# read the current X rotation of the xRotator object ("derived"
object)
xCurrentRot = xRotator.orientation.to_euler().x

# if the current x rotation of the object is outside the allowed
range...
if xCurrentRot < MIN_ANGLE or xCurrentRot > MAX_ANGLE:
    # check to see if the excess is above or below and save the
appropriate max in refAngle
    if xCurrentRot > MAX_ANGLE: refAngle = MAX_ANGLE
    elif xCurrentRot < MIN_ANGLE: refAngle = MIN_ANGLE
    # calculate how much to rotate the object to get it back into
the allowed range of values
    deltaXCurrentRot = refAngle - xCurrentRot
    # apply the correction to the x component of the rotation
    xRotator.applyRotation([deltaXCurrentRot, 0, 0], 1)

# if the click sensor sent a positive pulse and it isn't being
held from a previous click...
if click.positive and not obj['hold']:
    # remember the click is being held down by setting the hold
property to True
    obj['hold'] = True
    # remember where in the screen the click happened. Notice
these two properties are being created on-the-fly in this script.
    obj['centerX'] = click.position[0]
    obj['centerY'] = click.position[1]
    # hide the mouse pointer
    hideMousePointer()

# in case the condition for the previous if wasn't evaluated to
"true", let's check if the sensor sending the positive pulse is
the unclick sensor, if so...
elif unclick.positive:
    # ... remember that the mouse button was released
    obj['hold'] = False
    # turn off the actuators that were rotating the camera in the
scene. If they are not explicitly deactivated, the camera could
keep moving after the user releases the mouse button
    cont.deactivate(viewRot)
    cont.deactivate(baseMover)
    # show the mouse pointer to indicate to the user the camera
isn't being moved
    showMousePointer()

# if the pointer is being moved and the mouse button is held down...
```

```

if move.positive and obj['hold']:
    # calculate how much the pointer moved from the place where
    the user clicked to start navigating, both in X and Y
    deltaX = (obj['centerX'] - move.position[0])*obj['pointerSpe
ed']
    deltaY = (obj['centerY'] - move.position[1])*obj['pointerSpe
ed']
    # reset the position of the pointer to the place where the
    user pressed down the mouse button
    bge.render.setMousePosition(obj['centerX'], obj['centerY'])

    # in case the vertical movement of the pointer will cause the
    camera to rotate outside of the allowed range...
    if (xCurrentRot + deltaY*math.pi/180 > MAX_ANGLE) or
    (xCurrentRot + deltaY*math.pi/180 < MIN_ANGLE):
        # ... set the vertical movement to 0, to ignore it
        deltaY = 0

    # set the vectors for the rotation of the viewRot ("derived")
    and the baseMover ("base")
    viewRot.dRot = [deltaY, 0, 0] #deltaY is used to drive the X
    rotation of the "derived" object
    baseMover.dRot = [0, 0, deltaX] # deltaX is used to drive the
    Z rotation of the "base" object
    # turn on the two actuators, so that they perform the rotation
    given by the two vectors calculated right above
    cont.activate(viewRot)
    cont.activate(baseMover)

```

Let's save the file now, to avoid losing what we have done since the last save.

We just need to perform some small tasks to get the project finished.

3. First, let's switch to the Logic Editor (*Shift + F2*), locate the `move` controller, and click on the field with the notepad icon to select the `move.py` entry; then go to the look controller and select the `look.py` script from the list of available scripts.
4. Second, let's go to the 3D View and switch to Camera View (**View → Camera**) to see if the skydome is getting clipped in the distance. If the clipping happens, we just need to select the camera, then go to the **Object Data** tab of the Properties Editor, and set the **Clipping End** (in the **Lens** panel) to a higher value.

Now, the time has come; after all that hard work, our interactive walkthrough is ready to be used. Let's make sure we are in Camera View; press *P* and start navigating around the scene. Isn't it nice how we could do almost everything right inside Blender?

Objective Complete - Mini Debriefing

In this final task, we had to add a bit of sauce to the project by writing a couple of Python scripts. These scripts serve the purpose of allowing us to define the fine details of the behavior of our "game". Thankfully for us, the scripts were very simple in structure and quite intuitive in the logical flow they have.

Classified Intel

This task comes with the added complexity related to coding. When we start working on a programming language, we need to understand two basic elements of it: the language itself and the "predefined vocabulary" of that language.

Some aspects that are directly related to the Python programming language (not used only in the Blender Game Engine) are the keywords (such as "def" or "if") that constitute an integral part of the language itself; other elements of the language are the indentations of code, which define the "hierarchy" between the parts of it. These elements are found in any code written in the Python programming language, for example in a web application created with it.

The "predefined vocabulary" of a language is usually called libraries or packages and is tightly bound to the environment in which the code is being executed. For example, the code that we wrote is targeted specifically at the Blender Game Engine, so it makes use of libraries that only make sense within the context of the Blender Game Engine, such as the **bge library** (made available for our scripts by using the `import` keyword). If we take our Python code and try to run it outside the realm of the BGE, it will (almost for sure) not work, because the bge library exists only within the BGE. A very important thing to understand about libraries is that the Python language itself comes bundled with a very comprehensive set of libraries (such as `math`), which will not (necessarily) require a specific context (or realm, or platform) to be available for the programmer.

Mission Accomplished

That's it. A simple walkthrough from start to finish, done almost entirely within Blender. This project shows quite a few interesting techniques: We learned how to create a seamless skydome by baking a special kind of sky map into a texture. For the objects of the scene, we learned how to perform basic UV unwrapping and how to layer different textures to save computer resources while still getting a nice end result. One of the useful tricks we learned is to use the Selective Gaussian Blur in Gimp to soften the lightmaps baked in Blender. The lightmaps are noisy because we didn't set the Ambient Occlusion quality to a high value, to avoid long bake times. Finally, we got a good hang of what the Blender Game Engine offers us to play with, from logic bricks, to properties, and a little bit of scripting. All that together allows us to have a very nice interactive walkthrough.

Let's take a look at a nice screenshot of our interactive walkthrough:



You Ready To Go Gung HO? A Hotshot Challenge

There are a few more ways to improve the end result of this project. A few suggestions are:

- ▶ We could create a more complex model of the building, the floor, and the fence. Also, adding different kinds of trees and scattering them around the grass would help.
- ▶ Setting some objects in the scene to be "actors" would allow us to use the physics simulator to get a more realistic feeling in the navigation, including the possibility of avoiding the camera passing through objects. Tip: When using physics, don't use the `dLoc` property of the motion actuator, instead use `linV`.
- ▶ The baked textures are quite simple, but we can use the texture painting functionality in Blender to add some random variation to the lightmaps, thus improving the final look.
- ▶ The navigation mode can be greatly improved by using the orientation of the camera instead of just moving along the local Y axis of the `base` object. Notice, though, that this would certainly require big changes from what we did here.
- ▶ Definitely, the ultimate challenge would be to create a bigger scene, having more than one building, to the point that it would look much more like a real video game, and including all the previous ideas for improvement.

Project 6

Detailed Render of the Earth from Space

Home sweet home. We've got a very beautiful planet to live in, with lots of beautiful landscapes to look at; but maybe the most beautiful ones are those that we can see in the pictures taken from the space. Those are really awesome. Let's challenge ourselves and work to create an image resembling some of that beauty that we can see in nature.

Mission Briefing

Our purpose is to create a very detailed view of the earth from space. By detailed, we mean that it includes land, oceans, and clouds, and not only the color and specular reflection, but also the roughness they seem to have, when seen from space.

For this project, we are going to perform some work with textures and get them properly set up for our needs (and also for Blender's way of working).

What Does It Do?

We will create a nice image of the earth resembling the beautiful pictures that are taken from orbiting of the earth, showing the sun rising over the rim of the planet. For this, we will need to work carefully with some textures, set up a basic scene, and create a fairly complex setup of nodes for compositing the final result.

In our final image, we will get very nice effects, such as the volumetric effect of the atmosphere that we can see round its rim, the strong highlight of the sun when rising over the rim of the earth, and the very calm, bluish look of the dark part of the earth when lit by the moon.

Why Is It Awesome?

With this project, we are going to understand how important it is to have good textures to work with. Having the right textures for the job saves lots of time when producing a high-quality rendered image.

Not only are we going to work with some very good textures that are freely available on the Internet, but we are also going to perform some hand tweaking to get them tuned exactly as we need them. This way we can also learn how much time can be saved by just doing some preprocessing on the textures to create finalized maps that will be fed directly to the material, without having to resort to complex tricks that would only cause us headaches.

One of the nicest aspects of this project is that we are going to see how far we take a very simple scene by using the compositor in Blender. We are definitely going to learn some useful tricks for compositing.

Your Hotshot Objectives

This project will be tackled in five parts:

1. Preprocessing the textures
2. Object setup
3. Lighting setup
4. Compositing preparation
5. Compositing

Mission Checklist

The very key for the success of our project is getting the right set of quality images at a sufficiently high resolution. Let's go to www.archive.org and search for www.oera.net/How2.htm on the 'wayback machine'. Choose the snapshot from the **Apr 18, 2008** link. Click on the image titled **Texture maps of the Earth and Planets**. Once there, let's download these images:

- ▶ Earth texture natural colors
- ▶ Earth clouds
- ▶ Earth elevation/bump
- ▶ Earth water/land mask

Remember to save the high-resolution version of the images, and put them in the `tex` folder, inside the project's main folder.

We will also need to use Gimp to perform the preprocessing of the textures, so let's make sure to have it installed. We'll be working with version 2.6.

Preprocessing the Textures

The textures we downloaded are quite good, both in resolution and in the way they clearly separate each aspect of the shading of the earth. There is a catch though—using the clouds, elevation, and water/land textures as they are will cause us a lot of headache inside Blender. So let's perform some better basic preprocessing to get finalized and separated maps for each channel of the shader that will be created.

Engage Thrusters

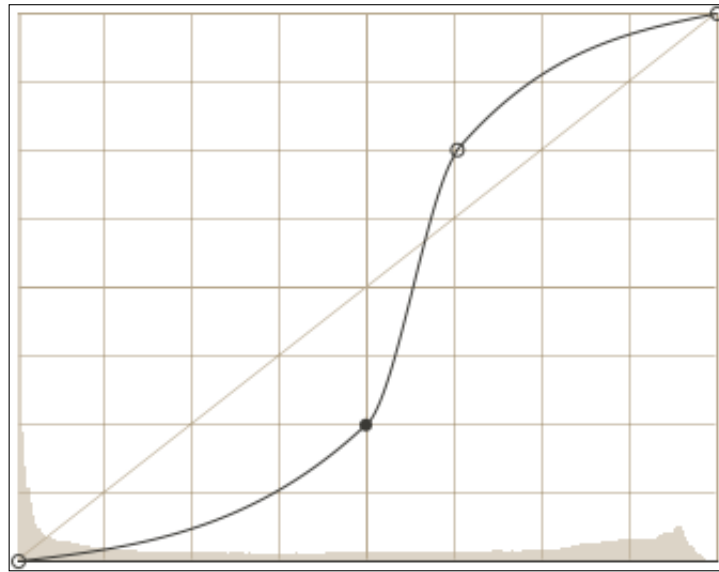
For each one of the textures that we're going to work on, let's make sure to get the previous one closed to avoid mixing the wrong textures.

Clouds Map

1. Drag the `EarthClouds_2500x1250.jpg` image from the `tex` folder into the empty window of Gimp to get it loaded.
2. Now locate the **Layers** window and right-click on the thumbnail of the **Background** layer, and select the entry labeled **Add Layer Mask...** from the menu. In the dialog box, select the **Grayscale copy of layer** option. Once the mask is added to the layer, the black part of the texture should look transparent.

If we take a look at the image after adding the mask, we'll notice the clouds seem to have too much transparency. To solve this, we will perform some adjustment directly on the mask of the layer.

3. Go to the **Layers** window and click on the thumbnail of the mask (the one to the right-hand side) to make it active (its border should become white). Then go to the main window (the one containing the image) and go to **Colors | Curves...** In the **Adjust Color Curves** dialog, add two control points and get the curve shown in the next screenshot:



The purpose of this curve is to get the light gray pixels of the mask to become lighter and the dark ones to get darker; the strong slope between the two control points will cause the border of the mask to be sharper.

4. Make sure that the **Value** channel is selected and click on **OK**.
Now let's take a look at the image and see how strong the contrast of the image is and how well defined the clouds are now.
5. Finally, let's go to **Image** → **Mode** → **RGB** to set the internal data format for the image to a safe format (thus avoiding the risk of having Blender confused by it). Now we only need to go to **File** → **Save A Copy...** and save it as `EarthClouds.png` in the `tex` folder of the project. In the dialogs asking for confirmation, make sure to tell Gimp to apply the layer mask (click on **Export** in the first dialog). For the settings of the PNG file, we can use the default values. Let's close the current image in Gimp and get the main window empty in order to start working on the next texture.

Specular Map

6. Let's start by dragging the image named `EarthMask_2500x1250.jpg` onto the main window of Gimp to get it open. Then drag the image `EarthClouds_2500x1250.jpg` over the previous one to get it added as a separate layer in Gimp.
7. Now, we need to make sure that the images are correctly aligned. To do this, let's go to **View** → **Zoom** → **4:1 (400%)**, to be able to move the layer with pixel precision easily. Now go to the bottom right-hand side corner of the window and click-and-drag over the four-arrows icon until the part of the image shown in the viewport is one of the corners. After looking at the right place, let's go to the **Toolbox** and activate the **Move** tool. Finally, we just need to drag the clouds layer so that its corner exactly matches the corner of the water/land image. Then let's switch to another zoom level by going to **View** → **Zoom** → **1:4 (25%)**.
8. Now let's go to the **Layers** window, select the **EarthClouds** layer, and set its blending mode to **Multiply (Mode drop-down, above the layers list)**.
9. Now we just need to go to the main window and go to **Colors** → **Invert**.
10. Finally, let's switch the image to RGB mode by going to **Image** → **Mode** → **RGB** and we are done with the processing. Remember to save the image as `EarthSpecMap.jpg` in the `tex` folder of the project and close it in Gimp.

The purpose of creating this specular map is to correctly mix the specularity of the ocean (full) with one of the clouds that is above the ocean (null). This way, we get a correct specularity, both in the ocean and in the clouds. If we just used the water or land mask to control specularity, then the clouds above the ocean would have specular reflection, which is wrong.

Bump Map

The **bump map** controls the roughness of the material; this one is very important as it adds a lot of detail to the final render without having to create actual geometry to represent it.

11. First, drag the `EarthElevation_2500x1250.jpg` to the main window of Gimp to get it open. Then let's drag the `EarthClouds_2500x1250.jpg` image over the previous one, so that it gets loaded as a layer above the first one.
12. Now zoom in by going to **View** → **Zoom** → **4:1 (400%)**. Drag the image so that you are able to see one of its corners and use the move tool to get the clouds layer exactly matching the elevation layer. Then switch back to a wider view by going to **View** → **Zoom** → **1:4 (25%)**.
13. Now it's time to add a mask to the clouds layer. Right-click on the clouds layer and select the **Add Layer Mask...** entry from the menu. Then select the **Grayscale copy of layer** option in the dialog box and click **Add**.

What we have thus far is a map that defines how intense the roughness of the surface in each point will be. But there's a problem: The clouds are as bright as or even brighter than the Andes and the Himalayas, which means the render process will distort them quite a lot. Since we know that the intensity of the roughness on the clouds must be less, let's perform another step to get the map corrected accordingly.

14. Let's select the left thumbnail of the clouds layer (color channel of the layer), then go to the main window and open the color levels using the **Levels** tool by going to **Colors** → **Levels...** In the **Output Levels** part of the dialog box, let's change the value **255** (on the right-hand side) to **66** and then click on **OK**.

Now we have a map that clearly gives a stronger value to the highest mounts on earth than to the clouds, which is exactly what we needed.

15. Finally, we just need to change the image mode to RGB (**Image** → **Mode** → **RGB**) and save it as `EarthBumpMap.jpg` in the `tex` folder of the project.

Notice that we are mixing the bump maps of the clouds and the mountains. The reason for this is that working with separate bump maps will get us into a very tricky situation when working inside Blender; definitely, working with a single bump map is way easier than trying to mix two or more. Now we can close Gimp, since we will work exclusively within Blender from now on.

Objective Complete - Mini Debriefing

This part of the project was just a preparation of the textures. We must create these new textures for three reasons:

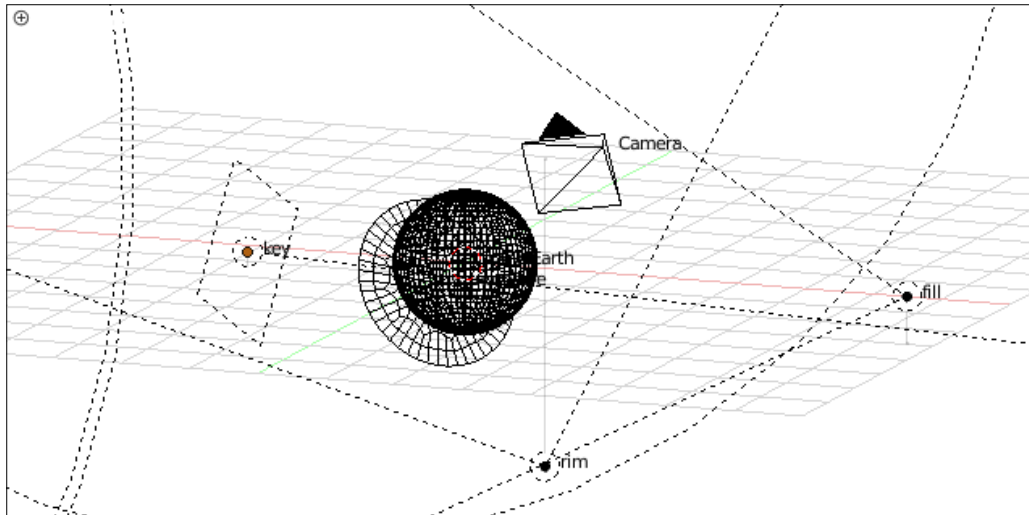
- ▶ To get the clouds' texture having a proper alpha channel; this will save us trouble when working with it in Blender.
- ▶ To control the spec map properly, in the regions where there are clouds, as the clouds must not have specular reflection.
- ▶ To create a single, unified bump map for the whole planet. This will save us lots of trouble when controlling the Normal channel of the material in Blender.



Notice that we are using the term "bump map" to refer to a texture that will be used to control the "normal" channel of the material. The reason to not call it "normal map" is because a normal map is a special kind of texture that isn't coded in grayscale, like our current texture.

Object Setup

For this project, we are going to need a very simple scene—two objects, a camera, and a very basic lighting setup. The real magic will be done by using the compositor. To guide our steps, let's take a look at a screenshot showing the finished scene:



The previous screenshot clearly shows the two objects (a sphere and a disc), the camera, and the three lights used for the scene. In this task, we will work on setting up the two objects.

Prepare for Lift Off

Let's open Blender, select and delete the default cube and light (keep the camera), and then save the file as `earth.blend` in the project's working directory.

Engage Thrusters

1. Start by making sure that the 3D Cursor is located at the origin of the 3D world by going to **View** → **Align View** → **Center Cursor and View All** (or by pressing *Shift + C*).
2. Now let's add a **UV Sphere** (**Add** → **Mesh** → **UV Sphere**) and leave the settings at their default values. Now go to the **Object** tab of the **Properties Editor** and set some properties as follows:
 - ❑ Name: planetEarth (first text entry)
 - ❑ Z Rotation: -148.85° (**Transform** panel)

- ❑ Layers: 1 (disable any other layer, **Relations** panel)
- 3. To get a softer appearance for the sphere, let's set its shading to **Smooth** (Tool Shelf sidebar) and add a **Subdivision Surface** modifier (**Modifiers** tab of the **Properties Editor**); make sure the **Subdivisions Level** for **Render** is 2.
- 4. With the basic settings for the object in place, let's go to the **Material** tab of the **Properties Editor**, add a new material, set its name to **earth**, and set it as follows:
 - ▶ Diffuse color: 0.0 Red, 0.0 Green, 0.0 Blue (**Diffuse** panel)
 - ▶ Specular panel:
 - ❑ Color: 1.0 Red, 0.95 Green, 0.88 Blue
 - ❑ Specular Shader Model: Blinn (drop-down menu)
 - ❑ Intensity: 0.0
 - ❑ Hardness: 40
 - ❑ IOR: 10.0

This will give us a basic material to play with. Notice that the diffuse color was set to black (zero), which means that we are going to add the color from the diffuse texture. The same logic will be applied to the specular intensity; we set it to 0 in the material level to drive it by using a texture in the additive blend mode.

- 5. Now let's go for the texture setup. Go to the **Texture** tab, add a new texture, change its name to 'diff', its type to **Image or Movie**, and set it as follows:
 - ▶ Mapping panel:
 - ❑ Projection: Sphere
 - ❑ Offset: 1.5 X
 - ▶ Image panel:
 - ❑ Load the image `EarthMap_2500x1250.jpg` from the `tex` folder
 - ▶ Influence panel:
 - ❑ Diffuse color: Enabled, 1.0
 - ❑ Blend mode: Add

We just perform the normal sphere mapping of the texture and add its color values to the diffuse color channel. The offset in the mapping is used to get the preview showing a part of the map that will allow us to easily check that the textures are working correctly.

-
6. The second texture must be called 'spec', its **Type** set to **Image or Movie**, and we can set it as follows:

- ▶ Mapping panel:
 - ❑ Projection: Sphere
 - ❑ Offset: 1.5 X
- ▶ Image panel:
 - ❑ Load the image `EarthSpecMap.jpg`
- ▶ Influence panel:
 - ❑ Specular Intensity: enabled, 1.0
 - ❑ Blend: Add
- ▶ Image Sampling panel:
 - ❑ Use Alpha: Disabled

This time we use one of the textures created by properly mixing the original ones. The only difference with the previous settings is that this texture will affect the specular intensity, and that we also disabled the use alpha option, since keeping it enabled would cause Blender to apply it incorrectly. Let's take a look at the preview panel (make sure to view it using a sphere) and see how the ocean has the correct specular intensity applied, whilst the earth doesn't have specular reflection at all.

7. The third texture must be named 'clouds' and its **Type** set to **Image or Movie**; the remaining settings are as follows:

- ▶ Mapping panel:
 - ❑ Projection: Sphere
 - ❑ Offset: 1.5 X
- ▶ Image panel:
 - ❑ Load the image `EarthClouds.png`
 - ❑ Premultiply: Enabled
- ▶ Influence panel:
 - ❑ Diffuse color: Enabled, 1.0
 - ❑ Blend mode: Mix

- ▶ Use Alpha: Enabled (**Image Sampling** panel)

The clouds texture is also straightforward. As we have an image with a proper alpha channel, we just need to enable its use and everything works correctly. Notice the usage of the **Premultiply** option in the **Image** panel. This is commonly needed for PNG images to be handled correctly by Blender.

8. The fourth and last texture for this material must be named 'bump', its **Type** set to **Image or Movie**, and its other settings made as follows:
 - ▶ Mapping panel:
 - Projection: Sphere
 - Offset: 1.5 X
 - ▶ Image panel:
 - Load the image `EarthBumpMap.jpg`
 - ▶ Influence panel:
 - Normal: Enabled, -0.1
 - Blend: Mix

As we can see, the bump map texture is applied to the Normal channel of the material. Notice that we use a negative value to get the correct effect.

That's it for the **planetEarth** object. As we have very good quality textures, we just need to map them properly without having to create complicated setups. Now is a good time to save the file.

Once done with the planet, let's create the disk that will allow us to create the atmospheric effect.

9. Go to the 3D View and add a new circle (**Add** → **Mesh** → **Circle**; remember to be in **Object Mode**). In the settings for the circle (Tool Shelf, lower part), let's set the vertices value to 12 and enable the fill option to get a disc. Now let's go to the **Object** tab in the **Properties Editor** and set it as follows:
 - ▶ Name: atmosphere
 - ▶ Transform panel:
 - Location: -0.597 X, 0.488 Y, -0.429 Z
 - Rotation: 61.744° X, 0.054° Y, 51.069° Z
 - Scale: 1.256 X, 1.256 Y, 1.256 Z

- ▶ Relations panel:
 - ❑ Layers: 2 (unset any other layer)
 - ❑ Pass index: 1
- 10. We just need to set one more thing for this object to get it ready: Go to the **Modifiers** tab in the **Properties Editor**, add a **Subdivision Surface** modifier, and set its **Render Subdivisions Level** to 3. That's all we need for this object.
- 11. Let's now set up the camera properly; select it, then go to the **Object** tab in the **Properties Editor** and set it as follows:
 - ▶ Transform panel:
 - ❑ Location: 3.041 X, -2.669 Y, 2.351 Z
 - ❑ Rotation: 60.129° X, 0° Y, 48.157° Z

Now let's switch to **Camera View (View → Camera)** to take a look at the planetEarth and atmosphere objects.

Objective Complete - Mini Debriefing

In this task, we worked exclusively on setting up the objects for our scene. Even though we have only two objects, setting them up required some work, as we need detailed material.

The more complex object in our scene is just a basic UV Sphere with a material having textures for various channels—diffuse, specular, and normal. Now we see how useful it was to perform the preprocessing on the textures. The set up of the material was very straightforward, sparing us the trouble of playing around with exotic tricks.

Lighting Setup

We'll be using a very basic lighting setup, using the classic technique with three lights—key, fill, and rim. For the key light, we'll be using an Area lamp to take advantage of the softness of its 'light'. The fill and rim lights will be just spot lights used because of the cone shape their 'lights' conform to, allowing us to focus directly on the main sphere.

Engage Thrusters

1. Let's start by adding an **Area Lamp (Add → Lamp → Area)**, then go to the **Object** tab in the **Properties Editor** and set it as follows:
 - ▶ Name: key
 - ▶ Transform panel:
 - ❑ Location: -2.991 X, -0.533 Y, 0.024 Z

- ❑ Rotation: -90.1° X, 30.604° Y, 80.551° Z
 - ▶ Layers: 1 (disable any other layer) (**Relations** panel)
- 2. After setting the **Object** properties of the key lamp, let's go to the **Object Data** tab of the **Properties Editor** and set the lamp properties as follows:
 - ▶ Lamp panel:
 - ❑ Color: 1.0 Red, 0.85 Green, 0.63 Blue
 - ❑ Energy: 0.1
 - ❑ Distance: 30.0
 - ❑ Gamma: 1.0
 - ❑ This Layer Only: Enabled
 - ❑ Specular: Enabled
 - ❑ Diffuse color: Enabled
 - ▶ Area Shape panel:
 - ❑ Shape: Square
 - ❑ Size: 2.0
 - ▶ No Shadow (**Shadow** panel)

What we did was set up a basic area lamp, using a low energy value (it's very close to the **planetEarth** object) and a warm color. We also enabled the **This Layer Only** option to get it to affect only the **planetEarth** object. This light will be the main one in our setup, thus the name 'key'.

- 3. Now let's add the second light of the setup; go to **Add** → **Lamp** → **Spot**, then go to the **Object** tab of the **Properties Editor** and set it as follows:
 - ▶ Name: fill
 - ▶ Transform panel:
 - ❑ Location: 7.62 X, -2.98 Y, 0.66 Z
 - ▶ Rotation: 85.35° X, -0.832° Y, 68.705° Z
 - ▶ Layers: 1 (disable any other layer, **Relations** panel)
- 4. For the lamp properties, let's go to the **Object Data** tab in the **Properties Editor** and adjust the settings as follows:
 - ▶ Lamp panel:
 - ❑ Color: 0.067 Red, 0.106 Green, 0.218 Blue
 - ❑ Energy: 0.6

- ❑ Falloff: Inverse Square
- ❑ Distance: 25.0
- ❑ This Layer Only: Enabled
- ❑ Specular: Disabled
- ❑ Diffuse: Enabled
- ▶ Spot Shape panel:
 - ❑ Size: 45°
 - ❑ Blend: 0.15
- ▶ No Shadow (**Shadow** panel)

Notice that this time we use a spot lamp to get some blending effect from its cone shape. Notice also that the energy value seems to be higher than the one for the key light, but that's because area lamps are very sensitive to their energy value, so in reality the 'key' (area) lamp will have a stronger effect than the 'fill' (spot) one. The role of the fill light is to give a soft lighting to the area that the key light will not be affecting.

5. To finish the lighting setup, let's add another **Spot Lamp** (**Add** → **Lamp** → **Spot**) and set its object properties as follows (**Object** tab in the **Properties Editor**):
 - ▶ Name: rim
 - ▶ Transform panel:
 - ❑ Location: -1.43 X, 6.85 Y, -4.31 Z
 - ❑ Rotation: 121.85° X, -6.75° Y, 187.66° Z
 - ▶ Layers: 1 (disable any other layer, **Relations** panel)
6. The lamp properties (**Object Data** tab in the **Properties Editor**) must be set as follows:
 - ▶ Lamp panel:
 - ❑ Color: 0.067 Red, 0.105 Green, 0.218 Blue
 - ❑ Energy: 100.0
 - ❑ Falloff: Inverse Square
 - ❑ Distance: 25.0
 - ❑ This Layer Only: Enabled
 - ❑ Specular: Disabled
 - ❑ Diffuse: Disabled

- ▶ Spot Shape panel:
 - Size: 45°
 - Blend: 1.0
- ▶ No Shadow (**Shadow** panel)

This light will serve only the purpose of helping the viewer to know the shape of the planet more easily by lighting a thin border in the dark zone of the **planetEarth** object. Notice the very high value used for the energy.

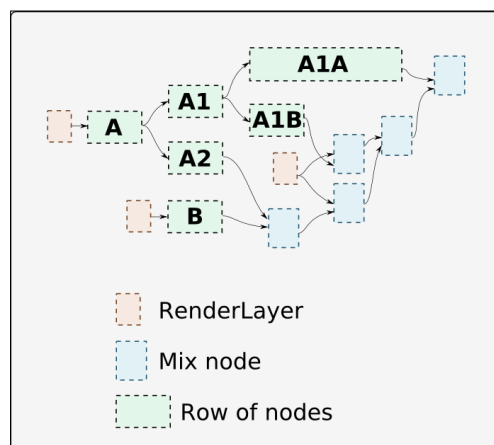
Objective Complete - Mini Debriefing

The lighting setup that we created is a very common setup—a light acts as the main (key) light working as the primary light source. A second light is used to fill the areas that cannot be lit by the key light. Notice that the filling must be subtle to keep the idea of shadow. Finally, the rim light is needed to help the viewer in 'reading' the shape of the object, which is very important to separate the main subject from the rest of the scene.

The nice thing about compositing is that we don't have to put much effort into the scene; but just make sure that it offers enough richness to be used as the starting point for compositing.

Compositing Preparation

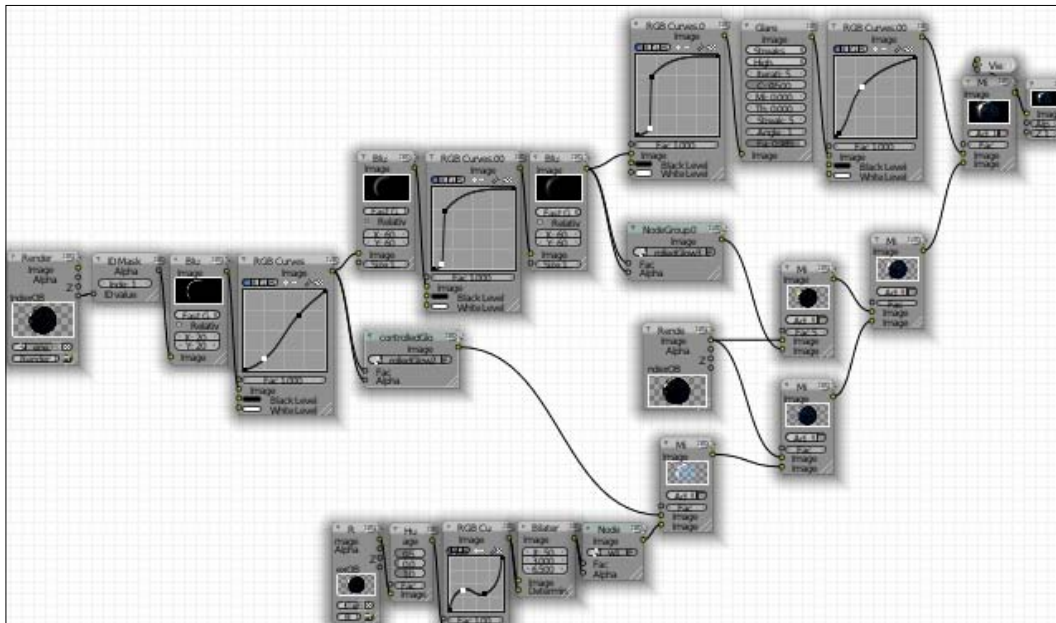
The setup that we are going to create is a bit complex, so we better perform some preparation tasks in advance. Let's start by looking at a diagram summarizing the general structure and defining some notation to aid in understanding the written explanation.



Notice the three different kinds of blocks used in the previous diagram. They are needed

to understand better the individual screenshots of each part of the setup, as we see them. The **RenderLayer** and **Mix** blocks are individual nodes. But the blocks of type **Row of nodes** represent a group of nodes connected in a straight row. The reason to use multiple RenderLayer nodes (remember that we only have one RenderLayer set up in the scene) is to avoid having connections coming from the very left-hand side to the right-hand side, which would make it very difficult for us to understand the setup.

Having seen the simplified version of the node setup, let's take a look at the actual, finished setup, as shown in the following screenshot:



Let's take a couple of minutes to compare this screenshot with the previous diagram, to make sure the general structure of the setup is completely understood. It is suggested to identify each group of nodes by their different heights, to make it easy to recognise each one of them in their individual screenshots.

Now let's work on the current task, which is just performing some preparation steps before creating the actual node setup; namely the configuration of the Render Layer (**Render** tab of the **Properties Editor**) and the creation of a group of nodes to be duplicated later.

Engage Thrusters

At this point, we're almost done with the scene setup, the only things we need to do are setting the world background to black and adjusting some settings for the compositing.

1. To set the background color for the world, we just need to go to the **World** tab of the **Properties Editor** and set the **Horizon Color** to entirely black.

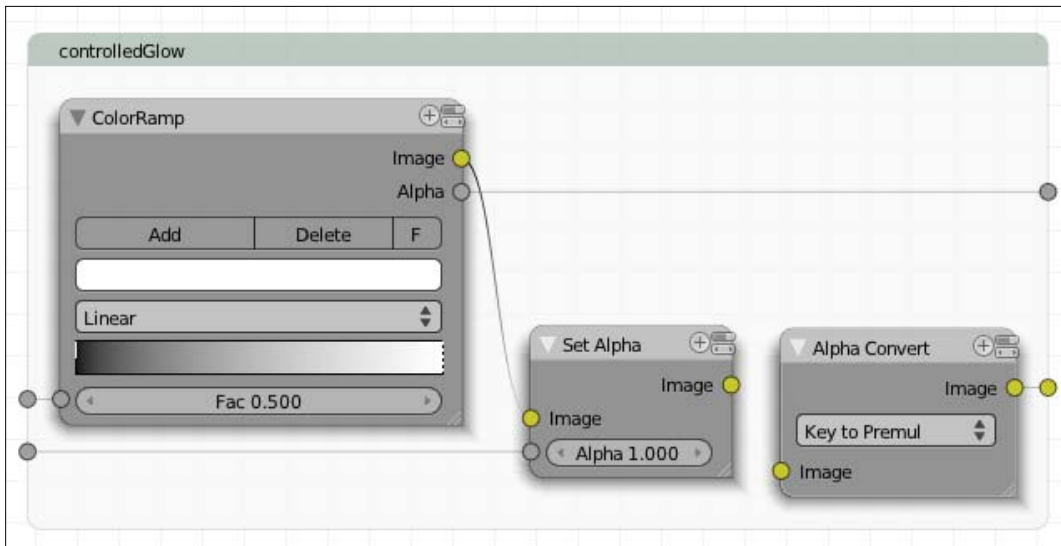
The final details of the scene setup are required for feeding the compositing setup with the correct data and telling Blender to use the composite result instead of the raw render.

2. Let's go to the **Render** tab of the **Properties Editor** and adjust the settings as follows:

- ▶ Layers panel:
 - ❑ Scene: layers 1 and 2
 - ❑ Layer: layers 1 and 2
 - ❑ Passes: enable Combined, Z, and Object Index
- ▶ Dimensions panel:
 - ❑ Resolution: 1920 X, 1080 Y, 100%
- ▶ Post Processing panel:
 - ❑ Compositing: Enabled

The only special things we need to do here are to include the Object Index pass (needed to extract the atmosphere object in the compositing) and making sure the render pipeline will take into account the compositing setup that we'll be creating. For the render size, we set values that play well with the settings in the compositing nodes that we'll create.

3. Let's also rename the scene to 'earth-scene' before going on into the **Node Editor**.
4. Now let's switch to the Compositing screen of our file by using the drop-down menu that is immediately on the right-hand side of the **Help** entry of the main menu. In the **Node Editor**, let's enable the Compositing nodes option in the header of the editor (located in the bottom of it) and also enable **Use Nodes**.
5. Then let's disconnect the link between the default render layer node and the composite node (drag it away from the input of the **Composite** node) and move the composite node to the right-hand side.
6. Create three nodes without connecting them to the main flow of the setup. The reason to do this is because this group is used in three different places of the setup. We can create a small 'template' to easily duplicate it later. Let's look at the following screenshot, showing the group alone:



To create this group of nodes we just need to add three nodes:

- ▶ **Add → Converter → ColorRamp**
 - ▶ **Add → Converter → Set Alpha**
 - ▶ **Add → Converter → Alpha Convert**
7. Once the three nodes have been created, let's connect the image output of the **ColorRamp** node into the **Image** input of the **Set Alpha** node, then connect the image output of the **Set Alpha** node into the **Image** input of the **Alpha Convert** node.
 8. Now, let's select the three nodes (*Shift + right-click*) and duplicate them twice (*Shift + D*) to get three distinct copies. Don't forget to move the new copies to get them clearly separated.
 9. Once the three copies are created, let's select the three nodes of the first copy, create a group (*Ctrl + G*), and set its name to **controlledGlow1**. Then let's create two more groups with the other two duplicates and name them **controlledGlow2** and **controlledGlow3**, respectively. It is important to duplicate the nodes before grouping them, since the three groups will have different settings. Keep the groups around to be used later when needed in the setup.

Objective Complete - Mini Debriefing

This task was just about performing a few miscellaneous tasks that will get our scene ready for the compositing setup that we'll be creating in the next task.

We started by performing some general scene configuration by setting up the **World** options and the Render Layers. Then we switched to the **Node Editor** and created a template for a group of nodes that will be used in the final setup.

Classified Intel

This time we worked on creating three Node Groups, which is a very nice way of "packaging" some nodes into a "group" node. This has the obvious advantage of giving the setup the appearance of being simpler than it actually is. But there is a more powerful reason to create node groups—the ability to reuse them later, both in the current file and in other files.

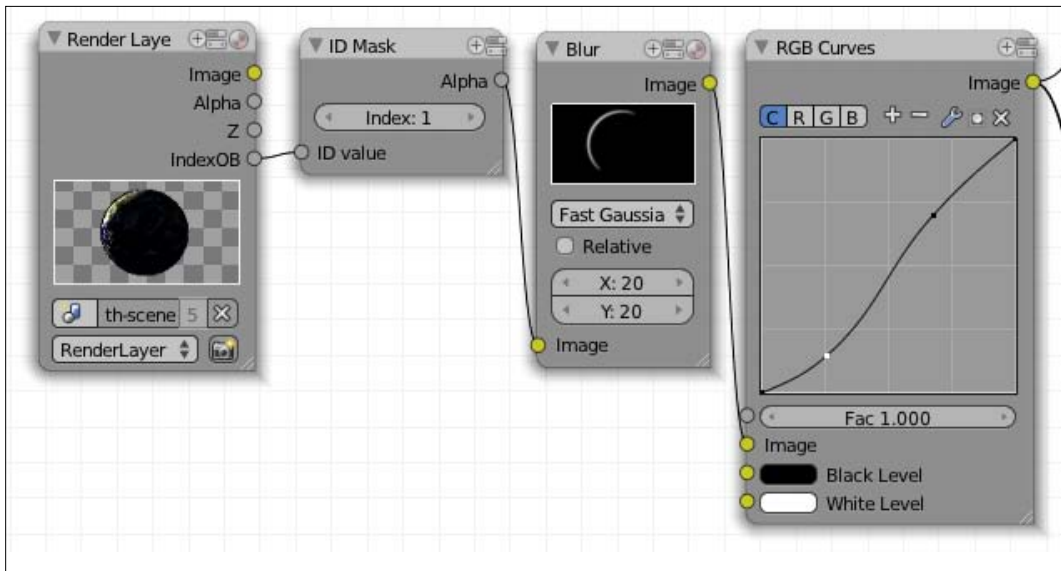
Node groups are saved as **datablocks** that can be browsed when using the **Append or Link** command, which helps a lot when we want to "copy" some functionality from a previous node setup or even to have a nicely organized and helpful library containing groups of nodes for doing common tasks.

Compositing

The scene is fully set up and ready for the final task—**compositing**. If we render the scene as it currently is, the result will be quite boring, but the good news is that we can take it to the next level by applying some nice compositing love.

Engage Thrusters

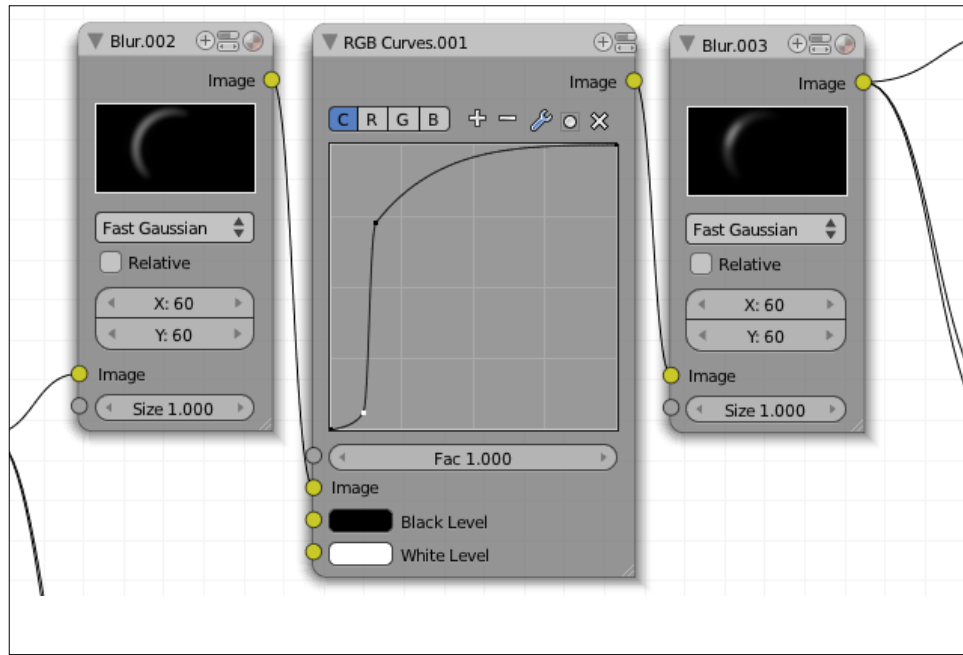
Now we can start building the real setup. Let's start with the "A" group of nodes:



10. To create this row, let's start by panning the view to see the **Render Layers** node, then add an **ID Mask** node (**Add** → **Convertor** → **ID Mask**), connect the **IndexOB** output of the **Render Layers** node to the **ID value** input of the **ID Mask**, then set the **Index** parameter to 1 (the ID of the atmosphere object). Now add a **Blur** node (**Add** → **Filter** → **Blur**), connect the **Alpha** output of the **ID Mask** to the **Image** input of the **Blur**, and set the type of blur to **Fast Gaussian** and the **X** and **Y** radii to **20**.

Finally, let's add an **RGB Curves** node (**Add** → **Color** → **RGB Curves**), connect the **Image** output of the **Blur** node to the **Image** input of the **RGB Curves** node, and set the curve for the "C" channel, as shown in the previous screenshot. Let's remember that the **Image** output of the **RGB Curves** node is the output that will be connected to the "A1" and "A2" rows of nodes.

Now, let's create the "A1" row of nodes; the following screenshot will give us the guidance we need:



11. To create this row, let's start by adding a **Blur** node (**Add** → **Filter** → **Blur**), then set its type to **Fast Gaussian** and both the **X** and **Y** values to **60**. Right after that, let's create an **RGB Curves** node (**Add** → **Color** → **RGB Curves**) and set its curve for the "C" channel as shown in the previous screenshot. Then let's select and duplicate the **Blur** node we created previously and move it to the right, after the **RGB Curves** node.

Finally, let's connect the image output of the "A" row into the image input of the first **Blur** node, then connect the image output of the first **Blur** node into the image input of the **RGB Curves**, and the image output of the **RGB Curves** to the image input of the **Blur** node.

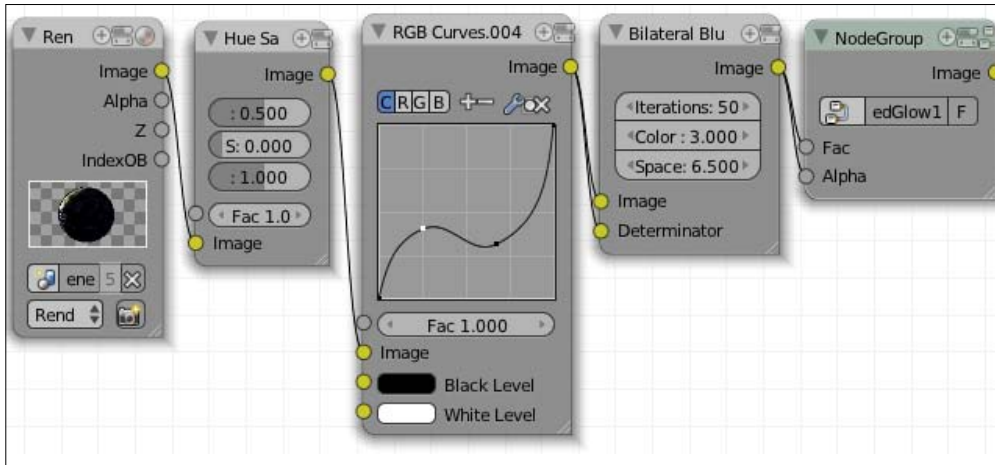


While creating the nodes, you may notice the small thumbnails shown in the screenshots do not appear. This is because the scene hasn't been rendered yet. Once we get to render it for the first time, the compositing pipeline will get the data to work with and the thumbnails will be shown accordingly.

12. The "A2" row will just be one of the groups of nodes we created in the very beginning, let's take the **controlledGlow2** group and move it right below the "A1" row. Next, connect the image output of the "A" row both to the **Fac** and the **Alpha** inputs of the **controlledGlow2** node group.
13. Now we need to edit the internal **ColorRamp** of this group. Let's select it and hit the *Tab* key to access the internal nodes of the group. Select the **ColorRamp** node and edit it as follows:
 - ▶ Interpolation: Ease
 - ▶ Color stop 1:
 - ❑ Color: 0.0 Red, 0.328 Green, 1.0 Blue
 - ❑ Position: 0.068 (the left border of the "a" of "Ease" is a good guide)
 - ▶ Color stop 2:
 - ❑ Color: 0.457 Red, 1.0 Green, 0.907 Blue
 - ❑ Position: around 0.71 (the left-hand side border of the second "e" of "Delete" is a good guide)

Once done with the **ColorRamp**, let's finish editing it by pressing the *Tab* key.

Next, we must work on the "B" row of nodes. Let's see the next screenshot for it:



14. For this row, we start by adding a **Render Layers** node (**Add** → **Input** → **Render Layers**), then a **Hue Saturation Value** node (**Add** → **Color** → **Hue Saturation Value**), and set its **Saturation** value to 0.0, then we must add an **RGB Curves** node and set its curve for the "C" channel as shown in the screenshot.

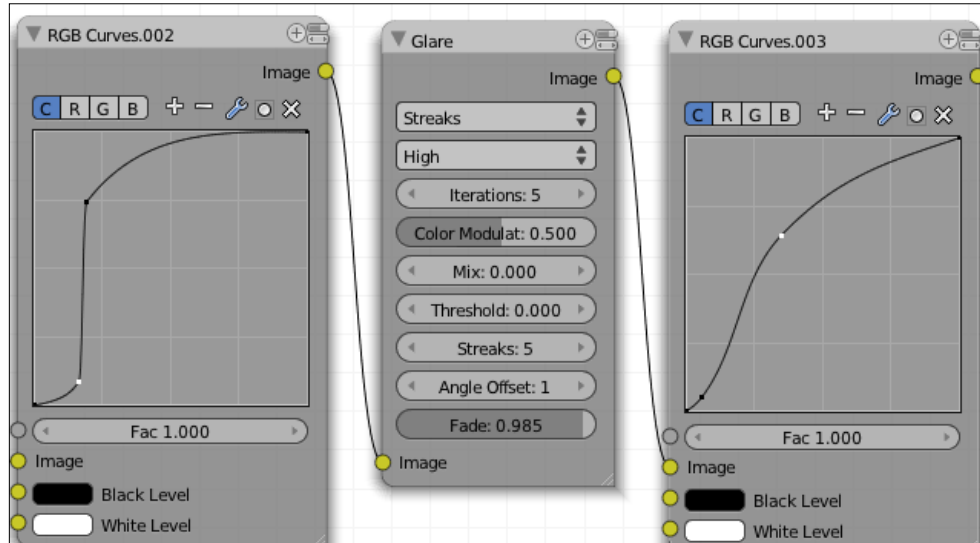
Next, let's add a **Bilateral Blur** node (**Add** → **Filter** → **Bilateral Blur**) and set its **Iterations** to 50, **Color Sigma** to 3.0, and **Space Sigma** to 6.5. Finally, let's take the **controlledGlow1** node group and put it as the last node in the row.

We must now connect the nodes: Let's connect the image output of the **Render Layers** node to the image input of the **Hue Saturation Value** node, then connect the image output of the **Hue Saturation Value** node to the image input of the **RGB Curves** node. Next, we connect the image output of the **RGB Curves** node to both the image and determinator inputs of the **Bilateral Blur** node.

Finally, we must connect the image output of the **Bilateral Blur** node to both the **Fac** and **Alpha** inputs of the **controlledGlow1** node group.

15. For this node group, we must edit its internal **ColorRamp**, so let's select it. Hit the *Tab* key and make sure to set the **ColorRamp** as follows:
- ▶ Interpolation: Ease
 - ▶ Color stop 1:
 - ❑ Color: 0.0 Red, 0.328 Green, 1.0 Blue
 - ❑ Position: 0.0
 - ▶ Color stop 2:
 - ❑ Color: 1.0 Red, 0.0 Green, 0.0 Blue
 - ❑ Position: around 0.43 (the right border of the "F" of "Fac" is a good guide)
16. Let's go back to the normal editing of nodes by pressing *Tab*. Remember the output of the "B" row is the image output of the **controlledGlow1** node group.

Now it's time to work on the "A1A" row of nodes. Let's take a look at the corresponding screenshot:



17. For this row, we need to add two **RGB Curves** nodes (**Add** → **Color** → **RGB Curves**) and a **Glare** node (**Add** → **Filter** → **Glare**). Let's edit the curve for the "C" channel of each **RGB Curves** node as shown in the previous screenshot and set the parameters of the **Glare** node as follows:

- ❑ Glare Type: Streaks
- ❑ Quality: High
- ❑ Iterations: 5
- ❑ Color Modulation: 0.5
- ❑ Mix: 0.0
- ❑ Threshold: 0.0
- ❑ Streaks: 5
- ❑ Angle Offset: 1.0
- ❑ Fade: 0.985

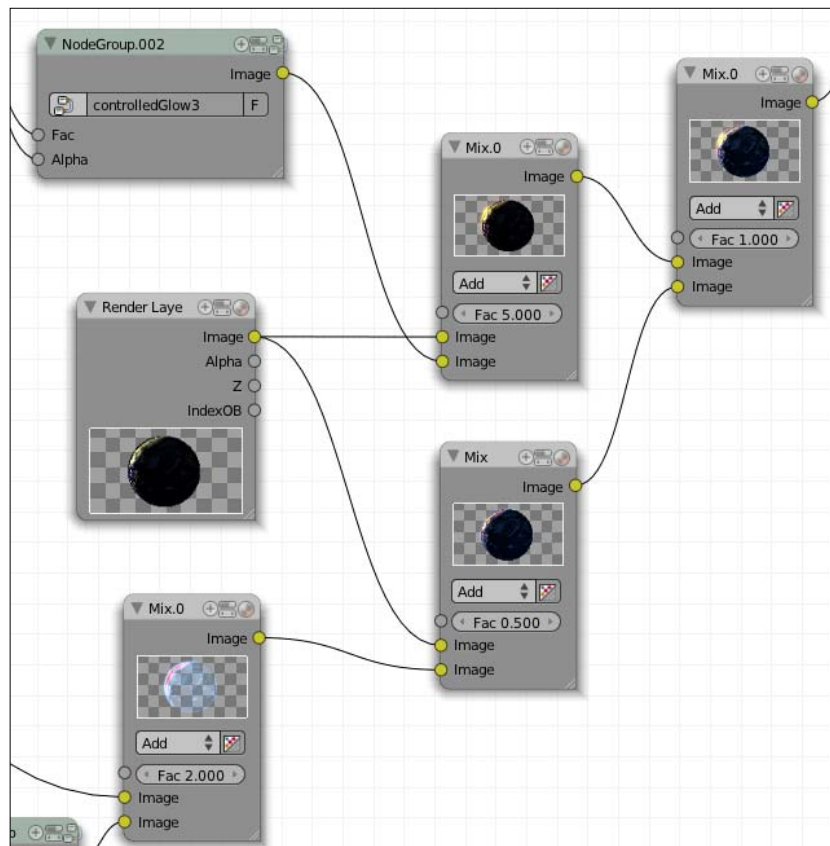
18. The connections are as follows: Connect the image output of the "A1" row to the image input of the first **RGB Curves** node, then connect the image output of the first **RGB Curves** into the image input of the **Glare** node. Finally, connect the image output of the **Glare** node into the image input of the second **RGB Curves** node of this row. This way we get the "A1A" row complete.

19. The next row is "A1B". For this row, we are just going to use the **controlledGlow3** node group, then open it (*Tab*) to access its internal **ColorRamp** node, and set it as follows:

- ▶ Interpolation: Ease
- ▶ Color stop 1:
 - Color: 0.135 Red, 0.024 Green, 0.0 Blue
 - Position: 0.0
- ▶ Color stop 2:
 - Color: 0.924 Red, 0.998 Green, 0.005 Blue
 - Position: around 0.3 (the right border of the second "d" of "Add" is a good guide)

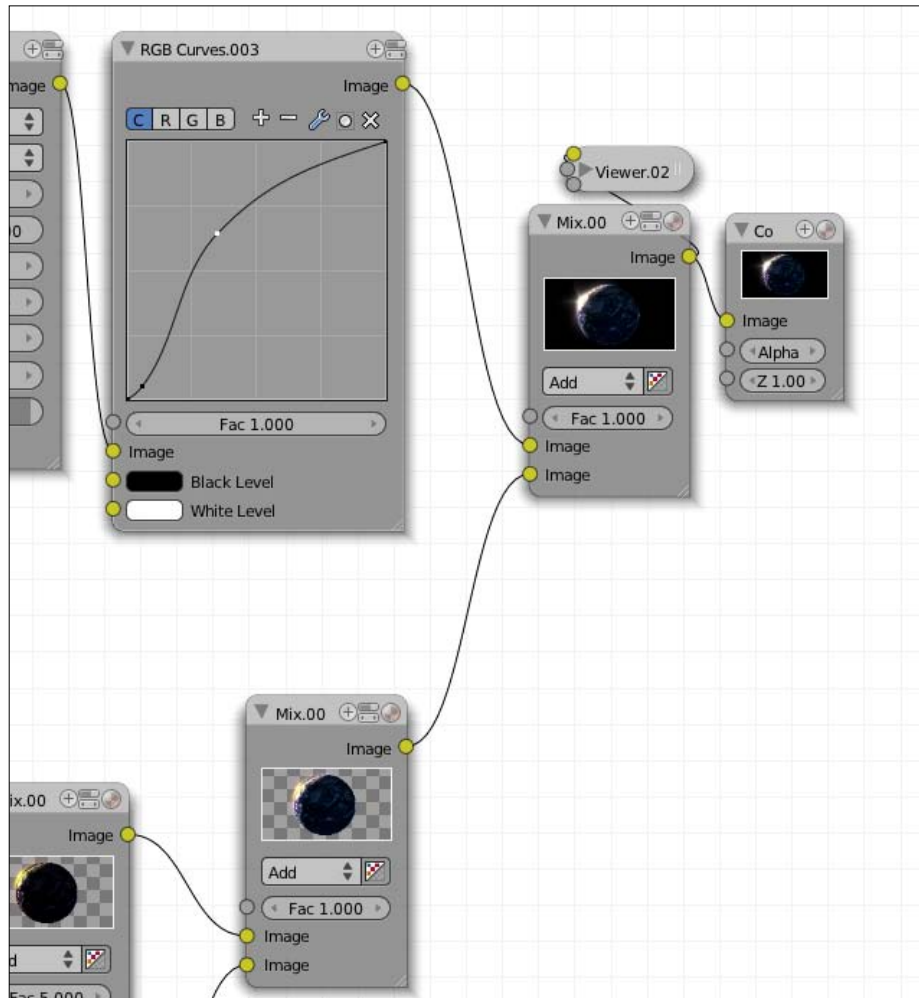
20. Now let's get out from editing the **controlledGlow3** node group by pressing *Tab* and make sure to connect the image output of the "A1" row to both the **Fac** and **Alpha** inputs of the **controlledGlow3** node group.

The only task remaining to be done is to create and connect the mixing nodes. Let's start by looking at the following screenshot:



21. Let's start by duplicating an existing **Render Layers** node and moving it to be right below the **controlledGlow3** node group.
22. Add a **Mix** node (**Add** → **Color** → **Mix**), set its mode to **Add**, its **Fac** to **2.0**, and connect the image output of the **controlledGlow2** node group to the top image input. Then connect the image output of the **controlledGlow1** output to the bottom image input.
23. Next, let's add two more **Mix** nodes; set the first to **Additive** mode and its **Fac** to **5.0**. Then connect the image output of the **controlledGlow3** node group to the bottom image input and the image output of the **Render Layers** node to the top image input. For the other **Mix** node, set its mode to **Add**, its **Fac** to **0.5**, and connect the image output of the **Render Layers** node to the top image input and the image output of the mix node with a **Fac** of **2.0** to the bottom image input.

24. Now we need another **Mix** node; set it to **Add** mode and its **Fac** to **1.0**; the top image input must be connected to the image output of the mix node with a **Fac** of **5.0** and the bottom image input must be connected to the mix node with a **Fac** of **0.5**.
25. To finish the setup, we only need to add a last **Mix** node, set its mode to **Add**, its **Fac** to **1.0**, and feed it with the image output of the "A1A" row (top input) and the image output of the other mix node with a **Fac** of **1.0** (bottom input), as shown in the following screenshot:



26. The last step is just connecting the image output of the last mix node to the image input of the composite node.

And that's it! Your full node setup is ready to process the rendered image and get it to look very nice!

Objective Complete - Mini Debriefing

In this part of the project, we worked exclusively on the node setup to create the final composite that will get our render looking very nice.

We start by generating a simple blurred version of the fragment of the atmosphere object that is visible in the rendered image by first extracting its mask from the render (**ID Mask** node), then blurring it, and finally performing a bit of tweaking with an **RGB Curves** node.

Row "A1" is used to generate another blurred version of the arc of the atmosphere object. But this time we perform a trick by first blurring, then applying a curve, and then blurring again. This way, we get a blurred image showing only the center of the arc, which we will use to simulate the sun 'rising' on the rim of the earth.

The purpose of row "A1A" is to take the result of row "A1" and convert it to a nice, strong highlight of the sun. First we apply a curve to increase the contrast a lot. Then we apply a glare filter to create the highlights and finally, we perform a small correction with another curve.

In row "B", we start by converting the rendered image to grayscale (**Saturation** to **0.0**); we then perform a very strong darkening of the image with a curve followed by a **Bilateral Blur** filter that works almost the same as the common **Blur** filter, but tries to preserve the sharp borders of the image. Finally, we add a **controlledGlow** node group that works by taking the image and mapping it to a **ColorRamp**, thus converting the soft grayscale look into a very colorful one. Both **controlledGlow2** and **controlledGlow3** perform the exact same work as **controlledGlow1**; the only difference is that the **ColorRamps** settings on each one are different. The usage of the **Set Alpha** and **Convert Alpha** nodes is needed to get image buffers that are useful for the mixing process.

The five mix nodes that we used are all set to **Add** mode, as we are starting with an image that is mostly dark and need to get it to look colorful, and have some strong highlights. The value of the **Fac** parameter is raised depending on how strong the influence of the bottom image input should be.

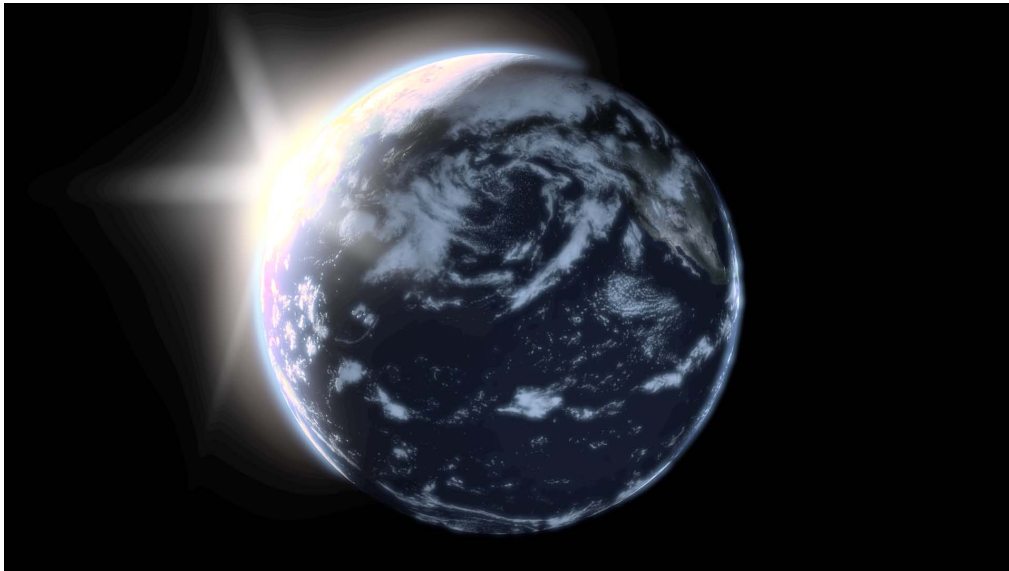
It is highly recommended to use **Viewer** nodes to inspect the result of each node in the chain and understand clearly what the logic is that is driving it.

Classified Intel

In this part, we used two new nodes: The **ID Mask** and the **Set Alpha**. The **ID Mask** node is used to automatically generate a mask showing the silhouette of the object with the **Object Index** given as the parameter. As we set the object index to be different for the atmosphere object, we can extract it from the other objects with the use of an **ID Mask** node. The **Set Alpha** node serves the purpose of defining exactly how we want the alpha of an image to be. In this setup, we just used the same blurred images that were fed into the **ColorRamps**, mostly to avoid increasing the complexity but also because it really makes a lot of sense to use those grayscale images as alpha channels for the bright highlights that we created.

Mission Accomplished

We started with just a few very good textures and a basic scene and finished with a very nice final render of the earth. The main reason for having got so far with such a simple setup (except for the compositing part) is that we got to work with very good quality textures, very clean and well worked out. Once the textures provided us with a nicely mapped earth, having the three most important channels (diffuse, specular, and bump), we just had to work carefully on the compositing to finish the job. So, here it is, our final rendered image:



The resolution of our image even allows us to take a look at some fine details, which give the final render a very good appearance. Let's do it and look for beautiful spots on our rendered image.

You Ready To Go Gung HO? A Hotshot Challenge

After all this hard work, there are still some things that you can do to polish the final render even more:

- ▶ How about adding a nice background? (There is a very nice star field map in the wayback machine copy of www.oera.net.)
- ▶ The dark part of the earth is using the same diffuse map as the part of the earth that is being lit. However, it would be great if we applied the diffuse map showing how the cities of the earth are lit in the night to the dark side. This change would surely require major adjustments on the node setup for compositing.
- ▶ The volumetric look of the rim of the earth should cover more than just the rim; it would be great to get it doing that.

Project 7

Animate a Humanoid Character

A very common, not so simple, and a very nifty exercise to do when learning to animate is a walk cycle. A **walk cycle** is very useful as it allows us to go through many of the aspects of animating a character, but it does not demand us to work on highly complex rigging setups. Let's take the character that we created in the previous chapter, add to it the required elements to be able to animate it, and create a basic walk cycle with it.

Mission Briefing

Starting with the model created in the modeling a basic humanoid character project, we will be adding a basic armature with some simple configuration to make it useful for our purpose.

With the armature properly set up, we'll perform three steps to create the final animation; we'll start by creating the key poses of the character, then adding the breakdown poses in between, and finally adding a small tweak to improve the overall result. In the end, we'll get a nice, basic walk cycle that we can later attach to a path to get it moving along it (not included in this tutorial).

What Does It Do?

This project will guide us through creating an armature from scratch, discussing along the way some basic concepts related to rigging. After the armature is ready, we'll start working on creating an animation, again from scratch, of a walk cycle for our character.

Why Is It Awesome?

This project will allow us to perform a fast tour of the general ideas for creating a rigged system and using it to create an animation.

First, we'll have the opportunity to learn the basic ideas about rigging, by creating a very basic rig that can serve the purpose of creating the walk-cycle animation. This exercise will use the two basic tools that are available, parenting and constraining, and will also let us discuss a bit about the whole mindset behind it.

Secondly, on the animation part, we'll get a fast and clear overview of some basic concepts to get started with, by using the structured method for animation called pose-to-pose. We'll also have enough time as to discuss a few topics belonging to the more advanced aspects of this topic.

It's important to notice that both rigging and animation are huge fields by themselves, so this chapter will only give a very brief overview that can help you get started, by having the right ideas in mind from the very beginning.

Your Hotshot Objectives

We'll be performing six tasks to create our nice walk-cycle animation, as follows:

1. Creating the armature.
2. Armature setup.
3. Skinning.
4. Animation keys.
5. Animation breakdowns.
6. Animation tweaks.

Mission Checklist

First of all, we need to copy the `character.blend` file from the "modeling a basic humanoid character" project (from a sample chapter of the book) to the directory holding the files for the current one. Let's name the copied file `walkcycle.blend`, then open Blender, and load it.

Creating the Armature

As of now, we have the mesh of the character that we are going to animate, but there is a part missing: The "skeleton structure" that will allow us to perform the actual animation task. This "skeleton structure" is called an "armature" in Blender and the first task that we are going to do is just creating its components; later on we will work on setting it up for our purposes.

Engage Thrusters

1. To add our armature, let's ensure that we are in **Object Mode** and the 3D cursor is located at the 3D world origin (**View** → **Align View** → **Center Cursor and View All** or press *Shift + C*).
2. Now let's go to **Add** → **Armature** → **Single Bone**. This will leave us with a long octahedron located right between the feet of our character. Let's change the name of the armature to `char_skel` (**Item** panel, **Properties Sidebar**) and switch to **Edit Mode** for the armature object.

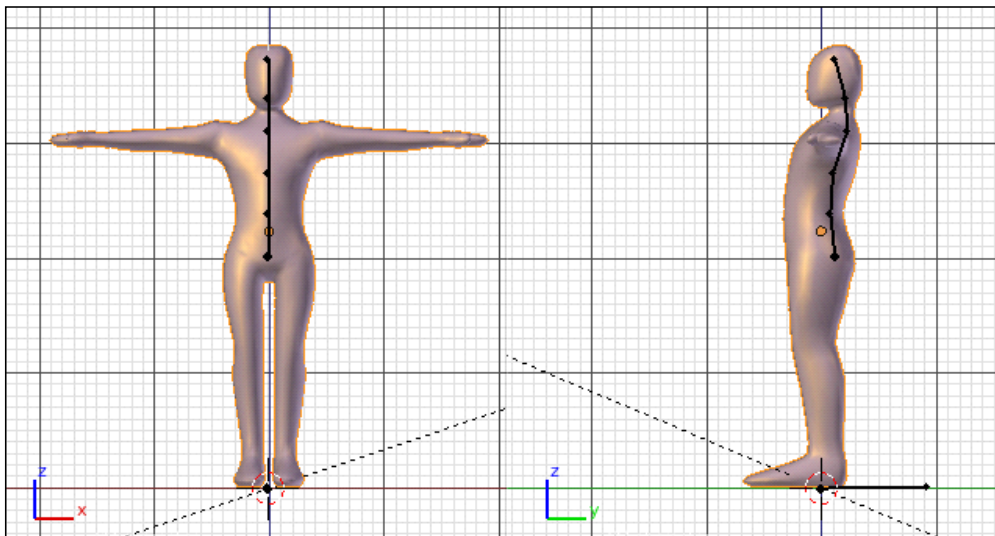


The **armature** is a special kind of object in Blender. It is composed of "bones" that are modifiable when in **Edit Mode**. Each bone has two "tips", called **tip** (thinner part of the octahedron) and **root** (thicker part of the octahedron). Another special aspect of armatures is that they have an extra mode called **Pose Mode**, used for some of the setup tasks and also for the actual animation.

3. Select the entire bone that is created by default (right-click in the middle of it), duplicate it (**Armature** → **Duplicate**), and move it up around 1.9 units (G key, Z key, then type 1.9) to get it inside the torso of our character.
4. As we need to see the bones and they must be located inside the volume of our mesh, let's enable X-Ray visualization of the bones. Let's go to the **Object Data** tab of the **Properties Editor** and check the **X-Ray** option in the **Display** panel. Remember that the 3D viewport must have solid shading enabled (Z key, once or twice).

The duplicated bone is the starting one for a 'chain' of bones forming the spine of the armature.

- Let's select the tip of it and move it -0.6 units along the Z Axis (G key, Z key, then type '-0.6'). Now let's perform two successive extrusions of the selected tip (to create a new bone each time) and move each newly extruded bone around 0.33 units upwards (G key, Z key, then type '0.33'). This should leave us with the tip of the last bone around the same level as the shoulders. Then continue with two successive extrusions to create the neck and head bones. Remember that these bones must be moved only along the Z axis.
- Now switch to **Right View (View → Right)** to tweak the position of the bones. Start by selecting the tip of the bone located between the feet. Then move it so that the bone is running horizontally, with the tip oriented towards the right (positive direction of the Y axis).
- Next, select the tips of the bones forming the spine-neck-head of our character to perform some fine tweaking. Continue tweaking to get them to look like the next screenshot. Notice that the "sticks" option has been activated (**Display** panel, **Object Data** tab of **Properties Editor**) to make it easier to understand the image:



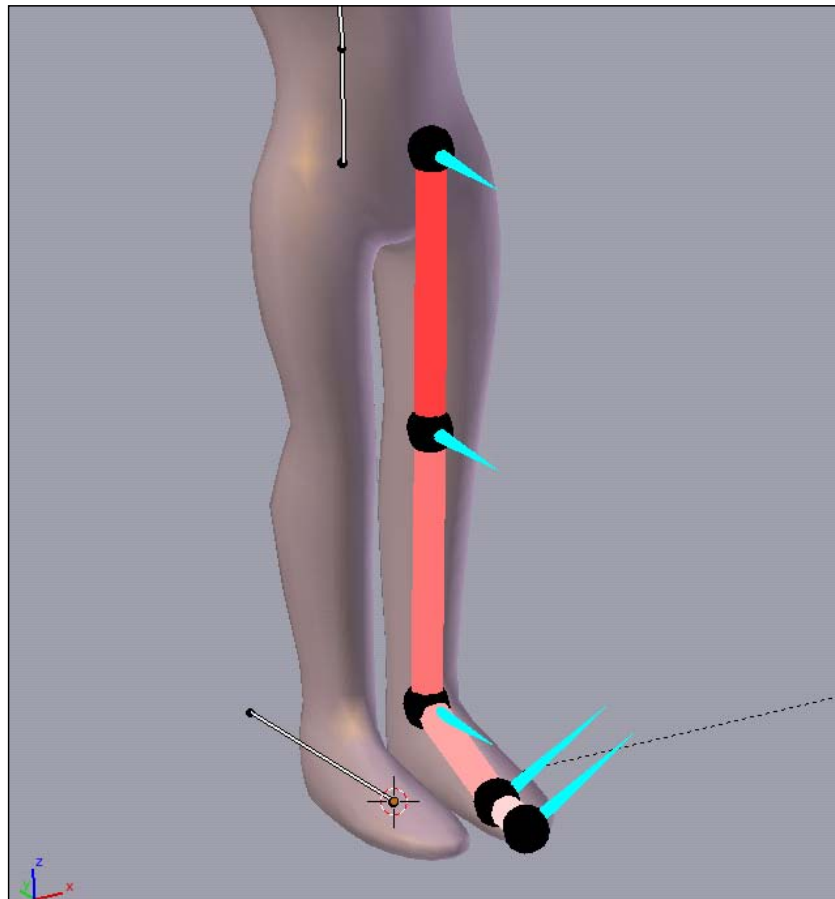
- Once the spine is done, let's switch back to **Front View (View → Front)** to create the chain of bones for the character's left leg.

This time we'll be working with a very nice and intuitive tool for creating chains of bones automatically inside the volume of a mesh.

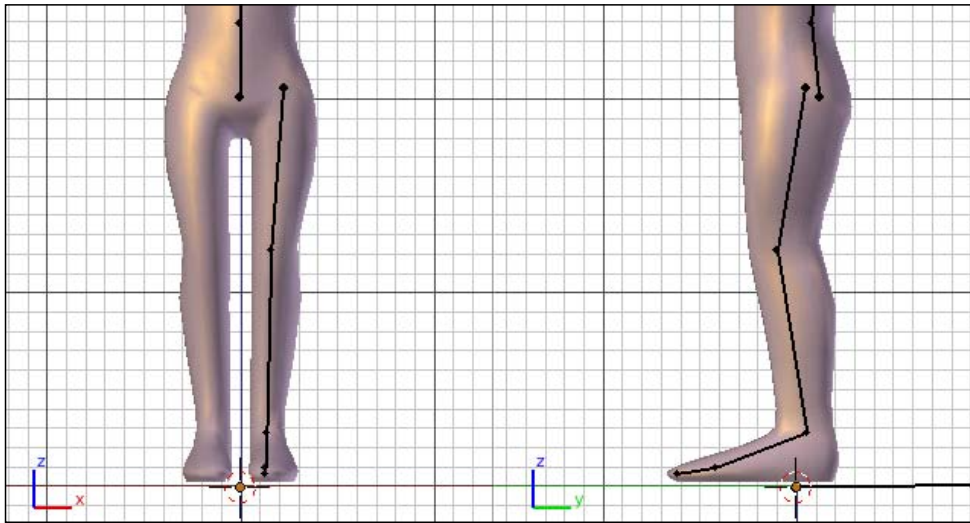
9. Let's open up the **Properties Sidebar** (*N* key), locate the **Skeleton Sketching** panel, enable it, and also enable the **Quick Sketching** option (visible when the panel is opened). Let's also go to the header of the 3D View, locate the **Snap Element** drop-down menu (next to the magnet icon), set it to **Volume**, and make sure the magnet button isn't pressed down.

With the **Skeleton Sketching** tool enabled and the snapping options correctly set up, let's now sketch the chain. Remember to hold the *Ctrl* key and put the mouse pointer over the surface of the volume (not over the empty space).

10. Now let's click once on the waist (a bit above the level of the root of the first spine bone and between this and the border of the waist), then click on the knee, and then click on the thin part of the ankle. Now let's rotate the 3D view to be able to see the foot better and add two sketch points—one close to the toes (but leaving some space) and one in the very tip of the foot. After performing this operation, we should have a sketch as shown in the following screenshot:

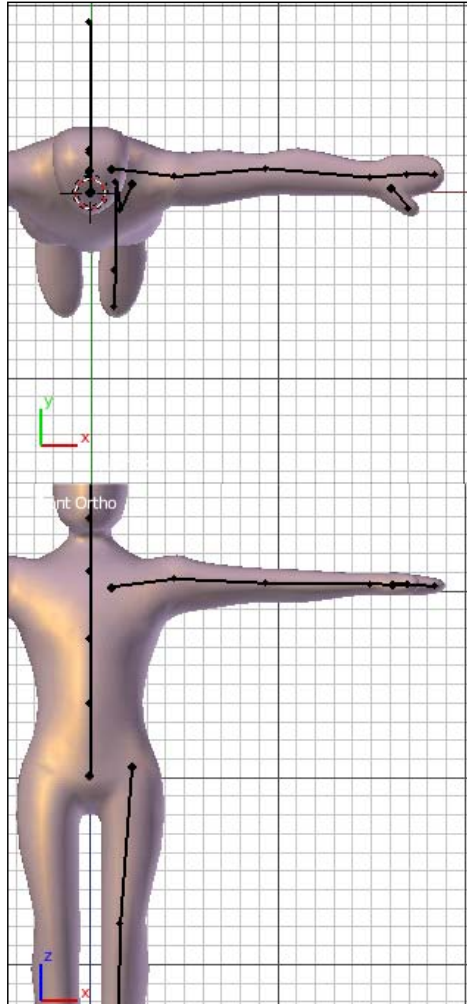


11. Let's rotate around to check that the sketched points are located inside the volume of the mesh. When everything's okay, let's right-click to get the sketch converted into actual bones. Let's now uncheck the **Skeleton Sketching** panel (**Properties Sidebar**) to perform some manual tweaking on the generated bones.
12. To improve the leg chain, let's now switch to **Right View (View → Right)** and move the root of the first bone (the one closer to the spine) along the Y axis to be closer to the back part of the mesh. Let's also tweak around the tips of the other bones to give a little curvature to the chain. Take a look at the next screenshot to know what shape we are aiming for:



13. Next, we must create the chain of bones for the arm. Let's again enable the **Skeleton Sketching** panel (with the **Quick Sketching** option enabled) and switch to **Front View (View → Front)** to start sketching. Don't forget to hold the **Ctrl** key pressed when clicking on the mesh.
14. Start the arm chain by clicking on it a bit to the right-hand side and below the tip of the third spine bone (on the chest of the character). Then click on the shoulder joint (a bit closer to the top than to the bottom) and continue by clicking on the right-hand side in the joint of the upper arm and the forearm. Then click on the wrist and finally, rotate the view and zoom it to add two bones to the simple hand of our character. Finish the sketching by right-clicking anywhere and perform a fast sketching by adding two points in the thumb of the hand, then right-clicking. Let's now disable the **Skeleton Sketching** panel (**Properties Sidebar**) to perform some manual tweaking.

15. For tweaking the arm bones, let's switch to **Top View (View → Top)**. First, select the root of the first bone of the chain and move it around 0.06 units along the positive direction of the Y axis and let's also work on the placement of the joints to get something resembling the result shown in the next screenshot:



Pay close attention to the shape of the chain, specially the 'corner' (top view) in the joint of the upper arm and the forearm. Also, notice that the first bone goes slightly up from its root to its tip (front view).

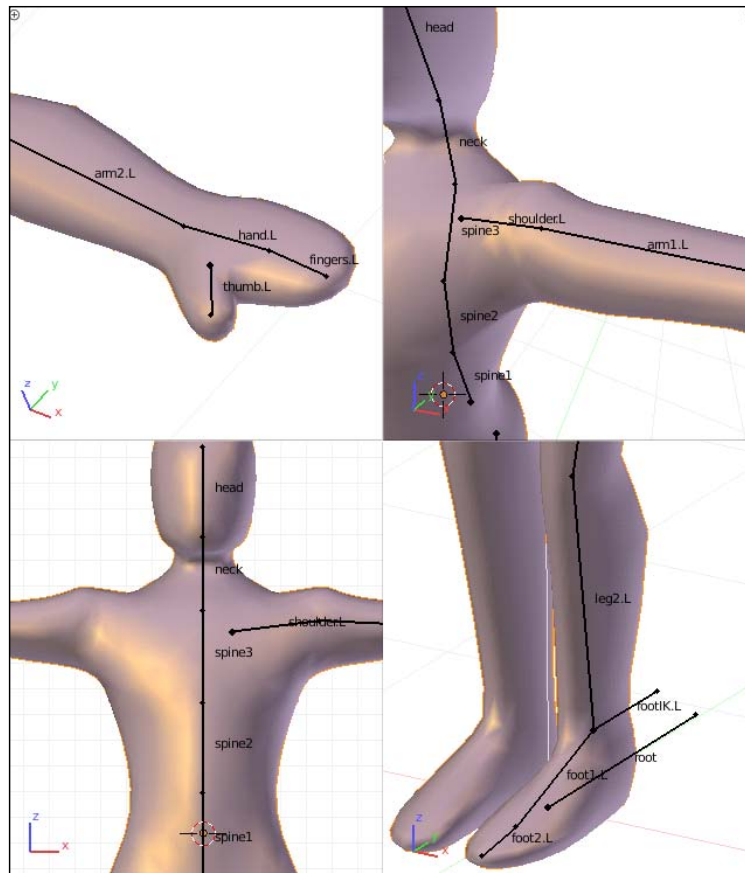
There is only one last bone to be added, which will serve to create the only mechanism that we are going to add to our armature.

16. Switch to **Right View** (**View** → **Right**), select the ankle joint (only the tip, not the entire bone), extrude it (*E* key), and then move it straight along the Y axis (*Y* key).
17. After adding the bone, we must detach it from the chain. To do this, select the new bone, go to the **Bone** tab of the **Properties Editor**, locate the **Relations** panel, and clear the entry on the **Parent** field (Click on the entry, press *Delete*, and then press *Enter*).

The names of the bones are the default ones, so the next step we should do is change them to more meaningful ones.

18. Let's make sure we have the **Properties Sidebar** open, then locate the **Item** panel to have it at hand. What we must do is select each bone, set its name, and then go for the next one. These are the names that we must set:
 - ❑ Spine chain: spine1, spine2, spine3, neck, head
 - ❑ Arm chain: shoulder.L, arm1.L, arm2.L, hand.L, fingers.L, thumb.L
 - ❑ Leg chain: leg1.L, leg2.L, foot1.L, foot2.L, footIK.L (bone extruded from the ankle joint)
 - ❑ Main bone: root (first bone, sitting in between the feet)

The following screenshot shows the names of the bones besides each one:



To finish the creation of the armature, let's just duplicate the left part and mirror it to create the right arm and leg.

19. Start by switching to **Front View (View → Front)**, then make sure that there are no bones selected (press the A key, once or twice), initiate the **Border Select** tool (**Select → Border Select**), and draw selection rectangles containing the arm and then the leg. Make sure that you have not selected bones from the spine or the root bone.

With the bones to be duplicated already selected, let's now make sure that we have the proper settings for the duplicate and mirror operation.

20. First, let's put the 3D Cursor at the origin of the 3D World (set its coordinates to **0** in the **View** panel of the **Properties Sidebar**). Set the **Pivot Point** to the 3D cursor (second drop-down to the right-hand side of the **Object Mode-Edit Mode** drop-down, header of the 3D View).

21. Now we just need to duplicate the selected bones (**Armature** → **Duplicate**), press *Escape*, and then use the Mirror tool with the X Global option (**Armature** → **Mirror** → **X Global**, and then press *Enter*). This way, we get both the arm and the leg for the right side of our character.
22. There is a final tweak to be done, though. With the duplicated bones selected, let's go to **Armature** → **Flip Names**. This last operation will set the names of the duplicated bones appropriately for their side (**.L** for the left, **.R** for the right).

And that's it for the creation of the armature, let's save our file and continue working on the next task.

Objective Complete - Mini Debriefing

In this task, we just added a very simple armature for our character. We started with the basic bone for a default armature and then duplicated it to create the chain of bones for the spine-head. For the leg and arm, we used a very nice tool for sketching bones called **Skeleton Sketching**. For this tool to work properly for us, we had to set some snap options (Volume snap) and use the *Ctrl* key while clicking to sketch (the *Ctrl* key activates the snapping; the snapping mode was not enabled as such).

We also prepared a bone that will be used to create a rig mechanism later (**footIK.L**) and detached it from the chain that we extruded it from.

For the sake of keeping our project organized, we renamed each bone, giving meaningful names and using the **.L** suffix (as a hint for Blender to be able to assist us later when using some tools).

Finally, we mirrored the bones on the left-hand side of our character to create the ones on the right-hand side and used a very useful command to get the naming of the new bones correct. This way, each **.L** bone has a matching **.R** bone.

Armature Setup

Our armature is now ready to be set up using some configuration options that will make it a lot more useable later, when animating it. The setup process is needed not only to reduce the complexity of using the armature for animation, but also to create very sophisticated "puppet-like" systems that can help the animator a lot. For this project, we'll only use a couple of basic options for the armature setup—parenting and IK constraint.

Engage Thrusters

The first task that we must perform is the parenting process.

1. Let's make sure that we are in **Edit Mode** for the `char_skel` armature, then go to the **Bone** tab of the **Properties Editor**, locate the **Relations** panel, and then the **Parent** textbox (with a small bone icon to the left); now apply the parenting as indicated:
 - ❑ shoulder.L and shoulder.R must be parented to spine3
 - ❑ thumb.L must be parented to hand.L; thumb.R must be parented to hand.R
 - ❑ leg1.L and leg1.R must be parented to spine1
 - ❑ spine1 must be parented to the root
 - ❑ footIK.L and footIK.R must be parented to the root

Remember that the bone that must be selected to set up its parent is the child bone; in the previous list, we must select the **shoulder.L** bone, and then set its parent in the **Bone** tab of the **Properties Editor** to **spine3**.

This very easy task also gives us the most basic tool for setting up an armature, that is creating the hierarchy of bones. From now on, when animating a bone that has child bones, the child bones will be affected by the animation of the parent bone. We must note that the bones created by extrusion are automatically parented to the bone they are extruded from.

The second part of the setup is about creating a very common but simple mechanism that will enhance the level of control that we can have over our armature.

2. Let's switch to **Pose Mode** (using the same drop-down where we find **Object Mode** and **Edit Mode**), then select the **leg2.L** bone, go to the **Bone Constraints** tab of the **Properties Editor**, add a new **Inverse Kinematics** constraint, and set it as follows:
 - ❑ Target: `char_skel`
 - ❑ Bone: `footIK.L`
 - ❑ Chain Length: 2

This will cause the **leg2.L** bone to be "controlled" by the **footIK.L** bone, by means of this constraint. The value of 2 for the **Chain Length** parameter is needed to limit the controlling effect from going beyond the leg chain and into other parts of the armature, which is something we don't want to have at all.

3. Let's create another **Inverse Kinematics** constraint, this time for the **leg2.R** bone, using these settings:
 - ❑ Target: char_skel
 - ❑ Bone: footIK.R
 - ❑ Chain Length: 2

This time we simply repeated the same procedure, but using the right settings for the right leg of our armature.

The final task that we must perform to get the armature setup done is recalculating the roll of the bones.

4. Let's switch to **Edit Mode**, select all the bones (press the A key, once or twice), then go to **Armature** → **Bone Roll** → **Recalculate with Z-Axis Up**. This operation causes the internal rotation of the bones to be set in such a way that we can animate it later and make use of a very useful mirroring tool. Let's save the file before continuing.

For the sake of analyzing the effect of the setup steps that we just performed, let's take a couple of minutes to play with the armature in **Pose Mode**. Select a bone, move it around, rotate it, and see how it affects other bones in the armature; do this for all bones in the armature so that you can see how the entire skeleton works. When done with this exploration task, let's select all the bones (the A key) and make sure to perform a clearing of all transformations (**Pose** → **Clear Transform** → **Clear Location**, then **Clear Rotation**, then **Clear Scale**).

Objective Complete - Mini Debriefing

This task was a very straightforward one, as we are creating a very basic armature setup, the procedure was quite short. First, we set up the parenting relationship between various bones in the armature, which is the basic tool that we can use for adding some functionality to the armature. Then we worked on using a constraint for two bones in our armature, the constraint we used is a widely used one, called **Inverse Kinematics**. With these two simple tasks, our armature offers enough functionality for us to work out the walk cycle that we intend to create.

Classified Intel

Even though the task was quite easy to perform and very short too, it doesn't mean that the setup of an armature is a trivial task. Let's discuss the general ideas behind this task in more depth.

Before discussing the technical parts, let's first ask ourselves—what is the purpose of performing all these adjustments to the armature? The answer can be summarized as—sharpening the axe before cutting a tree. The armature is already the structure that will drive the mesh when animating, so it is important that we can really have control over it. The purpose is twofold—avoiding undesired behaviors (for example, a knee bending backwards), and giving the animator a lot of control over the armature (and thus, the mesh) without having to perform complex tricks. As the desired scenario is having a very flexible rig and making life easy for animators, we must resort to some technical methods for solving different kinds of problems, and the solution offered by Blender allows us to create special relationships (parenting and constraints) between different bones. Playing with these "building blocks", we can construct very complex rigs that have a lot of animation functionality emerging directly from the setup, thus fulfilling the purpose of aiding the animator. It's true that in this project we are using a very simple setup, but the whole idea is the same, no matter how complex the setup is.

Parenting

For one part, we have parenting. Parenting is the most common relationship that we can establish between two or more objects in a 3D package, not only when talking about animation, but also for other purposes.

The idea behind parenting is getting the child object to "follow" the transformations of its parent. If the parent moves, the child element moves along with it, if the parent is scaled, the child item will also be scaled accordingly. This creates the idea of dependency, which is the whole goal. Parenting is what allows us to create a strong dependency upon another object's state. Notice, though, that this dependency is quite simplistic: All transformations of a parent (move, scale, and rotate) affect its children.

Constraints

For the more complex aspects, we have constraints. Constraints are a mechanism to create different kinds of dependencies between objects, some of them simple (for example **Copy Location**) and some of them complex (for example **Inverse Kinematics**). The useful thing about constraints is that they offer many parameters, which come with their own distinct set of problems. These problems can be resolved by using them with the correct tweaks to their parameters. A good example is the **Copy Location** constraint: It can be used to transfer the entire location from the target object to the constrained one, or it can only be applied to one of the coordinates.

The **Inverse Kinematics** constraint that we are using in our setup is one that allows us to easily control the feet of our character, thus helping a lot to get the normal behavior of feet: Once they hit the ground, they stay quiet, no matter how much other parts of the body move. The **Inverse Kinematics** constraint achieves this by adjusting multiple properties of the bones of the chain automatically for us.

Skinning

Now we must establish the way the armature is going to affect the mesh. For this, it is necessary to relate each bone to a set of vertices and also define the amount of influence that it will have over them. This type of "relationship" between bones and vertices is called skinning. The way Blender does skinning is by assigning an influence value on each vertex for each bone: 0 means no influence, while 1 means full influence. The actual tools for creating this "weighting" are weight painting or using the automatic population of those collections of data; we'll just go on with the automatic method, which gives us a weighting that is good enough for our purposes, but can also be tweaked with the other method if needed.

Engage Thrusters

Let's switch the `char_skel` armature to **Edit Mode** to perform a couple of tweaks to avoid getting a bad skinning result.

1. Select the root bone, then go to the **Bone** tab of the **Properties Editor** and disable the **Deform** panel. Do the same for the **footIK.L** and **footIK.R** bones. The reason to perform this operation is simple—these bones are a part of the mechanism of our armature; their job is not to deform the mesh, but to control other bones, so we must specifically tell Blender that they should not be used for calculating deformations. Now let's switch to **Object Mode**.
2. Select the character object (mesh), then, also select the `char_skel` object (both objects must be selected now), and go to **Object** → **Parent** → **Set**. Then choose **With Automatic Weights** from the list. This will tell Blender to perform a special calculation to try to determine how each bone must affect the mesh, and set up the different configuration settings needed for the correct relationship to be established (adding a modifier and populating vertex groups).
3. Now let's select only the character object, then go to the **Modifiers** tab of the **Properties Editor**, locate the **Armature** modifier that was placed below the **Subdivision Surfaces** modifier (thus being applied after it), and click on the arrow pointing up to get it moved to the first place in the modifiers list. This will cause the armature to affect the mesh before the **Subdivision Surfaces** modifier is applied. Now let's enable the **Quaternion** option and disable the **Bone Envelopes** option of the modifier, to improve the deformation (quaternion) and to not use the bone envelopes method for deformation (using only the vertex groups created automatically by Blender when using the **With Automatic Weights** option).

4. Let's take a couple of minutes to check the deformation mechanism that we are going to use: vertex groups. With the character object selected, let's go to the **Object Data** tab of the **Properties Editor**, locate the **Vertex Groups** panel and check the names of the elements in that list; do they sound familiar? Sure they do! Each vertex group is named after the bone for which it is providing the influence information.
5. To check the data the vertex groups are storing, let's now go to the 3D View and switch to **Weight Paint** mode (only the character object must be selected). Now the mesh is shown with a blue color and we can check each vertex group by clicking on its name in the **Vertex Groups** panel (**Properties Editor**). A more intuitive way is to right-click on any bone of the armature (the armature should be visible, as we enabled the X-Ray option early on in the project) to check its associated vertex group. By default, Blender indicates no influence (weight of 0) with a strong, dark blue and full influence (weight of 1) with a strong, vivid red. Let's take a couple of minutes to inspect how the vertex group for each bone is populated with the weight information defined by the automatic weighting that Blender offers.

Of course, we can modify the influence of each bone, by editing the weighting information stored in its associated vertex group; to do this, we just need to enable the Tool Shelf Sidebar (**View | Tool Shelf**), adjust the brush options (notice the weight value shown right below the brush example image and the **X Mirror** option in the **Options** panel), and paint over the mesh. Even though the current weighting information is not perfect, it's good enough for the purpose of our walk-cycle exercise, so we won't be playing with the brush options to modify the vertex groups' information.
6. Now, let's switch to **Object Mode**, select the `char_skel` armature, switch it to **Pose Mode**, and take a couple of minutes to try out how the armature is now deforming the mesh, which is exactly what we needed to have our setup complete. Remember to select all the bones and clear their transformations (**Pose** → **Clear Transform**) before continuing with the next part. Also, let's remember to save the file.

Objective Complete - Mini Debriefing

The third step of the process was dedicated to take the finished armature and making it affect the mesh of our character. We started by marking some bones not to be used for deformation, then we used the automatic weighting command that Blender offers to generate the vertex group information (thus saving us a *lot* of work). Finally, we just adjusted a couple of parameters of the **Armature Modifier** added to the character object and explored around to see how the weighting mechanism works and can be modified.

Animation Keys

The preparation steps are done, our armature is ready to be animated. In this first part, we will work on creating the main poses of the walk cycle, which are the starting points of the process.

Getting ready

Since we'll be working on animation, it's better to use a different screen; let's go to the screens drop-down list (next to the **Help** item of the main menu) and select the Animation screen. Then let's go down to the Timeline and enable the recording button (automatic keyframe insertion for **Objects** and **Bones**). Let's set the end frame to 40 (**End** parameter, to the left of the recording icon) as well.

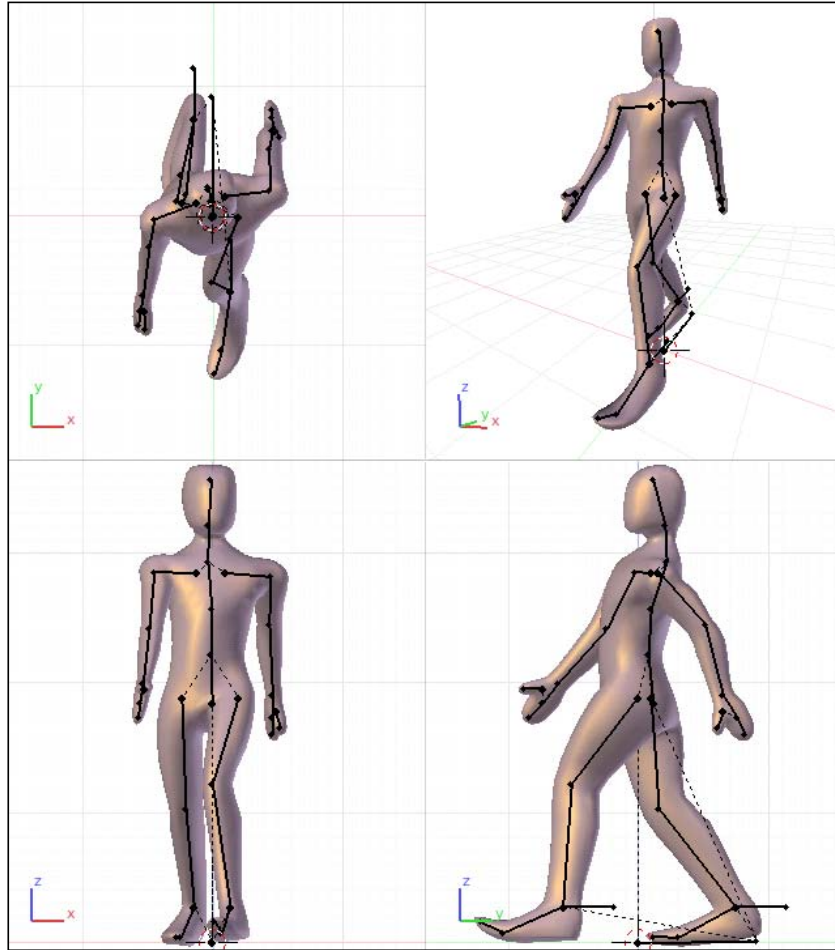
It's also important to know what the editors on the left-hand side panel do—the upper one is the **Dope Sheet** and the lower one is the **Graph Editor**. The **Dope Sheet** is the primary visualization tool for animation keys that we will be inserting; each key gets represented by a diamond and each row corresponds to one bone of the armature of the character. The **Graph Editor** is used to control the exact shape of the curve, defining the way a certain property of a bone (rotation, location) changes from one value to the next. A good way to explain how these editors work together can be to think of the **Dope Sheet** as the one where we define poses in time and the **Graph Editor** as the one in which we work on refining the transformation that happens between poses.

Engage Thrusters

1. Let's go to the 3D Viewport and switch to **Right View (View → Right)**, then select the armature and make sure that it's in **Pose Mode** to start working with it. Let's also make sure that we are in frame 1 of our animation (check the label of the green, vertical bar of the **Dope Sheet**) and set the view mode to **Orthographic (View → View Persp/Ortho, if it's in perspective)**.
2. Select the **footIK.L** bone and move it -0.65 units along the Y Axis (G key, Y key, then type -0.65), notice that the autokeying option has inserted the keyframes for this bone for us.
3. Now let's select the **footIK.R** bone and move it 0.69 units along the Y axis (G key, Y key, then type 0.69). This will cause the left foot of the character to be put in front and the right foot to move behind.
4. Now select the **spine1** bone and move it -0.18 units along the Z axis (G key, Z key, then type -0.18) to get the feet touching the IK controllers and have a little bending on the knees.

5. Continue by selecting the **foot1.L** bone, and rotating it to get it flat over the horizontal line of the grid, then select the **foot2.L** bone and rotate it to get the toes bending upwards around 30°.
6. For the right foot, let's select the **foot1.R** bone, rotate it until the only part passing through the horizontal level of the floor is the toes, then rotate the **foot2.R** bone to get the toes running horizontally on the level of the floor. This will give us the basic pose for the lower part of the body.
7. For posing the torso, let's now select the **spine1** bone and rotate it -10° on the Z axis (*R* key, *Z* key, then type -10); this will cause the torso to swing according to the foot that is in front (left foot).
8. The upper part of the body must be posed by switching first to **Front View (View → Front)**, then selecting the **shoulder.L** bone and rotating it around 10° downwards (clockwise).
9. Continue by selecting the **arm1.L** bone and rotating it down until the arm gets very close to the waist (around 70°).
10. For the right arm, let's perform the same operations, but keep the shoulder a bit higher than the left one, to break the symmetry.
11. Now let's switch to **Right View (View → Right)**, select the **spine3** bone and rotate it 20° on the Z axis (*R* key, *Z* key, then type 20) to compensate the rotation that we did on the **spine1** bone.
12. Now let's work on the arms. Select the **arm1.L** bone, rotate it -30° (*R* key, then type -30), then select the **arm2.L** bone and rotate it 20° (*R* key, then type 20).
13. Finally, let's select the **hand.L** bone and rotate it -10° (*R* key, then type -10) to indicate the fact that it has a swinging movement following the arm, but a bit delayed.
14. For the right arm, let's start by selecting the **arm1.R** bone, rotate it 45° (*R* key, then type 45) and add a small rotation (around 10° is okay) to each one of the **arm2.R** and **hand.R** bones.

15. Let's now switch to **Front View (View → Front)** to check if the arms have separated too much from the waist (which is very likely) and adjust them by rotating the **arm1.L** and **arm1.R** bones to correct the extra rotation. Let's take a look at the following screenshot to check our progress:



At this point, we have completely defined the initial pose of our character.

16. Let's now switch to **Right View (View → Right)**, select all the bones (A key, once or twice) and go to **Pose → Insert Keyframe** and select **LocRotScale** from the list. This will insert some extra keyframes for the bones that we didn't pose, ensuring that this pose will occur exactly as we see it on frame 1, even after we add more poses later.

This initial pose must be repeated at frame 41, to really get the cycling effect.

17. With all the bones selected, let's locate the three buttons to the extreme right-hand side of the header of the 3D View and click on the one on the left to copy the current pose. Now let's go to frame 41 (press the up arrow key four times) and click on the button showing the white arrow coming out from the clipboard to paste the exact same pose.
18. Now go to frame 21 (press the down arrow key twice) and click on the rightmost button on the header of the 3D View to paste the mirrored pose. We can now go to the Timeline, click on the play button to get the basic animation playing, and rotate around the 3D View to see the "sliding" walk cycle that we have now.

Having created these three key poses of our character, we have completed this task. In the next one, we'll work on adding breakdown poses to improve the transition between key poses. Remember to save the file now.

Objective Complete - Mini Debriefing

In this task, we worked only on defining the three key poses for our animation. As we are creating a walk cycle, we just created a base pose on frame 1, copied it into the clipboard, and then pasted it into frame 41 and its mirror into frame 21.

The whole idea behind these key poses is that they define the action of our character. Even when lacking good breakdowns to go from one pose to the next, the action is clearly recognizable and starts taking shape. This way of organizing the process is quite useful, since it gives a clear progression from start to finish.

One of the nicest things that we did was using the smart paste of poses, including the mirrored paste, which depends on the correct naming of the bones; that's the reason we used the **.L** and **.R** suffixes on the bones when creating the armature.

Classified Intel

There are two basic methods for animating, the first is called "straight ahead" and the second one is "pose-to-pose". Each one has its advantages and disadvantages, of course, and it's also possible to work using some type of mix of the two.

The straight ahead method is about posing the character following the flow of time, in chronological order. This method allows the animator to improvise but has the disadvantage of not having the timing and the flow of action clear from the beginning, which can be risky and can easily get out of control.

The pose-to-pose method is a more structured and organized method, in which the first things that are defined are the key poses and the general timing of the action. The big advantages are the iterative nature that it has and the very controlled timing; the disadvantage is that the animator can't exercise the same level of freedom as when animating with the straight-ahead method.

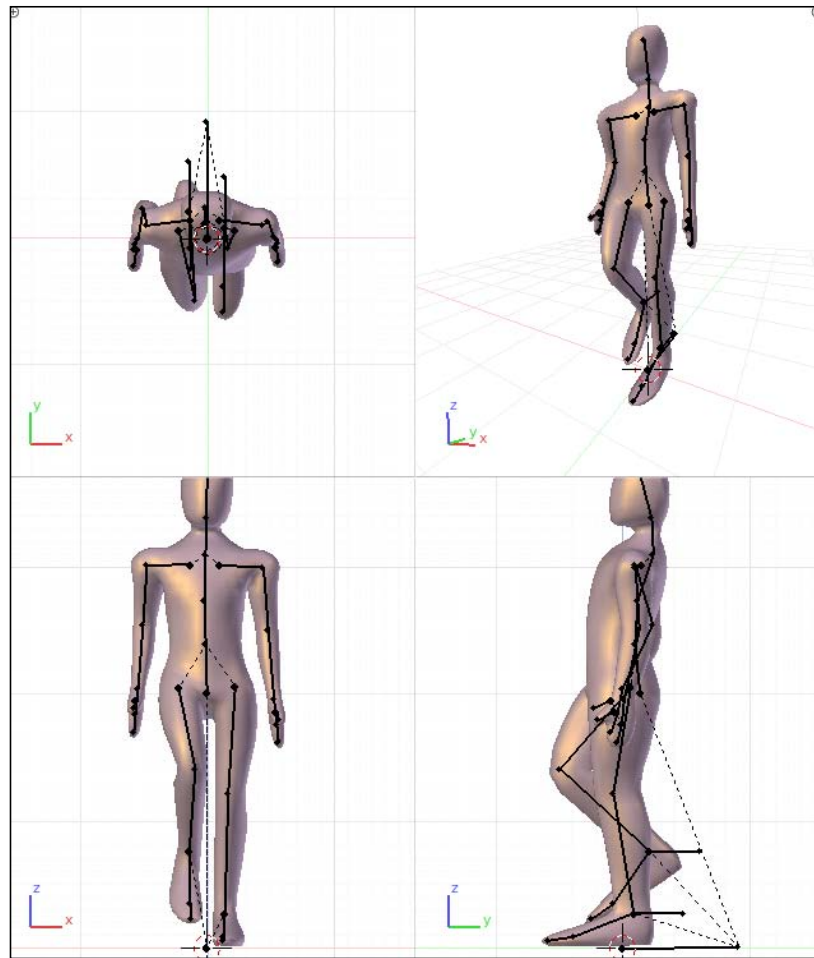
In this project, we are applying the pose-to-pose method, mostly because it allows us to have a clear idea of the process and the explanation follows a clear and methodical path.

Animation Breakdowns

For our animation to start having some life, we need to add a couple of poses between the three key poses created previously. These poses are called breakdowns because they mark the transition between two key poses.

Engage Thrusters

1. Let's go to frame 11 and start working on the breakdown pose.
2. Select the **spine1** bone and move it 0.17 units along the Z axis (G key, Z key, then type 0.17) so that the body of our character appears to be upright.
3. Continue by selecting the **foot1.L** bone, and clearing its rotation (**Pose** → **Clear Transform** → **Rotation**) to get it aligned horizontally. Let's also clear the rotation for the **foot2.L** bone.
4. Now let's select the **foot1.R** bone and move it 0.48 units on the Z axis (G key, Z key, then type 0.48) and 0.13 units on the Y axis (G key, Y key, then type 0.13).
5. Then select the **foot2.R** bone and clear its rotation (**Pose** → **Clear Transform** → **Rotation**).
6. For the hands, let's select the **hand.L** bone and rotate it 10° (R key, then type 10), and the **hand.R** bone and rotate it -5° (the R key, then type -5).
7. Now switch to **Front View (View** → **Front)**, select the **spine1** bone, and rotate it -2° (R key, then type -2). Continue by selecting the **spine3** bone and rotating it 3° (R key, then type 3), and finish by tweaking the arms by rotating **arm1.L** and **arm1.R** to get the arms close to the torso. This will give us the breakdown pose that we are looking for. Let's check the following screenshot for the finished breakdown pose:



8. With the pose done, let's select all the bones and go to **Pose → Insert Keyframe** and then select the **LocRotScale** option from the list.
9. Right after filling the entire animation channel with keyframes, let's copy the current pose to the clipboard (the icon with the arrow pointing inside the clipboard), go to frame 31 (press the up arrow key twice), and paste the mirrored pose (third button of the group).

And that's it. We now have two breakdown poses in between the three key poses. If we hit the play button in the Timeline, we can see that our animation is already looking like a walk cycle. Since the timing we chose was so long, the walk action must feel quite slow, but that can be changed by going to the **Render** tab of the **Properties Editor** and setting the FPS (Frames per Second) value of the **Dimensions** panel to a value around 30.

Objective Complete - Mini Debriefing

This task was very short: Just creating a base breakdown pose and then adding its mirrored duplicate on another frame of the walk cycle.

The purpose of the breakdown poses that we created is to create a believable transition between the key poses added initially. In this case, for the walk cycle we just needed to perform some minor tweaks from the default transition that Blender was generating.

Animation Tweaks

We'll work on adding a couple of tweaks to our animation and discuss a bit about further improvements that can be done but are not covered in this project.

Engage Thrusters

1. Let's set the frame of the scene to 6 and switch to **Right View (View → Right)**.
2. Now let's select the **spine1** bone and move it down -0.03 units (G key, Z key, then type -0.03) to give some emphasis to the lowering movement of the torso while going from the pose in frame 1 to the one in frame 11. As this tweak affects the left foot, let's make sure that it gets corrected to look horizontal, as it must be.
3. Now let's go to frame 16, select the **spine1** bone and move it 0.02 units along the Z axis (G key, Z key, then type 0.02); this will again create some emphasis on the movement of the torso, this time upwards.
4. Continue the tweaking by going to frame 26 and moving the **spine1** bone around -0.03 units along the Z axis (G key, Z key, then type -0.03).
5. Let's also apply the rising of the torso, this time on frame 36. Make sure to check if a foot gets an unwanted effect from the torso tweak and correct it appropriately.

At this point, we have quite a nice animation of our character having a nice in-place walk. The process was quite straightforward and Blender offered us everything we needed for creating the entire scene.

Objective Complete - Mini Debriefing

This last task was about adding a small touch of refinement to the animation of the torso; the idea behind this is that some bones of the armature must get extra attention and be further refined to improve the overall look of the action.

Classified Intel

Our animation looks good enough to consider it complete. There are however, lots of ways to improve it. Let's talk a bit about some procedures and tips that can be useful.

It's quite likely that if we check the movement of some elements, we will find them to describe a trajectory that doesn't look exactly right. In the case of feet, it's possible that the attachment to the floor isn't perfect and there is some bumping along the way. One possible solution to that kind of problem can be working on the curves (**Graph Editor**) for the bones that have the undesired movement and relaxing the corresponding curves.

One of the most important aspects of the animation of our character is the swinging of the arms and the way the movement seems to start in the torso and come along through the bones in the chain. We can watch our animation carefully to check how well that effect is achieved with our current animation and find spots that need to improve. For this kind of refinement there are two possibilities—adding extra keyframes to create new poses or offsetting the keyframes from the current poses. The first option has the disadvantage of creating too many keyframes to work with and the second one has the problem of disorganizing our very organized keyframes in the **Dope Sheet**.

A very important improvement would be to check that the trajectories described by the limbs are nice arcs, for this we can use the options available in the **Motion Paths** panel of the **Object Data** tab of the **Properties Editor** to have a visual representation of the paths and check the corrections that are applied. Arcs are a very important topic in animation, as they are the natural trajectories described by any limb of any living creature.

Finally, there's the possibility to exaggerate the poses of our character to give it a stronger personality. This part requires knowing about acting for animation, which is a complex topic by itself.

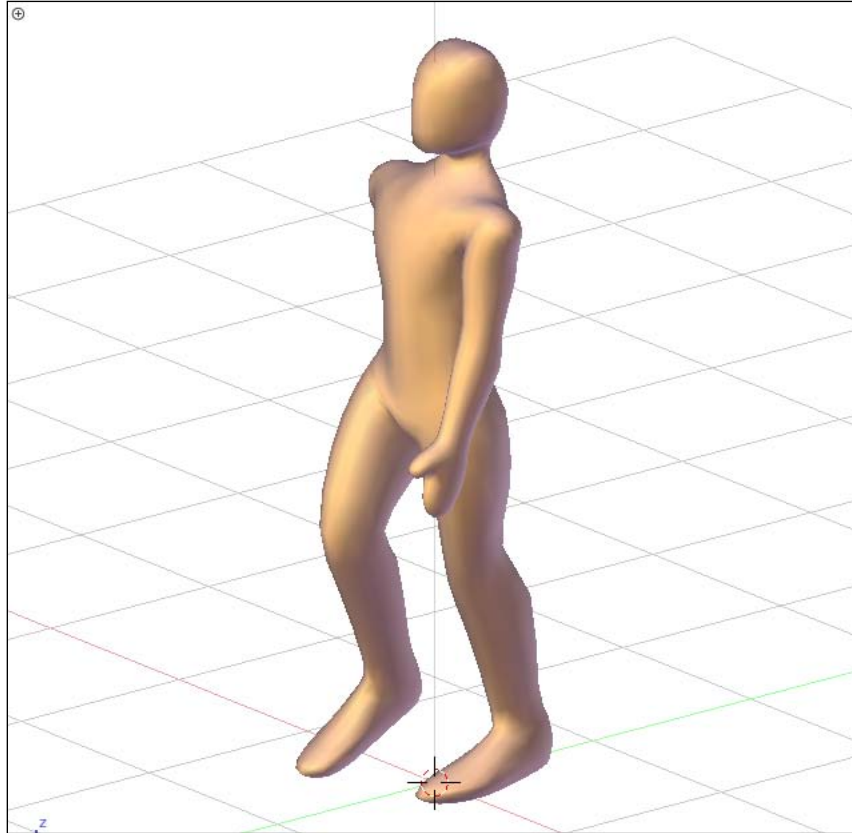


A very famous book on animation is the *Animator's Survival Kit*, written by Richard Williams. It's highly recommended by professional animators all around forums on the Internet.

Last but not least, the armature setup that we created is a very simple one, so it will definitely fall short when we start trying to do some complex poses with it. In the case of a real-world character, the setup is way more complex and requires a lot of constraints and configuration to create a powerful-yet-easy-to-use system. This task is accomplished by professional riggers, who know how to create complex mechanisms to give the animators a "puppet" that is very nice to use.

Mission Accomplished

This was quite a nice project. We started with the character created in another project and added a basic armature to it to get it ready for animation; we then proceeded with creating a basic walk cycle animation for our character, applying the pose-to-pose method, and getting a pretty good final result.



In the previous screenshot, we can see a frame taken from the final animation.

You Ready To Go Gung HO? A Hotshot Challenge

As said before, there are lots of ways to improve the result achieved in this project. Let's indicate a few:

- ▶ Get the character to perform his walk along a path, instead of doing it on the same place.
- ▶ Animate the head to give it a more natural and a less robotic look.
- ▶ Add more complex controls to the rig, such as a mechanism to control the orientation of the knees and elbows.
- ▶ The armature can be improved for the animator by using custom objects for the bones. This way, the animator can work with nice visual widgets, instead of the common shape of bones.
- ▶ A very good exercise would be to set up a project in which the character and the rig live in a library file and the actual animation is performed on another file using Blender's link mechanism to reuse the character and armature from the library file.

Project 8

Create a Snail

One of the nicest things that the computer graphics scientists have achieved is the possibility of mixing computer generated images with live footage. For this to be possible, there are a lot of aspects that must be correctly controlled and some quite sophisticated shading methods that must be used. Let's now work on creating a small snail and composite it over an empty leaf in real footage to get a good grasp of how complex a project of this kind can be.

Mission Briefing

We'll be creating a snail from scratch, setting the shaders for the shell and the body to have a realistic look, and then composite it over live footage in order to give it some nice place to live.

What Does It Do?

With this project, we are going to learn some very nice techniques and tricks of the trade to use in future 3D projects.

The snail's shell will be created using a basic method for doing parametric modeling, which will allow us to have a very high degree of control and, at the same time, avoid the complicated (and boring) issues of trying to do it by hand. We will then improve on the basic shape achieved with this technique by using some hand tweaking and the always-useful sculpting tools.

The shading part will require us to create a simple UV Unwrap from the shell mesh and learn to use the procedural textures in a quite sophisticated way, giving us very good results without having to resort to hand-painted textures that would require us to have a good deal of painting ability. We'll be using the **Subsurface Scattering** option a lot, which is available in Blender and very useful for creating realistic organic shaders.

Finally, we'll use a common trick to get shadows below the snail and make it very easy to composite over the live footage.

Why Is It Awesome?

The best parts of this project, besides the nice result in the end, will be the techniques that we're going to apply to get it done. We'll learn to apply a very easy, fast, and useful method to save time when trying to UV Unwrap a complex mesh, using the shape keys functionality of Blender. We are also going to use procedural textures applied according to the UV Map to get them nicely laid over the surface. Another nice aspect will be to apply the warp and stencil options for textures. This will give us a lot of control to create very complex shaders using only procedural textures.

All these techniques will give us very powerful tools to approach future projects and make our lives easier when solving some real life problems.

Your Hotshot Objectives

This project is separated into six tasks, as shown next:

1. Modeling the shell
2. Shading the shell
3. Modeling the body
4. Shading the body
5. Scene setup
6. Compositing

Mission Checklist

For this project, we're going to need a background image on which we will composite the final rendered snail. Let's go to www.flickr.com, locate the search box, and type "strawberry leaves after the rain close up" (without the quotes), then click on the thumbnail showing an image from the user **anyjazz65** (while clicking, make sure that the image ID 45115614 is shown in the URL to be visited). Once in the image page, let's choose the 1024x768 pixels version and save it as `bg.jpg` in the `tex` directory, inside the project's main directory.

Modeling the Shell

We'll be creating the shell shape first. For this task, we're going to apply a very nice technique for getting the main shape easily done and then proceed to perform some hand tweaking.

Prepare for Lift Off

Let's fire up Blender, delete the default cube from the scene, and save the file as `snail.blend` in the project's directory.

Engage Thrusters

1. To start with the modeling of the shell, let's switch to **Front View** (Numpad 1) and go to **View** → **View Persp/Ortho** (Numpad 5) to set it to **Orthographic**. Let's also make sure to have the 3D Cursor located at the origin of the 3D World (**View panel**, in the **Properties Sidebar**).
2. Then go to **Add** → **Mesh** → **Circle**, go to the **Tool Shelf** (**View** → **Tool Shelf**), and set the number of vertices to 10 and enable the **Align to View** option. Let's also go to **Object** → **Apply** → **Rotation** (*Ctrl* + *A* → **Rotation**) to ensure the object gets its rotation reset.
3. Before switching to **Edit Mode**, let's now change the name of the circle to shell (**Item panel**, in the **Properties Sidebar**).
4. Now let's switch to **Edit Mode**, select all the vertices (*A* key, once or twice) and move them 2.89 units along the X axis (*G* key, *X* key, then type 2.89).
5. The next step is to extrude the whole circle. Press *Escape* to keep the extrusion in place, then go to **Mesh** → **Vertices** → **Hooks** → **Hook to New Object** to create an empty object to use as the hook for the extruded vertices.
6. Let's now switch back to **Object Mode**, select the empty object added as the hook, and set its name to **empHook**.
7. Let's make sure the 3D Cursor is located at the origin of the 3D World; add a new Empty object and set its name to **empArray**.
8. Finally, let's select the **empHook** and **empArray** objects (in that order), go to **Object** → **Parent** → **Set** (*Ctrl* + *P*), and select the **Object** entry from the list.

At this point, we have the basic setup of objects from which the shell shape can be created very easily.

9. Let's now select the shell object (if it's not visible, switch the view shading to wireframe), go to the **Modifiers** tab of the **Properties Editor**, and add a new **Array Modifier**, right below the **Hook Modifier**. Let's now set the array modifier as follows:

- ▶ Fit Type: Fixed Count
 - ❑ Count: 36
 - ❑ Relative Offset: Disabled
 - ❑ Merge: Enabled
 - ❑ Object Offset: enabled
 - ❑ Offset Object: empArray (click on the entry with the yellow cube to show the selection list)

These options complete the setup for shaping the shell.

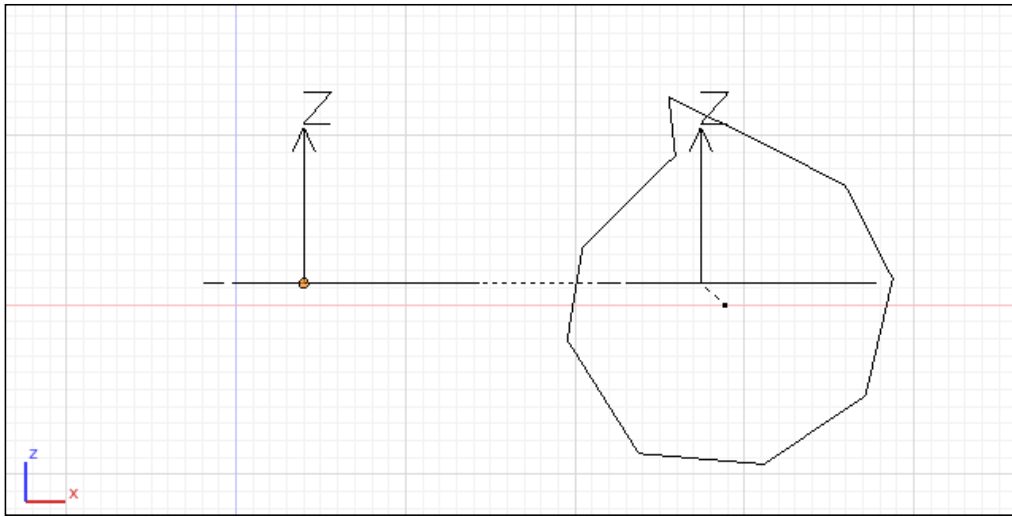
10. Now let's select the **empArray** object, go to the **Object** tab of the **Properties Editor**, locate the **Transform** panel, and set it as follows:

- ❑ Location: 0.403 X, -0.864 Y, 0.130 Z
- ❑ Rotation: 0° X, 0° Y, 28° Z
- ❑ Scale: 0.918 X, 0.918 Y, 0.918 Z

After applying these values, the shape of the shell will be created automatically, avoiding us lots of complicated issues when modeling it by hand.

Let's now perform a final tweak to the base mesh to give a nice touch to the shell.

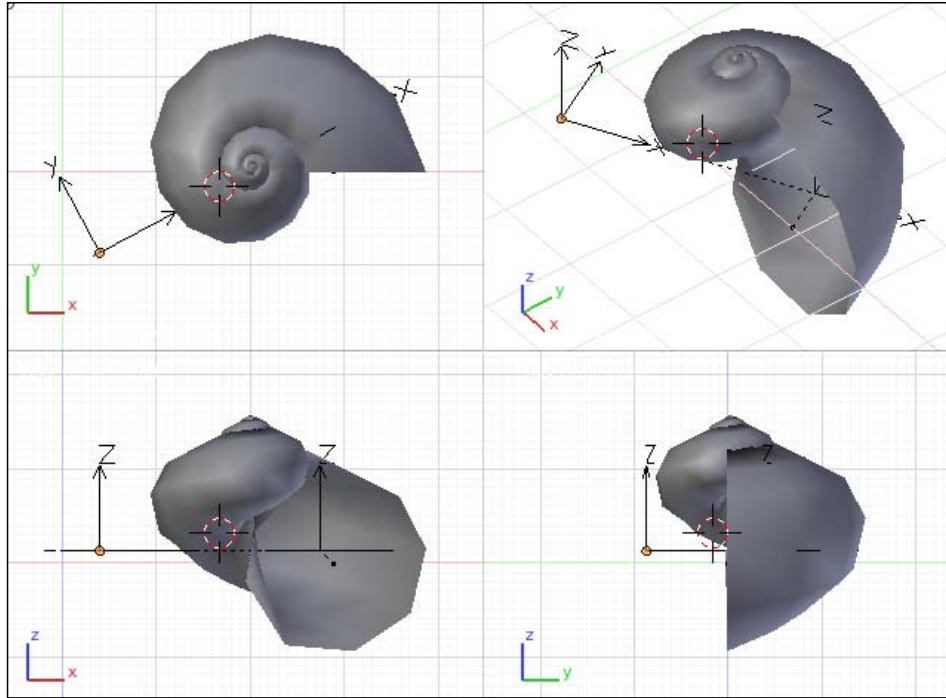
11. Select the shell object, go to the **Modifiers** tab of the **Properties Editor**, locate the eye icon in both the hook and array modifiers, and disable it.
12. Now let's switch to **Front View (View → Front)**, switch to **Edit Mode**, and shape it as shown in the next screenshot (remember to switch to wireframe mode and select the vertices using the **Border Select** tool, as we have extruded the ones right over the original ones):



The purpose of this modification is getting each successive ring in the spiral touching nicely the one it wraps around. It will be clearly visible when the **Subdivision Surface Modifier** gets applied, later in the project.

13. Now let's again enable the eye icon in the hook and array modifiers for the shell object to get the tweaked shell as shown.
14. It's also a good time to perform a correction on normals: Select all the vertices and go to **Mesh** → **Normals** → **Recalculate Outside** (*Ctrl + N*).

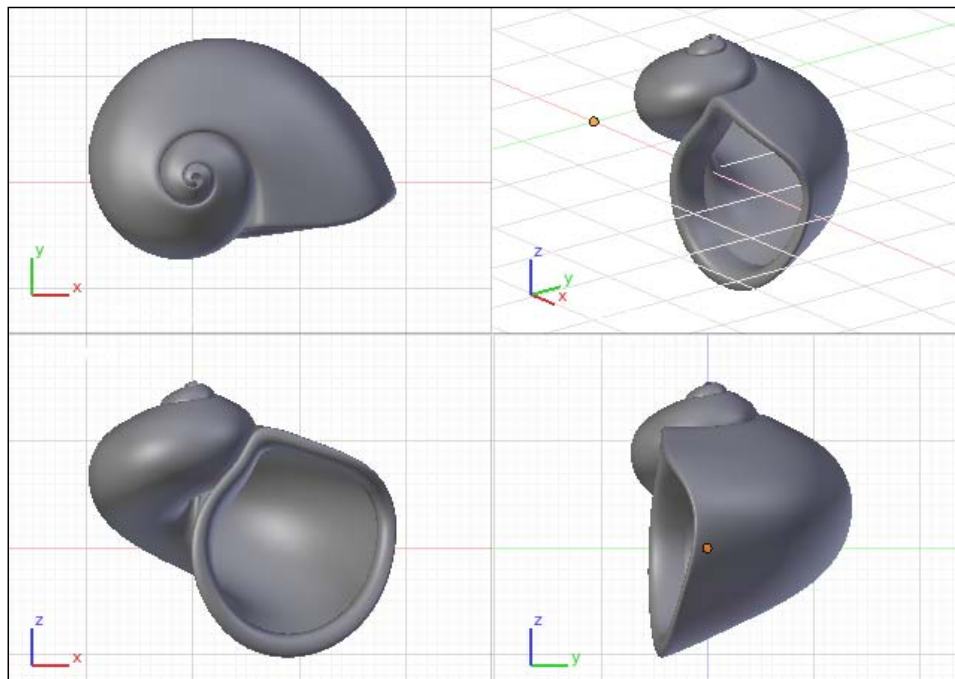
15. Now switch back to **Object Mode** and set the face shading to **Smooth (Tool Shelf Sidebar)** to get it looking smoothed. Let's take a look at the next screenshot showing the progress so far:



Once we have the basic shape ready, it's time to start tweaking it to get it finished.

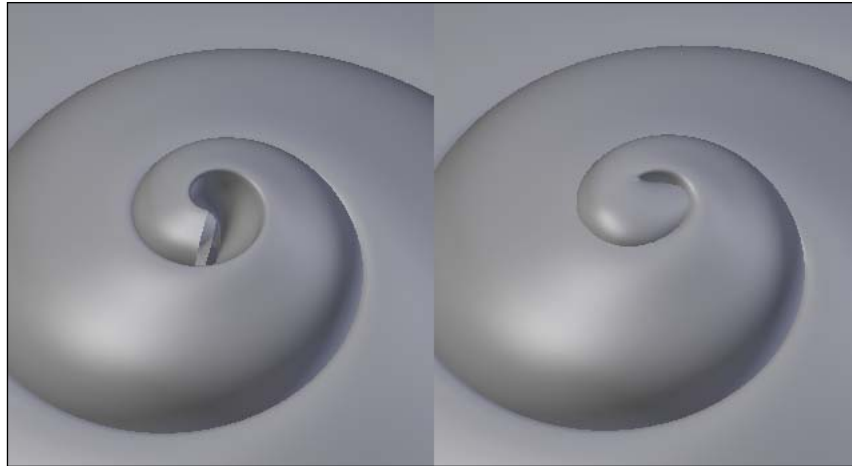
16. Let's make sure that we have the shell object selected, then go to the **Modifiers** tab of the **Properties Sidebar**, click on the **Apply** button of the **Hook modifier**, and then on the **Apply** button of the **Array modifier**. This way we get the modifiers converted into regular mesh on which we can work directly.
17. Now let's switch to **Edit Mode**, select the ring of vertices on the border of the wider part of the shell shape, switch to **Right View (View → Right)**, extrude the selection, press *Escape*, and then move the extrusion -0.4 units along the Y axis (G key, Y key, then type -0.4). Continue scaling the extrusion by a factor of 1.078 (S key, then type 1.078), and then move the extrusion 0.021 units along the X axis (G key, X key, then type 0.021) and -0.085 units along the Z axis (the G key, Z key, then type -0.085).

18. With the last extrusion still selected, let's now perform a second extrusion and move it -0.04 units along the Y axis (G key, Y key, then type -0.04). Continue by performing a third extrusion and scaling it down by a factor of 0.9 (S key, then type 0.9). Finally, let's perform a fourth extrusion, press *Escape*, and then move it 0.33 units along the Y axis (G key, Y key, then type 0.33).
19. Now switch to **Object Mode**, go to the **Tool Shelf** and set the face shading to **Smooth** to ensure the new faces have the correct shading set.
20. The second part of tweaking requires us to use the sculpt tools. As a preparation step, let's go to the **Modifiers** tab of the **Properties Editor** and add a **Subdivision Surfaces Modifier** to the shell object. Now let's switch to **Right View** (**View** → **Right** or press Numpad 3) and also switch to **Sculpt Mode** (same drop-down used for switching between **Object Mode** and **Edit Mode**).
21. Let's zoom out from the shell object, set the brush to **Grab** (G key), set the size of the brush at around 50% of the height of the aperture of the shell (F key to set the brush size), and tweak the aperture according to the next screenshot:



The purpose of performing this small deformation on the aperture of the shell is to get the body of the snail (to be added later) matching nicely with the shell.

Finally, we just need to work out the apex of the shell to get the final mesh that we are aiming at. Let's look at the next screenshot showing the final result, so that we know beforehand what we are trying to achieve:



22. With the shell selected, let's switch to **Edit Mode**. Now zoom in and pan to see the apex as it comes from the array modifier.
23. Select the first row of vertices running around the empty cone inside the first whorl and then scale them down until the sharp shape of the internal border of the first whorl looks soft.
24. Then let's select the ring of vertices forming the ring of the "tip"; continue by switching to **Top View (View → Top)**, then click in the very center of the apex to get the 3D Cursor located there, and set the Pivot Point to 3D Cursor (second drop-down to the right-hand side of the **Object Mode** drop-down). Now let's enable the **Proportional Editing Tool (Connected mode)** and set its type to **Smooth** to have some help in the tweaking that we're going to do. Rotate the selection -75° (the *R* key, then type -75) and adjust the **Proportional Editing Tool** influence (mouse wheel up/down), until the other rings affected shape a nice curve.
25. Right after that, switch to **Right View (View → Right)**, click on the left-hand side of the apex (close to it, but on empty space), and then apply a rotation of 40° (*R* key, then type 40). Finally, move the selection 0.09 units along the Y axis (*G* key, *Y* key, then type 0.09) to get the apex nicely closed. Don't forget to save the file now.

Objective Complete - Mini Debriefing

We started by setting up three objects to interact in such a way that the basic shape of the shell was created very easily by using the array modifier as a parametric modeling tool. We used two empties to control the set up: One empty worked as a hook to control the extrusion generated from the original circle and the second empty played two roles—one as the controller of the offset of the array modifier and one as the parent of the other empty. Together, they allowed the array modifier to do its magic.

After generating the basic shell shape, we continued by applying the two modifiers used in the setup (hook and array) to get a mesh in which we could tweak any vertex directly. Having that mesh, we continued working out some hand tweaking—extruding the aperture, tweaking it with a sculpting brush, and finally tweaking the apex to look nicely closed.

Shading the Shell

Now that our shell is fully modeled, we can work on shading it. For this task we're going to apply a couple of tricks that are not so obvious in Blender, which will allow us to get the task done very fast.

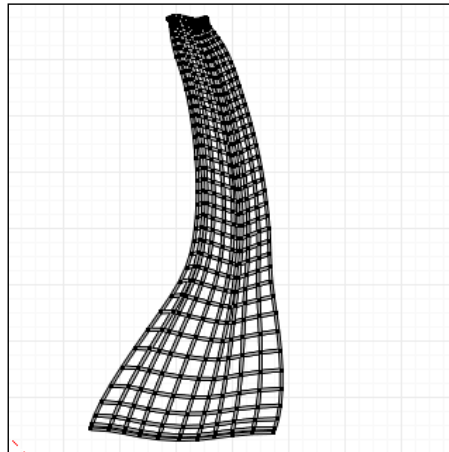
Engage Thrusters

1. Start by ripping a new editor below the 3D View and setting it to be a **UV/Image Editor**. Then go to the 3D View and switch to **Front View** (Numpad 1).
2. Now switch to **Edit Mode** (the shell object must be selected). Also, switch to **Edge Selection Mode** (*Ctrl + Tab*) and select the edge loop (*Alt + right-click*) of the edge running diagonally from the top-left-hand side to bottom-right-hand side that touches the first whorl from the aperture. An easy way to check that the selection is the correct one is by first checking in wireframe (*Z* key) if the edge loop runs from the aperture to the apex, then switching to solid shading (*Z* key), rotating the view around the object, and not being able to see it at all; only the edges in the aperture are visible.
3. Now let's go to **Mesh** → **Edges** → **Mark Seam** (*Ctrl + E* → **Mark Seam**) to set this edge loop as the seam that Blender will use to unwrap the mesh into its UV Map.

Now is time to apply the useful trick to avoid performing the boring manual tweaking of the UV Map that we're going to create.

4. Switch to **Object Mode** (*Tab* key), go to the **Object Data** tab of the **Properties Editor**, locate the **Shape Keys** panel, and click twice on the plus icon to the right-hand side of the empty list; this will add two entries to the list: **Basis** and **Key 1**. Select the **Key 1** entry from the list and then go to the 3D View to work on the mesh.

5. Switch to **Edit Mode** (*Tab* key) to work on the shell mesh, then select all the vertices (*A* key, once or twice), go to the **Tool Shelf**, apply the **Smooth Vertex** tool, and set its **Smooth Iterations** parameter to **100**. This way, we get a very thin, curly shape that resembles the shape of the shell.
6. Now let's go to **Mesh** → **UV Unwrap...** → **Unwrap** and then go to the **Tool Shelf** to set the **Method** parameter to **Conformal**. After this, we can take a look at the **UV/Image Editor** and see the unwrapped mesh.
7. Now let's go to the **UV/Image Editor**, select all the vertices (*A* key, once or twice), scale them by a factor of 3.5 along the X axis (*S* key, *X* key, then type 3.5), move them close to the center of the grid, and then scale them down by a factor of 0.95 (*S* key, then type 0.95). This will give us the final UV Map that we need for our shading process. Let's take a look at the next screenshot showing the finalized UV Map:



At this point, we have a nice and clean UV Map.

8. Let's switch to **Object Mode** (*Tab* key), go to the **Object Data** tab of the **Properties Editor**, make sure the **Key 1** item is selected (**Shape Keys** panel), delete it, then select the **Basis** item, and delete that as well.

Notice that the **UV Texture** panel now has an entry called **UVTex**, which represents the UV Map, created automatically when we unwrapped the mesh. As a final check, let's make sure that the shell shape is kept intact after deleting the shape keys, and the UV Map that we created is also preserved.

Now that we have the UV Map correctly set up, it's time to create the material for the shell of our snail.

9. Let's go to the **Material** tab of the **Properties Editor**, add a new material, set its name to **shell**, and set it as follows:

- ▶ Specular panel:
 - ❑ Color: 1.0 Red, 0.918 Green, 0.781 Blue
 - ❑ Specular Shader Model: Blinn
 - ❑ Intensity: 1
 - ❑ Hardness: 150
 - ❑ IOR: 10.0
- ▶ Subsurface Scattering panel: Enabled
 - ❑ IOR: 1.3
 - ❑ Scale: 0.1
 - ❑ Color: 0.987 Red, 0.955 Green, 0.76 Blue
 - ❑ RGB Radius: 15.028 Red, 4.664 Green, 2.541 Blue
 - ❑ Blend Color: 0.8
 - ❑ Blend Texture: 0.6
 - ❑ Scattering Weight Front: 1.0
 - ❑ Scattering Weight Back: 0.5
 - ❑ Error: 0.01

With the material settings in place, let's go to the **Texture** tab of the **Properties Editor** to add the textures for our material.

10. Start by adding a new texture, set its name to **distort**, its type to **Musgrave**, and set it as follows:
 - ▶ Musgrave panel:
 - ❑ Type: fBM
 - ❑ Lacunarity: 6.0
 - ❑ Size: 0.6
 - ▶ Size: 0.5 X, 0.5 Y, 0.5 Z (**Mapping** panel)

- ▶ Influence panel:
 - ❑ Color: Disabled
 - ❑ Normal: Enabled, -0.01
 - ❑ Warp: Enabled, 0.07
11. The first texture is ready; now let's go for the second one. Select the second slot from the textures list, add a new texture, set its name to **rings**, its type to **Marble**, and set it as follows:
- ▶ Marble panel:
 - ❑ Wave type: Tri
 - ❑ Size: 0.6
 - ❑ Depth: 4
 - ❑ Turbulence: 8.0
 - ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: UVTex
 - ❑ Size: 1.0 X, 32.0 Y, 1.0 Z
 - ▶ Influence panel:
 - ❑ Color: Enabled, 1.0
 - ❑ Normal: Enabled, 0.1
12. As this texture requires a ramp as well, let's go to the **Colors** panel, enable the **Ramp** option, change the **Interpolation** to **Ease**, and set it as follows (remember to use the **Add** button to insert new color stops):
- ▶ Color Stop 1:
 - ❑ Position: 0.0
 - ❑ Color: 0.044 Red, 0.008 Green, 0.008 Blue
 - ❑ Alpha: 0.97
 - ▶ Color Stop 2:
 - ❑ Position: 0.737
 - ❑ Color: 0.078 Red, 0.06 Green, 0.053 Blue
 - ❑ Alpha: 1.0

- ▶ Color Stop 3:
 - ❑ Position: 0.819
 - ❑ Color: 0.846 Red, 0.948 Green, 1.0 Blue
 - ❑ Alpha: 0.99
 - ▶ Color Stop 4:
 - ❑ Position: 0.936
 - ❑ Color: 0.965 Red, 1.0 Green, 0.769 Blue
 - ❑ Alpha: 0.98
 - ▶ Color Stop 5:
 - ❑ Position: 1.0
 - ❑ Color: 1.0 Red, 1.0 Green, 1.0 Blue
 - ❑ Alpha: 1.0
13. To add the next texture, select the third slot from the texture list, add a new texture, set its name to **marks**, its type to **Marble**, and set it as follows:
- ▶ Marble panel:
 - ❑ Size: 1.0
 - ❑ Depth: 6
 - ❑ Turbulence: 50.0
 - ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: UVTex
 - ❑ Size: 2.0 X, 6.0 Y, 2.50 Z
 - ▶ Influence panel:
 - ❑ Color: Enabled, 0.35
14. For this texture, we also need to add a color ramp. So let's go to the **Colors** panel, enable the **Ramp** option, change the **interpolation** to **Cardinal**, and set the ramp as follows:
- ▶ Color Stop 1:
 - ❑ Position: 0.0
 - ❑ Color: 0.044 Red, 0.024 Green, 0.009 Blue
 - ❑ Alpha: 1.0

- ▶ Color Stop 2:
 - ❑ Position: 1.0
 - ❑ Color: 0.383 Red, 0.117 Green, 0.022 Blue
 - ❑ Alpha: 0.65
- ▶ Color Stop 3:
 - ❑ Position: 1.0
 - ❑ Color: 1.0 Red, 0.636 Green, 0.204 Blue
 - ❑ Alpha: 0.4

15. Now let's add a fourth texture, set its name to **vblend**, its type to **Blend**, and set it as follows:

- ▶ Orientation: Vertical (**Blend** panel)
- ▶ Contrast: 5.0 (**Colors** panel)
- ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: UVTex
 - ❑ Offset: 0.0 X, -0.2 Y, 0.0 Z
- ▶ Influence panel:
 - ❑ Color: Disabled
 - ❑ Stencil: Enabled

16. The last texture to be added (fifth slot) must be named **rootNoise**, its type set to **Clouds**, and its configuration set as follows:

- ▶ Clouds panel:
 - ❑ Noise: Hard
 - ❑ Size: 0.6
 - ❑ Depth: 6
- ▶ Colors panel:
 - ❑ Brightness: 2.0
 - ❑ Contrast: 3.0

- ▶ Mapping panel:
 - ❑ Coordinates: UV
 - ❑ Layer: UVTex
- ▶ Influence panel:
 - ❑ Color: enabled, 1.0
 - ❑ Default Color: 0.022 Red, 0.0 Green, 0.002 Blue (This option is located below "RGB to intensity").

And that's all for the shell's material. This will give us the nice finished look that we are looking for, and we only used procedural textures! Now, let's save the file.

Objective Complete - Mini Debriefing

The shading process for the shell was done in three steps—creating a UV Map, setting the **Subsurface Scattering** options for the base material, and adding the five textures to finely control the surface's look.

In the first step, we started by defining the seam for UV unwrapping on an edge loop that runs along the whole "tube" that forms the shell. This edge loop was chosen in such a way that there are no visible discontinuities when rendering. We then continued by creating two shape keys—the basis to save the original state of the mesh and the **Key 1** to be edited into a "softer" mesh, which could be fed into the UV unwrapping tool of Blender to get a better UV map than using the original mesh shape. The trick was applying the **Smooth** tool to the **Key 1** shape key, calling the **UV Unwrap** tool using it, and then deleting all the shape keys. The reason to delete the **Key 1** shape key before the **Basis** was to ensure that the original shape of the mesh was preserved, using the UV map generated from the **Key 1** shape. This **UV Map** is needed to ensure the proper application of certain textures.

The second step was defining the basic material's properties. The most relevant aspect of this step was setting up the **Subsurface Scattering** options for our material, which enables a special kind of treatment of the object to get a "waxy" look.

The final step was just setting up five textures on the material, to have finer control over different aspects of it. The first texture was set up to warp the coordinates of the second one, which affects both color and the normals of the surface. The second texture uses the UV map's texture coordinates to ensure that the marble texture (producing just horizontal, deformed lines) would look like rings on the actual object. The third texture is used to add a soft color variation on top of the basic color provided by the second one. The fourth texture is used to apply a vertical stencil onto the next one. This stencil runs vertically and only allows the upper part (if seen on the UV map) of the fifth texture to affect the final material. The purpose this time is to add a saturated and dark red color only to the region of the shell close to the apex. The final texture just provided some color information with a bit of noise to be applied on the region that the fourth texture would allow it to affect.

Modeling the Body

The next task is creating the body for our snail. Now that we have the shell done, we can use it as a guide for getting the right proportions on the body, which allows us to work, with some freedom, on the modeling.

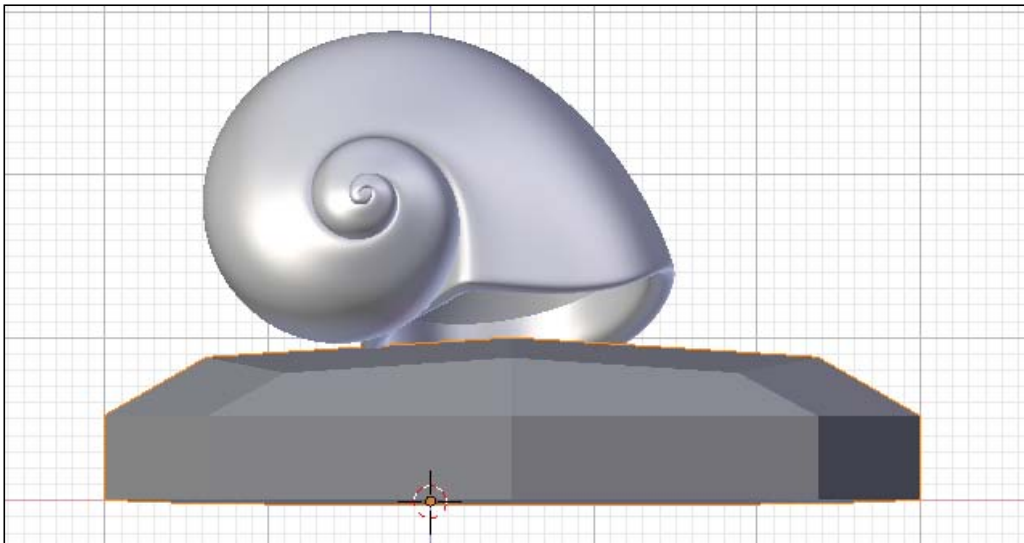
Prepare for Lift Off

Let's make sure that that shell object is in **Object Mode**, then switch to **Top View (View → Top)**, select it, and move it -2.42 units along the X Axis (G key, X key, then type "-2.42"); then switch to **Right View (View → Right)**, rotate it -75° (R key, then type "-75"), and then move it 0.8 units along the Z Axis (G key, Z key, then type "0.8"). Now we are ready to start working on the body.

Engage Thrusters

1. We'll start the body of the snail by adding a cube to our scene. Make sure the 3D Cursor is located at the origin of the 3D World (**View panel, Properties Sidebar**), then go to **Add → Mesh → Cube**, change its name to **body (Item panel, in the Properties Sidebar)**, and switch to **Edit Mode (Tab key)** and **Front View (View → Front** or press Numpad 1) to start working on it.
2. Deselect all the vertices of the cube and switch to wireframe shading (Z key), then use the **Border Select** tool (**Select → Border Select** or press B) to select the four lower vertices, and then move them 1 unit along the Z axis (G key, Z key, then type 1).
3. Continue by deselecting all the vertices (A key). Then, using the **Border Select** tool (**Select → Border Select** or press B), select the four vertices on the right, and then move the selection 2 units along the X Axis (G key, X key, then type 2).

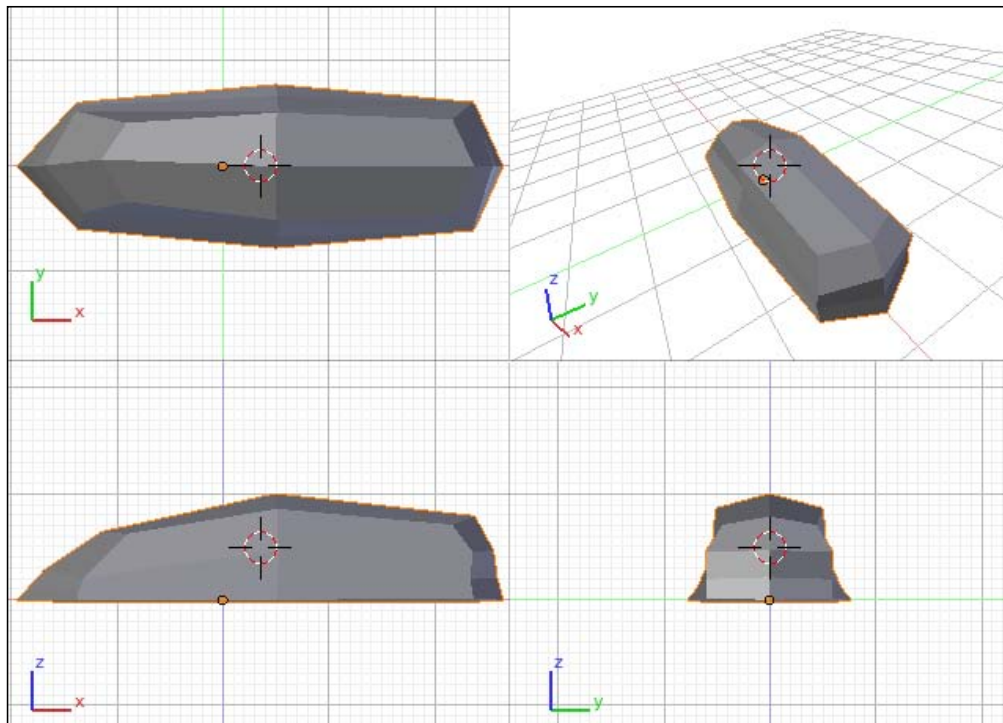
4. Now, deselect all the vertices (A key), select the four vertices on the left-hand side and move them -1 unit along the X axis (G key, X key, then type -1). Then switch to **Right View** (**View** → **Right** or press Numpad 3), select all the vertices (A key, once or twice), and scale the selection by a factor of 0.8 along the Y axis (S key, Y key, then type **0.8**).
5. Finally, let's use the **Loop Cut and Slide** tool to add a horizontal edge loop very close to the bottom face (**Edge Slide Percent** of **0.9** is okay) surrounding it.
This will give us the basic proportions for the body of the snail.
6. To continue working on the snail's body, let's go to the **Modifiers** tab of the **Properties Editor** and add a new **Subdivision Surfaces Modifier**, using a value of 1 for the **View Subdivisions** parameter. Then let's click on the **Apply** button of the modifier to get it converted into actual mesh data that we can work with directly.
7. Switch to **Edit Mode** (Tab key), go to **Right View** (**View** → **Right**), switch to wireframe (Z key), deselect all the vertices (A key), use the **Border Select** tool (**Select** → **Border Select** or press B) to select all the vertices living on the positive side of the Y axis (don't select those located at Y = 0), and then delete them (**Mesh** → **Delete, Vertices**).
8. Now add a **Mirror** modifier (**Modifiers** tab of the **Properties Editor**), deselect the **X** option, enable the **Y** option, and enable the **Clipping** option.
9. Now let's switch to solid shading (Z key) and check that the body of the snail looks as shown in the following screenshot (**Front View**):



10. Having the basic shape in place, we can start working on the fine tweaking for it. Let's switch to **Sculpt Mode** and set the brush to **Grab** (G key) to get on it.
11. Let's work first on the foot (tail) of the snail. Zoom in to get to see only the leftmost part of the body. Let's also reduce the size of the brush so that it is possible to pick individual vertices from the mesh with it (we don't want to affect the lower part of the body, which must remain entirely flat) and work on it so that it looks like a tail from **Front View**.
12. Let's now switch to **Left View** (**View** → **Left** or press *Ctrl* + Numpad 3), rotate the view a bit to get to see the foot downwards, and work on it to break the "boxy" look of the foot and shape it into something more resembling a tail. Be careful, though, and avoid getting it too thin.
13. Next, let's work on the middle part of the body. Switch to **Right View** (**View** → **Right**), rotate the view a bit to the right to get to see the side of the body, and work on "carving" a bit on the side of the snail. Use the grab brush, this time, let's just move the only vertex on the side inside the body, to get it looking a bit sunk.

The last part of this round of tweaks is working on the head (right part, from **Front View**).
14. Let's switch to **Top View** (**View** → **Top**) and work on the rightmost part of the body to get it having a flat look; start by moving the vertices on the mirror plane to the left to align them more with the other vertices. Then go to **Front View** and tweak around the vertices on the side to improve the shape.

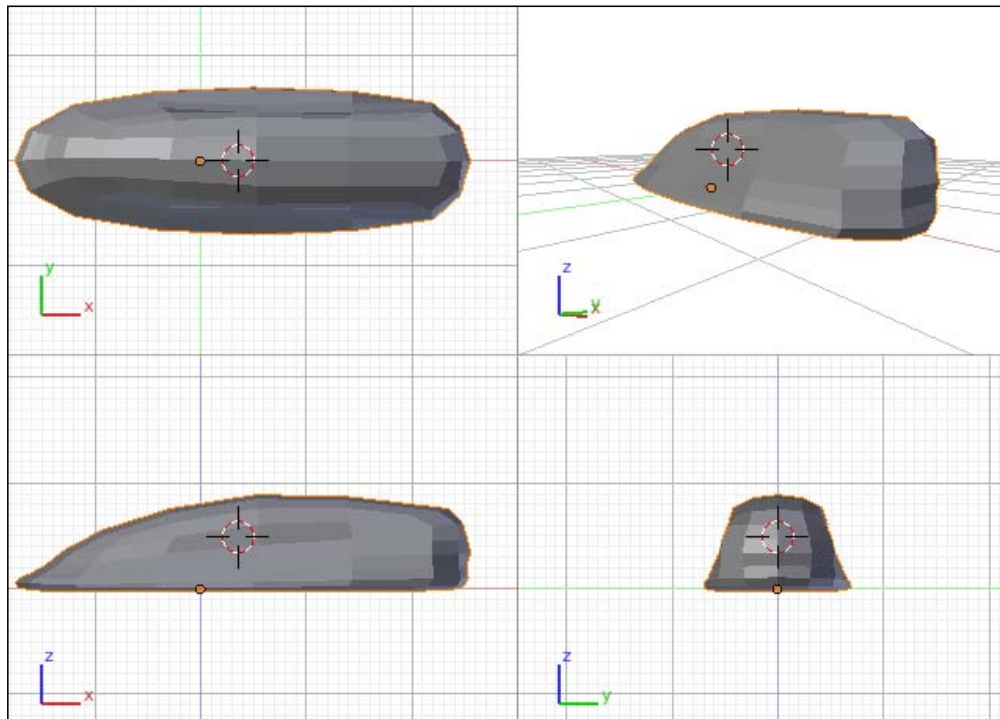
Let's now take a look at how the body should look by now (the shell can be hidden by selecting it and pressing *H*; to get it back, press *Alt* + *H*):



15. The finest kind of detailing that we could get from the low resolution mesh has been achieved, so it's time to add a new **Subdivision Surfaces Modifier (Modifiers** tab, in the **Properties Editor**) to the body object, set its **View Subdivisions** to **1**, click on **Apply**, and start performing some more tweaks with the more detailed mesh that we now have available.

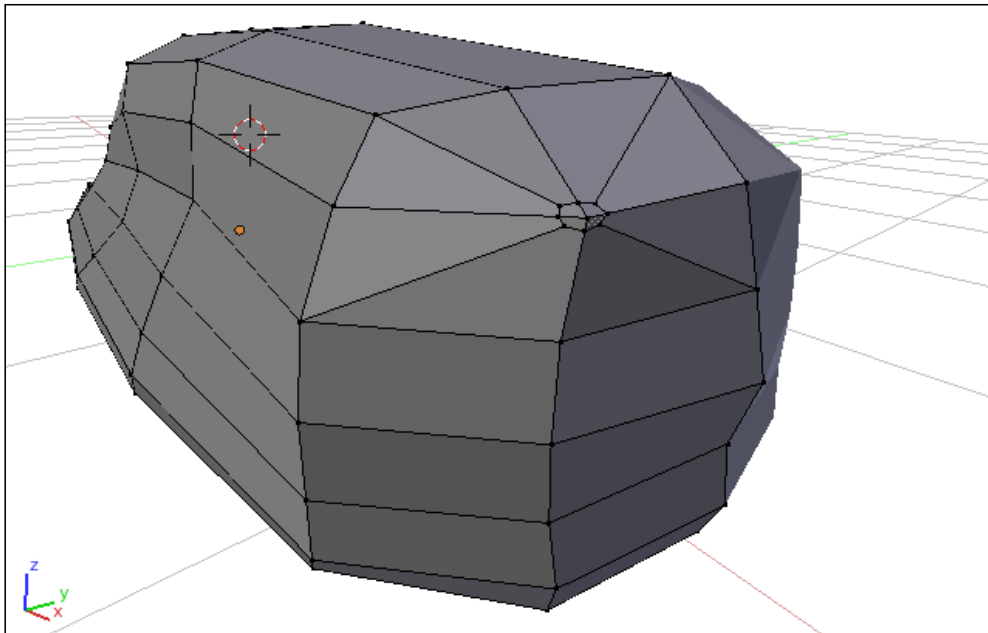
The second round of tweaks will be just about the same as the first one, with the exception that this time we must concentrate on sculpting out some more fine detail.

16. The first part will be to work on the foot (tail), to get a stronger curve down from the upper part of the body into the thick border that the snail has, because of the way it attaches to surfaces; then we work out adding some more curvature to the sunk part of the middle of the body. Finally, we just refine some more the flat aspect that the head must have. A screenshot is here to help us know where we are heading to:



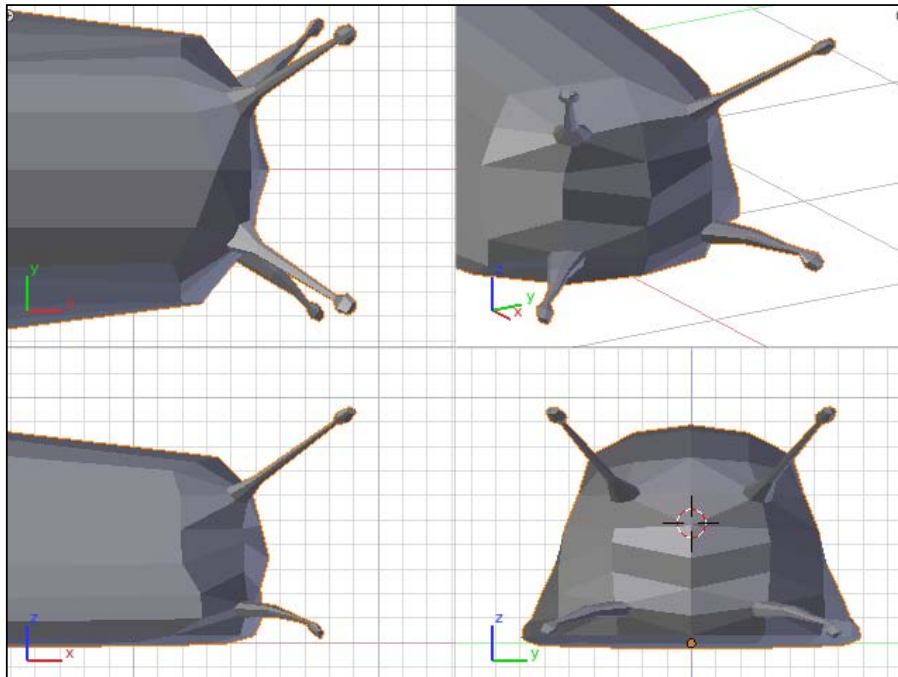
This will get us the finished shape for the body. Now we just need to go into modeling the eyestalks and the tentacles.

17. To model the eyestalks, we must switch to **Right View (View → Right** or press Numpad 3), select the four edges surrounding the vertex on the top-left-hand corner of the group of faces in the center of the face, then use the **Subdivide** tool and tweak the newly created vertices to form a circular shape around the interior vertex. Let's take a look at the following screenshot to see the desired result:



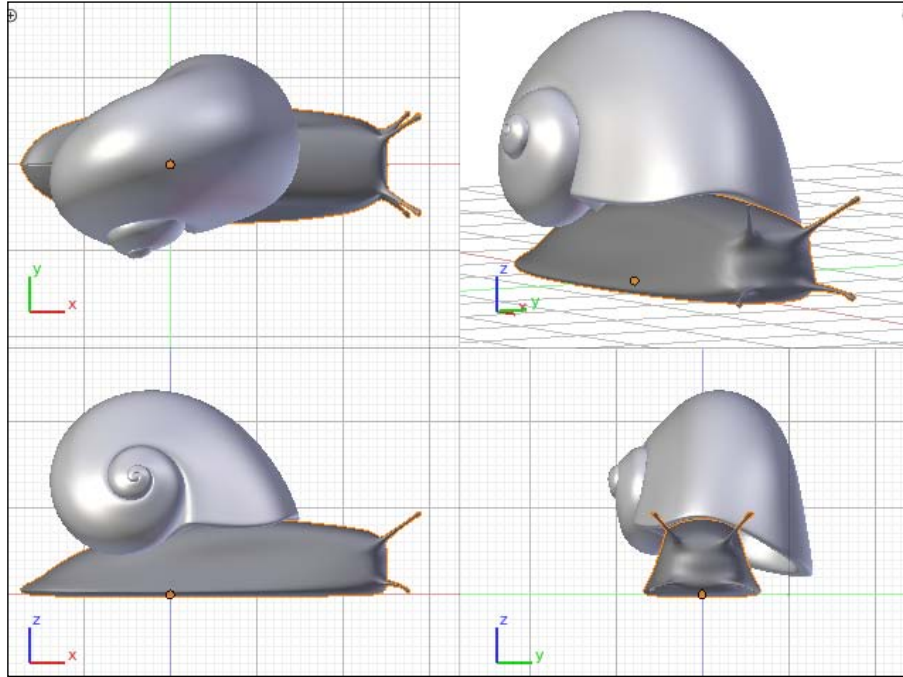
18. Now we must select the four faces inside the circular shape we just created, go to **Front View** (**View** → **Front** or press Numpad 1), switch to **Face Selection Mode** (**Ctrl** + **Tab**, **Face**), and perform some successive extrusions. The first extrusion must be moved 0.15 units along the normal (don't press *Esc* after extruding, just move it along the normal). The second extrusion must be moved 0.5 units along the normal, the third extrusion must be moved 0.04 units along the normal, and finally, the fourth extrusion must be moved 0.04 units along the normal.
19. With the extrusions in place, let's select the first ring (the one we created on the body) and scale it up to properly shape the eyestalk, then continue with the second ring, leave the third unscaled, and scale the fourth and fifth to get a nice ball in the tip of the eyestalk, for the eye itself. The purpose is to get a nice, curvy, shape for the eyestalks.

20. For the tentacles, we just need to perform the very same steps that we did for the eyestalks, but this time on the fourth vertex coming down from the one around which we created the base ring for the eyestalk. Be careful to get the tentacles being shorter than the eyestalks; around half the length would be okay. Again, a screenshot can be very useful to ensure that we are on the right track:



21. After doing this, let's switch to **Object Mode** (*Tab* key), add a **Subdivision Surfaces Modifier** to the body object, and set its **View Subdivisions** parameter to 2 (do not click on **Apply**). Then switch to **Right View** (Numpad 3), select the shell object (if it's hidden, press *Alt + H*), and move it down and to the right to get it correctly placed over the body of the snail.

And that's it for the modeling of the body; let's check the next screenshot showing us the progress so far:



Objective Complete - Mini Debriefing

The modeling of the body was accomplished in three steps: Creating the base mesh, two rounds of tweaks using the grab brush (**Sculpt Mode**), and adding the eyestalks and tentacles.

We started by creating a simple cube, adjusting the proportions using the shell as helper, then added an edge loop around the bottom face, and then applied a **Subdivision Surfaces Modifier** with **1** level of subdivision to get some low-detailed mesh from which to start the next step.

The second step was about taking the base mesh, using the grab brush to perform a first round of tweaks to get the general shape done, then applying a **Subdivision Surfaces Modifier** with level **1** to increase the level of detail of the mesh, and performing a second round of tweaks with the grab brush to improve the shape of the body and get it finished.

For the eyestalks and tentacles, we just added rings around vertices in the head, followed by some successive extrusions and scalings to get them shaped correctly.

Shading the Body

Now let's finalize the body by adding a proper material to it. We'll again be using the **Subsurface Scattering** option, along with some fine detailing by using textures.

Engage Thrusters

1. Let's make sure that we are working in **Object Mode** and select the body object.
2. Then go to the **Material** tab of the **Properties Editor**, add a new material, change its name to **body**, and set it as follows:
 - ▶ Specular panel:
 - ❑ Color: 1.0 Red, 1.0 Green, 1.0 Blue
 - ❑ Specular Shader Model: Blinn
 - ❑ Intensity: 0.715
 - ❑ Hardness: 37
 - ❑ IOR: 10.0
 - ▶ Ambient: 0.0 (**Shading** panel)
 - ▶ Subsurface Scattering panel: Enabled
 - ❑ IOR: 1.3
 - ❑ Scale: 0.015
 - ❑ Scattering Color: 0.452 Red, 0.446 Green, 0.365 Blue
 - ❑ RGB Radius: 14.266 Red, 12.0 Green, 2.036 Blue
 - ❑ Blend Color: 0.9
 - ❑ Blend Texture: 0.5
 - ❑ Scattering Weight Front: 1.0
 - ❑ Scattering Weight Back: 1.0
 - ❑ Error: 0.01
3. After setting the material, let's go to the **Texture** tab of the **Properties Editor**, add a new texture, change its name to **distort-bump**, its type to **Clouds**, and set it as follows:
 - ▶ Influence panel:
 - ❑ Color: Disabled
 - ❑ Normal: Enabled, 0.171

-
- ❑ Warp: Enabled, 0.3
4. Now select the second slot in the textures list, add a new texture, change its name to **bump**, its type to **Voronoi**, and set it as follows:
 - ▶ Noise Size: 0.15 (**Voronoi** panel)
 - ▶ Size: 15.0 X, 3.0 Y, 3.0 Z (**Mapping** panel)
 - ▶ Influence panel:
 - ❑ Color: enabled, 1.0
 - ❑ Normal: enabled, 0.5
 5. Let's also go to the **Colors** panel, enable the **Ramp** option, set the **Interpolation** to **Ease**, and set the ramp as follows:
 - ▶ Color Stop 1:
 - ❑ Position: 0.016
 - ❑ Color: 0.424 Red, 0.439 Green, 0.398 Blue
 - ❑ Alpha: 1.0
 - ▶ Color Stop 2:
 - ❑ Position: 1.0
 - ❑ Color: 1.0 Red, 1.0 Green, 1.0 Blue
 - ❑ Alpha: 0.0
 6. Let's continue by adding a third texture; change its name to **spots**, its type to **Voronoi**, and set it as follows:
 - ▶ Voronoi panel:
 - ❑ Coloring Intensity: 0.91
 - ❑ Noise Size: 0.15
 - ❑ Feature Weights 1: 1.0
 - ❑ Feature Weights 2: -0.111
 - ❑ Feature Weights 3: 0.194
 - ❑ Feature Weights 4: -0.084
 - ▶ Size: 15.0 X, 3.0 Y, 3.0 Z (**Mapping** panel)
 - ▶ Color: enabled, 0.589 (**Influence** panel)

7. This texture also needs us to add a color ramp; let's go to the **Colors** panel, enable the **Ramp** option, set the **Interpolation** type to **Ease**, and set the ramp as follows:
 - ▶ Color Stop 1:
 - ❑ Position: 0.0
 - ❑ Color: 0.094 Red, 0.098 Green, 0.087 Blue
 - ❑ Alpha: 0.86
 - ▶ Color Stop 2:
 - ❑ Position: 0.647
 - ❑ Color: 1.0 Red, 1.0 Green, 1.0 Blue
 - ❑ Alpha: 0.0
8. The fourth texture must be named **blend**, its type set to **Blend**, and we must set it as follows:
 - ▶ Blend panel:
 - ❑ Progression: Easing
 - ❑ Orientation: Vertical
 - ▶ Contrast: 5.0 (**Colors** panel)
 - ▶ Mapping panel:
 - ❑ X Mapping: X
 - ❑ Y Mapping: Z
 - ❑ Z Mapping: Y
 - ▶ Influence panel:
 - ❑ Color: Disabled
 - ❑ Stencil: Enabled
9. Finally, let's add a fifth texture, change its name to **diffuse**, its type to **Musgrave** and set it as follows:
 - ▶ Mapping panel:
 - ❑ Offset: 0.0 X, 1.0 Y, 0.0 Z
 - ❑ Size: 0.1 X, 0.1 Y, 0.1 Z
 - ▶ Color: Enabled, 0.895 (**Influence** panel)

10. Once again, we must add a color ramp; let's go to the **Colors** panel, enable the **Ramp** option, set the **Interpolation** to **B-Spline**, and set the ramp as follows:

- ▶ Color Stop 1:
 - ❑ Position: 0.412
 - ❑ Color: 0.036 Red, 0.046 Green, 0.037 Blue
 - ❑ Alpha: 1.0
- ▶ Color Stop 2:
 - ❑ Position: 1.0
 - ❑ Color: 0.042 Red, 0.018 Green, 0.018
 - ❑ Alpha: 0.0

Now we have the material for the snail's body finished and ready to be used in the final render. Don't forget to save the file.

Objective Complete - Mini Debriefing

The shading of the body required the standard procedure, which is defining some basic properties at the material's level and then layering multiple textures to add some detail to it.

The most important aspect of the general properties of the material is the configuration for the **Subsurface Scattering** property. As the body of the snail is an organic material, it needs to have a "fleshy" look, which is accomplished by enabling this property.

The first two textures in the texture stack affect the normals of the surface, thus giving the illusion of roughness. The first texture's job is just to slightly deform the second one, which is the one used to drive the normal.

The third texture in the stack is used to provide some small grains of color over the surface already "deformed" by the first and second textures.

The fourth texture acts as a stencil (mask) to control the region that the fifth texture will affect. The result is the dark color in the upper part of the body of the snail.

Classified Intel

We have been using two properties of the textures that, although they appear like any other common property, provide us with very powerful tools to get the normal procedural textures behaving in very nice and powerful ways.

The first parameter is **warp**. The job of this parameter is to use the information provided by a texture as a "deformation" to be applied onto the next texture on the stack. The actual technicalities behind it are a bit complicated and we're not going to discuss them, but the idea is that the white areas of the warping texture will cause the most deformation (proportional to the value of the warp parameter), while the black areas won't provide deformation.

The second parameter is **stencil**. This parameter allows a texture to be used as a mask to filter which areas of the surface will be affected by the information provided by the next texture on the stack; the white areas of the stencil texture will allow full influence of the next texture, while the black areas will filter it out entirely (thus muting the effect).

Scene Setup

The only remaining tasks that we have to do now are adding a plane for our snail, shaping it according to the surface on which it will be composited on the real footage, and adjusting the camera properly.

Engage Thrusters

We'll start by adding a curve to deform the body of the snail.

1. Let's switch to **Front View (View → Front)**; make sure the 3D Cursor is located at the origin of the 3D World (**View panel**, in the **Properties Sidebar**) and the view is in **Orthographic** mode (**View → View Persp/Ortho** or press Numpad 5).
2. Go to **Add → Curve → Bezier** to create the curve; with the curve selected, let's switch to **Edit Mode (Tab key)**, switch to **Right View (View → Right** or press Numpad 3), set the view to wireframe (Z key), select all the control points (A key, once or twice), and rotate them 90° (R key, then type 90). Then switch back to **Front View (View → Front** or press Numpad 1).
3. To shape the curve according to our needs, let's select the leftmost control point and move it -0.42 units along the X axis (G key, X key, then type -0.42). Now select the rightmost control point, move it -0.31 units along the X axis and 0.29 units along the Z axis (G key, type -0.31, press Tab twice, then type 0.29).
4. Now let's extrude the rightmost control point and move the extrusion 1.64 units along the X axis and -0.15 units along the Z axis (G key, type 1.64, press Tab twice, then type -0.15); continue by scaling down the handlers by a factor of 0.565 (S key, then type 0.565) and then rotate it by 5° (R key, then type 5).
5. Finally, let's select the left handler of the middle control point and move it -0.078 units along the X axis and 0.052 units along the Z axis (G key, type -0.078, press Tab twice, then type 0.052). This will get us the needed curve shape.

6. Let's now switch to **Object Mode** (*Tab* key) and change the name of the curve to "deformer".

The curve that we just created is needed to deform the body of the snail.

7. To use it as intended, let's select the body object, then go to the **Modifiers** tab of the **Properties Editor**, add a new **Curve Modifier**, click on the **Object** parameter entry, and select the deformer object from the list. We also need to go to the 3D View, select the deformer object and move it -1.1 units along the X axis (*G* key, *X* key, then type -1.1).

As the snail must project a shadow to get it to look correct over the leaf in the real footage, we need to create a plane below the snail.

8. Let's make sure the 3D Cursor is located at the origin of the 3D World (**View panel, Properties Sidebar**), go to **Add** → **Mesh** → **Plane** (*Shift + A*, **Mesh** → **Plane**), switch to **Edit Mode** (*Tab* key), and scale it (all the vertices must be selected) by a factor of 5.5 (*S* key, then type 5.5).
9. Now switch back to **Object Mode** (*Tab* key), go to the **Modifiers** tab of the **Properties Editor** and add two modifiers: **Subdivision Surfaces** and **Curve**. On the **Subdivision Surfaces** modifier, change its type to **Simple** and set the **Subdivision Levels** to **6** both for **View** and for **Render**; in the **Curve** modifier, let's use the "deformer" object as the **Object** parameter.
10. Now go to the **Material** tab of the **Properties Editor**, add a new material, change its name to "shadow" and enable the **Shadows Only** option (**Shadow** panel).
11. Finally, let's go to the 3D View and move the plane -0.2 units along the Z axis (*G* key, *Z* key, then type -0.2).
12. Next, we're going to adjust the camera. Select it, go to the **Object** tab of the **Properties Editor**, locate the **Transform** panel, and set it as follows:
 - ▶ Location: -5.0 X, -19.31 Y, 19.52 Z
 - ▶ Rotation: 37.023° X, 25.946° Y, -51.775° Z



The camera parameters are given explicitly, but the process to get them was done by using the camera fly mode (camera view, then press *Shift + F*). The camera fly mode allows us to interactively move the camera, by driving it with the scroll wheel (back and forth) and controlling the trajectory by moving the mouse pointer.

13. We also need to set up the lamp correctly. Let's add a lamp (in case there is none). Go to the **Object** tab of the **Properties Editor** and set its location as 4.08 X, 1.0 Y, 5.9 Z (**Transform** panel); then let's go to the **Object Data** tab and set it as follows:

- ▶ Lamp panel:
 - ❑ Type: Point
 - ❑ Color: 1.0 Red, 1.0 Green, 1.0 Blue
 - ❑ Energy: 1.0
 - ❑ Distance: 30.0
 - ❑ Specular: Enabled
 - ❑ Diffuse: Disabled
- ▶ Shadow panel:
 - ❑ Method: Ray Shadow
 - ❑ Samples: 16
 - ❑ Soft Size: 2.0
 - ❑ Ray Sample Method: Adaptive QMC

14. Let's continue by configuring the **World** settings; let's go to the **World** tab of the **Properties Editor** and set it as follows:

- ▶ Horizon Color: 0.046 Red, 0.022 Green, 0.05 Blue (**World** panel)
- ▶ Ambient Occlusion panel: Enabled
 - ❑ Factor: 0.9
 - ❑ Mode: Add
- ▶ Gather panel:
 - ❑ Gather Method: Raytrace
 - ❑ Distance: 1.0
 - ❑ Sample Method: Adaptive QMC
 - ❑ Samples: 8

15. Finally, let's go to the **Render** tab of the **Properties Editor** to adjust the **Resolution** of the image (**Dimensions** panel) to 1024 X, 768 Y and 100%. Let's also enable the **Anti-Aliasing** panel and set the samples to 5.

That's it for the scene setup. Let's save the file now, before continuing with the compositing task.

Objective Complete - Mini Debriefing

In this task, we performed various adjustments and additions to the scene to get it ready for compositing. We started by adding a curve to deform the shape of the body of the snail via a modifier, followed by adding a plane below the snail and setting its material to only render the shadow (not the color) to make it easy to compose the render over the live footage. We then proceeded to configure the camera, to get the right framing for the shot, and the lamp, to get it affecting only the specular component of the shader and casting some shadows. Then we worked on the world settings to add some ambient occlusion (diffuse lighting) and set its quality. Finally, we adjusted the resolution of the render to play nicely with the live footage over which we are going to put our snail.

Compositing

To finish our project, we only need to perform some basic compositing to get the rendered snail nicely integrated into the live footage used as the background.

Engage Thrusters

1. Let's switch to the **Compositing** screen to get a **Node Editor** available.
2. Now go to the **Node Editor**, select the **Compositing Nodes** option from the type of setup that we are going to create, and enable the **Use Nodes** checkbox to start creating the setup.
The default setup comes with two nodes—a **Render Layers** node directly connected to a **Composite** node.
3. Let's hold *Ctrl* and click-and-drag with the left mouse button cutting the curve connecting them in order to separate them.
4. Let's now go to **Add** → **Input** → **Image**, then click on the **Open** button of the **Image** node and load the `bg.jpg` image; let's put this node below the **Render Layers** node.
5. Now go to **Add** → **Color** → **AlphaOver**, move this node between the **Render Layers** node and the **Composite** node and enable the **Convert Premul** option.
6. Then connect the **Image** output from the **Render Layers** node into the lower **Image** input socket of the **AlphaOver** node and the **Image** output from the **Image** node to the upper image input socket of the **AlphaOver** node. Finally, let's connect the **Image** output from the **AlphaOver** node to the **Composite** node. Let's also check that the **Fac** parameter of the **AlphaOver** node is **1.0**.

Now we can press *F12* to perform the render of our scene and get to see the nice result of all the hard work we've put into it. Don't forget to save the file and the rendered image.

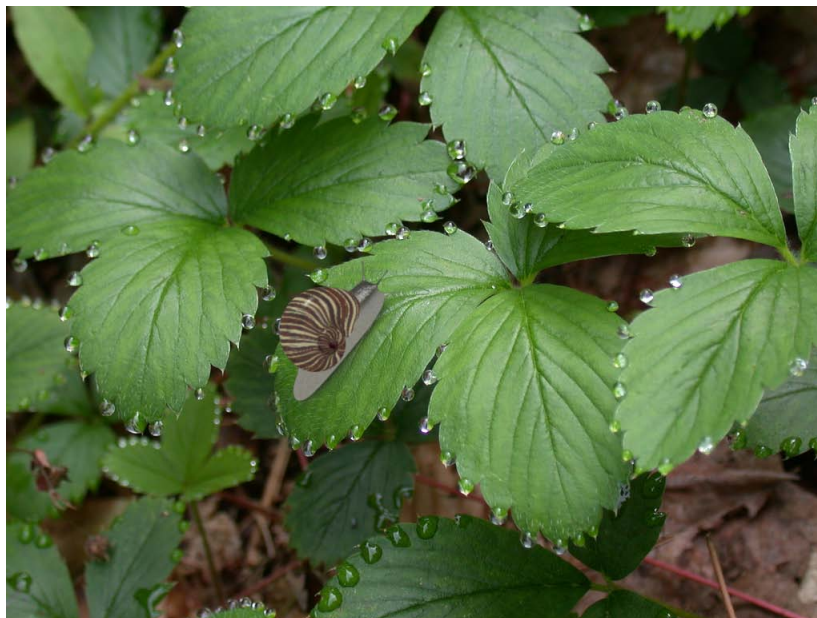
Objective Complete - Mini Debriefing

This was a very straightforward task—we just created a basic node setup mixing two sources: the snail from the 3D scene and the live footage that we downloaded in the preparation part of the project.

This time we used the **AlphaOver** node, which allows us to get an image composited over another, using the Alpha channel to decide where to let the lower image to show through. The **Convert Premul** option is needed to get the image converted into a more suitable format for this kind of mixing.

Mission Accomplished

There we have it, a very nice snail composited over a leaf from live footage. Let's check our final image, as shown in the following screenshot:



Our lovely snail is going to enjoy eating all those leaves!

You Ready To Go Gung HO? A Hotshot Challenge

Let's explore some ideas of how we can improve the result:

- ▶ Using the sculpting tools available in Blender, we can work out the roughness of the skin of the snail to replace the basic procedural bump-map that we applied.
- ▶ The lighting on the snail would look way better if we get the lower part looking darker, to better match the global lighting conditions of the scene.
- ▶ As the snail is surrounded entirely by leaves, we can improve the lighting on it by adding some lights that simulate the green light bouncing off from the leaves.
- ▶ The shape of the snail could be improved by adding some deformation to its border; changing the very straight shape that it has now. Once again, the sculpt tools can be very helpful for this.
- ▶ The ultimate control for the shaders can be achieved by using hand-painted textures that would require UV mapping to be properly set up on the surface.

Index

Symbols

3D car models 107
3D sketch
 patches, creating for 115, 116

A

Add mode 245
Adjust Color Curves dialog 222
alpha_control plane 159
AlphaOver node 60, 306
ambient occlusion 305
animation
 tweaks, adding to 270, 271
animation keys 264-267
animation methods
 about 267
 pose to pose 267, 268
 straight ahead 267
armature
 about 251
 creating 251-258
 setting up 258-261
AutoMerge Editing option 21

B

Background Images panel 112
bake functionality 174
Bilateral Blur filter 245
Bilateral Blur node 240
Blender 2.49b 10, 17
Blender 2.50 10, 18, 63
Blender Game Engine (BGE) 173
 objects, setting up for 205-211

blend_fade texture 160
blend texture
 settings 300
Blinn shader model 31, 44
Blur node 56, 238
body
 modeling, for snail 290-297
 shading, for snail 298-301
body material
 settings 298
Border Select tool 23, 290
breakdown pose 268-270
bricks material 180
building
 modeling 176-183
 shading 200-204
building environment
 cement floor, creating 184
 fence, creating 185
 fence, shading 197, 198
 floor, shading 194-196
 grass, creating 183, 184
 grass, shading 191-194
 skydome, creating 186
 skydome, shading 188-190
 building environmenttrees, shading 199
bump map 223
bump texture 228

C

car modeling and shading project
 doors, creating 125-131
 features 108
 flecksA material, settings 144
 glass material, settings 140
 initial patches, creating 111-124

- lights material, settings 141
- overview 107, 108
- patches, joining 131-137
- requisites 109, 110
- rimMetal material, settings 141
- shading 140-147
- simple render setup 147, 148
- SpecFlecksB material, settings 144
- steps 109
- tire material, settings 140
- wheels, creating 137-139
- WideSpec material, settings 146
- cement floor**
 - creating, for building environment 184
- character.blend file 250**
- characters**
 - key poses, creating for 264-268
- char_skel armature 262**
- Clipping option 114**
- closing image**
 - creating, for professional demo reel project 154, 155
- cloudDensity object**
 - settings 92, 93
- cloud material**
 - settings 93, 94
- cloudPoints texture**
 - settings 94, 95
- clouds texture 227**
- ColorRamp 57**
- composite node 37**
- compositing 236, 305**
- Computer Graphics 7**
- constraints 261**
- Convert Premul option 306**

D

- datablocks 165, 166, 236**
- decimate modifier 95**
- decorative trees**
 - creating, for building environment 185
- demoreel.blend file 162**
- Detailed render of earth from space project**
 - checklist 220
 - compositing set up, preparing 232-236
 - features 220

- lights, setting up 229-232
- objectives 220
- objects, setting up 225-229
- overview 219
- scene, compositing 236-246
- textures, preprocessing 221-224
- working 219

diff texture 226

Discombobulator 16

Discombobulator script 18

dLoc property 217

E

earth.blend file 225

EarthClouds layer 223

Edge Select Mode 13, 21, 119

Edge Slide tool 73

empArray object 278

env file 90

environment

- modeling 183-187

- shading 187-199

F

Face select mode 181

Fac parameter 42, 245

fakeRefl texture

- settings 88, 89

fence

- creating, for building environment 185

- shading, for building environment 197, 198

fighter aircraft

- about 63

- animating 99-102

- modeling 66-85

- shading 86-89

Fighter Aircraft Flying Through Storm project

- checklist 65, 66

- features 64

- fighter aircraft, animating 99-102

- fighter aircraft, modeling 66-85

- fighter aircraft, shading 86-89

- final scene, assembling 96-99

- objectives 65

- overview 63, 64

- simple lightning effect, animating 103, 104

- steps 64
- storm cloud, modeling 90-96
- storm cloud, shading 90-96
- fighterBody material**
 - settings 86
- fighterBody object 66, 81**
- fighterCabin material**
 - settings 88
- fighterParent object**
 - about 99
 - properties 97
- fighterStabilators object 89**
- fighter-storm.blend file 66**
- fill light 229**
- final scene, Fighter Aircraft Flying Through Storm project**
 - assembling 96-99
- final scene, Starship Shield Impact**
 - assembling 42-51
 - fcompositing 52-61
 - organizing 42-51
- final video**
 - composing, for professional demo reel project 167-170
- flecksA material**
 - settings 144
- floor**
 - shading, for building environment 194-196
- frames material 178**
- FullSkiesDusk0001 texture 109**

G

- Gimp 109, 110, 223**
- Glare node 40**
- glass material**
 - settings 140
- GLSL Shading 194**
- Graph Editor 100**
- grass**
 - creating, for building environment 183, 184
 - hading, for building environment 191-194
- Grayscale copy of layer option 221**
- greenBlots texture**
 - settings 87

H

- hi-tech machines 63**
- Hotshot Challenge 62**
- Hue Saturation Value node 240**

I

- IK constraint 258**
- import keyword 216**
- individual scenes**
 - creating, for professional demo reel project 162-165
- Influence panel 31**
- initial patches**
 - creating, for car model 111-124
- interactive walkthrough project**
 - building environment, modeling 183-187
 - building, modeling 176-183
 - building, shading 200-204
 - environment, shading 187-199
 - features 174
 - objectives 174
 - overview 173
 - requisites 175
 - scripts, coding 211-216
- intro image**
 - creating, for professional demo reel project 154, 155
- Inverse Kinematics 260**
- IOR factor 31**

K

- keyframes 100, 271**
- key light 229**
- key poses**
 - creating, for characters 264-268

L

- lighting rig 47**
- lightmaps 216**
- lights**
 - setting up 229-232
- lights material**
 - settings 141

LocRotScale option 269
Loop Cut and Slide tool 14, 69, 70, 113

M

mainControl object 206
Material tab 30
Merge tool 122, 137
mesh datablock 167
mesh modeling 85
metaballs 90
Mirror Modifier 26, 114
Mix block 233
model
 creating, for mother ship 11-18
 creating, for space fighter 19-26
mother ship
 model, creating for 11-18
motherShip-detailed.blend file 17
motherShip object 15

N

nebula
 about 8
 compositing 36
nebula background
 creating 33-42
Node Editor 162
nodes
 creating 235
normal map 224
NURBS modeling 107

O

Object Center 26
object datablocks 167
objects
 setting up 225-229
 setting up, for Blender Game Engine (BGE) 205-211
Object tab 43
offline rendering 173
OpenEXR format 42

P

parenting 261
parenting process 259
particle system settings 34
patches
 creating, for 3D sketch 115, 116
planet
 about 8
 modeling 31-33
 shading 31-33
planetEarth object 230
point lamp
 settings 97
Pose Mode 251, 260
pose to pose method 268
Professional Demo Reel project
 about 151
 closing image, creating 154, 155
 features 153
 final video, composing 167-170
 individual scenes, creating for 162-165
 intro image, creating 154, 155
 objectives 153
 overview 151
 requisites 153
 studio setup, creating 155-162
 wire material, settings 156
 working 152
projects
 car modeling and shading 107-110
 detailed render of earth from space 219-221
 fighter aircraft flying through storm 63-66
 interactive walkthrough 173-175
 professional demo reel 151-153
 snail, creating 275, 276
 walk cycle animation 249, 250
Proportional Edit 9
Proportional Editing 82

Q

Quaternion option 262

R

Ramp option 286
raytrace method 46
Rectangular Selection Tool 110
reflections 31
Remove Doubles tool 126
RenderLayer block 233
renderlayer node 37, 38
Render Layer node 59
RenderLayers 9
RGB Curves node 58, 238
rim light 229
rimMetal material
 settings 141
root 251

S

scene
 compositing 236-246
 creating, in space 7
 setting up, for snail creation project 302, 304
scripts
 coding, for interactive walkthrough project
 211-216
Selective Gaussian Blur 216
Select Linked tool 137
sf_body_specNoise texture
 settings 29
sf_cabin_reflMap texture 28
sf_cristalCabin material
 about 26
 settings 27
sf_metalCannon material
 about 26
 settings 30
sf_shipBody material
 about 26
 settings 28, 29
sf_shipWings material
 about 26
 settings 30

shader 31, 108, 173
shader settings
 adjusting 45
shell
 modeling, for snail creation project 277-283
 shading, for snail creation project 283-289
shell material
 settings 285
shieldDistort texture
 settings 46
shieldFX texture
 settings 46, 47
simple lightning effect
 animating 103, 104
Skeleton Sketching tool 253, 258
sketch
 creating, for space scene 8
skinning 262, 263
skydome
 creating, for building environment 186
 shading, for building environment 188-190
skydome object
 settings 188
Smooth Iterations parameter 284
Smooth Vertex tool 284
snail.blend file 277
snail creation project
 about 275
 body, modeling for snail 290-297
 body, shading for snail 298-301
 compositing 305
 features 276
 overview 275
 scene, setting up for 302, 304
 shell, shading 283-289
 shell shape, creating 277-283
Snap option 137
space
 scene, creating in 7
space fighter
 about 8
 model, creating for 19-26
 shading 26-31
space scene
 about 7
 sketch plan, creating for 8
space-scene.blend file 10, 15, 163

- SpecFlecksB material**
 - settings 144
- spec texture 227**
- specular intensity 44**
- specular map**
 - creating 223
- Specular panel**
 - settings 45
- specular reflection 219**
- starFighter object 166**
- Starship Shield Impact**
 - Hotshot Challenge 62
- Starship Shield Impact project**
 - final scene, assembling 42-51
 - final scene, compositing 52-61
 - final scene, organizing 42-51
 - mother ship, modeling 11-18
 - nebula background, creating 33-42
 - planet, modeling 31-33
 - planet, shading 31-33
 - rendering 9
 - space fighter, modeling 19-26
 - space fighter, shading 26-31
- stencil parameter 302**
- storm cloud**
 - modeling 90-96
 - shading 90-96
- straight ahead method 267**
- studio setup**
 - creating, for professional demo reel project 155-162
- Subdivision Surface modifier 21, 83**
- Subsurface Scattering options 289**
- Subsurf modifier 26**
- sun lamps**
 - settings 98

T

- tex folder 175, 221**
- textures**
 - about 221
 - preprocessing 221-224
 - reasons, for creating 224
- tileable textures 173**
- tip 251**

- tire material**
 - settings 140
- Toggle Quad View button 111**
- topology 85, 86, 108**
- trees**
 - shading, for building environment 199
- tweaking 85**
- tweaks**
 - adding, to animation 270, 271

U

- UV Sphere 186**

V

- Video Sequence Editor 168, 171**
- Volkswagen cars blueprints**
 - URL 109
- volumetric effect 220**
- volumetric materials 96**
- volumetric shader 63**
- Voronoi type 47**

W

- walk cycle 249**
- walk cycle animation project**
 - about 249
 - animation keys 264-268
 - armature, creating 251-258
 - armature, setting up 258-261
 - breakdown pose 268-270
 - checklist 250
 - features 250
 - objectives 250
 - skinning 262, 263
 - tweaks, adding to animation 270, 271
- walkcycle.blend file 250**
- warp parameter 302**
- Weight Paint mode 263**
- WideSpec material**
 - settings 146
- windows material 178**
- wire material**
 - settings 156

Y

Y location channels 101



Thank you for buying Blender 2.5 HOTSHOT

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

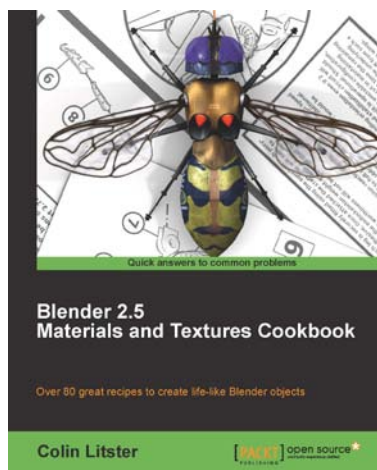
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



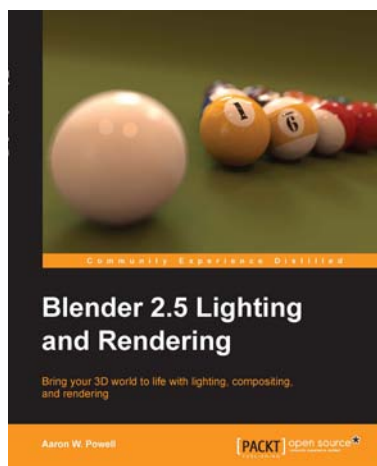
Blender 2.5 Materials and Textures Cookbook

ISBN: 978-1-84951-288-6

Paperback: 312 pages

Over 80 great recipes to create life-like Blender objects

1. Master techniques to create believable natural surface materials
2. Take your models to the next level of realism or artistic development by using the material and texture settings within Blender 2.5.
3. Take the hassle out of material simulation by applying faster and more efficient material and texture strategies



Blender 2.5 Lighting and Rendering

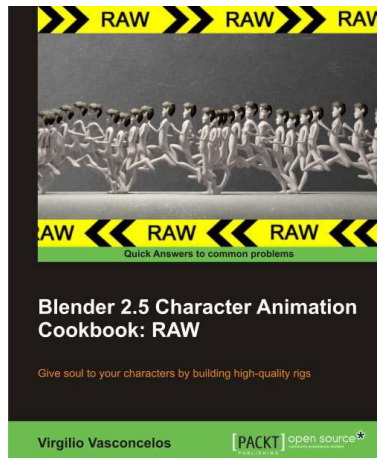
ISBN: 978-1-84719-988-1

Paperback: 252 pages

Bring your 3D world to life with lighting, compositing, and rendering

1. Render spectacular scenes with realistic lighting in any 3D application using interior and exterior lighting techniques
2. Give an amazing look to 3D scenes by applying light rigs and shadow effects
3. Apply color effects to your scene by changing the World and Lamp color values

Please check www.PacktPub.com for information on our titles

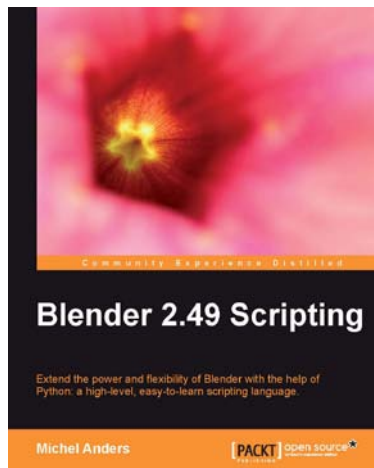


Blender 2.5 Character Animation Cookbook: RAW

ISBN: 978-1-84951-320-3 Paperback: 350 pages

Give soul to your characters by building high-quality rigs

1. Learn how to create efficient and easy to use character rigs
2. Understand and make your characters , so that your audience believes they're alive
3. See common approaches when animating your characters in real world situations



Blender 2.49 Scripting

ISBN: 978-1-849510-40-0 Paperback: 292 pages

Extend the power and flexibility of Blender with the help of the high-level, easy-to-learn scripting language, Python

1. Gain control of all aspects of Blender using the powerful Python language
2. Create complex meshes programmatically and apply materials and textures
3. Automate the rendering process and extend Blender's image manipulation capabilities

Please check www.PacktPub.com for information on our titles