13

Modeling Virtual Reality Web Application

Berta Buttarazzi and Federico Filippi
University of Roma "Tor Vergata"
Italy

1. Introduction

On the last decade the development of computers and ICT technology has allowed the emergence of a number of new disciplines that have significantly changed daily life and Virtual Reality is certainly one of the most interesting novelty. This technology has focused much of the work researchers. In fact in recent years virtual reality is used in many areas: robotic, simulations, training, entertainment (Video games offers fantastic characters to play and a vast interactive worlds to explore), medicine, etc. Even entire training programs can take place in a virtual reality. In this scenario at first was proposed various languages to introduce the third dimension but not to interact with objects. For against, 3D objects interaction can offer extremely realistic representations that enable users to move and react in a computer simulated environment.

To this purpose few attempts have been made to offer the opportunity to build virtual worlds in what you are completely immersed and where is possible to interact both objects it contains both with people represented by avatar, allowing to hear sounds, see pictures and movies, providing perceptions very similar to real ones.

Though Virtual Reality has reached a mainstream status, actually creating 3D application still requires much effort and only with proper, dedicated tools is it possible to produce convincing result in a timely manner.

Therefore in this chapter we introduce Quest 3D, that is a perfect software package for creating interactive 3D scene including product presentation, architectural visualizations, virtual trainings and computer games.

Quest 3D is a software for creating interactive 3D scene, in a unique style of programming. Instead of having to write thousands of lines of complex code, developers make use of a large set of a powerful building blocks. Both flexible and easy to use, Quest 3D appeals to designers, programmers and artist alike.

Working with Quest 3D means developing in a real-time: you are working directly on the end result. No time will be lost on compiling code or rendering image. While the logic tree scanning goes on, system shows the visualization of scene in the window application.

Quest 3D has a stunning set of graphics features. Large numbers of animated people, vegetation, shadows, fire and smoke effects and realistic water can all be easily added to a scene. Advanced features include physics simulation, path finding routines, database connectivity and networking support. By this way is possible design very simple scene like extremely complex projects: software product always will appear readable and smart. The high number of building blocks can be logically regrouped in folder, characterized by

Open Access Database www.i-techonline.com

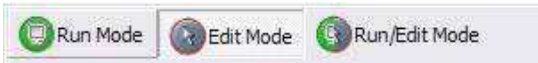
Source: Advances in Robotics, Automation and Control, Book edited by: Jesús Arámburo and Antonio Ramírez Treviño,
ISBN 78-953-7619-16-9, pp. 472, October 2008, I-Tech, Vienna, Austria

different name and colour for the various functions: a structure very similar to Java or C class. Blocks take input from the son blocks, which provide services to father. Every scanning of blocks follow a depth first manner from left to right in a single frame, realizing the real time rendering.
Quest3D gives the possibility to manage the logic of scene: the behaviour of objects, their interactions.
Programmer can follow an advanced and sophisticated approach of dynamic loading of the various portions of program, to isolate the different logical parts. This solution provide the possibility to use the all potentialities of graphic card, still limited for household approach of so heavy software programs.

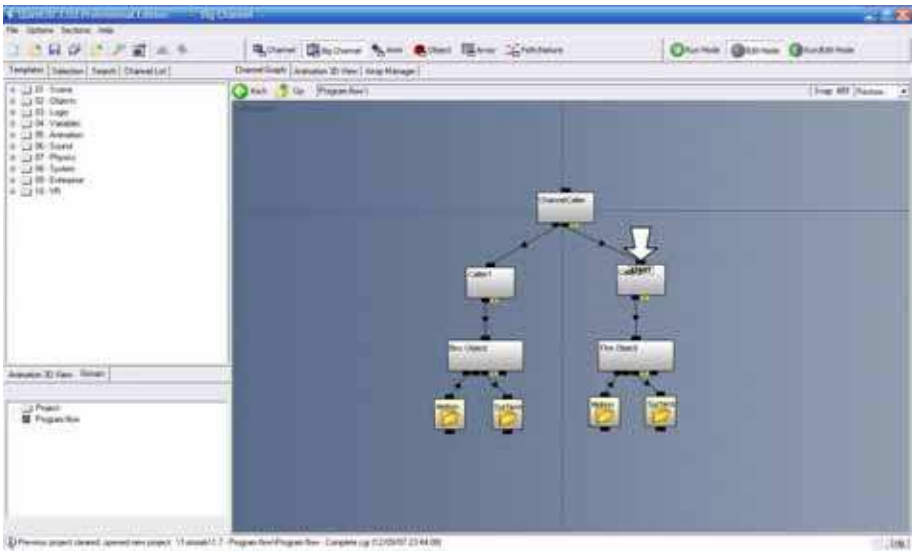
2. User Interface

Quest3D provides an *user-frienly* approach, by a graphic user interface, easy understandable, formed by windows, menu and other direct interaction elements with the developer. Let's show the elements:
This is more productive than worrying about how to achieve consistent results in different hardware and software environments.
Toolbar for working on a scene through the different perspectives

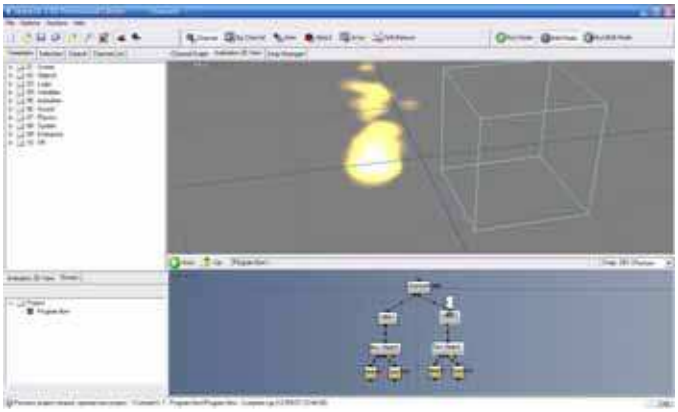
Toolbar for working on a scene through the different perspectives



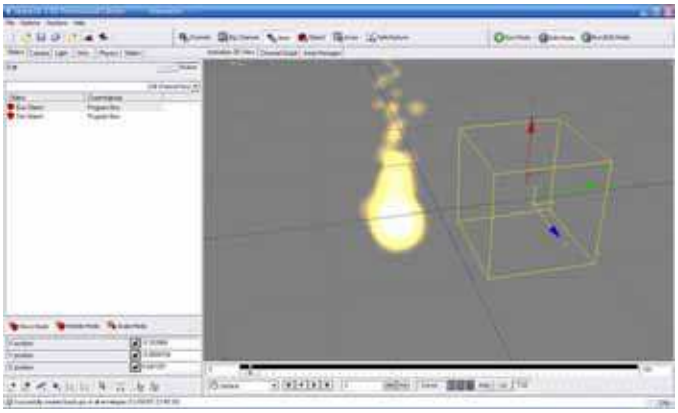
Switch buttons for modifing scenes (*Edit*) and visualize animate scenes (*Run*)
A scene: a set of *channels* connected by functional and hierarchical liabilities



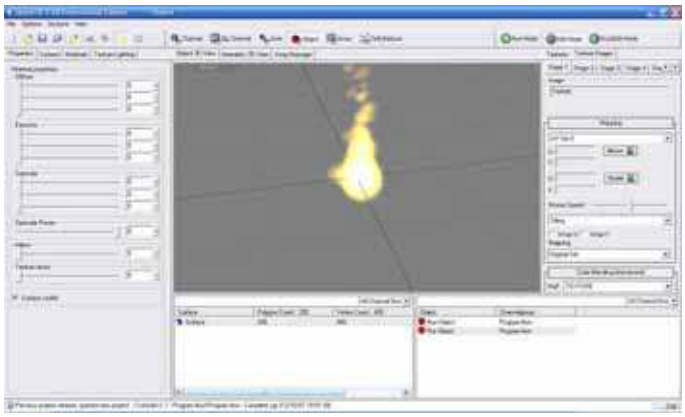
Channel sections show the structure of building blocks of a scene



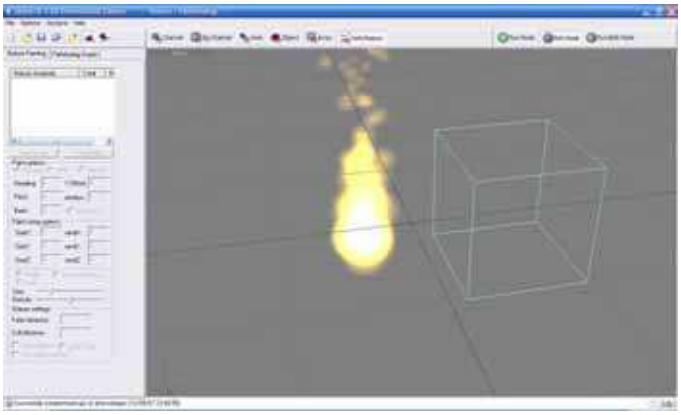
Animation section allow to animate a scene, deciding the behaviour in different temporal instants



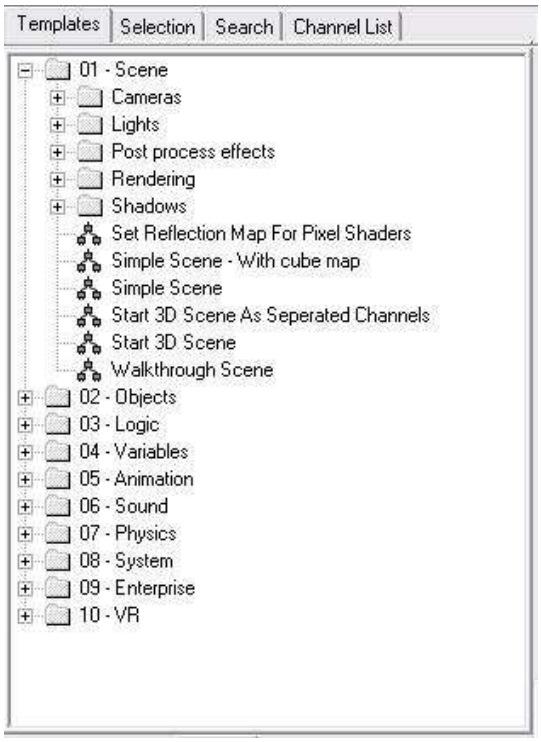
In *Object* section is possible select single objects which form the scene and its surfaces



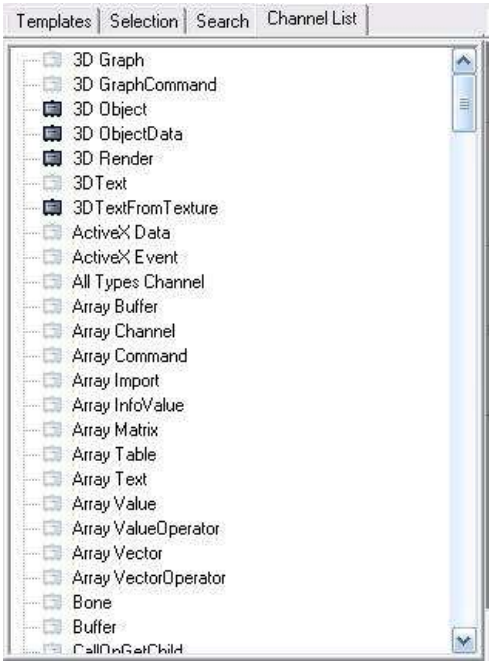
Path/Nature section allow to add naturalistic elements and phenomenon which make more relistic the scene



The elements which compose the scene are not created suddenly, but belong to the available channel set

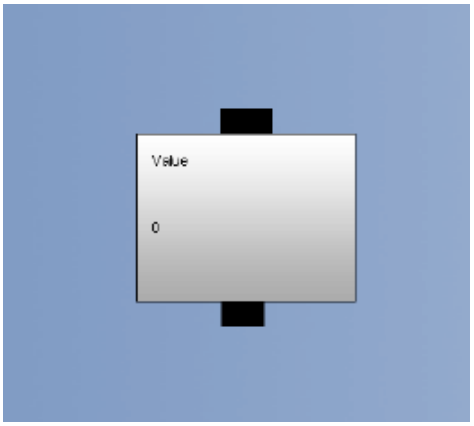


To semplify programing, Quest3D provides every kind of templates: predefined objects, cameras, lights, instruments to add audio file o creating animations, logical-mathematical funtionalities

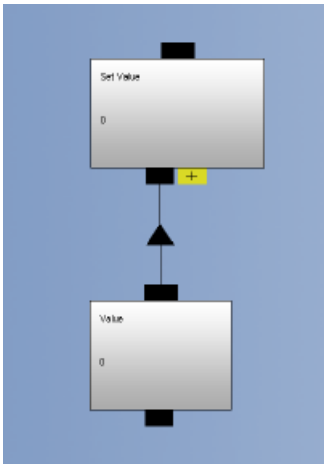


3. Channels

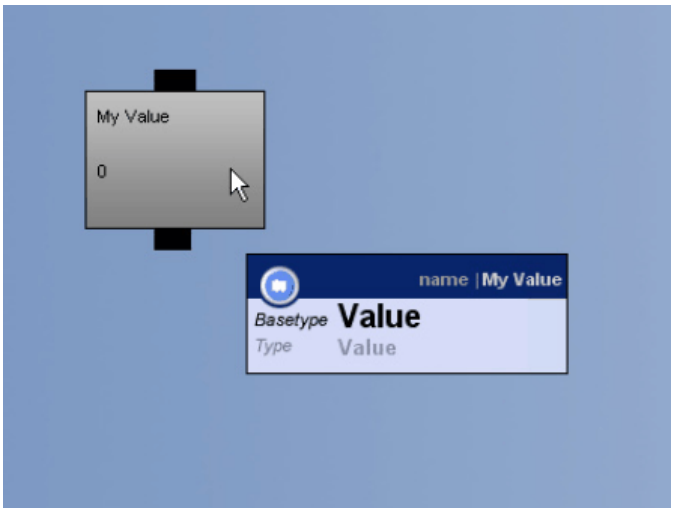
Quest3D programs are made with building blocks. These building blocks are called *channels*. Each channel has a specific function.



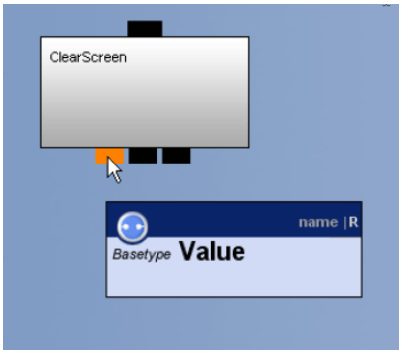
The small black squares above and beneath a channel are called *link squares*. Channels can be connected to each other by lines between top and bottom link squares.



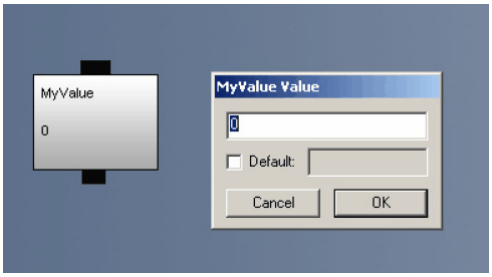
In this image, the upper channel is called the *Parent*. The lower channel is called the *Child*. Children are often used as input or output data for Parents.
The small black squares show which necessarily have to be the *channel* tipology of *childreen*, which *parents* may use like input.
Informations about a building block are visible, throught mouse moving, on specific pop-up windows.



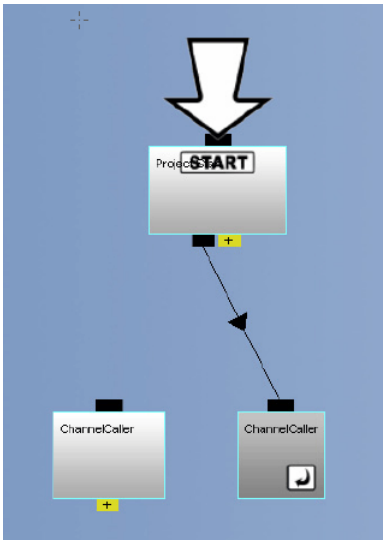
In addition to a name, every channel has a *Basetype* and a *Type*. A *Value* channel is of Basetype *Value* but also of Type *Value*.



The connected *channels* structures compose a *Channel Group*. A Quest3D project is formed by one or more *Channel Group*.



Double clicking on a channel to visualize *Channel Properties* windows, where are specified the characteristics of a block (name, value, behaviours).

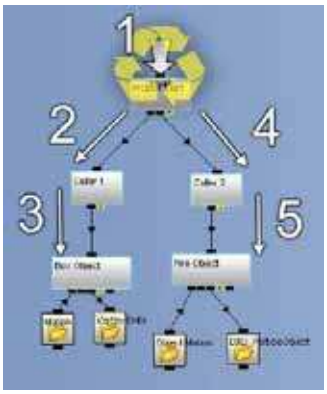


Shortcuts are references to a channel: instead of duplicating a block, is sufficient creating a shortcut (throught the menu by right click of mouse) to inheriting all the characteristics of the block.
Shortcuts are references to the original channel. Their only purpose is to keep Channel Groups more readable.

4. Program flow

A rendering engine is a software which, once download contents (from Internet, like a page HTML or XML, an image, ecc.) format the relative informations and show them on screen.
Quest3D works in real-time. This means that continually executes a complete project and upload the preview.

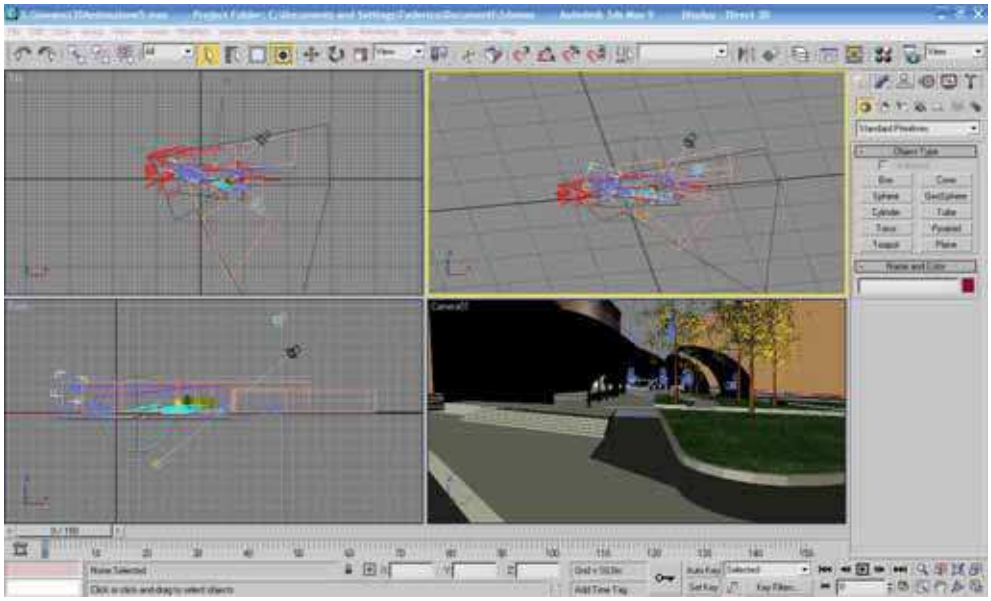
A complete cycle throught the *channel structure* is called *frame*: the set of all frames produces the rendering.



Framerate defines how many times on one second a program is executed, it depends on project complexity and on available hardware.

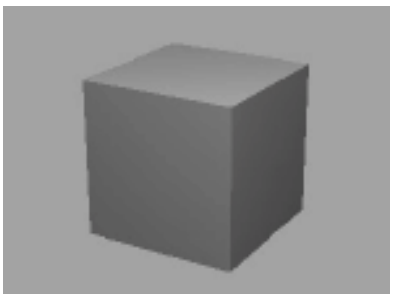
5. Modeling import

Quest3D is not used to create complex models from beginning. it allow to create interactive 3D scenes architectural presentations, virtual training, computer games, importing entities from available modelling tools.
Models are meanly created by Autocad, 3D Studio Max and Maya, and exported to Quest3D. Exporter of different tools are available on www.quest3d.com.

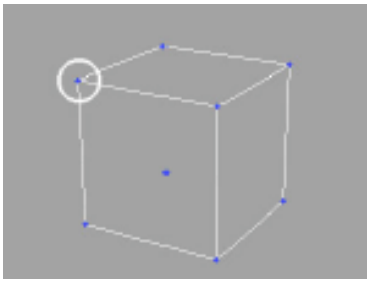


6. 3D Objects

3D scenes consist of virtual objects. Their appearing is defined by a number of elements.

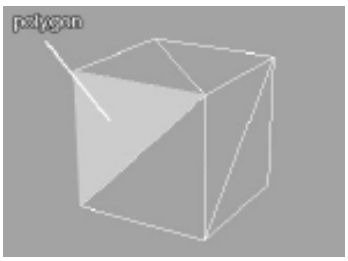


A vertex is a single point in 3D space. Vertexes are defined by X, Y, Z coordinates.



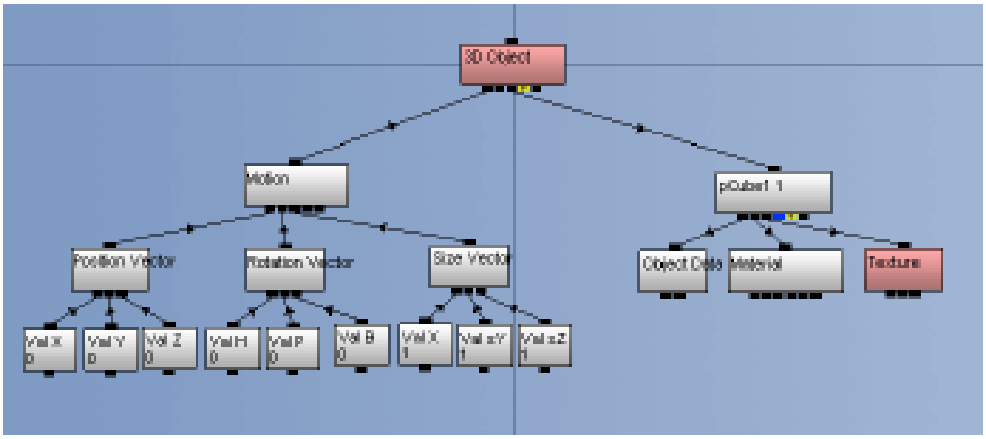
Simple models like a box consist of only few vertexes, while more complex objects can be built out thousands of points!

Surfaces consist of polygons, everyone of them could have an own vertexes number, although real-time graphics usually renders 3 or 4 (3 vertexes for each surface of Quest3D polygons).



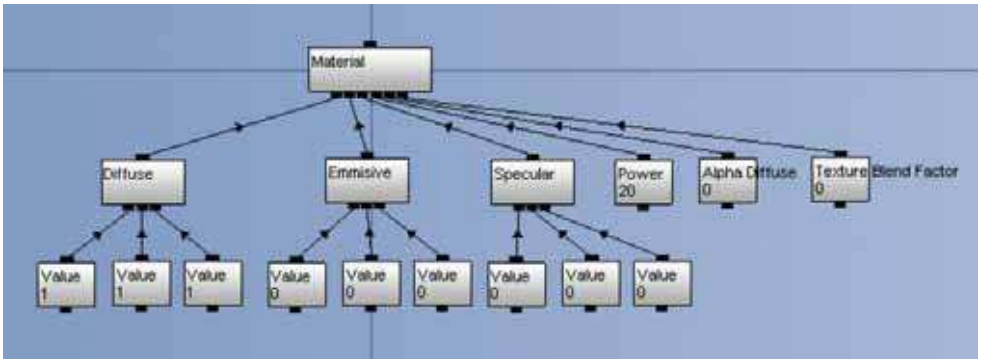
Individual polygons can be assigned a color, but often, another means to add detail is chosen. A ‘texture’ is an image that can be wrapped around a 3D shape.

Textures are wrapped around an object, to define its color and transparency.



Tipical components of a *3D Object*:

- The *3D ObjectData* channel contains the vertices and polygons of an object. It also contains information on how to wrap a texture around the shape.
- The *Material* channel is subdivided into a number of aspects: Diffuse, Emissive, Specular, Power, Alpha Diffuse and Texture Blend Factor.
- Il*Texture* channel represents the actual image applied on a surface: it contains a diffusive colour image (RGB), and an alpha image.
- Il channel *Motion*, throught its parameters, manages object's position in space.



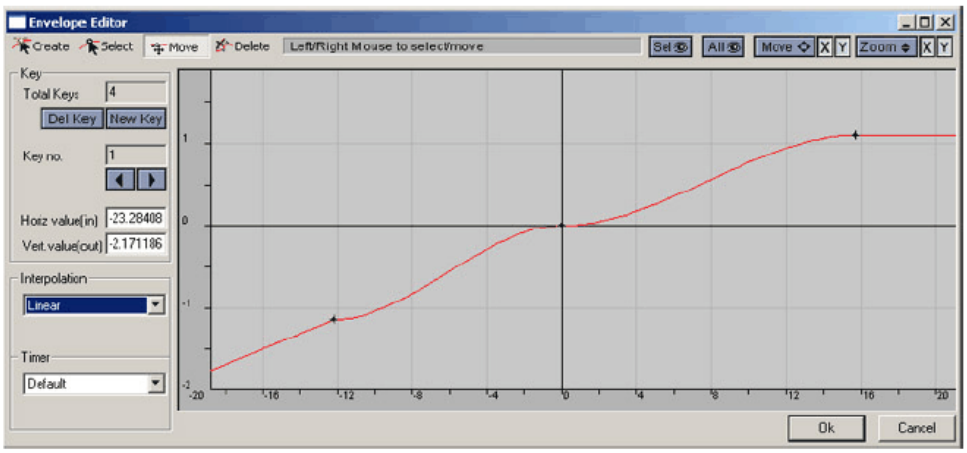
- The *Diffuse vector* contains an object's colour, and consists of three components: red, green and blue, or '*RGB*'.
- The *Emissive vector* describes the amount of self-lighting the model has. Without lights in your scene, an object with an Emissive vector of (0, 0, 0) is black.
- Lo *Specular vector* defines the colour and intensity of the highlight, while the Power value controls the size of the highlight.
- The *Alpha Diffuse* value defines the transparency of an object.
- The *Texture Blend Factor* value defines how much of a texture is blended with the Diffuse colour.

7. Animations

One of the most powerful features of Quest3D is that you can animate virtually every aspect of your scene. For instance, 3D objects can be moved, rotated and scaled over time. In addition, all of their surface properties can be animated, such as their color and transparency values. 3D character models can be brought to life by animating virtual skeletons inside of them. Music and sound can be triggered on certain location- or time-based events. Menu windows can be moved or faded in and out of the screen.
To realize animations Quest3D uses:

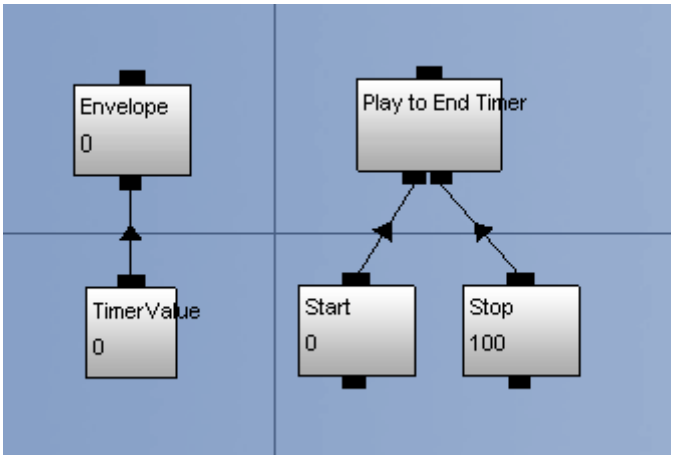
- *Envelope*

In Quest3D, animation often means changing values over time. The regular *Value* channel can only store one value. To store animation data, the *Envelope* channel is used. The interface of the *Envelope* channel looks like a mathematical graph. It works in a similar way as well. The horizontal axis represents the input (often time), which can be fed to the *Envelope* by connecting a (*Timer) Value* to its child link. The vertical axis represents the output. Through (X, Y) pairs and interpolation, any input can be converted to an output value.



- *Timers*

The *Timer Value* and *Timer Command* channels allow for easy animation control.
Examples of Timer Commands are 'Play', 'Stop', 'Rewind' and 'Play & Loop'.

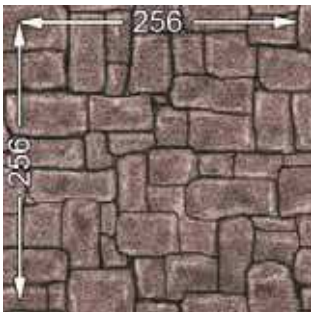


Timer Value channels are automatically adjusted for computer speed. In Quest3D, 25 frames equal exactly 1 second.

A Timer can be renamed throught the channel property window.
Each element of the scene can need of an own timer.

8. Surface properties

3D object appearing is defined by surfaces properties.



A 'texture' is an image that can be wrapped around a 3D shape. In real-time graphics, for performance reasons, their dimensions in pixels (width and height) must be a power of 2. In

Quest3D, they must also be greater than 8. Dimensions of 32x32, 128x256 and 256x256 pixels are all valid sizes.

Compression:

In addition, textures in Quest3D may be compressed. The image quality might slightly decrease, but compressed textures may use up to four times as little memory as uncompressed versions. Creating a 3D scene is constantly balancing between quality and performance.

Mipmaps:

Larger textures on distant surfaces might produce artifacts. By using various versions of increasingly smaller size, these distortions may be countered. These smaller versions of the same texture are called *mipmaps* and can be generated automatically by Quest3D.

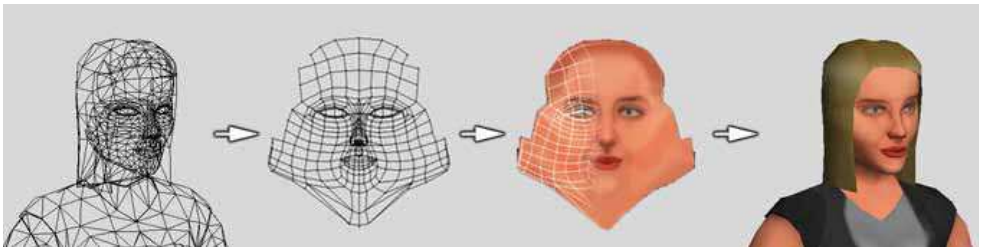
UV Mapping:

Textures are wrapped around a 3D object according to so called 'UV coordinate sets'. UV coordinates specify where each pixel of a texture is drawn onto a surface.

Quest3D can handle simple *UV mapping* methods such as *planar* and *cubic*, shown in the image below. These simple UV mappings can be moved and scaled.

Since UV coordinates are relative, once an object is properly 'UV mapped', textures of any size may be applied to the object.

Wrapping a texture around a 3D object usually requires more control and precision. In practice, UV mapping means placing every vertex of a 3D object onto a 2D image.

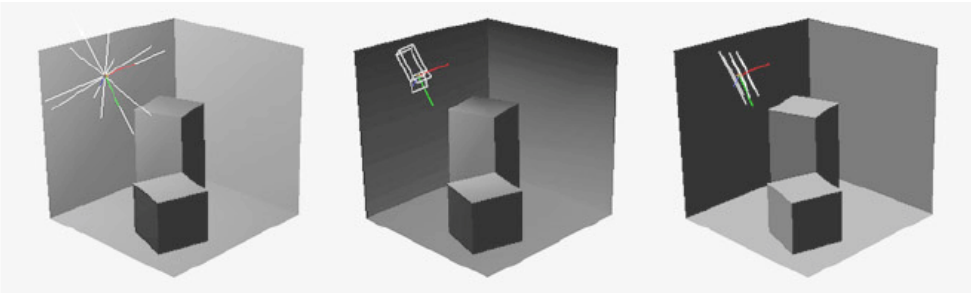


Transparency:
Transparency of surfaces may be handled in a number of ways, managing the characteristics of surfaces in *Object section*.
Texture stages:
More than one texture may be applied on a surface. The maximum number of 'texture stages' that can be used per surface depends on the graphics hardware. Most graphics cards support two stages, and newer graphics cards support three or four.
Quest3D tool allows different methods of textures blending on each stages.

9. Lighting and shadows

Without contrast between light and dark, it is much harder to establish depth in a scene.

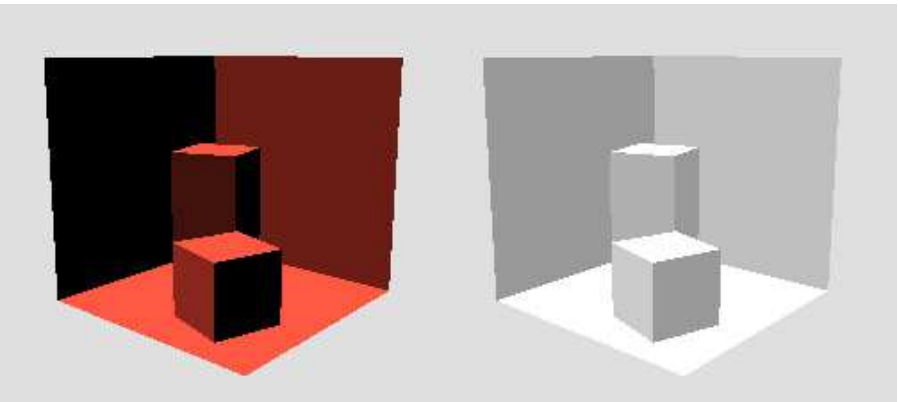
A right lightings, the shadow contrast are mean elements to create a realistic 3D environment! A number of lighting techniques are available in Quest3D:



A point light radiates light into every direction. A spotlight is defined by an inner and outer cone angle, resulting in a simulated fall-off effect. A directional light is simply a rotation vector along which the source emits light. All rays of a directional light are parallel to each other.

A Quest3D light can be colored through its 'Diffuse' vector. The 'Ambient' vector serves as a global 'Emissive' vector to all objects affected by the light source.

A 3D object's illumination may be influenced by changing its Diffuse and Emissive vectors. These can also be accessed in the Object Section, on the 'Material' tab. A higher Emissive vector will suggest a certain degree of self-illumination or ambient light.



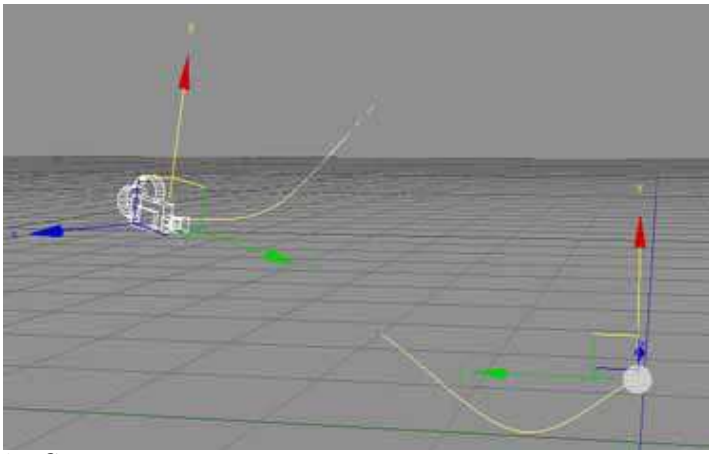
Shadows for dynamic objects, such as cars or walking characters, can be simulated using texture squares.

Quest3D also supports real-time shadows. The *Stencil Shadow* channel can be seen as a *Render* specialised in calculating shadows. The *Stencil Shadow* channel requires a light source position vector to work. 3D Objects may be linked to the *StencilShadow* by means of a *SoftwareStencilShadowObject*.
Real-time shadows require a lot of processing power.

Light mapping is the process of calculating lighting and shadows once, and storing the information into a texture. The resulting 'light maps' are then applied on an object on a second texture stage, and blended with the diffuse texture.

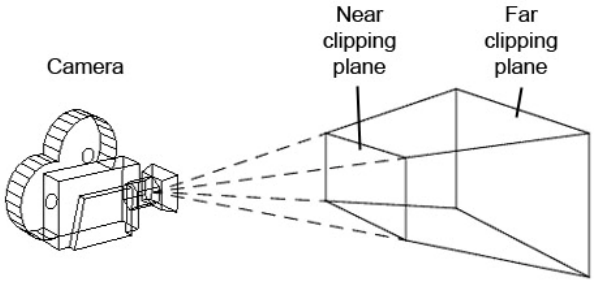
10. Cameras

A camera defines the point of view from which a scene is presented. Choosing the right camera for a project is important, as it influences the way users experience a scene in a very direct way.
Let's show the camera *templates* available in Quest3D.
Animation Camera:
An *Animation Camera* may be used to guide a user through a scene. The animation camera is similar to a real-life freehand cinematic camera. It may also be the preferred camera for non-interactive scenes such as visualisations, documentaries or screensavers.
Animation Camera with target:
An *Animation Camera with Target* always looks at a dummy object. Both the camera and the helper object can be animated.



Object Inspection Camera:
Quest3D is the perfect product visualisation package. An *Object Inspection Camera* can orbit around a 3D model, enabling the user to view the object from all angles. Such 'products' might be as small as technological gadgets, or as large as entire cityblocks.
1st Person Walkthrough Camera:
It is often desirable, and rewarding, to experience an architectural visualisation as if you were there. A *1st Person Walkthrough Camera* presents a scene at regular eye level, and allows the user to walk around freely. Since the view is close to real life, this type of camera provides the highest level of immersion in a scene. Many computer games make use of the 1st person perspective.
3rd Person Walkthrough Camera:
In contrast to the 1st person point of view, a 3rd person perspective places the camera just outside of a virtual character. The *3rd Person Walkthrough Camera* is attached to this character, or *avatar*, and follows it wherever it goes.
Every kind of camera is defined by particular characteristics:
zoom factor:
The zoom factor is a quick way to zoom a camera in or out.
The default value of '1.0' results in a normal view. A higher Zoom Factor brings the scene closer, a lower value sets it further away.

clipping planes:
The *clipping planes* of a camera are virtual boundaries between which a scene is rendered. Polygon surfaces outside of these boundaries are not calculated.
fog:
Fog can be used to blend various elements in a virtual environment together. It can add to the mood of a scene and suggest depth.
Fog can be used to blend various elements in a virtual environment together. It can add to the mood of a scene and suggest depth. A bonus of using fog is that the far clipping plane of the camera may be decreased to just behind the *Fog End* value. If used right, this will result in a smooth transition between the scene and the "nothing" beyond the far clipping plane.

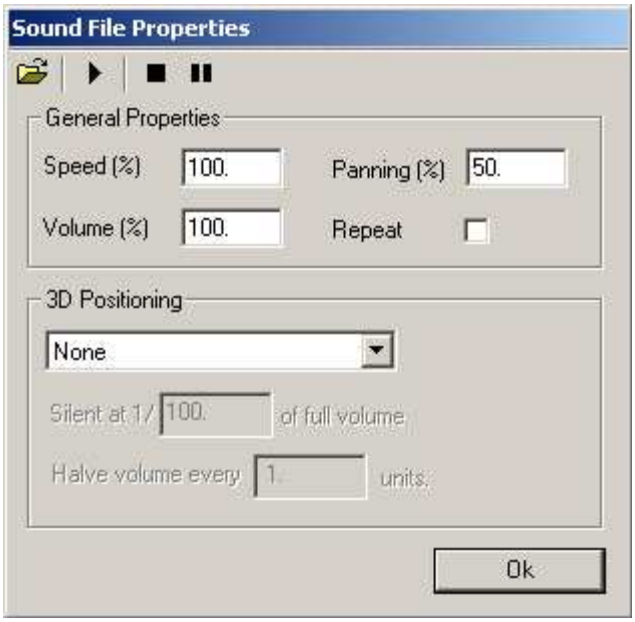


11.Sounds

From simple menu click sounds to full ambient music, *audio* can really add to a scene. It helps to set the mood, just as sharp graphics and lively animation can. The use of spoken voice samples can help users to better understand an application. Turn off the sound while watching television or playing a computer game, and you'll instantly miss the richness of audio.

Sound can be stored digitally on the computer, in many different formats. Quest3D supports both '.wav' and '.mp3' files.

The *Sound File* channel is used to store *.wav* samples within Quest3D. Its properties window contains a number of options including icons to load, play, stop and pause a sample. As the name suggests, the *Repeat* option allows for continuous playback.



Speed ranges from 0% to 200%, and volume ranges from 0% to 100%. Both can be dynamically changed through their respective child links.

Panning controls how the sound is divided between the left and right speaker: 0% means only the left one is used, 100% means the right one. 50% (default value) plays the sound at equal levels through both speakers.

The *Sound Command* channel allows you to control all aspects of a sample, such as playback and volume. It also allows for loading '.wav' files from harddisk. The drop-down list presents all the various options. A *Sound Command* must be called once to be executed, and affects all its children of the *Sound File* channel type.

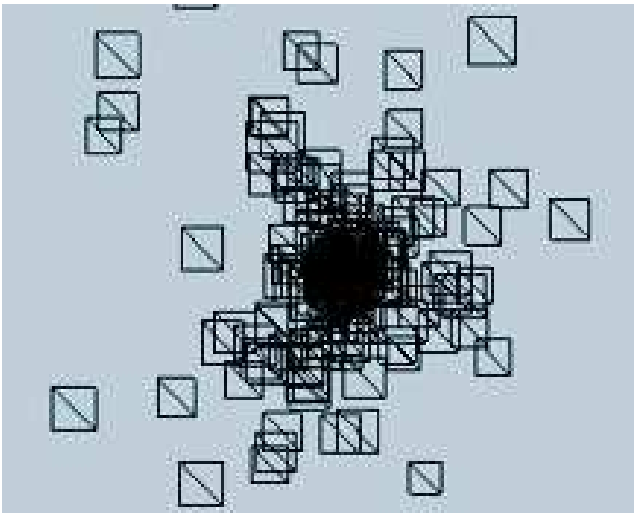
Besides simply playing a sound, it is also possible to place it in 3D space. 3D Positioned sound requires a *Listener* channel. It is usually connected to the location of the virtual camera, and automatically adjusts volume, panning and frequency by calculating a sound's relative position and distance. Only Mono sound samples can be positioned in 3D space.

The *3D Positioning* options of the *Sound File* channel define the quality of the simulation. Higher quality requires more cpu power.
The '.mp3' format is also supported in Quest3D. The *MP3 File* channel is used to store sound samples of this type.
Through its properties window it is possible to load a file from disk. The *Save File in Channelgroup* option allows you to store the sample data inside of the channel itself. This will of course increase the file size of the channelgroup. The *MP3File*'s only child may also contain a file name in the form of a *Text* channel.
The *MP3 Control* channel can be used to control playback and volume. It can also be used to retrieve information on total length and current playing position, and to skip to a specific position in the sample.

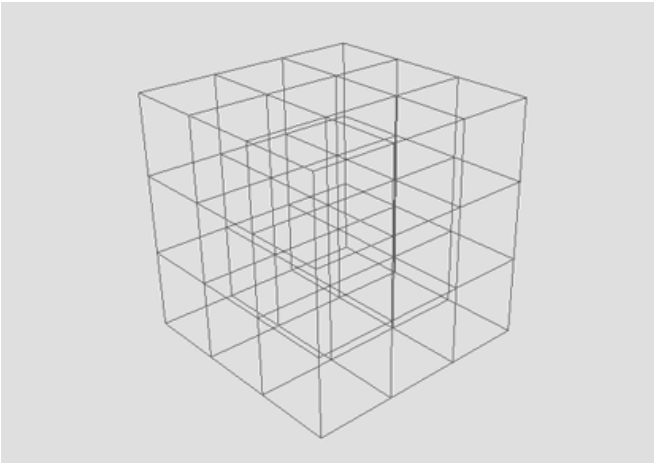
12. Particle sistems

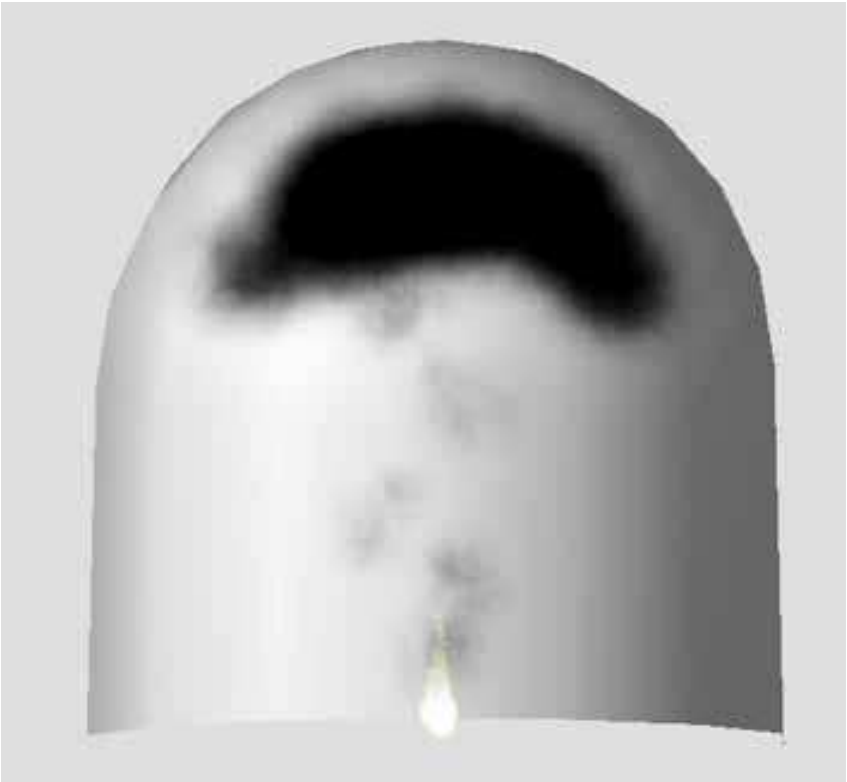
Many special effects in movies and games, such as fire, smoke, explosions and all kinds of magical effects, have one thing in common: they are all created using so called *particle systems*.

A particle system is a collection of objects released from an emitter and usually moved and changed over time. Particles are square objects facing the camera.
For every particle in the system, the position, rotation and size is stored. In addition, other attributes such as speed, color and transparency may be used. Forces such as gravity may be applied to all particles in the system.
Particles are released from an emitter. In Quest3D, this is in the form of an *3D Object Data* channel.
The particle system cycles through all of the vertices in the object and constantly emits a particle from one of these positions.
By randomly placing a number of vertices in 3D space, it appears as if the particles are released randomly as well. This produces convincing results for special effects such as fire and smoke.



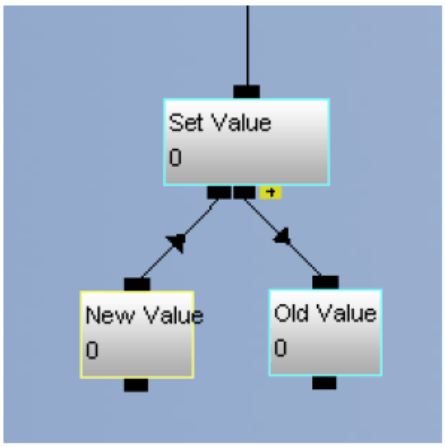
A *Particle Flow Grid* channel can be used to control the flow of the system. It divides a collision object into several parts and calculates forces inside of each of these. Forces can be created by the collision object, external objects and the particles themselves.

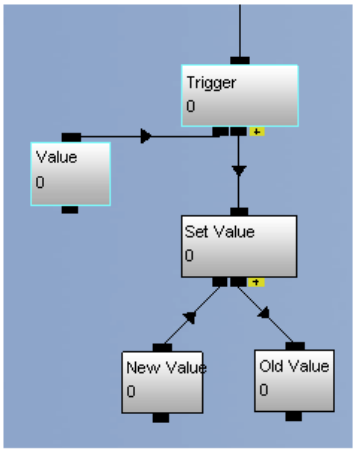




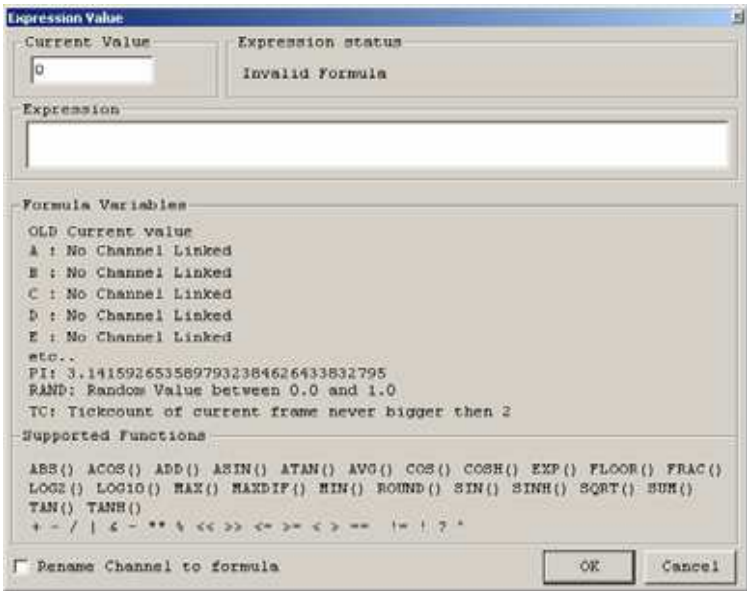
13. Mathematical operators

Values are at the heart of any Quest3D project. They are used to define an object's color, position and size for example. Values can also describe scores, program settings, menu option etc.
A simple instruction which describes mathematical expression is *Set Value*, which realizes a simple value association:

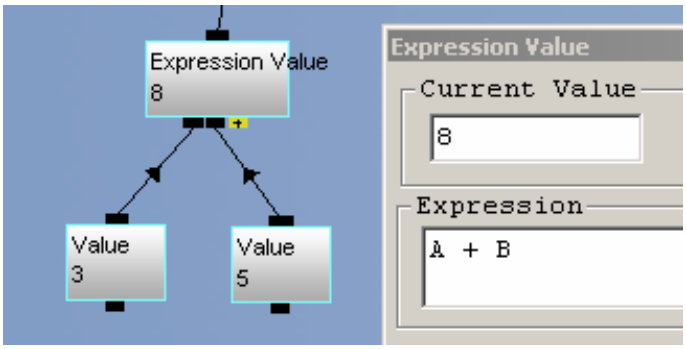




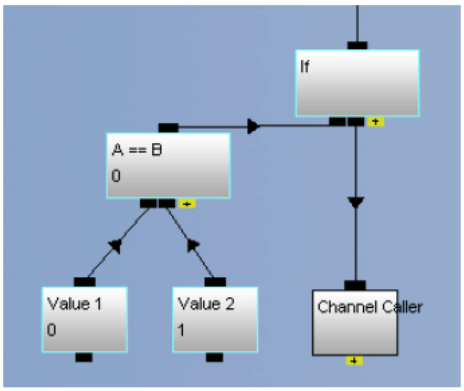
In most practical cases *Set Value* channels need to be carried out only once. Therefore, they are often called by *Trigger* channels.
Mathematical expressions can be described in Quest3D using the *Expression Value* channel. It serves as an advanced calculator. Its properties window contains a box for the actual formula, and help text describing all the usuable variables and operators.

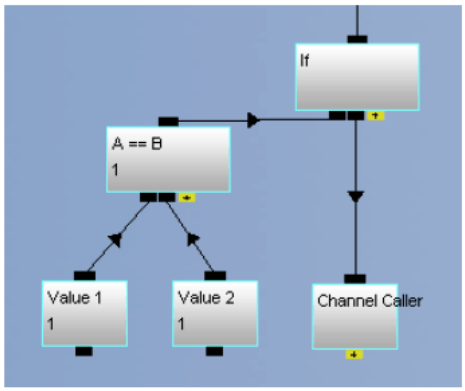


Values connected to the child links of the *Expression Value* can be used as part of the expression within the channel as well. The first child is presented by the letter 'A', the second by the letter 'B', and so forth.

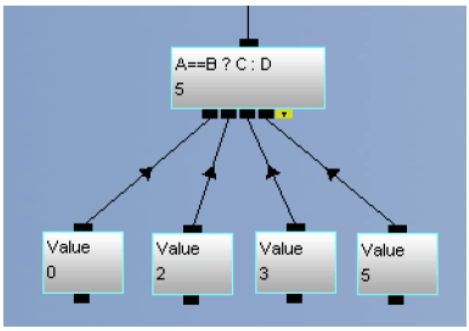


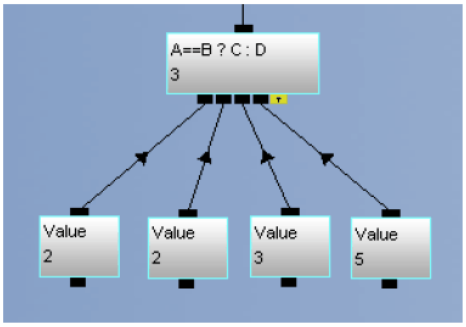
Multiple expressions can be linked to one another within the same channel using the logical operators AND (&&) and OR (||).
In this example the condition is met only if A==1 && B==2.





In this other case, the expression A==B ? C:D is based on identity between A and B.



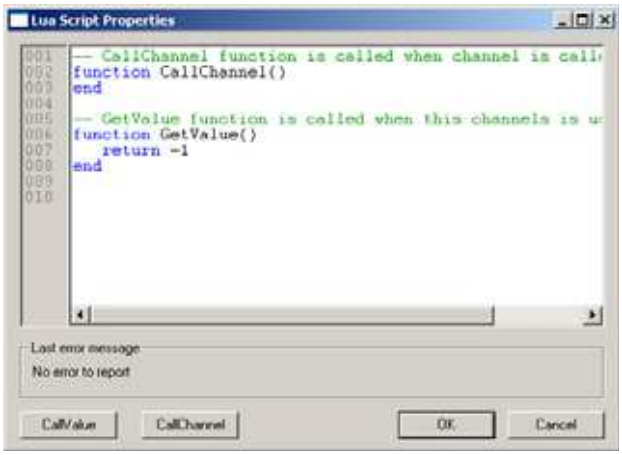


Mathematical functions are realized throught the *Envelopes*, main instruments for managing animations.
For advanced simulations and models, actual formula's may be used by inserting a number of coordinate pairs into the graph. Quest3D will handle the interpolation between these points.

This *Envelope graph* show the formula $y = x^2$

14. Lua Scripting

Quest3D channels are building blocks containing precompiled C++ and DirectX code. The entire set of channels provides functionality for many aspects of real-time 3D development. Despite the open and logical structure of the channels system, certain functionality can be handled more efficiently or more easily in script language.
Quest3D supports *Lua*, a free third party scripting environment.
Lua in Quest3D is especially applicable for cases such as loading and unloading channel groups, complex calculations and iterative structures (*for loops*).



Lua channels can have one of two functions, or both:

- The first is *CallChannel*, and is executed when the channel is simply called;
- The second is *GetValue*, and is executed when the *Lua* channel is used as a value. The actual value of the *Lua* channel itself is equal to the value the script 'returns' at the end of the function.

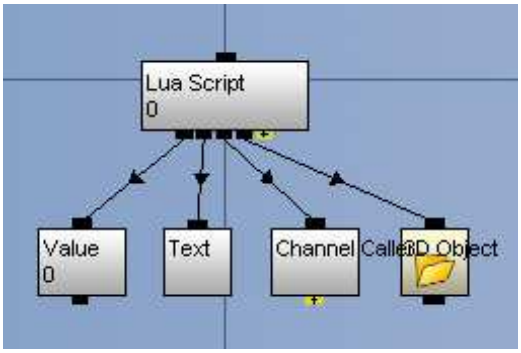
Just like in other programming and scripting languages, variables used in a *Lua* script must be *declared*. This statement assigns memory to the variable. The word *local* means that the variable is only used in one particular place, in this case the *Lua* channel. As soon as the script is done, the memory is cleared again.

```
local variable
local variable = 1
variable = variable + 1
```

Quest3D supports the following standard Lua group functions: *base*, *string* and *math*.

```
local variable = math.cos(value)
```

The first example calculates the cosine of the value between the brackets.



Any number of children of any type may be connected to a *Lua* channel.
Children may be accessed in a *Lua* script using the appropriate function.
As usual, they have to be declared first. The statement below accesses the first child connected to the *Lua* channel (at position '0').

```
local variable = channel.GetChild(0)
```

The actual value of a *Value* channel connected to a *Lua* channel may be accessed using the following commands.

```
local variable = channel.GetChild(0)
local value = variable:GetValue()
```

Text can also be retrieved, using the following statements:

```
local variable = channel.GetChild(0)
local text = variable:GetText()
```

The value or text of a child may be set using the following commands:

variable:SetValue(value)

variable:SetText(text)

Note the difference between using functions of predefined and local structures. For predefined structures such as 'channel', a function is preceded by a '.' (dot). For local structures such as variables, a function is preceded by a ':' (double colon punctuation mark).
Functions to load or unload *channel group* are very usefull.
For the *LUA group q*, the following instructions realize these statments:

q.LoadChannelGroup("group.cgr", "PoolName", instance)

q.RemoveChannelGroup("PoolName", instance)

15. Product publications

At a certain point, a Quest3D project may be finished and ready for distribution. Whether it is a modest free screensaver or an extensive commercial training application, the Quest3D program must be exported out of the editor and saved into a format the target audience can run.
Different options are available to publish a Quest3D project:

Can publish to type
.q3d player file
Web application
Screensaver
WinAmp plug-in
Stand-alone self-running .exe
Installer file

Internet ActiveX: .q3d and .htm:
files of type *.q3d* can also be viewed in an internet browser that supports ActiveX. The *.q3d* file must be linked from a regular webpage to show. Web pages containing Quest3D media elements will automatically try to download and install the Quest3D Viewer Package from the Quest3D.com website (www.quest3d.com).
Stand-alone programs: .exe:
a Quest3D project can also be published to a stand-alone self-executing *.exe* file. Besides DirectX, this type of publication requires no additional components.
Installation files: .exe:
installers are *.exe* archive files that copy and expand a program to a specified directory on a harddisk.

16. References

www.quest3d.com

Advances in Robotics, Automation and Control

Edited by Jesus Aramburo and Antonio Ramirez Trevino

ISBN 978-953-7619-16-9
Hard cover, 472 pages
Publisher InTech
Published online 01, October, 2008
Published in print edition October, 2008

The book presents an excellent overview of the recent developments in the different areas of Robotics, Automation and Control. Through its 24 chapters, this book presents topics related to control and robot design; it also introduces new mathematical tools and techniques devoted to improve the system modeling and control. An important point is the use of rational agents and heuristic techniques to cope with the computational complexity required for controlling complex systems. Through this book, we also find navigation and vision algorithms, automatic handwritten comprehension and speech recognition systems that will be included in the next generation of productive systems developed by man.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Berta Buttarazzi and Federico Filippi (2008). Modeling Virtual Reality Web Application, Advances in Robotics, Automation and Control, Jesus Aramburo and Antonio Ramirez Trevino (Ed.), ISBN: 978-953-7619-16-9, InTech, Available from:
http://www.intechopen.com/books/advances_in_robotics_automation_and_control/modeling_virtual_reality_web_application

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821