Playing with 3D games technology

Blender is a fast, powerful and free 3D creation suite. It's the first of the 3D packages to integrate a game engine and tools for editing game-logic and creating interactive animation.

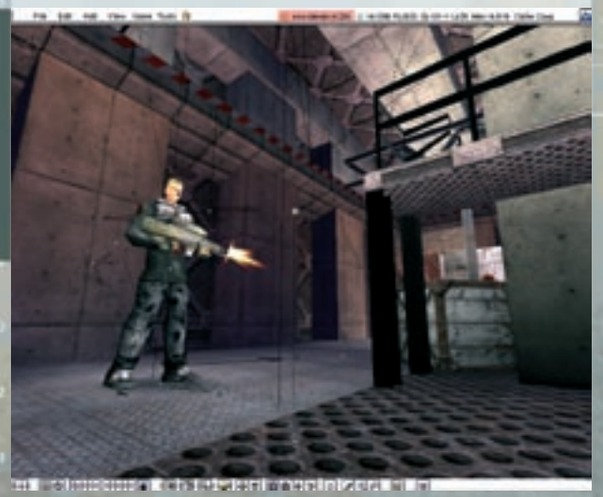
The Blender GameKit has an extensive section for people who are new to 3D, or new to Blender. It shows in step-by-step tutorials the fun of creating models, adding motion to them, and how to turn them into simple games. Experienced 3D artists will appreciate the more complex game demos, the character animation tutorials, and the advanced reference section.

About this book

The Blender Gamekit was produced by Not a Number, the company that developed Blender as a commercial product. Blender is now continued as an Open Source project led by Ton Roosendaal, the original Blender creator. Carsten Wartmann is a 3D designer and writer, renowned as the author of *The Blender Book* and *The Official Blender 2.0 Guide*.

The CDROM contains 10 playable and editable Blender game demos. It also contains the Blender Creator V2.24, for all platforms: Windows (98/2000/ME/XP), Linux (i86), FreeBSD (i386) and IRIX (6.5)

System specs: 450 Mhz processor, 64 MB memory, OpenGL accelerated 3D card (Nvidia, Matrox, ATI, 3Dlabs)



the official blender gamekit interactive 3d for artists

the official blender gamekit

interactive 3d for artists

Produced & edited by **Ton Roosendaal** and **Carsten Wartmann**

- Includes CDROM with 10 game demos and Blender 2.24
- Fully integrated 3D creation suite. No programming required.
- Includes Windows and Linux version

BlenderFund
www.blender.org

blender

Blender Gamekit

The official

Blender Gamekit

Interactive 3D for Artists

Produced & edited by **Ton Roosendaal** and **Carsten Wartmann**

Authors:
Michael Kauppi, Carsten Wartmann

Tutorials, game demos, CDROM content:
Joeri Kassenaar, Freid Lachnowicz, Reevan McKay,
Willem-Paul van Overbruggen, Randall Rickert, Carsten Wartmann

Editor:
Carsten Wartmann

Design and DTP:
Samo Korosec, froodee design, www.frodee.com

Production:
Ton Roosendaal

Published by:

Blender Foundation
Entrepotdok 42 t/o
1018 AD Amsterdam
the Netherlands

www.blender.org
www.blender3d.org

info@blender.org

Special thanks to the NaN Technologies crew:

Management:
Gil Agnew, Jan-Paul Buijs, Maarten Derks, Loran Kuijpers, Ton Roosendaal, Jan Wilmink

Software engineers:
Frank van Beek, Laurence Bourn, Njin-Zu Chen, Daniel Dunbar, Maarten Gribnau, Hans Lambermont, Martin Strubel, Janco Verduin, Raymond de Vries,

Content developers:
Joeri Kassenaar, Reevan McKay, WP van Overbruggen, Randall Rickert, Carsten Wartmann,

Website, support & community:
Bart Veldhuizen, Willem Zwarthoed,

Marketing:
Elisa Karumo, Sian Lloyd-Scriven

System administrators:
Thomas Ryan, Marco Walraven

Administration and backoffice:
Maartje Koopman, Annemieke de Moor, Brigitte van Pelt

Blender Gamekit Contents


001
003
2.03beta
004
2.04
game Blender
005
010
2.10
011
2.12
blenderGameKit
blender
POWER
CREATE
A
B
C
D
2.20
022
023
2.23


_part I ::
introduction

Chapter 2. What is this book about

Blender offers you a new and unique way to explore interactive 3-D graphics. This book will guide you through many aspects of making your own games and interactive 3-D graphics with Blender.

You can have fun with the ready made games on the CD instantly, but changing them or creating your own game is also great fun.

Blender is a fully integrated 3-D creation suite. It has all the tools for making linear animation and non-linear (interactive) 3-D graphics. All of these features are provided in one single application and gives the artist a very smooth workflow from design, to modeling, animating and on-to publishing of 3-D content. For example if you needed to make a demo trailer of a game you would need a modeler, a renderer, a video editing application and the game engine itself to produce the video. Blender offers you all these tools combined to produce interactive and linear 3-D content.

The book contains:

- Example game scenes to play with
- Example games and tutorial scenes to change and personalize
- Blender basics for making interactive 3-D graphics
- 3-D game technology basics
- Advanced tips and topics from professional Blender artists
- References for the Blender game engine

How to use this book?

First, you should install Blender on your computer. Blenders installation is a very easy process, but should you experience any difficulties with the installation process or running Blender, please read Section 29.1. With Blender installed, you can explore the games on the CD, which accompanies this book.

Chapter 1 and Part II in *Game Creation Kit* introduce you to Blender by enabling you to have fun with 3-D game technology, and teaches you how to use Blender, supported by many practical examples. Depending on your previous knowledge of Blender, you should then read the Blender Basics in Chapter 4.

You are now ready to start with the tutorials. They are divided into beginner, intermediate and advanced tutorials. If you run into problems please refer to the index and the glossary to find further information on what is available in this book. Also, be sure to join the huge and lively Blender Community (see Section 29.4), or ask our support if you run into troubles.

I hope you enjoy reading this book. My thanks go to the tutorial writers who have helped to produce the wonderful content of this book, the developers of Blender and all other people who have made this book possible.

Carsten Wartmann, February 2002

Chapter 3. Introduction to 3-D and the Game Engine by Michael Kauppi

3.1. Purpose of This Chapter

This chapter will introduce you to the world of three dimensional (3-D) computer graphics, first by introducing the general concepts behind 3-D and then by showing how those concepts are used in computer graphics. Then, it will introduce you to game engines, especially Blender's game engine, and three aspects that are often found in good games. This chapter is aimed at those who have little or no experience in 3-D or with game engines.

3.2. General Introduction to 3-D

3.2.1. 2-D overview

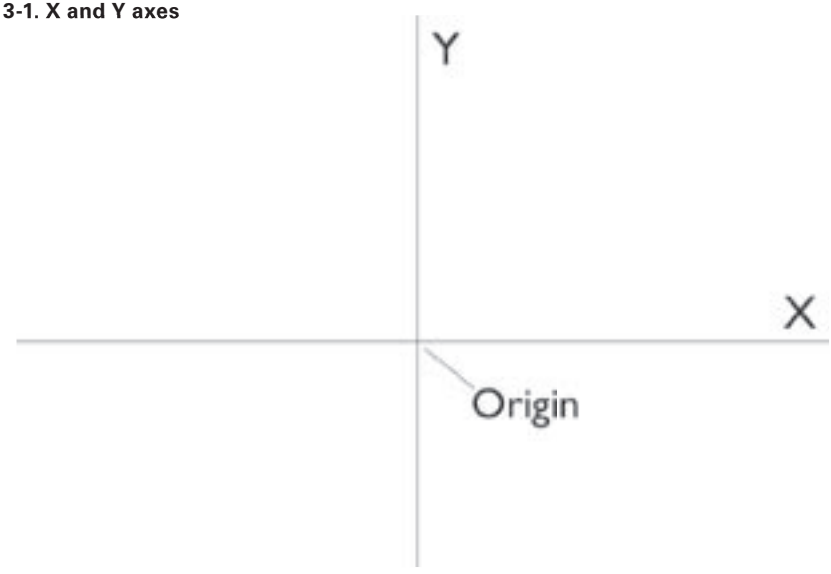
We'll begin our journey into 3-D with an overview of 2-D because most people reading this should already know the concepts behind 2-D or least be able to grasp them fairly quickly.

XY axes

You can think of 2-D as being a flat world. Imagine you put a blank piece of paper on a table, and look down at that paper.

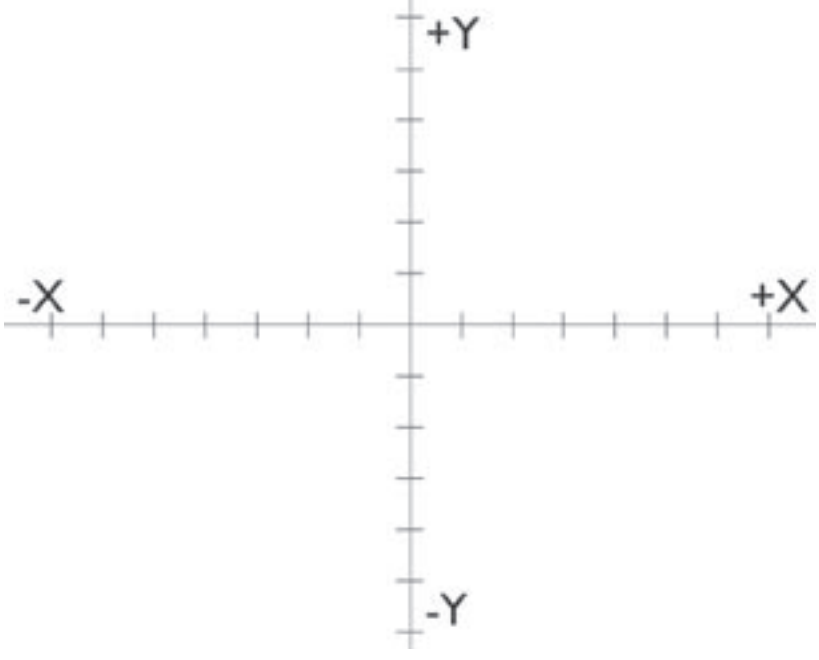
If that paper represented the 2-D world, how would you describe where things are located? You need some kind of reference point from which to measure distances.

Figure 3-1. X and Y axes



This is generally done by drawing two lines, called axes: one horizontal and the other vertical (Figure 3-1). The horizontal line is called the X-axis, and the vertical line is called the Y-axis. Where the axes cross is your reference point, usually called the "origin".

Figure 3-2. Positive and negative axes



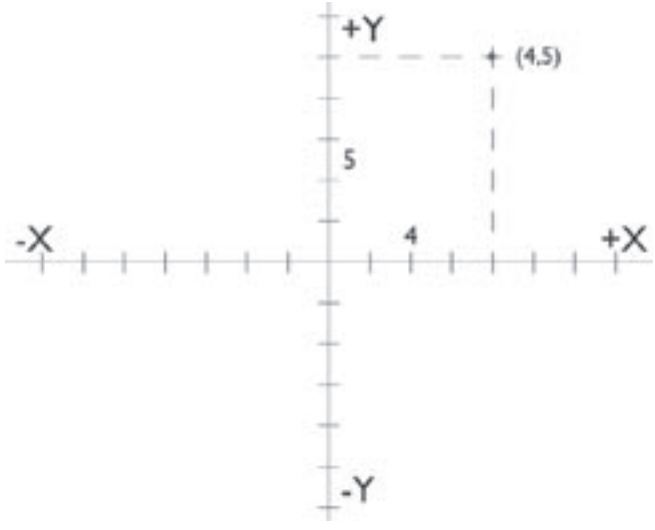
Along these axes, imagine a series of regularly spaced hash marks, like the lines on a ruler. To describe where something is, you count out the distance along the X and Y axes. Distances to the left and below the origin on the X and Y axes respectively are negative, while distances to the right and above the origin on the X and Y axes respectively are positive (Figure 3-3).

For example, if you wanted to describe where the dot in Figure 3-2 is located, you would count out 4 units along the X-axis (known as the X coordinate) and 5 units along the Y-axis (known as the Y coordinate).

Now with a default origin and XY coordinates, we can begin to describe 2-D geometry.

Points

Figure 3-3. Defining the position of a point in 2-D space

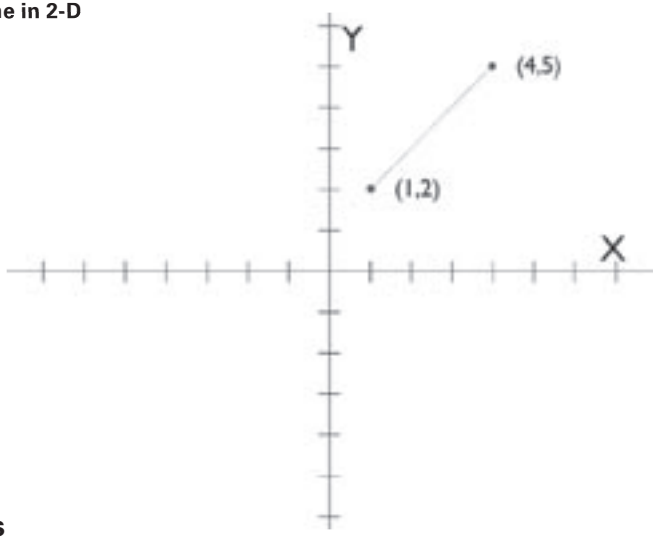


The dot from Figure 3-3 is the simplest object that can be described in 2-D, and is known as a point. To describe a point you only need an X and a Y coordinate.

Lines

The next simplest object we can describe in 2-D is the line. To describe a line, you only need to describe two points (Figure 3-4).

Figure 3-4. A line in 2-D



Polygons

By connecting three or more lines, you can begin to describe shapes, known as polygons. The simplest polygon is the three-sided triangle, next is the four-sided quadrangle, or quadrilateral, (usually shortened to quads), and so on, to infinity. For our purposes, we'll only work with triangles and quads.

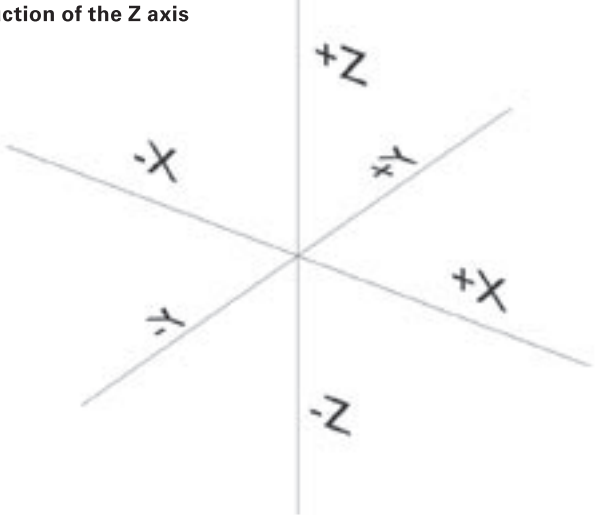
With this knowledge, it's now time to expand from 2-D to 3-D.

3.2.2. 3-D, the third dimension

As the name implies, 3-D has an extra dimension but the concepts we covered in the 2-D discussion above still apply.

Z axis

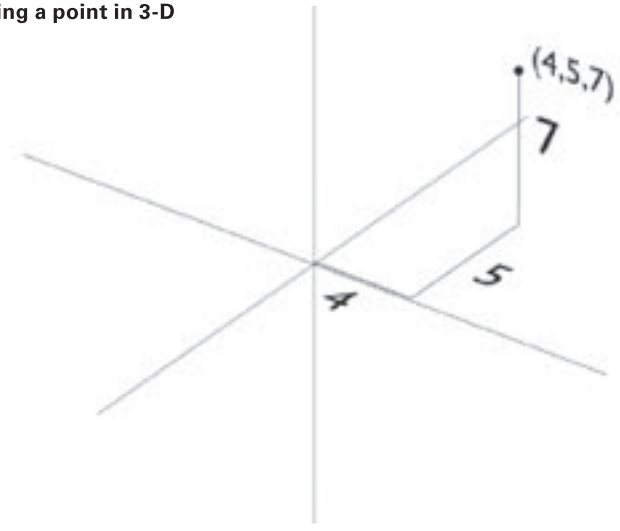
Figure 3-5. Introduction of the Z axis



Just like 2-D, we need a reference point from which to describe the location of things in 3-D. This is done by drawing a third axis that is perpendicular to both the X and Y axes, and passes through the origin. This new axis is usually called the Z-axis, and values above and below the origin are positive and negative respectively (Figure 3-5). By using this new axis we can describe objects as they exist in the real world.

Points

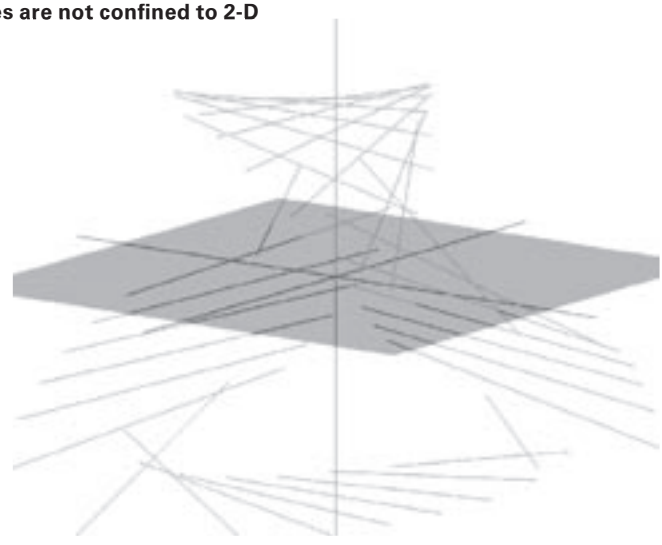
Figure 3-6. Defining a point in 3-D



To describe a point in 3-D, we now need three coordinates: the X, Y and Z coordinates (Figure 3-6).

Lines

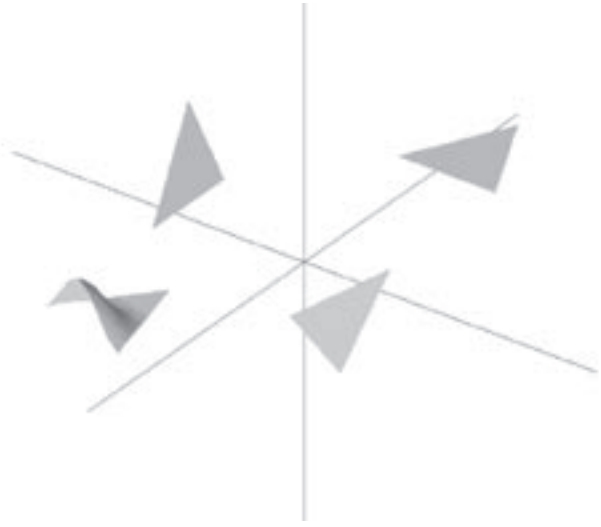
Figure 3-7. Lines are not confined to 2-D



As in 2-D, we can describe a line by defining two points, but now our line does not have to lay flat, it can be at any angle imaginable (Figure 3-7).

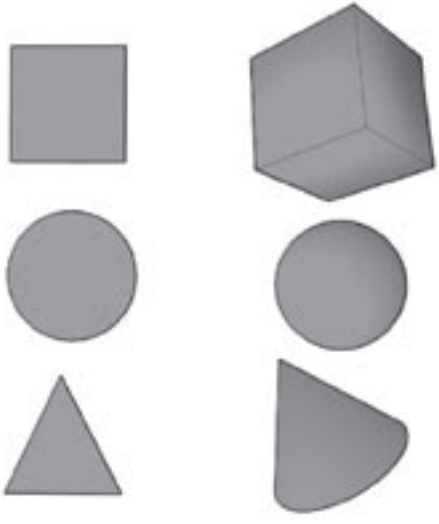
Polygons

Figure 3-8. Polygons are not confined to 2-D



By connecting lines, we can form polygons just like in 2-D. Our polygons, just like our lines, are no longer confined to the flat 2-D world (Figure 3-8). Because of this, our flat 2-D shapes can now have volume. For example, a square becomes a cube, a circle becomes a sphere and a triangle becomes a cone (Figure 3-9).

Figure 3-9. Some 2-D shapes and their 3-D counterparts



Now with the basics of 3-D covered, let's see how they relate to 3-D computer graphics.

3.2.3. 3-D computer graphics

By now, you should have the general concepts of 3-D in mind. If not, go back and reread the previous sections. Having these concepts in mind will be very important as you proceed through this guide. Next, we'll show you how the concepts of 3-D are used in 3-D computer graphics, also known as computer graphic images (CGI).

Terminology

A slightly different set of terms is used for CGI. Table 3-1 show how those terms relate to what you have learned so far.

Table 3-1. CGI Terminology

3-D term	Related CGI term
Point	Vertex
Line	Edge
Polygon	Polygon

Armed with our new terminology, we can now discuss CGI polygons.

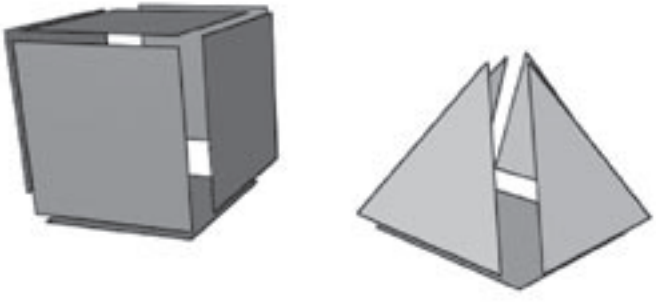
Triangles, quads

While theoretically, a polygon can have an infinite number of edges, the more edges there are, the more time it takes a computer to calculate that shape. This is why triangles and quads are the most common polygons found in CGI, they allow the creation of just about any shape and do not put too much stress on the computer to calculate. But how do you form shapes with triangles and quads?

Mesh

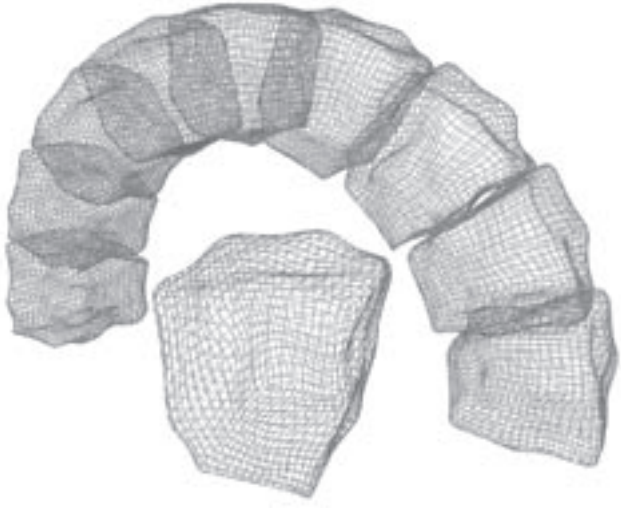
As discussed before, our polygons are no longer confined to the flat 2-D world. We can arrange our polygons at any angle we choose, even "bending" our polygons if necessary. By combining a series of polygons together at various angles and sizes, we can create any 3-D shape we want.

Figure 3-10. Combining polygons to more complex shapes



For example, six squares can combined to make a cube, and four triangles and a square form a pyramid (Figure 3-10). By increasing the number of polygons and manipulating their locations, angles and sizes we can form complex objects (Figure 3-11). As you can see, the more complex an object, the more it takes on a mesh-like appearance. In fact, the object in Figure 3-11 is being viewed in "wire mesh" mode. You'll often hear the term "mesh" used to describe any combination of CGI polygons.

Figure 3-11. Arch made of quad based blocks



Primitives

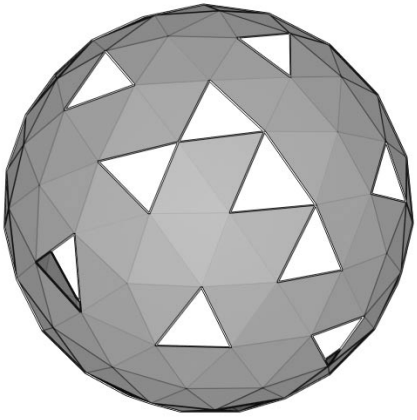
As shown above, we can create shapes by combining polygons, but to form basic shapes by hand (such as spheres, cones, and cylinders) would be very tedious. So 3-D applications like Blender have preprogrammed shapes called "primitives" that you can quickly add to a 3-D scene. Blender's mesh primitives include: planes, cubes, spheres, cones, cylinders and tubes. There are other primitives as well (not all of them mesh based), and you will learn about them as you develop your Blender skills.

Faces

Figure 3-12. Unfaced (left) and faced polygon (right)

Polygons can be faced or unfaced. You can think of an unfaced polygon as being made of just wire, while a faced polygon has a "skin" stretched over that wire (Figure 3-12). When you tell Blender to draw your 3-D scene, called rendering, the faced polygons will appear solid, while the unfaced polygons will appear as holes (Figure 3-13).

Figure 3-13. Unfaced polygons appear as holes in objects



Materials

Figure 3-14. Sphere objects with different materials

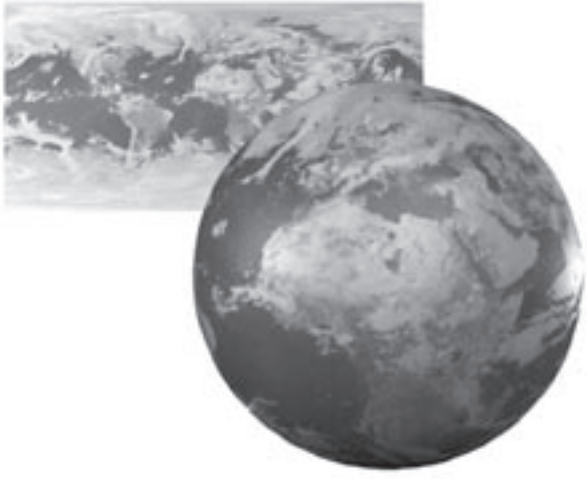
Look at objects around you, they have many characteristics. Some are shiny, some are matte. Some are opaque, some are transparent. Some appear hard, while others appear soft. To recreate these characteristics in the 3-D world, we apply a "material" to an object which tells Blender how to render the object's color, how shiny the object should appear, its perceived "hardness" and other properties (Figure 3-14).

Textures

Take a look at the things around you again. Besides their material properties, the things around you also have texture. Texture affects not only how something feels (smooth or rough), but also how something looks (colors and patterns). Since we can't touch what we make in the 3-D CGI world, we will focus on how things look.

Image maps

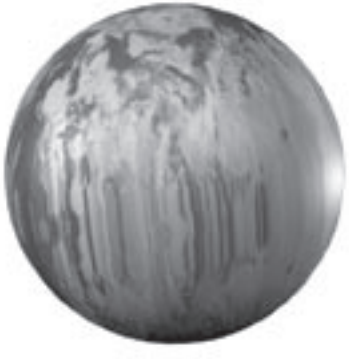
Figure 3-15. Map of the eart (back) wrapped around a sphere



A common method for applying textures is through the use of image maps. That is 2-D images which we then "wrap" around an object (see Figure 3-15).. Image maps allow us to represent minute detail on our models (objects) that would be difficult to model directly and that would greatly increase the number of polygons if we did model them. Using image maps lets us keep the number of polygons low on our models, thus letting Blender render our scenes faster, which is especially important for real-time rendering in the game engine.

UV mapping

Figure 3-16. Badly mapped earth texture

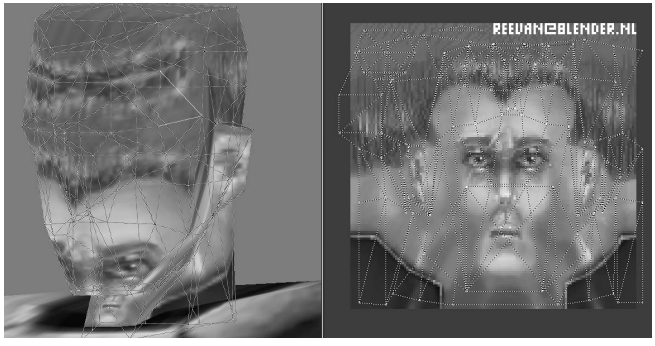


One common problem with image maps is the accurate wrapping of the maps around an object, especially a complex one. Many times the texture will not be aligned as we wish or it may "stretch" (Figure 3-16). A popular method for overcoming this problem is the use of UV mapping.

UV vs. XY coordinates

In order to continue, it is necessary to point out what UV coordinates are. As mentioned in the 3-D overview, you can describe a point (vertex) by giving its X, Y and Z coordinates. If you want to 'map' a 2-D image onto a 3-D object, the XYZ coordinates have to be transformed into two dimensions. These transformed coordinates are usually called the "UV coordinates". Instead of calculating UV coordinates automatically, you can define them yourself in Blender. This means, that for each vertex, not only a an XYZ coordinate is stored, but also the two values for U and V.

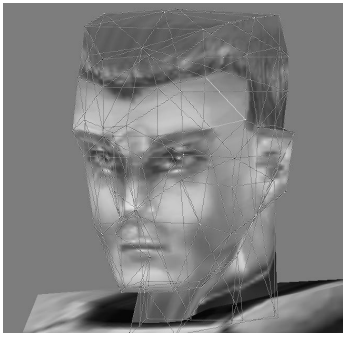
Figure 3-17. Badly positioned head texture



So, how does UV mapping work? Take a look at the head object in Figure 3-17. Each corner of the faces is a vertex, and each vertex has an XYZ and UV coordinate as explained earlier. Using Blender's UV editor, we unwrap the mesh, much like we do when we take a globe and lay it flat to make a map of the world, and lay that mesh on top of our 2-D image texture.

Then, by moving the unwrapped mesh's UV coordinates, we can tell Blender exactly where the texture should go when Blender wraps the texture around our 3-D object (Figure 3-18).

Figure 3-18. Finally placed texture



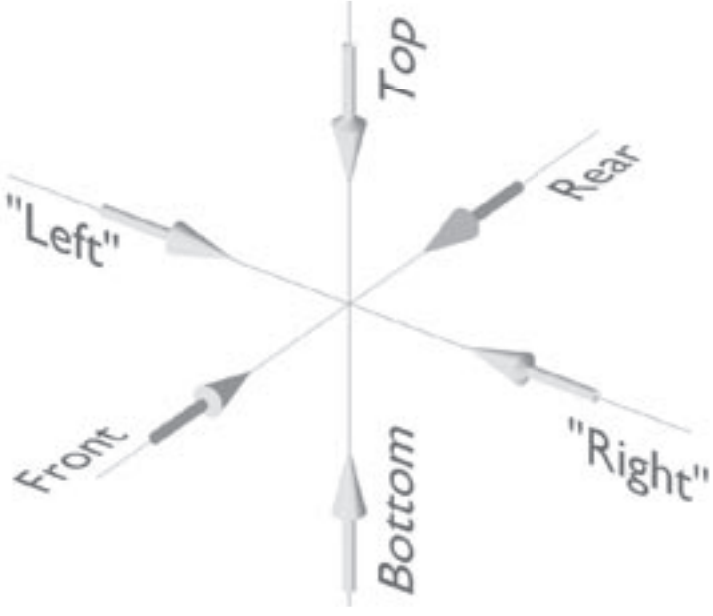
The reason it is called a UV editor and not a UVW editor, is that we make our adjustments in 2-D (UV) and Blender automatically takes care of the W coordinate when it wraps the texture around our model. Not having to worry about the third dimension makes our job easier in this case.

Viewing 3-D space

To do anything in 3-D, we need to be able to see what we are doing. This is accomplished using "views". This section will discuss the various views available in Blender ("standard", "interactive" and "camera" views), and the two view modes available. This section will not cover the steps you need to take to use the views. Those will be explained in Section 4.10. It will also mention the use of lights, which are not actually views but are necessary if you want to see anything when you render your 3-D scene and can be used to alter the mood of our scenes.

Standard

Figure 3-19. Blender's six fixed views



There are six pre-programmed standard views in Blender, each looking along a particular axis as shown in Figure 3-19. These views are generally used when modeling objects because they help to provide a sense of orientation. They are also useful if you get disoriented using the interactive view.

Interactive (free)

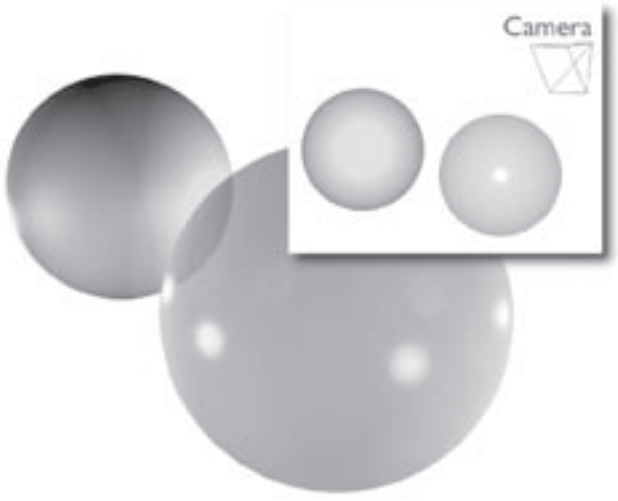
Figure 3-20. Guess the true shape of this object!

While the standard views are very useful for modeling, sometimes they don't help us visualize how an object will look in 3-D (Figure 3-20). This is when Blender's interactive view becomes useful. Blender's interactive view allows you to rotate your entire 3-D scene in any direction interactively (in real-time) to let you view things from any angle (Figure 3-21). This helps you visualize how your scenes and models will look.

Figure 3-21. Object from Figure 3-20in a perspective view

Cameras

Figure 3-22. Image from Figure 3-14 and how the camera was positioned in the scene



The standard and interactive views are generally not used when it is time to render your scenes (stills, animations or real-time rendering in the game engine). Instead, you use a camera view for rendering. You can think of this like a movie set. You are the director and can walk around and look at your set from any direction you want (standard and interactive views) to make sure everything is just as you want it, but when it is time to shoot the scene you need a camera. This is what your audience will see, and the same holds true for camera views (Figure 3-22).

View modes

Figure 3-23. Othogonal and perspective modes

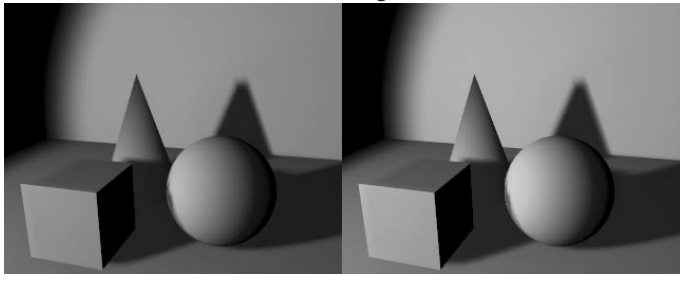
Here are two viewing modes for all the views in Blender: “orthogonal” and “perspective”. Orthogonal mode views everything without perspective, whereas the perspective mode, as the name implies, uses perspective (Figure 3-23). Orthogonal mode is useful when creating your models because there is none of the “distortion” associated with the perspective mode, and this helps your accuracy. The perspective mode, like the interactive view, can help give you a sense of what your model will

look like, but without the need to rotate the entire 3-D scene. Rotating the entire scene can be slow if it is very complicated.

Lights

When you are ready to render your scene, or play your game, you will need at least two things: a camera and lights. If you try to render without a camera you will get an error message, but if you try to render without a light all you will get is a black image. This is one of the most common mistakes for new to Blender users, so if you try to render something and all you get is a black square be sure to check if you've put in a lamp or not. For the interactive 3-D graphics, there can be scenes without light, but they usually look flat.

Figure 3-24. Same scene rendered with different lights

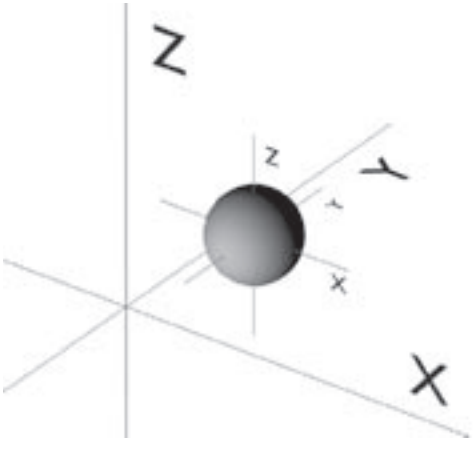


There is more to lights than just being able to see. Just like in real life, lights can help set the atmosphere or mood of a scene. For example, using a low blue light helps to create a "cool/cold" atmosphere, while a bright orange light might create a "warm" one (Figure 3-24). Lights can be used to simulate ambient light, muzzle flashes or any other effect where you would expect to see light.

Because you will be creating games with objects that move and change, there is another important concept we must cover:

Transformations

Figure 3-25. Local axis of an object



As touched on earlier, we describe the locations of objects in our 3-D worlds by using an origin and a XYZ coordinate system to measure with. The coordinates calculated from this default origin are known as global coordinates. In addition, an object's center serves as its own origin, and so the object can have its own XYZ axes (Figure 3-25). This is called a local origin, and a local coordinate system with local coordinates. But why is this important?

A game where nothing moves or changes will not get much of a following. The objects in your games will need to move, and this is one place where the concept of transformations becomes important. The three most common transformations are translation, rotation and scaling.

Table 3-2. Transformations

Transformation	Description
Translation	When an object move from point A to point B
Rotation	When an object spins around a particular point or axis
Scaling	When an object increases or decreases in size

When you make your games, you'll have to keep in mind that transformations are relative and can affect game play. When an object translates from point A to B in the global coordinate system, from that object's point of view, its local coordinate system doesn't necessarily move. For example, a character standing in a moving train seems to be stationary from their point of view. The train's speed may be 100 kph, but the character feels like they are standing still. Their local origin (their center) doesn't move as far as they are concerned.

However, if we look at the same character from the point of view of someone standing still outside the train, now the character is moving. From this second character's local point of view, they are standing still and the first character is moving, but neither are rotating. Or are they?

If we look from the point of view of another character, hovering in space, not only are both of the other characters on the Earth, rotating as the Earth rotates on its axis, but also as the Earth rotates around the Sun. So, how does this affect game play? Imagine everyone is trying to hit a stationary target on the train. The first character has the easiest job, a stationary target, the second character has to hit moving target, and the third character has to hit a target that is moving and experiencing two forms of rotation.This shifting of points of view is called "coordinate transformation", and as you can see, it can have an important impact on game play.

In most 3-D software packages you can work with these coordinate systems using so-called *"hierarchies"*. You can define one object as being the *"parent"* of another object; which then becomes a *child*. Now all transformations of the parent are also applied to its children. That way you only have to define motion for a parent to have all its children moving in the same way. In the solar system example, we humans all are in fact "children" of the Earth, which in turn is a "child" of the Sun.

One last point that needs to mentioned is that transformation is not restricted to just shapes. Materials, textures, and even lights can be moved, rotated and scaled. In fact, anything that exists in your 3-D world is actually an object and so is subject

to transformations. As your 3-D skills develop, you will learn how to use global, local and relative transformations to affect game play and to create interesting effects. Now that you have received a basic introduction to 3-D CGI, it's time to talk about game engines and aspects of good games.

3.3. Game Engines and Aspects of a Good Game

3.3.1. What is a game engine?

A game engine is software that simulates a part of reality. Through a game engine, you interact with a 3-D world in real-time, controlling objects which can interact with other objects in that world. If you have ever played a video game on a computer, a console or in a game arcade, you have used a game engine of some kind. The game engine is the heart of a game and consists of several parts. One part displays the 3-D world and its objects on your screen, drawing and redrawing your scenes as things change. Another part deals with decision making (known as game logic), for example, deciding when events like doors opening should occur. Another part simulates physics, such as gravity, inertia, momentum and so on. Yet another part detects when objects collide with each other, while another actually moves objects.

The game engine tries to simulate all these things as quickly as possible to provide a smooth fluid simulation.

For example, in a computer baseball game, the game engine will have the pitcher throw you a pitch (moving an object). As the ball travels the game engine will calculate all the physics that act on the ball, such as gravity, air resistance, etc. Then you swing the bat (or more accurately, you tell the game engine to swing the batter's bat) and hopefully hit the ball (i.e. collision detection between the ball and bat).

This is a very simplified example. he game engines you have used are much more complicated, and can take a team of programmers and a great deal of time to create. Or at least, that was the case until Blender's game engine was released.

3.3.2. Blender's game engine -- Click and drag game creation

Blender is the first game engine that can create complete games without the need to program. Through its click-and-drag graphical user interface (*GUI*), even those with no programming experience can enjoy the challenge of creating fun and exciting games.

After you create your 3-D world and 3-D objects, you only need to use a series of pull-down menus, simple key strokes and mouse clicks to add behavioral properties to that world and those objects and bring them to life. For professionals, this allows for the rapid prototyping of games, and for non-professionals, it's the first chance to produce their own games without having to spend years learning to program or the need for large programming teams. Of course, for those who can program, Blender

uses the Python scripting language to allow programmers to extend Blender's game engine even further.

This relative ease of use, though, hides the Blender game engine's true innovation...

3.3.3. "True" and "fake" 3-D game engines

Blender is a „true" 3-D game engine. Until recently, game logic (decision making) wasn't done on an object level. This meant that a „higher intelligence" (HI) in the game had to control all the objects, moving them when appropriate or keeping track of their condition (i.e. alive or dead). With the advent of „true" 3-D game engines, each object in a game is its own entity and reports such information back to the game engine.

For example, if you are playing a game where you walk through a maze that has hidden doors, in the past the HI would have had to decide when you were close enough to a hidden door and then open it. With Blender's game engine, the door itself can have a sensor function and will determine when another object is close enough, then the door will open itself.

Another example would be a shooting game. The gun has logic attached that detects when you pull the trigger, the gun then creates a new bullet object with a certain starting speed. The bullet, which is now its own entity, shoots out of the gun and flies through the air all the while being affected by air resistance and gravity. The bullet itself has sensors and logic as well, and detects whether it hits a wall or an adversary. On collision, the logic in the bullet and the logic in the collided object define what will happen.

In the past, when you pulled the trigger, the game engine would calculate whether a bullet fired at that time would hit the target or not. There was no actual bullet object. If the game engine determined that a hit would have occurred, it then told the object that had been hit, how to react.

The advantage of Blender's "real" 3-D game engine is that it does a better job of simulating reality because it allows for the randomness that occurs in the real world. It also distributes the decision load so that a single HI isn't required to decide everything.

While Blender provides you with the technology to create good games, it doesn't create them automatically. To create good games, you need to understand three important aspects of games.

3.3.4. Good games

If you analyze successful games, you will find that they have three aspects in varying degrees. This is known as the „Toy, immersive, goal" theory of game creation.

Toy

The toy aspect of a game refers to the immediate fun of just playing it. You don't need to think too much, you can just grab the mouse or the game controller and

start playing, much like you did with your toys when you were a child. You didn't need to read a manual on how to play with your toy cars, or spend time figuring out complicated strategy. In short, games with a high degree of toy are very intuitive. Think of your favorite arcade game at your local game arcade. Most likely you only needed one joystick and two or three buttons, or a simple gun with a trigger.

This doesn't mean that such games don't require skill, but that you can gain immediate enjoyment from playing them.

Immersive

The "immersive" aspect of a game is the degree to which a game makes you forget you are playing a game, sometimes called the "suspension of disbelief". Flight simulators or racing simulators are a good example of this. Realism is an important factor in this, and is one of the reasons that simulators have reached such an advanced level in realism. The "Mechwarrior" series and "WarBirds" are two excellent examples of immersive games which have very realistic environments, animations and sounds. They are fairly low on the toy aspect and take some time to learn to play, with almost every key on the keyboard used for some function.

The old one-button joysticks have been replaced with HOTAS (Hands On Throttle And Stick) systems consisting of a joystick with seven to ten buttons for one hand, a throttle device with an equal number of buttons or dials for the other and even pedals for your feet. These systems combine with the game to create an incredibly immersive environment. These games also often have a high degree of "goal".

Goal

The "goal" aspect of a game is the degree to which a game gives you a goal to achieve. This often involves a lot of strategy and planning. "Age of Empires" and "SimCity" are two games that are very goal oriented. Goal oriented games are often very low on the toy aspect, "SimCity" for example comes with a thick manual explaining all the intricate details of "growing" a successful city. This is not always the case though: "Quake" is a goal oriented game which also has a good deal of toy and immersive aspects to it.

Balance

When you create your games, you will have to strike a balance among the toy, immersive and goal aspects of your games. If you can create a game that has a high degree of each aspect, you'll most likely have a hit on your hands.

3.4. Conclusion

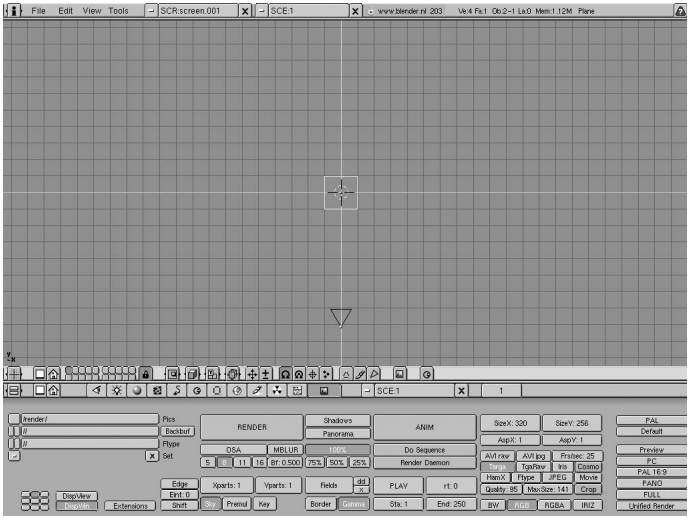
In this chapter you have been introduced to the basic concepts of 3-D including vertices, polygons, materials, textures, origins, coordinate systems and transformations. You have also been introduced to what makes a game work, both on a technological level with the discussion of game engines, and on a conceptual level with the discussion of what makes good games good.

The rest of this book will show you how to use Blender to put these concepts to work when creating games. Once you have finished this guide, you'll have all the tools you'll need to make games, the rest will fall to your own creativity. Good luck and we look forward to seeing you announce your games on Blender's discussion boards (see Section 29.4).

Chapter 4. Blender Basics

For beginners the Blender user interface can be a little confusing as it is different than other 3-D software packages. But persevere! After familiarizing yourself with the basic principles behind the user interface, you'll start to realize just how fast you can work in your scenes and models. Blender optimizes the day-to-day work of an animation studio, where every minute costs money.

Figure 4-1. The first start



Info: *The installation of Blender is simple just unpack it and place it in a directory of your chosing (or let the installer do it). The installation is described in detail in Section 29.1.*

After starting Blender you get a screen as shown in Figure 4-1. The big Window is a 3DWindow where your scene and objects are displayed and manipulated.

The smaller window, located below the 3DWindow, is the ButtonsWindow where you can edit the various settings of selected objects, and the scene.

4.1. Keys and Interface conventions

During its development, which followed the latest 3D graphics developments, an almost new language also developed around Blender. Nowadays, the whole Blender community speaks that language which Ton Roosendaal - the father of Blender - often calls "Blender Turbo language". This language makes it easy to communicate with other Blender users worldwide.

In this book we will markup keypresses as **AKEY**, **BKEY**, **CKEY**... and **ZKEY**. This will allow you to see what is done in a tutorial at a glance, once you know the shortcuts.

Keycombinations are marked up as **SHIFT-D** or **CTRL-ALT-A** for example.

The mouse buttons are nothing like keys and so are marked up as **LMB**, **MMB** and **RMB** for left, middle and right mouse button. It is recommended that you use Blender with a three button mouse. If you have a two button mouse you can substitute the middle mouse button by holding **ALT** and using the left mouse button (**LMB**).

References to interface elements (*GUI*, graphical user interface) are marked up in exclamation marks for example the "Load" Button.

Names from Blender's GUI and special Blender terms are written in a special way to make them stick out from the rest of the text. For example, the window showing the 3-D objects is called 3DWindow, other examples are ButtonsWindow, PaintFaceButtons or EditMode.

4.2. The Mouse

Blender is designed to be used with two hands: one hand using the keyboard, the other hand using the mouse. This prompts me to mention the 'Golden Rule of Blender':

Tip: *Keep one hand on your keyboard and one hand on your mouse!*

The mouse is particularly important because by using it you can control more than one axis at time. As far as possible, the mouse has the same functionality in all of Blenders's sections and windows.

Left Mouse Button (LMB)

With the left mouse button you can activate buttons and set the 3D-Cursor. Often "click and drag the left button" is used to change values in sliders.

Middle Mouse Button (MMB)

Tip: On systems with only two mouse buttons, you can substitute the middle mouse button with the ALT key and the left mouse button.

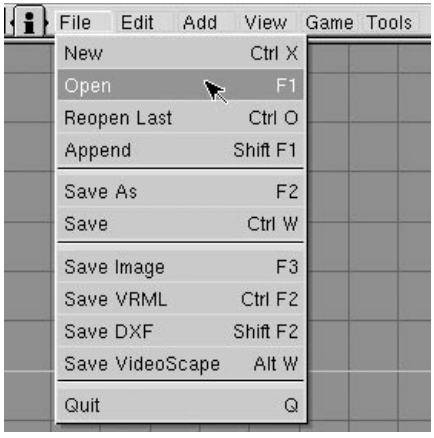
The middle mouse button is used predominantly to navigate within the windows. In the 3DWindow it rotates the view. Used together with **SHIFT** it drags the view, and with **CTRL** it zooms. While manipulating an object, the middle mouse button is also used to restrict a movement to a single axis.

Right Mouse Button (RMB)

The right mouse button selects or activates objects for further manipulation. Objects change color when they are selected. Holding **SHIFT** while selecting with the right mouse button adds the clicked object to a selection. The last selected object is the active object that is used for the next action. If you **SHIFT-RMB** an already selected object, it becomes the active object. One more click and you can de-select it.

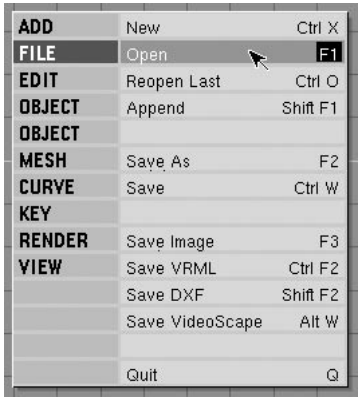
4.3. Loading and saving

Figure 4-2. FileMenu



In the Header of the InfoWindow, normally located on the top of the screen, you will find a menu. It offers you standard operations like file operations and changing of views.

Figure 4-3. Blender's main menu, the Toolbox



The **SPACE** key brings up the Toolbox, a large pop-up menu that offers you the most commonly used operations in Blender. The "FILE" entry allows you also to action file operations. Behind every command you will find the associated hotkey.

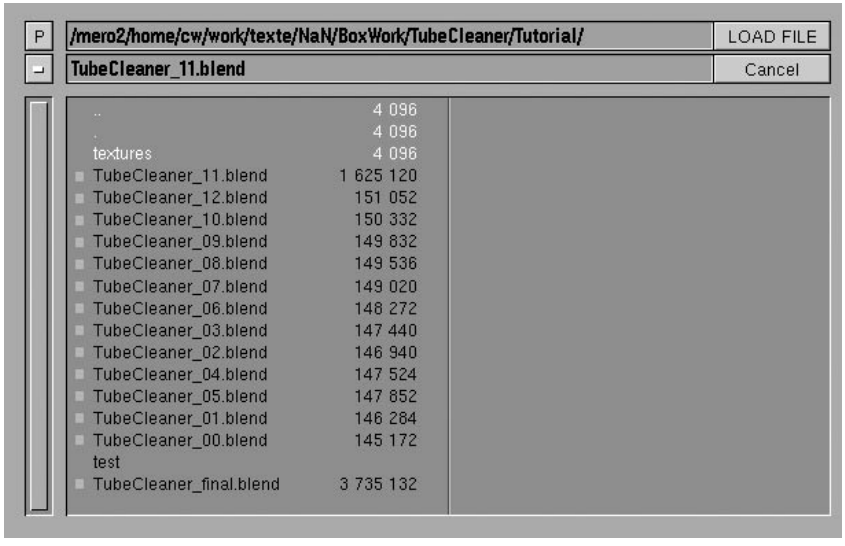
Tip: Use the toolbox to learn the hotkeys in Blender!

The most common file operations in Blender are the loading and saving of scenes. The quickest way to action these common functions is via the hotkeys: **F1** offers you a FileWindow to load a scene, **F2** a FileWindow to save a scene.

FileWindow

However you decide to initiate a file operation, you will always get its appropriate FileWindow.

Figure 4-4. Blender FileWindow



The main part of a FileWindow is the listing of directories and files. File types known by Blender are allocated a yellow square. A click with the **LMB** selects a file and puts the name into the filename-input. A **ENTER** or click on the "LOAD FILE" button will then load the file. Cancel the operation using **ESC** or the "Cancel" button. A **LMB**-click on a directory enters it. A shortcut to load files is the **MMB**, which quickly loads the file. You can also enter the path and filename by hand in the two inputs at the top of the FileWindow.

With the **RMB**, you can select more than one. The selected files are highlighted in blue.

Tip: *The PAD+ and PAD- keys increase and decrease the last number in a filename respectively. This is handy for saving versions while you work.*

The button labeled with a "P" at the upper left corner of the FileWindow puts you one directory up in your path. The MenuButton below it offers you the last directories you have visited, as well as your drives in Windows.

Figure 4-5. FileWindow Header with valuable information

The button labeled "A/Z" uses an alphabetical sorting, the clock button sorts by the file date, and the next button by the file size. Right of these buttons there is a piece of text that shows what kind of operation the FileWindow will do, e.g. "LOAD FILE".

The next button selects between long (size, permissions, date) and short filenames. The little ghost hides all files beginning with a dot. After that button, you have information about the free space remains on the disk, and how many megabytes big the selected files are.

Version control and backupfiles

Figure 4-6. Version control and backup settings in the InfoWindow

Blender follows a simple straightforward method to provide an "undo". When you enlarge the InfoWindow by pulling down the edge, you can see the controls for backups and version control.

With the activated "Auto Temp Save" button Blender will automatically write a backup after the number of minutes entered in the "Time:" button to the directory entered in the "Dir:" Button. Clicking "Load Temp" will load the last written temporary file.

When you write a file, Blender will keep the old file as `*.blend1` for backup. "Versions:" controls how many version files are written.

Beside these possibilities for disaster recovery, Blender writes a file `quit.blend` which contains your last scene into the temporary directory "Dir:" when you quit Blender.

4.4. Windows

All Blender screens consist of Windows. The Windows represent data, contain buttons, or request information from the user. You can arrange the Windows in Blender in many ways to customize your working environment.

Header

Every Window has a Header containing buttons specific for that window or presenting information to the user. As an example, the header of the 3DWindow is shown here.

The left-most button shows the type of the Window, clicking it pops up a menu to change the Window type.

The next button switches between a full screen and a tiled screen window. The button featuring a house graphic fills the window to the maximum extent with the information it is displaying.

Figure 4-7. HeaderMenu

A **RMB**-click on the Header pops up a menu asking you to place the Header at the "Top", the "Bottom", or to have "No Header" for that Window.

Click and hold with the **MMB** on the header, and then drag the mouse to move the header horizontally in case it doesn't fit the width of the window.

Edges

Every time you place the mouse cursor over the edge of a Blender window, the mouse cursor changes shape. When this happens, the following mouse keys are activated:

LMB

Drag the window edge horizontally or vertically while holding down the **LMB**. The window edge always moves in increments of 4 pixels, making it relatively easy to move two window edges so that they are precisely adjacent to each other, thus joining them is easy.

MMB or RMB

Clicking an edge with **MMB** or **RMB** pops up a menu prompting you to "Split Area" or "Join Areas".

"Split Area" lets you choose the exact position for the border. Split always works on the window from which you entered the edge. You can cancel the operation with **ESC**.

"Join Areas" joins Windows with a shared edge, if possible, which means that joining works only if Blender don't have to close more than one Window for joining.

4.5. The Buttons

Buttons offer the quickest access to DataBlocks. In fact, the buttons visualize a single DataBlock and are grouped as such. Always use a LeftMouse click to call up Buttons. The buttons are described below:

Blender button types

Button

This button, which is usually displayed in salmon color, activates a process such as „New" or „Delete".

TogButton

This button, which displays a given option or setting, can be set to either OFF or ON.

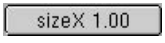
Tog3Button

This button can be set to off, positive or negative. Negative mode is indicated by yellow text.

RowButton

This button is part of a line of buttons. Only one button in the line can be active at once.

NumButton



This button, which displays a numerical value, can be used in three ways:

Hold the button while moving the mouse. Move to the right and upwards to assign a higher value to a variable, to the left and downwards to assign a lower value. Hold **CTRL** while doing this to change values in steps, or hold **SHIFT** to achieve finer control.

Hold the button and click **SHIFT-LMB** to change the button to a „TextBut". A cursor appears, indicating that you can now enter a new value. Enter the desired value and press **ENTER** to assign it to the button. Press **ESC** to cancel without changing the value.

Click the left-hand side of the button to decrease the value assigned to the button slightly, or click the right-hand side of the button to increase it.

NumSlider

Use the slider to change values. The left-hand side of the button functions as a „TextBut".

TextButton

This button remains active (and blocks the rest of the interface) until you again press **LMB**, **ENTER** or **ESC**. While this button is active, the following hotkeys are available:

ESC: restores the previous text.

SHIFT+BACKSPACE: deletes the entire text.

SHIFT+ARROWLEFT: moves the cursor back to the beginning of the text.

SHIFT+ARROWRIGHT: moves the cursor to the end of the text.

MenuButton

This button calls up a PupMenu. Hold **LMB** while moving the cursor to select an option. If you move the mouse outside of the PopUpMenu, the old value is restored.

IconButton

Button type „But“ activates processes.

IconToggle

Button type „TogBut“ toggles between two modes.

IconRow

As button type „RowBut“: only one button in the row of buttons can be active at once.

IconMenu

Click with **LMB** to see the the available options.

4.6. Windowtypes

DataSelect, SHIFT-F4

For browsing the data structure of the scene, and selecting objects from it.

3DWindow, SHIFT-F5

Main window while working in the 3D-space. It visualizes the scene from orthogonal, perspective, and camera views.

IpoWindow, SHIFT-F6

Creating and manipulating of so called IpoCurves, the animation curve system of Blender.

ButtonWindow, SHIFT-F7

The ButtonWindow contains all the buttons needed to manipulate every aspect of Blender. A brief overview follows after this section; for a more detailed explanation see the reference section of this manual.

SequenceEditor, SHIFT-F8

Post-processing and combining animations and scenes.

OopsWindow, SHIFT-F9

The OopsWindow (Object Oriented Programming System) gives a schematic overview of the current scene structure.

ImageWindow, SHIFT-F10

With the ImageWindow you can show and assign images to objects. Especially important with UV-texturing.

InfoWindow

The header of the InfoWindow shows useful information, it contains the menus and the scene and screen MenuButtons. The InfoWindow itself contains the options by which you can set your personal preferences.

TextWindow, SHIFT-F11

A simple text editor, mostly used for writing Python-scripts, but also a useful means by which you can insert comments about your scenes.

ImageSelectWindow

Lets you browse and select images on your disk. Includes thumbnails for preview.

SoundWindow, SHIFT-F12

For the visualization and loading of sounds.

ActionWindow

For editing the poses and animations of Armatures (Bones).

ButtonsWindow

The ButtonsWindow contains the buttons needed for manipulating objects and changing general aspects of the scene.

The ButtonsHeader contains the icons to switch between the different types of ButtonsWindows.

ViewButtons

The 3DWindow settings for a Window. It only features buttons if selected from a 3DWindow and will then provide settings for the grid or background images. Every 3DWindow can have its own settings.

 LampButtons, F4

The LampButtons will only display when a lamp is selected. Here you can change all of the parameters of a lamp, like its color, energy, type (i.e. Lamp, Spot, Sun, Hemi), the quality of shadows, etc.

 MaterialButtons, F5

The MaterialButtons appears when you select an object with a material assigned. With these clutch of buttons you can control every aspect of the look of the surface.

 TextureButtons, F6

These buttons let you assign textures to materials. These textures include mathematically generated textures, as well as the more commonly used image textures.

 AnimationButtons, F7

The AnimationButtons are used to control various animation parameters. The right section of the buttons are used for assigning special animation effects to objects, e.g. particle systems, and wave effects.

 RealTimeButtons, F8

These buttons are part of the real time section of Blender. This manual covers only linear animation.

 EditButtons, F9

The EditButtons offer all kinds of possibilities for you to manipulate the objects themselves. The buttons shown in this window depend on the type of object that is selected.

 WorldButtons

Set up global world parameters, like the color of the sky and the horizon, mist settings, and ambient light settings.

 Face/PaintButtons

These buttons are used for coloring objects at vertex level, and for setting texture parameters for the UV-Editor.

 RadiosityButtons

The radiosity renderer of Blender. Not covered in this manual.

ScriptButtons

Assigning of Python scripts to world, material, and objects (BlenderCreator).

DisplayButtons, F10

With the DisplayButtons you can control the quality and output-format of rendered pictures and animations.

4.7. Screens

Figure 4-8. Screen browse

Screens are the major frame work of Blender. You can have as many Screens as you like, each one with a different arrangement of Windows. That way you can create a special personal workspace for every task you do. The Screen layout is saved with the Scene so that you can have scene-dependant work spaces. An example of this is to have a Screen for 3-D work, another for working with Ipos and, a complete file manager to arrange your files and textures.

4.8. Scenes

Figure 4-9. Scene browse

Scenes are a way to organize your work and to render more than one scene in the Blender game engine for example to display a instruments panel overlay. Another possibility is to switch scenes from the game engine and this way changing levels of a game.

While you are adding a new scene, you have these options:

"Empty": create a completely empty scene.

"Link Objects": all Objects are linked to the new scene. The *layer* and *selection flags* of the Objects can be configured differently for each Scene.

"Link ObData": duplicates Objects only. ObData linked to the Objects, e.g. Mesh and Curve, are not duplicated.

"Full Copy": everything is duplicated.

4.9. Setting up your personal environment

With the possibilities listed above, you can create your own personal environment.

To make this environment a default when Blender starts, or after you reset Blender with **CTRL-X**, use **CTRL-U** to save it to your home directory.

4.10. Navigating in 3D

Blender is a 3-D program, so we need to be able to navigate in 3D space. This is a problem because our screens are only 2-D. The 3DWindows are in fact "windows" to the 3-D world created inside Blender.

4.10.1. Using the keyboard to change your view

Place your mouse pointer over the big window on the standard Blender screen. This is a 3DWindow used for showing and manipulating your 3D-worlds.

Info: *Remember that the window with the mouse pointer located over it (no click needed) is the active window! This means that only this window will respond to your key presses.*

Pressing **PAD1** (the number "1" key on the numeric pad) gives you a view from the front of the scene. In the default Blender scene, installed when you first start Blender, you will now be looking at the edge of a plane with the camera positioned in front of it. With holding the **CTRL** key (on some systems also **SHIFT** is possible), you can get the opposite view, which in this case is the view from the back (**CTRL-PAD1**).

PAD7 returns you to the view from the top. Now use the **PAD+** and **PAD-** to zoom in and out. **PAD3** gives you a side view of the scene.

PAD0 switches to a camera-view of the scene. In the standard scene you only see the edge of the plane because it is at the same height as the camera.

PAD/ only shows selected objects; all other objects are hidden. **PAD.** zooms to the extent of the selected objects.

Switch with **PAD7** back to a top view, or load the standard scene with **CTRL-X**. Now, press **PAD4** four times, and then **PAD2** four times. You are now looking from the left above and down onto the scene. The 'cross' of keys **PAD8**, **PAD6**, **PAD2** and **PAD4** are used to rotate the actual view. If you use these keys together with **SHIFT**, you can drag the view. Pressing **PAD5** switches between a perspective view and an orthogonal view.

Tip: *Use CTRL-X followed by ENTER to get a fresh Blender scene. But remember, this action will discard all changes you have made!*

You should now try experimenting a little bit with these keys to get a feel for their operation and function.

If you get lost, use CTRL-X followed by ENTER to get yourself back to the default scene.

4.10.2. Using the mouse to change your view

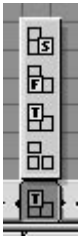
The main button for navigating with the mouse in the 3DWindow is the middle mouse button (**MMB**). Press and hold the **MMB** in a 3DWindow, and then drag the mouse. The view is rotated with the movement of your mouse. Try using a perspective view (**PAD5**) while experimenting -- it gives a very realistic impression of 3D.

With the **SHIFT** key, the above procedure translates the view. With **CTRL**, it zooms the view.

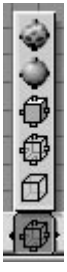
With the left-most icon, you can switch the window to different window types (e.g. 3DWindow, FileWindow, etc.). The next icon in the line toggles between a full screen representation of the window and its default representation. The icon displaying a house on it zooms the window in such a way that all objects become visible.

Next in the line, including the icon with the lock on it. We will cover this later on in the manual.

The next icon switches the modes for the local view, and is the mouse alternative for the PAD/ key. With the following icon you can switch between orthogonal, perspective, and camera views (keys PAD5 and PAD0).



The next button along toggles between the top, front, and side views. SHIFT selects the opposite view, just as it does when you use the keypad.



This button switches between different methods of drawing objects. You can choose from a bounding box, a wireframe, a faced, a gouraud-shaded, and a textured view.

With these icons you can translate and zoom the view with a **LMB** click on the icon and a drag of the mouse.

This overview should provide you with an idea of how to look around in 3D-scenes.

4.11. Selecting of Objects

Selecting an object is achieved by clicking the object using the right mouse button (**RMB**). This operation also de-selects all other objects. To extend the selection to more than one object, hold down **SHIFT** while clicking. Selected objects will change the color to purple in the wireframe view. The last selected object is colored a lighter purple and it is the *active* object. Operations that are only useful for one object, or need one object as reference, always work with the active object.

Objects can also selected with a `border'. Press BKEY to action this, and then draw a rectangle around the objects. Drawing the rectangle with the LMB selects objects; drawing with RMB deselects them.

Selecting and activating

Blender makes a distinction between *selected* and *active* .

Only one Object can be *active* at any time, e.g. to allow visualization of data in buttons. The active *and* selected Object is displayed in a lighter color than other selected Objects. The name of the active Object is displayed in the InfoHeader.

A number of Objects can be *selected* at once. Almost all key commands have an effect on *selected* Objects.

A single **RMB** click is sufficient to select and activate an Object. All other Objects (in the visible layers) are then *de-selected* in order to eliminate the risk of key commands causing unintentional changes to those objects. All of the relevant buttons are also drawn anew. Selections can be extended or shrunk using **SHIFT+RMB**. The last Object selected (or deselected) then becomes the *active* Object. Use Border Select (BKEY) to more rapidly select a number of Objects at one time. None of the Objects selected using this option will become *active*.

4.12. Copying and linking

Blender uses a object oriented structure to store and manipulate the objects and data. This will affect the work with Blender in many places. For example, the copying of objects or the use of Blender Materials.

In this structure an object can have its own data (in case of the Blender Game Engine Polygon-Meshes) or share this Mesh with more other objects.

So what is the advantage of that system?

1. Reduced size of the scene in memory, on disk or for publishing on the web
2. Changes on the ObData inherits to all Objects on the same time. Imagine you decide to change a house objects you have 100 times in your scene or changing the Material properties of one wall
3. You can design the logic and gameplay with simple place-holder objects and later swap them against the finished objects with a click of the mouse
4. The shape of objects (the MeshData) is changeable at runtime of the game without affecting the object or its position itself

Copy

The copy operation you are familiar with from other applications makes a true duplicate of the selected objects. Copying is done fastest with the keycommand **SHIFT-D** or also with the “Duplicate” entry in the Edit-Menu.

Linked Copy

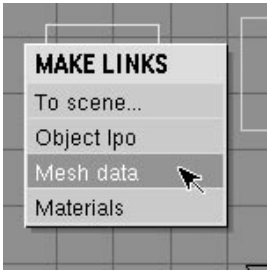
A linked copy is achieved by using the **ALT-D** key command. Unlike copying with **SHIFT-D**, the mesh forming the object is not duplicated, but rather linked to the new objects.

User Button

Another common method to create and change links and Blender interface element is the UserButton . This MenuButton allows to change links by pressing and holding the left mouse on it and choose a link from the appearing menu. If there are more possibilities than the Menu can hold, a DataBrowseWindow is opened instead.

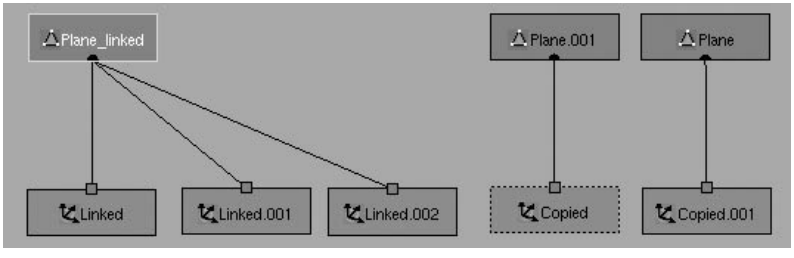
If an Object has more than one user, the UserButton will be blue and a number indicates the number of users (in the above image three). Selecting this number will make a copy of the Data and makes the object "Single User".

Linking



To link Data from the active to the selected Objects can be done with the key command **CTRL-L**. A menu will ask what data you want to link. This way you can choose to link the objects between scenes, or link Ipos (animation curves), MeshData or Materials.

Figure 4-10. Object visualisation in the OOPSWindow



The object-structure created by copy or linking actions can be visualized in the OOPSWindow **SHIFT-F9**. Here, the object "Linked" was copied two times with **ALT-D** and you can see that all three objects (Blender automatically generates unique names by appending numbers) are linked to the same MeshData "Plane_linked". The object "Copied" was copied with **SHIFT-D** resulting in two objects with their own MeshData.

4.13. Manipulating Objects

Most actions in Blender involve moving, rotating, or changing the size of certain items. Blender offers a wide range of options for doing this. See the 3DWindow section for a fully comprehensive list. The options are summarized here.

Grab

GKEY, Grab mode. Move the mouse to translate the selected items, then press **LMB** or **ENTER** or **SPACE** to assign the new location. Press **ESC** or **RMB** to cancel. Translation is always corrected for the view in the 3DWindow.

Use the middle mouse button to limit translation to the X, Y or Z axis. Blender determines which axis to use, based on the already initiated movement.

RMB and hold-move. This option allows you to select an Object and *immediately* start Grab mode.

Rotate

RKEY, Rotation mode. Move the mouse around the rotation center, then press **LMB** or **ENTER** or **SPACE** to assign the rotation. Press **ESC** to cancel. Rotation is always perpendicular to the view of the 3DWindow.

The center of rotation is determined by use of these buttons in the 3DWindowheader. The left-most button rotates around the center of the bounding box of all selected objects. The next button uses the median points (shown as yellow/purple dots) of the selected objects to find the rotation center. The button with the 3DCursor depicted on it rotates around the 3DCursor. The last button rotates around the individual centers of the objects.

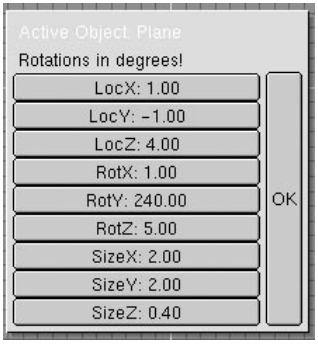
Scale

SKEY, Scaling mode. Move the mouse from the rotation center outwards, then press **LMB** or **ENTER** or **SPACE** to assign the scaling. Use the MiddleMouse toggle to limit scaling to the X, Y or Z axis. Blender determines the appropriate axis based on the direction of the movement.

The center of scaling is determined by the center buttons in the 3DHeader (see the explanation for the rotation).

While in scaling mode, you can mirror the object by pressing **XKEY** or **YKEY** to mirror at the x- or y-axis.

NumberMenu



To input exact values, you can call up the NumberMenu with **NKEY**. **SHIFT-LMB**-click to change the buttons to an input field and then enter the number.

EditMode

When you add a new object with the Toolbox, you are in the so-called *EditMode*. In EditMode, you can change the shape of an Object (e.g. a Mesh, a Curve, or Text) itself by manipulating the individual points (the vertices) which are forming the object. Selecting works with the **RMB** and the BorderSelect **BKEY** also works to select vertices. For selecting more vertices there is also CircleSelect, called by pressing **BKEY-BKEY**. "Painting" with the left mouse button selects vertices, painting with the middle button deselects.

While entering EditMode, Blender makes a copy of the indicated data. The hotkey **UKEY** here serves as an undo function (more accurately it restores the copied data).

As a reminder that you are in EditMode, the cursor shape changes to that of a cross.


001
003
2.03beta
004
2.04
game Blender
005
010
2.10
011
2.12
blenderGameKit
blender
POWER
CREATE
A
B
C
D
2.20
022
023
2.23


_intro ::
quickstart

Chapter 1. Quickstart

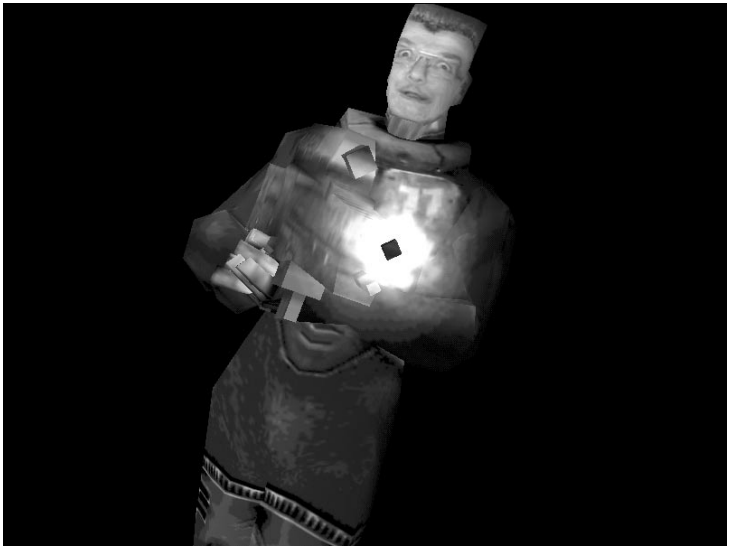


Figure 1-1. Calli going mad...

Have you ever wanted to personalize a computer game? Well, many game level editors will give you that possibility, but Blender goes a step further, by allowing you to create a completely *new* game.

In this quick-start chapter, I will show you how to map a face onto a game character.

The game character used here was made by Reevan McKay. You can read more about this in Chapter 22, which will show you many other things about character animation.

In Figure 1-1 you can see an image of an real-time 3-D animation created using the method which will be briefly described in this chapter. The scene is on the CD and called

`Tutorials/Quickstart/CalliGoingMad1.blend`.

This quick-start tries to be as self-contained as possible. Although it is good if you already know something about graphics, if you follow the instructions step-by-step all should go well.

Note: *If you have not installed Blender yet, please do so. The installation process is described in Section 29.1. Further hints about graphics hardware are given in Section 29.2.*

1.1. Simple face mapping

This section will show how to put a new face onto a ready-made character, there are some drawbacks to this method but it will get you started quickly.

Start Blender by double clicking its icon. It will open a screen as shown in Figure 1-2.

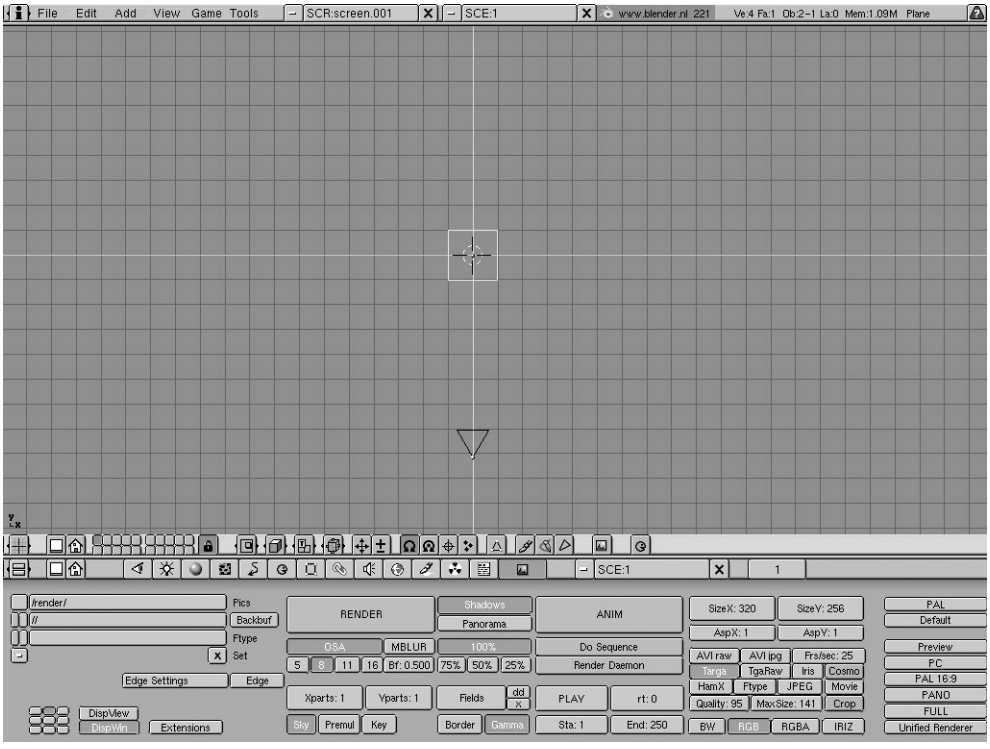
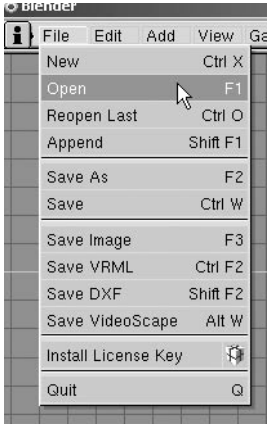
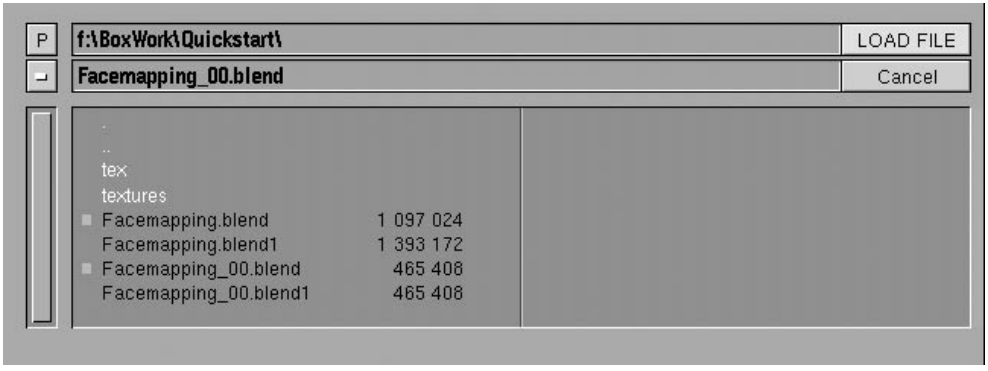


Figure 1-2. Blender just after starting it



Locate the file menu on the top left of the screen and choose “Open” by clicking it with the left mouse button (*LMB*). A big FileWindow appears which is used for all Blender loading and saving operations.

Figure 1-3. Blender FileWindow



The button labeled with a "P" at the upper left corner of the FileWindow puts you one directory up in your path. The MenuButton below brings you back to the last directories you have visited, as well as your mapped drives in Windows. Click and hold it with the left mouse button to change to your CDROM.

Now enter the directory `Tutorials/Quickstart/` and click with the left mouse on `Facemapping_00.blend`. Confirm your selection by clicking "LOAD FILE" at the top right of the FileWindow. Blender will load the file needed for the tutorial.

Note: *Please have a look at Section 4.1 for a explanation on how we will call interface elements and keyboard shortcuts (i.e. **PKEY**) in the tutorials.*

To have a quick look what this file is about, press **CTRL-RIGHTARROW**. The window layout changes to a bigger view of the character. Now press **PKEY** and the game engine will start. Using the controls from Table 1-1 walk around to have a closer look at the character.

Table 1-1. Quick-start controls

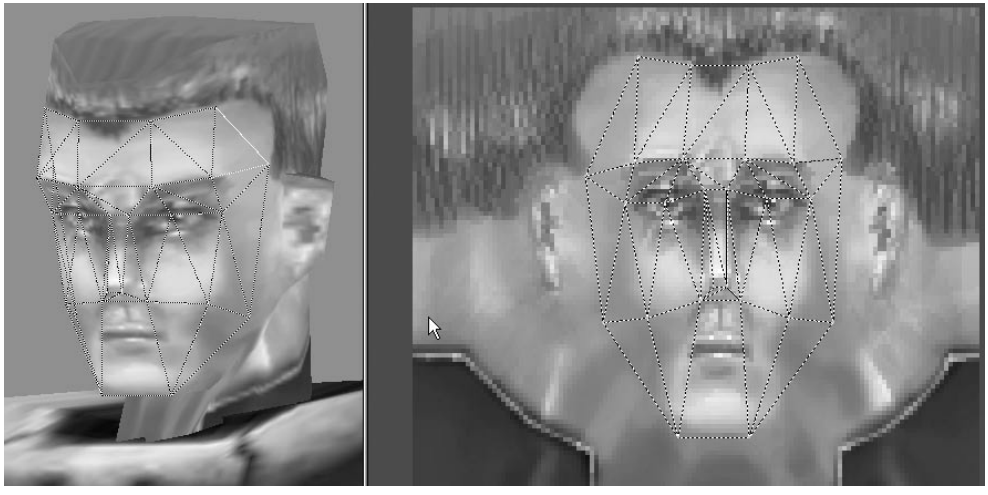
Controls/Keys	Description
WKEY	Move forward
DKEY	Move left
AKEY	Move right
SKEY	Move backwards
CTRL	Shoot
SPACE	Duck

Stop the game engine by pressing **ESC** when you have seen enough. Press **CTRL-LEFTARROW** to return to the window layout which we will now use to map a different face.

Move your mouse cursor over the left window with the 3-D view of the head and press **FKEY**. This will start the so-called "FaceSelectMode", which is used to manage and change textures on objects.

All polygons which belong to the face are now outlined and you can see them also in the right view showing the 2-D texture image of the face. This procedure is called *mapping* and will make the 2-D image appear where we want it on the 3-D object.

Figure 1-4. 3-D head and 2-D facemap

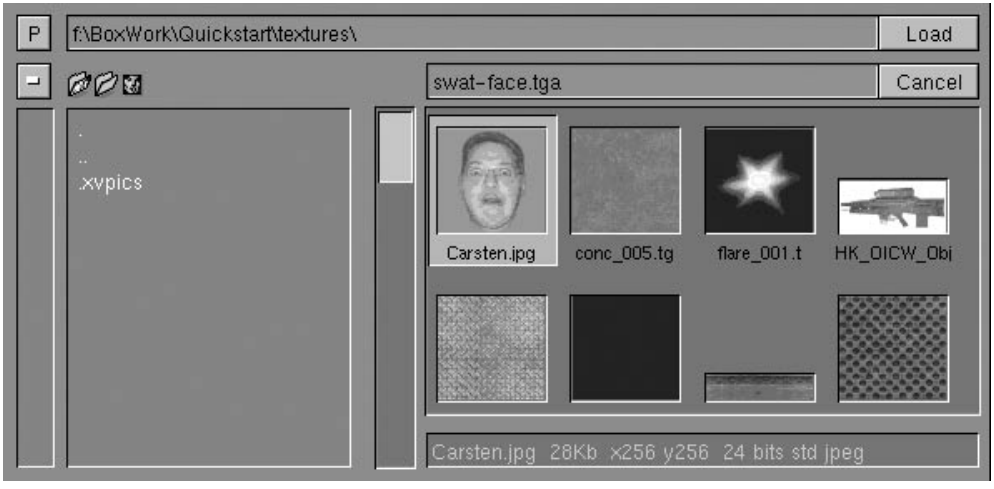


Locate the "Load" Button 

in the right ImageWindow and click it with the left mouse button. A FileWindow (in this case an ImageFileWindow) will open and lets you browse through your harddiscs and the CDROM again. Go to the directory `Tutorials/Quickstart/textures/`. The ImageFileWindow displays little thumbnail images to ease the choice of images (see Figure 1-5).

Tip: *You can also choose a picture of you or an other person. But if you are a beginner, I would suggest to use the supplied image for your first attempt. Blender can read and write Targa (*.tga) and JPEG (*.jpg) which are both common fileformats.*

Figure 1-5. ImageFileWindow



Click on the image **`Carsten.jpg`** (yes, its me, your tutorial writer) and click the "Load" Button on the top right of the ImageFileWindow to load it. The image will immediately appear in the 3-D view to the left.

Info: *Depending on your screen resolution you may need to zoom the right ImageWindow out a bit. Use the PAD- and PAD+ keys for zooming.*

The dimensions of my ugly face don't fit the previous mapping, so it'll look a bit distorted. Also, the color may not match exactly, making it look like a cheap mask.

Now move your mouse over the ImageWindow on the right and press **AKEY**, this selects (yellow color) all the control points here, called *vertices* in Blender. Now press **GKEY** and move your mouse, and all vertices will follow and you can watch the effect on the 3DView. Try to position the vertices in the middle of the face, using the nose as a reference. Confirm the new position with the left mouse button. If you want to cancel the move, press the right mouse button or **ESC**.

Info: *To have a better look at the head in the 3DView, you can rotate the head around using the middle mouse button (if you are using a 2 button mouse, hold ALT and use the left mouse button) and moving the mouse.*

To refine the placement of the texture on the head, you may now need to move the vertices more. Move your mouse over the ImageWindow on the right and press **AKEY** to de-select all vertices (they will turn purple). Now press **BKEY**. This will start the BorderSelect, and a crosshair will appear. Press and hold the left mouse button to draw a rectangle around vertices you want to select and release the mouse button. Now you can move these vertices by pressing **GKEY** and using the mouse. Press **LMB** to confirm the move. Control the effect by watching the head on the 3DView.

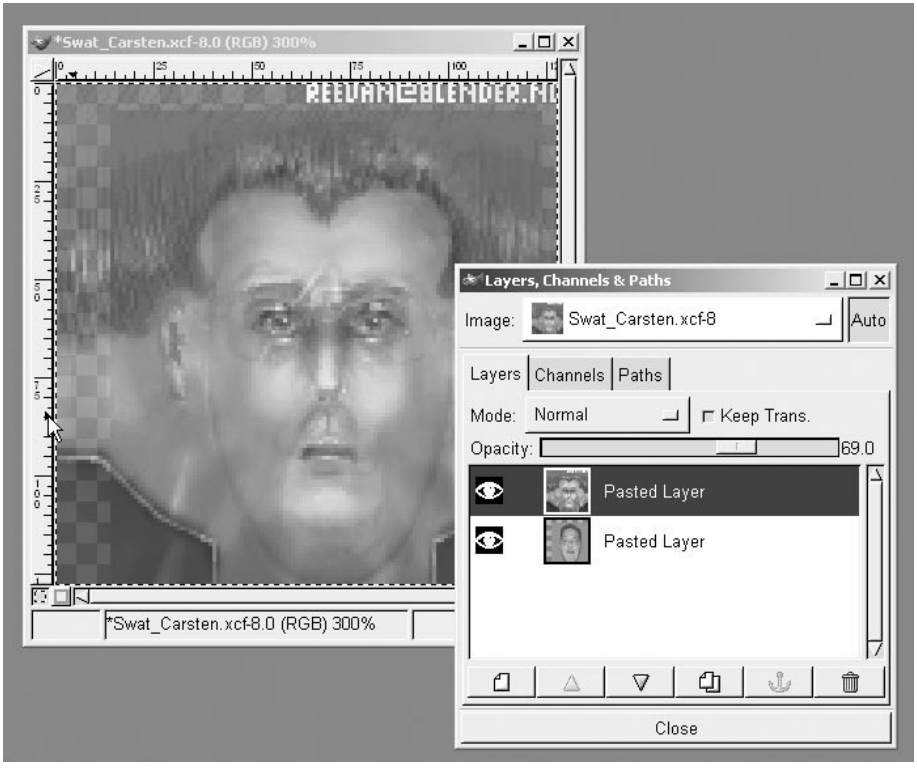
Info: *Don't give up too soon! Mapping a face needs practice, so take a break and play with the games on the CD, and try again later.*

If you want to look at your creation, switch to the full screen scene by pressing **CTRL-RIGHTARROW** and start the game engine with **PKEY.**

1.2. Using 2-D tools to map the face

This part of the tutorial will give you a brief guide on how to use a 2-D painting program to montage a face into the facemap. You should know how to work with layers in your application (if not please consult the documentation of your image editing program). I use the free software (GPL) GIMP (http://www.gimp.org/) but of course any other image manipulation programs (which supports layers) will do.

1. Load the image **`swat-face.tga`** and the face you want to use in your paint program.
2. Place the new face in a layer below the "swat-face.tga" and make the upper layer slightly transparent so that you can see the new face shining through.



3. Scale and move the layer containing the new face so that it fits to the "swat-face.tga" layer. Use the eyes, mouth and the nose to help you match them up. Also try to match the colors of the layers using the color tools of your 2-D program.
4. Make the upper layer non transparent again
5. Now use the eraser from your 2-D paint program to delete parts of the upper layer, the new face will appear at these points. Use a brush with soft edges so that the transition between the two layers is soft.
6. Collapse the layer to one and save the image as a Targa (`*.tga`) or JPEG (`*.jpg`) image. Maybe do some final touch-ups on the collapsed image, like blurring or smearing areas of transition.

Now load the scene `Facemapping_00.blend` from the CD. Press **FKEY** with your mouse over the 3DView on the left to enter FaceSelectMode.

Move your mouse to the right over the ImageWindow and click on the "Replace" button this time. This will replace the current texture in the whole file with your self-made texture. Find the map with your face on your disk, select it with the left mouse button and press "Load" in the ImageFileWindow. The new texture will now appear on the head.

Switch to the full screen again (**CTRL-RIGHTARROW**) and test the scene by starting the game engine with **PKEY**.


001
003
2.03beta
004
2.04
game Blender
005
010
2.10
011
2.12
blenderGameKit
blender
POWER
CREATE
A
B
C
D
2.20
022
023
2.23


_part II ::
playing with 3d
games technology

Using the „Pumpkin-Run“ example file, most of the core techniques for making a 3D game will be explained and illustrated. However, we can‘t make you a professional game designer in a few pages, or even with a complete book. But you can learn the basics here and have fun at the same time! You are also encouraged to experiment and play with Blender.

Things you will learn here:

- Loading and saving Blender scenes
- Manipulating objects and navigating in the scene
- Basic Texture mapping
- Playing interactive 3-D in Blenders integrated 3-D engine
- Adding interactivity to control game elements
- Camera control and lights
- Object animation
- Adding and using sound

And there is more! Many things will be covered in later chapters. But don‘t despair the most important thing is the gameplay. Even technically simple games can be entertaining for long times (“Tetris” for example). So concentrate on making the game fun for others or just enjoy creating stuff yourself!

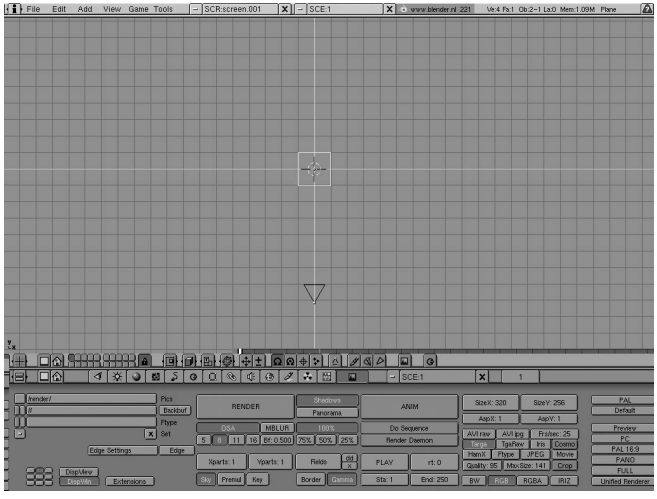
Advanced topics covered by later chapters

- Character animation the art of bring computer models to life. This complex topic demands many different capabilities from the game designer. This includes texturing, modeling, animation, good knowledge of natural motions etc.
- Special effects like bullets, explosions and similar things.
- Overlay interfaces and multiple scenes.
- Python scripting to simplify complex game logics. Python is a modern and efficient scripting language, which is integrated into Blender. Complex things can often be simplified with a few lines of Python.

Chapter 5. Modeling an environment

Start Blender by clicking its Icon. Blender will start with its default scene as shown in Figure 5-1.

Figure 5-1. Blender just after starting it



The big window is the 3DWindow, our window to the world of 3-D inside Blender scenes. The pink square is a simple plane, drawn in *wireframe*. We are currently looking onto the scene from above, a so called "TopView". The triangle is the representation of a Blender Camera.

Now move your mouse cursor over the Camera and press your right mouse button, this selects the Camera.

Info: *Blender uses the right mousebutton (RMB) for selecting objects!*

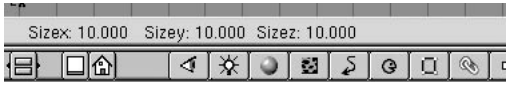
Now we will change the view of the scene. Move the mouse cursor into the big 3DWindow and press and hold the middle mouse button (**MMB**) and move the mouse to rotate the view.

Tip: *Blender is designed to work best with a three buttons mouse. However, if you have only a two buttons mouse you can substitute the middle mouse button by holding ALT and the left mouse button (ALT-LMB).*

Now return to the TopView of the scene by pressing **PAD-7**. These actions should give you a basic idea of how navigating in the 3-D space through a 2-D window works. More can be read on this topic in Section 4.10.

Select the plane again by pressing **RMB** with your mouse over it. The plane will be drawn in pink when your selection has been successful. We will now change the plane by scaling it.

Figure 5-2. Scaling info in the 3DWindow Header



Move the mouse over the selected plane, press **SKEY**, and move the mouse. You can see that the plane changes its size according to your mouse moves. Now hold **CTRL** while moving the mouse. The scale will only change in steps of 0.1. Scale the plane until the size is 10.0 for all axes. To do so look at the scaling information in the bar below the 3DWindow (see Figure 5-2) then press the left mouse button to finish the scaling operation.

Info: *If you can't scale to 10.0 or want to stop the scaling action, press **RMB** or **ESC**. Furthermore, **ESC** will abort every Blender procedure without making any changes to your object*

Figure 5-3. Splitting a window

I will now show you how to customize the Blender screen, and especially, the window layout. Move your mouse slowly over the lower edge of the 3DWindow (see Figure 5-3) until it changes to a double arrow. Now press the **MMB** or **RMB**, a menu will appear:

Click on "Split Area". Move the appearing line to the middle of the 3DWindow and press **LMB**, Blender splits the 3DWindow into two identical views of the 3-D scene.

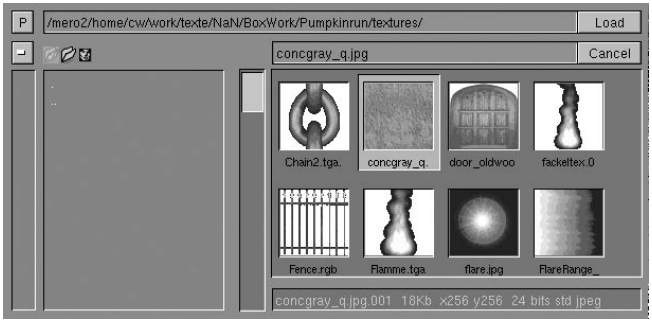
Move your mouse over the right window and press **SHIFT-F10**. The window will change to an ImageWindow this is the place in Blender to work with images and textures, which will color our real-time models.

Info: *All keypresses in Blender will be executed by the active window (that is the window with the mouse over it). There is no need to click a window to activate it.*

Move your mouse back to the plane in the left 3DWindow and select it again in case it is not selected anymore (i.e. not pink). Now press **ALT-Z**, the plane is now drawn in solid black. Press **FKEY** and the plane will turn white, the edges are drawn as dotted lines. With **FKEY**, we just entered the so called *FaceSelectMode*, used for selecting face and applying textures to models.

Move your mouse to the right window and locate and press the "Load" Button with **LMB**.

Figure 5-4. Thumbnail images in the ImageFileWindow



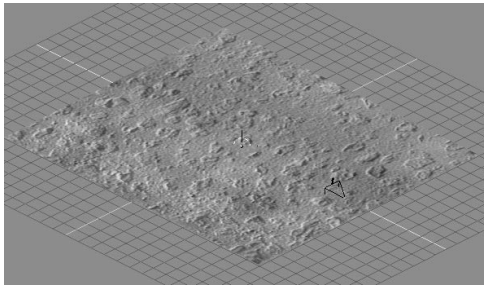
An ImageSelectWindow (Figure 5-4) opens.

Pressing and holding the MenuButton ⌟ with the left mouse button (**LMB**) will give you a choice of recently-browsed paths and, on Windows operating systems, a list of your drives.

The directory you are currently in, is shown in the top text-field. The ParentDir button P allows you to go up one directory.

Using these methods, go to your CD-ROM drive and browse for the folder `Tutorials/Pumpkinrun/textures/` and locate the `concgray_q.jpg` thumbnail. Click on it with the left mouse button and then choose "Load" from top right of the ImageSelectWindow.

Figure 5-5. Textured plane in 3DWindow



The texture now shows up in the 3DWindow to the left. If you see some strange colors in the texture, press **CTRL-K** over the 3DWindow. Now leave FaceSelectMode by pressing **FKEY**.

We have just created a very simple environment, but we used many of the steps needed to create more complex game levels.

It is now time to save your scene. To ease the process we will include the texture in the saved scene. To do so, choose "Pack Data" from the Tools-Menu. A little parcel-icon will appear in the menu bar to indicate that this scene is packed. Now use the FileMenu to browse to your harddisk (as described above), enter a name in the filename field (currently "untitled.blend") and click the "SAVE FILE" Button in the FileWindow. You can read more about saving and loading in Section 4.3.

Chapter 6. Appending an object from an other scene

Because we can't cover modeling and the general use of Blender as a tool to create whole worlds, we load ready-made objects. In fact, there is no special file-format in Blender to store objects, so all scenes can be used as archives to get objects from. So you can browse and re-use all the nice scenes on the CD-ROM.

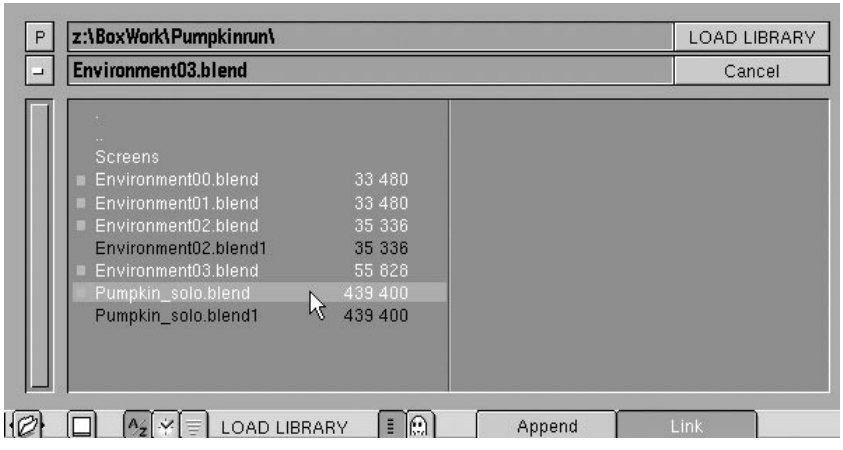
Figure 6-1. The hero!

Move your mouse over the ImageWindow (the right one from the last step) and press **SHIFT-F5** to change it to a 3DWindow. We now go into some action and need more views into 3-D space.

Press **SHIFT-F1** with your mouse over one of the 3DWindows, a FileWindow will appear in "Append" mode (see Figure 6-2), which allows us to load any Blender object from a scene into the current scene.

Info: *You can also use the FileMenu of Blender to access the Append function, but mostly a shortcut is faster.*

Figure 6-2. FileWindow in append mode

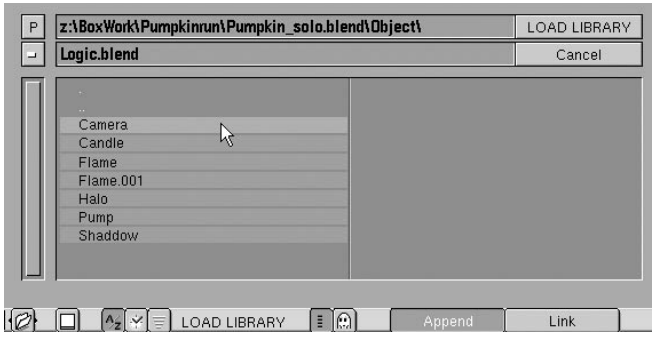


Pressing and holding the MenuButton with the left mouse button (**LMB**) will give you a choice of recently browsed paths and, on Windows operating systems, a list of your drives.

The directory you are currently in is shown in the top text-field. The ParentDir button P allows you to go up one directory.

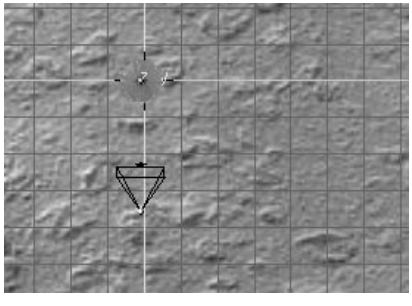
Using these methods, go to your CD-ROM drive and browse for the folder **`Tutorials/Pumpkinrun/`**. Now click with the left mouse button (**LMB**) on the filename "**`Pumpkin_Solo.blend`**". The file will be opened immediately.

Figure 6-3. Browsing in a *.blend file



When you have entered the file, the FileWindow will present you all the parts of the scene like in a file-browser. Now click on "Objects" with **LMB**. You will see the objects contained in that scene (see Figure 6-3). Select all objects by pressing **AKEY**. Confirm by pressing **ENTER** or clicking with **LMB** on the "LOAD LIBRARY" button.

Figure 6-4. The pumpkin in TopView after loading it into the environment



You can now see the pumpkin as an orange spot, sitting in the middle of the plane in the left 3DWindow.

Switch the right window to a second 3DWindow by pressing **SHIFT-F5** with the mouse over the window. You will get the same TopView as in the left 3DWindow but drawn in Wireframe.

We appended a Camera together with the objects we did append. Now select the Camera closer to the pumpkin with **RMB**. This is best done in the wireframe view. Move your mouse back to the left (textured) 3DWindow and press **CTRL-PAD0**. This changes the camera to the selected one and gives a nice view of the character.

Chapter 7. Start your (Game) Engines!

We can now start the Blender game engine already! While pointing with the mouse over the CameraView, press **PKEY** and you can see the pumpkin on our textured ground. The pumpkin character has an animated candle inside and you will see it flicker. To stop the game engine and return to BlenderCreator press **ESC**.

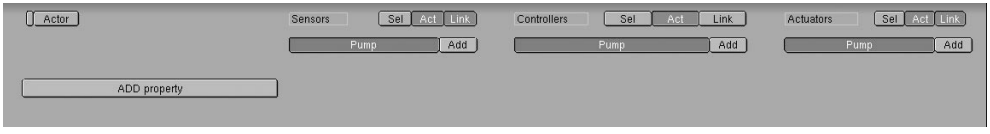
I hear you saying "That's nice, but where is the animation?" Well, give me a minute.

Move your mouse over the right 3DWindow and press **PAD3** to get a view from the side. Zoom into the view by pressing **PAD+** a few times or hold **CTRL-MMB** and move the mouse up, which will give you a smooth zoom. You also can move the view with the **MMB** and mouse movements while holding **SHIFT**. This way we prepare the view to move the pumpkin up.

Select the character with the **RMB** (click somewhere on the wireframe of the pumpkin), and it will turn pink to indicate that it is selected.

We will now enter the main command center for interactive 3-D in Blender. To do so press **F8** or click the RealtimeButtons icon  in the iconbar.

Figure 7-1. The RealtimeButtons

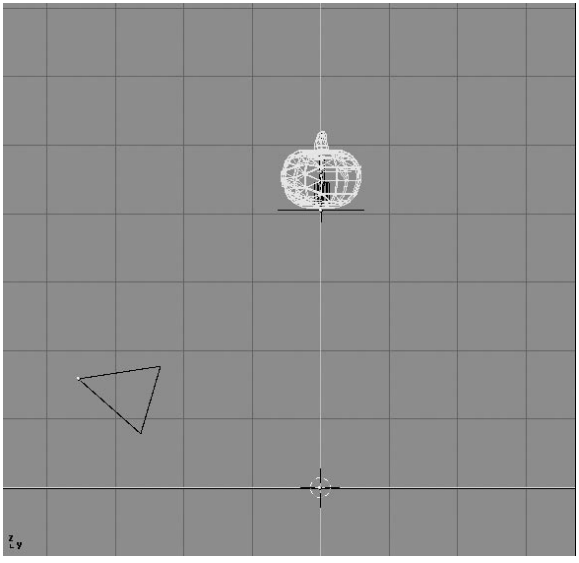


Locate the "Actor" button on the left in the RealtimeButtons and click it with the **LMB**. This makes our character in essence an actor. Two more buttons appear, so click on the "Dynamic" button. This changes the object in such a way that it reacts to physical properties, like gravity, bounce or forces applied to it. We won't take any notice of the bunch of buttons which appeared while clicking "Dynamic" for now.

If you now start the game engine you will not see much difference, but we will change that in a minute.

Zoom the right 3DWindow out a bit (do you remember? Use **CTRL-MMB** or **PAD+/ PAD-** to zoom). Make sure that the pumpkin is still selected (pink, if not then click it with **RMB**), press **GKEY** over the right 3DWindow and move the mouse. The character will follow your mousemovements in the 3DWindow. The **GKEY** starts the so called "GrabMode" which allows you to move objects within the 3-D space.

Figure 7-2. SideView after moving the pumpkin up



Move the object straight up until it disappears on the top of the CameraView (left 3DWindow) and confirm the new position with **LMB**. If you are unsure you can always cancel the operation with **ESC** or **RMB** and try again.

Now move the mouse to the left 3DWindow (the CameraView) and press **PKEY** to start the game engine. The Pumpkin falls and bounces nicely until it rests on the ground. Press **ESC** to exit the game engine

Chapter 8. Interactivity

The RealtimeButtons (**F8**) are logically divided into four columns. We have already used the leftmost column to set up the object parameters to make the pumpkin fall. The three right columns are used for building the interactivity into our game.

So lets move the pumpkin at our request.

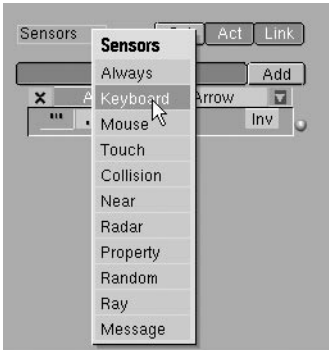
The columns are labeled as “Sensors”, “Controllers” and “Actuators”. You can think of Sensors as the senses of a life form, the Controllers are the brain and the Actuators are the muscles.

Press the “Add” button once for each row with the left mouse button to make one LogicBrick for the Sensors, Controllers and Actuators (see Figure 8-1).

Figure 8-1. Newly-created LogicBricks

The types of the added LogicBricks are nearly correct, for our first task, only the first one needs a change. Press and hold the MenuButton now labeled with “Always” and choose “Keyboard” from the pop up menu (see Figure 8-2).

Figure 8-2. Changing the LogicBrick type



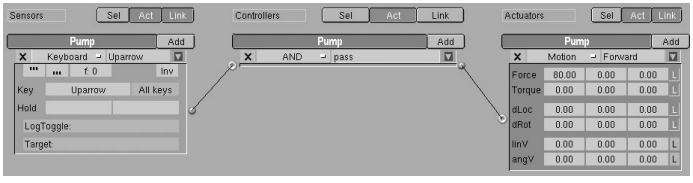
Now **LMB** click into the “Key” field of the Keyboard Sensor. The text “Press any key” appears. Press the key you want to use to move the player forward (I suggest **UPARROW**).

Now have a closer look at the Motion Controller. We will now define how the player should move. The first line of numbers labeled “Force” defines how much force will be applied when the Motion Controller is active. The three numbers stand for the forces in X, Y, and Z-Axis direction.

If you look closely at the wire frame view of the player you can see that the X-axis is pointing forward on the player. So to move forward we need to apply a positive force along the X-axis. To do so, click and hold on the first number in the "Force" row with the left mouse. Drag the mouse to the right to increment the value to 80.00. You can hold the **CTRL** key to snap the values to decadic values. Another way to enter an exact value is to hold **SHIFT** while clicking the field with the left mouse. This allows you to enter a value using the keyboard.

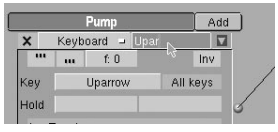
Having nearly created the configuration shown in Figure 8-3, we now need to "wire" or connect the LogicBricks. The wires will pass the information from LogicBrick to LogicBrick, i.e. from a Sensor to a Controller.

Figure 8-3. LogicBricks to move the player forward



Click and hold the left mouse button on the yellow ball attached on the Keyboard Sensor and drag the appearing line to the yellow ring on the AND Controller. Release the mouse and the LogicBricks are connected. Now connect the yellow ball on the right side of the AND Controller with the ring on the Motion Controller.

To delete a connection, move the mouse over the connection. The line is now drawn highlighted and can be deleted with an XKEY or DEL key-press.



Tip: Always name your Objects and LogicBricks, this will help you to find your way through your scenes and refer to specific LogicBricks later. To name a LogicBrick click into the name field with **LMB** (see figure above) and enter the name with the keyboard. However, Blender will name objects and LogicBricks automatically with generated unique names like "sensor1", "sensor2" or "act", "act1" etc., so you don't have to fear about name-collisions.

Now press PKEY to start the game engine and when you press the UPARROW key briefly, the player moves towards the camera.

To make the movement more interesting, we can now add a jump. To do so enter a 20.0 in the third (z-Axis, up) field of the Motion Controller. If you try it again in the game engine, you can see that there is a problem: If you hold the key pressed, the pumpkin will take off into space. This is because you also apply forces when in the air.

To solve this we have to ensure that the forces only get applied when the pumpkin touches the ground. That's where the Touch Sensor kicks in. Add a new Sensor by clicking on "Add" in the Sensors row. Change the type to "Touch" like you did for the Keyboard Sensor (see Figure 8-2).

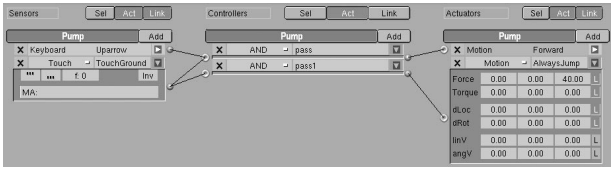
Wire the Touch Sensor to the AND Controller. Now the Keyboard and the Touch Sensor are connected to that controller. The type "AND" of the controller will only trigger the Motion Actuator when the key is pressed AND the player touches the ground. This is an easy way to add logic to your interactive scenes. As well as the

AND Controller there are also OR, Expression and Python (Blender's scripting language) and Controllers which all offers more flexibility to make your game-logic.

Tip: At this moment, space in the RealtimeButtons can get sparse. But besides changing the window layout we also can collapse the LogicBricks. To do so press the little orange arrow right beneath the brick's name (so that you are still able to see the connections, though the content is hidden).

To make the movement more dynamic, we will now add LogicBricks to make the pumpkin jump constantly. Add a new Controller and a new Actuator by clicking "Add" in the appropriate row. Name the new Actuator "AlwaysJump". Wire the Touch Sensor with the new AND Controller input and the output of the Controller to the new Motion Actuator "Always Jump".

Figure 8-4. LogicBricks for adding a constant jump



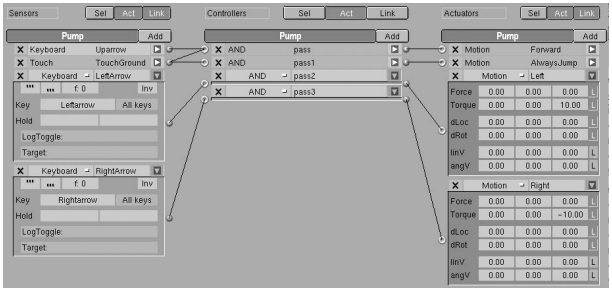
Yes, not only one Controller can be connected to two Sensors, a sensor can also "feed" two or more controllers. Start the game again with **PKEY**, the pumpkin jumps, **UPARROW** moves it forward.

More control

Now we add more LogicBricks to steer the player with the Cursorkeys.

Add a new Sensor, Controller and an Actuator by clicking on the "Add" Buttons. Change the Sensor type to "Keyboard" with the MenuButton. Don't forget to name the LogicBricks by clicking on the name field in the bricks. Wire the Sensor ("LeftArrow") with the Controller ("pass2") and the Controller output with the Actuator ("Left")

Figure 8-5. LogicBricks to steer the player



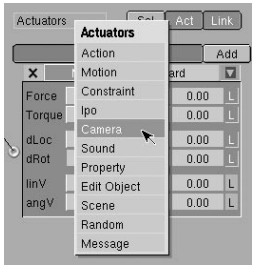
Enter "10.0" in the third field (Z-axis) in the "Torque" row. Torque is the force that turns the object. In this case it will turn the actor around its longitudinal axis. Try the change in the game engine, the pumpkin will turn left when you press **LEFTARROW**. Repeat the steps but change it to turn right. To do so use **RIGHTARROW** and enter a torque of "-10.0". See Figure 8-5.

Chapter 9. Camera control

In this folowing section, I will show you how to set up a camera which follows the actor, trying to mimic a real cameraman.

Move your mouse over the right 3DWindow (the wire frame view) and zoom out with **PAD-** or **CTRL-MMB** movements. Locate the second camera (the one further away from the player) and select it with **RMB**. With your mouse over the left, textured 3DWindow press **CTRL-PAD0**, this will change the view to the selected camera.

The view is now a bit strange because the camera lays exactly in the ground plane. Move your mouse above the right 3DWindow and press **GKEY** to enter the grab mode. Move your mouse up a bit until the pumpkin is approximately in the middle of the camera-view.



Ensure that the RealtimeButtons are still open (**F8**). Now add a Sensor, Controller and an Actuator as you learned above. Wire the LogicBricks and Change the Actuator into a Camera Actuator. The Camera Actuator will follow the Object in a flexible way which gives smooth motions.

Figure 9-1. Logic Brics for the following camera

Click the “OB:” field in the Camara Actuator and enter the name of the pumpkin object, here “Pump”. The camera will follow this object. Click and hold the “Height: ” field with the **LMB** and move the mouse to the right to increase the value to about 4.0. This is the height the camera will stay at.

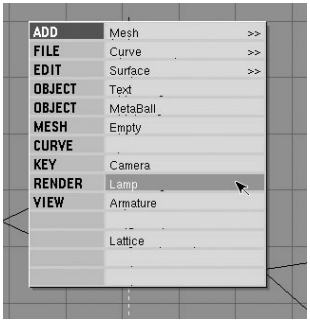
Tip: *Holding CTRL while adjusting a NumberButton will change the value in stages making it easier to adjust the value. SHIFT-LMB on a NumberButton lets you use the keyboard to enter values.*

The Min: and Max: fields determine the minimal and maximum distance the camera will get to the object. I chose “Min: 4.0” and “Max: 6.0”. Start the game engine to test the Camera Actuator. Experiment a bit with the values.

Chapter 10. Real-time Light

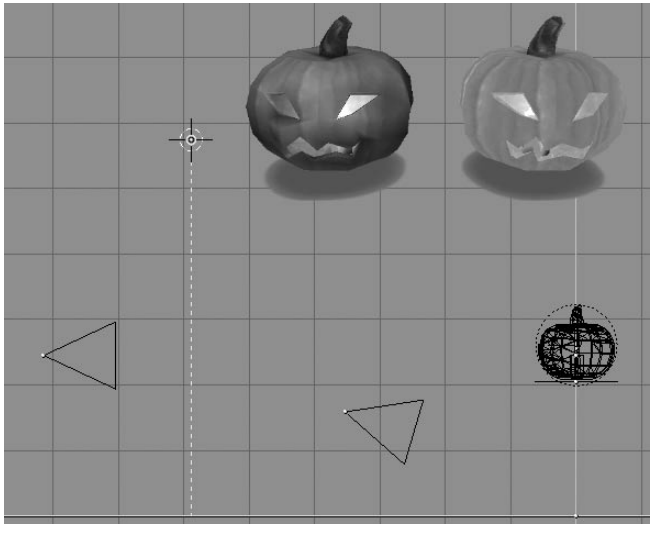
Real-time lighting in Blenders game engine is performed by the *OpenGL* subsystem and takes advantage of hardware accelerated transform and lighting ("T&L") if your graphics card provides it.

Place the 3DCursor  with **LMB** in the right 3DWindow approx. 3 grid units above the cameras. Use the Toolbox (**SPACE**), ADD Lamp.



Watch the effect on the pumpkin in the left textured view, while adding the lamp. For reference, the left pumpkin in Figure 10-1 is lit, the right one is not. Try to move the light around in the 3DWindows (make sure that the light is selected (pink, use **RMB** to select) and press **GKEY**), so you can see that the textured view gets updated in real-time. Moving the light under the pumpkin gives a scary look, for example.

Figure 10-1. Adding a light to the scene



Info: *The real-time lighting in Blender doesn't cast shadows. The shadow of the pumpkin is created differently. Also, bear in mind, that real-time lights cause a slowdown in your games. So try to keep the number of objects with real-time light as low as possible.*

More options for lamps and real-time lighting are covered in Section 26.7.

Chapter 11. Object Animation

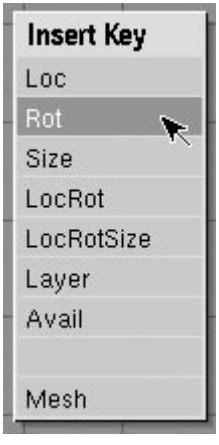
Here I will cover the basics of combining Blenders animation system with the game engine. The animation curves (Ipos) in Blender are fully integrated and give you full control of animations both in conventional (linear) animation and in the interactive 3-D graphics covered by this book.

Use **SHIFT-F1** or use the FileMenu "Append". Browse to book-CD, choose **`Tutorials/Pumpkinrun/Door.blend`**, click on Object, select all objects with **AKEY**, confirm with **ENTER**. This will append a wall with a wooden door to the scene. The pumpkin will bump against the walls and the door. The collision detection is handled by the Blender game engine automatically.

Switch the right 3DWindow to a TopView (**PAD7**) and zoom (**PAD+ or PAD-**) as needed to see the appended door completely. The door has the name and the axis enabled, so it should be visible. Select the door with **RMB** (it will turn pink).

We will now make a simple key frame animation:

1. Ensure that the FrameSlider (the current animation frame) is at frame 1 by pressing **SHIFT-LEFTARROW**



2. **IKEY**, select "Rot" from the menu

3. Now advance the animation time by pressing **CURSORUP** five times to frame 51. With the game engine playing 50 frames per second our animation will play now 2 seconds.

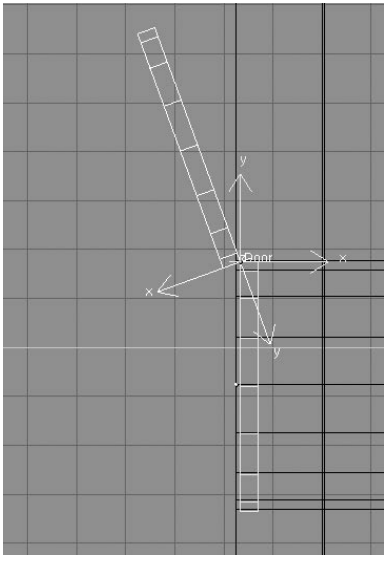
4. Press **RKEY** (be sure to have your mouse over the TopView) and rotate the door 150° clockwise. You can see the degree of rotation in the Header of the 3DWindow. To make it easier to rotate exactly, hold **CTRL** while rotating.

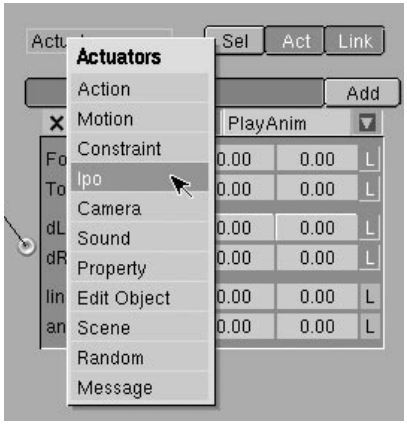
5. Now insert a second key by pressing **IKEY** and again choosing "Rot"

6. Move to frame 1 by pressing **SHIFT-LEFTARROW** and press **SHIFT-ALT-A**, you will see the animation of the door being played back. After 51 frames the animation will run to frame 250 and then repeat.

7. Press **ESC** to stop the playing animation.

Figure 11-1. Rotating the door

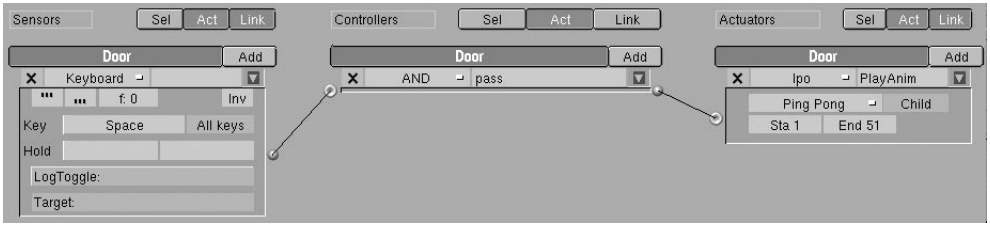




With the door still selected switch the ButtonsWindow to the RealtimeButtons, by pressing **F8**. Add a Sensor, Controller and Actuator, wire them, name them (see Chapter 8), change the Sensor to a Keyboard Sensor and change the Actuator to "Ipo" type.

Change the type of the Ipo Actuator to "Ping Pong" mode using the MenuButton. **SHIFT-LMB** "Sta" and change the value to "1", then change "End" to "51" (see Figure 11-2). This way the Ipo Actuator plays the door animation from frame 1 to 51 which opens the door. A new invocation of the Ipo Actuator will then close the door (because of playing it "Ping Pong").

Figure 11-2. LogicBricks for playing an Ipo in "Ping Pong" mode

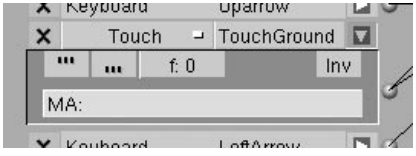


Play the scene (**PKEY**) in the textured view, and the door will now open and close when you press **SPACE** and can push the actor around if he gets hit by the door. To visualize the animation curves (Ipos) switch one window to an IpoWindow by pressing **SHIFT-F6**, see Section 24.2.

Chapter 12. Refining the scene

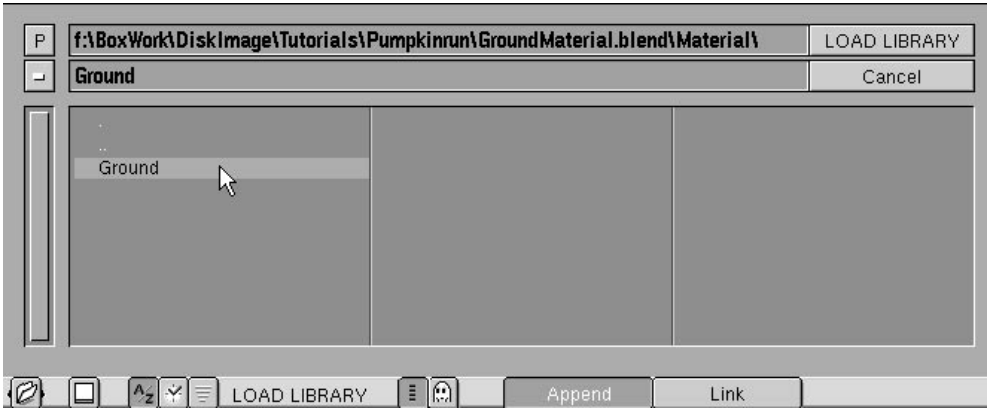
You may have noticed there is currently a problem in the file: The actor can climb the wall because we can jump on every touch of any object even on a wall.

Select the pumpkin and look at the Touch Sensor, the "MA:" field is empty. "MA: " stands for a material name. If you fill in a material name here, the Touch Sensor will only react to objects with this material. In our scene this would be the ground we created at the beginning. But what is the material name? In fact we have not defined a material. To make things easier we will once again make use of the ability to append ready-made elements from a different Scene (see Chapter 6).



Press **SHIFT-F1** whilst with your mouse over one of the 3DWindows, a FileWindow will appear in "Append" mode, which allows us to load any Blender object into an open scene. Go to your CD-ROM drive and browse for the folder "`Tutorials/Pumpkinrun/`" containing the file "`GroundMaterial.blend`". Now click with the middle mouse button (MMB) on the filename "GroundMaterial.blend". The file will be opened immediately. Alternatively, you can click with the left mouse button (**LMB**) and then confirm your selection with **ENTER**.

Figure 12-1. Browsing the GroundMaterial



When you have entered the file, the FileWindow will show you all the parts of the scene like in a file-browser. Now click with **LMB** on "Material", you will see the Materials contained in that scene (Fig. Figure 12-1). Select the "Ground" material with **RMB** and click "LOAD LIBRARY".

Back in the 3DWindow, select the ground plane and press **F5** to call the MaterialButtons. Locate the MenuButton in the ButtonsWindow Header, and

click and hold it with the left mouse button. Choose “0 Ground” from the menu. The zero in the name tells us that there were no objects already using the material.

Now, select the pumpkin again, switch back to the RealtimeButtons **F8** and enter “Ground” into the “MA:” field of the Touch Sensor.

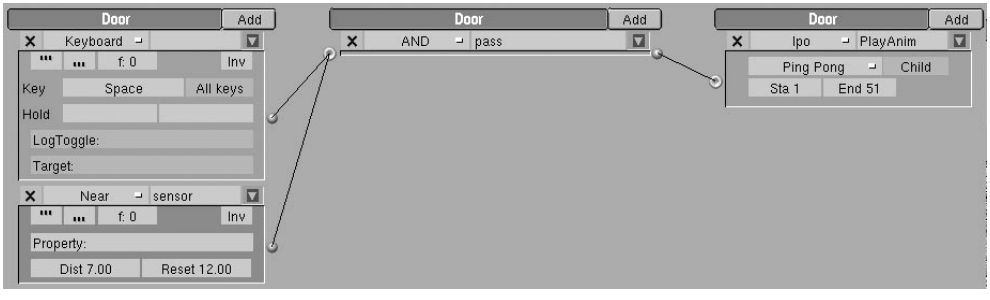
Info: *Whether a name is capitalized or not makes a difference in Blender. So a Material called “ground” is not the same as “Ground”. Blender will blank a button when you enter an object name which does not exist. This may sound frustrating but it really helps while debugging, because it prevents you from overlooking a simple typo.*

Try again to hop around, now you now cannot climb the wall anymore.

One last thing: it would be nice if the door would only open when the actor is close to it. The Near Sensor will help us here.

Add a new Sensor to the door and change the Sensor type to “Near”. Wire it to the existing AND Controller (see Figure 12-2). The “Dist:” field gives the distance at which the Near Sensor starts to react to the actor. By the way, it will react to every actor if we leave the “Property:” field empty. The “Reset: 12.0” field is the distance between the Near Sensor and the object where the Near Sensor “forgets” the actor. Try to make the “Dist:” setting to 4.0, now you need to come close to the door first but then you can back up before you press **SPACE** to open the door without the danger of getting hit. Now the door only opens when you press **SPACE** *and* the pumpkin is near the door.

Figure 12-2. Near Sensor

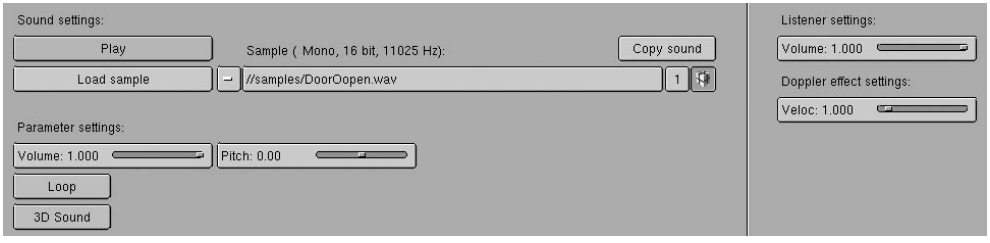


Chapter 13. Adding Sound to our scene

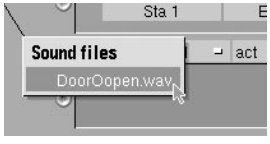
There is no game without sound, so I will show here how to add a sound to an event in Blender.

Locate the SoundButtons icon in the icons of the ButtonsWindow. Click it with the left mouse button to switch to the SoundButtons. Because there is no sound in the scene the window will be empty. Use the MenuButton (click and hold) to choose "OPEN NEW". A FileWindow opens, browse to get to the CD-ROM and load **`DoorOpen.wav`** from the directory **`Tutorials/Pumpkinrun/samples/`**.

Figure 13-1. The SoundButtons with a sound loaded

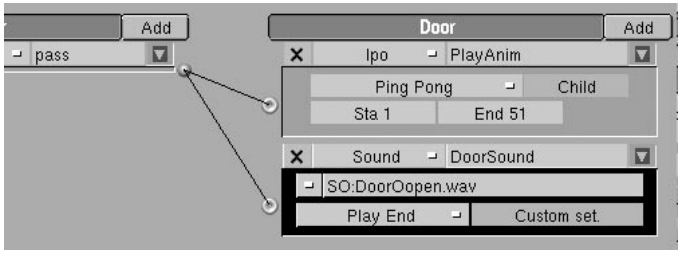


Blender is capable of creating 3-D sound (Sound located spatial in the scene) and provides many ways to influence the sound. But for the moment we can go with the defaults and don't need to touch any of these buttons. Of course you can play the sound by clicking on the big "Play" button.



Select the door object (**RMB**) and switch to the RealtimeButtons with **F8**. Add a new Actuator and change the type to "Sound". Wire it to the Controller (see Figure 13-2). Click and hold the solitary MenuButton in the SoundActuator and choose the sound file "DoorOpen.wav" from the pop-up menu. As a last step before you can try it, change the mode "Play Stop" to "Play End" this will mean the whole sound is played without stopping too early.

Figure 13-2. SoundActuator for the door

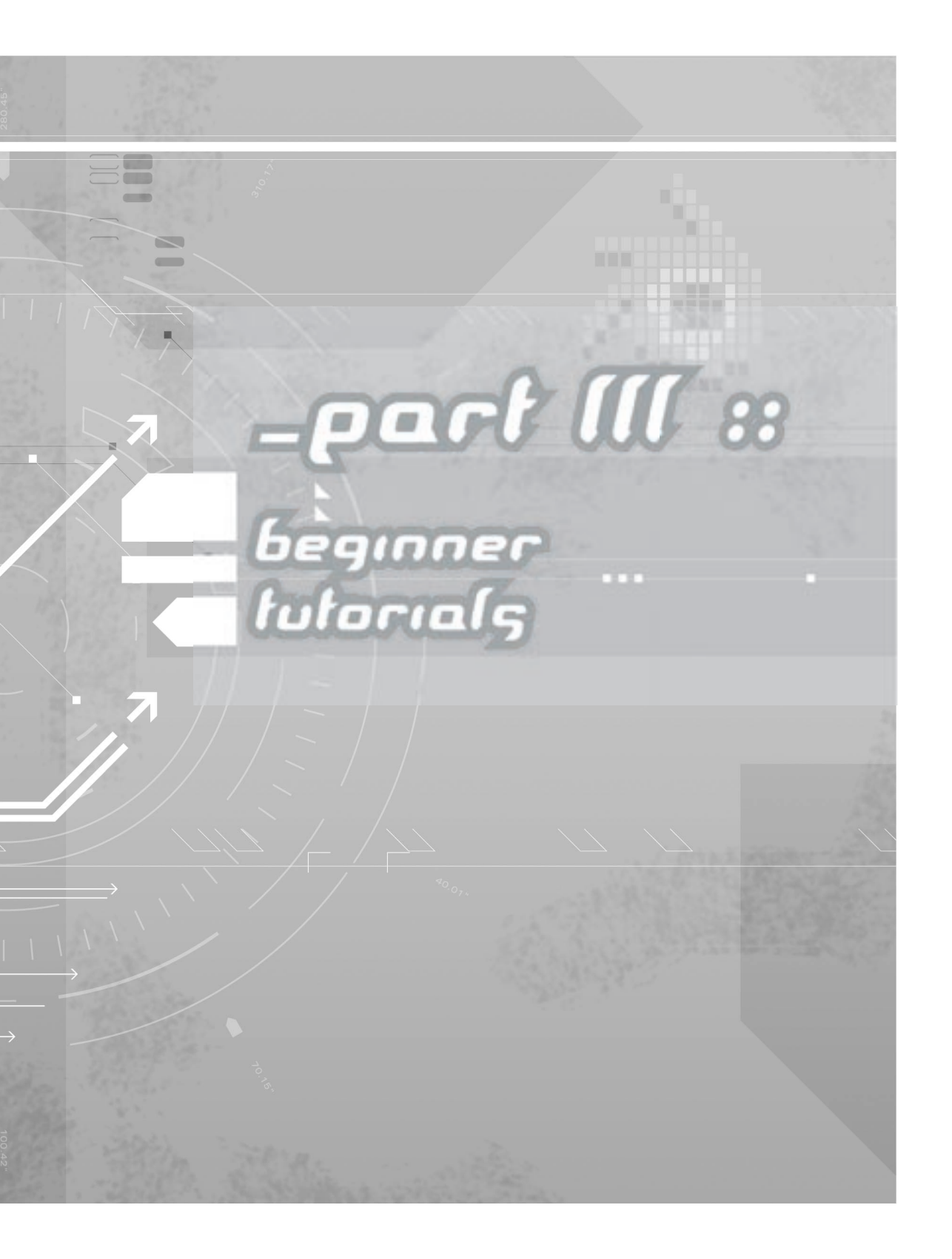


Chapter 14. Last words

After this chapter you should have an idea what it is about to make a game with Blender. We did some of the basic steps and you are now prepared to do other tutorials or start playing with ready made scenes or your own ideas.

I suggest that you continue with the Chapter 4, and read it at least one time so that you know where to look when you face problems. Also don't hesitate to use our support, see Section 29.4.


001
003
2.03beta
004
2.04
game Blender
005
010
2.10
011
2.12
blenderGameKit
blender
POWER
CREATE
A
B
C
D
2.20
022
023
2.23


_part III ::
beginner
tutorials

Beginner tutorials are aimed at beginners in interactive 3D graphics. We tried to make the tutorials as self-contained as possible, but if you experience problems please go to the Chapter 4 or use our support or the Blender Community (Section 29.4).

In the following tutorials, we use the “hands-on” approach to give you quick results, so make sure you follow the words and screenshots closely, and an explanation will follow later. Of course, you are invited to do your own experiments!

Chapter 15. Tube Cleaner, a simple shooting game

Figure 15-1. Tube Cleaner Title

Tube Cleaner was designed by Freid Lachnowicz. The game is a simple shooter game that takes place in a tube. Three kinds of enemies are present. Try to collect as many points and bullets as you can on your way up the tube!

To play the game in its final stage load the scene **`Demos/TubeCleaner.blend`** from the CD. It includes instructions.

This Tutorial is not supposed to explain how to make the full game. But you should be able to understand the extensions in the final result with the help of this book and this tutorial. And of course you are encouraged to change and extend the game to your liking!

Figure 15-2. Tube Cleaner game

Table 15-1. Tube Cleaner game controls

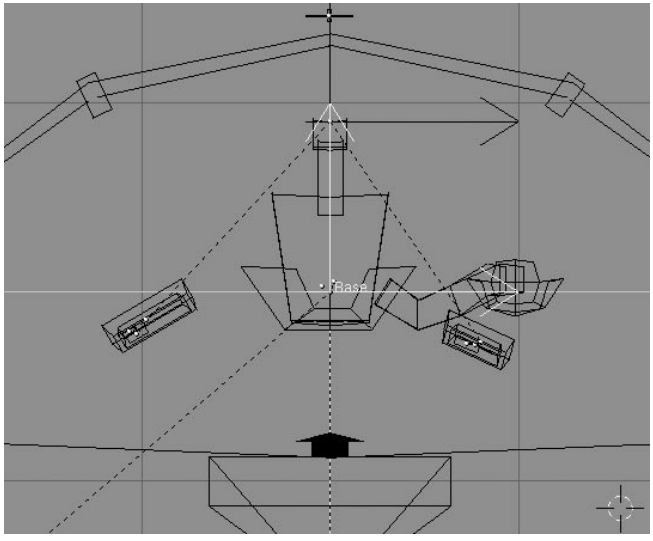
Controls	Description
CURSOR KEYS	rotate the cannon left, right, up and down
SPACE	Shoot

15.1. Loading the models

Start Blender and load `Tutorials/TubeCleaner/TubeCleaner00.blend` from the CD. This scene contains all models to start. To make the scene interactive, this tutorial will lead you through the following tasks:

1. Adding game logic to the gun, allowing it to move up, turn and shoot
2. Adding game logic to the enemies
3. Creating the score system including display
4. Providing "Extra Bullets"

Figure 15-3. Wireframe TopView in the loaded Tube Cleaner scene

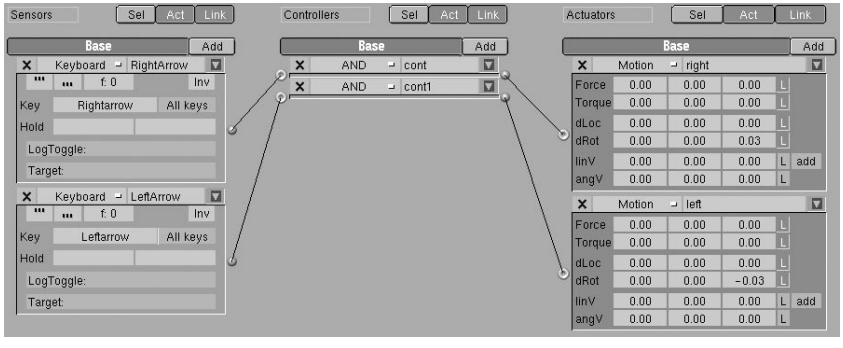


The scene contains a CameraView on the left, a wireframe view (view form top, TopView) on the right and a the RealtimeButtons on the bottom. In the TopView (Figure 15-3) you can see the "Base" object is already selected and active (purple color in wireframe). The "Base"-object will carry the cannon and will contain some of the global logic of the game. The cannon itself is *parented* to this "Base". This hierarchy will make our job later easier because we won't have to worry about composite movements.

15.2. Controls for the base and cannon

We start with the rotation around the vertical axis of the base. This will also rotate the cannon and the camera because they are parented to the base.

Figure 15-4. LogicBricks to rotate the gun



Make sure that the "Base" object is *selected* (purple, **RMB** to select if not) and click on the "Add" Buttons in the RealtimeButtons for each row of Sensors, Controllers and Actuators. In every row a new *LogicBrick* will appear.

Now link (wire) the LogicBricks by clicking and drawing a line from the little yellow balls (outputs) to the yellow donuts (input) of the LogicBricks. These connections will pass the information between the LogicBricks. Change the first LogicBrick to a Keyboard Sensor by click and hold its MenuButton with the left mouse button and select "Keyboard" from the pop up menu.

Info: *Please do the tutorial in Part II in Game Creation Kit if you have problems with the creating, changing and linking of LogicBricks.*

Now, click the "Key" field with the **LMB** and press the **RIGHTARROW** key when prompted by "Press any key" in the Keyboard Sensor. The "Key" field now displays "Rightarrow" and the Keyboard Sensor now reacts to this key only.

Now change the third number in the "dRot" row of the Motion Actuator to 0.03. Do this by using **SHIFT-LMB** on the number and entering the value with the keyboard. The three fields always denote the three axis (X,Y,Z) of an object. So we will rotate around the Z-axis.

Now move your mouse cursor over the CameraView and press **PKEY** to start the game. You should now be able to rotate the gun with the right cursorkey.

Info: *You should always name your LogicBricks and other newly created elements in your scenes (click on the default name and edit with your keyboard). This will help you to find and understand the logic later. Take Figure 15-4 as a reference.*

Use the same procedure as above to add LogicBricks to rotate the gun to the left. Use **LEFTARROW** as key in the Keyboard Sensor and enter "-0.03" in the third "dLoc" field of the Motion Actuator.

As you can see the space in the RealtimeButtons is getting sparse with only six LogicBricks. Use the Icon in the LogicBricks to collapse the LogicBricks to just a title. Now you also see another good reason to properly name LogicBricks.

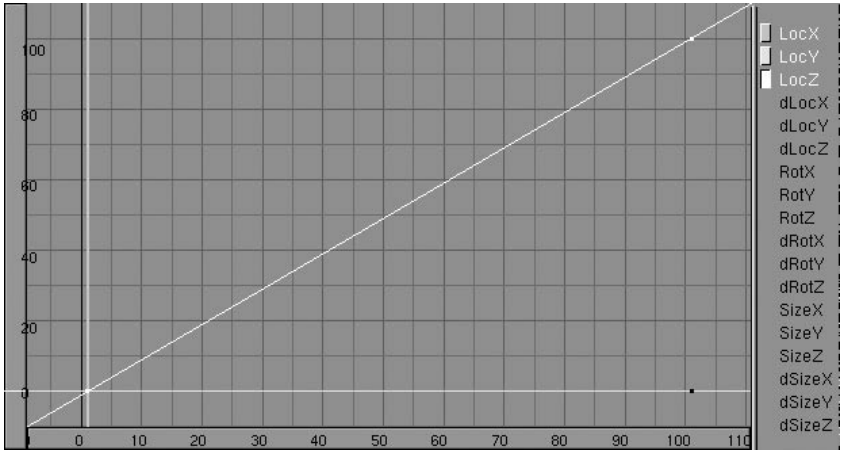
15.2.1. Upwards Movement

In „Tube Cleaner" we want to have a continuous upwards movement within the tube. We could achieve this similarly to the rotation of the gun, but there is another possibility which will give us much more control over the movement and also allows you to move down to a specific level of the tube.

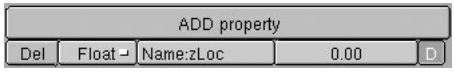
The method used here is to combine the possibilities of Blender's game engine and its powerful animation system. Move your mouse cursor over the CameraView and press **ALT-A**, Blender will play back all the animations defined in the scene. Press **ESC** to stop the playback. But so far none of these animations is played back by the game engine. We have to tell the objects to play the animation. This way we can interactively control animations, for example play, stop or suspend.

Move your mouse cursor over the wireframe view and press **SHIFT-F6**. The window will change to an IpoWindow (see Figure 15-5), meant for displaying and editing Blender's animation curves. The IpoWindow is organized into axes, here the axes are the horizontal axis, showing the time in Blender's animation frames and the vertical axis showing Blender units. The yellow vertical line is the animation curve for the movement along the Z-Axis of the "Base" object, meaning upward movement for our object. So to move the object 10 units up you can move the Ipo cursor (green line) with a left mouse click to frame 10. The CameraView will reflect this immediately.

Figure 15-5. IpoWindow with the animationcurve for upward movement



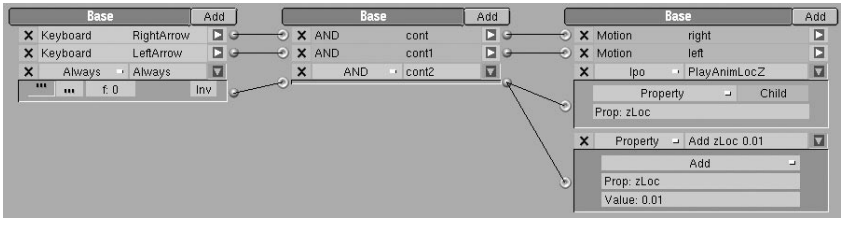
To play this animation in the game engine, we use a special LogicBrick the "Ipo Actuator", set to "Property" type. A Property is a variable owned by a game object, which can store values or strings. We will now create a new property which will hold the height (zLoc(ation)) of the "Base" object. To do so click on "ADD property" in the RealtimeButtons for the "Base" object.



Click on "Name:" and change the default name "prop" to "zLoc", this Property will hold the height of the gun in the Tube.

Info: *As touched on earlier, Blender uses capitalization to distinguish between names of Objects and Properties. So "zloc" is not the same as "zLoc".*

Figure 15-6. LogicBricks for the upward movement



Continue creating the LogicBricks from Figure 15-6. The Always Sensor triggers the logic for every frame of the game engine animation, ensuring constant movement. The AND Controller just passes the pulses to two Actuators, both are connected to the Controller and will be activated at the same time.

The Ipo Actuator will play the Ipo animation according to the value in the Property "zLoc". To get a constant motion we increase the "zLoc" Property every frame with the Property Actuator . Here it is from the type "Add" which adds "Value:" (here 0.01) to the "zLoc" Property. Try to change the "Value:" to get a feeling for the speed of the animation. If you'd like to move the cannon downwards try entering -0.01 into the "Value:" field. After experimenting a bit please use 0.01 for the value field as shown in Figure 15-6. To play the game so far move your mouse to the CameraView and press **PKEY**.

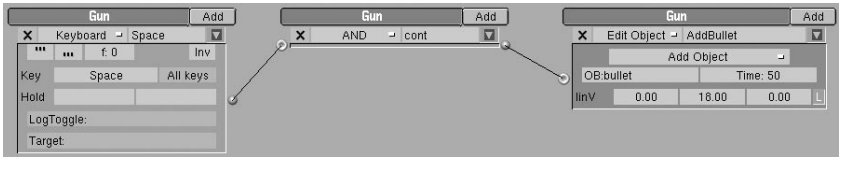
Info: *Blender can show you the used Properties and their values while the game runs. To do so, choose "Show debug properties" from the "Game" menu and activate the "D" Buttons (Debug) for every Property you'd like to have printed on screen*

15.3. Shooting

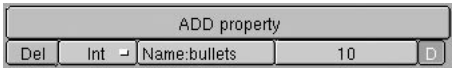
Switch the IpoWindow back to a 3DWindow by pressing **SHIFT-F5** over the IpoWindow. Now select the "Gun" object with the right mouse. You can click on every wire from the "Gun" object, a proper selection will be reflected in the ButtonsWindow Header ("OB:Gun") and in the RealTime Buttons where "Gun" will appear in the columns for the LogicBricks.

Now add a Sensor, Controller and Actuator to the "Gun" object and wire them as you learned earlier in this tutorial. Change the Sensor to a Keyboard Sensor (please name the Sensor "Space") and choose (click the "Key" field) **Space** as trigger for the gun. Change the Actuator to an "Edit Object" Actuator. The default type is "Add Object" which adds an object dynamically when the Actuator is triggered.

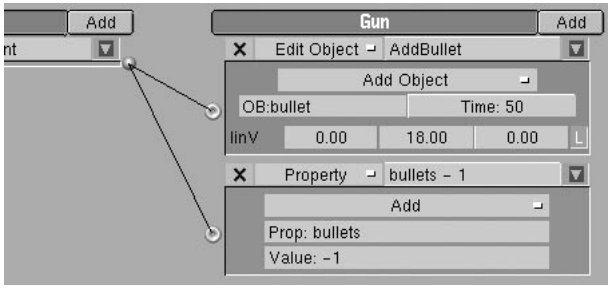
Figure 15-7. LogicBricks to fire the gun



Enter "bullet" into the "OB:" field by clicking it with **LMB** and using the keyboard to enter the name. This will add the object "bullet", a pre-made object, which is on a hidden layer in the scene. Enter 18.00 as the second number in the "linV" fields. This will give the bullet an initial speed. Also activate (press) the little "L" Button behind the "linV" row. This way the bullets will always leave the gun in the direction aimed. Enter 50 in the "Time:" field this will limit the lifetime of the bullets to 50 frames, avoiding ricochets to bounce forever. As usual, now try to run the game now and shoot a bit.

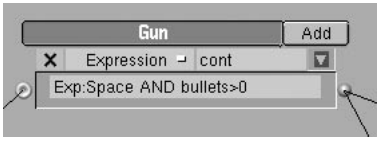


So far we have unlimited ammunition. To change this we again use a Property storing the number of bullets left. Add a new Property by clicking on "ADD property". Name this Property "bullets" and change its type to "Int" with the MenuButton now labeled "Float" (the standard type for new Properties). An "Int(eger)" Property only holds whole numbers, this is ideal for our bullets as we don't want half bullets. **SHIFT-LMB** on the field to the right of the "Name:" field to enter 10 here. This is the initial number of bullets available at the start of the game.



To actually decrease the number of bullets on every shot we use the "Property Add" Actuator that we also used to make the base of the cannon move up. So add another Actuator by clicking on "Add" in the Actuator's column of the gun. Wire it to the AND Controller we created in the last step. Change the Actuator type to "Property" choose "Add" as the action. Enter "bullets" in the "Prop:" field and -1 in the "Value: " field. This will subtract 1 (or add -1) from the Property "bullets" on every shot triggered by **Space**.

So far the gun doesn't take any notice of the number of bullets. To change this we will use an Expression Controller which allows to add single line expressions to process game logic. Change the AND Controller with the MenuButton to an Expression Controller. Then click then on the "Exp:" field and enter "Space AND bullets>0" (without the quotation marks) and press **ENTER**. Here "Space" is the name (exactly as you typed it in the LogicBrick) of the Keyboard Sensor and "bullets" is the bullets Property. The Controller now only activates the following Actuators if the Sensor "Space" is active (meaning that **Space** is pressed) AND bullets is bigger than zero. Try again to run the game, you now can only shoot 10 times. Read more about Expressions in Section 26.9.



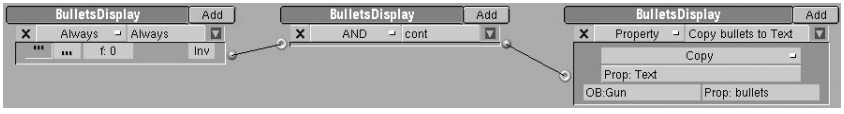
The last step to make the gun work is to make the bullets display functional. Select the display (the right one with the flash on it) with the right mouse button. It is best to hit the little dot on the "1". The name "BulletsDisplay" should appear in the RealtimeButtons and in its header as "OB:BulletsDisplay". Alternatively you can zoom into the wireframe view to make the selection easier (see Section 4.10).

You can see in the RealtimeButtons that there is already a Property called "Text" for the display object. The object has a special text-texture assigned which will display any information in the Property called "Text" that is on it. You can test this by changing the value "10" in the Property, the change will be displayed immediately in the CameraView.

Because Properties are local to the object they are owned by, we have to find a way to pass the value of Properties between objects. This is done inside a scene (but will not work across scene borders) with the Property Copy Actuator.

Add a line of LogicBricks to the "BulletsDisplay" like you did before and wire them. Change the Actuator to a Property Actuator, type "Copy". See Figure 15-8.

Figure 15-8. LogicBricks to display the amount of bullets



Enter "Text" into the first "Prop:" field, this is the name of the Property we copy into. Enter "Gun" into the "OB:" field, again watch for correct capitalization, Blender will blank the input field if you enter a non-existing object. From this object we will get the value. Enter "bullets" into field "Prop:" beneath "OB:" this is the Property name from which we get the value.

Start the game and shoot until you have no more bullets.

15.4. More control for the gun

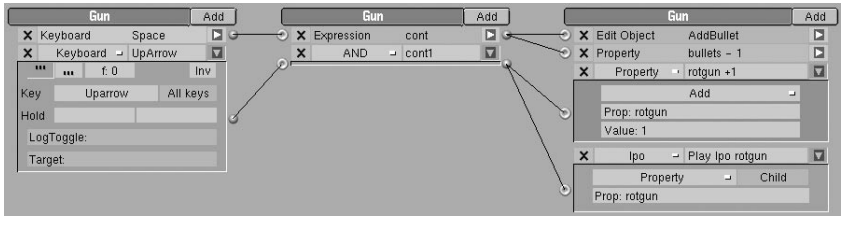
Tilting the gun will add more freedom in movement and dynamic to the game. We will use a similar technique as for the movement up, by combining animation curves and LogicBricks.

Select the gun again, and collapse the LogicBricks by clicking their ▽ arrow icons, this will give us more space for the coming logic.

As for the upward movement the gun, it already contains a motion curve that we can use. We also need a Property which contains the actual rotation (tilt, rotation around x-axis). So add a new Property by clicking on "ADD property" and name it "rotgun".

Again use the "Add" Buttons to add a Sensor, Controller and one... nope, you are right this time two Actuators. You already know this from the upward movement. We need one Actuator to change the Property and one to play the Ipo. Wire the new LogicBricks, as shown in Figure 15-9. The collapsed LogicBricks are the ones you made for shooting.

Figure 15-9. LogicBricks to rotate the gun upwards



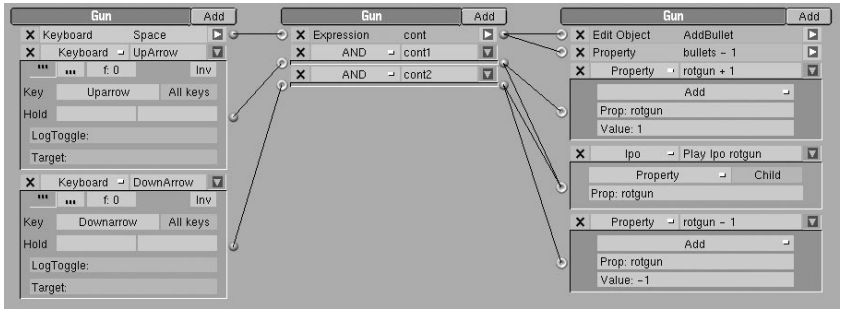
Change the Bricks according to Figure 15-9 and enter all the necessary information. Using the Property Add Actuator we increase "rotgun" by one every time **UpArrow**

is pressed and we play the Ipo according to the "rotgun" Property, this will rotate the gun up.

Note that I activated the pulse mode icon for the Keyboard Sensor, this will give a keyboard repeat here, so the gun will rotate as long as you press the key without the need to release it.

Now test the rotation, and you will see that the gun only rotates a specific amount and then stops. This is done with the animation curve (Ipo), you can visualize the curve when you switch a Window to an IpoWindow with **SHIFT-F6** (use **SHIFT-F5** to return to the 3DWindow). You can see that the curves go horizontally from frame 21 (horizontal axis), meaning no further rotation is possible. You also see that we need to make the "rotgun" negative to rotate down.

Figure 15-10. Completed LogicBricks to tilt the gun

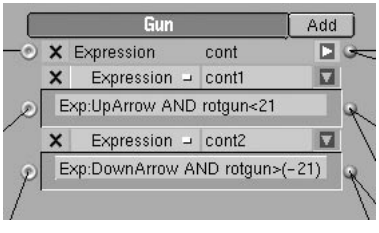


Again add a Sensor, Controller and one (yes, this time, it's really only one) Actuator. Wire and name them as shown in Figure 15-10. Note that we use the Ipo Actuator for the tilting down also. This is perfectly ok, we can save this way a LogicBrick. It would have also been ok to use a second Ipo Actuator here.

Info: *If you prefer "pilot-controls" just swap the UpArrow and DownArrow in the Keyboard Sensors.*

There is one drawback: if you press **UpArrow** for too long the gun will stop rotating but "rotgun" is still incremented. This will cause that the cannon is not rotating back immediately when you press **DownArrow** again. To prevent this we can use Expressions again, see Figure 15-11 for the correct Expressions.

Figure 15-11. Expressions to correctly stop the rotation



These Expressions will stop changing "rotgun" when "rotgun" is already greater than 21 or less than -21.

Info: *It is time to save your project now! Blender scenes are usually very compact so saving only takes seconds. First you need to pack (include in the Blenderfile) the textures, use the Tools menu "Pack Data" to do so. This way you can also send the file to a friend by e-mail. Then use the FileMenu or save with the keyboard command F2. See Section 4.3.*

15.5. An enemy to shoot at

It is now time to add something to shoot at. Select the "Target" object with the right mouse.

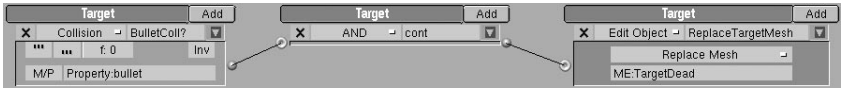
The tasks a game logic on the enemy has to do are:

1. Reaction to hits (collisions) with the bullets
2. Make a silly face when hit, and die
3. Add some points to the players score

You can see that I try to keep the game logic on the target itself. Although it is possible to put all that to the player or any other central element, it would make it very complex and difficult to maintain the logic on this object. Another advantage of using local game logic is that it makes it much easier to re-use the objects and the logic, even in other scenes.

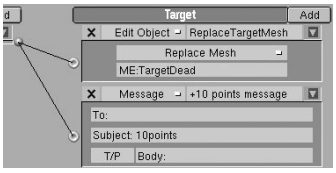
We start again with by adding a Sensor, Controller and an Actuator and wiring them. You should be familiar with that procedure by now. To react to a collision, change the Sensor to a Collision Sensor. Enter "bullet" into the "Property:" field, this is the name of a Property carried by the bullet, this way the Collision Sensor will only react to collisions with bullets.

Figure 15-12. LogicBricks to make the target look silly



Change the Actuator to Edit Object and choose "Replace Mesh" as the type. Enter "TargetDead" into the "ME:" field. This mesh show the dead target and will be shown from now on when you hit the target. The dead target is on a hidden Layer, you can look at it when you switch Layer 11 on and off by pressing **SHIFT-ALT-1KEY** (see Section 24.1).

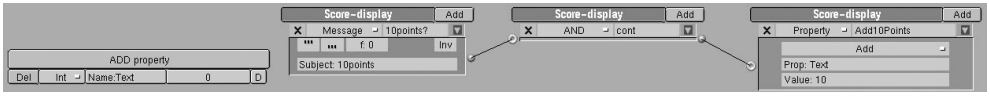
To score our hit we will Blender's messaging system. This allows us to send messages from objects to objects. In this case we will signal to the score-display to add some points to our score. So add a second Actuator to the target, wire it with the existing Controller and change it to a Message Actuator.



We can leave the "To:" and "Body:" fields blank, just fill in the "Subject:" field with "10points". This is equal to shouting into the room an the score-keeper will note the score.

We now need to set up the score-display to react to the score messages. Select the "Score-display" object and add LogicBricks as shown in Figure 15-13.

Figure 15-13. LogicBricks to count the score

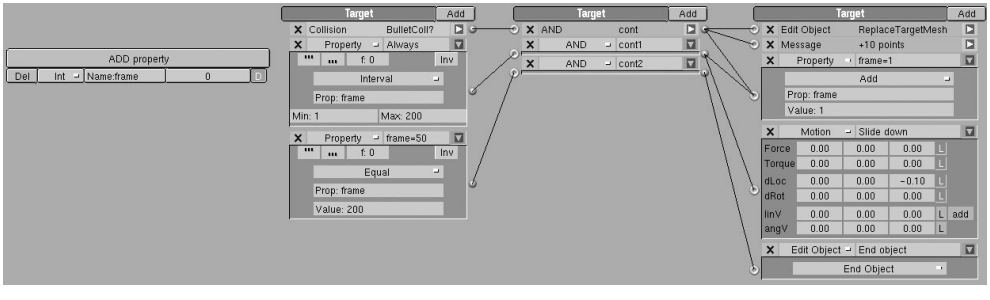


The "Score-display" is again an object with a special texture on it showing the content of the "Text" property as explained for the bullets-display. Be sure to change the 999 to zero or the score will start with 999 points. The Message Sensor will only hear messages with the "10points" subject and then trigger the Property Actuator to add 10 to the "Text" Property which is then displayed. This way it is also very easy to add different scores to different actions, just add a new line of LogicBricks listening for different subjects and then add the appropriate amount of points.

You should now try out the game so far and shoot at the enemy. This should add 10 points to your score. If anything fails to work, check especially the correct wiring, and that the names and capitalization of Properties and message-subjects are as they should be.

In the final game the targets start to slide down the tube, look at Figure 15-14 for a possible solution for that. The simple target also has the drawback that it will still add a score when you hit a dead target.

Figure 15-14. Advanced animation for dead targets



We have already used most of these LogicBricks. Together with the reference (see Chapter 27) and the final game on the CD you can now try to extend the file or just enjoy playing the game. Don't desperate and keep on experimenting. By breaking the task into small steps, even complex logic is possible without getting lost.

Chapter 16. Low poly modeling

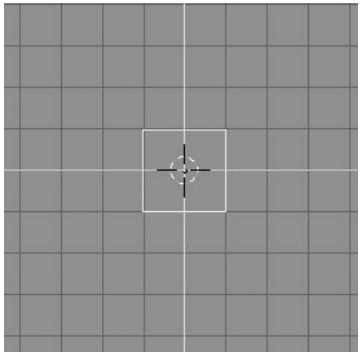
by W.P. van Overbruggen

In this tutorial, we're going to model a low polygon car, a 50's style car to be exact. Since we're aiming for a real-time model which can be used in games we'll set a polygon limit of 1000 triangles for the entire car. Even though most recent console and PC racings game have cars of up to 4000 triangles the 1000 triangle limit should still give us enough space to add nice details and still keep it acceptable on almost any recent computer with a 3-D graphics card.

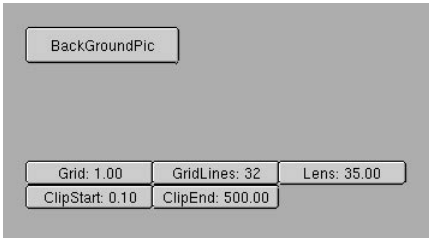
Figure 16-1. Racer game

You can load and play a complete game with the same style car from the CD: `Demos/55wheels.blend`

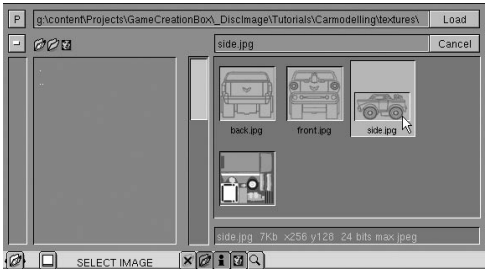
16.1. Loading an image for reference



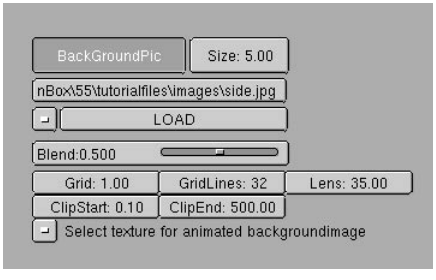
To make the whole process easier we will be using an image displayed in the back-buffer of blender as a easy guide. So first up we will load the image into the image back-buffer. This is done by pressing **SHIFT-F7** in the 3DWindow which takes us to the back-buffer window.



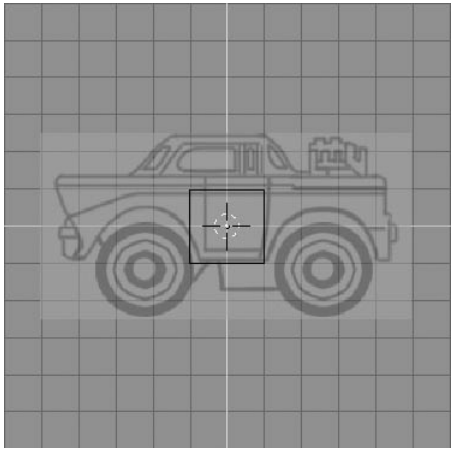
To load the guide image hit the big "BackGroundPic" button this will open up a new set of buttons including the "Load" button. Press it.



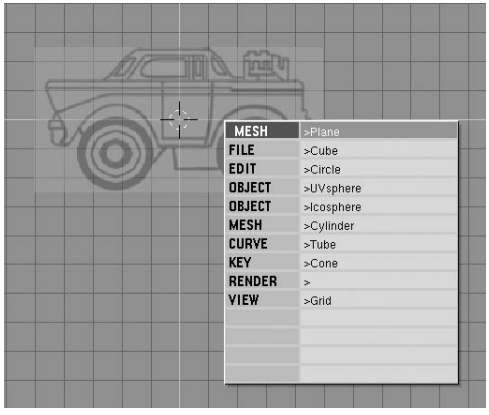
This is the image window, normally used for loading textures but we will now use it for selecting the image which we will be using for reference. Go to the folder **`Tutorials/Carmodelling/`** which contains the 3 images called **`front.jpg`**, **`back.jpg`** and **`side.jpg`**. Load the **`side.jpg`** by clicking the left mouse button on the image and hitting **ENTER** to confirm.



Once the image is loaded we will head back to the 3DWindow by hitting **SHIFT-F5**.

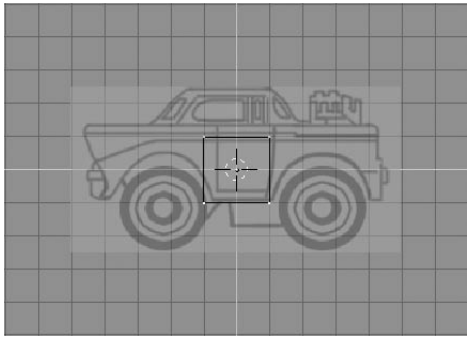


16.2. Using the reference image.

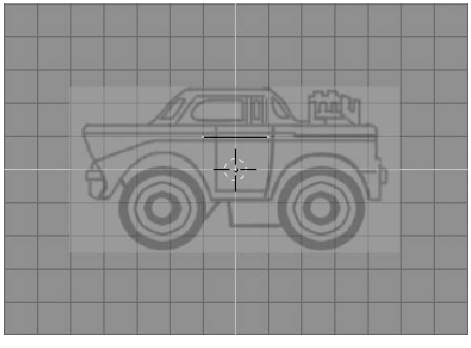


Alright -- let's get started with modeling this car. Make sure you are in FrontView by hitting **PAD1**. Select the default plane with the right mouse button, then hit **XKEY** to delete it. Next up we will add a new plane by hitting **SPACE** and selecting "Mesh->Plane" from the menu.

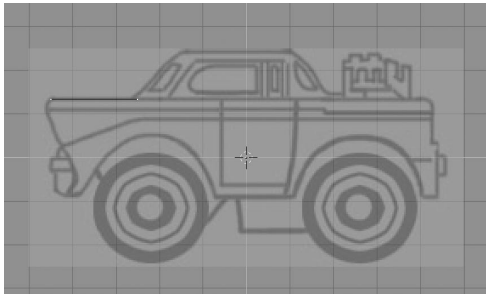
III



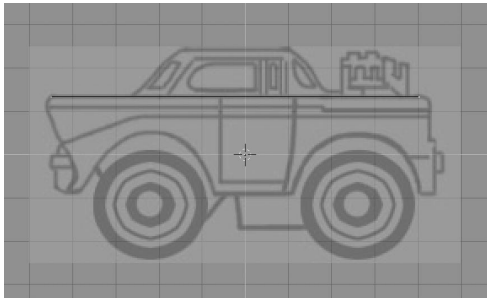
Now that we have a plane in FrontView we will first delete the bottom two vertices which can easily be done by selecting both bottom vertices.



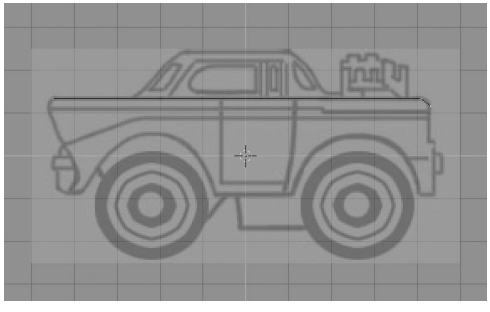
They can either be selected by holding **SHIFT** and selecting them with right mouse button or by hitting **BKEY** for BorderSelect and while holding left mouse dragging a border around the two bottom vertices. Once selected, delete them by pressing **XKEY**.



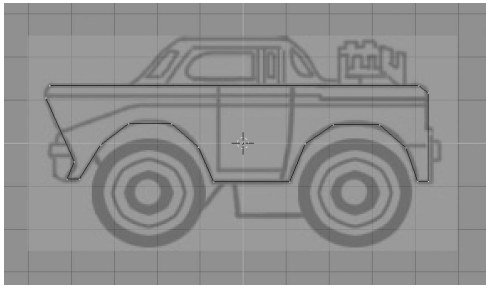
Now select the other 2 vertices in the same manner and line them up with the back of the body of the car by pressing **GKEY** and moving the vertices. Confirm the new position by clicking **LMB**.



Select the right vertex with **RMB** and move (**GKEY**) it to the front of the car.



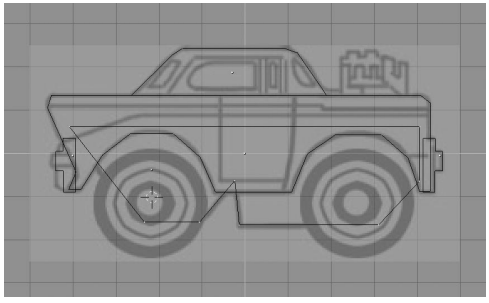
Next up we will start adding additional vertices to complete the side view of the car. Make sure you have the right vertex selected and hit **EKEY** to extrude the vertex into a new vertex. Hit **ENTER** to confirm the pop up request. To create a smooth front we will move the new vertex a little bit to the right and little bit lower. Press **LMB** when you are satisfied with the position.



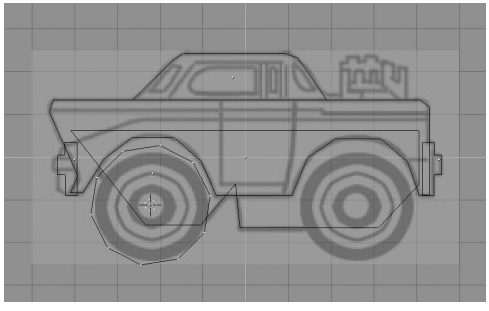
Now with this method we will trace the entire side of the body of the car, try to trace it with as few vertices as possible. To connect the last two vertices to each other, select them both (hold **SHIFT** to select the second one with **RMB**), then press **FKEY** to connect them. When you are finished leave EditMode (**TAB**) to complete the outline.

In the same way trace the other parts of the car, like the top and the bumpers. Use the image above as a reference.

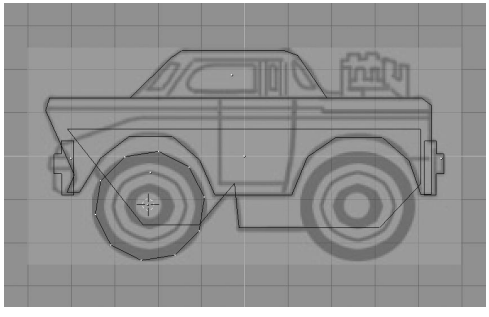
16.3. Outlining the Wheels



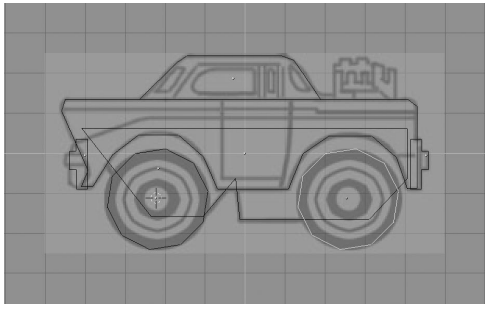
First place the 3DCursor  in the middle of the wheel, on the reference image by pressing the left mouse button over the wheel's center. New objects are always added on the exact position of the 3DCursor. You can always zoom the view by holding **CTRL-MMB** and moving the mouse up and down. Panning the view is done by holding **SHIFT-MMB** and moving the mouse.



Hit SPACE and choose "Mesh>Circle" from the top entry of the ToolBox, a menu will pop up stating how many vertices you want the circle to consist of, set this to 10 by holding **SHIFT**, pressing left mouse at the value and filling in 10. To confirm it click on the "OK" button.



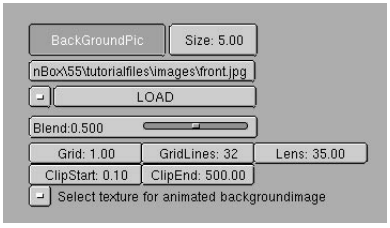
Now the circle might not be the same size as the wheel is, so we will fix this by pressing **SKEY** (for scale) and moving the mouse closer to the center of the circle.



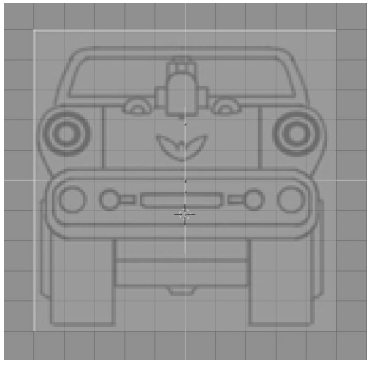
When scaling the wheel to the approximate size of the image, you can hold down **CTRL** to scale down in steps of 1.0 to scale, you can also scale in even smaller steps by holding **SHIFT**. If you are pleased with the result press **LMB** and leave EditMode (**TAB**) afterwards.

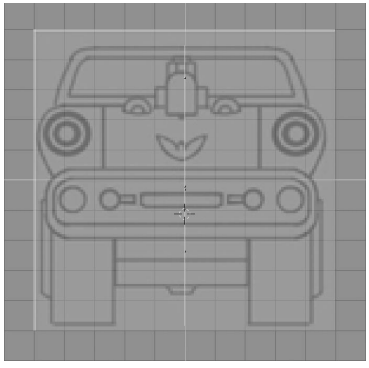
The last thing we will do in SideView for now is select the wheel and hit **SHIFT-D** to duplicate it, once this is done move it into the position of the front wheel and press the left mouse button to confirm the new location.

16.4. Loading the front image

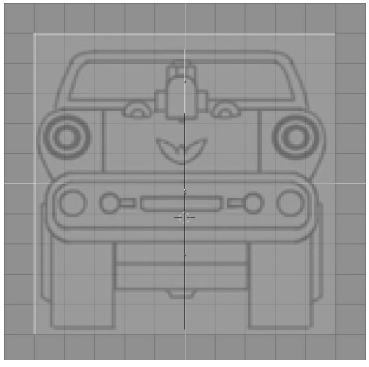


By using the things we have learned earlier in this tutorial we will load the `front.jpg` of the car into the backbuffer which we will use to create the front of the car. Press **SHIFT-F7** over the 3DWindow to do so.

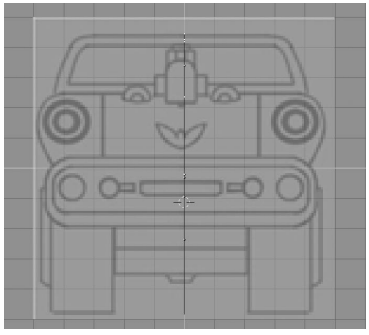




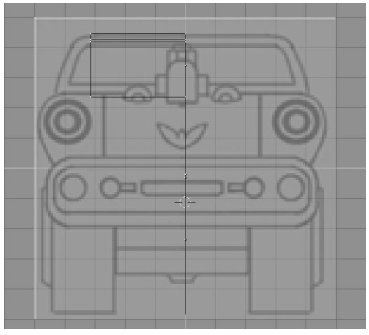
First we'll go into SideView by pressing **PAD3**. Now as you might notice the scale of the front image and the model we traced isn't the same size here. Lets quickly fix this by selecting all of the objects by hitting **BKEY** and draw a border around them. Then hit **SKEY** to scale it to match the front image. Press **LMB** to confirm the new scale.



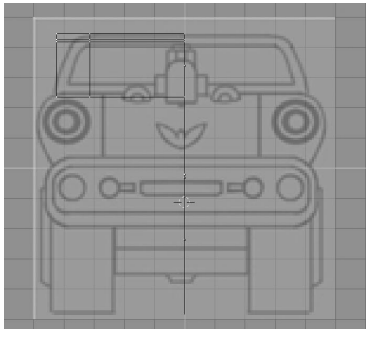
Once this is done, we will select the top of the car with using the right mouse button.



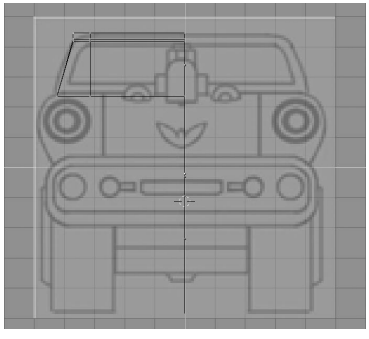
Press **TAB** to go into EditMode and select all the vertices with **AKEY**,



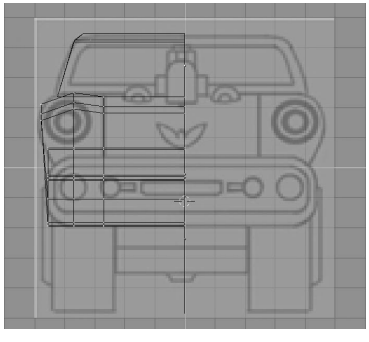
...then press **EKEY** to extrude the selected vertices alongside the image, position them just before the top starts to curve as seen in the screenshot. Press **LMB** to confirm the extrude.



We will now extrude it a second time to create a smoother top.



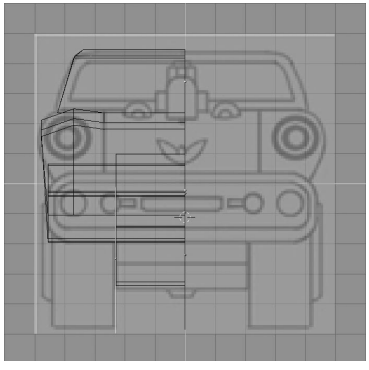
Once the third row of vertices is in place, select the top vertices and move them to the right to follow the shape a bit more closely.



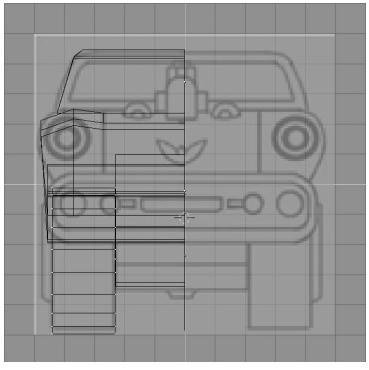
Following this principle, we're going to extrude all the other parts in the same manner.

From time to time, make sure you rotate the view by holding **MMB** and moving the mouse. Also switch between FrontView and SideView to make sure the vertices are placed in the right places.

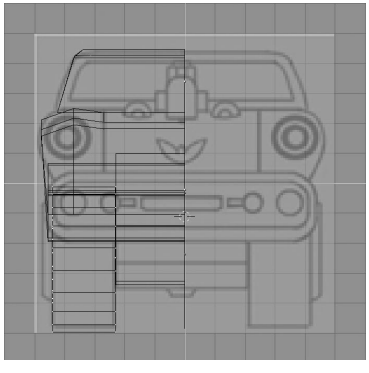
As you can see, we're only building one side of the car, this saves time since a car usually symmetrical and we can easily duplicate this side later on, flip it and re-attach it to the original side of the car to create a whole model.



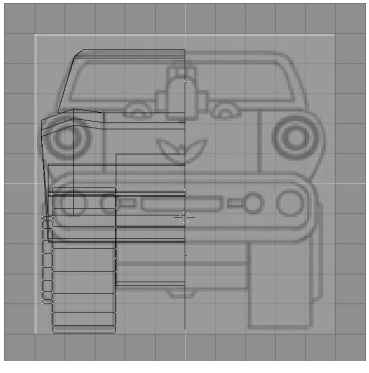
Now, when you select the wheels, make sure you first place them correctly in the FrontView in relation to the image, since we originally traced them in FrontView, Blender positioned them along the same line as the other parts of the car.



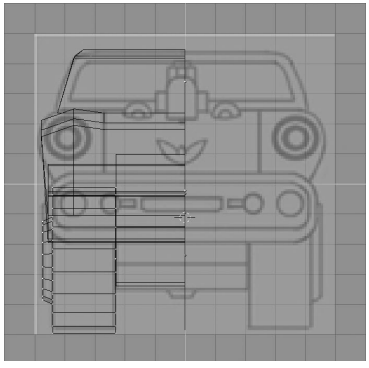
Then extrude them once, but don't leave the EditMode yet.



To create the hubcap shape hit **EKEY** once more to extrude them but don't move them this time. Instead press **SKEY** to scale them inwards to the size of the hubcaps. Confirm with **LMB**.



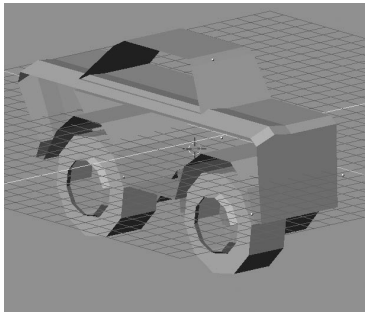
Hit **EKEY** once more and move the extruded vertices to the left to create the hubcap. Once positioned confirm once again with **LMB**.



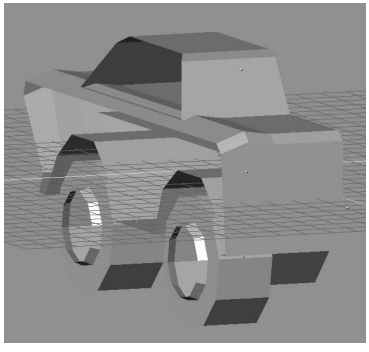
Hit **SKEY** one more time and scale them a little bit smaller to make sure they are not at a 90 degree angle.

III

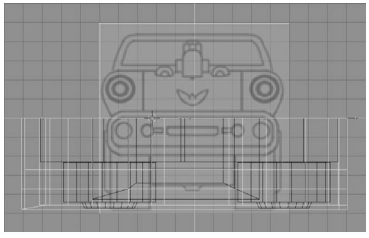
16.5. A quick break



Alright, lets have a quick look at how it looks in 3-D. This is easily done by pressing the middle mouse and moving the mouse to go into a 3-D view. Were getting there as you might see.

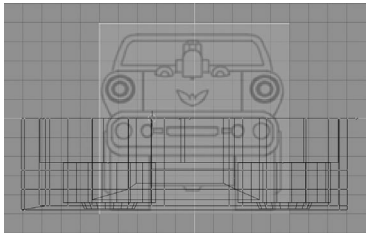


Press **ZKEY** to look at the car in shaded mode,

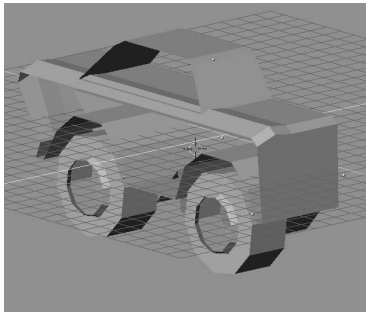


...now don't be alarmed...there are some holes in the model, but they will be closed up soon.

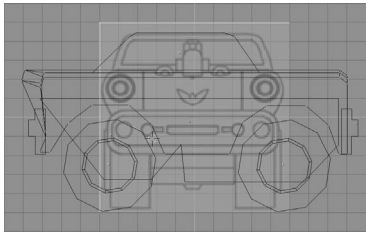
First we have to finish up a little work on the rear of the car. Go into topview by pressing **PAD7** and switch back to wireframe with **ZKEY**. Of course, this type of car needs the classic "wings" on the back.

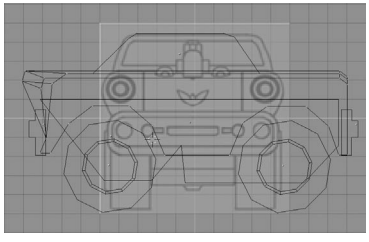


So select the body of the car and go into EditMode (**TAB**). Select the last two rows of vertices and deselect the two vertices second from the side of the car as shown in the image.

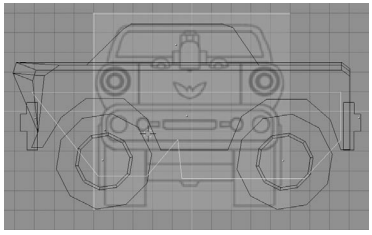


Grab them with **GKEY** and while holding **CTRL** move them 1 grid unit to the right.

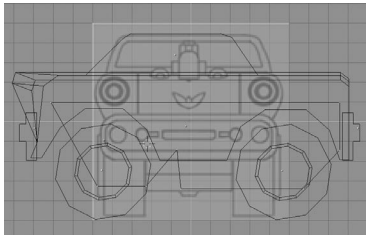




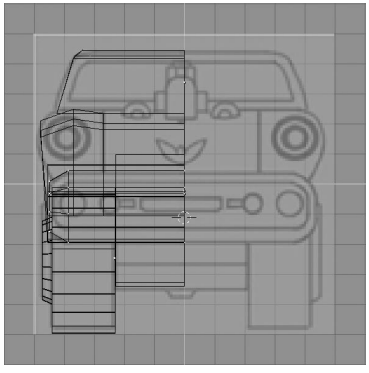
Go back to FrontView (**PAD1**) and smooth out the trunk of the car by moving the vertices in a smoother curve.

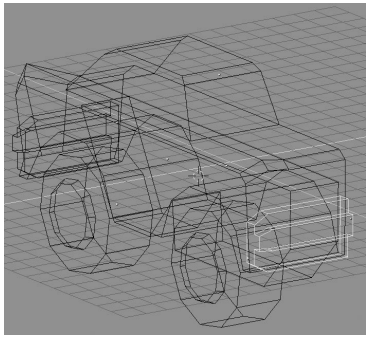


Now while still in FrontView, you might notice that the chassis is currently poking through the back of the car.



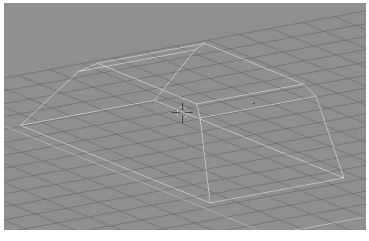
This is quickly fixed by moving the chassis' back vertices more to the right as shown in the screenshots.



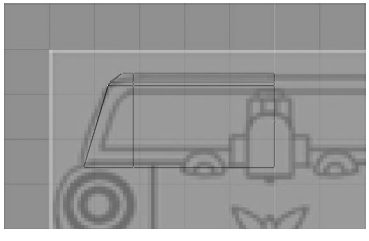


To finish this section, let's go back to FrontView to add the bumpers to the car using the same extrude method as we used previously.

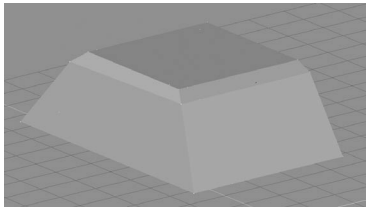
16.6. Closing up the holes



Now its time to close those holes in the side of the car. Select the top of the car and go into LocalView, done by pressing **PAD/** . LocalView allows you to work on a single object without the clutter of other objects in your view.



This view is especially handy for more complex models but it will be quite useful here too. Go into SideView (**PAD3**) once more and enter EditMode by pressing **TAB**.

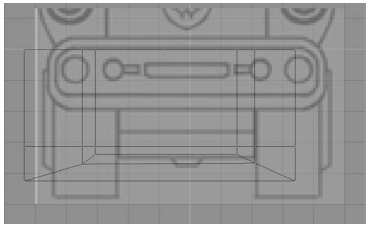


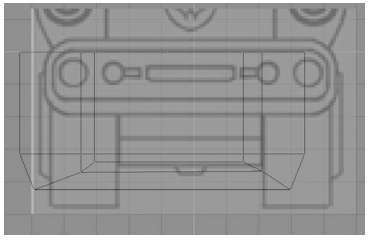
Select the top row of vertices on the outside, then hit **FKEY** to create a face to close it up. After that, select the bottom 4 vertices and hit **FKEY** to close the first part of the side.

You need 4 or 3 vertices to create a face in blender. “Faces” also mentioned earlier as “polygons” can be squares otherwise known as “quads” or triangles in Blender.

If you get an error “Can’t make edge/face”, check if you don’t have more than four vertices selected.

Tip: *Blender can also help you to fill faces with more than four vertices. Use SHIFT-F for this. However, when the vertices are not in one level this may result in ugly fills and unwanted triangle faces.*

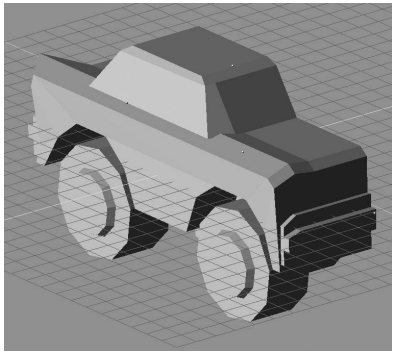




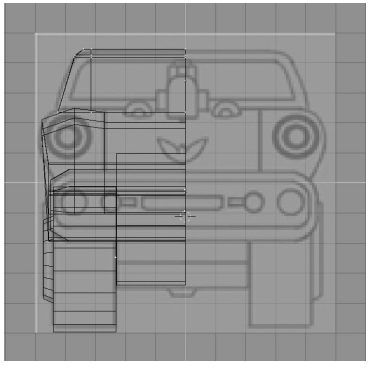
Switch back to TopView and we’ll smooth the edges of the top by moving the vertices at the edges inwards slightly.

Try to keep the faces you create as clean as possible, -- quads are preferred -- as this will save a lot of time when it comes to texturing the model. Following these steps and close up the sides of the model using the described technique.

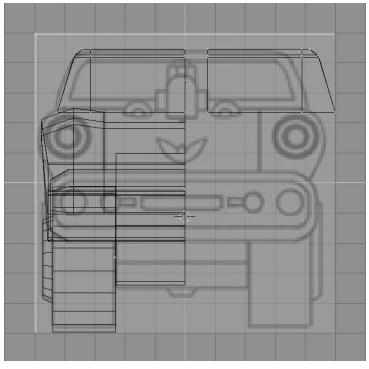
After every hole is closed up you will have something similar to this screenshot.



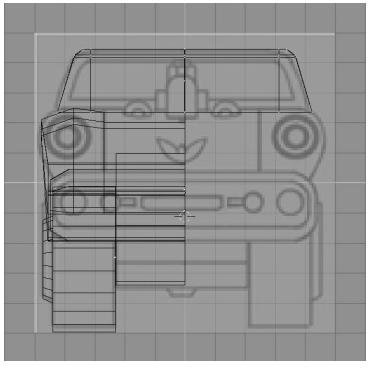
16.7. Flip it



Now its time to flip this half car over and create a complete one. Make sure you're in SideView (**PAD3**), select the top of the car and go into EditMode.

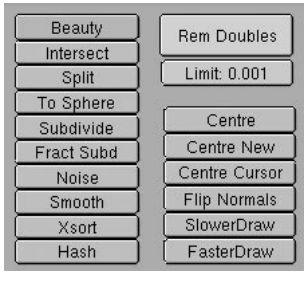


Select everything by hitting **AKEY** and pressing **SHIFT-D** to duplicate it. Now to actually flip it we first press **SKEY** and then **XKEY** without touching the mouse, because were going to flip the model over the X axis. Press **ENTER** to confirm the action.



It is also possible to flip objects across the Y axis with **YKEY**.

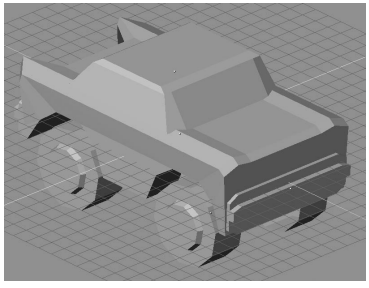
To connect the two parts we line up the middle row of vertices by moving the new part to the right and selecting the middle line of vertices (best done with Borderselect **BKEY**).



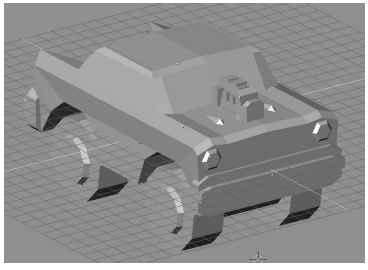
Because we now have two rows of identical vertices we can remove the vertices that are covering each other. Press **F9** to go into the EditButtons and click on the "Remove doubles" button to remove the double vertices.

If the vertices are close enough to each other you should have gotten a popup stating “removed: 6”. Good job, you now have removed 6 vertices and the model is now joined together. If you get a lower number increase the “Limit:” value a bit and try again to remove the remaining doubles.

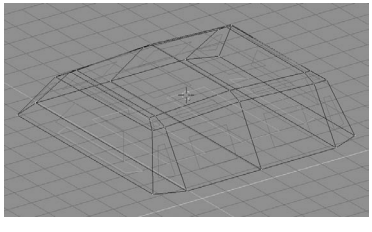
16.8. Finishing things off



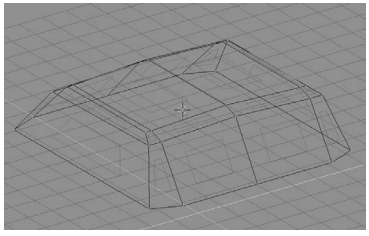
Once this is all done we should have a very nice low poly car if I say so myself!



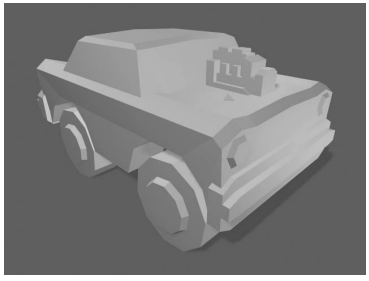
As you can see in the top right of your Blender screen, the model consists of 406 faces, and quite a lot of them are quads, a graphics card can only handle triangle polygons, don’t worry the graphics card will sort this out by itself. However we have to be aware that the 406 faces number then doesn’t mean we have 406 polygons, the real polygon count should be around 800 polygons, which is pretty decent. Now, if you want, you can add extra detail to the car like headlights, tail lights and a big ol’ engine block on the hood to make it a bit more interesting.



As a final note, once the model is finished you can look for hidden faces to optimize it. A useful tip for this is to turn on the “Draw faces” button in the EditButtons. I’ll take the top of the car as an example. In the screenshot, I have selected the faces that can’t be seen and are just a waste of polygons.



You can delete faces by selecting them, pressing **XKEY** and clicking on faces. And voila another eight polygons saved.



I hope you enjoyed this introduction to Blender's powerful modeling tools and fast interface in modeling objects.

You can get the final model from the CD as `Tutorials/Carmodelling/lowpolycar.blend`


001
003
2.03beta
004
2.04
game Blender
005
010
2.10
011
2.12
blenderGameKit
blender
POWER
CREATE
A
B
C
D
2.20
022
023
2.23


_part IV ::
intermediate
tutorials

The intermediate tutorials explain various individual aspects of making interactive 3-D graphics with Blender. This includes level design, the physics provided by the game engine, special effects like smoke and fire or making real-time displays for simulations or games.

The tutorials teach each of these aspects using ready made scenes to give you a quick learning curve, but they have been designed to be adapted by you for your own scenes.

Chapter 17. Super-G

Super-G is an Olympic discipline, and a mix between alpine downhill and slalom. I (Carsten Wartmann) decided to use this combination because it gives a nice mix between mayhem speed and sharp curves.

Figure 17-1. Super-G Ski Racing

You can load the file **`Demos/SuperG.blend`** from the CD and play it. You get a point every time you pass between two goals. Additionally your completion time is measured. This is already enough to create a little competition between friends, although it is not a complete game.

Table 17-1. Super-G game controls

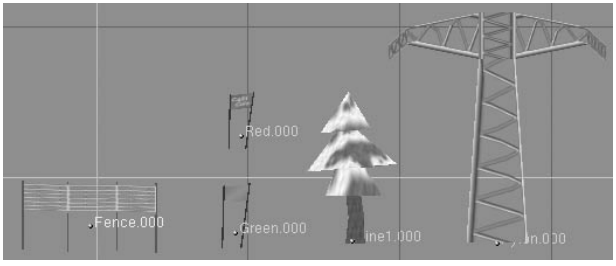
Controls	Description
CURSOR LEFT/RIGHT	Steer
CURSOR UP	Accelerate
CURSOR DOWN	Decelerate (a bit, drive a half circle to really stop)

The two tutorials in this chapter deal with the process of level creation and design.

17.1. Adding objects to the level

Load **`Tutorials/SuperG/SuperG00.blend`** from the CD. This file has no objects like trees or flags. You can race it (press **CTRL-LEFTARROW** for a full-screen view) but it is a bit boring and there will be no scores.

Figure 17-2. Some ready-made objects for Super-G

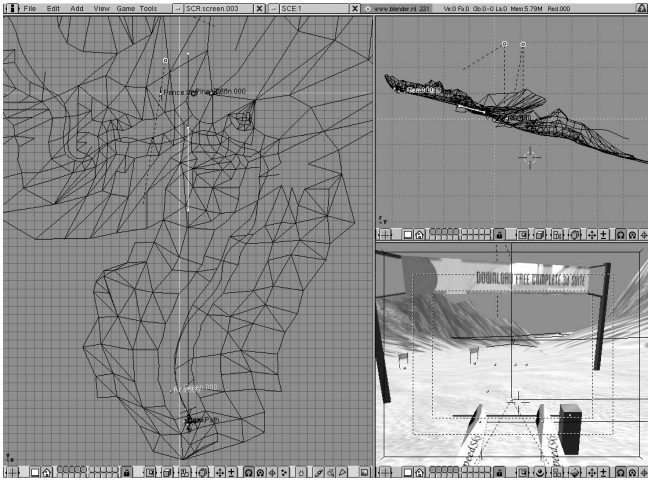


Contained in the file **`Tutorials/SuperG/LevelElements.blend`** are some objects which are ready-made for inclusion.

You can load these Objects to the level by appending them to the level scene. Use **SHIFT-F1** or "Append" from the FileMenu. Append all or only some objects, but if you want to append the flags "Green.000" and "Red.000" then you also need to append "Red_dyna" and "Green_dyna", these are needed for proper functionality.

I suggest you to place the flags first. In the scene there is a object called "TrackPath" which you can use as guide to place the flags. For placing the flags I suggest the window layout from Figure 17-3. It is a TopView left for the placing in the X-Y plane, a SideView to adjust the height and a CameraView to control the height of the elements.

Figure 17-3. Window layout for placing objects in the level



Select the red flag (**RMB**), and then extend the selection to the green flag with **SHIFT-RMB**. Using the grabber (**GKEY**) in the TopView, move both flags to the beginning of the track. Confirm the position with **LMB**. Without deselecting, move your mouse to the SideView and move both flags until you can see them sticking out of the snow in the CameraView.

Now as the first goal is set, we can continue making more. With the first two flags

still selected, place your mouse cursor over the TopView. Now copy the two flags with the **ALT-D**, the copied flags are now in GrabMode, so move them along the path and confirm the position with **LMB**.

Info: *We used ALT-D here, this makes a linked copy, meaning that the dozens of objects we will create in the end share their mesh, saving in filesize. See Section 4.12.*

Again move your mouse to the SideView and position the new flags in height. At this moment it is starting to be hard to see if the position is correct in the CameraView, depending on how far you placed the new flags.

One possible solution for this problem is to change the CameraView to a PerspectiveView and then always navigate to the new flag position by rotating, panning and zooming the view. Another method would be to create a new camera and move this camera along the track while creating new objects.

Blender offers a very convenient and fun way to place a camera. The so called "FlyMode". Create a new Camera with the Toolbox in the old CameraView. This newly created Camera will have the same point of view as the old one, so you can't see a difference straight away, but it is now the active camera. While with your mouse over the CameraView, press **SHIFT-F** to start FlyMode. Move your mouse and you can see the camera is rotating and banking according to your movements. Speed up by repeatedly pressing **LMB** and decrease speed using **MMB**. There are two ways to end FlyMode, pressing **ESC** will stop and set the camera back to its old position. Pressing **SPACE** will stop FlyMode and keep the current position.

After repositioning the camera you can select the flags again and continue with the copy and place cycle until all objects are placed on track. The placement of the other objects works the same way. To switch back to the player view, select the camera attached to the player and make it active by pressing **CTRL-ALT-PAD0**.

17.2. Object placing with Python

Even with a lot of experience of navigating in 3-D world, the task of placing many objects gets boring and distracting. To avoid this, I will show now you an advanced technique taking advantage of Python scripting inside Blender's game engine and Blender Creator. This tutorial is meant to give you an idea of how to use and combine Blenders tools to achieve complex tasks in a short time. It requires some basic understanding of a programming language.

Blender's game engine can be used to let objects fall onto the level ground. The problem is that these positions will only be valid as long as the game engine runs. Although it is possible to have hundreds of objects with game logic needed to achieve this task, it will slow down the game and make it unplayable on slower computers. The solution is to store the position in a file on disk and the use the positions out of this file in Blender Creator to place objects without the game logic.

Load the scene `Tutorials/SuperG/SuperG00.blend` as base for this tutorial. Then append all Objects from `Tutorials/SuperG/LevelElements.blend` to this scene.

Place all the elements you want to have in the level in TopView, use the same procedures to copy and move as described in the previous chapter, but with no need to place them exactly on the ground. Make sure that all elements are placed above the ground.

The object "DesignHelper" contains the LogicBricks and Python script we need.

Now we need to select all of the objects to place. Switch to the ObjectBrowse with **SHIFT-F4**, here you can select multiple objects with **RMB**. Press **ENTER** when you are satisfied with your selection.

Switch back to the 3DWindow with **SHIFT-F5** and extend the selection with the "DesignHelper" object. Now copy the LogicBricks of the helper to all other selected objects with **CTRL-C** and choose "Logic Bricks" from the menu.

Open a TextWindow with **SHIFT-F11** and browse to the "WritePositions" Python script.

Figure 17-4. Script to write object positions

```
 1 import GameLogic
 2
 3 # CHANGE the file/path name!  (e.g. c:\temp\objs.txt)
 4 FILE="/tmp/objs.txt"
 5
 6 f = open(FILE,"a",8192)   # opening for appending
 7
 8 contr = GameLogic.getCurrentController()
 9 owner = contr.getOwner()
10
11 # get position
12 pos = owner.getPosition()
13
14 f.write(owner.getName()[2:]+" "+str(pos[0])+"
   "+str(pos[1])+" "+str(pos[2])+" \n")
15
16 f.close()
```

Change the filename in line 4 to reflect your hard disk layout and needs. Now start the game engine and wait until all objects are settled on the ground, then stop the game engine. You can examine the file created by the script now, it is a regular text file containing the names and positions of the objects line by line. You can open this file in any text editor.

To read back the positions, change the filename in the "ReadPosition" script and press **ALT-P** with your mouse over the TextWindow.

Figure 17-5. Script to apply the object positions

```
 1 import Blender
 2
 3 # CHANGE the file/path name!  (e.g. c:/temp/objs.txt)
 4 FILE="/tmp/objs.txt"
 5 f = open(FILE,"r",8192)
 6
 7 lines = f.readlines()
 8
 9 for line in lines:
10     words=line.split(" ")[:-1]
11     obj=Blender.Object.Get(words[0])
12     obj.LocX=float(words[1])
13     obj.LocY=float(words[2])
14     obj.LocZ=float(words[3])
15
16     Blender.Redraw()
17
```

Throughout this tutorial I have shown you the most common way in which to build the level design phase of a game. During the development of any bigger game or interactive 3-D application, you would need to write a set of tools or scripts to help you with your work. These tools will take some effort initially to write, but they will help you in the future when you wish to further develop or change your games. Also, the tools or scripts can be re-used in future projects as well. Blender's approach, to contain all tools for 3-D content creation combined with the Python capabilities, gives you a fast and flexible toolset for nearly all aspects of your work.

Chapter 18. Power Boats

Figure 18-1. "Power Boats" 3rd person perspective

The „Power Boats" game was designed by me (Carsten Wartmann) with the idea of making a fun game of driving around in a powerful boat. I tried to make it a bit more of a simulation than an arcade game, but because I have never driven a real power boat, I doubt I was very successful.

I added some elements that make a game, like the counting and timing of laps. However, to make it a real game there is more needed, like an intro, more levels, and of course other boats (CPU controlled) to race against.

So enjoy driving around, explore my scene and of course expand or change it! I am very curious as to what people can do with it in terms of making it a game or creating new levels.

Figure 18-2. "Power Boats" 1st person perspective

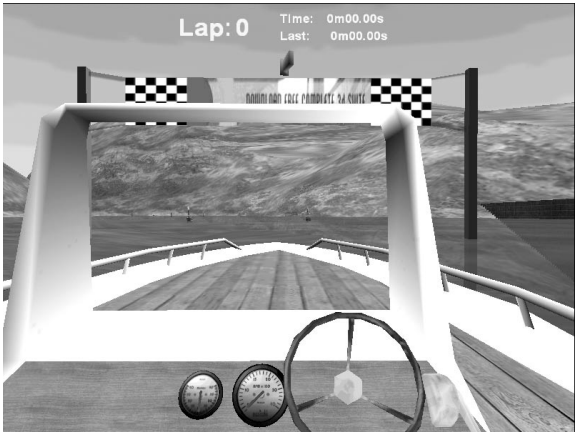


Table 18-1. Powerboats game controls

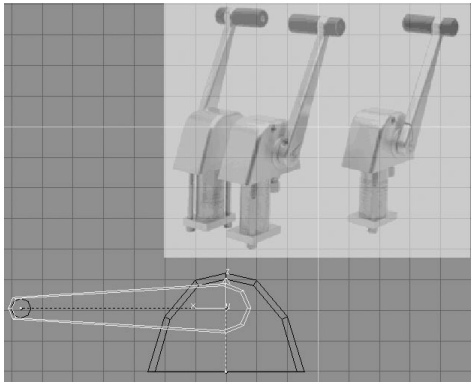
Controls	Description
CURSOR LEFT/RIGHT	Steering, you need to have some speed to steer
QKEY	Increase throttle
AKEY	Decrease throttle
F1	First person perspective
F2	Camera behind boat
F3	3rd person perspective
F4	Helicopter camera

The tutorials in this chapter deal with methods of making real-time cockpit instruments and how to pass information between game elements.

18.1. Engine control

The engine control is mainly for showing the throttle position, so it is in fact an instrument.

Figure 18-3. Engine control in 3DWindow with modeling help



Load the file **`EngineControl00.blend`** from the **`Tutorial/Powerboats/`** directory on the CD or model one yourself.

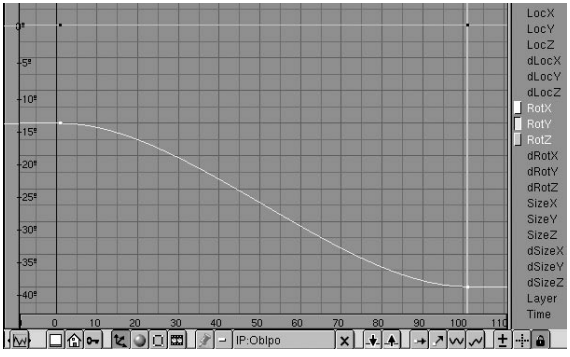
Select the lever ("EngineCArm") with the right mouse button, we are going to create an animation curve for this object. The object should not be rotated, so that we can use one main axis for rotating the lever.

Info: *Ipo animation curves in Blender are global to the scene (world) axis. To overcome this we can use object hierarchies done by parenting. After parenting we can rotate or move the parent object and the Ipos on the child will get their reference from the parent object.*

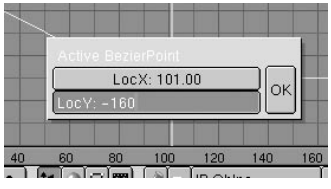
Rotate the lever around the Y-axis about 15 degrees to determines the neutral position. Insert a keyframe with **IKEY** and choose "Rot" from the pop up menu. You should have an IpoWindow (**SHIFT-F6**) open so you now can see horizontal lines of the new generated Ipos.

Advance the current frame by pressing **CURSORUP** ten times. Rotate the lever another 20 degrees and insert another "Rot" key frame by pressing **IKEY**. We can't rotate the lever completely to the end position because Blender then will try to interpolate (Ipo stands for "interpolation"-(curves)) the key positions and that will lead to an unwanted result of Ipos for more than one axis.

Figure 18-4. Animation curves (Ipo) for all three axes



Move your mouse cursor over the IpoWindow and select the curved line (RotY when you use the prepared scene) with **RMB**. Now enter EditMode (**TAB**) for this curve. The right key should be selected (yellow, **RMB** to select). With your mouse over the IpoWindow press **NKEY** and enter -160 in the "LocY" field.



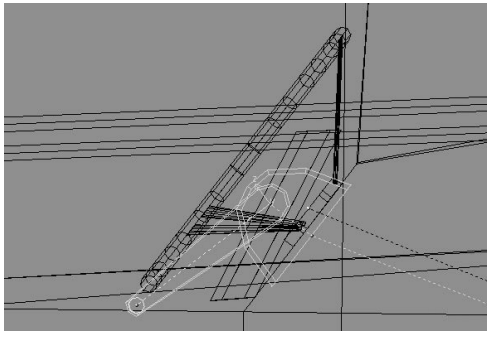
You can now test the Ipo by pressing **ALT-A** over any 3DWindow. Save the file now.

Load "Powerboats_00.blend", this file does not have the engine control and instruments for the FPS perspective, so we can use it for this tutorial.

Use **SHIFT-F1** or "Append" from the FileMenu and browse to the prepared engine control file (your file or "EngineControl_02.blend" from the CD). Enter "Objects" and select all three objects (**RMB**), then press **ENTER** or click the "LOAD LIBRARY" button. The engine control is now appended to the actual scene together with the animation curve and preserving the object hierarchies.

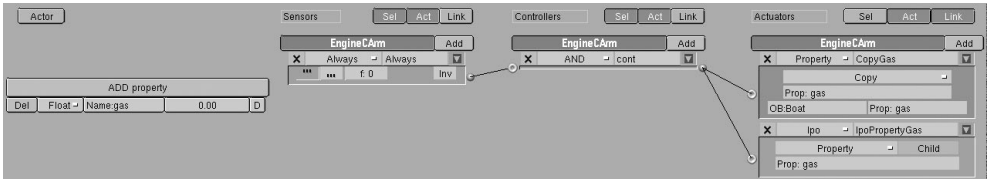
Zoom out until you see the “mountain size” engine control. Select the base of the control, use rotate, scale and move until it fits to the boat and is positioned nicely on the dashboard. Now extend the selection by **SHIFT-RMB** on the boat mesh (“Hull”), and parent the engine control to the boat with **CTRL-P**.

Figure 18-5. Engine control mounted to the dashboard



Now the control is mounted to the boat, but it needs to be made functional. Select the conrol lever (“EngineCArm”) with **RMB** and switch to the RealtimeButtons **F8**. Create the LogicBricks as shown in Figure 18-6

Figure 18-6. Engine control LogicBricks

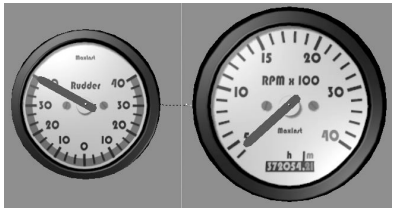


These LogicBricks copy the “gas” property (the throttle position) from the Boat object to the local “gas” Property of the lever object. The values of “gas” reach from 0% to 100%, and we play the Ipo with the Property Ipo Actuator according to this value, so that the lever will always reflect the “gas” Property.

18.2. Cockpit instruments

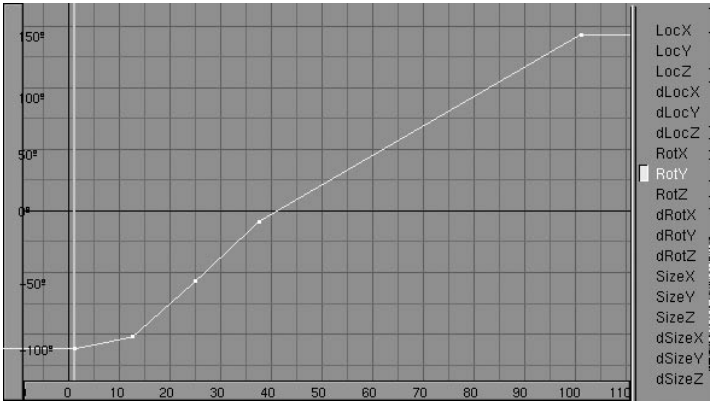
The procedure to make cockpit instruments is very similar to the engine control. In fact the engine control is not a control here but also an instrument.

Figure 18-7. Intruments



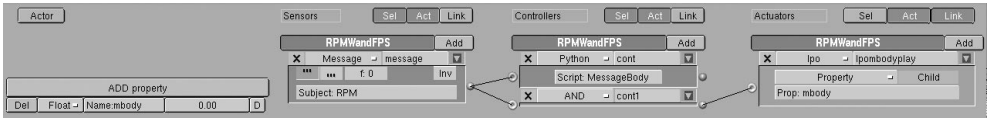
The instruments are modeled in Blender an then the rendered image is used as texture in the Power Boats scene. You can see and change the instruments in the scene **`Tutorials/Powerboats/Instruments.blend`**. One thing to note is that the scale of the revolution counter is not linear. To make it display the revolutions correctly, a complicated formula is needed. Or even better with Blenders Ipo curves, we can visually calibrate any displays very easily by just moving Ipo handles.

Figure 18-8. Ipo for the revolution counter



Another difference is the way we pass the value for the revolutions to the instrument. For the engine control we used the Copy Property Actuator. This is an easy way to do it, but will not work across scene borders. When you switch to a 3rd person view in the PowerBoats game, you get the instruments as an overlay, this is done by rendering an overlay scene onto the main scene. In this case we need the rpm value on two instruments and across a scene border. The solution here is to use Blender's messaging system. The rpm value is then sent by the engine, and every instrument can listen to this message and process it. This way adding an instrument (in this case for example a warning when the rpm gets too high) is very easy.

Figure 18-9. LogicBricks to receive rpm messages and play the Ipo



You can see in Figure 18-9 that we receive messages with the subject "RPM" to trigger the "MessageBody" (see Figure 18-10) script. This script extracts the message body into the property (which needs to exist on the object) "mbody". The Property Ipo Actuator then plays the Ipo according to the value in "mbody".

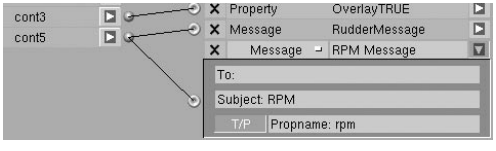
Figure 18-10. Script to extract message bodies

```
 1 # Extracts message bodies
 2 from types import *
 3
 4 cont = GameLogic.getCurrentController()
 5 mess = cont.getSensor("message")
 6 me   = cont.getOwner()
 7
 8 bodies = mess.getBodies()
 9 if bodies!=None:
10     for body in bodies:
11             if type(me.mbody) is StringType:
12                     me.mbody = body
13             else:
14                     me.mbody = float(body)
```

To make this script work without changing it, the Message Sensor has to be named "message" and there needs to be a Property "mbody". The script will convert the message body into the correct type (i.e. String or Integer) of the Property.

Info: *To take full advantage of Python, we recomment installing a complete Python distribution, it also comes with a very good guide to Python. Because the development of Python is very fast too, we suggest to use Python 2.0 for the best compatibility with this version of Blender. All games in the* `Demos/` *directory on the CD work without a full Python installation.*

Now there is one open point: how does the messages need to be send? That's where the Message Actuator kicks in.



Note the "Subject:" field and that the "T/P" Button is activated.

The lessons learned in this tutorials can be applied to many different games and interactive 3-D applications, where you need controls or displays. We showed you two possibilities: one of how to pass information between objects and the other how to use Blender as a tool to produce textures for the game engine. The use of Blender's animation curves gives us the possibility to change and adjust the displayed information graphically.

Chapter 19. BallerCoaster

by Martin Strubel

This tutorial will cover some detail about real natural behavior, so fasten your seat belts for some physics in the last section! But please don't skip to the next chapter yet, it's not all that bad, and of course we will start with the fun stuff first.

The Blender game engine is able to simulate the natural behavior of rolling balls quite nicely - this fact suggests to build a roller coaster, or rather, making a trade mark of it: a BallerCoaster!

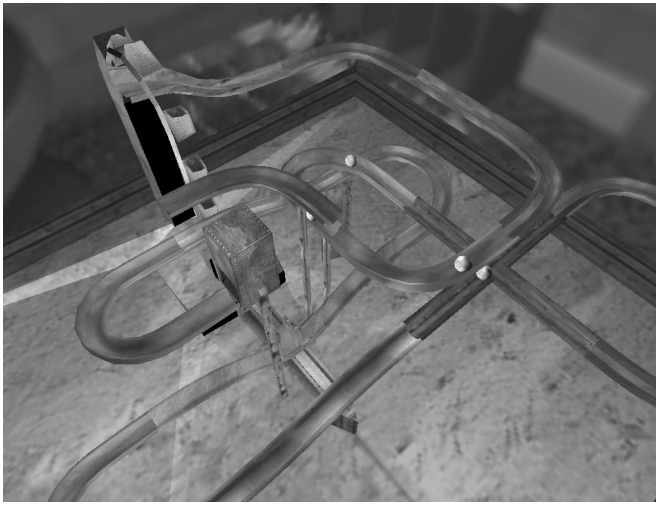
We want to save you, dear reader, much time, though. Therefore, we provide you with a kit of "prefabricated" elements such as curves, slopes and other path types which you can simply plug together. We will also show you how to generate your own path elements using Blender's *bevel* curves.

At this point I would like to say thank you to my dear mate Freid who modeled the room environment and let me use it for the background.

19.1. Assembling a track

To give you an idea of a possible result first, look at the provided demo file: `Demos/BallerCoaster.blend`, see also Figure 19-1

Figure 19-1. BallerCoaster demo



Start Blender, and load the demo file using **F1**. To start the demo, move the mouse over Blenders 3DWindow and press **PKEY**. I have also added some extras, accessible using the following hotkeys:

Table 19-1. BallerCoaster game controls

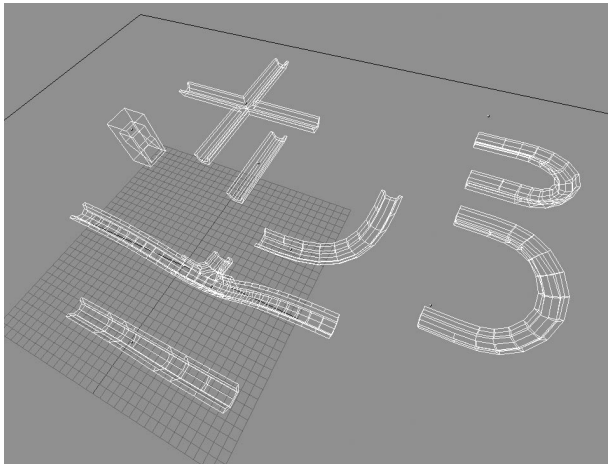
Controls	Description
NKEY	Make the ball dispenser spawn some fresh balls
SKEY	Switch track elements (just try!)
SPACE	Toggle between ball camera (following red ball) and observer's perspective
AKEY	Switch fixed camera view
ENTER	Restart demo

When you have become a little dizzy by looking through the crazy ball camera, it's probably time to make your own track now -- we will look at the demo again later when discussing some game logic: some (little) intelligence attached to the objects. Switch to the screen "elements" with **CTRL-RIGHTARROW**. You will now see a bunch of path elements which are waiting to be grabbed and moved with the mouse (see Figure 19-2). This is how you do it:

1. Switch to TopView using **PAD7**
2. Select an element with **RMB**.
3. Press **ALT-D** to make a *linked* copy of the object (see Section 4.12).
4. Press **GKEY** to move the element, while holding **CTRL**. This will snap the position of the object to the grid, helping you with the alignment of the elements. The same works with rotation: press R, rotate with the mouse while holding down **CTRL**.
5. Switch to FrontView or SideView using **PAD1** resp. **PAD3** and move your object to the desired place (with **CTRL** hold).

If you accidentally forgot to hold **CTRL** while moving the object, you can always snap it to the grid again by pressing **SHIFT+SKEY**, "Sel->Grid".

Figure 19-2. Path elements



Assuming that you have plugged together some elements to a decent track, we now want to test it! Grab the ball dispenser (the little rusty box) and place it above the track. Press **PKEY** to run the demo and **NKEY** to add some balls. You will see the balls disappear when they hit the ground plane. This is intentional, because we don't want to crowd the scene with too many dynamic objects -- you will notice, that if you add a lot of balls at once by firing **NKEY**, the whole animation may get quite slow. This is due to the dynamics calculations which take quite some CPU time.

19.2. Game Logic

To get a nice cycle to work as in the demo, we will need to add some more logic. Let us have a look at the logic of the simple elements we have used until now.

- Ball dispenser: Pressing **NKEY** produces a new ball
- Balls: If they hit the ground, they are destroyed

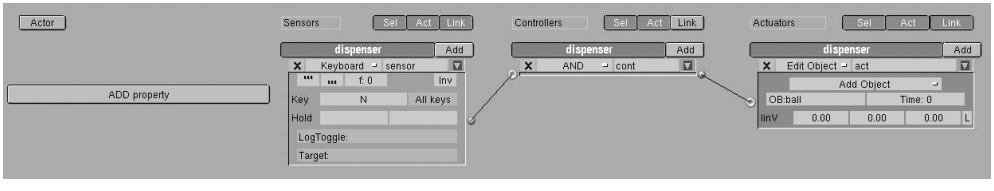
In Figure 19-3 you can see the LogicBricks of the object "dispenser", as displayed in the RealtimeButtons **F8**. The keyboard event simply adds a new object "ball" at its position, marked by the little pink *PivotPoint*. This ball object is located in layer 10 (**ZEROKEY**).

Info: *Objects to be added inside the game engine must always be in a non visible (inactive) layer! Read more about layers in Section 24.1.1.*

Now change to layer 10 by pressing **0KEY** and select the ball to check its attached logic. Tip: Press **HOMEKEY** in the 3DWindow if you feel lost in 3-D space -- this will help you to locate your objects.

The ball logic is simple: When the ball collides with an object with the Property "death", it will cause its own end. Now get back to layer 6 and select the ground plane - it has indeed the property "death", shown on the left side of the RealtimeButtons. It's that easy!

Figure 19-3. LogicBricks for the dispenser



In layer 10 you will also find the bucket object which is used to fake the ball elevator. This one has a more complex logic and is not too elegant, but it's still simpler than defining an animation for each single bucket. We will not go into the details of its logic though, this is left to the reader.

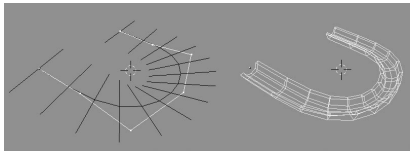
19.3. Making track elements

It may get a little boring after a while, assembling all the prefabricated stuff. We will now show you how to make the track elements yourself.

Bevel curves

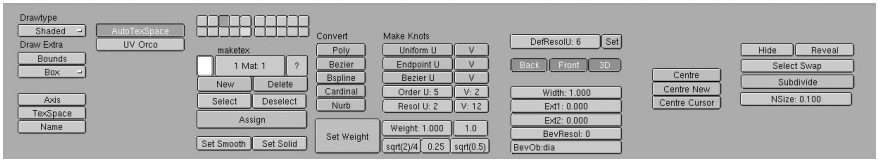
In the demo file, select the screen “elements” from the screen menu (or press **CTRL-RIGHTARROW**). Here you find all the element prototypes, in the form of *beveled curves*. This curve type consists of a path (a 3D curve) and a diameter curve, which is extruded and oriented along the path. Let’s have a closer focus on this (see Figure 19-4).

Figure 19-4. Beveling a track



1. Select one of the bevel curves with **RMB** and press **TAB** to switch into EditMode.
2. You can now move the control vertices of the path as usual, and change the local orientation with **TKEY** and by moving the mouse. The orientation of the path is immediately reflected in the railway like representation.
3. Pressing **TAB** again will recalculate the extrusion of the diameter curve.

Figure 19-5. EditButtons for the bevelcurve



The easiest way to create new tracks is just, to make a true duplicate (using **SHIFT-DKEY**) of an existing track element and modify its curve.

If you want to start from the bottom though, these are the steps of reconstruction:

1. Activate the EditButtons by **F9**.
2. Add a Bezier Curve by pressing **SPACE**->Curve->Bezier Curve (a NURBS curve works too).
3. Add a second, preferrably closed curve (Bezier Circle), modify it to an U shape. Rename this object to, say “dia” by entering the name in the “OB:” field (Figure 19-5)
4. Select the first path curve again, activate the “3D” setting in the EditButtons and enter the name of the U shape curve (“dia”) in the field “BevOb”.

5. Adapt the resolution of both of the curves to the most acceptable low polygon count. You might want to convert the diameter curve into a Polygon Curve by clicking the "Convert Poly" Button (Figure 19-5) to get the best control over the shape.

Finally, the bevel object must be converted into a mesh for the Game Engine. This is done by selecting the object and pressing **ALT-C** for "Convert Curve To: Mesh". But careful! You are advised to make a duplicate copy (**SHIFT-DKEY**) of the curve objects before conversion, because the original curve object will be lost otherwise.

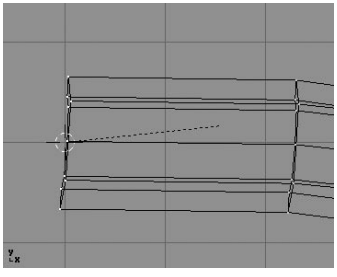
A few more tips about curve editing:

- If the resolution of a curve near a control vertex is to great, change its weight by selecting the vertex, choose the weight from the weight button group (or enter manually by holding **SHIFT** while clicking on the "Weight:" button) and apply "Set Weight".
- You may also want to play with the order setting of the curve to modify the bending shape. But remember that the minimum number of control vertices must be greater than the order. For our needs, the "Endpoint" option might also be the most desired one.

Plastic surgery -- some mesh beautification

After conversion to the mesh, some alignment corrections of vertices of the path element's ends might be necessary, see Figure 19-6. Otherwise, ugly effects can occur after "plugging" slightly intersecting path elements together.

Figure 19-6. Aligning vertices



Use the Border Select tool **BKEY** to select all the end vertices, then go for the following steps:

1. Set the 3DCursor to a grid point using the LMB and pressing **SHIFT-SKEY**, then **3KEY** ("Curs->Grid") to snap the cursor to the grid.
2. Press **DOTKEY** to use the 3DCursor as scaling center.
3. Place the mouse cursor on the right of the 3DCursor to indicate the direction for the scale constraint, then press **SKEY** followed by **MMB** to constrain the scaling in *X* direction. Hold down **CTRL** and scale the vertex group's *X* components to 0.0, yielding a perfect alignment in *X*.

Texturing

We will not go into much detail about texturing here, but give you a quick intro on how to glue a texture on your newly created track element:

1. Select the desired mesh and hit **FKEY** to enter FaceSelect mode. Press A to select all faces.
2. Open an ImageWindow **SHIFT-F10** and either choose an image from the MenuButton or load a new image by clicking "Load Image".
3. Move your mouse pointer over the 3-D view and press **UKEY**. This will pop up a mapping menu. Use the "Cube" mapping option; as a result, you will see the so called "UV mapping" of all selected faces in the ImageWindow.
4. You can edit these mappings by the same methods of vertex selection and transformation as you used in the 3DWindow.

19.4. The nature behind BallerCoaster

This is only for the interest of those who want to know the answer to the question: why does this demo behave quite like the real world, how is it done?

Gravity

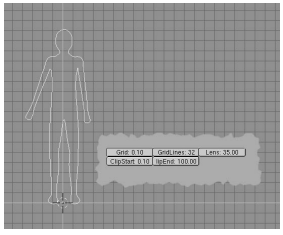
Most of the phenomena around us, as observed daily, are based on forces. We are aware that there is a mystic force between any entity in space, especially omnipresent between earth and ourselves -- gravity! In the simplified world of our BallerCoaster, you can observe several types of motion: free falling, the static touch of an object with the ground, rolling, sliding, or collisions. All these cases are handled by the built in physics engine, so that for the user there is in this simple demo no need to take over control of the motion - the driving force is the gravity.

Dimensions

Some words about dimension: a question that is often asked by users is: what is the measure of Blender grid units in the real world? Meters? Inches? Feet?

The answer is actually: It's the way you want it to be. Very helpful, isn't it? Let us find a better answer: If you look up the gravity settings in the WorldButtons, the default value is 9.81 - which means, that with this setting, one Blender unit is equivalent to 1 meter in the real world. But you can easily calculate and set the gravity acceleration in inches, and use a Blender unit as an inch, with the same physical behavior. If you want to build a consistent world with the same scaling, but different physics, it really helps to model everything in its absolute size (say, if you want to model a 2 meter tall ogre, you would use 2 Blender units), and use the default gravity, if you're keen on the true natural behavior. Then later on you can tweak the gravity factor, or even use different worlds with different gravities. Figure 19-7 shows an example on "human" (172 cm height) dimensions, the grid units are set in the current 3DWindow by pressing **SHIFT+F7**.

Figure 19-7. Dimensions



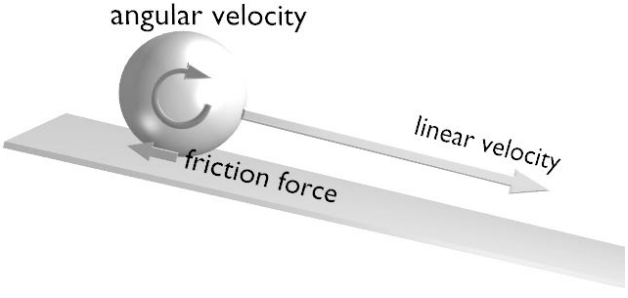
In our demo, the balls have a size of 0.04 units, being equivalent to 4cm. We use a gravity lower than 9.81, which slows down the motion a bit - we don't want to get too sea sick by the crazy ball camera view. Moreover, a displayed grid unit is equivalent to 0.05 Blender units, as set in the ViewButtons (press **SHIFT+F7** in the active 3DWindow).

Rolling, rolling, rolling

The most occurring motion state in our BallerCoaster is the rolling case. We don't want to get into scientific details - you might have already got sick of these, hearing them in physics class. But what is causing the rolling motion anyway? It's about friction! If you pull your balky donkey (who is of course refusing to walk) by its leash, you will need quite some force to get him moving. This force, is caused by the friction between his feet and the ground, is called friction force. If you put a non-rolling object on an inclined plane, you can find different cases of motion again, depending on the friction between the object and the ground as well as the inclination of the plane:

- Sticking: the external resulting force acting on the object is not great enough to compete with the friction force. This is a case of "static friction"
- Sliding: the applied force is greater than the static friction force threshold - it's a case of kinetic or dynamic friction.

Figure 19-8. Forces on a rolling object



For a rolling object, the same friction states occur, but the effect is different. In both of the cases, the friction force will apply a torque to the object and make it roll (Figure 19-8 illustrates this).Therefore, no matter how high the friction is, the rolling object is not decelerated (unless it's sliding). But sliding can occur in different variants:

- The ball does not move very fast, but spins on the underground. For physicists, this means: its rotational energy is about to be converted into a linear kinetic energy (some part of the energy is burned in the friction process)

- There's little rotation, but a lot of linear motion. The ball will get a rotational acceleration by the kinetic friction. Also, some energy will be lost in the sliding process (your donkey's feet can get HOT!).

So, the simple conclusion of our physics aspects: If your donkey refuses to move, put it into a barrel and roll it!

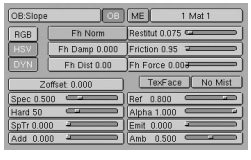
What do we have to worry about now? The case of sliding or sticking is handled by the physics engine. We must only specify the material parameters of the objects involved, meaning, the friction coefficients. Friction is normally a parameter describing the interaction of two materials, in Blender this is a little simplified and friction is expressed as a value per one material. For the interaction of two materials, the minimum of both of the involved friction values is taken.

Setting dynamic material parameters

This is a quick guide to dynamic materials (also examine the demo and see Section 26.6):

- Select the object, switch to the MaterialButtons **F5** and choose a material "per object". Normally, materials are assigned to a mesh, but we want to assign the material to the object itself, so that we can use several objects with a shared mesh, but not shared materials. Material assignment per object is selected using the "OB" button (Figure 19-9)
- Click the "DYN" button to switch to the dynamic parameters of the material.
- Use a low restitution (elasticity) of the materials to avoid too much jumping. For the two materials' interaction, the maximum restitution value is taken.
- Use a relatively high friction on the balls and a lower friction on the track elements - remember, the minimum of the friction values is used.

Figure 19-9. Dynamic Settings in the MaterialButtons



Last remarks

When you have an older 3-D graphics accelerator or computer, the demo will probably run at a slow frame rate and it may happen that balls get lost because they don't respect collisions at all above a certain speed (quantum physics people call this effect "tunneling"). This is of course an undesired effect in a game engine, but as we chose for speed rather than accuracy, you will have to live with that fact (or consider upgrading your PC).

Happy rolling!

Chapter 20. Squish the Bunny - Creating Weapon Effects for a First-Person Shooter

The tutorial written by Randall Rickert outlines an approach to creating weapon effects in Blender, using the file `Tutorials/SquishBunny/stb-tutorial.blend` as an example of the techniques. You will learn to add a smoke trail effect to a rocket. These, as well as more sophisticated techniques, are demonstrated in the game scene when the player fires a rocket.

Table 20-1. Squish the Bunny game controls

Controls	Description
WKEY	Move forward
SKEY	Move backward
AKEY	Sidestep left
DKEY	Sidestep right
SPACE	Jump
LMB	Shoot a rocket
Mouse movement	Rotate and look

Figure 20-1. Squish the Bunny demo file

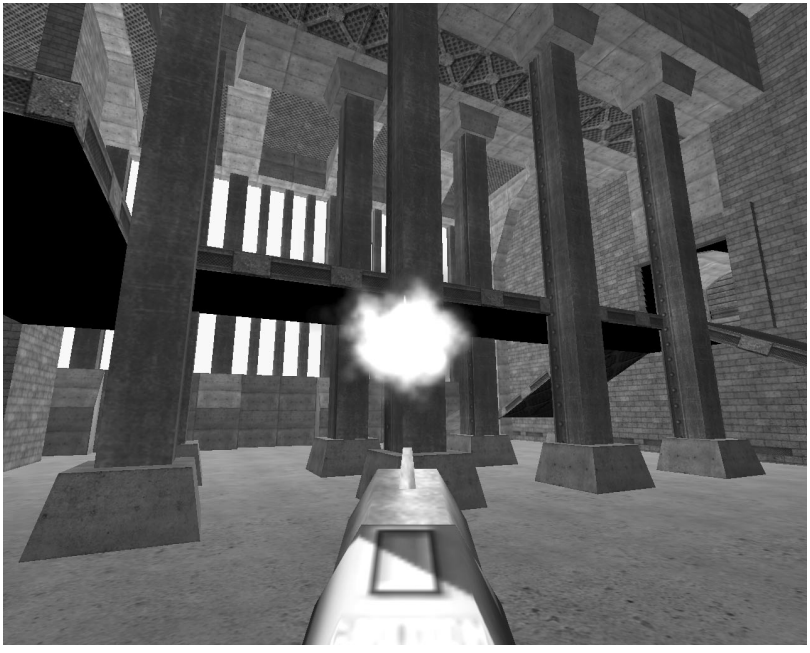
The “Squish the Bunny” demo file was made by:

Table 20-2. The “Squish the Bunny” demo file was made by

Artist	Role
Randall Rickert	Design and Logic
Randall Rickert and Reevan McKay	Eye Candy
Janco Verduin	Ear Candy

20.1. Introduction

Figure 20-2. The generator room



This tutorial will outline an approach to creating weapon effects in Blender. We will be editing a simple first-person shooter game scene as our example. In this scene you can run around in an immersive environment, firing a rocket launcher which makes fiery explosions and leaves scorch marks on the walls. This scene could form the basis for a game in which you fight your way out of a fortress, carry out a mercenary mission, or test your mettle in a gladiator-style tournament. The goal of this tutorial is to show you how to make some of the eye candy which is essential in making such a game engaging. A scene showing the effects in action is `Demos/Squish-the-Bunny.blend` (Figure 20-2).

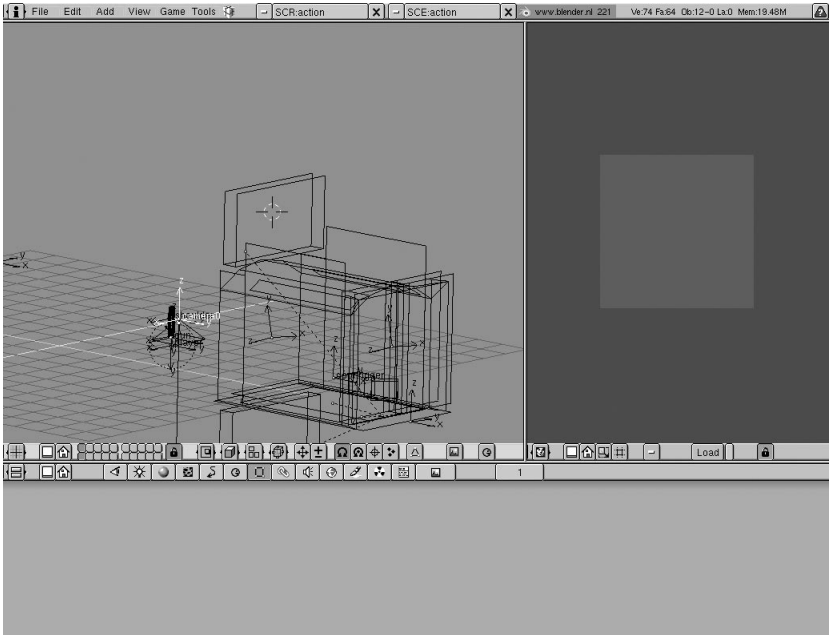
20.2. Getting Started

Start Blender and open the file `Tutorials/SquishBunny/stb-tutorial.blend`. This file has an environment model and a human-perspective camera with enough game logic to allow the player to navigate using the keyboard and mouse in the style of some popular first-person shooter games, and to shoot a rocket from the rocket launcher. To see it in action, press **PKEY**. Use **WKEY**, **SKEY**, **AKEY**, and **DKEY** to move around, and **SPACE** to jump. Move the mouse to turn and to look up and down. **LMB** will shoot a rocket. Press **ESC** when you are ready to exit the game.

20.3. A Trail of Smoke

Our first step toward spicing up the action will be the addition of a smoke trail behind the rocket. A trail of smoke will make the fast-moving rocket more visible and it will add a lot of visual depth to the scene, giving the player a better sense of the space.

Figure 20-3. The editing screen



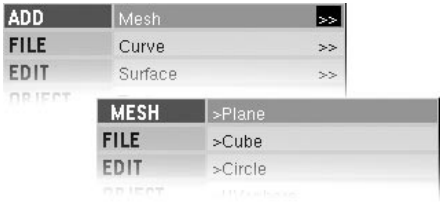
After exiting the game, press **CTRL-LEFTARROW** to switch from the current screen to one which is better suited editing the scene. You will see a 3DWindow on the left side of the screen with a wire frame view of the scene, an ImageWindow on the right, and a ButtonsWindow at the bottom showing the EditButtons (Figure 20-3).

20.4. Building a Puff of Smoke

To represent the puffs of smoke which make up the trail of the rocket, we will use a special kind of polygon called “halo”.

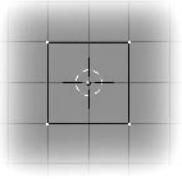
A halo face is a polygon which always faces the camera. When used with a texture containing transparency information in an alpha channel this type of polygon can create the illusion of volume, because the camera’s perspective won’t flatten the polygon by viewing it from the edge.

Figure 20-4. Add a plane mesh



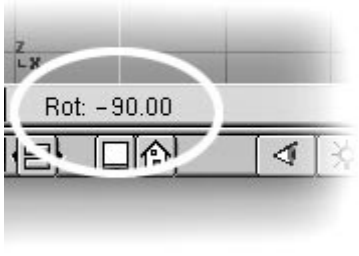
With your mouse in the 3DWindow, press **PAD_7** to switch to TopView (make sure you have **PAD_NUMLOCK** turned on). Press **SPACE** and from the pop up menu select "ADD>Mesh>Plane" (Figure 20-4). A plane is added to the scene and automatically placed in EditMode (Figure 20-5). Press **PAD_1** to switch to FrontView.

Figure 20-5. The plane in EditMode



For each polygon, one side is considered to be the face side. For the sake of speed, the game engine only renders the face and not the back of each polygon. We need to orient the face of the plane toward the negative end of the X axis (the left side of the window), as this is the side of an alpha face which will face the camera.

Figure 20-6. Amount of rotation

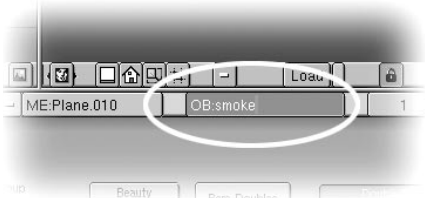


Press **RKEY** to begin rotating the vertices. Hold **CTRL** to constrain the rotation to 5° increments as you rotate the vertices -90° (anti-clockwise). You can see the degrees of rotation in the WindowHeader at the bottom of the 3DWindow (Figure 20-6). When the rotation reaches -90° click **LMB**, which will apply the rotation and end RotationMode. Press **TAB** to leave EditMode.

Press **RKEY** again. Now you are changing the basic orientation of the object (not just moving the vertices of the polygon). Rotate the object 90° (clockwise), facing upwards so that you will be able to see the texture on the plane once you applied it.

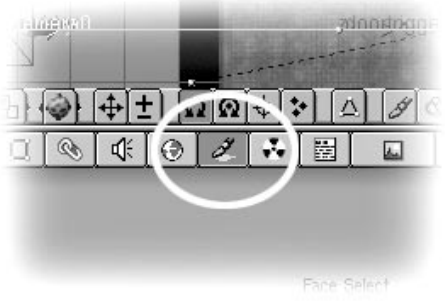
Turn your attention to the EditButtons (**F9**). Change the name of the object to "smoke" (Figure 20-7) by clicking into the "OB:" field and enter the new name. This will allow you to refer to the smoke object in the game LogicBricks, which will be necessary a little bit later.

Figure 20-7. Change the name of the smoke object



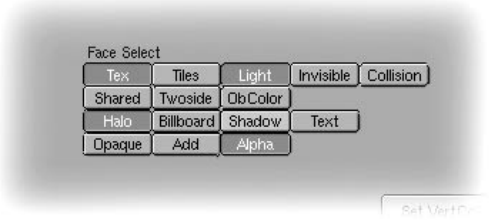
In the 3DWindow, press **PAD_7** to return to TopView. Press **ALT-Z** to change the DrawMode of the 3DWindow from wireframe to *OpenGL* textured. The smoke object will turn black. Press **FKEY** to enter FaceSelectMode. The smoke object will turn white. This mode lets you modify settings for each face of a mesh object. The single face of the mesh is automatically selected.

Figure 20-8. PaintButtons

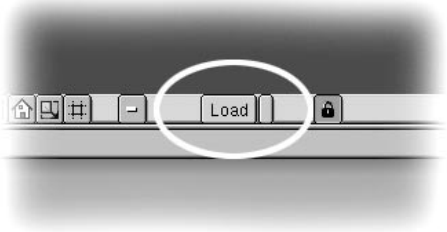


With your mouse in the ButtonsWindow at the bottom of the screen, change the view from EditButtons to the PaintFaceButtons by selecting the PaintFaceButtons icon (Figure 20-8). Select the "Tex", "Halo", and "Alpha" buttons, and de-select the "Collision" button (Figure 20-9). This tells Blender to display a texture on the face, to rotate the face toward the camera, to use the alpha channel of the texture for transparency, and not to calculate collisions with this face.

Figure 20-9. Face settings for the smoke plane

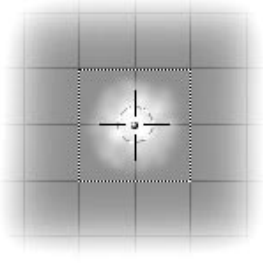


Apply a smoke texture to the face by clicking **LMB** the "Load" button in the header of the Image Window, navigating to **`Tutorials/SquishBunny/texture/smoke.tga`**, and click with **MMB** on it.



You will see the texture appear in the ImageWindow. It is completely white, but the transparency in the alpha channel will make it look like a puff of smoke or steam, as you can see in the 3DWindow (Figure 20-10). Press **FKEY** to exit FaceSelectMode.

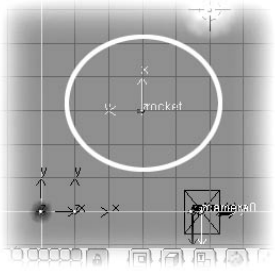
Figure 20-10. Smoke texture applied to the plane



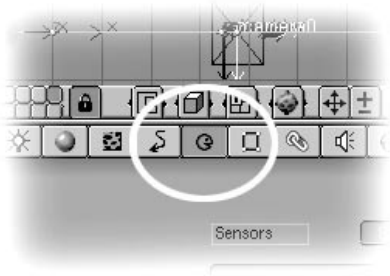
20.5. Adding game logic to the smoke

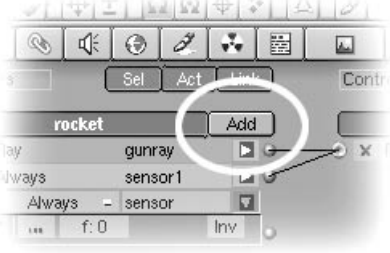
You now have a smoke puff object. The next step is to create the pieces of logic which tells Blender how to use this object. With your mouse in the 3DWindow, press **SHIFT-0** to turn on layer 10, where the rocket object is located. You will see a very small object labeled rocket. Select it with **RMB** (Figure 20-11).

Figure 20-11. Selecting the rocket object



Change the view in the ButtonsWindow from PaintButtons to RealtimeButtons by selecting the RealtimeButtons icon in the ButtonsWindow Header. The RealtimeButtons display the game logic for the rocket.

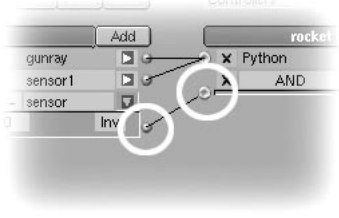




Click with **LMB** on the "Add" button at the top of the Sensors column to add a sensor. Add a Controller and an Actuator in a similar way. Be sure to add the Actuator by using the "Add" button beside the rocket button, and not the one for the "hiteffect" object.

Connect the newly-created LogicBricks together in a chain by holding **LMB** and dragging the mouse from the ball of the sensor to the doughnut of the controller, and from the ball of the controller to the doughnut of the actuator (Figure 20-12).

Figure 20-12. Connecting LogicBricks



Change the type of actuator from "Motion" to "Edit Object" by selecting it from the MenuButton at the top of the brick (Figure 20-13). In the OB: button, type the name of the object to be added (which is "smoke" in this case). Change the value

in the "Time" button to "32" (Figure 20-14). This step tells Blender that each smoke puff added to the scene by this actuator should be removed from the scene after 32 cycles of the game engine. In other words, each smoke puff lives for 32 game cycles. The game engine makes 50 cycles per second, so our smoke puffs live about 2/3 of a second. We could give them a longer life, but having a lot of additional objects in the scene simultaneously can cause a big decrease in speed. You might be thinking that the number 32 still sounds a bit arbitrary. 32 was chosen because it fits the animation we will apply to the smoke in the next section.

Figure 20-13. Changing the Actuator type

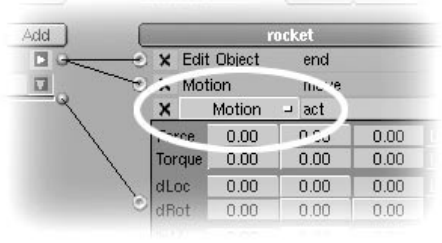


Figure 20-14. Settings in the "Edit Object" Actuator

After seeing the size of the rocket in comparison to the size of the smoke object, it might be apparent that this smoke object is too big. Select the smoke object again then press **SKEY** to begin changing its size. Continue reducing the size of the smoke object by moving your mouse toward the center of the object until it is about one fourth of its original size. As in RotationMode, you can monitor the exact value by watching the Header of the 3DWindow.

Objects must be on a hidden layer in order for the "Add Object" Actuator to add them to the scene. Press **MKEY** to move the smoke object to a different layer. You will see a grid representing the available layers.

Press **0KEY** to specify that it should be moved to layer 10, and press **ENTER**. Hide layer 10 again by pressing **SHIFT-0**.

You don't have to return to the full-screen window to test the smoke trails. You can run the game engine in this screen to see the results without viewing the title screens again. Press **PAD_0** in the 3DWindow to view the scene through the camera (CameraView), then press **PKEY** as before to start the game engine. You should see a trail of white smoke when you fire a rocket.

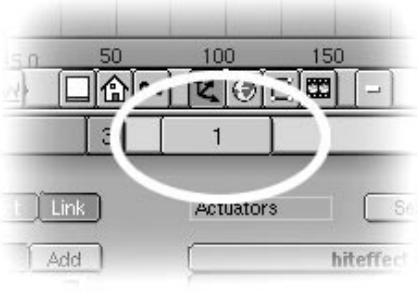
20.6. Animating the Smoke

The smoke puffs will look a lot better if they expand and fade away rather than simply sitting still until they vanish suddenly.

Figure 20-15. Changing the window type to IpoWindow

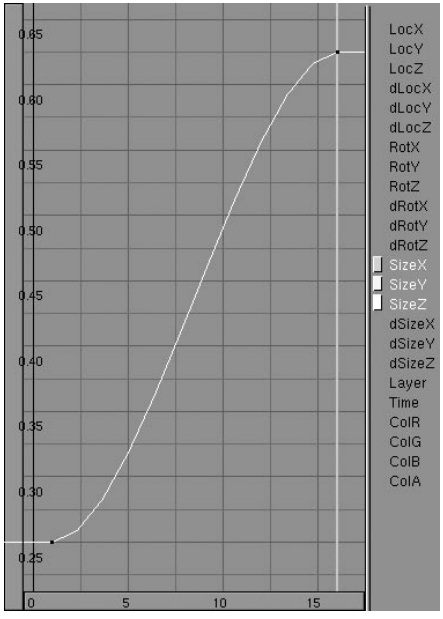
Exit the game, return to a wire frame view, turn on layer 10 again, and select the smoke object (if it's not still selected). Change the ImageWindow to an IpoWindow by moving your mouse into the ImageWindow and pressing **SHIFT-F6** or selecting the IpoWindow icon from the WindowType drop down menu. This menu is accessed by clicking on the WindowType icon at the far left side of the WindowHeader (Figure 20-15). If you don't see the WindowType icon, you may need to scroll the header by holding **MMB** and dragging it to the right.

Figure 20-16. The CurrentFrame Button



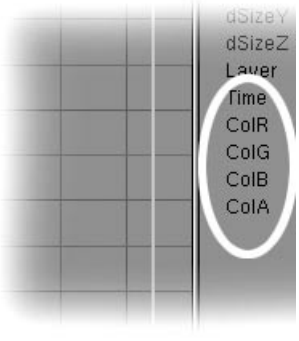
We will animate the smoke object's size by specifying key sizes at certain points in the object's timeline. Blender will plot a smooth transition between these values (called keys or *keyframes*), and this transition will be visualized as an IpoCurve. The IpoWindow will allow you to see and edit the IpoCurves which describe how the smoke object changes over time. The vertical axis of the IpoWindow represents the value being animated. The horizontal axis represents time in animation frames, each of which is equal to 1/25th of a second.

Figure 20-17. IpoCurves for the size of the smoke object



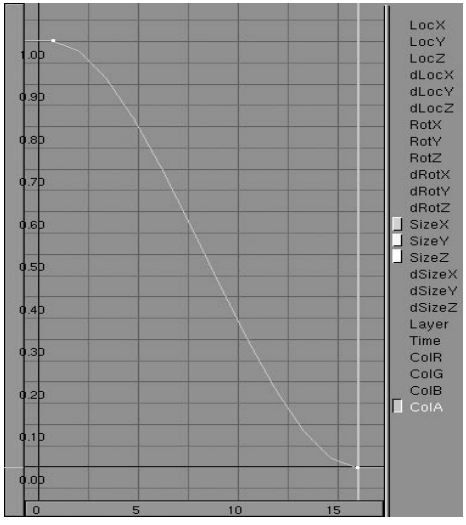
Look at the CurrentFrame Button in the header of the ButtonsWindow (Figure 20-16) to be sure the current frame is frame 1. If it is not, pressing **SHIFT-LEFTARROW** will cause Blender to jump to frame 1. In the 3DWindow press **IKEY** to call the "InsertKey" menu, and select "Size". Press **RIGHTARROW** until the Current Frame Button shows that you are on frame 16. Using the same technique you used before, increase the size of the smoke object to about 2½ times its current size. Insert another size key as before. As you insert these keys, you can see curves being plotted in the IpoWindow. To frame the IpoCurves so that they are easier to see, press **HOME** with your mouse in the IpoWindow (Figure 20-17).

Figure 20-18. Object color channels



We will animate the color and opacity of the smoke object by drawing IpoCurves for the object color channels directly into the IpoWindow. In order for these curves to take effect, we must return to FaceSelectMode (**FKEY**) and select one additional setting for the face in the FacePaintButtons. Activate the button labeled "ObColor". When you select this button you will see the smoke turn black because there are no values yet for the object color.

Figure 20-19. Object color Ipos



Leave FaceSelectMode and return your attention to the IpoWindow. The labels "ColR", "ColG", "ColB", and "ColA" refer to the red, green, blue, and alpha object color channels, respectively (Figure 20-18). Useful values for these channels will are in the range from 0.0 to 1.0. Select the "ColA" channel by clicking with the **LMB** on it. It will turn white. Add the first key with **CTRL-LMB** in the IpoWindow at frame 1.0 and value 1.0, and the second key by repeating the procedure at frame 16 and value 0.0 (Figure 20-19).

In a similar fashion, select each of the "ColR", "ColG", and "ColB" channels in turn, and to each one add a key at frame 1.0 with a value of about 0.8, giving the smoke a light gray color.

Figure 20-20. Ipo Actuator settings

We have created some animation data for the object, and now we must add the logic which will activate the animation. Return the ButtonsWindow to the RealtimeButtons view. In the same manner we used for adding logic to the rocket, add a Sensor, Actuator, and Controller to the smoke object and connect them together as before. This time we will change the Actuator type to "Ipo". Set the "Sta" (starting frame) button to 1 and the "End" (end frame) button to 16, because that is the frame range covered by our animation (Figure 20-20).

Remember to hide Layer 10 before switching to CameraView again and starting the game engine to see the results of your work.

Try animating the "ColR", "ColG", and "ColB" channels in interesting ways. The file **`Demos/Squish-the-Bunny.blend`** shows an example of a completed smoke effect. It also demonstrates how an explosion effect can be achieved. The explosion effect uses a complex mixture of Python programming, animation, and the LogicBricks in Blender's graphical game logic editor. Dissection of this file is left as an exercise for the more adventurous readers.


001
003
2.03beta
004
2.04
game Blender
005
010
2.10
011
2.12
blenderGameKit
blender
POWER
CREATE
A
B
C
D
2.20
022
023
2.23

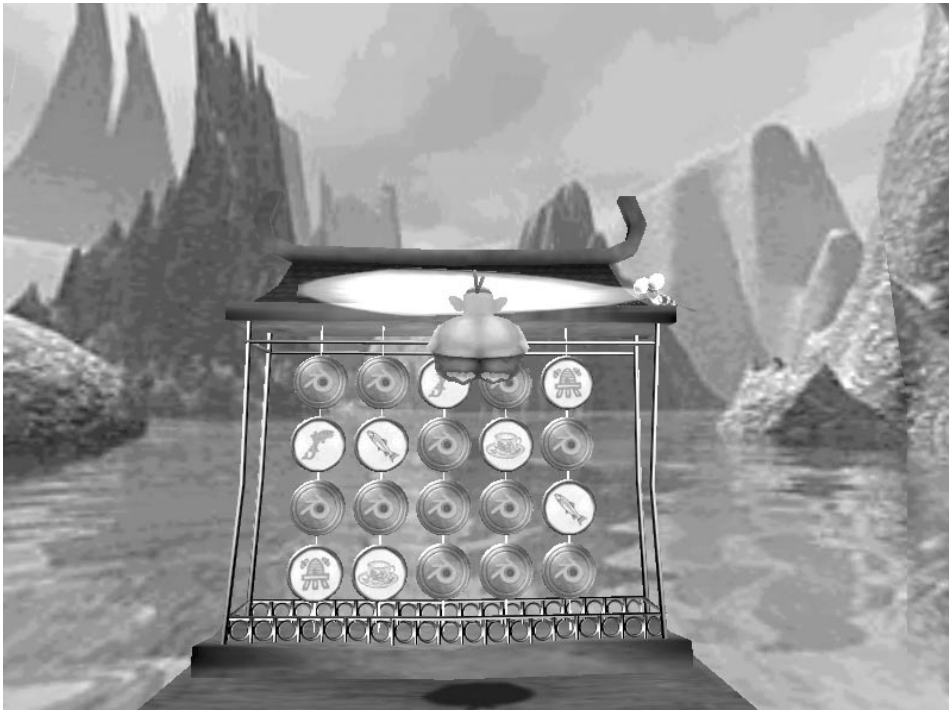

_part V ::
advanced
tutorials

The advanced tutorials require a deeper understanding of the techniques behind the scenes of Blender. For example, some basic knowledge of a programming language or character animation is strongly recommended. The character animation tutorial teaches you how to use the powerful tools of Blender and also gives some useful tips. However, to create natural movements you will need to practice and observe nature as well.

As you found in the intermediate tutorials Blender scenes are provided to give a framework for your experiments. Two of the scenes are puzzle type games and contain complex python scripts.

Chapter 21. Flying Buddha Memory Game

Figure 21-1. Buddha in action



„Flying Buddha“ is a game designed by Freid Lachnowicz (artwork, models, textures) and me (Carsten Wartmann , Python and game logic). The goal of the game is to find pairs of gongs, like in the good old „Memory“ game. Besides that it also includes some action elements, like the dragonfly which will become angry (note the indicator on the top-right) if you jump too high. Also it requires some good timing for the controls of the Buddha. The Buddha‘s goal, reaching „Zen“ is completed when he has found all pairs. For competitions the time needed to solve will be displayed on screen.

Table 21-1. Flying Buddha game controls

Controls	Description
CURSOR KEYS	Movement, you can steer in the air and slow down your fall
SKEY	Select a gong

Load the game **`Demos/FlyingBuddha.blend`** from the CD and have fun playing it!

21.1. Accessing game objects

Accessing individual objects from Blenders game engine is not trivial. In the "FlyingBuddha" game, I needed the possibility of randomly shuffling the gongs at game start.

Generally speaking, it is a good thing to have as much game logic as possible contained on the object that needs that logic. This helps when re-using this object in the same or even different (Blender) scenes. So my first idea was to let the gongs choose a new position themselves. But this soon turned out to be too complicated, because of synchronizing problems and complex logic in the gongs.

I then decided to use an object to control the shuffling process, and have the information about the other objects including their positions is gathered by a Near Sensor. This approach has many advantages. For example we can filter out whole groups of objects with the "Property:" field, we are not bound to a fixed number of objects etc.

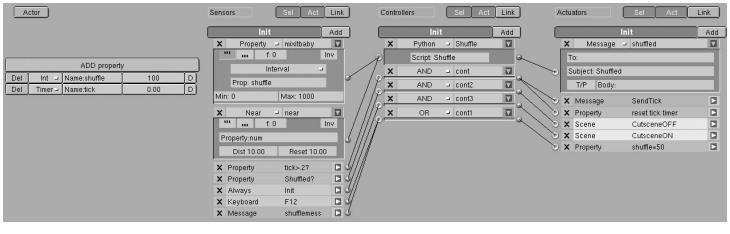
21.1.1. LogicBricks

Load the file **`Tutorials/Buddha/FlyingBuddha_simple.blend`** from the CD. It contains a simpler version of the "Flying Buddha" game, which does not contain all the intro scenes but is fully functional (press **CTRL-LEFT** for a full screen view). Use this file to explore the GameLogic.

The difference in the full game and this tutorial file are build-in debugging and testing logic. Most notably is that you can test the re-shuffling of the gongs every time by pressing **SPACE**. The logic for this is on the "Flamethrower" object (on layer 4).

Have a look at Figure 21-2 the interesting parts in this context are the Sensors "mixitbaby" and "near". Both are connected to a Python Controller, which is then connected to a Message Actuator. Also note the "shuffle" Property, this controls the number of swapped pairs. The other Bricks are not related to the shuffling, they are needed for other game parts.

Figure 21-2. LogicBricks for shuffling the gongs



As you can see, the Near Sensor only looks for objects carrying a Property with the name "num". Also make sure that the "Dist:" setting is high enough for the Near Sensor to cover all objects.

The Python Controller will be called by the Property Sensor as long the Property "num" is in the range from 0 to 1000.

21.1.2. Shuffle Python script

Open a TextWindow (**SHIFT-F11**, see Section 28.1) and choose the script "Shuffle" with the MenuButton.

Figure 21-3. Script to shuffle the gongs

```
 1 # Shuffle script, swaps positions of two gongs
 2
 3 import GameLogic
 4
 5 def ranint(min,max):
 6 return(int(GameLogic.getRandomFloat()*(max+1-min)+min))
 7
 8 contr = GameLogic.getCurrentController()
 9 owner = contr.getOwner()
10 key   = contr.getSensor("mixitbaby")
11 near  = contr.getSensor("near")
12 mess  = contr.getActuator("shuffled")
13
14 # collects all gongs
15 objs=near.getHitObjectList()
16
17 owner.shuffle = owner.shuffle - 1
18 if owner.shuffle<0:
19        GameLogic.addActiveActuator(mess,1)
20 else:
21        g1 = ranint(0,19)
22        g2 = ranint(0,19)
23
24        pos1 = objs[g1].getPosition()
25        pos2 = objs[g2].getPosition()
26        objs[g1].setPosition(pos2)
27        objs[g2].setPosition(pos1)
```

So lets have a look into the script. The lines 1 to 12 contain the usual initialization, and getting information about the Controller, Sensors, Actuators and the owner which is needed to access Properties. The definition of a new function in line 5 is used to make a random function which returns an integer number in a specified range. This will save much typing later.

In line 15 the first important step is done, using the method **`getHitObjectList()`** of the "near" object we collect all game objects with the Near Sensor into the list **`objs`**.

In line 17 we decrement the Property "shuffle" by one.

The if-block beginning in line 18 executes the Message Sensor connected to the Python Controller if the Property shuffle is less then zero, the message can then be used to start the game.

The else-block is executed when owner.shuffle is bigger than zero. This means that gongs need to be swapped.

In lines 21-22 we get two random numbers into the variables g1 and g2. The numbers will be in a range from 0 to 19 because we have 4x5=20 gongs. g1 and g2 are the indices of the gongs we want to swap in the next lines. Note that lists in python start with the element "0".

In the lines 24-25 the script reads the positions of the gong-objects using the random indices. The method used here is **`getPosition()`**. You can insert a "print pos1,pos2" statement after line 25 to actually see the gong positions while running the game.

Info: *Python is an autodocumenting language. Use a "print dir(object)" statement to find out what methods an object provides.*

The final two lines then swap the positions of the two gongs. The first obj indexed as objs[g1] is set to **`pos2`** which is the position of the first gong. Same for the other gong. You can see the shuffling process in the game itself by looking at the gongs from the backside.

In this tutorial I showed you how to use Python in the game engine to access and change objects in a scene. We used this approach to keep the game logic local on the objects. If you are used to non object-oriented programming languages or systems, this may appear strange to you at first .But this approach has many advantages. or example, you don't need to change the logic while editing your scenes or adding objects, the script will even work when adding objects while running the game. Also, re-using the logic in other scenes is much easier this way.

Chapter 22. Game Character Animation using Armatures by Reevan McKay

Figure 22-1. Character animation in the game engine

The new armature system opens up new possibilities for character animation in the Blender game engine, but can be somewhat intimidating for new users. This tutorial guides you through the steps involved in building an armature and creating actions that can be used for smooth character animation in the game engine. Check the file `Demos/ePolice.blend` for a decent example using the character from this tutorial.

22.1. Preparing the Mesh

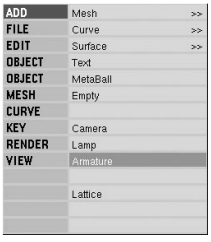
This tutorial assumes you have already modeled a character that you want to use in an animation. Due to the high cost of calculating skeletal deformation, you will get better performance by using fewer vertices in your meshes. It pays to spend some time to optimize your models before continuing. You can use the file `Tutorials/CharacterAnimation/animationtut.blend` as base for this tutorial.

Many aspects of blender's game and animation engines depend on the fact that you have modeled your character and armature using the correct coordinate system. The FrontView (**PAD1**) should show the front of your character and armature. If this is not the case, you should rotate your mesh so that the front of the character is displayed in the FrontView, and apply the rotations as described in the next step.

Before adding bones and animating a mesh, it is a good idea to make sure that the base mesh object does not have any rotation or scaling on it. The easiest way to do this is to select the mesh and apply any existing transformations with **CTRL-A**->"Apply size/rot".

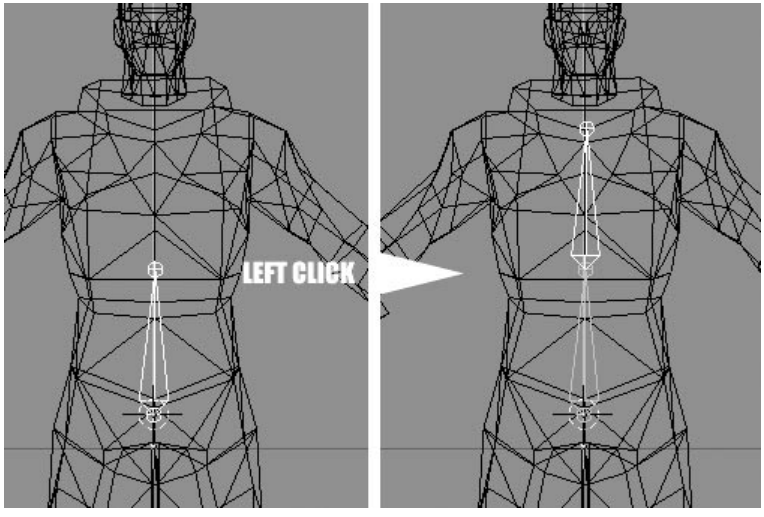
22.2. Working with Bones

The next step is to build the skeleton or "armature" that will be used to deform the character. A single armature object can contain an entire hierarchy of bones, which makes editing animations easier. Add an armature object from the toolbox by pressing **SPACE**->"ADD->Armature".



You will see a yellow bone appear. You can reposition its endpoint by moving the mouse. When you are more or less satisfied with its position (you can still edit it later), **LMB** to finalize the bone. At this point a new yellow bone will appear, attached to the end of the first bone. You can continue to add connected bones in this fashion (Figure 22-2). If you do not want to create another bone, you can press **ESC** to cancel the current (yellow) bone. The bones you added previously will not be affected.

Figure 22-2. Adding bones



Armatures have an EditMode similar to meshes. You can determine if you are in EditMode or not by looking at the EditMode icon in the 3DWindow Header. As with meshes, you can toggle in and out of EditMode using **TAB**. While you are in EditMode, you can add and remove bones, or adjust the rest position of existing bones. The rest position of your armature should correspond to the untransformed position of the mesh you want to deform, and you should build the armature inside

the mesh, like a skeleton inside a human body.

While you are in EditMode, you can reposition a bone by selecting one or more of its endpoints and using the standard transformation tools such as scaling (**SKEY**), rotation (**RKEY**) and translation (**GKEY**).

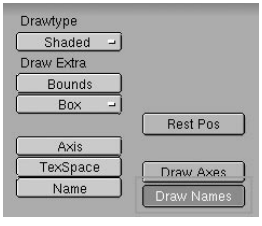
You can also extrude selected points to form new bones with **EKEY**.

One or more bones can be selected by selecting their start and end points. Like meshes, you can select all of the bone points within a region by using **BKEY** or you can select or deselect all bones in an armature with **AKEY**. You can also select an entire bone chain at once by moving the mouse over any one of the chain's points and pressing **LKEY**. Selected bones can be deleted with **XKEY** or duplicated with **SHIFT-D**.

22.3. Creating Hierarchy and Setting Rest Positions

22.3.1. Naming Bones

It is a good idea to give meaningful names to the bones in your armature. This not only makes it easier to navigate the hierarchy, but if you follow a few simple naming rules, you can take advantage of the pose-flipping features. You can have the bone names displayed on the model by selecting the armature, switching to the EditButtons window (**F9**) and clicking the green "Draw Names" button.



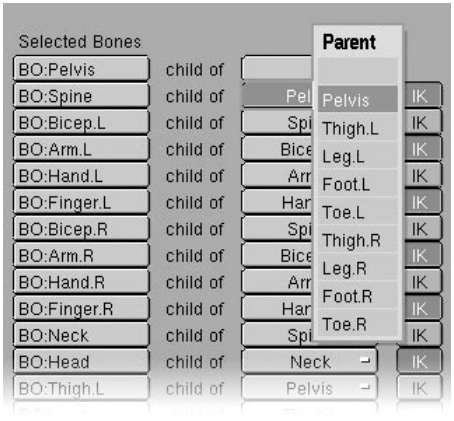
For symmetrical body elements such as arms or legs, it is a good idea to append ".left" or ".right" (or simply ".l" and ".r") suffixes to each part. This information is used when flipping poses. An example of this would be to name the right arm "Arm.Right" and the left one "Arm.Left". Non-paired limbs such as the head or chest do not need any special naming.

When re-using the same action on different armatures, the engine looks at the names of the bones in the armature, and the names of the animation channels in the action. When there is a perfect match (capitalization matters), the animation data in the action will be applied to the appropriate bone. If you want to take advantage of action re-use, make sure that all your skeletons use the same naming convention.

22.3.2. Parenting Bones

To establish parenting relationships within an armature, you must first make sure the armature is in EditMode. Select only the bones you wish to modify or if you prefer, select all bones with **AKEY**) and switch to the EditButtons with **F9**. You will see a list (Figure 22-3) of the selected bones and next to each bone in the list you will see a "child of" label and a pull-down menu. To make a bone the child of another bone, simply select the appropriate parent from the pull-down menu. Note that the menu only contains the names of bones that could be valid parents. This prevents you from accidentally making a loop in parents (such as making an arm the parent of the chest, which should be parent of the arm).

Figure 22-3. Parenting bones in the EditButtons



To clear a parenting relationship, set the "child of" menu to the first (empty) choice in the menu.

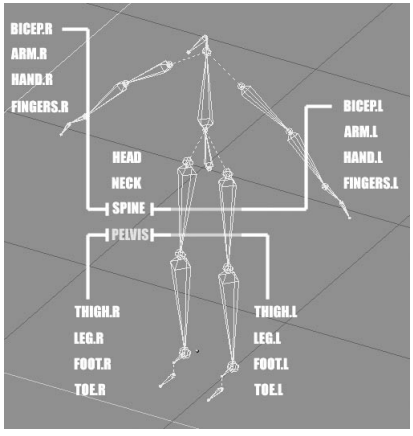
Parenting is much easier if you have already named your bones, though it is not necessary.

Pressing the "IK" button (IK means here "inverse kinematics") next to the parenting menu will ensure that the root of the child is connected to the tip of the parent. This is not so important for game models since the IK solver is not active in the game engine, but it can be a useful way to define a bone "chain" which can be selected with **LKEY**.

22.3.3. Basic Layout

For a typical humanoid character, the following hierarchy is recommended (Figure 22-4). Some characters may benefit from additional bones for elements such as flowing skirts or hair.

Figure 22-4. Typical bone layout for a humanoid character

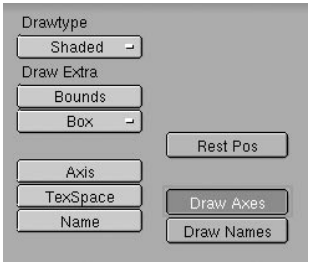


22.3.4. Coordinate System Conventions

Before going on, it is a good idea to clear any rotation or scaling that may have been assigned to the armature. Leave EditMode and with the armature object selected, apply the transformations with **CTRL-A**.

The center point of the armature (represented by a small yellow or purple dot) should be located on the ground, between the character's feet. If this is not the case, enter EditMode for the armature, select all bones with **AKEY** and move the bones so that the center point is at the correct location.

The final step before preparing the mesh for deformation is to ensure that the bones in the armature have consistent orientations. Each bone is like an individual object with its own coordinate system. You can see these coordinate systems by selecting the armature object, switching to the EditButtons with **F9** and clicking on the green "Draw Axes" button.



Generally you want to make sure that the Z-axis for each bone points in a consistent direction. In most cases this means having the Z-axis point upwards. You can adjust the roll angle of a bone by selecting it in EditMode, pressing **NKEY** and adjusting the "roll" field.

If you are going to be re-using actions on different armatures, it is very important that both armatures have their bones oriented in the same way. If this is not the case, you will notice a lot of strange flipping happening when you assign the action.

22.4. Establishing Mesh Deformation Vertex Groups

22.4.1. Creating Groups

Once your armature is established, it is time to specify which bones will affect which vertices of the mesh. This is done using vertex groups. To access the vertex grouping features, select the mesh you will be deforming and enter EditMode. Switch to the EditButtons and find the „Group“ column (Figure 22-5). Normally you will have one vertex group for each bone in the armature. A vertex can belong to more than one group, which is how smooth deformation is achieved. In order to be recognized by the armature, the vertex groups must have exactly the same names as the bones they are associated with (capitalization matters). To create a new vertex group, click on the “NEW” button and edit the name in the text button that will appear.

Figure 22-5. Group buttons in the EditButtons

You can see a list of all of the current deformation groups by clicking on the menu next to the group name button (Figure 22-6). Selecting an item from this menu changes the active deformation group.

Figure 22-6.

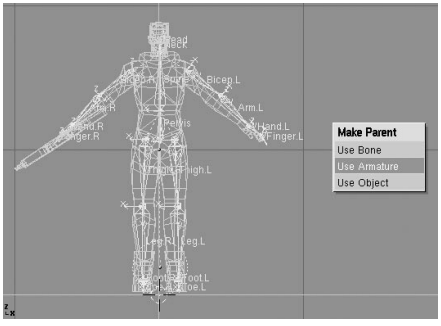
You can assign vertices to the currently active deformation group by selecting vertices and clicking the “Assign” button. The selected vertices will be assigned to the active group with the weight specified in the “Weight” slider. You can remove vertices from the current deformation group by selecting them and clicking the “Remove” button.

Create vertex groups for all of the bones in your armature (making sure the names of the groups and bones match) and assign vertices to the appropriate groups. Make sure that every vertex is affected by at least one bone. For this “first pass” of the deformation process, try to keep things simple by leaving the weight set to “1.000” and avoid having vertices being assigned to more than one group.

22.4.2. Attaching the Mesh to the Armature

At this point, you are ready to attach the mesh to the armature. Make sure that the mesh and the armature are correctly lined up and that you are not in EditMode (Figure 22-7). Select the mesh first and while holding SHIFT, select the armature and press CTRL-P->”Use Armature”.

Figure 22-7. Lined up Mesh and Armature



22.4.3. Testing the Skinning

Once you have attached the mesh to the armature, you are ready to start testing the deformation. It often takes a fair amount of tweaking to get satisfying results. You want to avoid the excessive pinching or stretching that can occur when vertices are not assigned to the right bones. We’ll spend more time on that later. For now, we’ll take a look at how to pose the armature, which is a skill needed for testing the deformation.

22.4.4. PoseMode

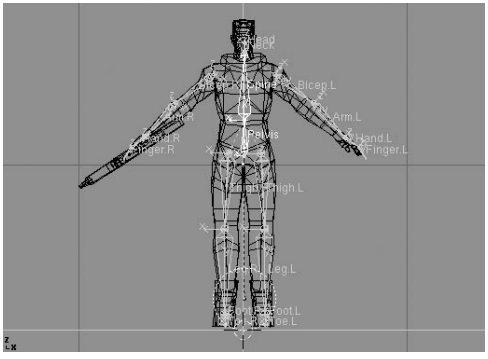
In addition to EditMode, armatures have a PoseMode. This is used to position bones within the armature. Setting keyframes in PoseMode defines an „action“ for the

armature, and the game engine will use these actions to animate the character.

Note: *Note that only transformations performed in PoseMode will be incorporated into the action (and therefore the game engine). Rotations, scalings and translations performed in ObjectMode cannot be recorded in actions.*

You can toggle in and out of PoseMode by selecting an armature and pressing **CTRL-TAB** or by clicking on the PoseMode icon in the 3DWindow Header bar. When in PoseMode, the armature will be drawn in blue, with selected bones drawn in a lighter shade.

Figure 22-8. Armature in PoseMode

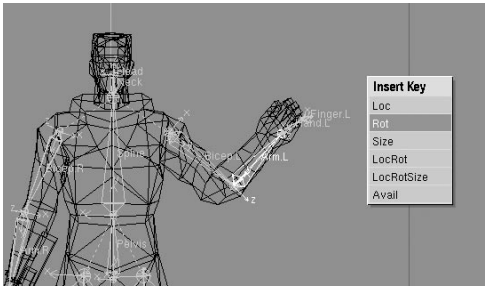


To manipulate bones in PoseMode, select bones by using **RMB** on them and use the standard transformation keys for scaling, rotation and translation. Note that you cannot add or remove bones in PoseMode, and you cannot edit the armature's hierarchy.

At any time, you can clear the pose you have made and return to the armature's rest position by clearing the rotation, scaling and translation components using **ALT-R**, **ALT-S** and **ALT-G** respectively.

You can set keyframes for a pose by selecting one or more bones and pressing **IKEY**. and choosing one of the transformation channels to key from the pop up menu.

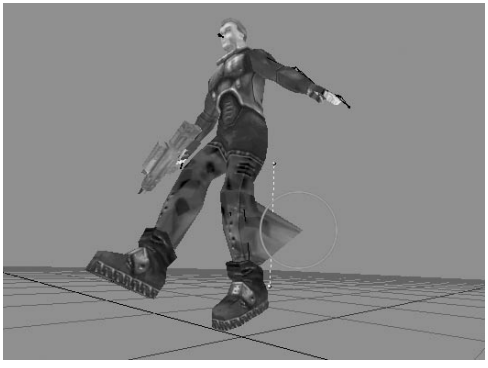
Figure 22-9. Selected "Arm.L" bone



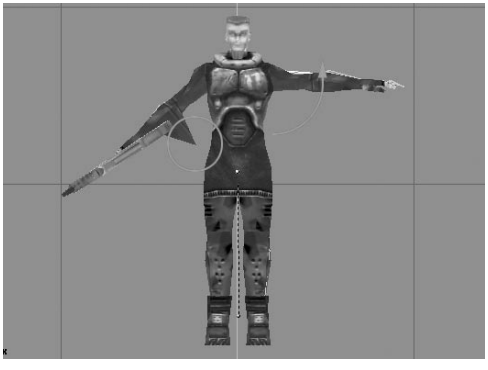
22.5. Weight Editing

In PoseMode, manipulate the limbs of the armature through their typical range of motion and watch carefully how the mesh deforms. You should watch out for deformation artifacts caused by any of the following:

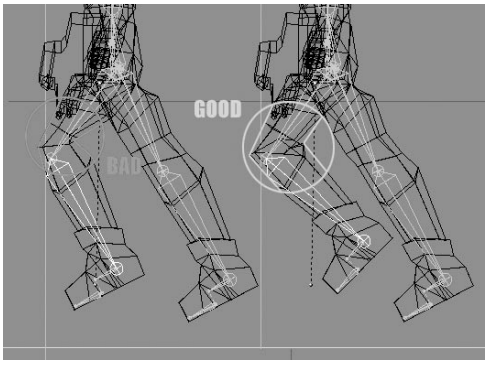
- Vertices that are not assigned to any bones can be easily detected by moving the root of the character's hierarchy (usually the hips or pelvis) and seeing if any vertices are left behind.



- Vertices that are not connected to the correct bones. If you move a limb (such as the arm) and notice vertex "spikes" protruding from other parts on the body, you will have to enter EditMode for the mesh and remove the offending vertices from the vertex group.



- Pinching or creasing caused by inappropriate vertex weighting. This effect is most visible in the joints of limbs such as arms and legs. Often it is a symptom of vertices that are members of too many groups. The easiest way to fix this is to use the weight painting tool.

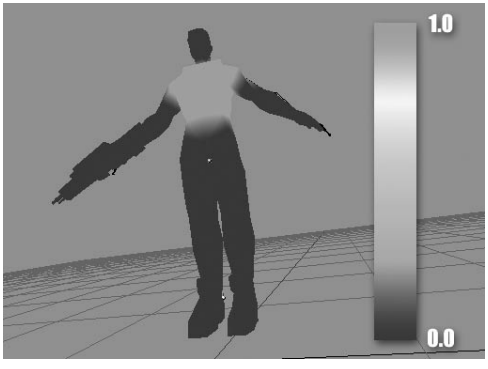


To adjust vertex weights, you have the choice of manually assigning weights using the method outlined above, or you can use the weight painting tool.

This feature lets you "paint" bone influence onto the mesh and see the resulting deformation in real-time. Make sure you are in wireframe or untextured mode with **ZKEY** or **SHIFT-Z**. Access weight painting mode by selecting the mesh and clicking on the weight-paint icon in the 3DWindow Header.

In weight paint mode the mesh is displayed with a "false color" intensity spectrum similar to the view from an infrared camera (Figure 22-10). Blue areas have little or no influence from the current deformation group while red areas have full influence. As you change the active deformation group in the editbuttons, you will see the coloring on the model change.

Figure 22-10. Weight painted character



Painting weights onto the model works somewhat similarly to vertex painting. **LMB** paints onto the area beneath the cursor. Pressing **UKEY** undoes the last painting operation. The cursor size and opacity settings in the VertexPaintButtons are used to determine your brush settings and the "Weight" field in the EditButtons is used to determine the "color" you are using (0.000 is the blue end of the spectrum and 1.000 is red).

To remove weight from a group of vertices, set the vertex weight to 0.000 and paint over the area. Note that you do not need to have the mesh in EditMode to change the active deformation group, or to change the weight.

22.6. Animation

Animation is very important for conveying the impression of life in a game world, and for giving the player an immediate reward for his or her decisions. Game characters typically have a set of animated actions they can perform, such as walking, running, crawling, making attacks or suffering damage. With the armature system, you can design each animation in a separate action and define which actions play at which times using LogicBricks.

Note that at the time of writing, animation constraints (including the IK Solver) do not work in the game engine. This means that game animation must be done using forward kinematics exclusively.

22.6.1. Multiple Actions and Fake Users

If you want to create multiple actions in a blender file, remember that only one action can be assigned to an armature at a time. If the other actions do not have any users, they will not be saved when the file is saved. To prevent additional actions from disappearing, you can add fake users .

To create a fake user for an action, press **SHIFT-F4**. This lets you browse the various objects and data blocks in the blender file. You may need to click on the “P” button once or twice to find the root of the file’s structure, which should look like Figure 22-11. From there, descend into the Action directory, and select the actions you want to protect with **RMB**. Pressing **FKEY** will add a fake user to the selected items (indicated by the capital “F” that appears next to the action name), preventing them from being accidentally removed from the file.

Figure 22-11. Structure of the file in the DataSelectWindow

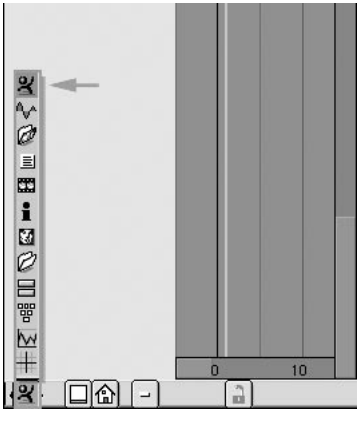
22.6.2. Creating an Idle Cycle

The simplest action to create for a character is the „idle“ or „rest“ position. This action can be played when the character is not doing any other action. A good idle animation gives the game character the illusion of life by keeping it moving even when the player is not actively issuing any control commands.

Since the character is not moving through the level while playing the idle animation, we don't have to worry about synching the animation with the physics system.

To create a new action, split the view by clicking with **MMB** on one of the window borders, selecting "Split Area", and **LMB** to set where the split will appear. Change the type of the newly created window to ActionWindow by **LMB** on the WindowType icon in the Header and choosing the topmost icon (Figure 22-12).

Figure 22-12. Switch to ActionWindow



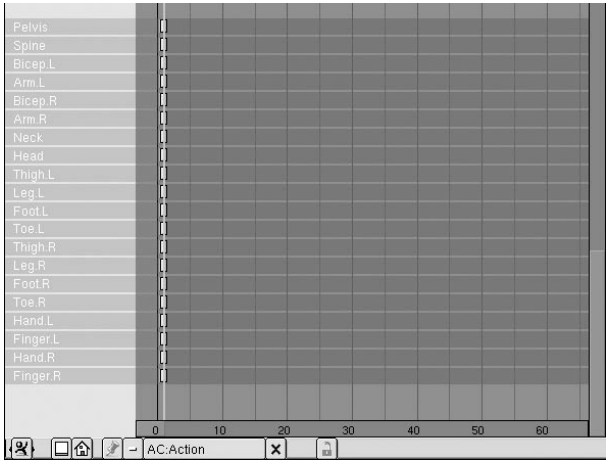
Go to a 3DWindow and select the armature you wish to animate. Enter PoseMode and make sure that Blender is on frame 1 by changing the number in the frame counter button in the header of the ButtonsWindow, or by using **LEFTARROW** and **DOWNARROW**.

You are now ready to make the first frame of the idle animation, using the PoseMode techniques described earlier. What this pose looks like will depend largely on the personality of your character and the type of game you are making. Consider which actions might immediately follow the rest position. If the character is supposed to be able to fire a weapon quickly, the rest position might involve holding the weapon in a ready-to-fire stance. A less fast-paced game might have the character adopt a more relaxed "at-ease" pose.

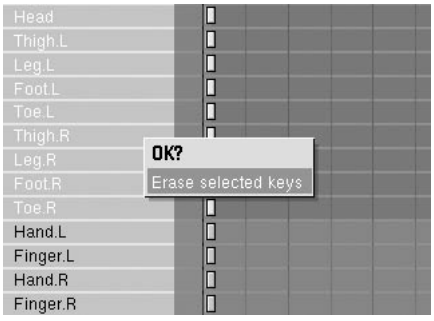
When you are satisfied with the first frame, select all of the bones in PoseMode and insert a rotation key by pressing **IKEY**->"Rot". Next, deselect all bones and select only the character's root bone (usually the pelvis or hips) and insert a "Loc" key. Normally only the root bone gets location keys, while all bones get rotation keys.

When you insert keys, you should notice that new channels appear in the action window (Figure 22-13). The yellow rectangles represent selected keyframes and grey rectangles represent unselected keyframes. You can move keyframes in the action window by selecting them and grabbing them with **GKEY**.

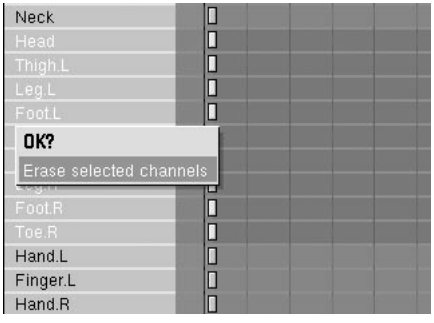
Figure 22-13. Keys in the ActionWindow



Keyframes can be deleted by selecting them and pressing **XKEY**->"Erase selected keys".



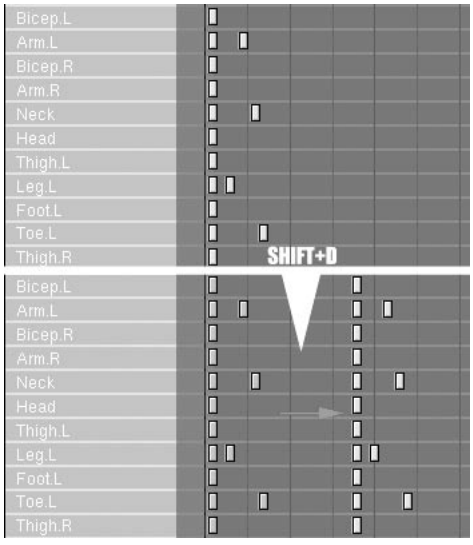
You can erase an entire action channel (all keyframes for a particular bone), by selecting one or more action channels by **SHIFT-RMB** on the channel names in the column at the left. Selected channels are displayed in blue, while unselected channels are displayed in red. Pressing **XKEY**->"Erase selected channels" with the mouse over the channel list deletes the selected channels.



In order to create an action that loops smoothly, you will need to copy the first frame and duplicate it as the last frame of the animation. There are two main ways of doing this.

- The first way is to select all of the bones in PoseMode and click on the "Copy Pose" button. This will copy the transformations from the selected bones into a temporary buffer. You can paste the pose at another keyframe by changing the frame and clicking the "Paste Pose" button. Note that this doesn't necessarily set keyframes at the new point in time, unless you have activated the KeyAC option in the info window. If you have not activated KeyAC and you want to make keyframes after pasting the pose, you can press **IKEY**-"Avail".

- The second way to copy a block of keyframes is even easier. In the ActionWindow, select the vertical column of keyframes for all channels (hint: use **BKEY** to select with a bounding rectangle). Pressing **SHIFT-D** will duplicate the keyframes. You can move the block to a new point in the timeline and drop them by **LMB**. To ensure that the keyframes stay on whole frame increments, hold down **CTRL** while dragging.

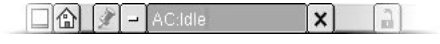


Idle animations tend to be fairly long, since the motion involved is typically subtle and shouldn't be seen to loop too often. The last frame should be at least 100 or higher.

While animating, you can "scrub" through the animation by holding the left mouse button and dragging the mouse in the action window. This will move the position of the green "current frame" indicator. In this way you can test parts of the animation to make sure they play smoothly. You can also play the whole animation by moving to the first frame and pressing **ALT-A** with the mouse over a 3DWindow. To see the animation in a loop, set the "Sta" and "End" values in the Display Buttons (**F10**) window to match the start and end frames of your loop.

At this point you can go back in and add additional key frames between the start and end. Remember to keep the motion reasonably subtle so that the player doesn't notice the repetitive nature of the action. Good elements to add are breathing effects, having the character adjust its grip on any weapons or equipment, and slight head turns.

When you are satisfied with the action, give it a name by editing the name field in the ActionWindow Header. Also make sure to create a fake user for the action to prevent it from disappearing from the file when you create your next action.



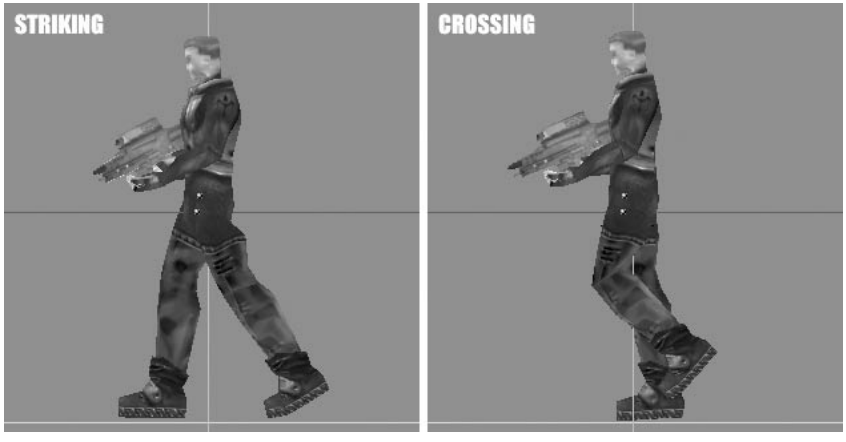
22.6.3. Creating a Walk Cycle

Another very important action is the character's walk cycle. This animation will be used when the character is moving through the level. This animation is somewhat more complicated, since we have to consider how the animation will interact with the physics system.

When creating the walk cycle, it is generally best to animate it in such a way that the character seems to be walking on a treadmill. The forward motion will be provided by the game's physics system at run time.

A walk-cycle actually consists of two steps. One for the left foot and one for the right foot. For each step there are two main key frames: the striking pose and the crossing pose (Figure 22-14). The striking pose represents the moment when one foot has just been planted on the ground and the other is about to be lifted. The crossing pose represents the moment when the two legs cross each other under the character's center of gravity: one foot is on the ground and moving backwards, while the other is lifted and is moving forwards.

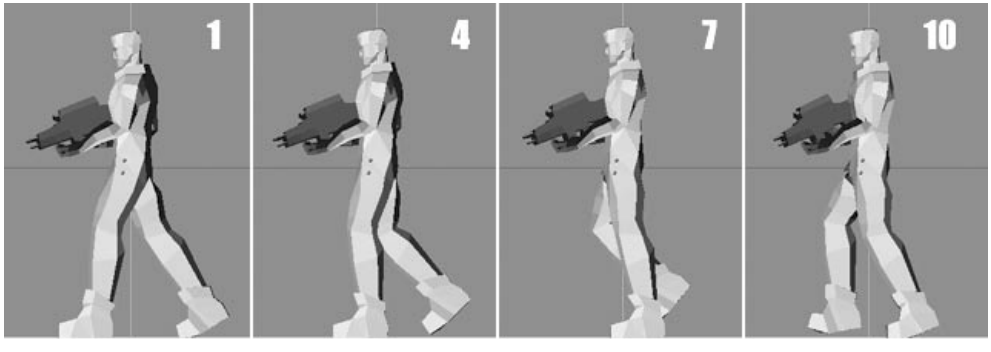
Figure 22-14. Striking and crossing

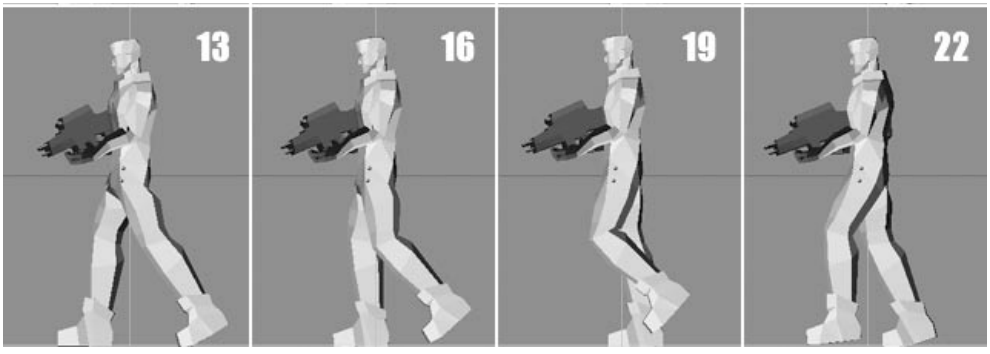


To start creating this animation, switch to the action window and create a new blank action by clicking on the Action menu and choosing "Add New". This will create a copy of any action that may have already been on the character. Name the action and make sure the animation is blank, by moving the mouse over the channel list, selecting all channels with **AKEY** and deleting them with **XKEY**->"Erase selected channels".

For this animation, we'll make a 24 frame walk cycle. We'll set five key frames to get a basic walking motion established. Once that's done you can go back and add additional key frames to smooth out the motion and improve the animation (Figure 22-15).

Figure 22-15. Keyframes for the walkcycle





The first thing to do is to set the striking pose for the left foot. This pose will be copied and pasted as the last frame of the action to ensure the animation loops smoothly. Note that if you later make changes to this first frame, you should copy those changes to the last frame again.

The striking pose has the following characteristics:

- The leading leg is extended and the foot is firmly on the floor.
- The trailing foot has the toes on the floor and the heel has just left the ground.
- The pelvis is lowered, to bring the feet down to the level of the floor.
- If the walk cycle incorporates arm swinging, the arms oppose the legs. If the left leg is advanced, the left arm will be swung back, and vice versa.

When you are satisfied with the pose, insert rotation keyframes for all bones, and insert an additional location keyframe for the pelvis bone. Copy this pose to the end of the animation loop, which will be frame 25. Frame 25 will not actually be played however; we will end the loop at frame 24 when playing. Since frame 25 is a duplicate of frame 1, the animation should play back seamlessly.

If you built the character's armature using the naming conventions and coordinate systems recommended earlier in the tutorial, you can take advantage of the character's axial symmetry by copying the striking pose and pasting it flipped. To do this, go to the first frame, and select all bones in PoseMode. Click the "Copy Pose" button and set the active frame to the middle of the animation (in this case, frame 13). To paste the pose, click the "Paste Flipped" button.

Set "Avail" keyframes for the appropriate bones. Note that if your animation does not incorporate arm swinging (for example if the character is carrying a weapon), you might choose to only select the pelvis and legs when copying and pasting the pose. Otherwise, the character will seem to switch the weapon from one hand to the other.

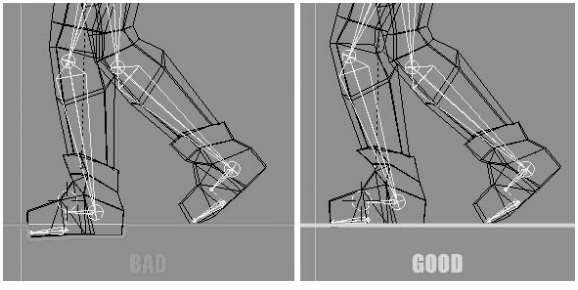
The next task is to create the crossing pose. The first one will occur halfway between the first frame of the animation and the flipped pose you just created (i.e. frame 7). The crossing pose has the following characteristics:

- The planted foot is underneath the character's center of gravity.
- The lifted foot is crossing past the planted leg.
- The pelvis is raised to bring the feet up to the level of the floor.
- If the arms are swinging, the elbows will be crossing each other at this point.

Set "Avail" keyframes for all bones on this frame and copy the pose. Advance the active frame to halfway between the end of the animation and the second striking pose (frame 19) and paste the pose flipped.

At this point test your animation loop. It is a good idea to go in and look at it frame by frame with **LEFTARROW** and **RIGHTARROW**. If you see frames where the feet seem to push through the floor, adjust the height of the pelvis accordingly and set "Loc" keyframes, or adjust the rotation of the bones in the offending leg and set "Rot" keyframes for them (Figure 22-16).

Figure 22-16. Bad positions of the character's legs



If you prefer working with IpoWindows, you can edit action channel Ipos directly, though this is not always required. To do this, select an action channel in the ActionWindow, make a window into an IpoWindow with **SHIFT-F6** and click on the ActionIpo icon in the IpoWindow Header.

- Note that Action Ipos display rotations in quaternions instead of Euler angles. This gives nice predictable results when working with complex combinations of rotations, but can be a bit unusual to work with. The best tactic is to set the action value of the quaternions by inserting "Rot" keyframes in pose mode, and only using the IpoWindow to adjust the way the curves are interpolated.
- To view several different Ipos in different windows, you can use the "pin" icon in the IpoWindow Header buttons to prevent the displayed Ipo from changing when you change object selection.

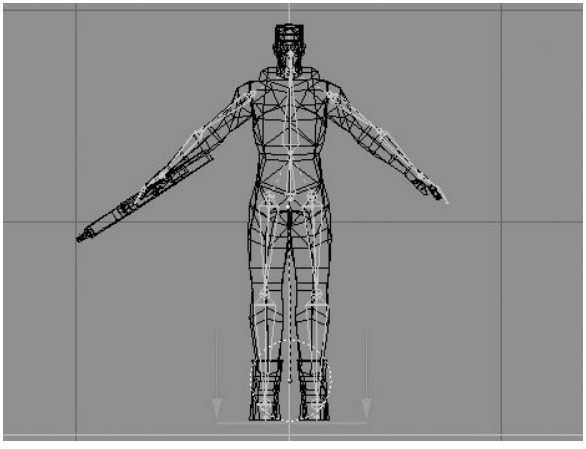
When you are done, make sure to add a fake user for this action to prevent it from getting lost when the file is saved.

22.7. Game Logic

When adding game logic to your character, make sure to put the game logic on the armature object itself, rather than on the mesh.

If you are making your character a dynamic physics object, you may need to adjust the center point of the armature based on the size of the dynamic object to make sure that the character's feet touch the floor. The character's feet should touch the bottom of the dotted dynamic object sphere (Figure 22-17).

Figure 22-17. Dynamic object for the character



Generally speaking, you will need to add an Action Actuator for each different action that your character can perform. For each actuator, you will need to set the start and end frames of the animation, as well as the name of the action.

Since version 2.21, Blender has the ability to create smooth transitions or to blend between different game actions. This makes the motion of game characters appear much more natural and avoids motion "popping". To access this functionality, all you have to do is adjust the "Blending" parameter in the Action actuator. This field specifies how long it will take to blend from the previous action to the current one and is measured in animation frames. A value of "0" indicates that no blending should occur. Note that this blending effect is only visible in the game engine when the game is run (Figure 22-18).

Figure 22-18. Blending Actions

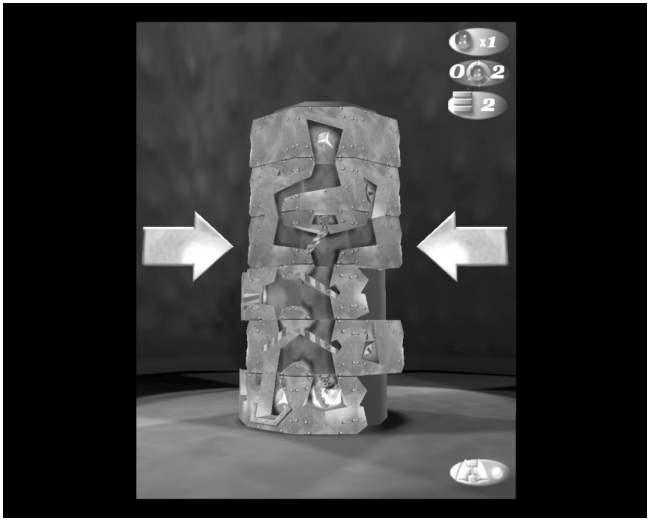
Action actuators have many of the same playback modes as Ipo actuators, including looping, playing, flippers and property- driven playback.

You should try and design your game logic so that only one action is triggered at a time. When multiple actions are active at the same time, the game engine can only display one of them on any given frame. To help resolve ambiguity, you can use the "Priority" field in the action actuator. Actions with lower numbers will override actions with higher numbers.

Chapter 23. Blenderball

Blenderball is made by Joeri Kassenaar, based on an idea by W.P. van Overbruggen. Blenderball is an action puzzle game where you have to guide a ball through a dangerous maze. The gameplay can reach from the original puzzle to a two player game, or just an image puzzle.

Figure 23-1. Blenderball game



In the Blenderball game you have to guide the balls through a maze, avoiding traps and dead ends. The goal is to fill all pockets at the end ring with balls.

Table 23-1. Blenderball controls

Controlls	Description
ARROWLEFT	Turn ring segment left
ARROWRIGHT	Turn ring segment right
ARROWUP	Go one ring segment up
ARROWDOWN	Go one ring segment down
DEL	Turn camera left
PgDn	Turn camera right
AKEY	Toggle automatic camera tracking
CTRL	Track camera new

Table 23-2. Blenderball displays

Display	Description
	Balls left to solve level
	Balls collected / Balls needed to solve level
	Current level
	Auto camera / manual camera indicator

On the CD there are many assets you can use to start your own games based on Blenderball. Look in the folder `Tutorials/Assets/Blenderball/` for commented sources of the Blender scenes, new images and sounds.

V

23.1. Customize the Blenderball image puzzle

Figure 23-2. Blenderball image puzzle

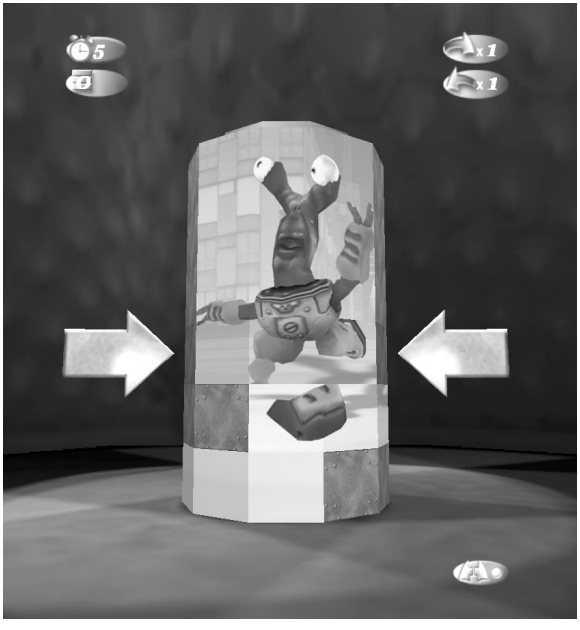


Table 23-3. Blenderball image puzzle controls

Controlls	Description
ARROWLEFT	Turn ring segment left
ARROWRIGHT	Turn ring segment right
ARROWUP	Go one ring segment up
ARROWDOWN	Go one ring segment down
DEL	Turn camera left
PgDn	Turn camera right
AKEY	Toggle automatic camera tracking
CTRL	Track camera new

Table 23-4. Blenderball image puzzle displays

Display	Description
	Time left to solve puzzle
	Current level
	Left turns available
	Right turns available
	Auto camera / manual camera indicator

Load `Tutorials/Blenderball/Blenderballimagegame00.blend`, this scene is ready for changing the images. If you'd like to play a game first, switch to the full screen view by pressing **CTRL-LEFTARROW** and the start the game engine with **PKEY**. After that return to the editing layout by pressing **CTRL-RIGHTARROW**.

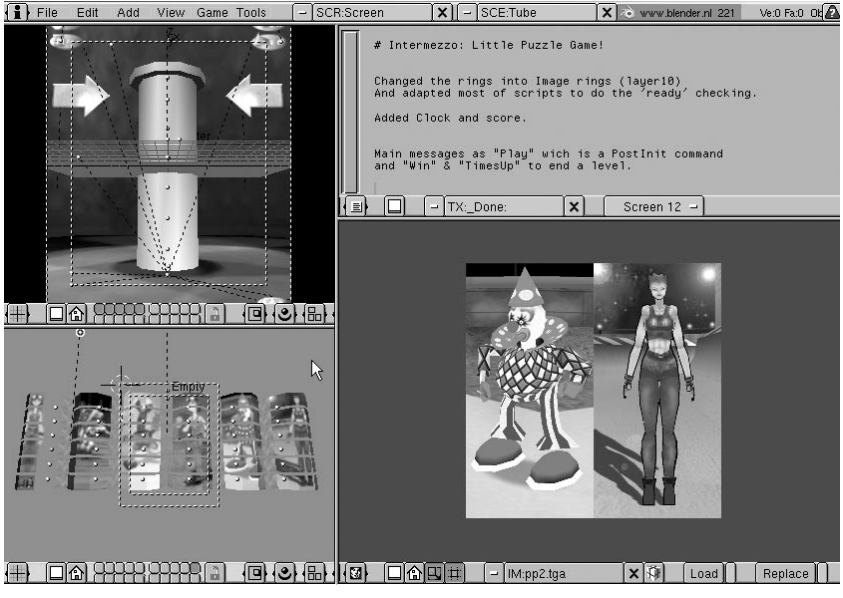
Try to get all rings in a position that the image is shown correctly. Use the cursor keys to select the ring and rotate the rings segment. There is a hard time limit.

Now, prepare the images you want to use in your favorite 2-D image manipulation program. Here are the guidelines to make images that fit into the game:

- Images have to be square, containing two images side by side

- For performance reasons you should not use textures bigger than 512x512 pixels in size
- Blender can use Targa (***.tga**) and JPEG (***.jpg**) images

Figure 23-3. Screen to change images



Select one of the ring segments with **RMB** in the lower left window. Now press **FKEY** to enter FaceSelectMode, the image to the right will update according to your selection. Now press the "Replace" button in the ImageWindow to choose a different image from Disk.

If your image meets the requirements mentioned before, it should now be on the rings and you can play your customized game.

23.2. Changing the levels of the Blenderball game

The Blenderball tutorials are in the advanced section for a good reason. If you have already had a look at the files, you'd have seen that the game logic is very complex and mostly done with Python. It would have been possible to use the normal LogicBricks for most tasks, but this would have lead to a file being very inflexible, hard too overlook and difficult to change. Python gives us the option to create new levels by changing a few lines of code.

Load the **`Tutorials/Blenderball/Blenderball00.blend`** file. Its window layout is configured to work on new levels. Use **CTRL-RIGHTARROW** and **CTRL-LEFTARROW** to switch between the folowing screens:

ChooseRings

A Screen to select and choose the rings. It also contains the python script to where you can change the levels

Play

This is the full screen view to play Blenderball

Work

A Screen to play and change the Python level

Switch to the screen "Work" and have a look at the TextWindow containing the "InitRingRotation.py" script.

Figure 23-4. Python list holding inital ring rotations

```
  1 # List that holds initial rotations (L=left, R=right),
Levelnumber, Balls, number of pockets
  2 mixed= [ "L0 R0 L0 R0 R0 R0 :01 :03 01",
  3          "R0 R0 R0 R0 R0 R1 :02 :03 02",
  4          "R1 R1 R1 L0 R1 R2 :03 :03 03",
  5          "R2 R2 R1 R2 L0 R2 :04 :05 03",
  6          "R3 R3 R1 R2 L1 L0 :05 :03 03",
```

```
 7          "L1 L1 L1 R2 R1 R2 :06 :05 03",
 8          "R4 L4 R4 L4 R4 L3 :07 :07 05",
 9          "L1 L1 R0 L2 L1 R2 :08 :06 05",
10          "L0 R0 L1 R2 L0 L2 :09 :05 05",
11          "R1 R1 R1 R2 L1 R2 :10 :07 05",
12          "L4 R4 L2 R2 R1 R2 :11 :07 06",
13      "R2 L2 L5 L1 R0 R1 :12 :07 07" ]
```

Have a look at the list called "mixed". It contains the initial rotations of the rings in the levels line by line. So line two in the script (first line of the list) is used for the first level. A "L" means rotation to the left, and a "R" to the right. The number behind the letter determines how many steps the ring is rotated.

Now change the first "R0" in the first line to "R1" and press **PKEY** with the mouse in the textured CameraView to start the game engine. As opposed to the unmodified game (there are no rotation of rings in the first level) is that the second ring is now rotated to the right one step (looking from top to base).

The numbers at the end of each line are: the level number, the number of balls you have available to solve the level and how many pockets in the last ring have to be filled to solve the level.

Figure 23-5. Python list for the ring layout

```
 1 # make a list of rings to spawn, originals are hidden in
layer 8
 2 ring_name= [
 3     ["3RingIn",     "3RingWarpOut",   "3RingFlipIn",
"3RingFlipOut", "3Ring2FlipWarp", "3RingEndM"],
 4     ["3RingIn",     "3RingWarpOut",   "3RingFlipIn",
"3RingKick",    "3Ring2FlipWarp", "3RingEnd2"],
 5     ["3RingIn",     "3RingWarpOut",   "3RingExtraBall",
"3RingFlipOut", "3RingWarpIn",    "3RingEnd3"],
 6     ["3RingIn",     "3RingWarpOut",   "3RingForceBall",
"4RingSturn",   "3Ring2FlipWarp", "3RingEnd3"],
 7     ["3RingIn",     "3RingFlipIn",    "3RingExtraForce",
"2RingSturn",   "3RingForceBall", "5RingEnd3"],
 8     ["3RingInWarp", "3RingForceBall", "3RingKick",
"3RingFlipOut", "3RingWarpIn",    "3RingEnd3"],
```

```
 9     ["3RingIn",     "3RingKick",      "3RingFlipIn",
"3RingTruw",    "3RingSideFlip",  "8RingEnd5"],
10     ["3RingIn",     "3RingForceBall", "3RingFlipIn",
"1RingTruw",    "3RingSturn",     "8RingEnd5"],
11     ["3RingIn",     "3RingKick",      "3RingFlipIn",
"3RingTruw",    "3RingSideFlip",  "8RingEnd5"],
12     ["3RingIn",     "3RingWarpOut",   "3RingFlipIn",
"1RingTruw",    "3RingWarpIn",    "8RingEnd7"],
13     ["3RingIn",     "3RingWarpOut",   "3RingExtraForce",
"3RingTruw",    "3Ring2FlipWarp", "8RingEnd7"],
14     ["3RingWarpOut","3RingFlipIn",    "1RingTruw",
"3RingIn",      "3RingWarpIn",    "8RingEnd7"],
15     ["3RingIn",     "3RingFlipIn",    "3RingWarpOut",
"3RingTruw",    "3RingWarpIn",    "8RingEnd7"]
16             ]
17
```


The list in Figure 23-5 shows how the levels are made. Each ring has a unique name and you can assemble new levels. Use the "ChooseRings" screen to look at the rings. When you select a ring you will see the name in the Header of the ButtonWindow (e.g. OB:3RingSideFlip). Use this name in the "ring_name" Python list.

For example, change the first line of the ring_name list to:

```
 1      ["3RingIn", "3RingWarpOut", "3RingKick", "3RingSideFlip
","3Ring2FlipWarp", "3RingEndM"],
```

You can add completely new levels by adding new lines to both of the Python lists.

This tutorial shows how to use advanced Python scripting to build complex game logic. However, it also shows how easy it is to edit the game levels in just a few single lines of text. This can be done by anyone who knows how to use a text-editor. Plus, it also helps when loading levels from a disk or even from the Internet. Last but not least it provides a very flexible interface for a separate level-editor! You don't have to deal with a complex file-structure just produce a simple human-readable text file.


001
003
2.03beta
004
2.04
game Blender
005
010
2.10
011
2.12
blenderGameKit
blender
POWER
CREATE
A
B
C
D
2.20
022
023
2.23


_part VI ::
reference ...

The reference section will be your guide to exploring the Blender 3-D game engine further after following the tutorials. To learn modeling and linear animation refer to the „Official Blender 2.0 guide“.

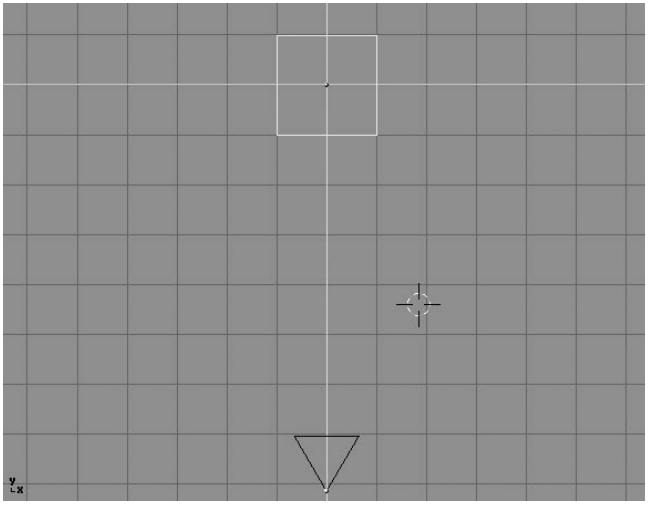
Chapter 24. Blender Windows and Buttons

This section describes the most important Blender-Windows and Buttons you need to create interactive content. Because Blender is a fully integrated application for creating both linear animations and stills plus real-time 3-D content, there are numerous buttons and window types that need to be explained. To explore the linear capabilities of Blender please refer to our other documentation (Section 29.5).

24.1. The 3DWindow

The 3DWindow is the most important window for working and navigating inside 3-D scenes. It is also used to play the interactive content. So a good knowledge of the options and capabilities will help you to make your scenes or explore scenes from the CD.

Figure 24-1. The 3DWindow canvas



The standard 3DWindow has:

A grid.The dimensions (distance between the gridlines) and resolution (number of lines) can be set with the ViewButtons. This grid is drawn as infinite in the presets of *ortho* ViewMode (Top, Front, Right view). In the other *views*, there is an finite „floor“. Many of the Blender commands are adjusted to the *dimension* of the grid, to function as a standard unit. Blender works best if the total „world“ in which the user operates continually falls more or less within the total grid floor (whether it is a space war or a logo animation).

Axes in color codes. The reddish line is the X axis, the green line is the Y axis, the blue line is the Z axis. In the Blender universe, the “floor” is normally formed by the X and Y axes. The height and „depth“ run along the Z axis.

A 3DCursor. This is drawn as a black cross with a red/white striped circle. A left mouse click (**LMB**) moves the 3DCursor. Use the SnapMenu (SHIFT+S) to give the 3DCursor a specific location. New Objects are placed at the 3DCursor location.

Layers (visible in the header buttons). Objects in "hidden" layers are not displayed. All hotkey commands and tools in Blender take the layers into account: Objects in the hidden layers are treated as *not* selected. See the following paragraph as well.

ViewButtons. Separate variables can be set for each 3DWindow, e.g for the *grid* or the *lens*. Use the **SHIFT+F7** hotkey or the WindowType button in the 3DHeader. The ViewButtons are explained in detail elsewhere in this manual.

24.1.1. 3DHeader

WindowType (IconMenu)

As with every window header, the first button allows you to set the window type.

Full Window (IconTog)

Maximize the window, or return it to its original size; return to the old screen setting. Hotkey: **ALT-CTRL+UPARROW**

Home (IconBut)

All Objects in the visible layers are displayed completely, centered in the window. Hotkey: **HOMEKEY**.

Layers (TogBut)

These 20 buttons show the available layers. In fact, a layer is nothing more than a *visibility flag*. This is an extremely efficient method for testing Object visibility. This allows the user to divide the work functionally.

For example: Cameras in layer 1, temporary Objects in layer 20, lamps in layers 1, 2, 3, 4 and 5, etc. All hotkey commands and tools in Blender take the layers into account. Objects in 'hidden' layers are treated as *unselected*.

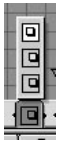
Use a left mouse click for the buttons, **SHIFT+LMB** for *extend select* layers.

Hotkeys: **1KEY**, **2KEY**, etc. **0KEY**, **MINUSKEY**, **EQUALKEY** for layers 1,2,3,4, etc. Use **ALT>+(1KEY**, **2KEY**, ... **OKEY**) for layers 11, 12, ... 20. Here, as well, use **SHIFT**+hotkey for *extend select*.

Lock (TogBut)

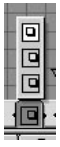
Every 3DWindow has it's own layer setting and active Camera. This is also true for a Scene: here it determines which layers - and which camera - are used to render a picture. The *lock* option links the layers and Camera of the 3DWindow to the Scene

and vice versa: the layers and Camera of the Scene are linked to the 3DWindow. This method passes a layer change directly to the Scene and to all other 3DWindows with the "Lock" option ON. Turn the "Lock" OFF to set a layer or Camera *exclusively* for the current 3DWindow. All settings are immediately restored by turning the button back ON.



LocalView (IconTog)

LocalView allows the user to continue working with complex Scenes. The currently selected Objects are taken separately, centered and displayed completely. The use of 3DWindow layers is temporarily disabled. Reactivating this option restores the display of the 3DWindow in its original form. If a picture is rendered from a LocalView, only the Objects present are rendered *plus* the visible lamps, according to the layers that have been set. Activating a new Camera in LocalView does not change the Camera used by the Scene. Normally, LocalView is activated with the hotkey PAD_SLASH.



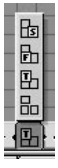
View Mode (IconMenu)

A 3DWindow offers 3 methods for 3-D display:

Orthonormal
: Blender offers this method from every view, not just from the X, Y or Z axes.

Perspective
: You can toggle between *orthonormal* and *perspective* with the hotkey **PAD_5**.

Camera
: This is the *view* as rendered. Hotkey: **PAD_0**.



View Direction (IconMenu)

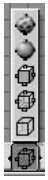
These pre-sets can be used with either *ortho* or *perspective*. Respectively, these are the:

TopView, hotkey **PAD_7**

FrontView, hotkey **PAD_1**

RightView, hotkey **PAD_3**

The hotkeys combined with **SHIFT** or (**CTRL**) give the opposite view direction. (Down View, Back View, Left View)



Draw Mode (IconMenu)

Set the drawing method. Respectively:

BoundBox.
The quickest method, for animation previews, for example.

WireFrame.
Objects are drawn assembled of lines.

Solid.
Z-buffered with the standard *OpenGL* lighting. Hotkey: **ZKEY**, this toggles between WireFrame and Solid.

Shaded.
This is as good an approach as is possible to the manner in which Blender renders - with Gouraud shading. It displays the situation from a single frame of the Camera. Hotkey: **SHIFT+Z**. Use **CTRL+Z** to force a recalculation of the view.

Textured.
Realtime textures (UV textures) are shown.

Objects have their own Draw Type, independent of the window setting (see „EditButtons->DrawType"). The rule is that the *minimum* DrawMode is displayed.

View Move (IconBut, click-hold)

Move the mouse for a *view* translation. This is an alternative for **SHIFT+MMB**.

View Zoom (IconBut, click-hold)

Move the mouse vertically to zoom in and out of the 3DWindow. This is an alternative for **CTRL+MMB**.

These buttons determine the manner in which the Objects (or vertices) are *rotated* or *scaled*.

Around Center (IconRow)

The midpoint of the *boundbox* is the center of rotation or scaling. Hotkey: **COMMAKEY**.

Around Median (IconRow)

The median of all Objects or vertices is the center of rotation or scaling.

Around Cursor (IconRow)

The 3DCursor is the midpoint of rotation or scaling. Hotkey: **DOTKEY**.

Around Individual Centers (IconRow)

All Object's rotate or scale around their own midpoints. In EditMode: all vertices rotate or scale around the Object midpoint.

EditMode (IconTog)

This button starts or terminates EditMode. Hotkey: **TAB** or **ALT+E**.

VertexPaint (IconTog)

This button starts or terminates VertexPaintMode. Hotkey: **VKEY**.

FaceSelect (IconTog)

This button starts or the FaceSelect mode. Hotkey: **FKEY**.

Proportional Vertex Editing Tool (IconTog)

The Proportional Vertex Editing tool can be activated with the Icon in 3DWindow header, or **OKEY**.

The Proportional Editing tool is then available in Editmode for all Object types. This tool works like a "magnet", you select a few vertices and while editing (grab, rotate, scale) the surrounding vertices move proportionally with it. Use the NumPad-plus and NumPad-minus keys to adjust the area of influence, this can be done "live" while editing.

You can choose between a sharp fallof and a smooth falloff.

OpenGL Renderer (IconTog)

A left mouse click renders the current view in *OpenGL*. **CTRL-LMB** renders a animation in OpenGL. The rendered pictures are saved as in the DisplayButtons indicated.

24.1.2. The Mouse

The mouse provides the most direct access to the 3DWindow. Below is a complete overview:

Left mouse

Position the 3DCursor.

CTRL + left mouse

In EditMode: create a new vertex.

left mouse (click-hold-draw)

These are the Gestures. Blender's gesture recognition works in three ways:

Draw a straight line: start *translation* mode (Grabber)

Draw a curved line: start *rotation* mode.

Draw a V-shaped line: start *scaling* mode.

Middle mouse (click-hold)

Rotate the direction of view of the 3DWindow. This can be done in two ways (and can be set in the UserMenu):

The trackball method. In this case, where in the window you start the mouse movement is important. The rotation can be compared to rotating a ball, as if the mouse grasps and moves a tiny miniscule point on a ball and moves it. If the movement starts in the middle of the window, the *view* rotates along the horizontal and vertical window axes. If the movement begins at the edge of the window, the *view* rotates along the axis perpendicular to the window.

The turntable method. A horizontal mouse movement always results in a rotation around the global Z axis. Vertical mouse movements are corrected for the view direction, and result in a combination of (global) X and Y axis rotations.

SHIFT+MMB (click-hold)

Translate the 3DWindow. Mouse movements are always corrected for the view direction.

CTRL+MMB (click-hold)

Zoom in/out on the 3DWindow.

Right mouse

Select Objects or (in EditMode) vertices. The last one selected is also the *active* one. This method guarantees that a maximum of 1 Object and 1 vertex are always selected. This selection is based on graphics (the wireframe).

SHIFT+RMB

Extend select Objects or (in EditMode) vertices. The last one selected is also the *active* one. Multiple Objects or *vertices* may also be selected. This selection is based on graphics too (the wireframe).

CTRL+RMB

Select Objects on the Object-centers. Here the wireframe drawing is not taken into account. Use this method to select a number of identical Objects in succession, or to make them active.

SHIFT+CTRL+RMB

Extend select Objects. The last Object selected is also the *active* one. Multiple Objects can be selected.

Right mouse (click-hold-move)

Select and start *translation* mode, the Grabber. This works with all the selection methods mentioned.

24.1.3. NumPad

The numeric keypad on the keyboard is reserved for *view* related hotkeys. Below is a description of all the keys with a brief explanation.

PAD_SLASH

LocalView. The Objects selected when this command is invoked are taken separately and displayed completely, centered in the window. See the description of 3DHeader->LocalView.

PAD_STAR

Copy the rotation of the *active* Object to the current 3DWindow. This works as if this Object is the camera, without including the translation.

PAD_MINUS, PAD_PLUS

Zoom in, zoom out. This also works for Camera ViewMode.

PAD_DOT

Center and zoom in on the selected Objects. The *view* is changed in a way that can be compared to the LocalView option.

PAD_5

Toggle between *perspective* and *orthonormal* mode.

PAD_9

Force a complete recalculation (of the animation systems) and draw again.

PAD_0

View from the current *camera*, or from the Object that is functioning as the *camera*.

CTRL+PAD_0

Make the *active* Object the *camera*. Any Object can be used as the camera. Generally, a Camera Object is used. It can also be handy to let a spotlight function temporarily as a camera when directing and adjusting it. **ALT+PAD_0** reverts to the previous *camera*. Only Camera Objects are candidates for the "previous camera" command.

PAD_7

TopView. (along the negative Z axis, Y up)

SHIFT+PAD_1

DownView. (along the positive Z axis, Y up)

PAD_1

FrontView. (along the positive Y axis, Z up)

SHIFT+PAD_1

BackView. (along the negative Y axis, Z up)

PAD_3

RightView. (along the negative X axis, Z up)

SHIFT+PAD_3

LeftView. (along the positive X axis, Z up)

PAD_2, PAD_8

Rotate using the *turntable* method. Depending on the view, this is a rotation around the X and Y axes.

PAD_4 PAD 6

Rotate using the *turntable* method. This is a rotation around the Z axis.

SHIFT+(PAD_2, PAD_8)

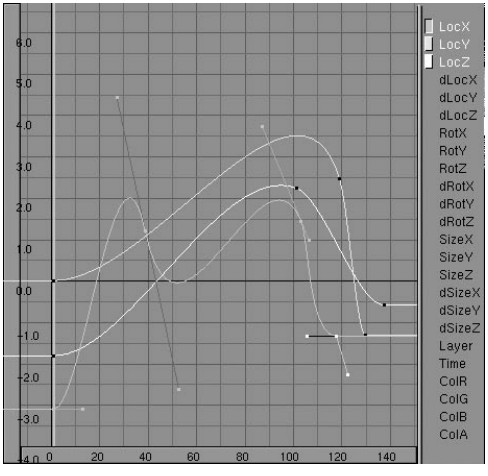
Translate up or down; corrected for the current view.

SHIFT+(PAD_4,PAD_6)

Translate up or down; correct for the view.

24.2. IpoWindow

Figure 24-2. The IpoWindow



The IpoWindow allows you to visualize and manipulate animation curves which can control nearly every aspect of an animation inside Blender. For the Blender GameEngine, the most important aspects are the positions or rotations of objects, and the color of objects.

24.2.1. IpoHeader

WindowType (IconMenu)

As with every window header, the first button enables you to set the window type.

Full Window (IconTog)

Maximize the window or return it to the previous window display size; return to the old screen setting. HotKey: **(ALT-)CTRL+UPARROW**

Home (IconBut)

All visible curves are displayed completely, centered in the window. HotKey: **HOMEKEY**.

IpoKeys (IconTog)

This is a drawing mode for the animation curves in the IpoWindow (the IpoCurves). Yellow vertical lines are drawn through all the vertices of the curves. Vertices of different curves at the same location in "time" are joined together and can easily be selected, moved, copied or deleted.

This method adds the ease of traditional *key* framing to the animation curve system.

For Object-Ipos, these IpoKeys can also be drawn and transformed in the 3DWindow. Changes in the 3-D position are processed immediately in the IpoCurves.

Ipo Type

Depending on the *active* Object, the various Ipo systems can be specified with these buttons.

Object Ipo (IconRow)

Settings, such as the location and rotation, are animated for the *active* Object. All Objects in Blender can have this Ipo block.

Material Ipo (IconRow)

Settings of the *active* Material are animated for the *active* Object.

A NumBut is added as an extra feature. This button indicates the number of the active Texture *channel*. Eight Textures, each with its own mapping, can be assigned per Material. Thus, per Material-Ipo, 8 curves in the row "OfsX, OfsY, ...Var" are available.

Speed Ipo (Icon Row)

If the *active* Object is a *path* Curve, this button can be used to display the speed-Ipo.

Lamp Ipo (IconRow)

If the *active* Object is a Lamp, this button can be used to animate light settings.

World Ipo (IconRow)

Used to animate a number of settings for the WorldButtons.

VertexKey Ipo (IconRow)

If the *active* Object has a VertexKey, the keys are drawn as horizontal lines. Only one IpoCurve is available to interpolate between the Keys.

Sequence Ipo (IconRow)

The active Sequence Effect can have an IpoCurve.

The DataButtons can be used to control the Ipo blocks themselves.

Ipo Browse (MenuBut)

Choose another Ipo from the list of available Ipos. The option "Add New" makes a complete copy of the current Ipo. This is not visible; only the name in the adjacent button will change. Only Ipos of the same type are displayed in the menu list.

IP: (TextBut)

Give the current Ipo a new and unique name. After the new name is entered, it appears in the list, sorted alphabetically.

Users (NumBut)

If this button is displayed, there is more than one user for the Ipo block. Use the button to make the Ipo "Single User".

Unlink Ipo (IconBut)

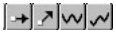
The current Ipo is unlinked.

Copy to Buffer (IconBut)

All selected IpoCurves are copied to a temporary buffer.

Paste from Buffer (IconBut)

All selected *channels* in the IpoWindow are assigned an IpoCurve from the temporary buffer. The rule is: the sequence in which they are copied to the buffer is the sequence in which they are pasted. A check is made to see if the number of IpoCurves is the same.



Extend mode Constant (IconBut)

The end of selected IpoCurves are horizontally extrapolated.

Extend mode Direction (IconBut)

The ends of selected IpoCurves continue extending in the direction in which they end.

Extend mode Cyclic (IconBut)

The full length of the IpoCurve is repeated cyclically.

Extend mode Cyclic Extrapolation (IconBut)

The full length of the IpoCurve is extrapolated cyclically.

View Zoom (IconBut, click-hold)

Move the mouse horizontally or vertically to zoom in or out on the IpoWindow. This is an alternative for **CTRL+MMB**.

View Border (IconBut)

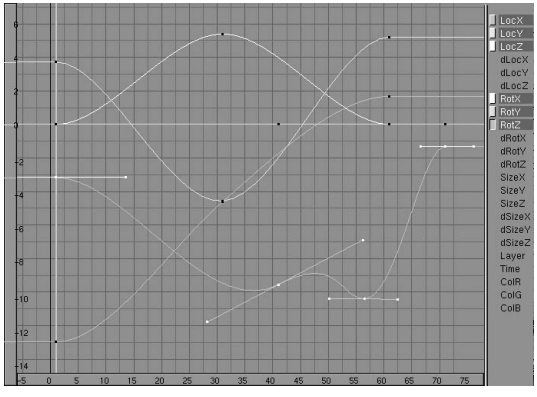
Draw a rectangle to indicate what part of the IpoWindow should be displayed in the full window.

Lock (TogBut)

This button locks the update of the 3DWindow while editing in the IpoWindow, so you can see changes made to the Ipo in real-time in the 3DWindow. This option works extremely well with relative vertex keys.

24.2.2. IpoWindow

Figure 24-3. The IpoWindow



The IpoWindow shows the contents of the Ipo block. Which one depends on the Ipo Type specified in the header.

The standard IpoWindow has a grid with the time expressed horizontally in frames and vertical values that depend on the *channel*. There are 2 sliders at the edge of the IpoWindow. How far the IpoWindow is zoomed in can be seen on the sliders, which can also be used to move the *view*.

The right-hand part of the window shows the available *channels*.

To make it easier to work with rotation-IpoCurves, they are displayed in degrees (instead of in radials). The vertical scale relation is: 1.0 "Blender unit" = 10 degrees.

In addition to the IpoCurves, the VertexKeys are also drawn here. These are horizontal blue lines; the yellow line visualizes the *reference* Key.

Each *channel* can be operated with two buttons:

IpoCurve Select (TogBut)

This button is only displayed if the *channel* has an IpoCurve. The button is the same color as the IpoCurve. Use the button to select IpoCurves. Multiple buttons can be (de)selected using **SHIFT+LMB**.

Channel Select (TogBut)

A *channel* can be selected whether there is an IpoCurve or not. IpoCurves are only drawn for selected channels. Multiple *channels* can be (de)selected using **SHIFT+LMB**.

24.2.3. The Mouse

CTRL+LMB

Create a new vertex. These are the rules:

There is no IpoBlock (in this window) *and* one *channel* is selected: a new IpoBlock is created along with the first IpoCurve with one vertex.

There is already an Ipolock, and a *channel* is selected without an IpoCurve: a new IpoCurve with one vertex is added

Otherwise a new vertex is simply added to the selected IpoCurve.

This is *not* possible if multiple IpoCurves are selected or if you are in EditMode.

Middle mouse (hold-move)

Depending on the position within the window:

On the *channels*; if the window is not high enough to display them completely, the visible part can be shifted vertically.

On the sliders; these can be moved. This only works if you are zoomed in.

For the rest of the window; the *view* is translated.

CTRL+MMB (hold-move)

Zoom in/out on the IpoWindow. You can zoom horizonally or vertically using horizontal and vertical mouse movements.

Right mouse

Selection works the same here as in the 3DWindow: normally one item is selected. Use **SHIFT** to expand or reduce the selection (*extend* select).

If the IpoWindow is in IpoKey mode, the IpoKeys can be selected.

If at least 1 of the IpoCurves is in EditMode, only its vertices can be selected.

VertexKeys can be selected if they are drawn (horizontal lines)

The IpoCurves can be selected.

Right mouse (click-hold-move)

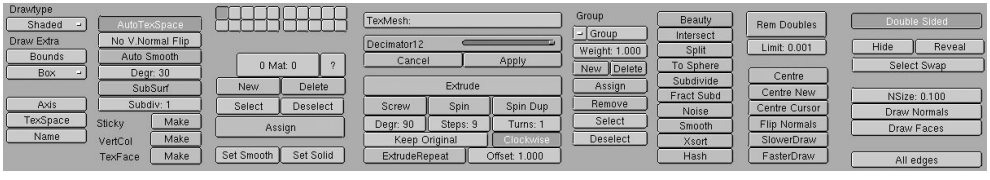
Select and start *translation* mode, i.e. the Grabber. The selection can be made using any of the four selection methods discussed above.

SHIFT+RMB

Extend the selection.

24.3. EditButtons

Figure 24-4. The EditButtons (F9)



The settings in this ButtonsWindow visualize the ObData blocks and provide tools for the specific EditModes. Certain buttons are redrawn depending on the type of ObData. The types that can be used in the Blender game engine are: Mesh, Empty, Armature, Lamp and Camera. Options and Buttons that are not appropriate for the Blender game engine are not described here.

The DataButtons in the header specify which block is visualized. Mesh is used as an example here, but the use of the other types of ObData is identical.

Mesh Browse (MenuBut)

Selects another Mesh from the list provided.

ME: (TextBut)

Gives the current block a new and unique name. The new name is inserted in the list and sorted alphabetically.

Users (But)

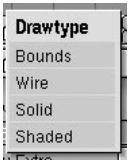
If the block is used by more than one Object, this button shows the total number of Objects. Press the button to change this to "Single User". An exact copy is then created.

OB: (TextBut)

Gives the current Object a new and unique name. The new name is inserted in the list and sorted alphabetically.

This group of buttons specifies Object characteristics.

DrawType (MenuBut)



Choose a preference for the standard display method used in the 3DWindow from the list provided. The "DrawType" is compared with the "DrawMode" set in the 3DHeader; the least complex method is the one actually used.

The types, listed in increasing degrees of complexity, are:

Bounds. A bounding object is drawn in the dimensions of the object.

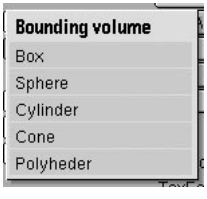
Wire. The wire model is drawn.

Solid. Zbuffered with the standard OpenGL lighting.

Shaded. This display, which uses Gouraud shading, is the best possible way to view the manner in which Blender renders. It depicts the situation of a single frame from the Camera's point of view. Use **CTRL+Z** to force a recalculation.

The „Draw Extra" options are displayed above the selected DrawType.

BoundBox (TogBut)



A bounding object is displayed in the dimensions of the object.

Box (MenuBut)

With this MenuButton you can choose between different bound-objects.

Axis (TogBut)

The axes are drawn with X, Y and Z indicated.

Name (TogBut)

The name of the Object is printed at the Object centre.

The *layer* setting of the Object. Use **SHIFT-LMB** to activate multiple layers.

Set Smooth (But)

This sets a *flag* which specifies that rendering must be performed with normal interpolation. In EditMode, it works on the selection. Outside EditMode everything becomes "Smooth".

Set Solid (But)

This sets a *flag* which indicates that rendering must be "Solid". In EditMode this works on the selection. Outside EditMode everything becomes "Solid".

24.3.1. EditButtons, Mesh

AutoTexSpace (TogBut)

This option automatically calculates the texture area, after leaving EditMode. You can also specify a texture area yourself (outside EditMode, in the 3DWindow; **TKEY**), in which case this option is turned OFF.

No V.Normal Flip (TogBut)

Because Blender normally renders as faces double-sided, the direction of the normals (towards the front or the back) is automatically corrected during rendering. This option turns this automatic correction off, allowing "smooth" rendering of faces that have sharp angles (smaller than 100 degrees). Be sure the face normals are consistently set in the same direction (**CTRL+N** in EditMode). The direction of the normals is especially important for real-time models, because the game engine renders them single sided for reasons of speed.

AutoSmooth (TogBut)

Automatic smooth rendering (not faceted) for meshes. Especially interesting for imported Meshes done in other 3-D applications. The Button "Set smooth" also has to be activated to make "Auto Smooth" work. The smoothing isn't displayed in the 3DWindow.

Degr: (NumBut)

Determines the degree to which faces can meet and still get smoothed out by "Auto Smooth".

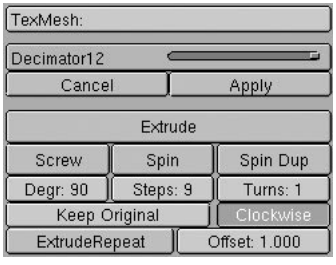
Make VertCol (But)

A color can be specified per vertex. This is required for the VertexPaint option. If the Object DrawType is "Shaded", these colors are copied to the vertex colors. This allows you to achieve a *radiosity*-like effect (set MaterialButtons->VertCol ON). If the Mesh is "Double Sided", this is automatically turned off.

Make TexFace (But)

Assigns a texture per face. This will be set automatically when you use the UV-Editor to texture a real-time model. Unchecking this option clears all UV coordinates.

Decimator (NumSli)



Decimator (NumSli)

This slider will reduce your mesh faces to the number you indicate with the slider. Watch your mesh closely to see if the number of faces you demand is still enough to retain the desired shape.

Note: *Mesh decimation will remove UV coordinates and vertexcolors!*

Cancel (Button)

Resets the mesh to its original state before decimation.

Apply (Button)

Decimates according to the value indicated in the decimation slider. After using "Apply" there is no way back!

Extrude (But)

The most important of the Mesh tools: Extrude Selected. In EditMode "Extrude" converts all selected *edges* to *faces*. If possible, the selected faces are also duplicated. Grab mode starts immediately after this command is executed. If there are multiple 3DWindows, the mouse cursor changes to a question mark. Click in the 3DWindow in which "Extrude" must be executed. HotKey: EKEY.

Screw (But)

This tool starts a repetitive "Spin" with a screw-shaped revolution on the selected vertices. You can use this to create screws, springs or shell-shaped structures.

Spin (But)

The "Spin" operation is a repetitively rotating "Extrude". This can be used in every view of the 3DWindow, the rotation axis always goes through the 3DCursor, perpendicular to the screen. Set the buttons "Degr" and "Steps" to the desired value.

If there are multiple 3DWindows, the mouse cursor changes to a question mark. Click in the 3DWindow in which the "Spin" must occur.

Spin Dup (But)

Like "Spin", but instead of an "Extrude", there is duplication.

Degr (NumBut)

The number of degrees by which the "Spin" revolves.

Steps (NumBut)

The total number of "Spin" revolutions, or the number of steps of the "Screw" per revolution.

Turns (NumBut)

The number of revolutions the "Screw" turns.

Keep Original (TogBut)

This option saves the selected original for a "Spin" or "Screw" operation. This releases the new vertices and faces from the original piece.

Clockwise (TogBut)

The direction of the "Screw" or "Spin", can be clockwise, or counterclockwise.

Extrude Repeat (But)

This creates a repetitive "Extrude" along a straight line. This takes place perpendicular to the view of the 3DWindow.

Offset (NumBut)

The distance between each step of the "Extrude Repeat". HotKey: **WKEY**.

Vertex Group Buttons

This group of Buttons is meant for assigning vertices and weights to the bones of an Armature. Besides the "Weight" Button all options are only drawn when the active object is in EditMode.

Group Browse (MenuBut)

Browse the defined groups of vertices for this mesh. The text button shows the actual vertex group name. Click it with **LMB** to edit the name.

Weight (NumBut)

Sets the weight for groups and for use in WeightPaint

New (But)

Creates a new vertex group

Delete (But)

Deletes the actual vertex group

Assign (But)

Assigns the selected vertices to the actual group

Remove (But)

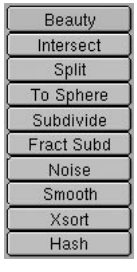
Removes selected vertices from the actual group

Select (But)

Selects all vertices from the actual group

Deselect (But)

Deselects all vertices from the actual group



Intersect (But)

Select the faces (vertices) that need an intersection and press this button. Blender now intersects all selected faces with each other.

Split (But)

In EditMode, this command "splits" the selected part of a Mesh without removing faces. The split sections are no longer connected by *edges*. Use this to control *smoothing*. Since the split parts can have vertices in the same position, we recommend that you make selections with the **LKEY**. HotKey: **YKEY**.

To Sphere (But)

All selected vertices are blown up into a spherical shape, with the 3DCursor as a midpoint. A requester asks you to specify the factor for this action. HotKey: **WKEY**.

Beauty (TogBut)

This is an option for "Subdivide". It splits the faces into halves lengthwise, converting elongated faces to squares. If the face is smaller than the value of "Limit", it is not longer subdivided.

Subdivide (But)

Selected faces are divided into quarters; all edges are split in half. HotKey: **WKEY**.

Fract Subd (But)

Fractal Subdivide. Like "Subdivide", but now the new vertices are set with a random vector up or down. A requestor asks you to specify the amount. Use this to generate landscapes or mountains.

Noise (But)

Here Textures can be used to move the selected vertices up a specific amount. The local vertex coordinate is used as the texture coordinate. Every Texture type works with this option. For example, the Stucci produces a landscape effect. You can also use Image textures to express them in relief.

Smooth (But)

Shortens all edges with both vertices selected. This flattens sharp angles. HotKey: **WKEY**.

Xsort (But)

Sorts the vertices in the X direction. This creates interesting effects with (relative) Vertex Keys or "Build Effects" for Halos.

Hash (But)

This makes the sequence of vertices completely random.

Rem Doubles (But)

Remove Doubles. All selected vertices that are closer to one another than the "Limit" are combined and redundant faces are removed.

Centre (But)

Each ObData has its own local 3-D space. The null point of this space is placed at the Object center. This option calculates a new, centred null point in the ObData.

Centre New (But)

As above, but now the Object is placed in such a way that the ObData appears to remain in the same place.

Centre Cursor (But)

The new null point of the object is the 3DCursor location.

Flip Normals (But)

Toggles the direction of the face normals. HotKey: **WKEY**.

SlowerDraw, FasterDraw. (But)

When leaving EditMode all edges are tested to determine whether they must be displayed as a wire frame. Edges that share two faces with the same normal are never displayed. This increases the "recognizability" of the Mesh and considerably speeds up drawing. With "SlowerDraw" and "FasterDraw", you can specify that additional or fewer edges must be drawn when you are not in EditMode.

Double Sided (TogBut)

Only for display in the 3Dwindow; this can be used to control whether double-sided faces are drawn. Turn this option OFF if the Object has a negative "size" value (for example an X-flip).

Hide (But)

All selected vertices are temporarily hidden. HotKey: **HKEY**.

Reveal (But)

This undoes the "Hide" option. HotKey: **ALT+H**.

Select Swap (But)

Toggles the selection status of all vertices.

NSize (NumBut)

The length of the face normals, if they have been drawn.

Draw Normals (NumBut)

Indicates that the face normals must be drawn in EditMode.

Draw Faces (NumBut)

Indicates that the face must be drawn (as Wire) in EditMode. Now it also indicates whether faces are selected.

AllEdges (NumBut)

After leaving EditMode, all edges are drawn normally, without optimization.

24.3.2. EditButtons, Armatures

Rest Pos (But)

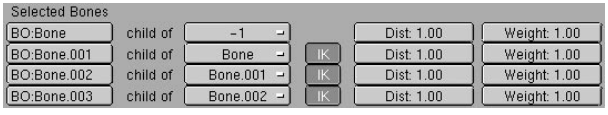
Disables all animation and puts the armature into its initial (resting) position.

Draw Axes (But)

Draw bone axes

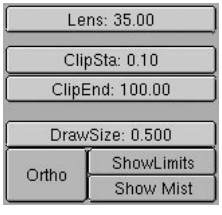
(But)

Draw bone names



Buttons to name, organize and build hierarchies of bones. See also Section 22.3.

24.3.3. EditButtons, Camera



Lens (NumBut)

This number is derived from the lens values of a photo camera: “120” is telelens, “50” is normal, “28” is wide angle.

ClipSta, ClipEnd (NumBut)

Everything that is visible from the Camera’s point of view between these values is rendered. Try to keep these values close to one another, so that the Z-buffer functions optimally.

DrawSize (NumBut)

The size in which the Camera is drawn in the 3DWindow.

Ortho (TogBut)

A Camera can also render orthogonally. The distance from the Camera then has no effect on the size of the rendered objects.

ShowLimits (TogBut)

A line that indicates the values of “ClipSta” and “ClipEnd” is drawn in the 3DWindow near the Camera.

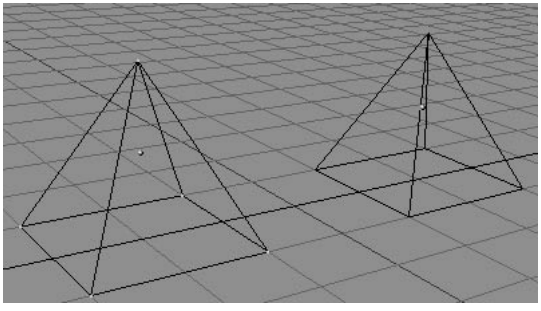
ShowMist (TogBut)

A line that indicates the area of the "mist" (see WorldButtons Section 24.5) is drawn near the Camera in the 3DWindow.

24.4. EditMode

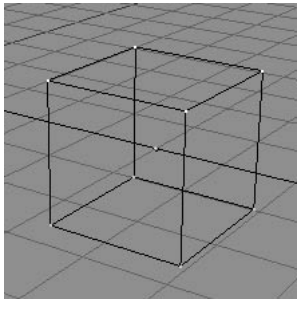
When working in 3-D space, you can basically perform two types of operations: operations that affect the whole object and operations that affect only the geometry of the object itself but not its global properties such as the location or rotation.

In Blender you switch between these two modes with the **TAB**-key. A selected object outside EditMode is drawn in purple in the 3DWindows (in wireframe mode). To indicate the EditMode the Object's vertices are drawn. Selected vertices are yellow, non-selected are purple.



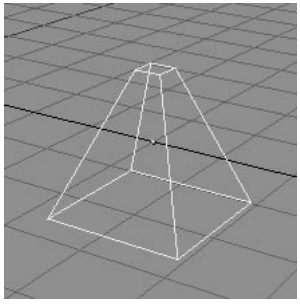
Vertices can be selected like objects with the **RMB**, holding **SHIFT** allows you to select more than one vertex. With some vertices selected you can use **GKEY**, **RKEY** or **SKEY** for manipulating the vertices as you would for whole objects.

Add a cube to the default scene. Use the 3DCursor to place it away from the default plane or use **GKEY** to move it after leaving EditMode.

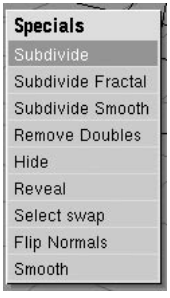


Switch the 3DWindow to a side view (**PAD3**), select the cube if it is deselected and press **TAB** to enter the EditMode again. Now press **BKEY** for the BorderSelect and draw a rectangle with the **LMB** around the top four vertices of the cube (you can only see two vertices, because the other two are hidden behind the first two!).

The top vertices change to yellow to indicate that they are selected. You can rotate the view to make sure you really have selected four vertices.



Now press **SKEY** and move the mouse up and down. You will see how the four vertices are scaled. Depending on your movement you can make a pyramid or a chopped-off pyramid. You can now also try to grab and rotate some vertices of other objects to get a feeling for the EditMode.

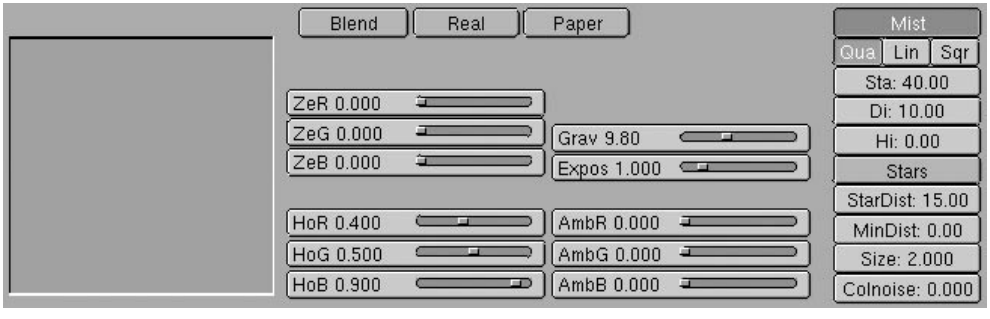


Using **WKEY** you can call up the "Specials"-menu in EditMode. With that menu you can quickly access the functions often needed for polygon-modeling. You can find the same functionality in the EditButtons **F9**.

24.5. WorldButtons

The WorldButtons define global options for the scene. Only the options appropriate for the Blenders game engine are explained here.

Figure 24-5. The WorldButtons



HoR, G, B (NumSli)

Here you define the color of the world, rendered where no other object is rendered.

Grav (NumSli)

Defines the gravity of the world. This influences the force you need to move an object up for example and how fast it will accelerate while falling.

Mist (TogBut)

Activates the rendering of Mist. This blends objects at a certain distance into the horizon color.

Qua, Lin, Sqr (RowBut)

Determines the progression of the mist. Quadratic, linear or inverse quadratic (square root), respectively. “Sqr” gives a thick “soupy” mist, as if the scene is viewed under water.

Sta (NumBut)

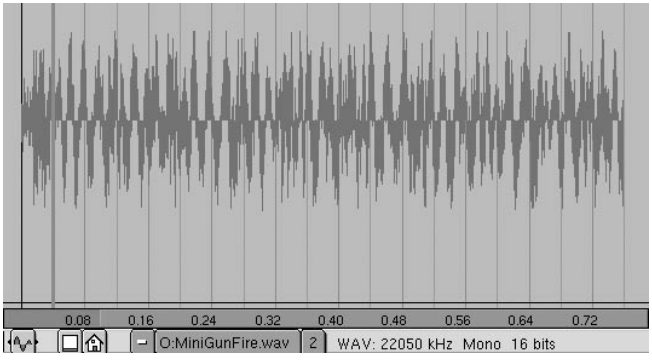
The start distance of the mist, measured from the Camera.

Di (NumBut)

The depth of the mist, with the distance measured from “Sta”.

24.6. SoundWindow

Figure 24-6. The SoundWindow



The SoundWindow is used to load and visualize sounds. You can grab and zoom the window and its content like every other window in Blender.

The green bar indicates the position of the FrameSlider. This can be used to synchronize a sound with an Ipo animation. In the lower part of the window you also have an indicator of the sound length in seconds.

In the SoundWindow Header see the usual window buttons, the user buttons and some information about the sound.

Chapter 25. Real-time Materials

Materials for Blenders game engine are applied with vertex-paint or UV-Textures.

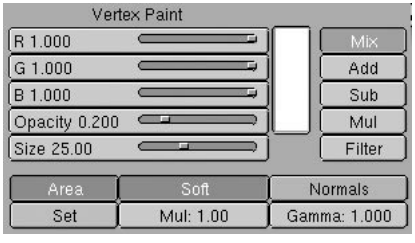
With VertexPaint you can paint on Meshes, giving them solid colors or patterns of color. VertexPaint is also a very valuable tool to make the suggestion of light on faces and even more important to vary textures. Without using the CPU intense real-time lighting you can create the impression of a colored lamp shining on objects, darken corners or even paint shadows.

Textures have a big impact on the look and feel of your game or interactive environment. With textures, you are able to create a very detailed look even with a low poly model. With alpha channel textures, you are also able to create things like windows or fences without actually modeling them.

25.1. Vertex Paint

To start VertexPaint press **VKEY** or select the VertexPaint icon in the 3DWindow Header. The selected object will now be drawn solid. You can therefore now draw on the vertices of the object while holding **LMB**, the size of the brush is visualized by a circle while drawing. **RMB** will sample the color under the mouse pointer.

Figure 25-1. Vertex Paint related Buttons in the Paint/FaceButtons



Enter the Paint/FaceButtons to see the sampled color. Here you can also find more options to control VertexPaint:

R, G, B (NumSli)

The active color used for painting.

Opacity (NumSli)

The extent to which the vertex color changes while you are painting.

Size (NumSli)

The size of the brush, which is drawn as a circle during painting.

Mix (RowBut)

The manner in which the new color replaces the old when painting: the colors are mixed.

Add (RowBut)

The colors are added.

Sub (RowBut)

The paint color is subtracted from the vertex color.

Mul (RowBut)

The paint color is multiplied by the vertex color.

Filter (RowBut)

The colors of the vertices of the painted face are mixed together, with the opacity factor.

Area (TogBut)

In the *back* buffer, Blender creates an image of the painted Mesh, assigning each face a color number. This allows the software to quickly see what faces are being painted. Then, the software calculates how much of the face the brush covers, for the degree to which paint is being applied. You can set this calculation with the option "Area".

Soft (TogBut)

This specifies that the extent to which the vertices lie within the brush also determines the brush's effect.

Normals (TogBut)

The vertex normal (helps) determine the extent of painting. This causes an effect as tough you were painting with light.

Set (But)

The "Mul" and "Gamma" factors are applied to the vertex colors of the Mesh.

Mul (NumBut)

The number by which the vertex colors are multiplied when "Set" is pressed.

Gamma (NumBut)

The number by which the clarity (Gamma value) of the vertex colors are changed when "Set" is pressed.

25.2. TexturePaint

To start TexturePaint select the TexturePaint icon  in the 3DWindow Header.

Info: *TexturePaint needs a textured object to work. See Section 25.3. You also need to unpack a packed texture first (see Section 25.3.5).*

You can now paint on the texture of the object while holding the **LMB**. **RMB** will sample the color located under the mouse pointer.

Enter the Paint/FaceButtons to see the sampled color. Here you can also find more options to control TexturePaint:

R, G, B (NumSli)

The active color used for painting.

Opacity (NumSli)

The extent to which the color covers the underlying texture.

Size (NumSli)

The size of the brush.

25.3. The UV Editor

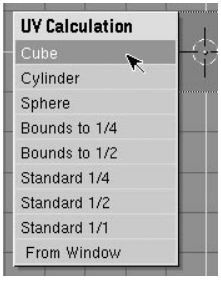
The UV editor is fully integrated into Blender and allows you to map textures onto the faces of your models. Each face can have individual texture coordinates and an individual image assigned. This can be combined with vertex colors to darken or lighten the texture or to tint it.

To start UV editing, enter FaceSelect mode with the **FKEY** or the FaceSelect icon in the 3DWindow Header. The mesh is now drawn *Z-Buffered*. In textured mode (**ALT-Z**) untextured faces are drawn in purple to indicate the lack of a texture. Selected faces are drawn with a dotted outline.

To select faces use the right mouse button, with the **BKEY** you can use BorderSelect and the **AKEY** selects/deselects all faces. While in FaceSelect mode you can enter EditMode (**TAB**) and select vertices. After leaving EditMode the faces defined by the selected vertices are selected in FaceSelect mode. The active face is the last selected face: this is the reference face for copy options.

RKEY allows you to rotate the UV coordinates or VertexColors.

25.3.1. Mapping UV Textures



When in FaceSelectMode (**FKEY**) you can do a calculate UV textures for selected faces by pressing **UKEY**. A menu will give you the following choices:

Cube

Cubic mapping, a requester asks for a scaling property

Cylinder

Cylindrical mapping calculated from the center of the selected faces

Sphere

Spherical mapping calculated from the center of the selected faces

Bounds to...

The UV coordinates are calculated using the projection of the 3DWindow and then scaled to a bound box of the desired size

Standard...

Each face gets the default set of square UV coordinates

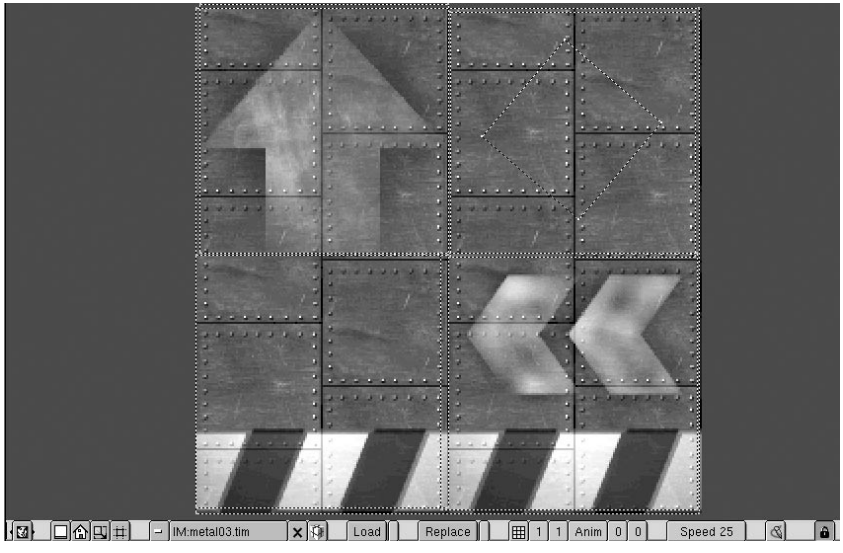
From Window

UV coordinates are calculated from the active 3DWindow

25.3.2. The ImageWindow

To assign images to faces you need to open an ImageWindow with **SHIFT-F10**.

Figure 25-2. The Image Window



The first Icon keeps UV polygons square while editing this is a big help while texturing. Just drag one or two vertices around and the others follow to keep the polygon square. The second one keeps the vertices inside the area of the image.

With the UserBrowse (MenuButton) you can browse, assign and delete loaded images on the selected faces.

"Load" loads a new image and assigns it to the selected faces. "Replace" replaces (scene global) an image on all faces assigned to the old image. The small buttons to the right of the "Load" and "Replace" buttons open a FileWindow without the thumbnail images.

The grid icon enables the use of more (rectangular) images in one map. This is used for texturing from textures containing more than one image in a grid and for animated textures. The following two number buttons define how many parts the texture has in X and Y direction. Use **SHIFT-LMB** to select the desired part of the image in GridMode.

The "Anim" button enables a simple texture animation. This works in conjunction with the grid settings, in a way that the parts of the texture are displayed in a row in game mode. With the number buttons to the right of the "Anim" button you define the start and end part to be played. "Speed" controlls the playback speed in frames per second.

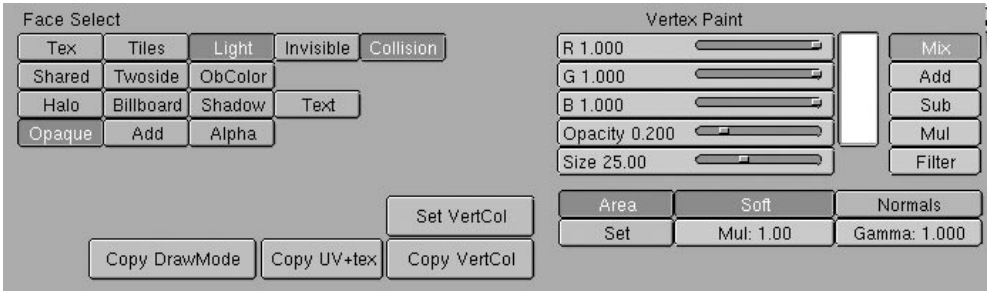
With the lock icon activated, any changes on the UV polygons in the ImageWindow are shown in real-time in the 3DWindows (in textured mode).

Vertices in the ImageWindow are selected and edited (rotate, grab) like vertices in EditMode in the 3DWindows. Drag the view with the middle mouse, zoom with PAD+ and PAD-.

25.3.3. The Paint/FaceButtons

When in FaceSelect mode, you can access the Paint/FaceButtons with the Icon in the ButtonsWindow Header. In the Paint/FaceButtons you'll find all functions to set the attributes for faces and access the VertexPaint options.

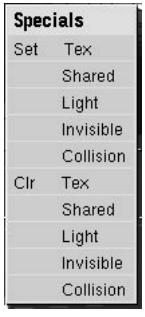
Figure 25-3. The Paint/FaceButtons



The following modes always work on faces and display the setting of the active

face. Two colored lines in the 3DWindow and the ImageWindow indicate the active face. The green line indicates the U coordinate, the red line the V coordinate. To copy the mode from the active to the selected faces use the copy buttons ("Copy DrawMode", "Copy UV+tex" and "Copy VertCol") in the Paint/FaceButtons. In FaceSelect mode the special menu has some entries to quickly set and clear modes on all selected faces, see Figure 25-4.

Figure 25-4. The special menu for the FaceSelectMode



Face modes

Tex

This enables the use of textures. To use objects without textures, disable "Tex" and paint the faces with VertexPaint.

Tiles

This indicates and sets the use of the tile mode for the texture, see Section 25.3.2.

Light

Enables real-time lighting on faces. Lamps only affect faces of objects that are in the same layer as the lamp. Lamps can also be placed on more than one layer, which makes it possible to create complex real-time lighting situations. See also Section 26.7.

Invisible

Makes faces invisible. These faces are still calculated for collisions, so this gives you an option to build invisible barriers, etc.

Collision

The faces with this option are evaluated by the game engine. If that is not needed, switch off this option to save resources.

Shared

With this option vertex colors are blended across faces if they share vertices.

Twoside

Faces with this attribute are rendered twosided in the game engine.

ObColor

Faces can have color that can be animated by using the ColR, ColG, ColB and

ColA Ipos. Choosing this option replaces the vertex colors.

Halo

Faces with this attribute are rendered with the negative X-axis always pointing towards the active view or camera.

Billboard

Faces with this attribute are pointing in the direction of the active view with the negative X-axis. It is different to "Halo" in that the faces are only rotated around the Z-axis.

Shadow

Faces with this attribute are projected onto the ground along the Z-axis of the object. This way they can be used to suggest the shadow of the object.

Text

Faces with this attribute are used for displaying bitmap-text in the game engine, see Section 25.4.

Opaque

Normal opaque rendered faces. The color of the texture is rendered as color.

Add

Faces are rendered transparent. The color of the face is added to what has already been drawn. Black areas in the texture are transparent, white are fully bright. Use this option to achieve light beam effects, glows or halos around lights. For real transparency use the next option "Alpha".

Alpha

The transparency depends on the alpha channel of the texture.

25.3.4. Avaible file formats

Blender uses OpenGL (http://www.opengl.org/) to draw its interface and the game engine. This way we can provide the such great cross-platform compatibility. In terms of using textures, we have to pay attention to several things before we're able to run the game on every Blender platform.

- The height and width of textures should be to the power of 64 pixels (e.g. 64x64, 64x128, 128x64 etc.) or Blender has to scale them (in memory not on disk!) to provide OpenGL compatibility
- The use of textures with a resolution above 256 x 256 pixels is not recommended if you plan on publishing your game, because not all graphic cards support higher resolutions.

Blender can use the following file formats as (real-time) textures:

Targa

The Targa or TGA (*.tga extension) file format is a lossless compressed format, which can include an alpha channel.

Iris

Iris (*.rgb) is the native IRIX image format. It is a lossless compressed file format, which can include an alpha channel.

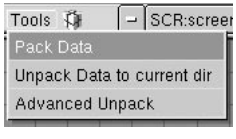
Jpeg

A lossy compressing (it uses a compression which leaves out parts of the image which the human eye can hardly see) file format (*.jpg, *.jpeg) is designed for photos with very small file sizes. Because of its small footprint it is a very good format for distribution over the net. It has no support for alpha channels and because of the quality loss due to compression it is not a recommended format to work with during the design phase of a game.

25.3.5. Handling of resources

For publishing and easier handling of Blender's files, you can include all resources into the scene. Normally textures, samples and fonts are not included in a file while saving. This keeps them on your disk and makes it possible to change them and share them between scenes. But if you want to distribute a file it is possible to pack these resources into the Blendfile, so you only need to distribute one file, preventing missing resources.

Figure 25-5. The ToolsMenu



The functions for packing and unpacking are summarized in the ToolsMenu. You can see if a file is packed if there is a little "parcel" icon to the right of the ToolsMenu. After you packed a file, all new added resources are automatically packed (AutoPack).

When working with textures, sounds or fonts you will notice a pack-icon near the File- or Datablock-Browse. This icon allows you to unpack the file independently.

The Tools Menu entries

Pack Data

This packs all resources into the Blendfile. The next save will write the packed file to disk.

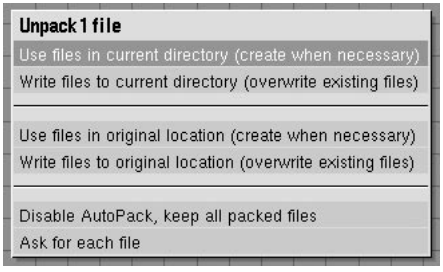
Unpack Data to current dir

This unpacks all resources to the current directory. For textures a directory "textures" is created, for sounds a "samples" directory and fonts are unpacked to "fonts".

Advanced Unpack

This option calls the Advanced Unpack Menu.

Figure 25-6. Advanced Unpack Menu



Advanced Unpack Menu entries

Use files in current directory

This unpacks only files which are not present in the current directory. It creates files when necessary.

Write files to current directory

This unpacks the files to the current directory. It overwrites existing files!

Use files in original location

This uses files from their original location (path on disk). It creates files when necessary.

Write files to original location

This writes the files to their original location (path on disk). It overwrites existing files!

Disable AutoPack, keep all packed files

This disables AutoPack, so new inserted resources are not packed into the Blendfile.

Ask for each file

This asks the user for the unpack options of each file.

25.4. Bitmap text in the game engine

Blender has the ability to draw text in the game engine using special bitmap fonts textures. These bitmap fonts can be created from a TrueType or a Postscript outline font. For an explanation of how to create a bitmap font look for the Tutorial How to create your own bitmap fonts (http://www.blender.nl/showitem.php?id=44) on the Blender site.

To get bitmap text or numbers displayed on a single face you need a special bitmap with the font rendered onto it. Then create a property named “Text” for your object and map the first character (“@”) of the text-bitmap on it. Check the “Text” face attribute for the face Paint/FaceButtons. The property can be any type, so a Boolean Property will also be rendered as “True” or “False”.

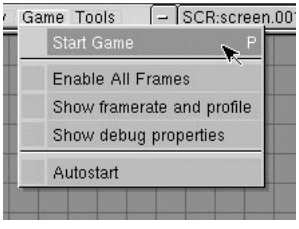
Chapter 26. Blenders game engine

Technically speaking the Blender game engine is a framework with a collection of modules for interactive purposes like physics, graphics, logic, sound and networking. Functionally the game engine processes virtual reality, consisting of content (the world, it's buildings) and behaviors (like physics, animation and logic). Elements in this world - also called GameObjects - behave autonomously by having a set of tools called LogicBricks, and Properties. For comparison, the Properties act as the memory, the Sensors are the senses, the Controllers are the brain and the Actuators allow for actions in the outside world (i.e. muscles).

At the moment, the Controllers can be scripted using python, or simple expressions. The idea is that the creation of logical behavior can be edited in a more visual way in the future, so the set of controllers expands with AI state machines, etc. Controllers could be split in control centers, like an audio visual center, motion center, etc.

26.1. Options for the game engine

Figure 26-1. The GameMenu



Options from the GameMenu

Start Game (PKEY)

Start the game engine, stop the engine with **ESC**, Blender will return to the Creator.

Enable All Frames

With this option checked the game engine runs without dropping frames. This is useful while recording to a Targa-Sequence or when you need to make sure that all collisions are calculated without loss on slower computers.

Show framerate and profile

With this menu option checked, the game engine will show some information on how fast the game runs and how the work is distributed.

Show debug properties

With this option checked, all Properties marked for debug output (D) are printed on screen while the game engine is running.

Autostart

Enable Autostart for the scene.

26.2. Options in the InfoWindow

In the InfoWindow, you can make your personal defaults for certain aspects of Blender. They will be saved with the Blender default scene when you press **CTRL-U**.

Figure 26-2. InfoWindow options

Blender game engine options in the InfoWindow

Vertex Arrays

Disable the use of vertexarrays. Vertexarrays normally speed up the calculation on complex scenes. If your *OpenGL* system does not support vertexarrays you can switch them off using this option.

No sound

Disable audio output.

No Mipmaps

Don`t use texture *Mipmap*, this can speedup the game but will result in not so nice rendered textures.

Python:

Here you can enter an additional path where the Python interpreter of Blender should search for modules.

26.3. Command line options for the game engine

When Blender is called with the option „-h“ on a command line (shell window or DOS window) it prints out the command line parameters.

Figure 26-3. Blender command line options0

```
[cw@work cw]$ blender -h
Blender V 2.24
Usage: blender [options ...] [file]

Render options:
  -b <file>     Render <file> in background
    -S <name>   Set scene <name>
    -f <frame>  Render frame <frame> and save it
    -s <frame>  Set start to frame <frame> (use with -a)
    -e <frame>  Set end to frame (use with -a)<frame>
    -a          Render animation

Animation options:
  -a <file(s)>  Playback <file(s)>
    -m          Read from disk (Don't buffer)

Window options:
  -w            Force opening with borders
  -p <sx> <sy> <w> <h>  Open with lower left corner at <sx>,
<sy>
                        and width and height <w>, <h>

Game Engine specific options:
  -g fixedtime          Run on 50 hertz without dropping frames
  -g vertexarrays       Use Vertex Arrays for rendering (usually
faster)
  -g noaudio            No audio in Game Engine
  -g nomipmap           No Texture Mipmapping
  -g linearmipmap       Linear Texture Mipmapping instead of
Nearest (default)
Misc options:
  -d            Turn debugging on
  -noaudio      Disable audio on systems that support audio
  -h            Print this help text
  -y            Disable OnLoad scene scripts, use -Y to find out
why its -y
[cw@work cw]$
```

Command line options for the Blender game engine

-g fixedtime

With this option the game engine runs without dropping frames. This is useful while recording to a Targa-Sequence or when you need to make sure that all collisions are calculated without loss on slower computers.

-g vertexarrays

Disable the use of vertexarrays. Vertexarrays normally speed up the calculation on complex scenes. If your *OpenGL* system doesn't support vertex arrays you can switch them off using this option.

-g noaudio

Disable audio.

-g nomipmap

Don`t use texture *Mipmap*, this can speedup the game but will result in not so nicely rendered textures.

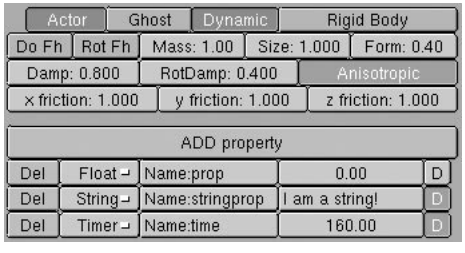
-g linearmipmap

Linear Texture mipmapping instead of nearest (default).

26.4. The RealtimeButtons

The RealtimeButtons are meant for making interactive 3-D worlds in Blender. Blender acts as a complete development tool for interactive worlds including a game engine to play the worlds. All this is done without compiling the game or interactive world. Just press **PKEY** and it runs in real-time. The main view for working with the Blender game engine are the RealtimeButtons ( ). Here you define your LogicBricks, which add the behavior to your objects.

Figure 26-4. RealtimeButtons left part



Info: *The word "games" is here used for all kinds of interactive 3D-content; Blender is not limited to making and play games*

The RealtimeButtons can logically be separated in two parts. The left part contains global settings for GameObjects.

This includes settings for general physics, like the damping or mass. Here you also define if an object should be calculated with the build-in physics, as an actor or should be handled as an object forming the level (like props on a stage).

Settings for GameObjects

Actor

Activating "Actor" for an object causes the game engine to evaluate this object. The Actor button will produce more buttons described below. Objects without the "Actor" button activated can form the level (like props on a stage) and are seen by other actors as well.

Ghost

Ghost objects that don't restitute to collisions, but still trigger a collision sensor.

Dynamic

With this option activated, the object follows the laws of physics. This option spawns new buttons that allow you to define the object's attributes in more detail.

Rigid Body

The "Rigid Body" button enables the use of advanced physics by the game engine. This makes it possible to make spheres roll automatically when they make contact with other objects and the friction between the materials is non-zero. The rigid body dynamics are a range of future changes to the game engine. Use the "Form:" factor to control the rolling speed.

Do Fh

This button activates the Fh mechanism (see Section 26.6). With this option you can create a floating or swimming behavior for actors.

Rot Fh

With this option set the object is rotated in such a way that the Z-axis points away from the ground when using the Fh mechanism.

Mass

The mass of a dynamic actor has an effect on how the actor reacts when forces are applied to it. You need a bigger force to move a heavier object. Note that heavier objects don't fall faster! It is the air drag that causes a difference in the falling speed in our environment (without air, e.g. on the moon, a feather and a hammer fall at the same speed). Use the "Damp" value to simulate air drag.

Size

The size of the bounding sphere. The bounding sphere determines the area with which collisions can occur. In future versions this will not be limited to spheres anymore.

Form

A form factor which gives you control over the behaiviour of “Rigid Body” objects.

Damp

General (movement) damping for the object. Use this value for simulating the damping an object receives from air or water. In a space scene you might want to use very low or zero damping, air needs a higher damping, use a very high damping to simulate water.

RotDamp

Same as “Damp” but for rotations of the object.

Anisotropic

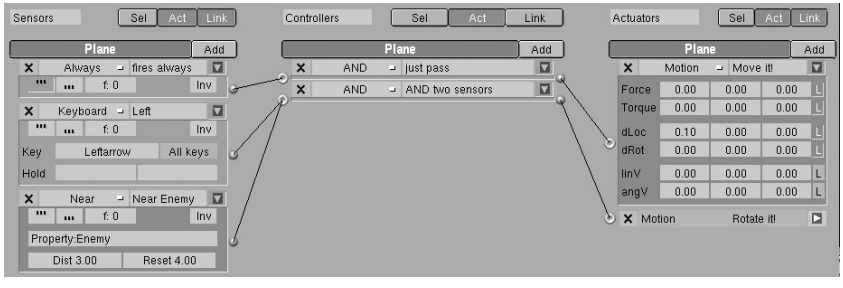
When an actor moves on a surface you can define a friction between the objects. Friction will slow down objects, because it is a force that works against any existing forces in the direction of the surface. It is controlled in the dynamic material settings (MaterialButtons F5, see Section 26.6). This friction works equally in all directions of movement.

With the “Anisotropic” option activated you can control the friction independently for the three axes. This is very helpful for racing games, where for example the car receives little friction in the driving direction (because of the rolling tires) and high friction sliding to the side .

Below the object settings you define the Properties of a GameObject. These Properties can carry values, which describe attributes of the object like variables in a programming language. Use “ADD property” to add properties (see Section 26.5).

The right part of the RealtimeButtons is the command center for adding logic to your objects and worlds. The logic consists of the Sensors, Controllers and Actuators.

Figure 26-5. Example of some LogicBricks



Sensors are like the senses of a life form; they react on key presses, collisions, contact with materials (touch), timer events or values of properties.

The Controllers are collecting events from the sensors and are able to calculate them to a result. These are similar to the mind or brain of a life form. Simple Controllers just do an AND on the inputs. An example is to test if a key is pressed AND a certain time has passed. There are also OR Controllers and you can also

use Python scripting and expressions in the Expression Controller to create more complex behavior.

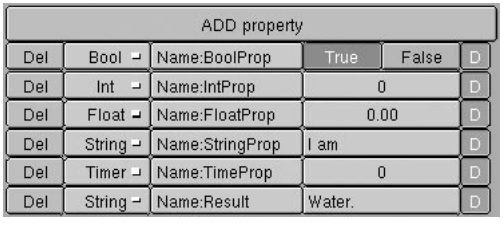
The Actuator actually performs actions on objects. A Motion Actuator for example is like a muscle. This muscle can apply forces to objects to move or rotate them. There are also Actuators for playing predefined animations (via Ipos), which can be compared to a reflex.

The logic is connected (wired) with the mouse, Sensors to Controllers and Controllers to Actuators. After wiring you are immediately able to play the game! If you discover something in the game you don't like, just stop the game engine, edit your 3-D world and restart. This way you can drastically cut down your development time!

26.5. Properties

Properties carry information bound to the object, similarly to a local variable in programming languages. No other object can normally access these properties, but it is possible to copy Properties with the Property Copy Actuator (see Section 27.3.7) or send them to other objects using messages (see Section 27.3.11).

Figure 26-6. Defining properties



The big "ADD property" button adds a new Property. By default a Property of the float type is added. Delete a Property with its "Del" button. The MenuButton defines the type of the Property. Click and hold it with the left mouse button and choose from the pop up menu. The "Name:" text field can be edited by clicking it with the left mouse button. **SHIFT-BACKSPACE** clears the name.

Note: *Property names are case sensitive. So "Erwin" is not equal to "erwin".*

The next field is different for each of the Property types. For the Boolean type there are two radio-buttons; choose between "True" and "False". The string-type accepts a string; enter a string by clicking in the field with the left mouse. The other types use a NumberButton to define the default value. Use **SHIFT-LMB** for editing it with the keyboard, click and drag to change the value with the mouse.

Property types

Boolean (Bool)

This Property type stores a binary value, meaning it can be “TRUE” or “FALSE”. Be sure to write it all in capitals when using these values in Property Sensors or Expressions.

Integer (Int)

Stores a number like 1,2,3,4,... in the range from -2147483647 to 2147483647.

Float

Stores a floating point number.

String

Stores a text string. You can also use Expressions or the Property Sensor to compare strings.

Timer

This Property type is updated with the actual game time in seconds, starting from zero. On newly created objects the timer starts when the object is “born”.

26.6. Settings in the MaterialButtons

Some physical attributes can be defined with the material settings of Blender. The MaterialButtons can be accessed via the  icon in the header of the ButtonsWindow or by pressing **F5**. Create a new material or choose an existing one with the MenuButton in the header.

In the MaterialButtons you need then to activate the “DYN” button to see the dynamic settings (See Figure 26-7).

Figure 26-7. Material settings for dynamic objects

Restitute

This parameter controls the elasticity of collisions. A value of 1.0 will convert all the kinetic energy of the object to the opposite force. This object then has an ideal elasticity. This means that if the other object (i.e. the ground) also has a Restitute of 1.0 the object will keep bouncing forever.

Friction

This value controls the friction of the objects material. If the friction is low, your object will slide like on ice, with a high friction you get the effect of

sticking in glue.

Fh Force

In conjunction with the "Do Fh" and/or "Rot Fh" (see Section 26.4) you make an object float above a surface.

"Fh Force" controls the force that keeps the object above the floor.

Fh Dist

"Fh Dist" controls the size of the Fh area. When the object enters this area the Fh mechanism starts to work.

Fh Damp

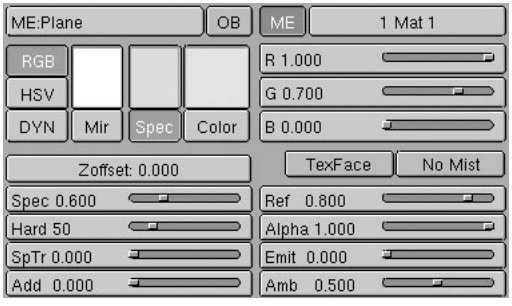
Controls the damping inside the Fh area. Values above 0.0 will damp the object movement inside the Fh area.

Fh Norm

With this button activated the object also gets a force in the direction of the face normal on slopes. This will cause an object to slide down a slope (see the example: FhDemo.blend (blends/FhDemo.blend)).

26.6.1. Specularity settings for the game engine

Figure 26-8. Specularity settings



Specularity settings in the MaterialButtons

Spec

This slider controls the intensity of the specularity.

Hard

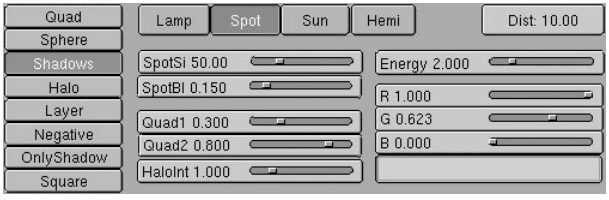
This slider controls the size of the specularity (hardness).

Spec color

Activating this button, switches the RGB (or HSV) sliders to define the specularity color.

26.7. Lamps in the game engine

Figure 26-9. LampButtons, settings for Blenders game engine



Lamps are created with the Toolbox (**SPACE**->ADD Lamp). For a selected lamp you can switch to the LampButtons (**F4**) to change the properties of that lamp. These properties are the color, the energy, etc. Due to the fact that the game engine is fully integrated in Blender, there are some buttons which are only useful for linear animation.

Common settings for all lamp types are the energy, and the color (adjustable with the RGB sliders).

To allow a face to receive real-time lighting in Blenders game engine, the face has to be set to "Light" in the Paint/FaceButtons (See Section 25.3). With the layer settings for lamps and objects (EditButtons, **F9**) you can control the lighting very precisely. Lamps only affect faces on the same layer(s) as the lamp. Per Layer you can use eight lamps (OpenGL limitation) for real-time lighting.

Lamp types for the game engine

Lamp

Lamp is a point light source.

Spot

This lamp is restricted to a conical space. In the 3DWindow the form of the spotlight is shown with broken lines. Use the SpotSi slider to set the angle of the beam.

Sun

The "Sun" lamp type is a directional light. The distance has no effect on the intensity. Change the direction of the light (shown as a broken line) by rotating the lamp.

Hemi

"Hemi" lamp type is currently not supported in the game engine.

The "Lamp" and "Spot" lights can be sensitive to distance. Use the "Dist:", "Quad1: " and "Quad2:" settings for this. The mathematics behind this are explained in the "Official Blender 2.0 Guide" (see Section 29.5).

26.8. The Blender laws of physics

All objects in Blender with the "Dynamic" option set (see Settings for GameObjects) are evaluated using the physics laws as defined by the game engine and the user.

The key property for a dynamic object is its mass. Gravity, forces, and impulses (collision bounce) only work on objects with a mass. Also, only dynamic objects can experience drag, or velocity damping (a crude way to mimic air/water resistance).

Note: *Note that for dynamic objects using dLoc and dRot may not have the desired result. Since the velocity of a dynamic object is controlled by the forces and impulses, any explicit change of position or orientation of an object may not correspond with the velocity. For dynamic objects it's better to use the linV and angV for explicitly defining the motion.*

As soon we have defined a mass for our dynamic object, it will be affected by gravity, causing it to fall until it hits another object with its bounding sphere. The size of the bounding-sphere can be changed with the "Size:" parameter in the RealtimeButtons. The gravity has a value of 9.81 by default: you can change this in the WorldButtons with the "Grav" slider. A gravity of zero is very useful for space games or simulations.

Note: *Use the "Damp:" and "RotDamp:" settings to suggest the drag of air or other environments. Don't use it to simulate friction. Friction can be simulated by using the dynamic material settings.*

Dynamic objects can bounce for two reasons. Either you have Do Fh enabled and have too little damping, or you are using a Restitute value in the dynamic material properties that is too high.

Note: *If you haven't defined a material, the default restitution is 1.0, which is the maximum value and will cause two objects without materials to bounce forever.*

In the first case, increasing the damping can decrease the amount of bounce. In the latter case define a material for at least one of the colliding objects, and set its Restitute value to a smaller value. The Restitute value determines the elasticity of the material. A value of zero denotes that the relative velocity between the colliding objects will be fully absorbed. A value of one denotes that the total momentum will be preserved after the collision.

Damping decreases the velocity in % per second. Damping is useful to achieve a maximum speed. The larger the speed the greater the absolute decrease of speed due to drag. The maximum speed is attained when the acceleration due to forces equals the deceleration due to drag. Damping is also useful for damping out unwanted oscillations due to springs.

Friction is a force tangent to the contact surface. The friction force has a maximum that is linear to the normal, i.e., the force that presses the objects against each

other, (the weight of the object). The Friction value denotes the Coulomb friction coefficient, i.e. the ratio of the maximum friction force and the normal force. A larger Friction value will allow for a larger maximum friction. For a sliding object the friction force will always be the maximum friction force. For a stationary object the friction force will cancel out any tangent force that is less than the maximum friction. If the tangent force is larger than the maximum friction then the object will start sliding.

For some objects you need to have different friction in different directions. For instance a skateboard will experience relatively little friction when moving it forward and backward, but a lot of friction when moving it side to side. This is called anisotropic friction. Selecting the "Anisotropic" button in the RealTimeButtons (**F8**) will enable anisotropic friction. After selecting this button, three sliders will appear in which the relative coefficient for each of the local axes can be set. A relative coefficient of zero denotes that along the corresponding axis no friction is experienced. A relative coefficient of one denotes that the full friction applies along the corresponding axis.

26.9. Expressions

Expressions can be used in the Expression Controller, the Property Sensor and the Property Actuator.

Table 26-1. Valid expressions

Expression type	**Example**
Integer numbers	15
Float number	12.23224
Booleans	TRUE, FALSE
Strings	"I am a string!"
Properties	propname
Sensornames	sensorname (as named in the LogicBrick)

Table 26-2. Arithmetic expressions

Expression	**Example**
EXPR1 + EXPR2	Addition, 12+3, propname+21
EXPR1 - EXPR2	Subtraction, 12-3, propname-21
EXPR1 * EXPR2	Multiplication, 12*3, propname*21
EXPR1 / EXPR2	Division, 12/3, propname/21
EXPR1 > EXPR2	EXPR1 greater EXPR2
EXPR1 >= EXPR2	EXPR1 greater or equal EXPR2
EXPR1 < EXPR2	EXPR1 less EXPR2

Table 26-3. Boolean operations

Operation	Example
NOT EXPR	Not EXPR
EXPR1 OR EXPR2	logical OR
EXPR1 AND EXPR2	logical AND
EXPR1 == EXPR2	EXPR1 equals EXPR2

Conditional statement: IF(Test, ValueTrue, ValueFalse)

Examples:

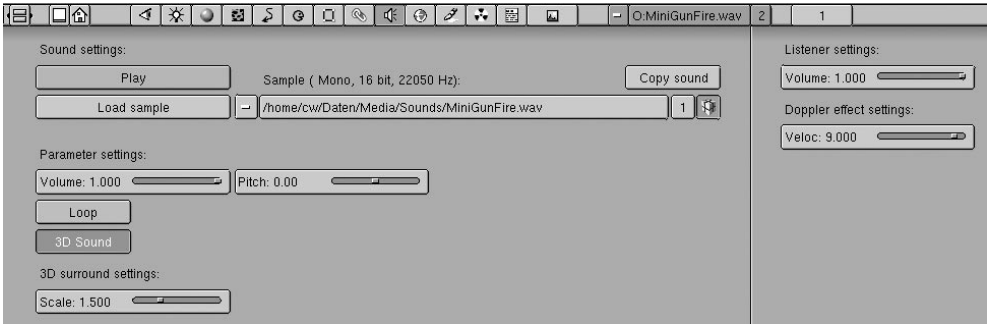
Table 26-4. Expression examples

Expression	Result	Explanation
12+12	24	Addition
	TRUE or FALSE	String comparison between a Property and a string
	TRUE	A string compare is done

26.10. SoundButtons

The SoundButtons  are used for loading and managing sounds for the Blender game engine. Look at Section 24.6 for a method to visualize the waveform.

Figure 26-10. The SoundButtons



In the SoundButtons Header you can see the name of the SoundObject (here "SO: MiniGunFire.wav"). This name is set to the name of the sound sample by default.

With the MenuButton you can browse existing SoundObjects and create new SoundObjects. The blue color of the sound name indicates that more than one user uses the sound, the number button indicates the number of users.

Listener settings

The “Listener settings” on the right side of the SoundButtons define global settings for the listener. The listener is the current camera. The “Volume:” slider sets the global volume of all sounds. The “Veloc:” slider controls the overall strength of the *Doppler effect*.

Sound settings

In the SoundSettings section you can then assign or load samples for the SoundObject. So the SoundObject name doesn’t have to be the name of the sample. For example you can use a SoundObject “SO:explosion” and then load “explosion_nuke.wav” later. You load samples using the “Load Sample” button in the SoundButtons. The sample name and the location on disk are shown in the text field to the right of the “Load Sample” button. Using the MenuButton to the left of the location, you can browse samples already loaded and assign one to the SoundObject.

Above the sample location Blender gives you some basic information about the loaded sample, like the sample frequency, 8 or 16bit and if the sample is Stereo or Mono.

The NumberButton indicates how many SoundObjects share the sample. When the pack/unpack button (parcel) is pressed, the sample is packed into the *.blend file, which is especially important when distributing files.

The “Play” button plays the sound, you can stop a playing sound with **ESC**.

The “Copy Sound” Button copies the SoundObject with all parameters.

Parameter settings

The “Vol:” slider sets the volume of the sample.

With the Pitch: value you can change the frequency of the sound. Currently there’s support for values between half the pitch (-12 semitones) and double the pitch (+12 semitones). Or in Hertz: if your sample has a frequency of 1000 Hz, the bottom value is 500 and the top 2000 Hz.

The “Loop” button sets the looping for the sample on or off. Depending on the play-mode in the Sound Actuator this setting can be overridden.

The “3D Sound” Button activates the calculation of 3-D sound for this SoundObject. This means the volume of the sound depends on the distance and position (stereo effect) between the sound source and the listener. The listener is the active camera.

The “Scale:” slider sets the sound attenuation. In a 3-D world you want to scale the relationship between gain and distance. For example, if a sound passes by the camera you want to set the scaling factor that determines how much the sound will gain if it comes towards you and how much it will diminish if it goes away from you.

The scaling factor can be set between 0.0. All positions get multiplied by zero, no matter where the source is, it will always sound as if it is playing in front of you (no 3-D Sound), 1.0 (a neutral state, all positions get multiplied by 1) and 5.0 which over accentuates the gain/distance relationship.

26.11. Performance and design issues

Computers get faster every month, nowadays nearly every new computer has a hardware accelerated graphics card. But still there are some performance issues to think about. This is not only a good design and programming style but also essential for the platform compatibility Blender provides. So to make a well-designed game for various platforms, keep these rules in mind:

1. Don't use properties in combination with AND/OR/Expr. controller as scripting language. Use the Python Controller.

2. Use as few inter-object LogicBrick connections as possible.

3. Use **ALT-D** (instanced mesh for new object) when replicating meshes, this is better than **SHIFT-D** (copies the mesh).

4. Alpha mapped polygons are expensive, so use with care.

5. Switching off the collision flag for polygons is good for performance. The use of "Ghost" is also cheaper then a regular physics object.

6. Keep the polygon count as low as possible. Its quite easy to add polygons to models, but very hard to remove them without screwing up the model. The detail should be made with textures.

7. Keep your texture-resolution as low as possible. You can work with hi-res versions and then later reduce them to publish the game (see Section 25.3).

8. Polygons set to "Light" are expensive. A hardware acceleration with a "Transform and Lighting" chip will help here.

9. Instead of real-time lighting use VertexPaint to lighten, darken or tint faces to suggest lighting situations.

Chapter 27. Game LogicBricks

The game logic in Blenders game engine is assembled in the RealtimeButtons. Here you wire the different LogicBricks together. The following is a brief description of all the LogicBricks currently available.

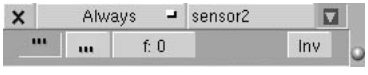
27.1. Sensors

Sensors act like real senses; they can detect collisions, feel (Touch), smell (Near), view (Ray, Radar).

27.1.1. Always Sensor

The most basic Sensor is the Always Sensor. It is also a good example for the common buttons every sensor has.

Figure 27-1. Common elements for Sensors



The button labeled “X” deletes the Sensor from the game logic. This happens without a confirmation, so be careful. The MenuButton to the right of the delete button (here labeled “Always”) allows you to choose the type of Sensor. Click and hold it with the left mouse button to get the pop up menu. Next is a TextButton, which holds the name of the Sensor. Blender assigns the name automatically on creation. Click the name with the left mouse button to change the name with the keyboard.

Note: *Name your LogicBricks and Blender objects to keep track of your scenes. A graphical logic scheme can become very complex.*

With the small arrow button you can hide the contents of the LogicBrick, so it only shows the top bar. This is very handy in complex scenes.

The next row of buttons is used to determine how and at which frequency a Sensor is “firing”. This topic is a bit complex, so we will give examples in more than one part of this documentation.

General things on pulses

Pulses coming from Sensors trigger both Controllers and Actuators. A pulse can have two values, TRUE or FALSE.

Each Controller is always evaluated when it receives a pulse, it doesn't matter whether the pulse is TRUE or FALSE. The input "gate" of a Controller remembers the last pulse value. This is necessary for Controllers being linked by multiple Sensors, then it can still do a logical AND or OR operation on all inputs. When a Controller is triggered, and after the evaluation of all inputs, it can either decide to execute the internal script or to send a pulse to the Actuators.

An Actuator reacts to a pulse in a different way, with a TRUE pulse it switches itself ON (makes itself active), with a FALSE pulse it turns itself OFF.

Figure 27-2. Pulse Mode Buttons

The first button activates the positive pulse mode. Every time the Sensor fires a pulse it is a positive pulse. This can be used, for example to start a movement with an Motion Actuator. The button next to it activates the negative pulse mode, which can be used to stop a movement.

Note: *If none of the pulse mode buttons are activated the Always Sensor fires exactly one time. This is very useful for initialising stuff at the start of a game.*

The button labeled "f:" (set to 41 here), determines the delay between two pulses fired by the Sensor. The value of "f:" is given as frames.

The "Inv" button inverts the pulse, so a positive (TRUE) pulse will become negative (FALSE) and vice versa.

VI

27.1.2. Keyboard Sensor

The Keyboard Sensor is one of the most often used Sensors because it provides the interface between Blender and the user.

The pulse mode buttons are common for every Sensor so they have the same functionality as described for the Always Sensor.

By activating the "All keys" Button, the Sensor will react to every key. In the "Hold" fields you can put in modifier keys, which need to be held while pressing the main key.

The Keyboardsensor can be used for simple text input. To do so, fill in the Property which should hold the typed text (you can use **Backspace** to delete chars) into the "Target:" field. The input will be active as long the Property in "LogToggle:" is "TRUE".

Python methods:

Import the Gamekeys module (see Section 28.3.3) to have symbolic names for the keys.

setKey(*int key*);

Sets the Key on which the Sensor reacts

int key **getKey**( );

Gets the key on which the Sensor reacts

setHold1(*int key*);

Sets the first modifier key

int key **getHold1**( );

Gets the first modifier key

setHold2(*int key*);

Sets the second modifier key

int key **getHold2**( );

Gets the second modifier key

list keys **getPressedKeys**( );

Gets the keys (including modifier)

list keys **getCurrentlyPressedKeys**( );

Gets the keys (including modifier) currently held

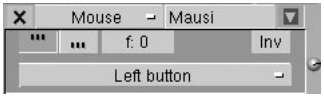
27.1.3. Mouse Sensor

Currently, the Sensor is able to watch for mouse clicks, mouse movement or a mouse over. To get the position of the mouse cursor as well you need to use a Python-script.

Use the MenuButton to choose between the Mouse Sensor types:

Mouse Sensor types

Left/Middle/Right Button



The sensor gives out a pulse when the correlate mouse button is pressed.

Python methods:

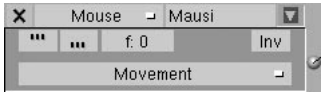
```
int xpos getXPosition( );
```

Gets the mouse's X-position

```
int ypos getYPosition( );
```

Gets the mouse's Y-position

Movement



The sensor gives out a pulse when the mouse is moved.

Python methods:

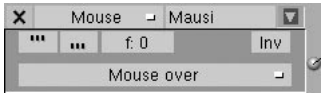
```
int xpos getXPosition( );
```

Gets the mouse x-position

```
int ypos getYPosition( );
```

Gets the mouse y-position

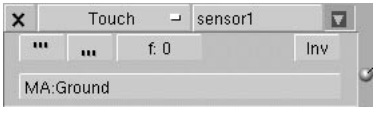
Mouse over



The sensor gives a pulse when the mouse cursor is over the object.

27.1.4. Touch Sensor

The Touch Sensor fires a pulse when the object it is assigned to, touches a material. If you enter a material name into the „MA:" text field it only reacts to this material otherwise it reacts to all touch.



Python methods:

The Touchsensor inherits from the Collision Sensor, so it provides the same Methods, hence the Methode names.

```
setProperty( (char* matname) );
```

Sets the Material the Touch Sensor should react to

```
char* matname getProperty( );
```

Gets the Material the Touch Sensor reacts to

```
gameObject obj getHitObject( );
```

Returns the touched Object

```
list objs getHitObjectList( );
```

Returns a list of touched objects

27.1.5. Collison Sensor

The Collision Sensor is a general Sensor used to detect contact between objects. Besides reacting to materials it is also capable of detecting Properties of an object. Therefore you can switch the input field from Material to Property by clicking on the „M/P“ button.

Python methods:

```
setProperty( (char* name) );
```

Sets the Material or Property the Collision Sensor should react to

```
char* name getProperty( );
```

Gets the Material or Property the Collision Sensor reacts to

```
gameObject obj getHitObject( );
```

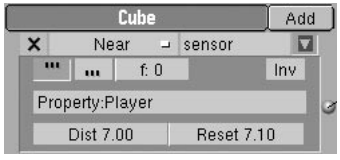
Returns the colliding Object

```
list objs getHitObjectList( );
```

Returns a list of objects that have collided

27.1.6. Near Sensor

The near sensor reacts to actors near the object with the sensor.



Note: *The near sensor only senses objects of the type "Actor" (a dynamic object is also an actor).*

If the "Property:" field is empty, the near sensor reacts to all actors in its range. If filled with a property name, the sensor only reacts to actors carrying a property with that name.

The spherical range of the near sensor you set with the "Dist" NumberButton. The "Reset" value defines at what distance the near sensor is reset again.

Python methods:

```
setProperty( (char* propname) );
```

Sets the Property the Near Sensor should react to

```
char* propname getProperty( );
```

Gets the Property the Near Sensor reacts to

```
list gameObjects getHitObjectList( );
```

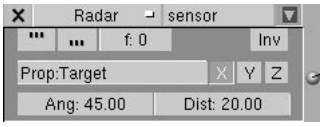
Returns a list of game objects detected by the Near Sensor.

```
gameObject obj getHitObject( );
```

Returns the object which triggered the Sensor

27.1.7. Radar Sensor

The Radar Sensor acts like a real radar. It looks for an object along the axis indicated with the axis buttons „X, Y, Z". If a property name is entered into the „Prop:" field, it only reacts to objects with this property.



In the "Ang:" field you can enter an opening angle for the radar. This equals the angle of view for a camera. The "Dist:" setting determines how far the Radar Sensor can see.

Objects can't block the line of sight for the Radar Sensor. This is different for the Ray Sensor (see Section 27.1.10). You can combine them for making a radar that is not able to look through walls.

Python methods:

```
setProperty( (char* name) );
```

Sets the Property that the Radar Sensor should react on

```
char* name getProperty( );
```

Gets the name of the Property

```
gameObject obj getHitObject( );
```

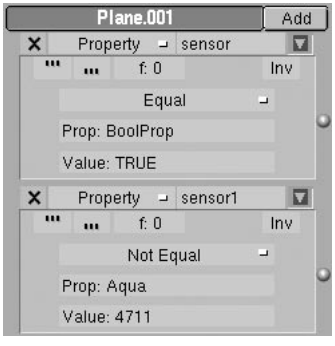
Returns the detected object that triggered the sensor

```
list objs getHitObjectList( );
```

Returns a list of detected objects

27.1.8. Property Sensor

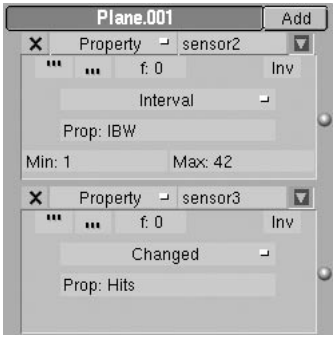
The Property Sensor logically checks a Property attached to the same object.



The "Equal" type Property Sensor checks for equality of the property given in

the “Prop:” field and the value in “Value:”. If the condition is true, it fires pulses according to the pulse mode settings.

The “Not Equal” Property Sensor checks for inequality and then fires its pulses.



The “Interval” type property sensor fires its pulse if the value of property is inside the interval defined by “Min:” and “Max:”. This sensor type is especially helpful for checking float values, which you can’t depend on to reach a value exactly. This is most common with the “Timer” Property.

The “Changed” Property Sensor gives out pulses every time a Property is changed. This can, for example, happen through a Property Actuator, a Python script or an Expression.

Python methods:

setProperty((char* propname));

Sets the Property to check

char* propname **getProperty**( );

Gets the Property to check

setType((int type));

Sets the type of the Property Sensor.

1. Equal
2. Not Equal
3. Interval
4. Changed

char* propname **getProperty**( );

Gets the type of the Property Sensor

```
setValue( (char* expression) );
```

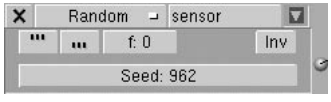
Sets the value to check (as expression)

```
char* expression getValue( );
```

Gets the value to check (as expression)

27.1.9. Random Sensor

The Random Sensor fires a pulse randomly according to the pulse settings (50/50 pick).



Note: *With a seed of zero the Random Sensor works like an Always Sensor, which means it fires a pulse every time.*

Python methods:

```
setSeed( (int seed) );
```

Set the seed for the random generation

```
int seed getSeed( );
```

Gets the seed for the Random Sensor

```
int seed getLastDraw( );
```

Gets the last draw from the Random Sensor

27.1.10. Ray Sensor

The Ray Sensor casts a ray for the distance set into the NumberButton „Range“. If the ray hits an object with the right Property or the right Material the Sensor fires its Pulse.

Note: *Other objects block the ray, so that it can't see through walls.*

Without a material or property name filled in, the Ray Sensor reacts to all objects.

Python methods:

`list [x,y,z]` **`getHitPosition`**`( );`

Returns the position where the ray hits the object.

`list [x,y,z]` **`getHitNormal`**`( );`

Returns the normal vector how the ray hits the object.

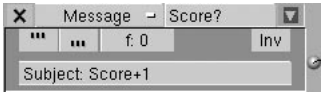
`list [x,y,z]` **`getRayDirection`**`( );`

Returns the vector of the Ray direction

`gameObject obj` **`getHitObject`**`( );`

Returns the hit Object

27.1.11. Message Sensor



The Message Sensor fires its pulse when a Message arrives for the object carrying the Sensor. The "Subject:" field can be used to filter messages matching the subject.

Python methods:

`list bodies` **`getBodies`**`( );`

Returns a list containing the message bodies arrived since last call

`int messages` **`getFrameMessageCount`**`( );`

Returns the number of messages received since the last frame

`setSubjectFilterText``(` *`string subject`* `);`

Sets the subject for the Message Sensor

27.2. Controllers

Controllers act as the brain for your game logic. This reaches from very simple decisions like connecting two inputs, simple expressions, to complex Python scripts which can carry artificial intelligence.

27.2.1. AND Controller

The AND Controller combines one, two or more inputs from Sensors. That means that all inputs must be active to pass the AND Controller.

27.2.2. OR Controller

The OR Controller combines one, two or more inputs from Sensors. OR means that either one or more inputs can be active to let the OR Controller pass the pulse through.

27.2.3. Expression Controller

With the Expression Controller you can create slightly complex game logic with a single line of „code“. You can access the output of sensors attached to the controller and access the properties of the object.

Note: *The expression mechanism prints out errors to the console or in the DOS window, so have a look there if anything fails.*

More on using Expressions can be found in Section 26.9.

27.2.4. Python Controller

The Python controller is the most powerful controller in game engine. You can attach a Python script to it, which allows you to control your GameObjects, ranging from simple movement up to complex game-play and artificial intelligence.

Enter the name of the script you want to attach to the Python Controller into the "Script:" field. The script needs to exist in the scene or Blender will ignore the name you type.

Note: *Remember that Blender treats names as case sensitive! So the script "player" is not the same as "Player".*

Python for the game engine is covered in Chapter Section 28.2

Python methods:

```
Actuator* getActuator( char* name , );
```

Returns the actuator with "name".

```
list getActuators( );
```

Returns a python list of all connected Actuators.

```
Sensor* getSensor( char* name , );
```

Returns the Sensor with "name".

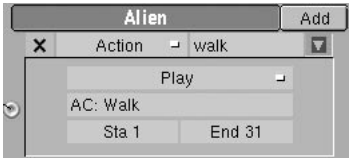
```
list getSensors( );
```

Returns a python list of all connected Sensors.

27.3. Actuators

Actuators are the executing LogicBricks. They can be compared with muscles or glands in a life form.

27.3.1. Action Actuator



Action play modes

Play

Plays the Action from „Sta“ to „End“ at every positive pulse the Actuator gets. Other pulses while playing are discarded.

Flipper

Plays the Action from „Sta“ to „End“ on activation. When the activation ends it plays backwards from the current position. When a new activation reaches the Actuator the Action will be played from the current position onwards.

Loop Stop

Plays the Action in a loop as long as the pulse is positive. It stops at the current position when the pulse turns negative.

Loop End

This plays the Action repeatedly as long as there is a positive pulse. When the pulse stops it continues to play the Action to the end and then stops.

Property

Plays the Action for exactly the frame indicated in the property entered in the field „Prop:“.

27.3.2. Motion Actuator

The Motion Actuator is maybe the most important Actuator. It moves, rotates or applies a velocity to objects.

The simplest case of using a Motion Actuator is to move the object. This is done with the “dLoc” values in the third row. Every time the actuator is triggered by an impulse it moves the object by the amount given in the “dLoc” row. The three values here stand for X-, Y- and Z-axis. So when you enter a 1.0 in the first field the object

is moved one unit per time unit of the game (the clock in the game engine ticks in frames, roughly 1/25 of a second, for exact timings use the Timer Property).

The buttons labeled "L" behind each row in the motion actuator, determine if the motion applied should be treated as global or local. If the button is pushed (dark green) the motion is applied based on the local axis of the object. If the button is not pressed the motion is applied based on the global (world) axis.

Force

Values in this row act as forces that apply to the object. This only works for dynamic objects.

Torque

Values in this row act as rotational forces (Torque) that apply to the object. This works only for dynamic objects. Positive values rotate counter-clock-wise.

dLoc

Offset the object as indicated in the value fields

dRot

Rotate the object for the given angle (36 is a full rotation). Positive values rotate clock-wise.

linV

Sets (overrides current velocity) the velocity of the object to the given values. When "add" is activated the velocity is added to the current velocity.

angV

Sets the angular velocity to the given values. Positive values rotate counter-clock-wise.

The Motion Actuator starts to move objects on a pulse (TRUE) and stops on a FALSE pulse. To get a movement over a certain distance, you need to send a FALSE pulse to the motion actuator after each positive pulse.

Python methods:

setForce(*float x , float y , float z , bool local*);

Sets the "Force" parameters for the Motion Actuator.

list [x,y,z,local] **getForce**( );

Gets the "Force" parameter from the Motion Actuator, local indicates if the local button is set (1).

setTorque(*list [x,y,z]*);

Sets the «Torque» parameter for the Motion Actuator.

`list [x,y,z] getTorque( );`

Gets the «Torque» parameter for the Motion Actuator.

`setdLoc( list [x,y,z] );`

Sets the dLoc parameters from the Motion Actuator.

`list [x,y,z] getdLoc( );`

Gets the dLoc parameters from the Motion Actuator.

`setdRot( list [x,y,z] );`

Sets the dRot parameters for the Motion Actuator.

`list [x,y,z] getdLoc( );`

Gets the dRot parameters from the Motion Actuator.

`setLinearVelocity( list [x,y,z] );`

Sets the linV parameters for the Motion Actuator.

`list [x,y,z] getLinearVelocity( );`

Gets the linV parameters from the Motion Actuator.

`setAngularVelocity( list [x,y,z] );`

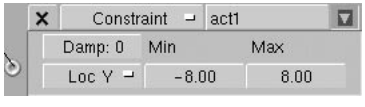
Sets the angV parameters for the Motion Actuator.

`list [x,y,z] getAngularVelocity( );`

Gets the angV parameters from the Motion Actuator.

27.3.3. Constraint Actuator

With the Constraint Actuator you can limit an object's freedom to a certain degree.



With the MenuButton you specify which channel's freedom should be constrained. With the NumberButtons "Min" and "Max" you define the minimum and maximum values for the constraint selected. To constrain an object to more than one channel simply use more than one Constraint actuator.

Python methods:

setDamp(*int damp*);
Sets the Damp parameter

int damp **getDamp**( );
Gets the Damp parameter

setMin(*int min*);
Sets the Min parameter

int min **getMin**( );
Gets the Min parameter

setMax(*int max*);
Sets the Max parameter

int max **getMax**( );
Gets the Max parameter

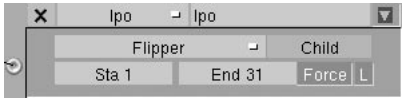
setMin(int min);
Sets the Min parameter

int min **getMin**( );
Gets the Min parameter

setLimit(*int limit*);
Sets the limit for the constraint. None = 1, LocX = 2, LocY = 3, LocZ = 4

int limit **getLimit**( );
Gets the constraint

27.3.4. Ipo Actuator



The Ipo Actuator can play the Ipo-curves for the object that owns the Actuator. If the object has a child with an Ipo (in a parenting chain) and you activate “Child” in the Actuator, the Ipo for the child is also played.

The “Force” Button will convert the “Loc” Ipo curves into forces for dynamic objects. When pressed, the “L” Button appears which cares for applying the forces locally to the objects coordinate system.

Ipo play modes

Play

Plays the Ipo from „Sta“ to „End“ at every positive pulse the Actuator gets. Other pulses received while playing are discarded.

Ping Pong

Plays the Ipo from „Sta“ to „End“ on the first positive pulse, then backwards from „End“ to „Sta“ when the second positive pulse is received.

Flipper

Plays the Ipo for as long as the pulse is positive. When the pulse changes to negative the Ipo is played from the current frame to „Sta“.

Loop Stop

Plays the Ipo in a loop for as long as the pulse is positive. It stops at the current position when the pulse turns negative.

Loop End

This plays the Ipo repeatedly for as long as there is a positive pulse. When the pulse stops it continiues to play the Ipo to the end and then stops.

Property

Plays the Ipo for exactly the frame indicated by the property named in the field „Prop:“.

Currently the following Ipos are supported by the game engine:

Mesh Objects

Loc, Rot, Size and Col

Lamps

Loc, Rot, RGB, Energy

Cameras

Loc, Rot, Lens, ClipSta, ClipEnd

Python methods:

SetType(*int type*);

Sets the type, 1 indicates the first play type from the menu button etc.

int type **GetType**( );

Sets the type, 1 indicates the first play type from the menu button

SetStart(*int frame*);

Sets the Sta: frame

SetEnd(*int frame*);

Sets the End: frame

int frame**GetStart**( );

Gets the Sta: frame

int frame**GetEnd**( );

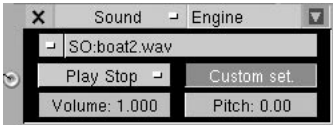
Gets the End: frame

27.3.5. Camera Actuator

The Camera Actuator tries to mimic a real cameraman. It keeps the actor in the field of view and tries to stay at a certain distance from the object. The motion is soft and there is some delay in the reaction on the motion of the object.

Enter the object that should be followed by the camera (you can also use the Camera Actuator for non-camera objects) into the "OB:" field. The field "Height:" determines the height above the object the camera stays at. "Min:" and "Max:" are the bounds of distance from the object to which the camera is allowed to move. The "X" and "Y" buttons specify which axis of the object the camera tries to stay behind.

27.3.6. Sound Actuator



The Sound Actuator plays a SoundObject loaded using the SoundButtons (see Section 26.10). Use the MenuButton to browse and choose between the SoundObjects in the scene.

Sound play modes (MenuBut)

Play Stop

Plays the sound for as long as there is a positive pulse.

Play End

Plays the sound to the end, when a positive pulse is given.

Loop Stop

Plays and repeats the sound, when a positive pulse is given.

Loop End

Plays the sound repeatedly, when a positive pulse is given. When the pulse stops the sound is played to its end.

Custom set. (TogBut)

Checking the „Custom set.“ button will copy the SoundObject (sharing the sample data) and allows you to quickly change the volume and pitch of the sound with the appearing NumberButtons.

Python methods:

`float gain` **`getGain`**`( );`

Get the gain (volume) setting.

`setGain``( float gain );`

Set the gain (volume) setting.

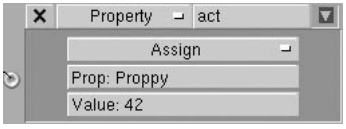
`float pitch` **`getPitch`**`( );`

Get the pitch setting.

`setPitch``( float pitch );`

Set the pitch setting.

27.3.7. Property Actuator



Property modes

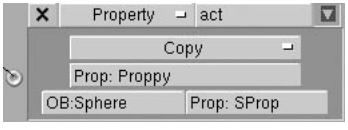
Assign

Assigns a value or Expression (given in the „Value“ field) to a Property. For example with an Expression like „Proppy + 1“ the „Assign“ works like an „Add“. To assign strings you need to add quotes to the string („...“).

Add

Adds the value or result of an expression to a property. To subtract simply give a negative number in the „Value:“ field.

Copy



This copies a Property (here „Prop: SProp“) from the Object with the name given in „OB: Sphere“ into the Property „Prop: Proppy“. This is an easy and safe way to pass information between objects. You cannot pass information between scenes with this Actuator!

Python methods:

```
SetProperty( string name );
string name GetProperty( );
SetValue( string value );
string value GetValue( );
```

More on using Expressions can be found in Section 26.9.

27.3.8. Edit Object Actuator

This actuator performs actions on Objects itself, like adding new objects, deleting objects, etc.

Edit Object Actuator types

Add Object

The Add Object actuator adds an object to the scene. The new object is oriented along the X-axis of the creating object.

Info: *Keep the object you'd like to add on a seperate and hidden layer. You will see an error message on the console or debug output when not following this rule.*

Enter the name of the Object to add in the „OB:" field. The „Time:" field determines how long (in frames) the object should exist. The value „0" denotes it will exist forever. Be careful not to slow down the game engine by generating too many objects! If the time an object should exist is not predictable, you can also use other events (collisions, properties, etc.) to trigger an „End Object" for the added object using LogicBricks.

With the "linV" buttons it is possible to assign an initial velocity to the added object. This velocity is given in X, Y and Z components. The "L" button stands for local. When it is pressed the velocity is interpreted as local to the added object.

Python methods:

setObject(*string name*);

Sets the Object (name) to be added

string name **getObject**( );

Gets the Object name

setTime(*int time*);

Time in frames the added Object should exist. Zero means unlimited

int time **getTime**( );

Gets the time the added object should exist

setLinearVelocity(*list [vx,vy,vz]*);

Sets the linear velocity [Blenderunits/sec] components for added Objects.

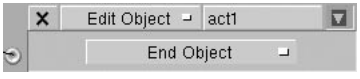
```
list [vx,vy,vz] getLinearVelocity( );
```

Gets the linear velocity [Blenderunits/sec] components from the Actuator

```
gameObject* getLastCreatedObject( );
```

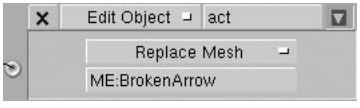
Gets a pointer to the last created object. This way you can manipulate dynamically added objects.

End Object



The "End Object" type simply ends the life of the object with the actuator when it gets a pulse. This is very useful for ending a bullet's life after a collision or something similar.

Replace Mesh



The "Replace Mesh" type, replaces the mesh of the object by a new one, given in the "ME:" field. Remember that the mesh name is not implicitly equal to the objectname.

Python methods:

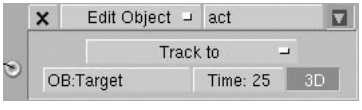
```
setMesh( string name );
```

Sets the Mesh for the ReplaceMesh Actuator to "name"

```
string name getMesh( );
```

Gets the Mesh-name from the ReplaceMesh actuator

Track to



The "Track to" type, rotates the object in such a way that the Y-axis points to the target specified in the "OB:" field. Normally this happens only in the X/Y plane of the object (indicated by the "3D" button not beeing pressed). With "3D" pressed the

tracking is done in 3D. The “Time:” parameter sets how fast the tracking is done. Zero means immedeately, values above zero produce a delay (are slower) in tracking.

Python methods:

```
setObject( string name );

string name getObject( );

setTime( int time );
```

Sets the time needed to track

```
int time getTime( );
```

Gets the time needed to track

```
setUse3D( bool 3d );
```

Set if “3D” should be used leading to full 3-D tracking

27.3.9. Scene Actuator

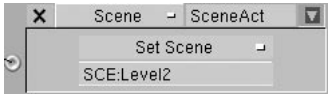
The Scene Actuator is meant for switching Scenes and Cameras in the game engine or adding overlay or background scenes.

Choose the desired action with the MenuButton and enter an existing camera or scene name into the text field. If the name does not exist, the button will be blanked!

Reset

Simply restarts and resets the scene. It has the same effect as stopping the game with **ESC** and restarting with **PKEY**.

Set Scene



Switch to the scene indicated into the text field. During the switch all properties are reset!

Python methods for all types of Scene Actuators:

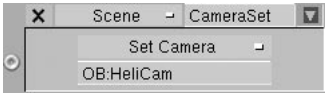
setScene(*char* scene*);

Sets the Scene to switch to

char* scene **getScene**( );

Gets the Scene name from the Actuator

Set Camera



Switch to the Camera indicated in the text field.

Python methods:

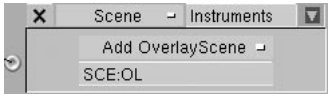
setCamera(*char* camera*);

Sets the Camera to switch to

char* camera **getCamera**( );

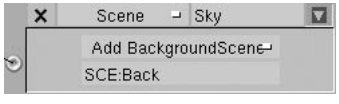
Gets the Camera name from the Actuator

Add OverlayScene



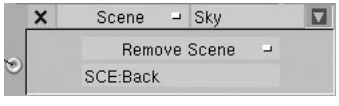
Adds an overlay scene which is rendered on top of all other (existing) scenes.

Add BackgroundScene



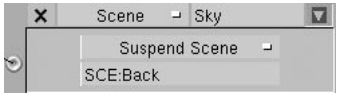
Adds a background scene which will be rendered behind all other scenes.

Remove Scene



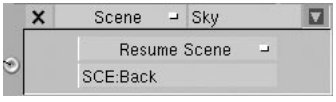
Removes a scene.

Suspend Scene



Suspends a scene until “Resume Scene” is called.

Resume Scene



Resumes a suspended Scene.

27.3.10. Random Actuator

An often-needed function for games is a random value to get more variation in movements or enemy behavior.

The Seed parameter is the value fed into the random generator as a start value for the random number generation. Because computer generated random numbers are only “pseudo” random (they will repeat after a (long) while) you can get the same random numbers again if you choose the same Seed.

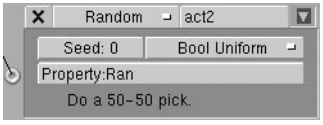
Enter the name of the property you want to be filled with the random number into the “Property:” field.

Random Actuators types

Boolean Constant

This is not a random function at all, use this type to test your game logic with a TRUE or FALSE value.

Boolean Uniform

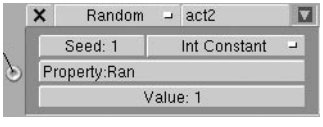


This is the classic random 50-50 pick. It results in TRUE or FALSE with an equal chance. This is like an (ideal) flip of a coin.

Boolean Bernoulli

This random function results in a boolean value of TRUE or FALSE, but instead of having the same chance for both values you can control the chance of having a TRUE pick with the „Chance“ parameter. A chance of 0.5 will be the same as „Bool Uniform“. A chance of 0.1 will result in 1 out of 10 cases in a TRUE (on average).

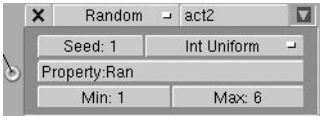
Integer Constant



For testing your logic with a value given in the „Value:“ field

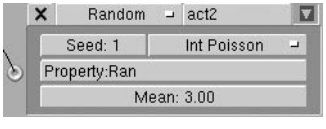
VI

Integer Uniform



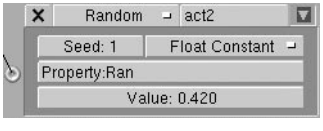
This random type randomly produces an integer value between (and including) „Min:“ and „Max:“. The classical use for it is to simulate a dice pick with „Min: 1“ and „Max: 6“.

Integer Poisson



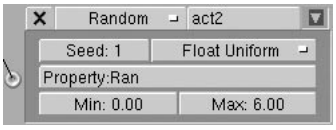
The random numbers are distributed in such a way that an average of „Mean:“ is reached with an infinite number of picks.

Float Constant



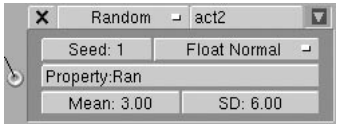
For debugging your game logic with a given value.

Float Uniform



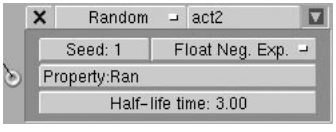
This returns a random floating point value between „Min:“ and „Max:“.

Float Normal



Returns a weighted random number around „Mean:“ and with a standard deviation of „SD:“.

Float Negative Exponential



Returns a random number which is well suited to describe natural processes like radioactive decay or lifetimes of bacteria. The „Half-life time:“ sets the average value of this distribution.

Python methods:

setSeed(*int seed*);

Sets the random seed (the init value of the random generation)

int seed **getSeed**( );

Gets the random seed (the init value of the random generation) from the Actuator

float para1 **getPara1**( );

Gets the first parameter for the selected random distribution

float para2 **getPara2**( );

Gets the second parameter for the selected random distribution

setProperty(*string propname*);

Sets the Property to which the random value should go

string propname **getProperty**( );

Gets the Property name from the Actuator

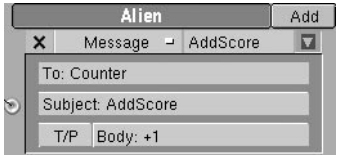
setDistribution(*int dist*);

Set the distribution, “dist = 1” means the fist choice from the type MenuButton

int dist **getDistribution**( );

Gets the random distribution method from the Actuator

27.3.11. Message Actuator



This LogicBrick sends a message out, which can be received and processed by the Message Sensor.

The “To:” field indicates that the message should only be sent to objects with the Property indicated by “To:”. The subject of the message is indicated in the “Subject: ” field. With these two possibilities you can control the messaging very effectively.

The body (content) of the message can either be a text (“Body:”) string or the content of a Property when “T/P” is activated (“Propname:”). See Section 27.1.11 on how to get the body of a message.

Python methods:

setToPropName(*char* propname*);

Sets the property name the message should be send to.

setSubject(*char* subject*);

Sets the subject of the message.

setBody(*char* body*);

Sets the body of the message.

setBodyType(*int bodytype*);

Sets whether the body should be text or a Property name.

Chapter 28. Python

Python (http://www.python.org/) is an interpreted, interactive, object-oriented programming language.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. Python is also usable as an extension language for applications that need a programmable interface.

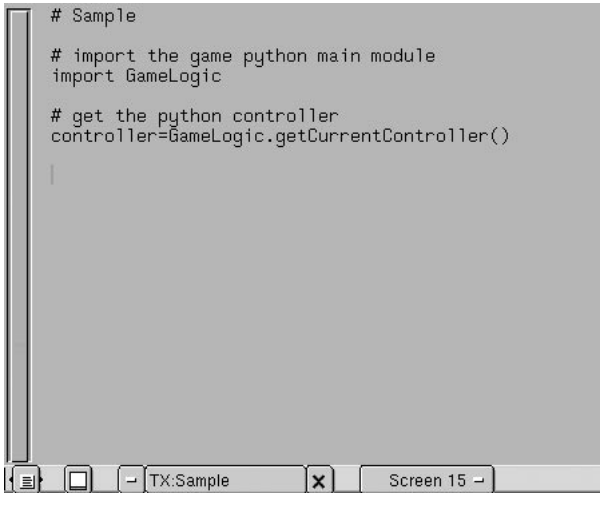
Beside this use as an extension language, the Python implementation is portable to (at least) all platforms that Blender runs on.

Python is copyrighted but freely usable and distributable, even for commercial use.

28.1. The TextWindow

The TextWindow is a simple but useful text editor, fully integrated into Blender. It's main purpose of it is to write Python scripts, but it is also very useful for writing comments in the Blendfile or to explain the purpose of the scene to other users.

Figure 28-1. The TextWindow



The TextWindow can be displayed with **SHIFT-F11** or by adjusting the IconMenu in the WindowHeader. As usual there is an IconBut to make the TextWindow fullscreen, the next MenuButton can be used to switch between text files, open new ones or add new text buffers. The "X"-shaped Button deletes a textbuffer after a confirmation.

With the MenuButton on the right side you can change the font used to display the text.

By holding **LMB** and then dragging the mouse you can mark ranges of text for the usual cut, copy & paste functions. The key commands are:

Keycommands for the TextWindow

ALT-C

Copies the marked text into a buffer

ALT-X

Cuts out the marked text into a buffer

ALT-V

Pastes the text from buffer to the cursor in the TextWindow

ALT-O

Loads a text, a FileWindow appears

CTRL-R

Reloads the current text, very useful for editing with an external editor

SHIFT-ALT-F

Pops up the Filemenu for the TextWindow

ALT-F

Find function

ALT-J

Pops up a NumButton where you can specify a line number that the cursor will jump to

ALT-U

Unlimited Undo for the TextWindow

ALT-R

Redo function, recovers the last Undo

ALT-A

Marks the whole text

28.2. Python for games

With Python integrated into the game engine you can influence LogicBricks, change their parameters and react to events triggered by the LogicBricks.

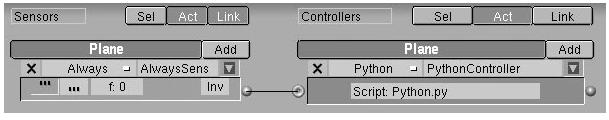
Besides that you can influence the GameObject that carries the Python Controller directly. This means moving it, applying forces or getting information from this object.

Info: *In addition to the Python in the game engine, Blender includes Python for modeling and animation tasks.*

28.2.1. Basic gamePython

The first step for using gamePython is to add at least a Sensor and a Python Controller to an object. Then add a new text file in the TextWindow. Fill in the name of that text file into the „Script:“ field of the Python Controller. You should now have a game logic setup like in Figure 28-2.

Figure 28-2. LogicBricks for a first gamePython script.



Now enter the following script into the TextWindow (you don't need to type the lines starting with "#", these are comments).

Figure 28-3. First Script

```
 1 # first gamePython script
 2 # gets the position of the owning object
 3 # and prints it on the console
 4
 5 import GameLogic
 6
 7 controller = GameLogic.getCurrentController()
 8 owner = controller.getOwner()
 9
10 print owner.getPosition()
```

The "print" command and errors from the Python interpreter will appear on the console from which you started Blender from, or in the DOS window, when running Blender under Windows. So it is helpful to size the Blender window in such a way that you can see the console window while programming Python.

This basic script only prints the position of the object that owns the Python Controller. Move your object and then restart the game engine with the **PKEY** to see the results changing.

Now we explain the function of the script line by line. Line five is maybe the most important line here. We import the "GameLogic" module which is the basis for all game Python in Blender.

In line seven we get the Controller, which executes the script and assigns it to the variable "controller".

In line eight we use the controller we got in line seven to get the owner, the

GameObject carrying the LogicBrick. You can see we use the method "getOwner()" to get the owner of our controller.

We now have the owner and we can use its methods to do things with it. Here in line 10 we use the "getPosition()" method to print the position of the gameObject as a matrix of the X, Y and Z values.

You may now wonder what other methods the PythonObjects have. Of course this is part of this documentation, but Python is "self" documenting, so we have other ways to get that information.

Add the following line to the end of the script from Figure 28-3:

```
1 print dir(owner)
```

Start the game engine again, stop it and look at the console window. You will see the following output:

[0.0, 0.0, 0.0]

['applyImpulse', 'disableRigidBody','enableRigidBody', 'getLinearVelocity', 'getMass', 'getOrientation', 'getPosition', 'getReactionForce','getVelocity', 'restoreDynamics', 'setOrientation', 'setPosition', 'setVisible', 'suspendDynamics']

The first line shows the position of the object, the next lines show the methods, that the "owner" provides. For example you see a 'getMass' method, which will return the mass of a dynamic object. With the knowledge of the "dir()" function you can ask Python objects for information, without consulting external documentation.

28.3. Game Python Documentation per module

28.3.1. GameLogic Module

`SCA_PythonController` **`getCurrentController`**`( );`

Returns the Controller object that carries the script.

`void` **`addActiveActuator`**`(` *`actuator , bool active`* `);`

This method makes the Actuator "actuator" active ("active=TRUE") or inactive ("active=FALSE").

`float` **`getRandomFloat`**`( );`

This function returns a random float in the range of 0.0...1.0. The seed is taken from the system time, so you get a different sequence of random numbers at every game start.

`setGravity``(` *`[gravityX,gravityY,gravityZ]`* `);`

Sets the world gravity.

28.3.2. Rasterizer Module

`int` **`getWindowWidth`**`( );`

This function returns the width of the Blender window the game is running in.

`int` **`getWindowHeight`**`( );`

This function returns the height of the Blender window the game is running in.

`void` **`makeScreenshot`**`(` *`char* filename`* `);`

This function writes a screenshot of the game as a TGA file to disk.

`enableVisibility``(` *`bool usevisibility`* `);`

This sets all objects to invisible when "usevisibility" is TRUE. The game can then set the visibility back to "on" for the necessary objects only.

`showMouse``(` *`bool show`* `);`

Shows or hides the mouse cursor while the game engine runs, depending on the show parameter. The default behaviour is to hide the mouse, but moving over the window border will reveal it again, so set the mouse cursor visibility explicitly with this function.

`setBackgroundColor``(` *`list [float R,float G,float B]`* `);`

Sets the Backgroundcolor. Same as the horizon color in the WorldButtons.

`setMistColor``(` *`list [float R,float G,float B]`* `);`

Sets the mist (fog) color. In the game engine you can set the mist color independantly from the backgroundcolor. To have a mist effect, activate "Mist" in the WorldButtons.

`setMistStart``(` *`float start`* `);`

Sets the distance where the Mist starts to have effect. See also the WorldButtons.

`setMistEnd``(` *`float end`* `);`

Sets the distance from MistStart (0% Mist) to 100% mist. See also the WorldButtons.

28.3.3. GameKeys Module

This is a module that simply defines all keyboard keynames (AKEY = 65 etc).

"AKEY", ..., "ZKEY", "ZERO_KEY" , ..., "NINEKEY", "CAPSLOCKKEY", "LEFTCTRLKEY", "LEFTALTKEY", "RIGHTALTKEY", "RIGHTCTRLKEY",

"RIGHTSHIFTKEY", "LEFTSHIFTKEY", "ESCKEY", "TABKEY", "RETKEY", "SPACEKEY", "LINEFEEDKEY", "BACKSPACEKEY", "DELKEY", "SEMICOLONKEY", "PERIODKEY", "COMMAKEY", "QUOTEKEY", "ACCENTGRAVEKEY", "MINUSKEY", "VIRGULEKEY", "SLASHKEY", "BACKSLASHKEY", "EQUALKEY", "LEFTBRACKETKEY", "RIGHTBRACKETKEY", "LEFTARROWKEY", "DOWNARROWKEY", "RIGHTARROWKEY", "UPARROWKEY", "PAD0", ..., "PAD9", "PADPERIOD", "PADVIRGULEKEY", "PADASTERKEY", "PADMINUS", "PADENTER", "PADPLUSKEY", "F1KEY", ..., "F12KEY", "PAUSEKEY", "INSERTKEY", "HOMEKEY", "PAGEUPKEY", "PAGEDOWNKEY", and "ENDKEY".

28.4. Standard methods for LogicBricks

All LogicBricks inherit the following methods:

```
gameObject*getOwner( );
```

This returns the owner of the GameObject the LogicBrick is assigned to.

28.4.1. Standard methods for Sensors

All sensors inherit the following methods:

```
int isPositive( );
```

True if the sensor fires a positive pulse. Very usefull for example to differentiate the press and release state from a KeyboardSensor.

```
bool getUsePosPulseMode( );
```

Returns TRUE if positive pulse mode is active, FALSE if positive pulse mode is not active.

```
setUsePosPulseMode( bool flag );
```

Set "flag" to TRUE to switch on positive pulse mode, FALSE to switch off positive pulse mode.

```
int getPosFrequency( );
```

Returns the frequency of the updates in positive pulse mode.

```
setPosFrequency( int freq );
```

Sets the frequency of the updates in positive pulse mode. If the frequency is negative, it is set to 0.

```
bool getUseNegPulseMode( );
```

Returns TRUE if negative pulse mode is active, FALSE if negative pulse mode is not active.

```
setUseNegPulseMode( bool flag );
```

Set "flag" to TRUE to switch on negative pulse mode, FALSE to switch off negative pulse mode.

```
int getNegFrequency( );
```

Returns the frequency of the updates in negative pulse mode.

```
setNegFrequency( int freq );
```

Sets the frequency of the updates in negative pulse mode. If the frequency is negative, it is set to 0.

```
bool getInvert( );
```

Returns whether or not pulses from this sensor are inverted.

```
setInvert( bool flag );
```

Set "flag" to TRUE to invert the responses of this sensor, set to FALSE to keep the normal response.

28.4.2. Standard methods for Controllers

Controllers have the following methods:

```
Actuator* getActuator( char* name , );
```

Returns the actuator with "name".

```
list getActuators( );
```

Returns a python list of all connected Actuators.

```
Sensor* getSensor( char* name , );
```

Returns the Sensor with "name".

```
list getSensors( );
```

Returns a python list of all connected Sensors.

28.4.3. Standard methods for GameObjects

The GameObjects you got with getOwner() provide the following methods.

applyImpulse(*list [x,y,z]* ,);

Apply impulse to the gameObject (N*s).

disableRigidBody( );

Disables the rigid body dynamics for the gameObject.

enableRigidBody(,);

Enables the rigid body dynamics for the gameObject.

setVisible(*int visible*);

Sets the GameObject to visible (int visible=1) or invisible (int visible=0), this state is true until the next frame-draw. Use "enableVisibility(bool usevisibility);" from the Rasterizer Module to make all objects invisible.

setPosition(*[x,y,z]*);

Sets the position of the gameObject according to the list of the X, Y and Z coordinate.

pylist [x,y,z] **getPosition**( );

Gets the position of the gameObject as list of the X, Y and Z coordinate.

pylist [x,y,z] **getLinearVelocity**( );

Returns a list with the X, Y and Z component of the linear velocity. The speed is in Blender units per second.

pylist [x,y,z] **getVelocity**( );

Returns a list with the X, Y and Z component of the velocity. The speed is in Blenderunits/second.

float mass **getMass**( );

Returns the mass of the GameObject.

pylist [x,y,z] **getReactionForce**( );

Returns a Python list of three elements.

suspendDynamics( );

Suspends the dynamic calculation in the game engine.

restoreDynamics( );

Suspends the dynamic calculation in the game engine.

Chapter 29. Appendix

29.1. Blender Installation

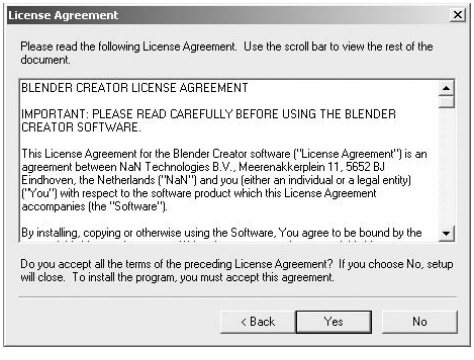
We want to ensure that the installation process is as easy as it can be. Usually the process consists of three easy steps:

1. Get Blender from CD or by downloading it
2. Uncompress the archive or use the installer
3. Start Blender

The Blender Windows version will work on 32 bit versions of Windows (Windows 9x, Windows ME, Windows NT and Windows 2000). Get the installer archive from our website, or locate it on the CD.

Double click on the installer icon. The installer will load and presents you with a splash screen and some important information about Blender. Read this information and click “Next” to proceed to the next screen.

Please read the license agreement carefully and agree by clicking on “Yes”. The next screen displays some general information on Blender. Press “Next” to skip it.



In the “Choose Setup Folder” screen, enter a valid path where you want to install Blender. Optionally you can browse to a directory using the “browse”-button next to the path. The path’s default is `C:\Program Files\Blender`.

After pressing “Next” in the “Choose Setup Folder” screen, Blender is installed on your hard disk.

The installer offers you the option to start Blender after the installation by activating the checkbox “Start Blender”. To start Blender later, you can use the automatically created shortcut on your desktop, or use the entry in the start-menu.

29.2. Graphics card compatibility by Daniel Dunbar

Blender requires a 3-D accelerated graphics card that supports OpenGL. We strongly recommend making sure you are using the latest version of the drivers for your graphics card before attempting to run Blender. See the Upgrading section below if you are unsure how to upgrade your graphics drivers.

Additionally here are some tips to try if you are having trouble running Blender, or if Blender is running with very low performance.

- Most consumer graphics cards are optimized for 16-bit color mode (High Color). Try changing the color mode you are using in the Display Properties .
- Some cards many not be able to accelerate 3-D at higher resolutions, try lowering your display resolution in the Display Properties (Figure 29-1).
- Some cards may also have problems accelerating 3-D for multiple programs at a time - make sure Blender is the only 3-D application running.
- If Blender runs but displays incorrectly, try lowering the hardware acceleration level in the Performance tab of the Advanced Display Properties (Figure 29-2).

29.2.1. Upgrading your graphics drivers

Graphics cards are generally marketed and sold by a different company than the one that makes the actual chipset that handles the graphics functionality. For example, a Diamond Viper V550 actually uses an NVidia TNT2 chipset, and a Hercules Prophet 4000XT uses a PowerVR Kyro chipset.

Often both the card manufacturer and the chipset maker will offer drivers for your card, however, we recommend always using the drivers from the chipset maker, these are often released more frequently and of a higher quality.

Table 29-1. Card manufacturers

Company	Commonly Used Chipsets
3Dfx, http://www.3dfx.com	3Dfx
AOpen, http://www.aopen.com	NVidia, SiS
ASUS, http://www.asus.com	NVidia
ATI, http://www.ati.com	ATI
Creative, http://www.creative.com	NVidia
Diamond Multimedia, http://www.diamondmm.com	NVidia, S3
ELSA, http://www.elsa.com	NVidia
Gainward, http://www.gainward.com	NVidia, S3
Gigabyte, http://www.gigabyte.com	NVidia
Hercules, http://www.hercules.com	NVidia, PowerVR
Leadtek, http://www.leadtek.com	3DLabs, NVidia
Matrox, http://www.matrox.com	Matrox
Videologic, http://www.videologic.com	PowerVR, S3

If you are not sure which chipset is in your graphics card consult the section on determining your graphics chipset Section 29.2.2.

Once you know which chipset your graphics card uses, find the chipset maker in the table below, and follow the link to that company's driver page. From there you should be able to find the drivers for your particular chipset, as well as further instructions about how to install the driver.

29.2.2. Determining your graphics chipset

The easiest way of finding out what graphics chipset is used by your card, is to consult the documentation (or the box) that came with your graphics card, often the chipset is listed somewhere (for example on the side of the box, or in the specifications page of the manual, or even in the title, i.e. a „Leadtek WinFast GeForce 256").

If you are unable to find out which chipset your card uses from the documentation, follow the steps below.

If you don't know what graphics card you have, go to the "Display Properties" dialog (Figure 29-1), select the "Settings" tab, and look for the "Display" field, where

you should see the names of your monitor and graphics card. Often the graphics card will also display its name/model and a small logo when you turn on the computer.

Once you know which graphics card you have, the next step is to determine which chipset is used by the card. One way of finding this out is to look up the manufacturer in the card manufacturers table and follow the link to the manufacturers website, once you are there, find the product page for your card model; the chipset that the card is based on should be listed somewhere on this page.

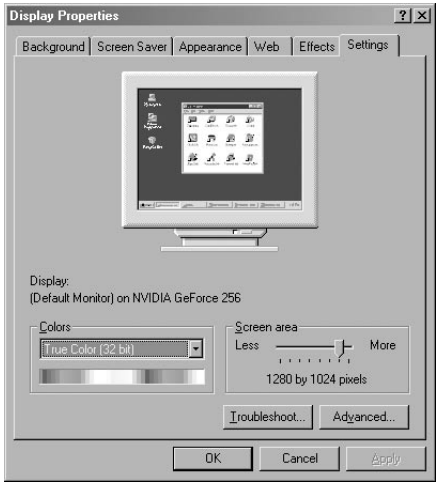
Table 29-2. Chipset manufacturers

Company	Chipsets	Driver Page
3Dfx	Banshee Voodoo	http://www.3dfx.com/downloads.htm
3DLabs	Permedia	http://www.3dlabs.com/support/drivers/index.htm
ATI	Rage Radeon	http://support.ati.com/products/pc/index.html
Intel	i740 i810 i815	http://developer.intel.com/design/software/drivers/platform/
Matrox	G200 G400 G450	http://www.matrox.com/mga/support/drivers/home.cfm
NVidia	Vanta Riva 128 Riva TNT/Geforce	http://www.nvidia.com/view.asp?PAGE=drivers
PowerVR	KYRO KYRO II	http://www.powervr.com/Downloads.asp
Rendition	Verite	http://www.micron.com/content.jsp?path=Products/ITG
S3 Graphics	Savage	http://www.s3graphics.com/DRVVIEW.HTM
SiS	300 305 315 6326	http://www.sis.com/support/driver/index.htm
Trident Microsystems	Blade CyberBlade	http://www.tridentmicro.com/videocomm/download/download.htm

Now that you know which chipset your card uses, you can continue with the instructions in the upgrading section (Section 29.2.1).

29.2.3. Display dialogs in Windows concerning the graphics card

Figure 29-1. Display Properties



The display properties dialog has many useful settings for changing the functioning of your graphics card. To open the display properties dialog, go to “Start Menu -> Settings -> Control Panel” and select the display icon, or right-click on your desktop and select “Properties”.

Figure 29-2. Advanced Display Properties

The advanced display properties dialog has settings for controlling the function of your graphics driver, and often has additional settings for tweaking the 3-D acceleration. To open the advanced display properties dialog, open the Display Properties as described above, then open the Settings tab, and click on the Advanced button in the lower right corner.

29.2.4. Graphics Compatibility Test Results

In the table good (or bad) *performance* refers to the speed of general 3-D drawing and is an indication of how well a game will perform. Good (or bad) *interactivity* refers to how fast the interface responds on the graphics card, and is an indication of how well the graphics card works for creating and editing 3-D scenes in Blender.

All tests are carried out with the latest drivers we could find. If the results on your system do not match ours, make sure you are using the latest drivers, as described in the Upgrading section (Section 29.2.1).

Table 29-3. Tested Chipsets

Chipset Manufacturer	Chipset Model	Windows 98	Windows 2000
3Dfx	Banshee	Works (very poor performance)	-untested-
	Voodoo 3000	Good performance, poor interactivity	Good performance, poor interactivity
	Voodoo 5500	Works (good performance)	-untested-
ATI	All-In-Wonder 128	Works (poor performance)	-untested-
	Rage II 3D	Works (poor performance)	-untested-
	Rage Pro 3D	Works (poor performance)	-untested-
	Radeon DDR VIVO	Good performance, good interactivity	Good performance, good interactivity
Matrox	Millennium G200	Ok performance, extremely poor interactivity, some drawing errors	Ok performance, very poor interactivity, some drawing errors
	Millennium G400	Good performance, poor interactivity	Good performance, poor interactivity
	Millennium G450	Good performance, very poor interactivity	Good performance, Very poor interactivity

NVidia	TNT	Good performance, good interactivity	Good performance, good interactivity
	Vanta	Good performance, good interactivity	Good performance, good interactivity
	TNT2	Good performance, good interactivity	Good performance, good interactivity
	GeForce DDR	Works (good performance)	-untested-
	GeForce 2	Works (good performance)	-untested-
PowerVR	Kyro	Good performance, good interactivity, some drawing errors	Good performance, good interactivity, some drawing errors
Rendition	Verite 2200	Works (poor performance), some drawing errors	-untested-
S3	Virge	Ok performance, good interactivity	-untested-
	Trio 64	Works (poor performance)	-untested-
	Savage 4	-untested-	Works (poor performance)
SiS	6326	Works (poor performance)	-untested-

29.3. Where to get the latest version of Blender

Since October 2002 Blender has become 'Free Software', with the sources available as GNU GPL. During the final editing of this book, new websites about Blender were being built. Check for the latest news about development http://www.blender.org or for general news on the product Blender itself: http://www.blender3d.org

29.4. Support and Website Community

Visit http://www.blender3d.org/GameKit/ where you will find updated support files and a discussion forum where you can post questions and meet with other readers of this book.

For more general feedback on Blender, the excellent independent user site http:/www.elysiun.com/ offers a wealth of links to galleries, tutorials and forums on many different topics. Elysiun is visited by 100s of Blender users each day.

Glossary

A-Z

Active

Blender makes a distinction between *selected* and *active*. Only one Object or item can be *active* at any given time, for example to allow visualization of data in buttons.

See Also: Selected.

Actuator

A LogicBrick that acts like a muscle of a lifeform. It can move the object, or also make a sound. See Section 27.3.

See Also: LogicBrick, Sensor, Controller.

Alpha

The alpha value in an image denotes opacity, used for blending and antialiasing.

Anti-aliasing

An algorithm designed to reduce the stair-stepping artifacts that result from drawing graphic primitives on a raster grid.

Back-buffer

Blender uses two buffers in which it draws the interface. This double-buffering system allows one buffer to be displayed, while drawing occurs on the back-buffer. For some applications in Blender the back-buffer is used to store color-coded selection information.

Bevel

Beveling removes sharp edges from an extruded object by adding additional material around the surrounding faces. Bevels are particularly useful for flying logos, and animation in general, since they reflect additional light from the corners of an object as well as from the front and sides.

Bounding box

A six-sided box drawn on the screen that represents the maximum extent of an object.

Channel

Some DataBlocks can be linked to a series of other DataBlocks. For example, a Material has eight *channels* to link Textures to. Each IpoBlock has a fixed number of available *channels*. These have a name (LocX, SizeZ, enz.) which indicates how they can be applied. When you add an IpoCurve to a channel, animation starts up immediately.

Child
Objects can be linked to each other in hierarchical groups. The Parent Object in such groups passes its transformations through to the Child Objects.

Clipping
The removal, before drawing occurs, of vertices and faces which are outside the field of view.

Controller
A LogicBrick that acts like the brain of a lifeform. It makes decisions to activate muscles (Actuators), either using simple logic or complex Python scripts. See Section 27.2.

See Also: LogicBrick, Sensor, Python, Actuator.

DataBlock (or "block")
The general name for an element in Blender's Object Oriented System.

Doppler effect
The Doppler effect is the change in pitch that occurs when a sound has a velocity relative to the listener. When a sound moves towards the listener the pitch will rise. when going away from the listener the pitch will drop. A well known example is the sound of an ambulance passing by.

Double-buffer
Blender uses two buffers (images) to draw the interface in. The content of one buffer is displayed, while drawing occurs on the other buffer. When drawing is complete, the buffers are switched.

EditMode
Mode to select and transform vertices of an object. This way you change the shape of the object itself. Hotkey: **TAB**.

See Also: Vertex (pl. vertices).

Extend select
Adds new selected items to the current selection (**SHIFT-RMB**)

Extrusion
The creation of a three-dimensional object by pushing out a two-dimensional outline and giving it height, like a cookie-cutter. It is often used to create 3-D text.

Face
The triangle and square polygons that form the basis for Meshes or for rendering.

FaceSelectMode
Mode to select faces on an object. Most important for texturing objects. Hotkey: **FKEY**

Flag
A programming term for a variable that indicates a certain status.

Flat shading
A fast rendering algorithm that simply gives each facet of an object a single color. It yields a solid representation of objects without taking a long time to render. Pressing **ZKEY** switches to flat shading in Blender.

Fps
Frames per second. All animations, video, and movies are played at a certain rate. Above ca. 15fps the human eye cannot see the single frames and is tricked into seeing a fluid motion. In games this is used as an indicator of how fast a game runs.

Frame
A single picture taken from an animation or video.

Gouraud shading
A rendering algorithm that provides more detail. It averages color information from adjacent faces to create colors. It is more realistic than flat shading, but less realistic than Phong shading or ray-tracing. The hotkey in Blender is **CTRL-Z**.

Graphical User Interface
The whole part of an interactive application which requests input from the user (keyboard, mouse etc.) and displays this information to the user. Blenders GUI is designed for a efficient modeling process in an animation company where time equals money. Blenders whole GUI is done in OpenGL.

See Also: OpenGL.

Hierarchy
Objects can be linked to each other in hierarchical groups. The Parent Object in such groups passes its transformations through to the Child Objects.

Ipo
The main animation curve system. Ipo blocks can be used by Objects for movement, and also by Materials for animated colors.

IpoCurve
The Ipo animation curve.

Item
The general name for a selectable element, e.g. Objects, vertices or curves.

Keyframe
A frame in a sequence that specifies all of the attributes of an object. The object can then be changed in any way and a second keyframe defined. Blender automatically creates a series of transition frames between the two keyframes, a process called „tweening."

Layer
A visibility flag for Objects, Scenes and 3DWindows. This is a very efficient method for testing Object visibility.

Link
The reference from one DataBlock to another. It is a „pointer“ in programming terminology.

Local
Each Object in Blender defines a *local* 3D space, bound by its location, rotation and size. Objects themselves reside in the *global* 3-D space.

A DataBlock is *local*, when it is read from the current Blender file. Non-local blocks (library blocks) are linked parts from other Blender files.

LogicBrick
A graphical representation of a functional unit in Blender's game logic. LogicBricks can be Sensors, Controllers or Actuators.

See Also: Sensor, Controller, Actuator.

Mapping
The relationship between a Material and a Texture is called the ‚mapping'. This relationship is two-sided. First, the information that is passed on to the Texture must be specified. Then the effect of the Texture on the Material is specified.

Mipmap
Process to filter and speed up the display of textures.

ObData block
The first and most important DataBlock linked by an Object. This block defines the Object *type*, e.g. Mesh or Curve or Lamp.

Object
The basic 3-D information block. It contains a position, rotation, size and transformation matrices. It can be linked to other Objects for hierarchies or deformation. Objects can be „empty“ (just an axis) or have a link to ObData, the actual 3-D information: Mesh, Curve, Lattice, Lamp, etc.

OpenGL
OpenGL is a programming interface mainly for 3D applications. It renders 3-D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Blenders whole interface and 3-D output in the real-time and interactive 3-D graphic is done by OpenGL.

Parent
An object that is linked to another object, the parent is linked to a child in a parent-child relationship. A parent object's coordinates become the center of the world for any of its child objects.

Perspective view
In a perspective view, the further an object is from the viewer, the smaller it appears. See orthographic view.

Pivot
A point that normally lies at an object's geometric center. An object's position and rotation are calculated in relation to its pivot-point. However, an object can be moved off its center point, allowing it to rotate around a point that lies outside the object.

Pixel
A single dot of light on the computer screen; the smallest unit of a computer graphic. Short for „picture element."

Plug-In
A piece of (C-)code loadable during runtime. This way it is possible to extend the functionality of Blender without a need for recompiling. The Blender plugin for showing 3D content in other applications is such a piece of code.

Python
The scripting language integrated into Blender. Python (http://www.python.org/) is an interpreted, interactive, object-oriented programming language.

Render
To create a two-dimensional representation of an object based on its shape and surface properties (i.e. a picture for print or to display on the monitor).

Rigid Body
Option for dynamic objects in Blender which causes the game engine to take the shape of the body into account. This can be used to create rolling spheres for example.

Selected
Blender makes a distinction between *selected* and *active* objects. Any number of objects can be *selected* at once. Almost all key commands have an effect on *selected* objects. Selecting is done with the right mouse button.

See Also: Active, Selected, Extend select.

Sensor
A LogicBrick that acts like a sense of a lifeform. It reacts to touch, vision, collision etc. See Section 27.1.

See Also: LogicBrick, Controller, Actuator.

Single User
DataBlocks with only one user.

Smoothing
A rendering procedure that performs vertex-normal interpolation across a face before lighting calculations begin. The individual facets are then no longer visible.

Transform
Change a location, rotation, or size. Usually applied to Objects or vertices.

Transparency
A surface property that determines how much light passes through an object without being altered.

See Also: Alpha.

User
When one DataBlock references another DataBlock, it has a user.

Vertex (pl. vertices)
The general name for a 3-D or 2-D point. Besides an X,Y,Z coordinate, a vertex can have color, a normal vector and a selection flag. Also used as controlling points or handles on curves.

Vertex array
A special and fast way to display 3-D on the screen using the hardware graphic acceleration. However, some OpenGL drivers or hardware doesn't support this, so it can be switched off in the InfoWindow.

Wireframe
A representation of a three-dimensional object that only shows the lines of its contours, hence the name „wireframe."

X, Y, Z axes
The three axes of the world's three-dimensional coordinate system. In the FrontView, the X axis is an imaginary horizontal line running from left to right; the Z axis is a vertical line; and Y axis is a line that comes out of the screen toward you. In general, any movement parallel to one of these axes is said to be movement along that axis.

X, Y, and Z coordinates
The X coordinate of an object is measured by drawing a line that is perpendicular to the X axis, through its centerpoint. The distance from where that line intersects the X axis to the zero point of the X axis is the object's X coordinate. The Y and Z coordinates are measured in a similar manner.

Z-buffer
For a Z-buffer image, each pixel is associated with a Z-value, derived from the distance in ‚eye space' from the Camera. Before each pixel of a polygon is drawn, the existing Z-buffer value is compared to the Z-value of the polygon at that point. It is a common and fast visible-surface algorithm.

Index