blender™

Reference to Blender's

real-time 3D engine

NOT A NUMBER 3D ENABLING TECHNOLOGY

Table of Contents

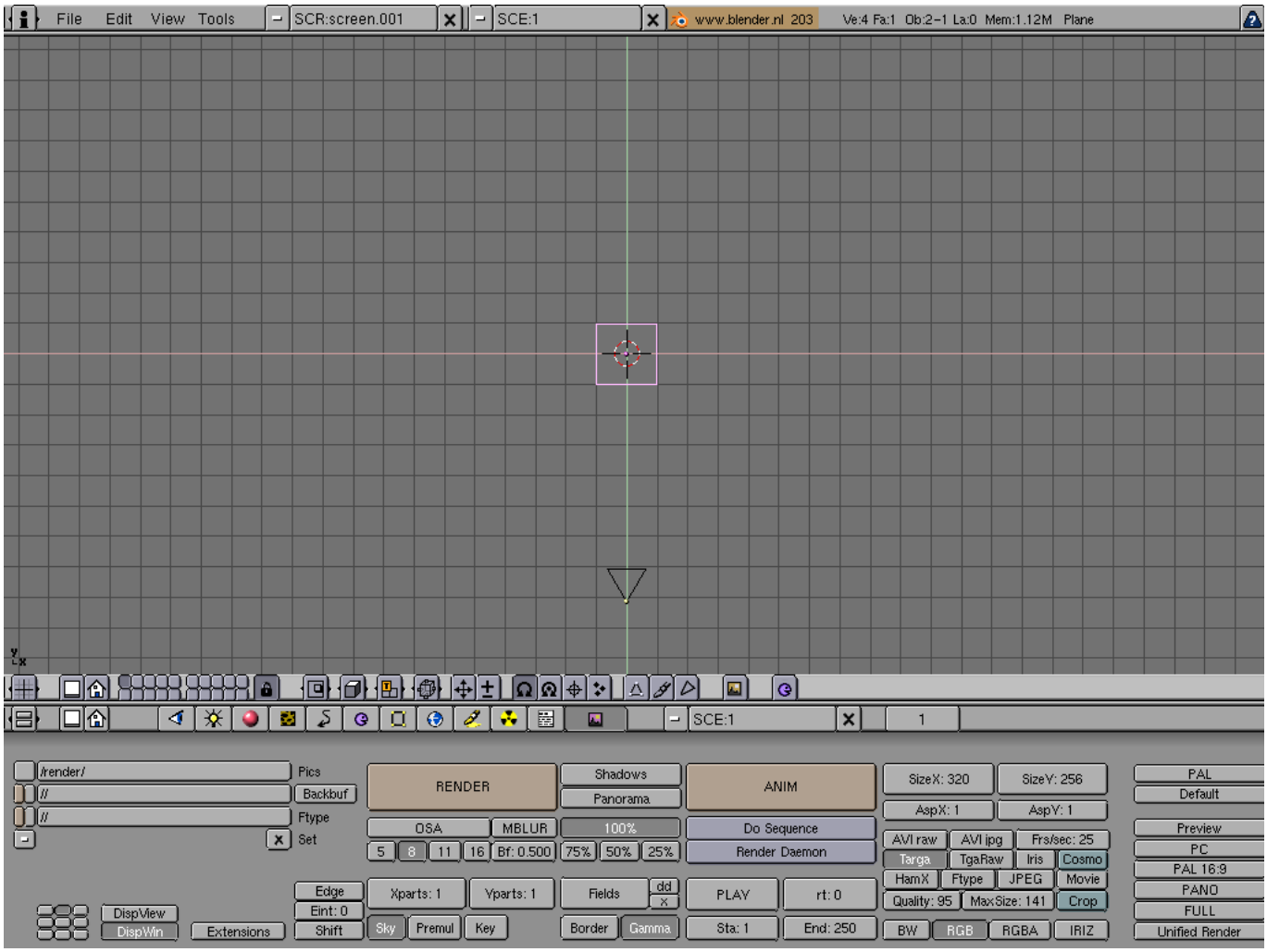
List of Tables

List of Figures

Chapter 1. Blender Basics

For beginners the Blender user interface can be a little confusing as it is different to other 3D 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 1-1. The first start



The installation of Blender is simple, just unpack it and put it in a directory of your chosing or use the installer to let it do the whole installation. Installation is described in detail in Section 1.1.

After starting Blender you get a screen as shown in figure 1.1. 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.

1.1. Blender Installation

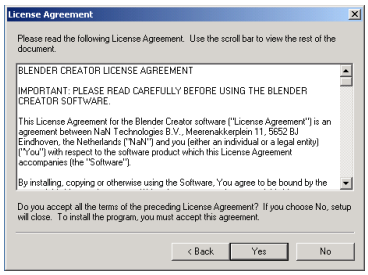
We want to ensure that the installation process is as easy as it can be. Usually the process consists of three easy steps:

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.

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 harddisk.
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.

1.2. Keys and Interface conventions

During its development, which always 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 Blenders 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 3D objects is called 3DWindow, other examples are ButtonsWindow, PaintFaceButtons or EditMode.

1.3. 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':

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 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)

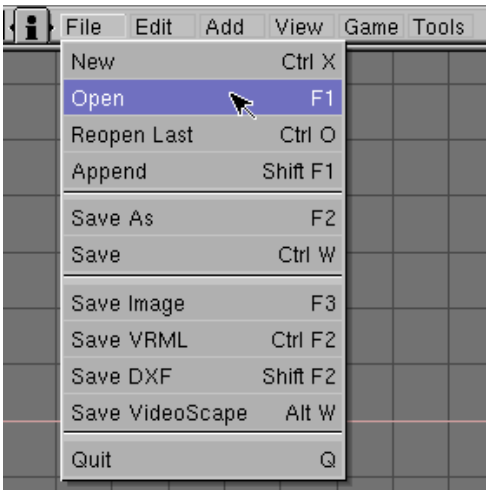
On systems with only two mouse buttons, you can substitute the middle mouse button with the **ALT** *key and the left mouse button.*

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 deselect it.

1.4. Loading and saving

Figure 1-2. 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 1-3. Blender's main menu, the Toolbox

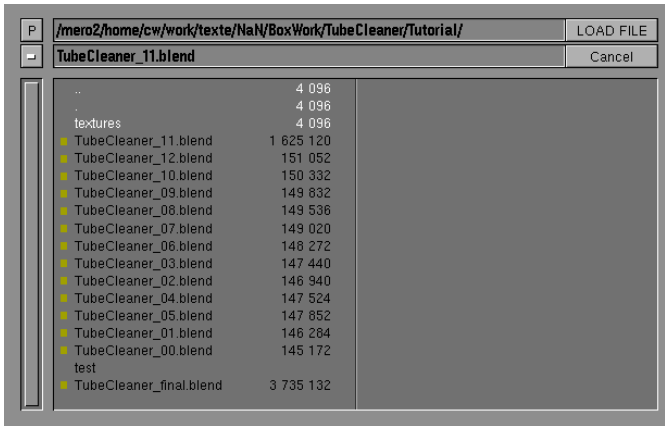
The **SPACE** key brings up the Toolbox, a large pop-up menu that offers you the most commonly used operations in Blender. The 'FILE' entry allows you also to action file operations. Behind every command you will find the associated hotkey.

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

Figure 1-4. Blender FileWindow



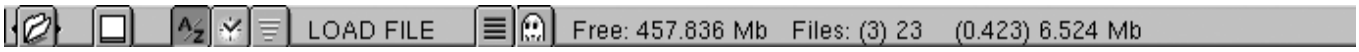
The main part of a FileWindow is the listing of directories and files. File types known by Blender are allocated a yellow square. A click with the **LMB** selects a file and puts the name into the filename-input. A **ENTER** or click on the 'LOAD FILE' Buttons will then load the file. Cancel the operation 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.

*The **PAD+** and **PAD-** keys increase and decrease the last number in a filename. 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 1-5. 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 remaining on the disk, and how many megabytes the selected files are.

Version control and backupfiles

Figure 1-6. 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:' controlls how many version files are written. Beside these possibilities for disaster recovery, Blender writes a file `quit.blend` containing your last scene into the temporary directory 'Dir:' when you quit Blender.

1.5. 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 1-7. 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 doesn't have to close more than one Window for joining.

1.6. 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:

Button

This button, which is usually displayed in salmon colour, 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', it activates processes.

IconToggle

Button type 'TogBut', it 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.

1.7. 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. 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 document.

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.

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.

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.

ScriptButtons

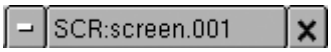
Assigning of Python scripts to world, material, and objects (Blender Creator).

DisplayButtons, F10

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

1.8. Screens

Figure 1-8. 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 3D work, another for working with Ipos and, a complete file manager to arrange your files and textures.

1.9. Scenes

Figure 1-9. Scene browse

Scenes are a way to organize your work and to render more than one scene in the Blender real-time 3D 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.

1.10. 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 you reset Blender with **CTRL-X**, use **CTRL-U** to save it to your home directory.

1.11. Navigating in 3D

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

1.11.1. 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.

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** 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 hight 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.

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

You should now try experiementing a little with these keys to get a feel for their operation and function. If you get lost, use **CTRL-X** followed by **RETURN** to get yourself back to the default scene.

1.11.2. Using the mouse to change your view

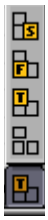
The main button for navigating with the mouse in the 3DWindow is the middle mouse button (**MMB**). Press and hold the **MMB** in a 3DWindow, and then drag the mouse. The view is rotated with the movement of your mouse. Try using a perspective view (**PAD5**) while experimenting it gives a very realistic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

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.

1.12. Selecting of Objects

Selecting an object is achieved by clicking the object using the right mouse button (**RMB**) This operation also deselects all other objects. To extend the selection to more than one object, hold down **SHIFT** while clicking. Selected objects will change 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 deselected 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.

1.13. 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 real-time 3D engine Polygon-Meshes) or share this Mesh with more other objects.

So what is the vantage 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

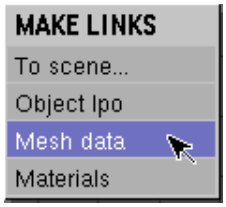
A linked copy is achived by using the ALT-D keycommand. The difference to the copy with SHIFT-D is that the mesh forming the object is not duplicated but linked to the new objects.

User Button

Another common method to create and change links and Blender interface element is the UserButton. This MenuButton allows you 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.

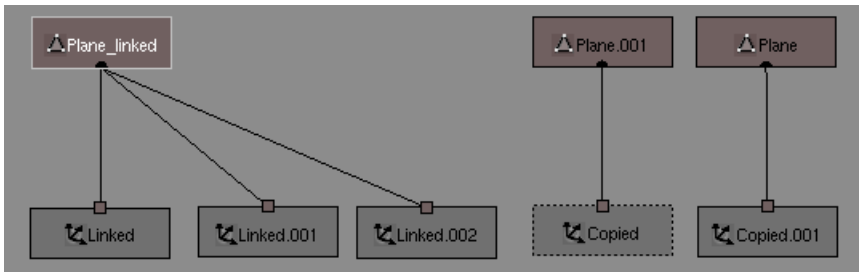
If an Object has more than one user the UserButton will be blue and a number indicates the number of users (in the above image three). Selecting this number will make a copy of the Data and makes the object 'Single User'.

Linking



To link Data from the active to the selected Objects can be done with the keycommand **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 1-10. Object visualization 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** 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.

1.14. 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 toggle 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.

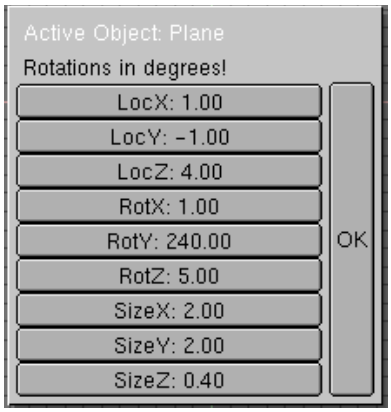
Rotate

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.

Chapter 2. 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 3D 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.

2.1. The 3DWindow

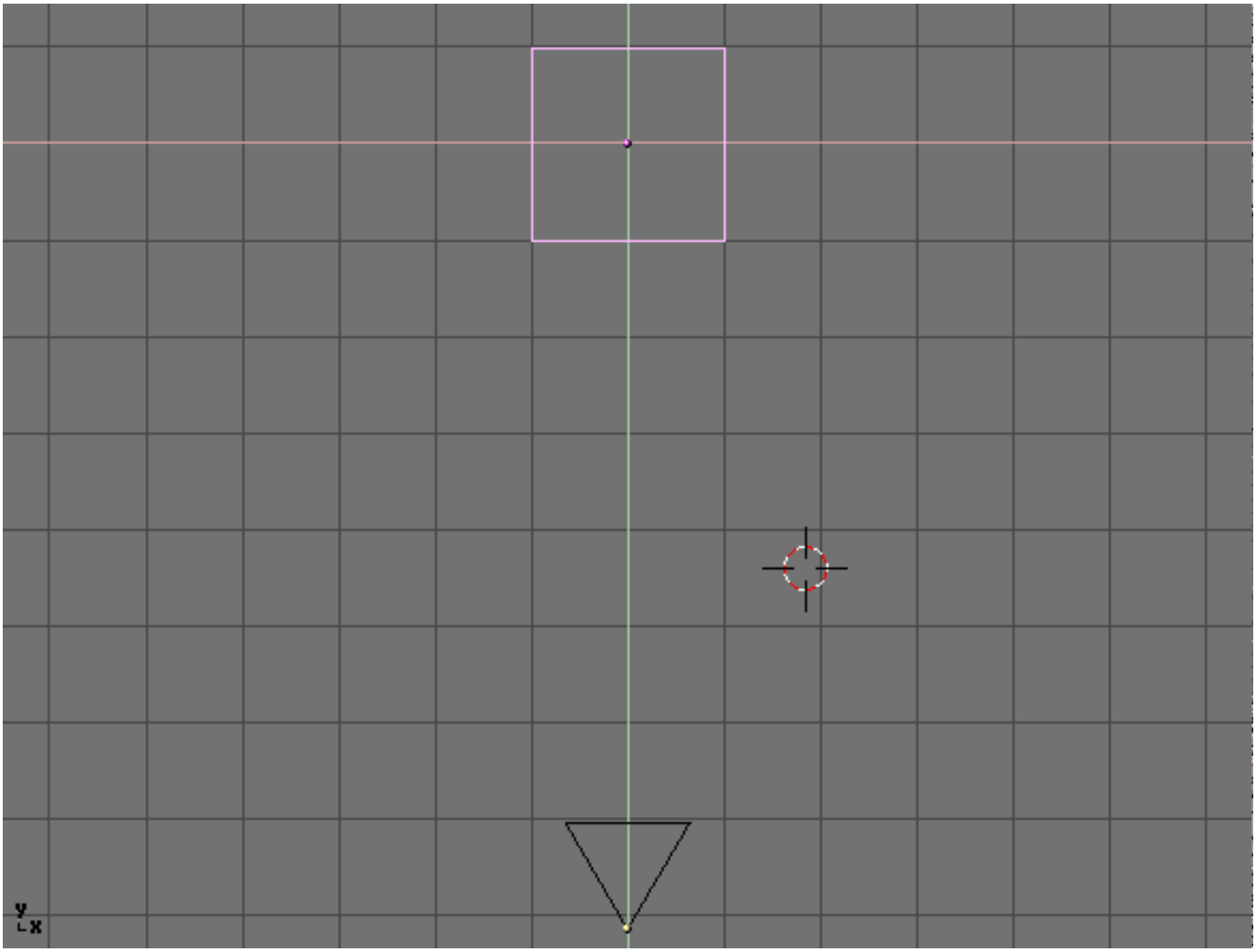
The 3DWindow is the most important window for working and navigating inside 3D scenes. It is also used to play the interactive content. So a good knowledge of the options and capabilities will help you to make your scenes or explore scenes from the CD.

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 colour 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 document.

Figure 2-1. The 3DWindow canvas



2.1.1. 3DHeader

WindowType (IconMenu)
As with every window header, the first button allows you to set the window type.

Full Window (IconTog)
Maximise 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 3D 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.
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 proportinally 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.

2.1.2. 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.

2.1.3. 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.

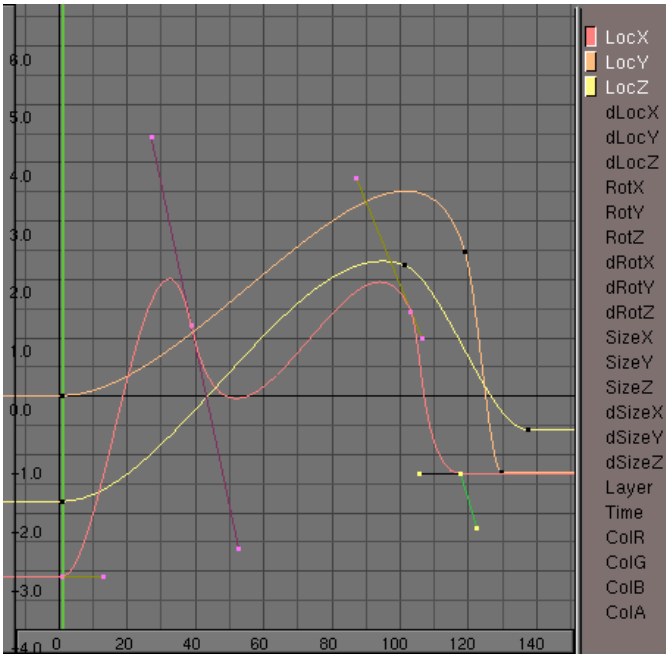
PAD_4 PAD 6
Rotate using the *turntable* method. This is a rotation around the Z axis.

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.

2.2. IpoWindow

Figure 2-2. The IpoWindow



The IpoWindow allows you to visualize and maniplulate animation curves which can control nearly every aspect of an animation inside Blender. For the Blender real-time 3D engine the most important aspects are for example the positions or rotations of objects, but also the color of objects.

2.2.1. IpoHeader

WindowType (IconMenu)

As with every window header, the first button enables you to set the window type.

Full Window (IconTog)
Maximise 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 3D 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**.

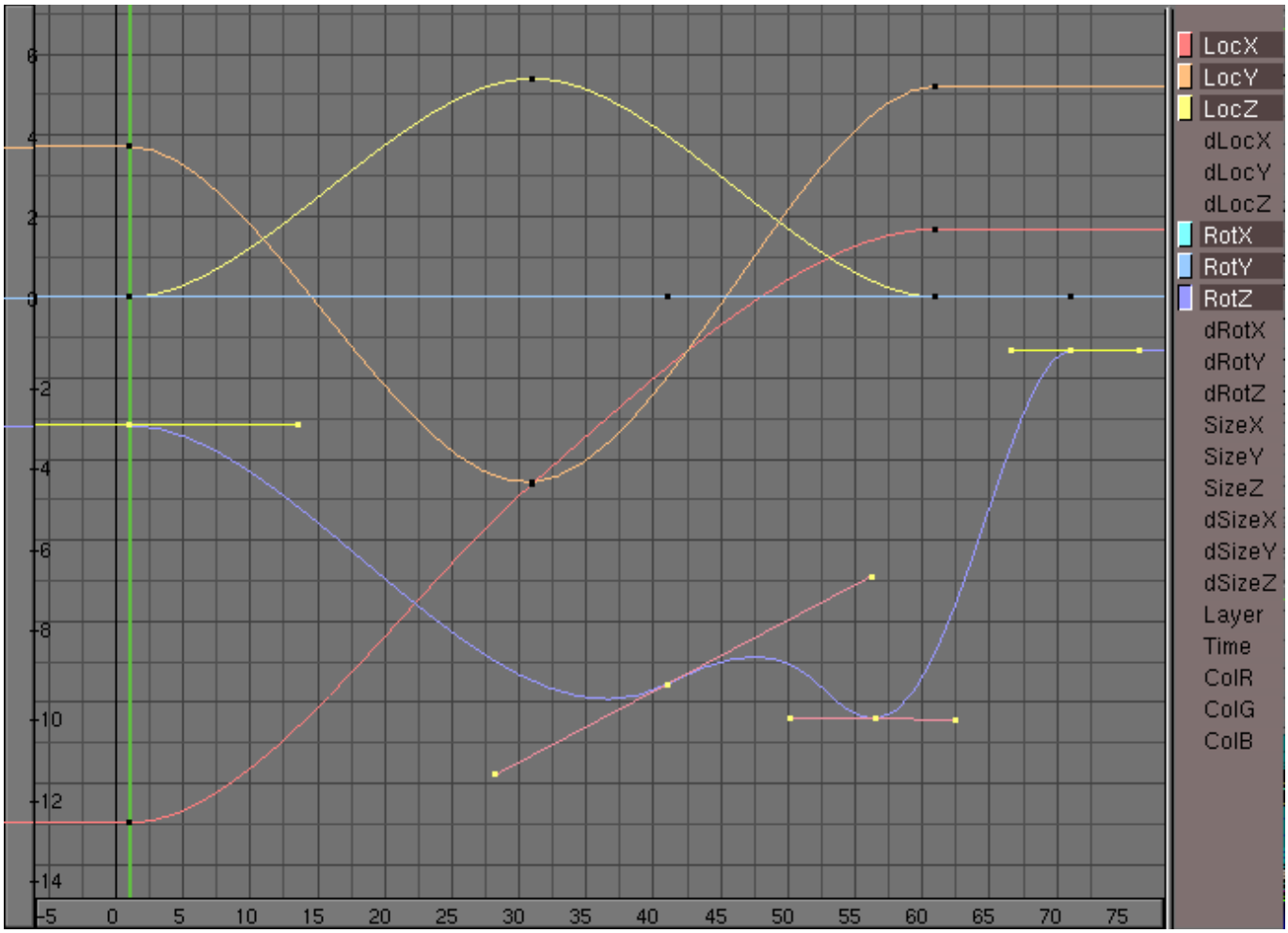
View Border (IconBut)
Draw a rectangle to indicate what part of the IpoWindow should be displayed in the full window.

Lock (TogBut)

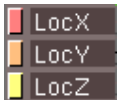
This button locks the update of the 3DWindow while editing in the IpoWindow, so you can see changes made to the Ipo in real-time in the 3DWindow. This option works extremely well with relative vertex keys.

2.2.2. IpoWindow

Figure 2-3. 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 colour 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**.

2.2.3. 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.

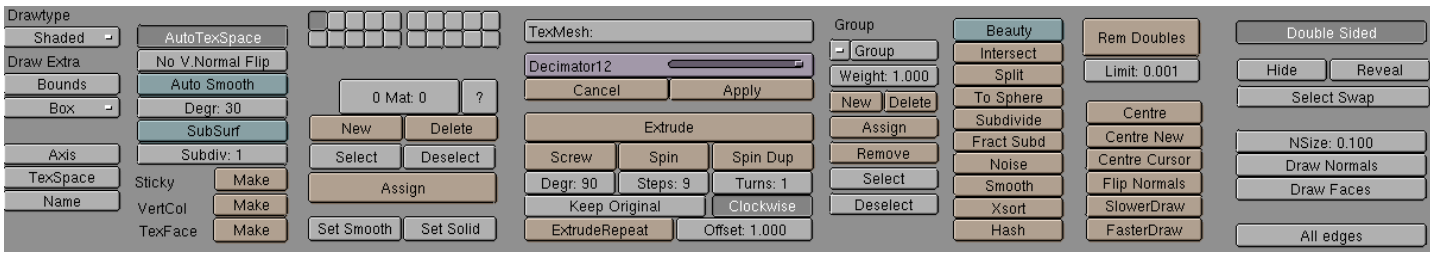
Right mouse
Selection works the same here as in the 3DWindow: normally one item is selected. Use **SHIFT** to expand or reduce the selection (*extend* select).
If the IpoWindow is in IpoKey mode, the IpoKeys can be selected.
If at least 1 of the IpoCurves is in EditMode, only its vertices can be selected.
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.

2.3. EditButtons

Figure 2-4. 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 real-time 3D engine are: Mesh, Empty, Armature, Lamp and Camera. Options and Buttons not appropriate for the Blender real-time 3D engine are not described here.

The DataButtons in the header specify which block is visualized. Mesh is used as an example here, but the usage of the other types of ObData is identical.

Mesh Browse (MenuBut)
Select another Mesh from the list provided.

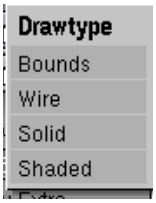
ME: (TextBut)
Give the current block a new and unique name. The new name is inserted in the list, sorted alphabetically.

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)

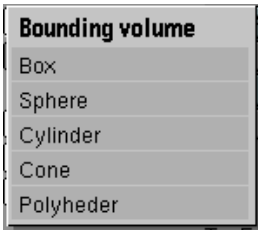
Give the current Object a new and unique name. The new name is inserted in the list, sorted alphabetically. This group of buttons specifies Object characteristics.

DrawType (MenuBut)



Choose a preference for the standard display method 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, in increasing degree 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 approach to the manner in which Blender renders. It depicts the situation of a single frame from the Camera. 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'.

2.3.1. 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 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 with faces that have sharp angles (smaller than 100 degrees). Be sure the face normals are set consistently in the same direction (**CTRL+N** in EditMode). The direction of the normals is especialy important for real-time modells, because the real-time 3D engine renders single sided for speed reasons.

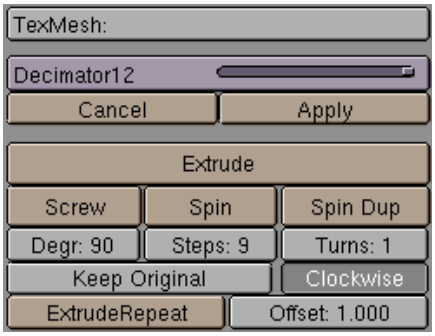
AutoSmooth (TogBut)
Automatic smooth rendering (not faceted) for meshes. Especially interesting for imported Meshes done in other 3D applications. The Button 'Set smooth' also has to be activated to make 'Auto Smooth' work. The smoothing isn't displayed in the 3D Window.

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. Will be automaticly set when you use the UV-Editor to texture a real-time modell. Unchecking this option clears all UV coordinates.

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.

Mesh decimation will remove UV coordinates and vertex colors!

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. 'Extrude' in EditMode 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 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’, 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)
Weight setting for groups and for use in WeightPaint.

New (But)
Create a new vertex group.

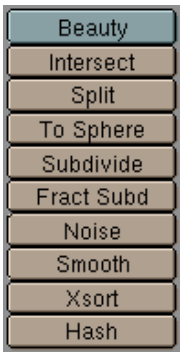
Delete (But)
Delete the actual vertex group.

Assign (But)
Assign the selected vertices to the actual group.

Remove (But)
Remove selected vertices from the actual group.

Select (But)
Select all vertices from the actual group.

Deselect (But)



Deselect 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. Or use Image textures to express them in relief.

Smooth (But)
All edges with both vertices selected are shortened. 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 closer to one another than the 'Limit' are combined and redundant faces are removed.

Centre (But)
Each ObData has its own local 3D 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 recognisability 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; 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)
Toggle 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.

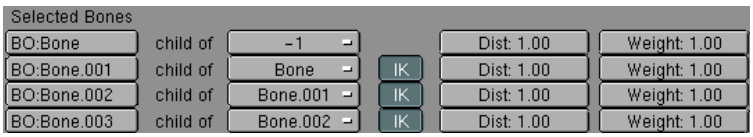
2.3.2. EditButtons, Armatures

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.

2.3.3. 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.

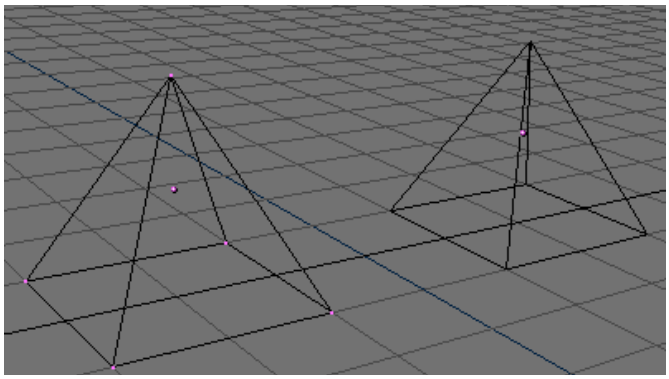
ShowLimits (TogBut)
A line that indicates the values of 'ClipSta' and 'ClipEnd' is drawn in the 3DWindow near the Camera.

ShowMist (TogBut)
A line that indicates the area of the 'mist' (see WorldButtons Section 2.5) is drawn near the Camera in the 3DWindow.

2.4. EditMode

When working in 3D 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.

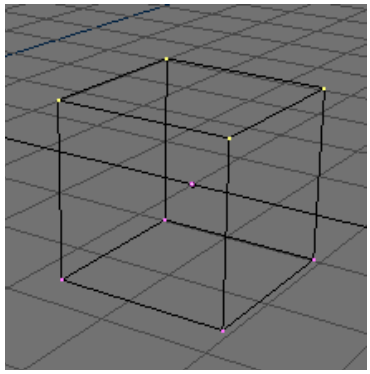
In Blender you switch between these two modes with the **TAB**-key. A selected object outside EditMode is drawn in purple in the 3DWindows (in wireframe mode). To indicate the EditMode the Object's vertices are drawn. Selected vertices are yellow, non-selected are purple.



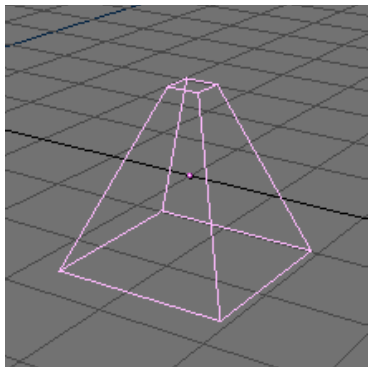
Vertices can be selected like objects with the **RMB**, holding **SHIFT** allows you to select more than one vertex. With some vertices selected you can use **GKEY**, **RKEY** or **SKEY** for manipulating the vertices as you would for whole objects. Add a cube to the default scene. Use the 3DCursor to 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!)

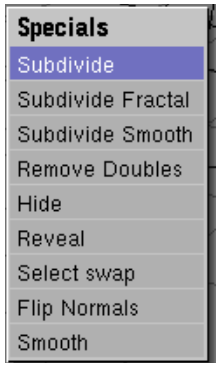
The top vertices change to yellow to indicate that they are selected. You can rotate the view to make sure you really have selected four vertices.



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.



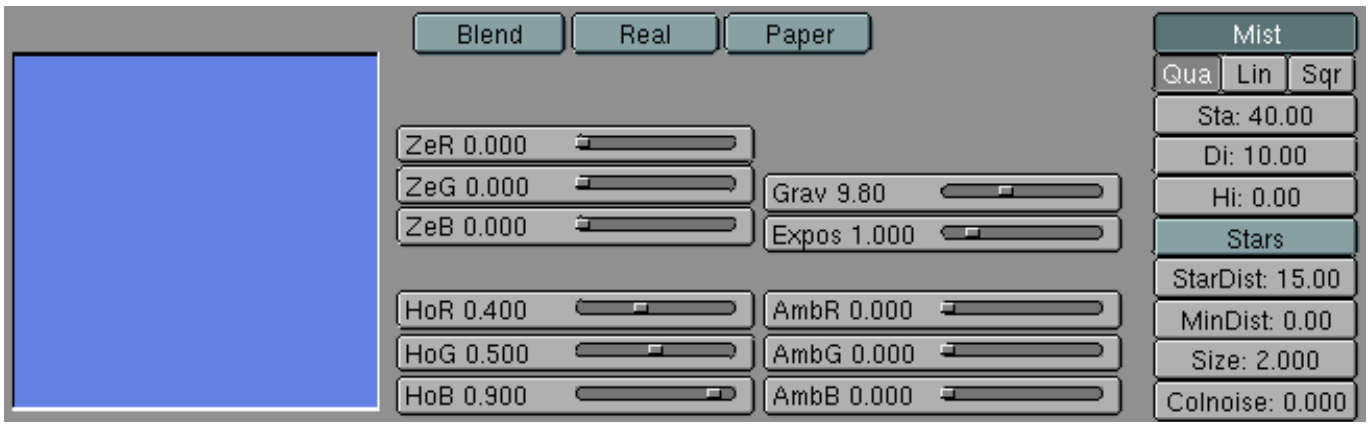
Using **WKEY** you can call up the 'Specials'-menu in EditMode. With that menu you can quickly access the functions often needed for polygon-modeling. You can find the same functionality in the EditButtons **F9**.



2.5. WorldButtons

The WorldButtons define global options for the scene. Only the options appropriate for Blender's real-time 3D engine are explained here.

Figure 2-5. 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'.

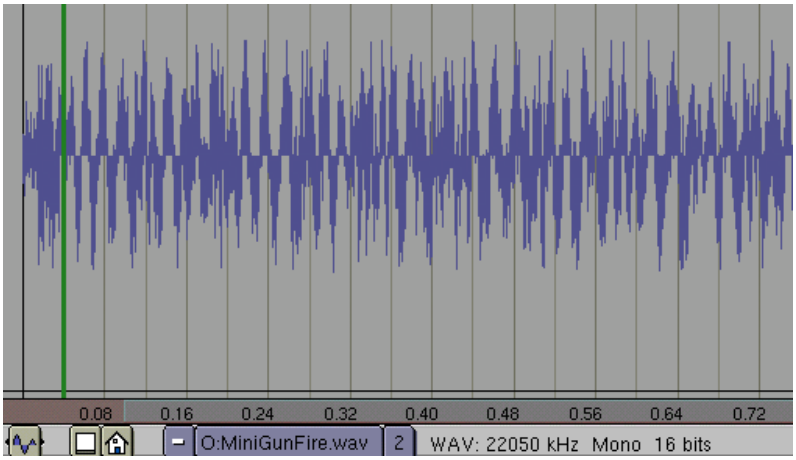
2.6. SoundWindow

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.

Figure 2-6. The SoundWindow



Chapter 3. Real-time Materials

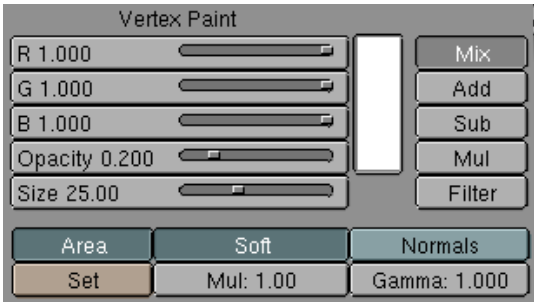
Materials for Blender's real-time 3D engine are applied with vertex-paint or UV-Textures. With VertexPaint you can paint on Meshes, giving them solid colors or patterns of color. VertexPaint is also a very valuable tool to make the suggestion of light on faces and even more important to vary textures. 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.

3.1. 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 3-1. 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.

3.2. TexturePaint

To start TexturePaint select the TexturePaint icon in the 3DWindow Header.

TexturePaint needs a textured object to work. See Section 3.3. You also need to unpack a packed texture first (see Section 3.3.5).

You can now paint on the texture of the object while holding the **LMB**. **RMB** will sample the color located under the mousepointer.

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.

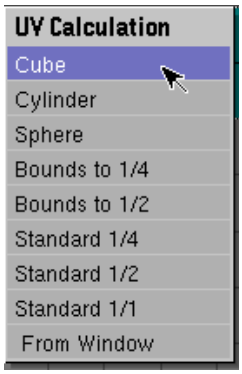
3.3. The UV Editor

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 vertexcolors 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.

3.3.1. 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.

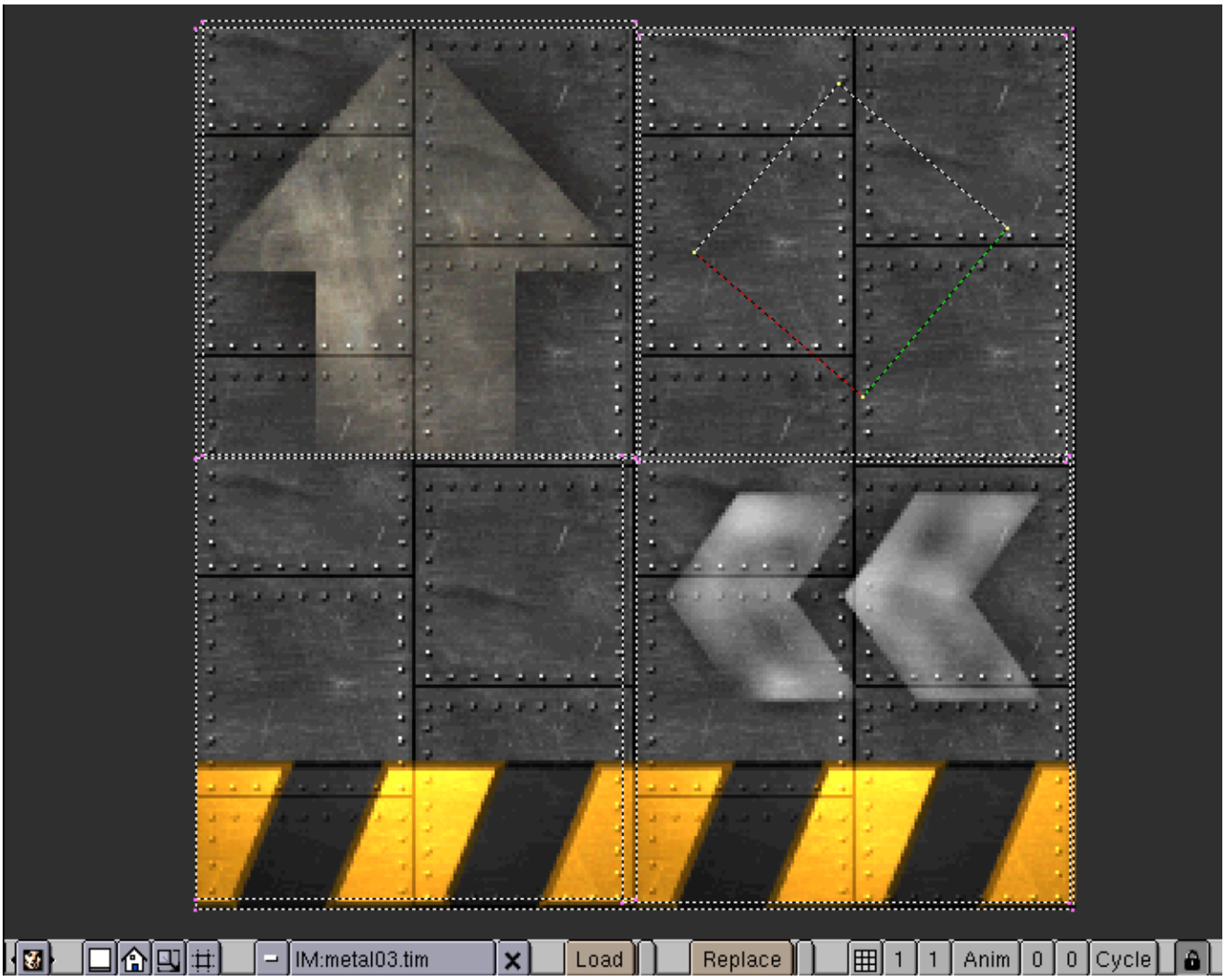
Sphere - Spherical mapping calculated from the center of the selected faces.

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.

Figure 3-2. The Image Window



3.3.2. The ImageWindow

To assign images to faces you need to open an Image Window with **SHIFT-F10**.

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.

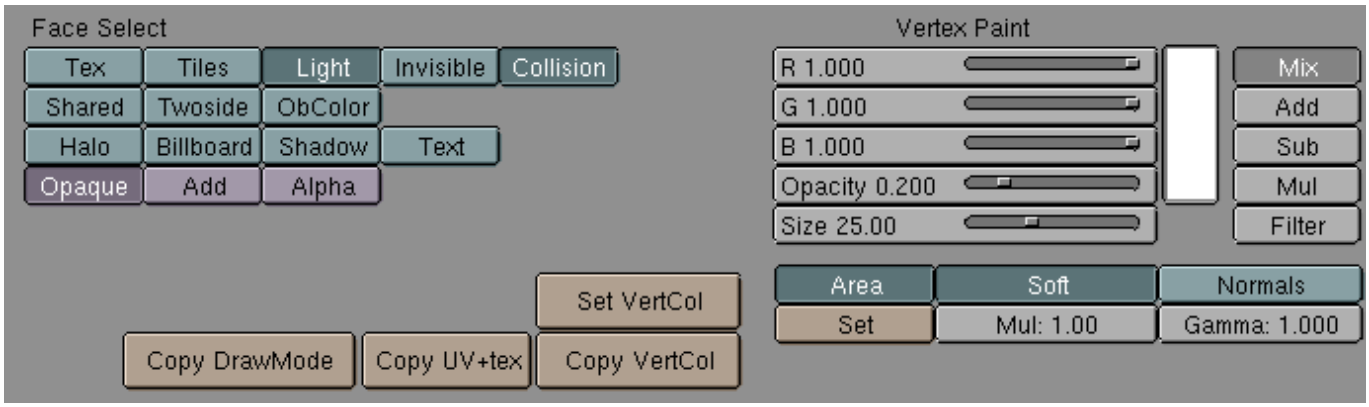
The 'Anim' button enables a simple texture animation. This works in conjunction with the grid settings, in a way that the parts of the texture are displayed in a row in game mode. With the number buttons to the right of the 'Anim' button you define the start and end part to be played. 'Cycle' switches between one-time and cyclic play. 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-**.

3.3.3. The Paint/FaceButtons

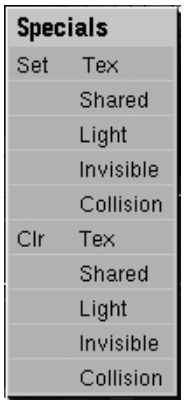
When in FaceSelect mode, you can access the Paint/Face Buttons with the Icon in the ButtonsWindow Header. In the Paint/FaceButtons you'll find all functions to set the attributes for faces and access the VertexPaint options.

Figure 3-3. The Paint/FaceButtons



The following modes always work on faces and display the setting of the active face. Two colored lines in the 3D 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 3-4.

Figure 3-4. 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 3.3.2.

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 4.7.

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 real-time 3D engine.

ObColor
Faces can have color that can be animated by using the ColR, ColG, ColB and ColA Ipos. Choosing this option replaces the vertexcolors.

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 real-time 3D engine, see Section 3.4.

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.

3.3.4. Avaible file formats

Blender uses OpenGL (http://www.opengl.org/) to draw its interface and the the real-time 3D 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.

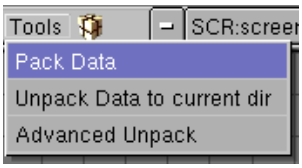
Iris
Iris (*.rgb) is the native IRIX image format. It is a lossless compressed file format, which can include an alpha channel.

Jpeg
A lossy compressed (it uses a compression which leaves out parts of the image which the human eye can hardly see) file format (*.jpg, *.jpeg) 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.

3.3.5. 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 3-5. 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.

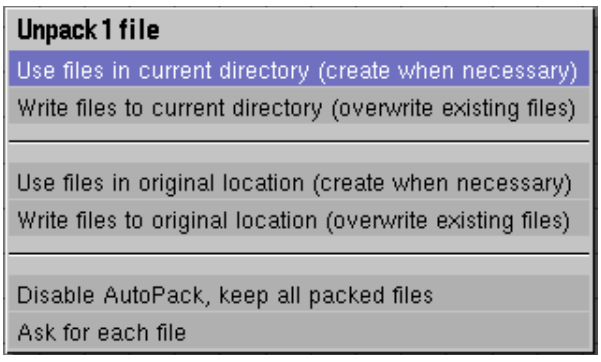
The Tools Menu entries

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 3-6. 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.

3.4. Bitmap text in the real-time 3D 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 will also be rendered as ‘True’ or ‘False’.

Chapter 4. Blender's real-time 3D engine

Technically the Blender real-time 3D engine is a framework with a collection of modules for interactive purposes like physics, graphics, logic, sound and networking. Functionally the real-time 3D 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 comparision 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.

4.1. Options for the real-time 3D engine

Figure 4-1. 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 are printed on screen while the real-time 3D engine is running.

Autostart
Enable Autostart for the scene.

4.2. 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 4-2. InfoWindow options

Blender real-time 3D 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.

4.3. Command line options for the real-time 3D 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 4-3. Blender command line options

```
[cw@work cw]$ blender -h
Blender V 2.21
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>

Real-time 3D 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 (de-fault)

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 real-time 3D engine

-g fixedtime
With this option the real-time 3D 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.

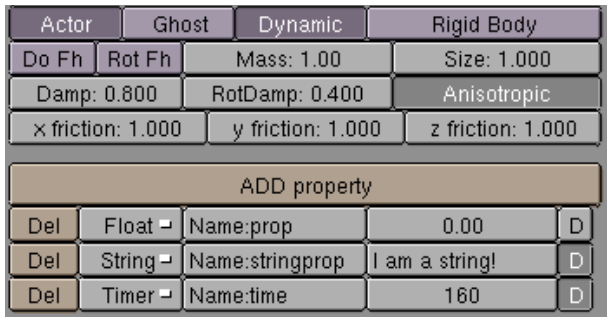
-g linearmipmap
Linear Texture mipmapping instead of nearest (default).

4.4. The RealtimeButtons

The RealtimeButtons are meant for making interactive 3D 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 real-time 3D engine are the RealtimeButtons. Here you define your LogicBricks, which add the behavior to your objects.

Figure 4-4. RealtimeButtons left part




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 real-time 3D engine.

Do Fh
This button activates the Fh mechanism (see Section 4.6). 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.

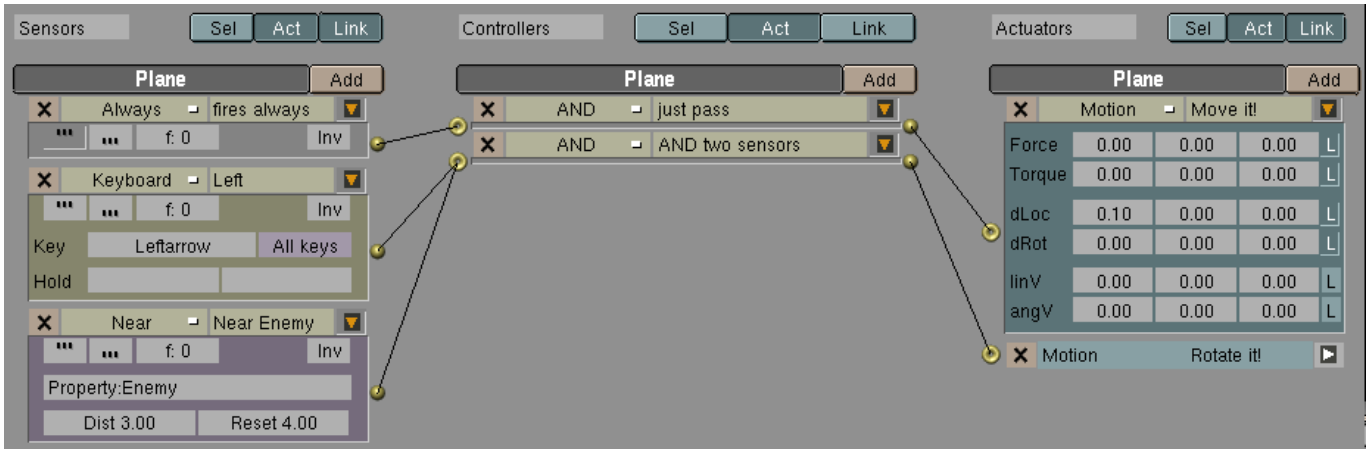
RotDamp
Same as 'Damp' but for rotations of the object.

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 4.6). 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 4.5).

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 4-5. Example of some LogicBricks



Sensors are like the senses of a life form; they react on keypresses, collisions, contact with materials (touch), timer events or values of properties.

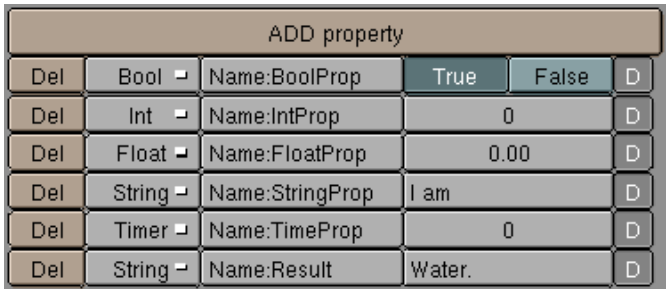
The Controllers are collecting events from the sensors and are able to calculate them to a result. These are similar to the mind or brain of a life form. Simple Controllers just do an AND on the inputs. An example is to test if a key is pressed AND a certain time has passed. There are also OR Controllers 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 real-time 3D engine, edit your 3D world and restart. This way you can drastically cut down your development time!

4.5. 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 5.3.7) or send them to other objects using messages (see Section 5.3.11).

Figure 4-6. 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.

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'.

4.6. 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 4-7).

Figure 4-7. 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 4.4) you make an object float above a surface.'Fh Force' controls the force that keeps the object above the floor.

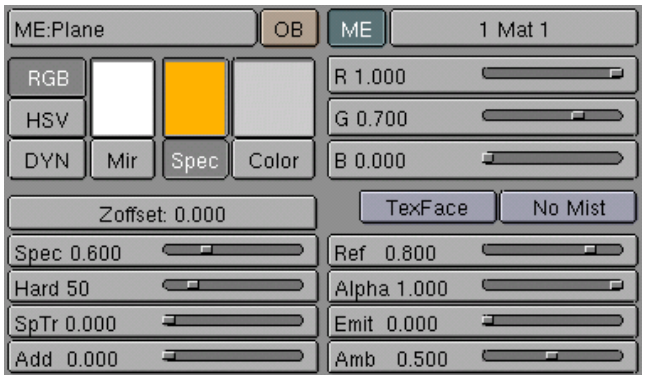
Fh Dist
'Fh Dist' controls the size of the Fh area. When the object enters this area the Fh mechanism starts to work.

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)).

4.6.1. Specularity settings for the real-time 3D engine

Figure 4-8. Specularity settings



Specularity settings in the MaterialButtons
Spec
This slider controls the intensity of the specularity.

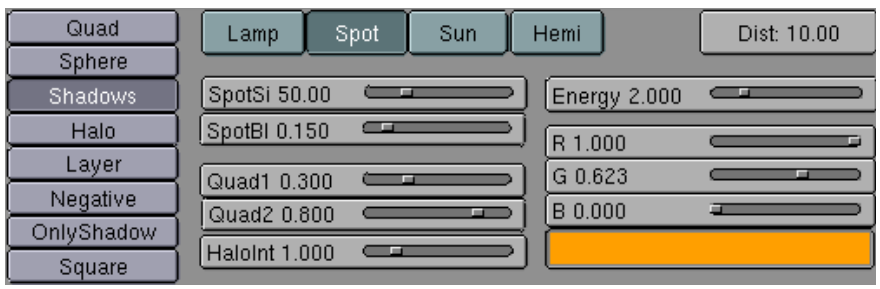
Hard
This slider controls the size of the specularity (hardness).

Spec color
Activating this button, switches the RGB (or HSV) sliders to define the specularity color.

4.7. Lamps in the real-time 3D 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.

Figure 4-9. LampButtons, settings for Blender's real-time 3D engine



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 Blender's real-time 3D engine, the face has to be set to 'Light' in the Paint/FaceButtons  (See Section 3.3). 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 real-time 3D 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'.

4.8. 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 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.

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.

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.

4.9. Expressions

Expressions can be used in the Expression Controller, the Property Sensor and the Property Actuator.

Table 4-1. 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 4-2. 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 4-3. Boolean operations

Operation	Example
NOT EXPR	Not EXPR
EXPR1 OR EXPR2	logical OR
EXPR1 AND EXPR2	logical AND
EXPR1 == EXPR2	EXPR1 equals EXPR2

Contitional statement: IF(Test, ValueTrue, ValueFalse)

Examples:

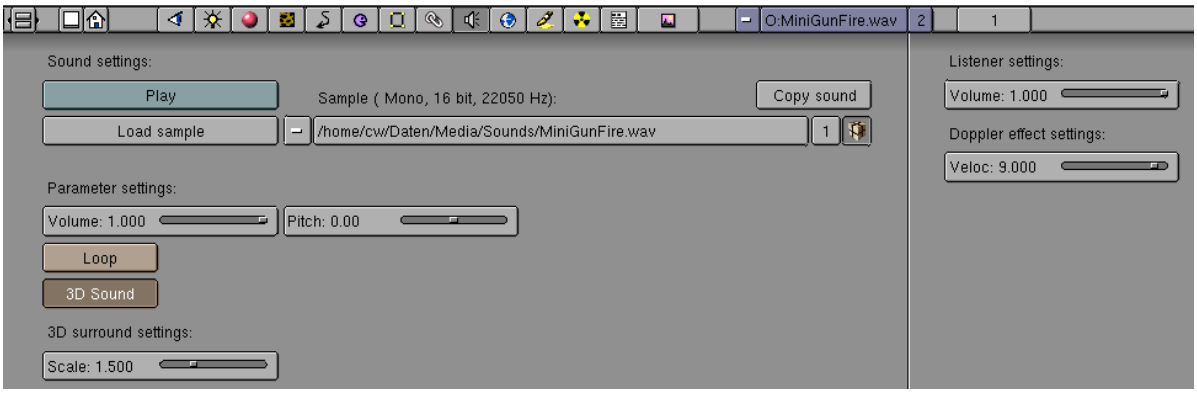
Table 4-4. Expression examples

Expression	Result	Explanation
12+12	24	Addition
property=='Carsten'	TRUE or FALSE	String comparison between a Property and a string
'Erwin'>'Carsten'	TRUE	A string compare is done

4.10. SoundButtons

The SoundButtons are used for loading and managing sounds for the Blender real-time 3D engine. Look at Section 2.6 for a method to visualize the waveform.

Figure 4-10. 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 overidden. The '3D Sound' Button activates the calculation of 3D 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 3D 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 3D Sound), 1.0 (a neutral state, all positions get multiplied by 1) and 5.0 which over accentuates the gain/distance relationship.

4.11. Performance and design style 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 3.3).
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 5. Game LogicBricks

The game logic in Blender's real-time 3D 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.

5.1. Sensors

Sensors act like real senses; they can detect collisions, feel (Touch), smell (Near), view (Ray, Radar).

5.1.1. Always Sensor

The most basic Sensor is the Always Sensor. It is also a good example for the common buttons every sensor has.

Figure 5-1. 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.

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 5-2. Pulse Mode Buttons

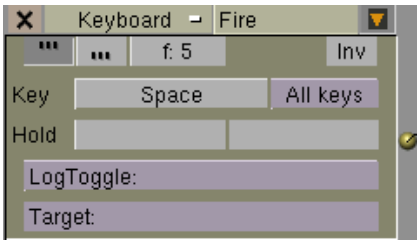
The first button activates the positive pulse mode. Every time the Sensor fires a pulse it is a positive pulse. This can be used, for example to start a movement with an Motion Actuator. The button next to it activates the negative pulse mode, which can be used to stop a movement.

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.

5.1.2. 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 6.3.3) 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.

5.1.3. 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

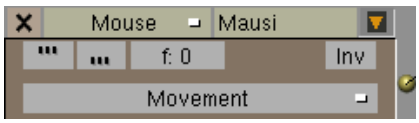
The sensor gives out a pulse when the correlate mouse button is pressed.

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.

5.1.4. 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.

5.1.5. Collision 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.

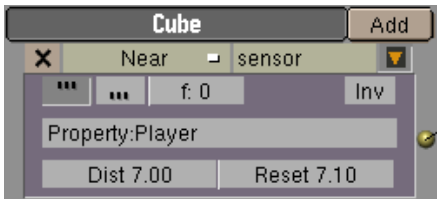
```
char* name getProperty( );
```
Gets the Material or Property the Collision Sensor reacts to.

```
gameObject obj getHitObject( );
```
Returns the colliding Object.

```
list objs getHitObjectList( );
```
Returns a list of objects that have collided.

5.1.6. Near Sensor

The near sensor reacts to actors near the object with the sensor.



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.

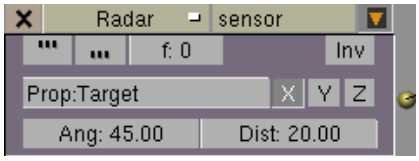
`char* propname getProperty( );`
Gets the Property the Near Sensor reacts to.

`list gameObjects getHitObjectList( );`
Returns a list of game objects detected by the Near Sensor.

`gameObject obj getHitObject( );`
Returns the object which triggered the Sensor.

5.1.7. 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 5.1.10). 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.

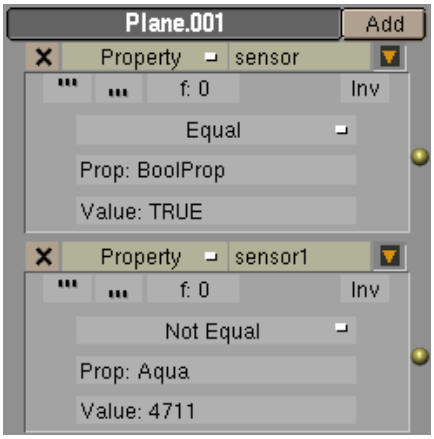
`char* name getProperty( );`
Gets the name of the Property.

`gameObject obj getHitObject( );`
Returns the detected object that triggered the sensor.

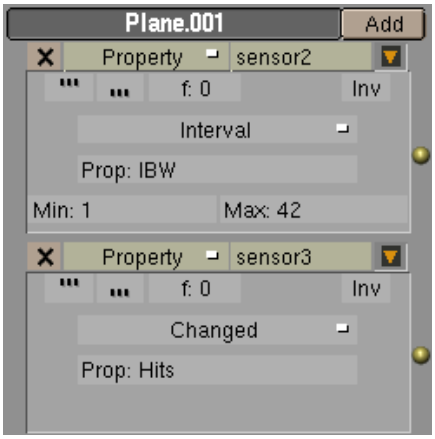
`list objs getHitObjectList( );`
Returns a list of detected objects.

5.1.8. 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

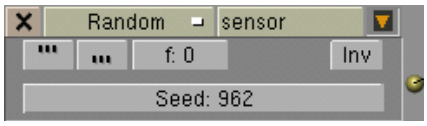
char* propname **getProperty**( );
Gets the type of the Property Sensor.

setValue((char* expression));
Sets the value to check (as expression).

char* expression **getValue**( );
Gets the value to check (as expression).

5.1.9. Random Sensor

The Random Sensor fires a pulse randomly according to the pulse settings (50/50 pick).



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.

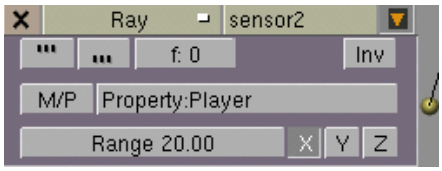
int seed **getSeed**( );
Gets the seed for the Random Sensor.

int seed **getLastDraw**( );
Gets the last draw from the Random Sensor.

5.1.10. 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.

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:

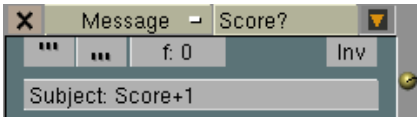
list [x,y,z] **getHitPosition**( );
Returns the position where the ray hits the object.

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.

5.1.11. 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.

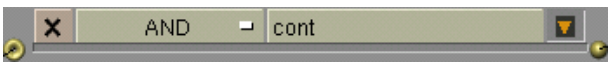
int messages **getFrameMessageCount**( );
Returns the number of messages received since the last frame.

setSubjectFilterText(string subject);
Sets the subject for the Message Sensor.

5.2. Controllers

Controllers act as the brain for your game logic. This reaches from very simple decisions like connecting two inputs, over slightly complex expressions, to complex Python scripts which can carry artificial intelligence.

5.2.1. AND Controller

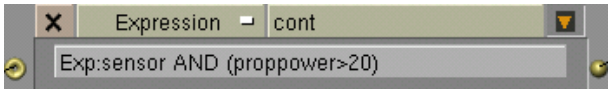


The AND Controller combines one, two or more inputs from Sensors. That means that all inputs must be active to pass the AND Controller.

5.2.2. 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.

5.2.3. Expression Controller

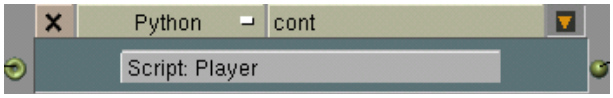


With the Expression Controller you can create slightly complex game logic with a single line of 'code'.
You can access the output of sensors attached to the controller and access the properties of the object.

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 4.9.

5.2.4. 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.

Remember that Blender treats names as case sensitive! So the script 'player' is not the same as 'Player'.

Python for the real-time 3D engine is covered in Chapter Section 6.2.

Python methods:

`Actuator* getActuator( char* name , );`
Returns the actuator with 'name'.

`list getActuators( );`
Returns a python list of all connected Actuators.

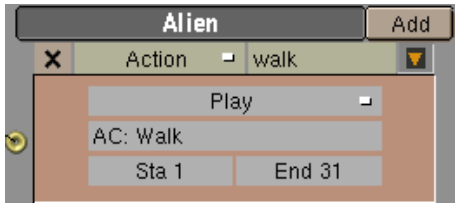
`Sensor* getSensor( char* name , );`
Returns the Sensor with 'name'.

list `getSensors`( );
Returns a python list of all connected Sensors.

5.3. Actuators

Actuators are the executing LogicBricks. They can be compared with muscles or glands in a life form.

5.3.1. 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.

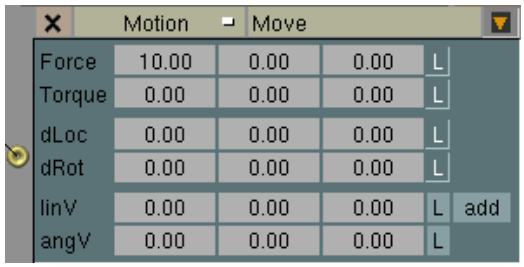
Flipper
Plays the Action from 'Sta' to 'End' on activation. When the activation ends it plays backwards from the current position. When a new activation reaches the Actuator the Action will be played from the current position onwards.

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:'.

5.3.2. 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 real-time 3D 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( list [x,y,z] , bool local );`
Sets the 'Force' parameter for the Motion Actuator.

`list [x,y,z] getForce( );`
Gets the 'Force' parameter for the Motion Actuator.

`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.

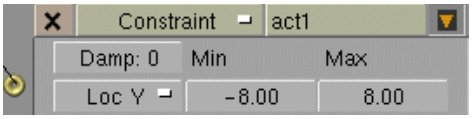
`list [x,y,z] getLinearVelocity( );`
Gets the linV parameters from the Motion Actuator.

`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.

5.3.3. 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.

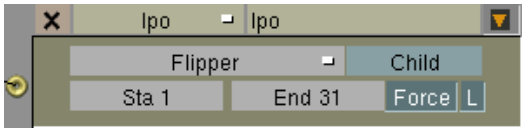
`setMin( int min );`
Sets the Min parameter.

`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.

5.3.4. 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 real-time 3D 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 frameGetStart( );
```
Gets the Sta: frame.

```
int frameGetEnd( );
```
Gets the End: frame.

5.3.5. 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.

5.3.6. Sound Actuator

The Sound Actuator plays a SoundObject loaded using the SoundButtons (see Section 4.10). 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.

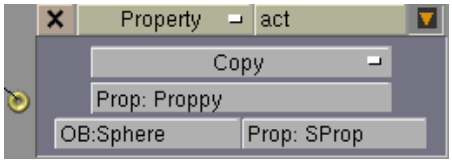
5.3.7. Property Actuator

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( );
```

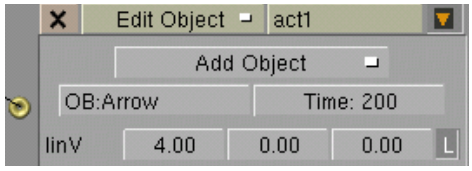
More on using Expressions can be found in Section 4.9.

5.3.8. 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.

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 real-time 3D 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.

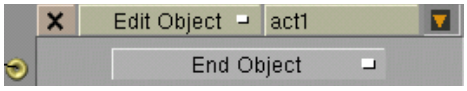
```
int time getTime( );
```
Gets the time the added object should exist.

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.

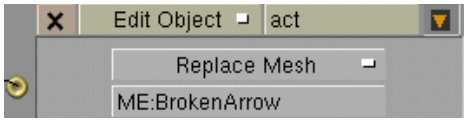
gameObject* **getLastCreatedObject**( );
Gets a pointer to the last created object. This way you can manipulate dynamically added objects.

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



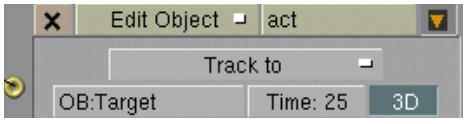
The 'Replace Mesh' type, replaces the mesh of the object by a new one, given in the 'ME:' field. Remember that the mesh name is not implicitly equal to the objectname.

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.

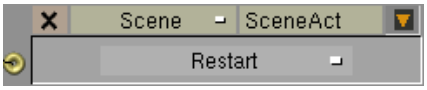
int time **getTime**( );
Gets the time needed to track.

setUse3D(bool 3d);
Set if '3D' should be used leading to full 3D tracking.

5.3.9. Scene Actuator

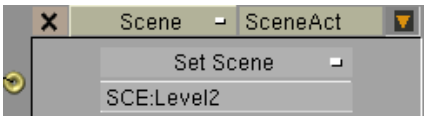
The Scene Actuator is meant for switching Scenes and Cameras in the game engine or adding overlay or background scenes. Choose the desired action with the MenuButton and enter an existing camera or scene name into the text field. If the name does not exist, the button will be blanked!

Reset



Simply restarts and resets the scene. It has the same effect as stopping the game with **ESC** and restarting with **PKEY**.

Set Scene



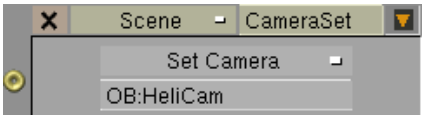
Switch to the scene indicated into the text field. During the switch all properties are reset!

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



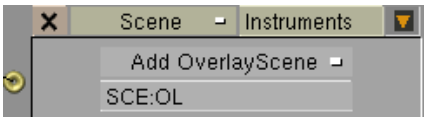
Switch to the Camera indicated in the text field.

Python methods:

setCamera(char* camera);
Sets the Camera to switch to.

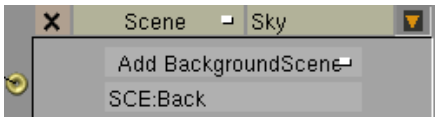
char* camera **getCamera**( );
Gets the Camera name from the Actuator.

Add OverlayScene



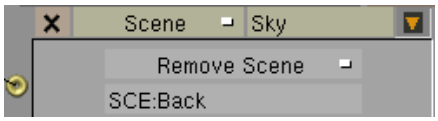
Adds an overlay scene which is rendered on top of all other (existing) scenes.

Add BackgroundScene



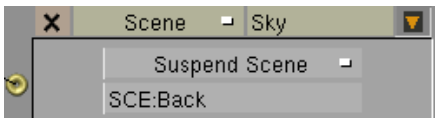
Adds a background scene which will be rendered behind all other scenes.

Remove Scene



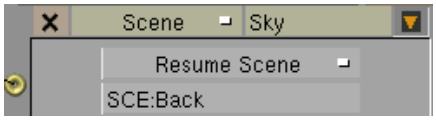
Removes a scene.

Suspend Scene



Suspends a scene until 'Resume Scene' is called.

Resume Scene



Resumes a suspended Scene.

5.3.10. 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.

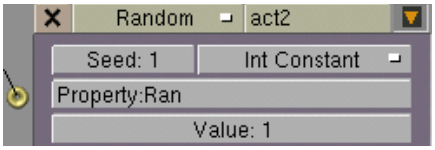
Boolean Uniform

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

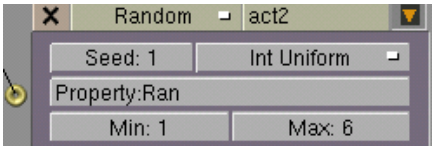
This random function results in a boolean value of TRUE or FALSE, but instead of having the same chance for both values you can control the chance of having a TRUE pick with the 'Chance' parameter. A chance of 0.5 will be the same as 'Bool Uniform'. A chance of 0.1 will result in 1 out of 10 cases in a TRUE (on average).

Integer Constant



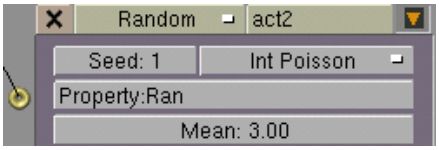
For testing your logic with a value given in the 'Value:' field.

Integer Uniform



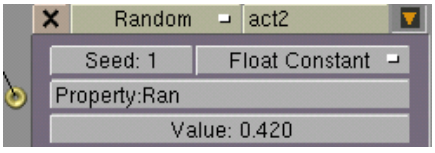
This random type randomly produces an integer value between (and including) 'Min:' and 'Max:'. The classical use for it is to simulate a dice pick with 'Min: 1' and 'Max: 6'.

Integer Poisson



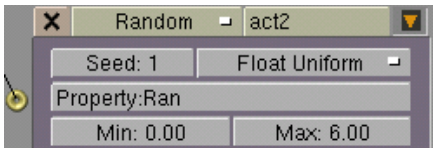
The random numbers are distributed in such a way that an average of 'Mean:' is reached with an infinite number of picks.

Float Constant



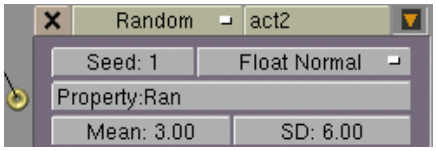
For debugging your game logic with a given value.

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.

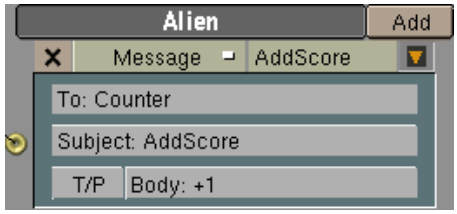
`setProperty( string propname );`
Sets the Property to which the random value should go.

`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.

5.3.11. 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 5.1.11 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.

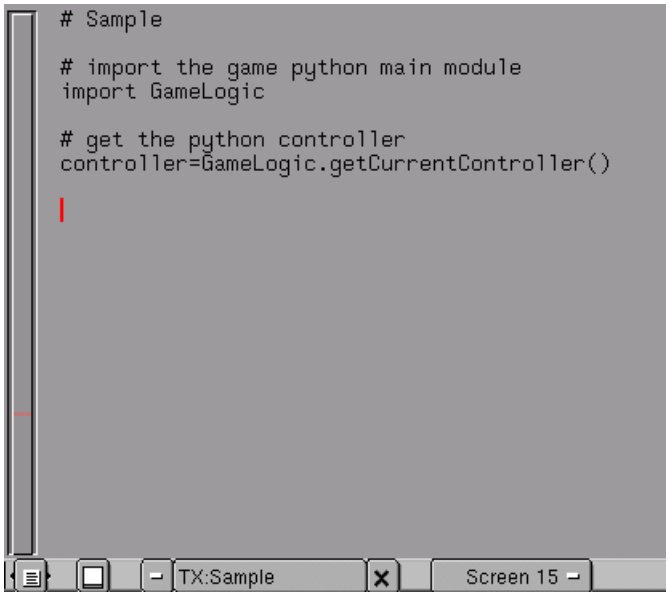
Chapter 6. Python

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.

6.1. 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 6-1. 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.

6.2. Python for games

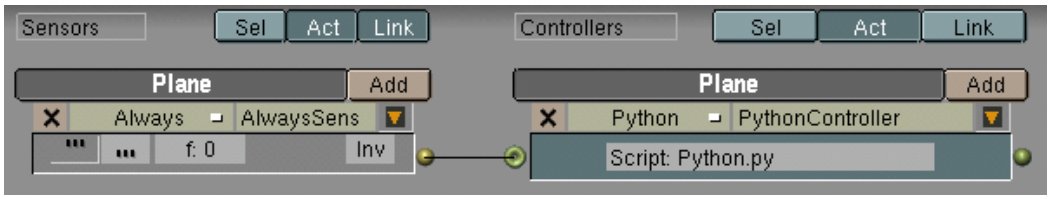
With Python integrated into the real-time 3D engine you can influence LogicBricks, change their parameters and react to events triggered by the LogicBricks. Besides that you can influence the GameObject that carries the Python Controller directly. This means moving it, applying forces or getting information from this object.

Additional to the Python in the game engine, Blender includes Python for modeling and animation tasks.

6.2.1. 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 6-2.

Figure 6-2. 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 6-3. 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 6-3: `1 print dir(owner)`

Start the real-time 3D 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', 'setOrien-tation','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.

6.3. Game Python Documentation per module

6.3.1. 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.

6.3.2. 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 real-time 3D 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.

6.3.3. 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**'.

6.4. Standard methods for LogicBricks

All LogicBricks inherit the following methods:

gameObject***getOwner**( );
This returns the owner of the GameObject the LogicBrick is assigned to.

6.4.1. 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.

6.4.2. 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.

6.4.3. 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 real-time 3D engine.

restoreDynamics( );
Suspends the dynamic calculation in the real-time 3D engine.

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 5.3. *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 5.2. *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 3D 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 **ALT-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 3D 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 3D 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 3D information: Mesh, Curve, Lattice, Lamp, etc.

OpenGL
OpenGL is a programming interface mainly for 3D applications. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Blenders whole interface and 3D output in the real-time and interactive 3D 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 the 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 5.1. *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 3D or 2D 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 3D 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.