NTNU
Innovation and Creativity

Combining graphics and video using graphics cards

Marcus Andre Vangli

Master of Science in Communication Technology
Submission date: August 2006
Supervisor: Leif Arne Rønningen, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Combinations of live video and synthesized graphics have been used in film production for a long time. However, for movies streamed to users over networks, the complete movie is normally stored on a single server. In the future, the various tracks of a movie are expected to be distributed, and the scene composed by the end-user equipment at reception time. In many cases (e.g., games) background objects can be synthesized by the user equipment, based on transferred commands, while important objects like faces have to be shot with high resolution and transferred.

In this project the following shall be carried out:

· Give an introduction to graphics and video presentation

· Propose and specify a distributed game using graphics and video

· Design parts of the game

· Implement, test and demonstrate parts of the game on a single PC with a graphics card (e.g., a NVIDIA GeForce card and NVSG development kit)

Faglærer og veileder: Leif Arne Rønningen

Assignment given: 17. January 2006
Supervisor: Leif Arne Rønningen, ITEM

PREFACE

This thesis was done in the period January to July 2006 at Department of Telematics, at the Norwegian University of Science and Technology

I would like to thank my advisor Leif Arne Rønningen for his valuable advices and comments during the work on both report and implementation. I would also like to thank Henrik Davidsen for help reading through the thesis.

Marcus Vangli, Trondheim 3 July 2006

ABSTRACT

This report contains an introduction to graphics and video technology. Furthermore the game Avatars-Online, which is a massive multiplayer online game is presented. Avatars-Online introduces a new concept of handling player-to-player interaction, which involves 3D-sound.

The report contains the answers to the tasks presented in the assignment and which was carried out successfully. Furthermore the work of this report has lead to two interesting ideas that should be further explored:

- Interactive meeting rooms using graphics and 3D sound
- Rapid construction of high quality 3D models

TABLE OF CONTENT

Figure List

Image List

1 Introduction

Interactive Entertainment
"If you're over 35, chances are you view video games as, at best, an occasional distraction... If you're under 35, games are a major entertainment and a part of life. In that sense, they are similar to what rock 'n' roll meant to boomers."
– *USA Today,* by Kevin Maney, November 17, 2004

The computer game industry has in the last decade become one of the biggest markets in the world and the industry sold software for over 7 billion US dollars in the USA 2004 alone and is still growing. This result is very close to the movie industry, which hit a result close to 9 billion US dollars in 2004. Since the market for interactive entertainment is of such a quantity and is still growing fast. In 2005 the gaming industry marked reached an all time high worldwide turnover reaching 32,6 billion US dollars. According to ABI Research an international economical analyze company the interactive gaming industry marked will be around 65,9 billion US dollars in 2011. [1]

The gaming industry is a major force for pushing the technology in graphic development. Most games developed are very similar to the games developed in the early eighties; the major difference is the improved graphics and playability. As a consequence of increased bandwidth, higher penetration of ADSL subscribers and increased data processing power computer games are no longer only played locally on PC, Mac or a game-consol, but instead players play against each other on the Internet. There are several popular massive-online games available, where the most popular is World of War Craft. It currently has more than 6 millions active players [2], where each player pays a monthly fee between 15-13$ [3], adding up to a total of (minimum) 936 million dollar a year, and the number of active players are still growing.

Arts and Entertainment (A&E) is the divided into two segments the movie industry (including both motion pictures and television shows) and the interactive entertainment industry. On the whole A&E is the #1 U.S. export generating more international revenue than the chemical/pharmaceutical, auto, aircraft or agriculture industries. A&E new job creation rate is 300% of the rest of the U.S. economy. Statistics show that by 2008, the U.S. Arts and Entertainment Industry will increase 5.4% (from $523 billion to $680 billion). This field alone was responsible for 5.2% of the GDP of the U.S. in 2002. [4]

Keeping the numbers presented above in mind, it is likely to draw the conclusion that new development in this field will evolve at an even greater pace than it has currently undergone the last decades.

1.1 Conventions used in this text

Images and figures are arranged into chapters for every sub chapter, the number of each figure will start with one and be incremented for each new figure or image. This results in many figures and images with the same names, to avoid ambiguity when referring to figures or images the following convention will be used.

- *Italics* - Variables, arguments, parameter names, spatial dimensions, matrix components, references to other chapters and the first occurrence of key terms
- **Bold** - Command and routine names and matrices.
- ***Bold/italics*** – Is used to highlight important statements that are considered essential when evaluating the text.

1.2 The report

The report is formed by the combination of the tasks listed below, where all tasks is considered equally important. In Chapter eight and nine, ideas and opinions gained from the work related to writing of this report is discussed and evaluated.

1. ***Give an introduction to graphics and video presentation***

2. ***Propose and specify a distributed game using graphics and video***

3. ***Design parts of the game***

4. ***Implement, test and demonstrate parts of the game on a single PC with a graphics card (e.g., a NVIDIA GeForce card and NVSG development kit)***

The structure of the report follows logically and successive the parts of the problem described above. However the implementation is done using an OpenGL interface instead of a NVSG development kit, as suggested in the problem description. The difference between the two approaches is minimal since both utilize the power embedded in the GPU.

1.3 Method

The following methods were used in this report:

Literature study
The author of this report had little or no prior experience in either the field of computer graphics or the field of video technology. Therefore literature study was regarded to be the most time consuming and important part in the process of writing this report. Important sources where considered; textbooks about computer graphics and video, similar reports, Internet and other works related to this subject.

Design and implementation
The author has some years of training and experience in the field of software design and programming in general, and is convinced that generic software design principles like UML (Unified Modeling Language) could be used in the design phase of the game, while more specialized techniques are needed to fulfill an adequate implementation.
The author had little to none prior experience in the particular field of video and computer graphics. To overcome this lack of experience some time was needed to become familiar with tools and the necessary APIs (Application Programmer Interface) that where to be used in the implementation part of the report. In order to become familiar with the APIs needed, tutorials and program examples where downloaded and used considerably.

---

2 LIMITATIONS

The scope of this report is according to the tasks presented in **1.2** is very broad; therefore the author has attempted to reduce the scope slightly. However the broadness of this report is also

one of its strengths, it allows the author to explore alternative paths that might else wise not have been explored due to strict limitations. With this report the author has attempted to, stay truth to the tasks presented in **1.2** and limit the scope of chapter 4 to parts that can be related to part **A**, **B** and to some extent **E** in Figure 1.

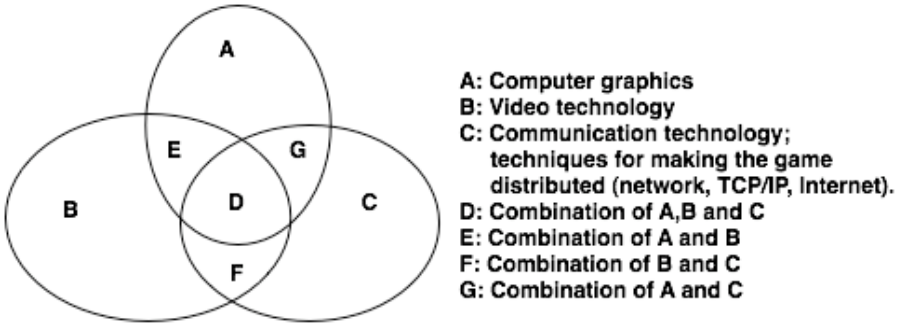


Figure 1

In Chapter 8 focus will be shifted towards exploring ideas, which can be related to part **E**, **D** and **F** in Figure 1. The idea is to build upon ideas and concepts explored in chapter 4.

To further limit the scope of this report, the different tasks presented in chapter **1.2** are considered equally important, but they imply a great variety in workload. Design is more time consuming than specification, as implementation and testing is more time consuming than the design phase. For that reason equal ness is measured in time devoted to each subject, the result is illustrated in *Figure 2* below.

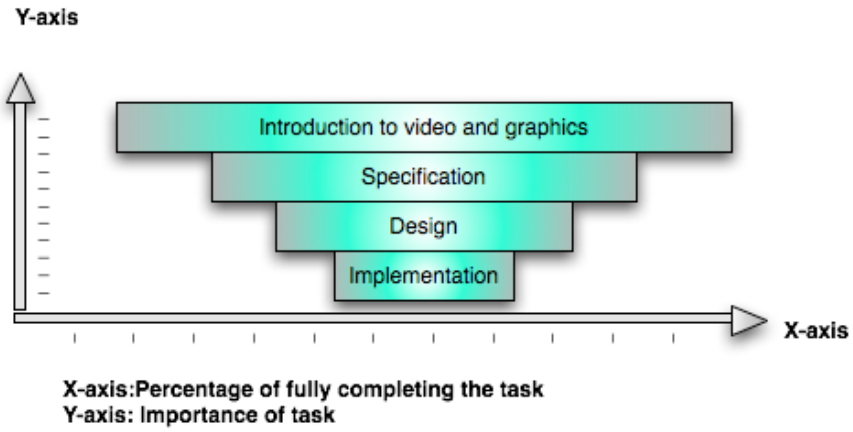


Figure 2

3 MOTIVATION

Convergence between the movie industry and the interactive entertainment industry might introduce new ways to enhance the manufacturing and development process of both industries. Several different techniques to blend live footage with computer graphics already exist, but it is very likely that new techniques will emerge.

The project itself was not intended to produce or result in a playable game, but to increase the author's knowledge of graphics and video, and to understand techniques widely used in computer graphics today (from a high abstraction level to the implementation itself). Thereby also gaining understanding of important requirements that particularly apply to development within the interactive entertainment business.

Through increased understanding of computer graphics and video technology, the author expect to identify certain areas within the design process where new techniques can be utilized to increase productivity. And hopefully be the originator of one or more concepts, which other might find interesting or at best beneficial.

Considering that interactive entertainment and the movie industry originally where treated as two distinct subjects, the borders between the two subjects have faded considerable over time. Today the movie industry has adapted and used techniques from computer graphics world as a major tool in the production of mainstream movies (Toy-Story, Star-Wars, Titanic and more). The close ties between the interactive entertainment industry and the movie industry is obvious, when considering that several computer games are based on mainstream movies, and the other way around. In the list below some of the goals the author wishes to achieve through carrying out this assignment is presented.

1. ***Increase the authors knowledge of graphics and video***
2. ***Understand techniques widely used in computer graphics***
3. ***Identify other scientific areas, which will have an impact on the design of interactive entertainment.***
4. ***Explore new concept and ideas involving graphics and video***
5. ***Identify areas in the design process where the concepts and ideas from 3 can be utilized in order to increase productivity.***

The success of massive multiplayer online games indicates that human interaction will be an important ingredient in the creation of the next generation of computer games. I believe that the next step in the evolution of interactive entertainment is the incorporation of different but relatively new technologies like, video-conferencing and multiple call sessions, and that they will serve as an integrated part of these games.

4 THEORY

4.1 COMPUTER GRAPHICS

Computer graphics have evolved considerable, since its early days when computer games where designed by "teenagers in their parents garages". In Image 1 and Image2 below a screenshot from Packman (the old legends in computer games) can be compared to a screenshot from the latest version of DOOM. The difference in level of detail in the two screenshots are striking, and provides a good indication of the high degree of transformation this branch of technology has been subjected to.

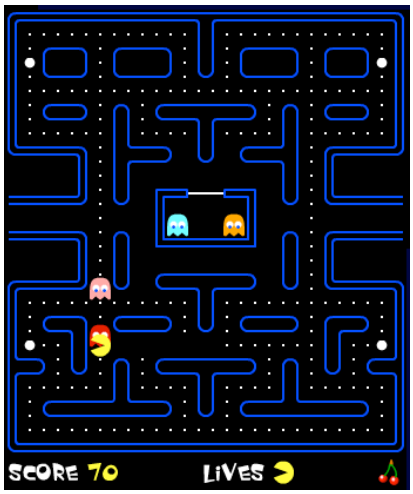


Image 2 Screenshot from the game Packman [5]

Image 1 Screenshot from the game DOOM III [6]

Computer generated graphics has also been extensively introduced as a vital part of many mainstream movies, examples are; Jurassic Park, Toy-story and more. In the computer world graphics is associated with techniques to create or manipulate images. There are several techniques and concepts that over the years have become standard terminology in the culture of computer graphics. I have therefore in this chapter defined some of the basic terminology and described some of the most commonly used techniques within computer graphics. [5] [6]

4.2 GRAPHICS TERMINOLOGY

4.2.1Alpha-Blending

Alpha: The alpha parameter describes the transparency of a graphical object. [30]

Blending is the technique of combining graphical objects by adding them on a pixel-by-pixel basis []. This technique is very useful when working with objects that are transparent; like a glass of water. The equation used in alpha blending is illustrated in Figure 3 from [7].

$$[r,g,b]_{blended} = \alpha[r,g,b]_{foreground} + (1-\alpha)[r,g,b]_{background}$$

Figure 3, where [r,g,b] is the red, green, blue color channels and alpha is the weighting factor.

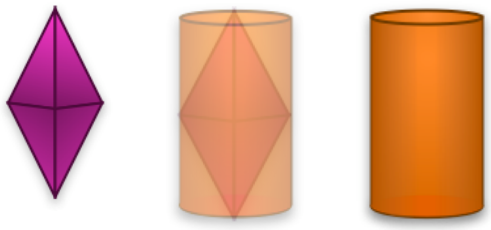


Figure 4

In Figure 4 above the concept of alpha blending is illustrated, the prism is blended with the cylinder by simply adjusting the alpha parameter of the two objects.

4.2.2 Bilinear Filtering

Bilinear filtering is a technique used when anti-aliasing a texture map. When texture is mapped onto a polygon, which is moving or rotating slowly, texels (is analogue to pixels but refers to a "pixel" in the texture map) tend to jump randomly from one pixel to another. This random pixel jumping is very noticeable and causes the texture image to jump and shears along pixel boundaries. Bilinear filtering eliminates this problem by taking a weighted average of four adjacent texels to create a single telex (a frame consisting of more than one texel). Based on text from [11]

4.2.3 Chroma-Keying

Chroma-Keying or texture transparency is a technique where a key color or certain color range in the texture map is made transparent. The result causes objects or images "behind the removed color/colors" to be revealed. Chroma-Keying is used to include objects that often are to complex to be easily modeled with polygons. Examples are the productions of weather forecasts, where a "blue-screen" is used to separate the weather forecaster from the rest of the scene. In this process the key color is the same as the "blue-screen", allowing the weather forecaster to be digitally sliced out the scene and appended upon the weather map.
Based on text from [11] and [8]

4.2.4 Depth Cueing

Depth cueing blends image colors with the background color (typically black) as distance from the camera viewpoint increases. This improves the perception of depth and shape for 3D objects tremendously, especially for complex curved surfaces or wireframes [9]. The depth cuing technique basically lowers the intensity of the pixels in an object, as the associated objects move away from the viewpoint. In practice depth cueing is commonly implemented by using the fog technique, described in chapter **4.2.7 Fog** or a 1D texture map.

The advantage of using a 1D texture map is that the map can be used to encode an arbitrary function of distance and it is also straightforward to implement. The process is illustrated in Figure 5. Based on text from [10].

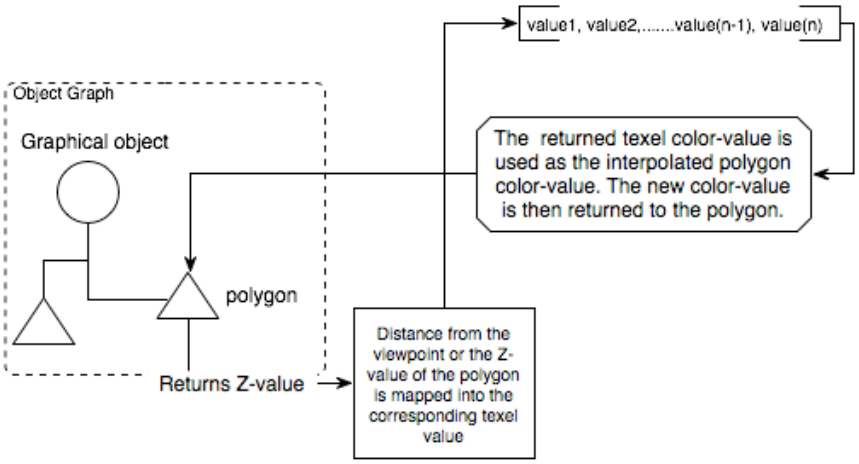


Figure 5

4.2.5 Double Buffering

Computer monitors and displays constantly redraws the visible image on the screen somewhere around 70 times a second. High quality graphical objects combined with complex movement in 3D space, could cause the display device to present the content of the buffer before the graphical operation is finished. Thus causing ugly artifacts such as flickering, tearing and shearing to appear on the display device. In order to hinder these artifacts from happening, a method called double buffering was invented. Basically double buffering partition the video-RAM (Random Access Memory) in two, where one is used for rendering while the other is being displayed. At the same time as one of the buffers is operated on the by the rendering engine the other buffer is being displayed. When a new frame is rendered the two buffers are switched. Thus hindering the display device from displaying frames before the graphic operation is complete.

Double buffering is a software operation can also be implemented in hardware, but is then referred to as page flipping. There also exist a variation of double buffering called triple buffering[1], which show no (or less) flickering, tearing and shearing.

4.2.5 Is based on text from [11] and [12].

4.2.6 Fog

Fog is a technique where the colors in graphical objects are blended with a given color, the fog color in order to give the viewer an illusion of depth. The distance at which the blending starts and stops can be specified in 3D space, and the way the fog color is increased throughout the region between these two points along the Z-axis in 3D space. Thus objects close to the viewpoint is seen with no change. As the distance from the viewpoint increases the color of the object fades toward the fog color, at the far end (along the z-axis) objects are seen with the full effect of the fog density. The fog effect is achieved by interpolating the color of the objects with the fog color. By using the density parameter to control the quantity of color merging, a method of depth cueing is provided. The fog technique or depth cueing by fading to black is illustrated in Image 3 and Image 4, where Image 3 is the original image, while Image 4 is the result of blending objects with a fixed color (the fog color, which in this case is black) as its pixels become farther away from the viewpoint. Based on text from [13] and [11]

---

[1] Se http://en.wikipedia.org/wiki/Triple_buffering for more information on triple buffering.

Image 3

Image 4

4.2.7 Gamma

All CRT and phosphor-based displays have a nonlinear relationship between signal voltage and light intensity. This causes a small change in the in the signal input at low voltage to produce a change in the output display brightness level. However the same small magnitude change in voltage at high voltage will not produce the same magnitude of change in the brightness output. Fortunately there is a transfer function, which counterweigh what you actually measured and what you should have fairly well. Using the following power function does this approximation:

- Output = Input^Gamma

Gamma can therefore be viewed as the difference between what you should have and what you actually measured. Based on [11] and [14].

4.2.8 Gamma Correction

It would be convenient for graphics programmers if all of the components of an imaging system were linear [14]. The voltage coming from an electronic camera would be directly proportional to the intensity (power) of light in the scene, the light emitted by a CRT would be directly proportional to its input voltage, and so on [14]. However as explained in 4.2.7 Gamma, most real-world devices do not behave as the linear curve in Figure 6, in practice most display devices have nonlinear signal-to-light-intensity or intensity-to-signal characteristics as illustrated in Figure 7 **[15]** Gamma correction can be thought of as the process of compensating for this nonlinearity, in order to achieve correct reproduction of intensity.

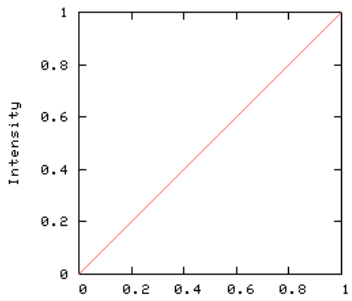


Figure 6 [15]

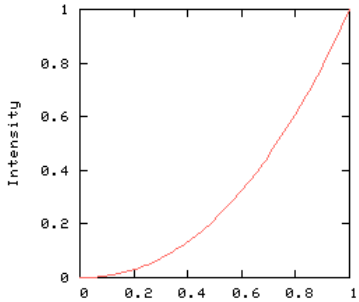


Figure 7 [15]

Figure 7 **[15]** and Figure 6 **[15]** describe the output intensity (y-axis) corresponding to the value of the input signal (RGB data (x-axis)). Ideally we would like the display device to reproduce the signal as described in Figure 6 **[15]**, but since most display devices behave according to Figure 7 **[15]**, the input-output deviance must be compensated. By processing the linear RGB input signal using the equation described in 4.2.7 Gamma the deviance between the input and output signal is compensated, this process is commonly referred to as gamma correction.
Based on [15], [11] and [16]

4.2.9 Lighting

There are many techniques for creating realistic graphical effects to simulate a real-life 3-D object on a 2-D display [11]. One technique is lighting. Lighting creates a real-world environment by means of rendering the different grades of darkness and brightness of an object's appearance to make the object look solid [11]. In practice this means that pixels that in 3D space which are close to the viewpoint are brighter than pixels situated at the far end of the viewpoint in 3D space. Accurate use of lightning creates an illusion of depth in the image.

4.2.10 Occlusion

Occlusion is the effect seen when an object is blocking the view to another object in 3D space. The effect is illustrated in Figure 8 where the cylinder is blocking the view to the cube. Based on [11]

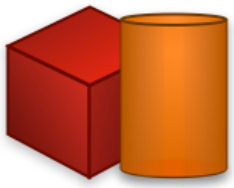


Figure 8

4.2.11 Palletized Texture

Palletized Texture means compressed texture formats, such as 1-, 2-, 4-, and 8-bit instead of 24-bit; this allows more textures to be stored in less memory [11].

4.2.12 Projection

The process of reducing three dimensions to two dimensions for display is called Projection. It is the mapping of the visible part of a three dimensional object onto a two dimension screen [11].
In Figure 9 We observe that objects in the virtual 3D space in are reduced in size when projected on the 2D display device. When comparing the amount of screen space devoted to the two objects (the sphere and the cube), we observe that the cube (A-B) occupies more space on the 2D display device than the sphere (B-C). The space occupied on 2D display of the object derives directly from its z-position in 3D space. In fact the projected size of an object

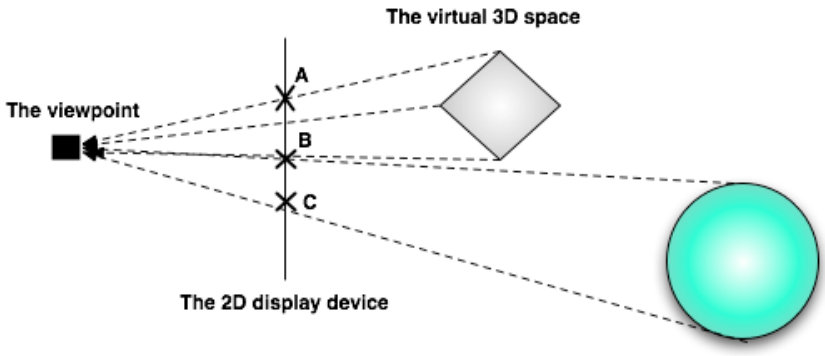


Figure 9

is proportional to its distance from the viewpoint measured along the z-axis in 3D space. Objects faraway appear smaller than objects closer to the viewpoint analogue to the sphere and cube in Figure 9, thus creating an illusion of depth.

4.2.13 Perspective Correction

A particular way to do texture mapping; it is extremely important for creating a realistic image. It takes into account the effect of the Z value in a scene while mapping texels onto the surface of polygons [11] As a 3D object moves away from the viewer, the length and height of the object become compressed, making it appear shorter as illustrated in Image 5 [19], where the far end of the picture is smaller than the part closest to the viewpoint. In the picture on the far right side of Image 5 [19] the viewpoint has been shifted from the left corner to directly in front of the picture.

Image 5 [19]

3D to 2D Transformation

Figure 10 only shows a line in 2D space being projected onto a 1D image plane, but the same argument can be applied to a 3D linear geometric primitive projected onto a 2D image plane [18]. If we think of the line AB in Figure 10 [] as the z-property of an object in 3Dspace, which must be displayed on a 2D image plane, we observe that AC and CA will appear to be of equal size. Since depth does not appear naturally in a 2D display device, measures have to be taken in order to preserve the illusion of depth. First of all the intensity of the pixels in an object is lowered, as the associated part of the object move away from the viewpoint. In Figure 10 [] we that the intensity of vertex A is 0.0 (max), while the intensity of vertex B is 1.0 (black as in far away), respectively the intensity in *a* and *b* is equal. If we interpolate the value in *a* and *b* we find that the intensity *c* should be 0.5. However if we follow the dotted line from the viewpoint in Figure 10 [] through *c* onto a point C on the line AB, we see that C is not the midpoint on AB. Since the intensity value varies linearly in 3D space, we find that the intensity of *c* should not be 0.5 [2].

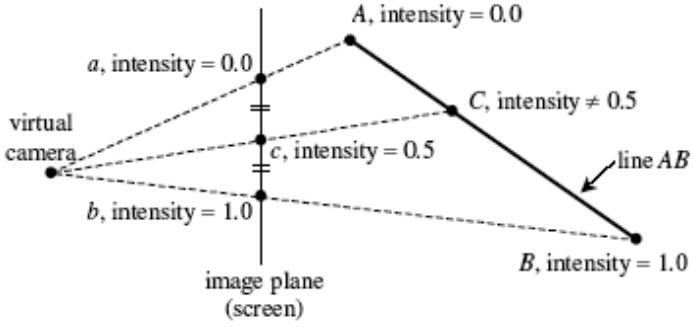


Figure 10 [17]

Assuming that the line AB in F9 is a part of a cube in 3D space, fitted with texture on the side facing the screen. To achieve the correct perspective view, the texture must be reduced in size so that it fit in square projected on to the image plane. Fitting of the texture to match the new

[2] For further information on perspective correction see [18].

projection on 2D screen, is done by appending a grid “on the” texture map and shrink the frame to the projected size and then calculate the new intensity of the pixels. The perspective correction process corrects the shape and modifies the pixels in each square of the grid to fit the new perspective. Logically perspective correction with a dense grid gives a better result than perspective correction with a spacious grid.
Based on [18], [19] and [11]

Without perspective correction, objects will appear to shift and 'tear' in an unrealistic way. True perspective correction is that the rate of change per pixel of texture is proportional to the depth. Since it requires a division per pixel, perspective correction is very computing intensive [11].
Based on [11] and [20].

4.2.14 Rendering

Rendering is the process of generating an image or frame from a model. The model is a description of 3D objects, and is defined in a strict language. In the 'graphics pipeline' the rendering process is the last major step, giving the final appearance to the models and animation
The rendering process involves mathematical models and formulas to add shading, color, and lighting information to the model. Finally the model is converted into pixels that can be displayed on a screen. Based on text from [21]

In the case of highly detailed 3D graphics, the rendering process can be very time consuming. There are basically two types of rendering processes:

- Pre-rendering, is often computationally intensive since pre-rendering in most cases involves complex models with rigorous resolution requirements. Typically pre-rendering is used for movie creation.

- Real-time rendering, the complexity of the models and the resolution requirements cannot be higher than that the graphics card/ 3D hardware accelerators can render the model in real-time (the frame rate of the display device). Real-time rendering allows a high degree of interactivity since user-input can have an instant effect on the scene displayed. Therefore real-time rendering is commonly used in 3D video games.

4.2.15 Z-buffer

The z-buffer is a part of memory that contains the distance from the viewpoint for each pixel. If the display device has a resolution of 640x480 the z-buffer is a 640x480 array. Only pixels from objects that can be viewed from the viewpoint are stored in the z-buffer. In Figure 11 we observe that the part A-B on the cube are projected onto the same pixels in the 2D display device as parts C-D of the sphere. Some of the projected pixels from the cube and the sphere will have coherent x and y coordinates. However the view to part C-D of

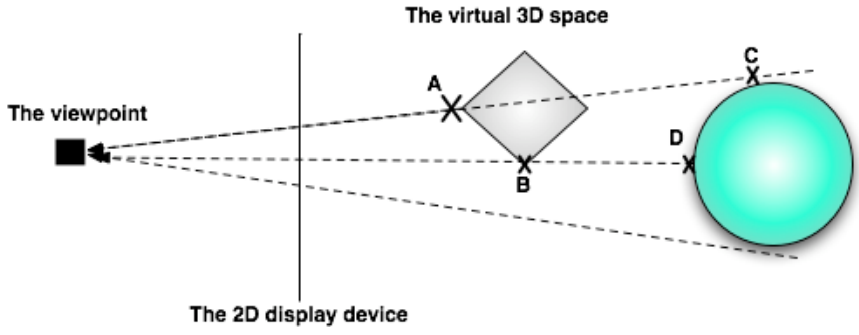


Figure 11

the sphere is blocked by the part A-B of the cube. Thus only the projection of the visible part of the sphere is stored in the z-buffer. The z-buffer will then contain the z-value of all the 640x480 pixels currently displayed on screen. Based on text from [22]

4.2.16 Z-buffering

Z-buffering is the process of creating and updating the z-buffer. Setting all z values to "infinity" clears the Z-buffer, and a new frame can be loaded into the buffer. When rendering objects, the engine assigns a z-value to each pixel: the closer the pixel to the viewer, the smaller the z-value. When a new pixel is rendered, its depth (z-value) is compared with the stored depth in the Z-buffer [11]. If the z-value of a new pixel is lower than the pixel currently stored in the Z-buffer the new pixel is written into the buffer, substituting the old pixel. Thus Z-buffering is a process that continuously removes hidden surfaces from a newly rendered frame, by using the depth value stored in the Z-buffer. Based on text from [11] and [23].

4.2.17 Z-sorting

3D objects are commonly modeled using polygons (see 4.2.20 Tessellated Models for further details). Z-sorting take advantage of the polygons z-parameter, and sorts the polygons in back-to-front (polygons are sorted according to their z-parameter, so that polygons closest to the viewpoint are moved to the very front of the polygon array and polygons with furthest away are moved to the front) order prior to rendering. The polygons are rendered by iterating the sorted array of polygons, so that the surfaces (polygons) closest to the viewpoint are rendered last. The result of rendering the array is always correct unless objects coincide with each other.
The advantage is not requiring memory for storing depth values [24]. The disadvantage is the cost in more CPU cycles and limitations when objects penetrate each other [24].
Based on text from [24].

4.2.18 Animation

Animation is a technique that achieves an illusion of motion through the use of sequences of images. The technique combines both modeling and rendering, with the handling of time. In its most basic form animation is only a sequence of images is shown continuously that together gives the viewer a feeling of motion. Based on text from [30].

4.2.19 3D-scanning

3D scanning uses range-finding technology to create measured 3D models. Such models are very useful when creating 3D models, and can then be utilized in applications that need rich visual imagery with close resemblance to real-life [3]. Today 3D scanning can be combined with regular RGB-cameras enabling the user to capture both color and depth of a scene (see 4.6.5 Video Cameras for further reading). Based on text from [24] and [25]

[3] 3D scanning equipment has been extensively used in manufacturing industry. 3D graphics is incorporated in all product development process phases - from conceptual design, detailed part and assembly design to shop floor manufacturing. For further reading [25] is recommended.

4.2.20 Tessellated Models

[Tessellated models: 3D-models where the planar surface is composed of repeated use of a single shape without gaps or overlapping.]

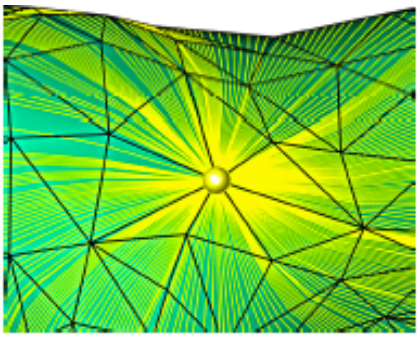


Image_6 [image of a typical TIN]

Tessellation is the process of subdividing a surface into smaller shapes. To describe object surface patterns, tessellation breaks down the surface of an object into manageable polygons. Triangles or quadrilaterals are two usually used polygons in drawing graphical objects because computer hardware can easy manipulate and calculate these two simple polygons. An object divided into quads and subdivided into triangles for convenient calculation.

Most real-worlds tessellated models are composed of complexes of triangles with shared vertices. These are generally known as triangular meshes or triangular irregular networks (TINs). Most graphical programs need to be able to handle these models effectively and also merge these models with texture.

TINs are commonly used to store and represent complex three-dimensional models. There are basically two approaches used for storing a TIN:

1) Treat each triangle as independent entities thus storing each triangle separately as a vector containing the vertices in the triangle.
2) Try to somehow share the common vertices. So instead of treating each triangle as independent entity we use two classes; class mesh and class mesh-triangle. The mesh class contains information of the material and an array called *vertices*, which contains the three vertices in the triangle. The class mesh-triangle contains indices [i_0, i_1, i_2] into the vertex array *vertices*, and a pointer *meshptr*, which points to the mesh-triangle parent.

Using the strategy in 1) results in storing a random vertex $\mathbf{R}_1$ on average six times. Since a large mesh typically has each vertex stored by about six triangles, although there can be any number of extreme cases. The strategy in 2) is bit more complicated to incorporate. But considering that the storage is reduced by a factor of two makes the complication well worth the effort. Especially when a large number of properties are added to the vertices.
Based on text from [30].

4.2.21 Graphics Pipeline

Graphics pipelining allows several sets of operations to be conducted successive, meaning that next operation can start before the foremost operation has finished. The pipelining technique has been used extensively in several areas of computer technology.

The graphics pipeline is composed of the following steps that are:

1. **Transforming**: The first stage is to import and fit each object to the "worlds". In practice this means scaling the imported 3D models so that the size of the models logically fit together. This stage is necessary because the original design of an airplane might be no larger than the design of a bike imported into the same 3D world. Thus the two imported

models must be resized to fit the 3D world, so that the airplane is larger than the bike. This process involves both shrinking and enlargement of the imported models.

2. **Transferring:** Once the all models has been fitted to logically suit the 3D world, the objects or models must be transferred to fit the perspective that the viewer will be looking from (the viewpoint). Thus all objects must be based on coordinates from the viewpoint. This stage involves rotating objects to fit the viewpoint, and resizing the objects. So that objects closer to the viewpoint appear larger than objects further away.
3. **Occlusion culling:** During this step hidden polygons that are hidden (blocked by other objects) from the viewpoint are removed. Techniques used in this process are Z-sorting (see 4.2.17 Z-sorting) and Z-buffering (see 4.2.16 Z-buffering).
4. **Lighting/illumination**: The illumination stage is fairly complicated. Not only does all light sources have to be accounted for, but lighting also affect the appearance of the models/objects by changing their color. In normal 3D, perfect lighting is expected. Achieving perfect lighting is computationally demanding and in most cases not possible to combine with real-time rendering[4]. Since most games have strict real-time constraints a compromise between speed and eye candy is necessary. Usually the compromises involve fewer light sources, a minimum of reflection and often only lighting from discrete sources are calculated.
5. **View-Frustum:** Summarizing the steps completed so far on the pipeline, objects and polygons that cannot be seen from the viewpoint have been "removed". Thus with basis in the viewpoint, we now have a lighted 3D scene, which gives our models their correct color. In this step a "view frustum[5]" is created since all the information in the 3D scene cannot be displayed on the screen. All models outside the view frustum are discarded, the problem occurs when a parts of a polygon is in the view frustum and the other part is outside. These polygons need to broken into smaller polygons, thus adding more vertices and polygons for rasterization later in the pipeline. Only the polygons needed to display the scene are saved, the rest are deleted at this stage.

Conversion: Basically step six makes 3D into 2D. Changing 3D into 2D involves a lot of work; all information is held in texels and vertices, which must be converted to pixels for the display device. This part involves techniques explained in; 4.2.12 Projection, 4.2.13 Perspective Correction (under 3D to 2D Transformation). Finally, texture (

6. 4.5.1 2D Texture Mapping), fog (4.2.6 Fog), alpha blending (4.2.1Alpha-Blending), anti-aliasing () depth (4.2.4 Depth Cueing), and all the other functions that make your graphics look great.

Based on text from [26], [27]

4.3 Image Capture and Storage

There are several different formats used to digitally store "real" images captured using digital cameras or flatbed scanners. Most scanners and cameras use a light-sensitive chip to record light. The dominant technologies are CCD and CMOS arrays [30], three different filters are used to digitally record the different RGB composition of each pixel. Camera manufactures have different proprietary algorithms for creating a single RGB image captured using the Bayer

[4] See 4.2.14 Rendering for more information on real-time rendering and pre-rendering.

[5] Frustum: Rectangular cone formed by lines from the camera to each corner of the projection plane [27].

mosaic that are somewhat more complicated than the obvious strategy of linear interpolation. The Bayer mosaic is displayed in *Figure 12*.

Bayer mosaic

G	B	G	B	G	B	G
R	G	R	G	R	G	R
G	B	G	B	G	B	G
R	G	R	G	R	G	R
G	B	G	B	G	B	G
R	G	R	G	R	G	R
G	B	G	B	G	B	G

Figure 12

To reduce the storage requirement, most image formats allow for some kind of compression. Viewed from a high level compression algorithms can be divided into two main categories *lossless* and *lossy*. If an image is compressed using a *lossless* algorithm all information lost in the process can be recovered. Using a *lossy* Compression algorithm the discarded information can be recovered.

Gif
This format indexes only 256 possible colors. The quality of the image depends on how carefully 256 colors have been chosen. This format typically works well for natural diagrams. Gif is a *lossy* format.

Jpeg
Jpeg is a *lossy* format that works well for natural images.

Tiff
This is a *lossless* format and usually a compressed 24-bit per pixel, but other options do exist.

PPM
PPM is a *lossless* format and usually a 24-bit per pixel, uncompressed format although other options do exist.

Based on text from [30]

4.4 INTERPOLATION

4.4.1 Linear Interpolation

Interpolation is one of the most commonly used mathematical techniques used in computer graphics. The technique is extensively used in the following processes:

- To form line segments in 2D and 3D where two points *a* and *b* are associated with a parameter *t* to form the line **p**=(1-*t*)*a*±*tb* where *tE[0,]*. Resulting in a straight line between the two points *a* and *b*.
- Another common linear interpolation is among the set of positions on one of the axis *x, y* or *z*, where only a limited set of points exist, the limited set of points can be connected by a straight line, using linear interpolation. It is natural to use parametric line equations for these segments. The parameter *t* is just the fractional distance between x_i and x_{i+1}:

$$p(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0).$$

Linear interpolation is also used to smooth color transitions, letting the different colors in the color palette have values that are related logically to each other.

4.4.2 Techniques Involving Interpolation

Anti-aliasing
Anti-aliasing interpolates pixels in the edge of an object with the background, this technique makes the edges appear to have better resolution. [24]

Texture Anti-aliasing
Texture Anti-aliasing is an interpolation technique used to remove texture distortion, stair-casing or jagged edges, at the edges of an object [24].

Texture Filtering
Removing the undesirable distortion of a raster image, also called aliasing artifacts, such as sparkles and blockiness, through interpolation of stored texture images [**Error! Bookmark not defined.**].

4.5 TEXTURE MAPPING

The technique of mapping a pixel-based image onto a three-dimensional surface is a
Mapping techniques add realism and interest to computer graphics images. Texture mapping applies a pattern of color to an object. Bump mapping alters the surface of an object so that it appears rough, dented or pitted. Texture mapping is also used to "carve out " three-dimensional objects with a desirable surface.
The function or image that is used is called a *texture map* and the process is called *texture mapping.* Texture mapping is classified by several different properties:

- The dimensionality of the texture function.
- The correspondence between points in the surface and points in the texture function.
- Whether the texture is primarily procedural or primarily a table look-up.

The common technique to handle variations of reflectance is to store the reflectance is to store the reflectance as a function or a pixel-based image and map it onto a surface.
Based on text from [28] and

4.5.1 2D Texture Mapping

In two-dimensional texture mapping, we have to decide how to paste the image on to an object. The image is often either scanned photographs or images created in a paint or drawing package. For each pixel in an object, we need to know where to look in the texture map to find the right color. When using a two-dimensional texture map coordinates in this map is often referred to as *uv,* which is used to create the reflectance **R**(*u, v*).

The key is to take the (*u,v*) coordinates of the image and associate them with points in the texture map. The process is illustrated in *Figure 13,* where the technique creates the appearance of grass, sky and bricks without the cost of rendering thousands of polygons.
Based on text and images from [30]

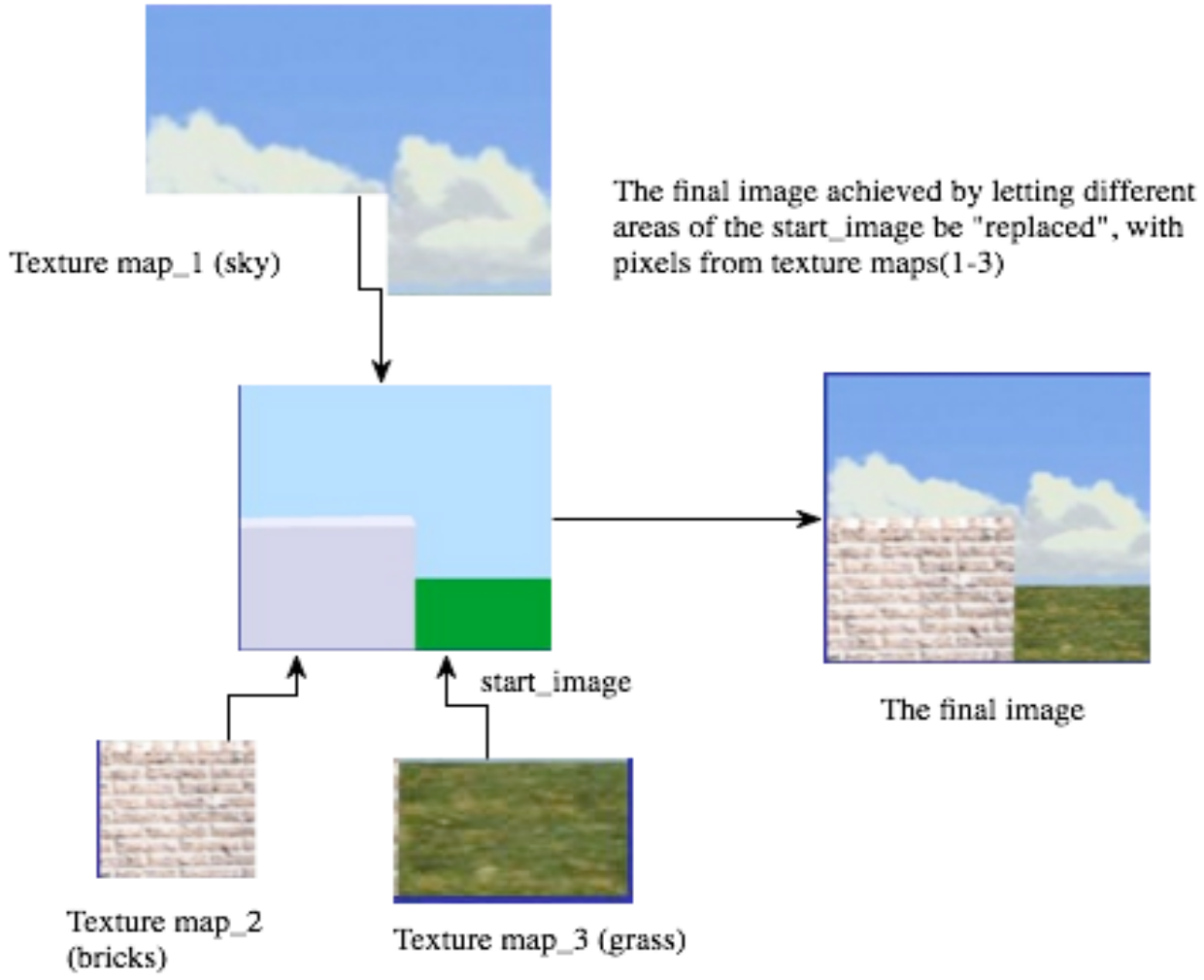


Figure 13

The technique is very intuitive when dealing only with two-dimensional objects and images. When the two-dimensional texture maps are to be used on three-dimensional objects the process becomes slightly more complicated. However the basics are very similar, but one needs to find a scheme to associate the uv-coordinates of the texture map with points on a three-dimensional surface.

4.5.2 3D Texture Mapping

For a map shape that's planar, we take an (*x,y,z*) value from the object and throw away (project) one of the components, which leaves us with a two-dimensional (planar) coordinate. We use the planar coordinate to look up the color in the texture map. In *Image 7* and
Image **8** below the result of texture fitted upon a three-dimensional object is illustrated.

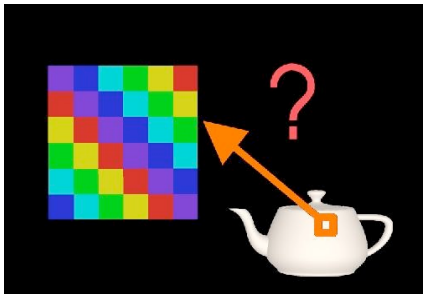


Image 8

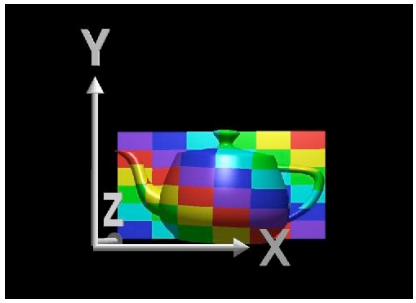


Image 7

The colored surface used as texture in
Image **8** above does not completely reveal what happens to the three dimensional object when the object is viewed from an angle were the objects z-component is visible. When viewing *Image 7* the result is very similar to what would be achieved using projector to project the colored texture map in
Image **8** onto a three-dimensional object.

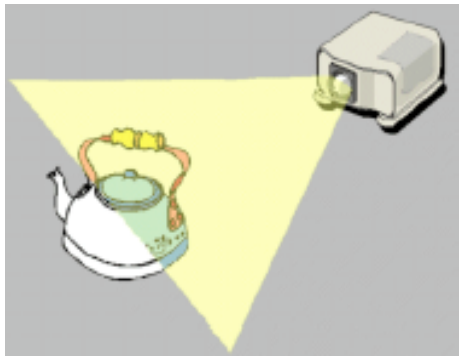


Image 9

The mathematically process used in texture mapping simulate the effect of projecting an image onto a three-dimensional object, as illustrated in *Image 9*. Therefore I have thoroughly described the mapping process in *Figure 14* below.

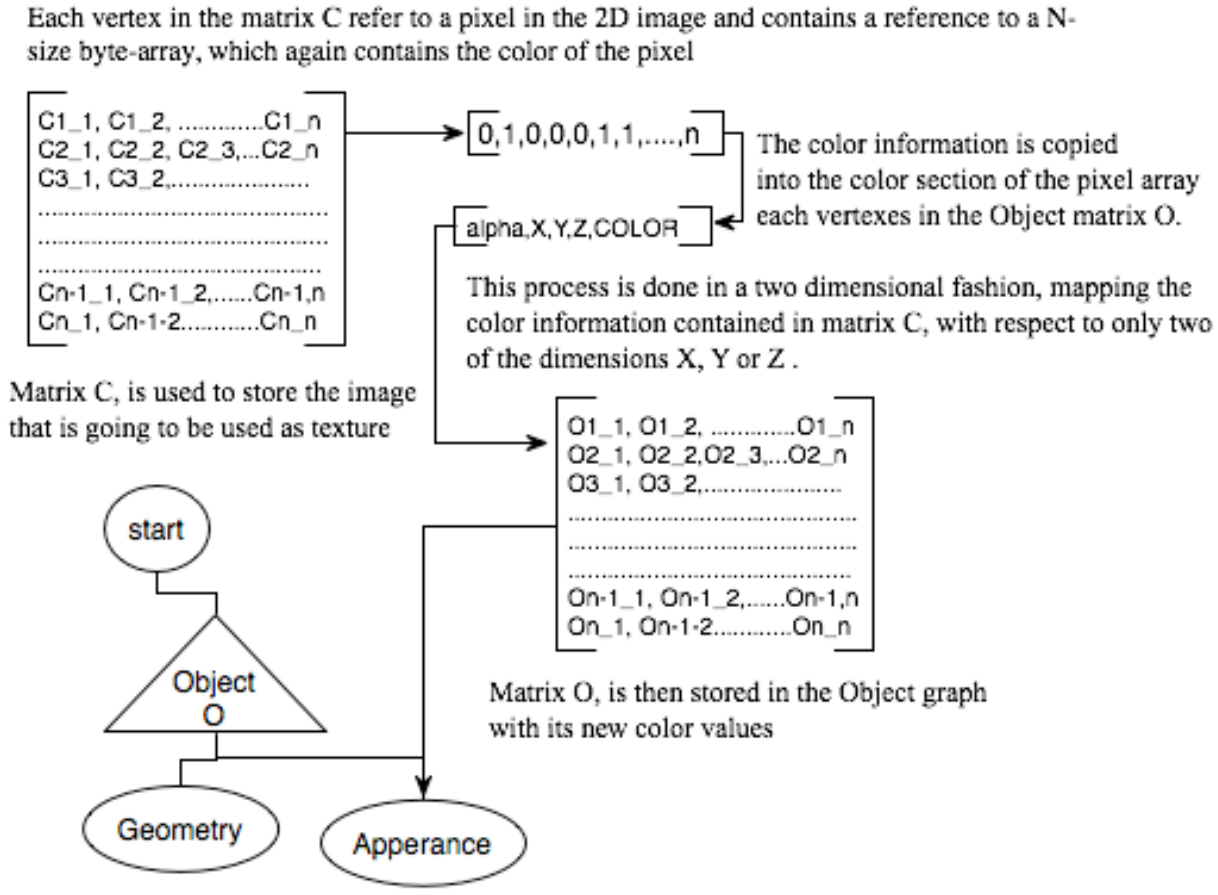


Figure 14

Viewed from another angel in *Image 10* the three-dimensional shape of the object is revealed, and one of catches with using a two-dimensional image as texture for a three-dimensional object becomes imminent.

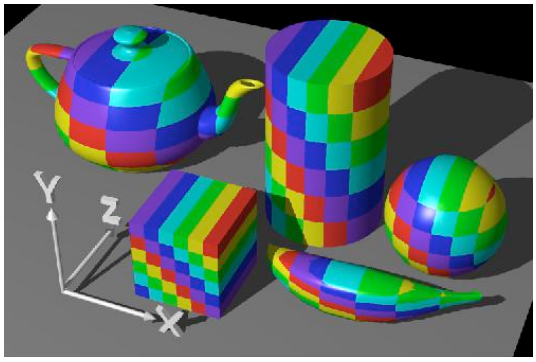


Image 10

Several textured-mapped objects that have a planar map shape. None of the objects have been rotated. In this case, the component that was thrown away was the z-coordinate.

You can determine which component was projected, by looking for color changes in coordinate directions.
When moving parallel to the x-axis or the y-axis an object's color changes.
However, movement along the z-axis does not produce a change in color. This is how you can tell that the z-component was eliminated.

The elimination of the z-component reveals one of the effects of using a two-dimensional map on three-dimensional objects. Another effect that it is not revealed due to the nature of the color-grid-map is that the two-dimensional map should be fitted specifically for the corresponding tree-dimensional object. *Image 11* is specifically fitted for a three-dimensional sphere. Note that the image has a different number of pixels horizontally and vertically so the image pixels have a non-uniform aspect ratio in (u,v) space.

Image 11

When examining the Image 11 closer one should be able to notice the excessive size of Greenland, which exactly corresponds to the shrinking that occurs when the map is applied to a sphere. In the Image 12 we can observe the effect the shrinking process. Since the image was specifically fitted for a sphere Greenland does not appear to be just the right size, also notice that areas close to equator has increased in size.

Image 12

The mapping procedure used in this image is slightly different than the one explained in Figure 14. To map a two-dimensional image onto a sphere, we first have to find a mapping scheme that will allow us to match the (u,v) space of the image onto the (u,v,w) space of the sphere. This process involves calculating the polar coordinates of the sphere and then use the mapping scheme described in *Figure 15.*
Based on text from [30] and [29].

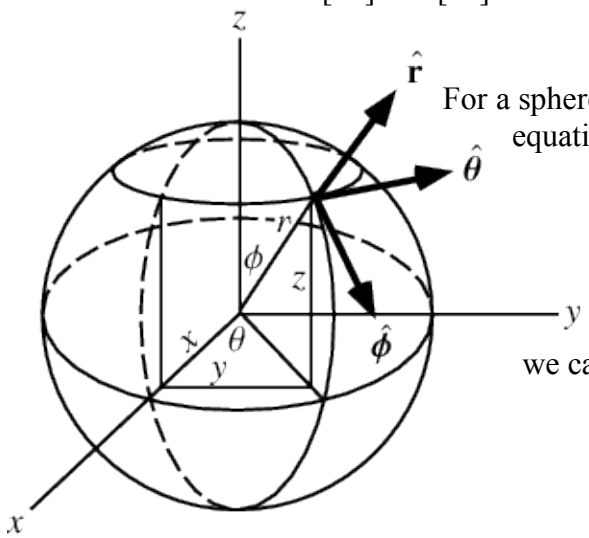


Figure 15

For a sphere with center (c_x, c_y, c_z) , the parametric equation of the sphere is :

$$x = x_c + R \cos\phi \sin\theta$$
$$y = y_c + R \sin\phi \sin\theta$$
$$z = z_c + R \cos\theta$$

we can find (θ, ϕ), where $[\theta, \phi] \in [0,\pi] \times [-\pi,\pi]$

$$\theta = \arccos\left((z-z_c)/R\right)$$
$$\phi = \arctan 2\,(y-y_c,\ x-x_c)$$

Thus the following formula can be used to find the right *uv*-coordinate for each point in the three-dimensional surface:

- $u = (\phi/2\pi)$
- $v = (\pi-\theta)/\pi$

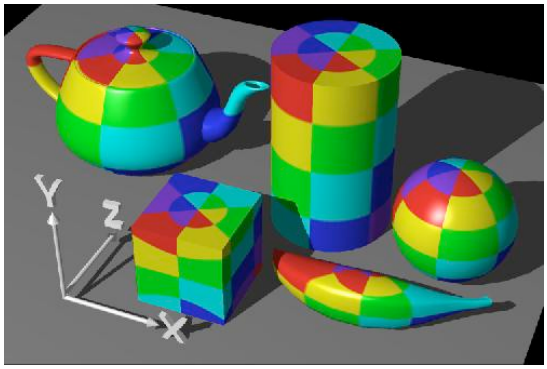


Image 13

This mapping is used in image (world). There is similar although likely more complicated way to generate coordinates for most three-dimensional shapes.
Based on text from [30]

To further illustrate this point the same two-dimensional texture map used in Image 7 has been used on the same object but this time with the use of polar coordinates, the result can be viewed in *Image 13*. There are different mapping schemes for most common three-dimensional object, like cylinders, cubes, pyramids and many others. When working with three-dimensional surfaces and "real" images captured by digital cameras the result of a texture mapping process can be difficult to predict. Using texture mapping on a wall similar to the one in *figure_1* suddenly involves several unexpected obstacles when the wall is modeled as a three-dimensional object like incorrect perspective view an other as we shall see in chapter *4.5.2 Tessellated Models*.

To use "photo-texture" on three-dimensional objects can be quite the challenge since the photos must be specifically "labeled" for the three-dimensional object, and finding the right "labeled" image (illustrated in Image 15 for your model can be quit the challenge. However "photo-texture" is very useful when one wants to se reflections of images in the three-dimensional object. The concept is illustrated in *Image_11.*
Based on text from [29] [30] [31],

Image 15

Image 14 from *[31]*

4.5.3 Displacement Mapping

Displacement mapping is a technique used to ad realism to three-dimensional objects, by actually changing their geometry. The technique uses a displacement map to exchange points on

the surface of the geometrical object with new points from the displacement map. A common simplification is that the displacement will be in the direction of the surface normal.

The normals on the base surface can be represented as ˆN(u, v). Using this representation the points on the new displaced surface P`(u, v) are defined as:

$$\mathbf{P}'(u,v) = \mathbf{P}(u,v) + d(u,v)\hat{\mathbf{N}}(u,v)$$

$$\text{where } \hat{\mathbf{N}}(u,v) = \frac{\mathbf{N}(u,v)}{|\mathbf{N}(u,v)|}$$

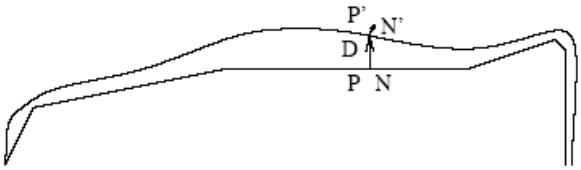


Figure 16 The points P are displaced in the direction of N with point P

Displacement mapping is straightforward to implement in a z-buffer code by storing the surface to be displaced as a fine mesh of many triangles. Each vertex in the mesh can be displaced along the normal vector direction. This result in large models, but it is quit robust.
Text based on [30] and [32]

4.5.4 Environment Maps

Often we would like to have a texture-mapped background and for objects to have specular reflection of that background. This is accomplished through the use of environment maps. An environment map can be implemented as a background function that takes in a viewing direction **b** and returns an RGB color from a texture-map.

There are many different techniques used to store an environment map, where some of the most commonly used are:

- A *cube map,* the object is encapsulated in a large cube and the surface of the object reflects texture of the cube. Cube mapped reflection is done by determining the vector that the object is being viewed at. This camera ray is reflected about the surface normal of where the camera vector intersects the object. This results in the reflected ray, which is then passed to the cube map to get the pixel, which the camera then sees as if it is on the surface of the object. This creates the effect that the object is reflective.

- A *sphere map*, which is very similar to a *cube map* basically is a spherical table indexed with spherical coordinates.

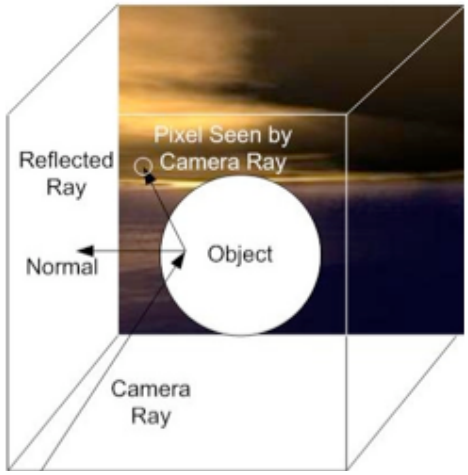


Figure 17 illustrates the use of a cube-map [33]

For more specific information on environment maps the following material is recommended:

- http://www.blender.org/modules/documentation/htmlI/x4881.html
- http://developer.nvidia.com/object/cube_map_ogl_tutorial.html

Based on text from [33] and [30]

4.6 Video

Since the 1920s when the black-and-white prototypes where introduced, television has evolved into and high quality color TV. The latest amendment to high quality color TV is HDTV (High definition TV). It is considered as the latest step in the direction of closing the gap between human perception of the real world, and the virtual reality displayed on a screen, but the hurdle of 3D TV still remains. Leading authorities like the IST (Information Society Technologies program, sponsored by the European Commission.) that 3D is the next major revolution in the history of TV. [34]

Video is basically a sequence of images or still pictures of a scene, where each images are shoot at a certain time interval. The concept is illustrated in Figure 18, where a sequence of pictures is sorted in timely order; variation in the pictures can adequately represent the motion in a scene. Based on text from [35].

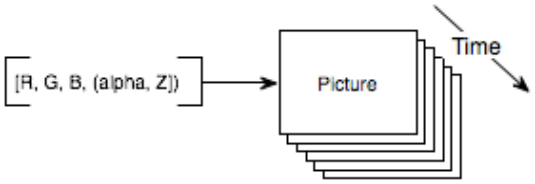


Figure 18

In this part I have tried to cover the most commonly used techniques in video technology. After reading this section of chapter 4 the reader should be able to understand the following:

- How a video signal is generated.
- How a video signal is represented for storage or transmission.
- How a video signal is converted between different digital representations.
- Understanding of MPEG-4 video compression.

4.6.1 Video Imaging

Images are captured through the use of a camera, where the optical lens is used to find the correct focus of the scene. By converting optical signals into electrical signals a photosensitive surface digitally records the focused image of the scene. There are basically two types of imagers[6]:

- Tube based imagers such as; vidicons , plumbicons or orthicons.
- Solid-state-sensor; charge-coupled devices (CCD).

There is one major difference between the two types of imagers. With tube based imagers the photosensitive surface are scanned with electron beams, and the two dimensional optical beam is translated into an electrical signal. With solid-state-sensors the electrical signal is read out directly.
The CCD technology is the most extensively used technology type of the two imagers' types. Text based on [35] and [36].

[6] For further information and comparison of imagers and their characteristics see: IVAR '94 Tutorial. Image Acquisition and Display, PatrickWambacq [36].

4.6.2 Raster Scan

When capturing images light from a scene is focused upon a photosensitive surface. The photosensitive surface records the intensities or shades of light as variable charges, and is scanned according to the pattern illustrated in Figure 19. A CCD (Charge Coupled Device) is commonly used as the photosensitive surface, which is an analog device that holds a variable charge and is capable of recording the varying shades of light. To convert the content of CCDs into the digital realm, analog to digital (ADC) converters quantify the variable charge into a discrete number of colors.

The raster scan begins in the top left corner (1) in Figure 19 where the electron beam is turned on (one line at the time). In (2) the beam is turned of to go back to the next line, and then turned off in (3) to go back to the2 top which is known as the “horizontal retrace”. [35] [37]

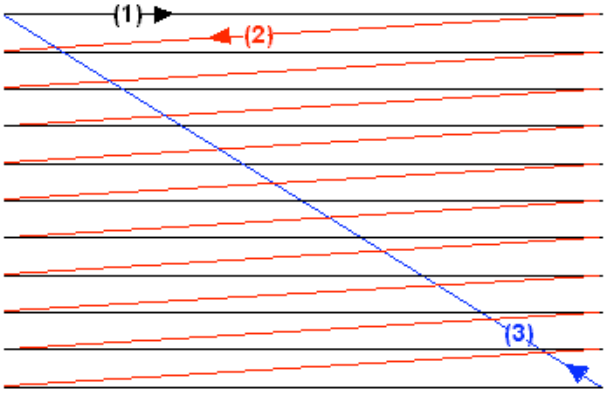


Figure 19, from [37].

4.6.3 Interlaced Raster Scan

Interlaced raster scan is very similar to a regular raster scan. Different however is that two fields sampled at sequentially different times are composed into one frame. This is done in order to reduce the perceived flicker associated with a regular raster scan.
The process involves scanning line 1, then line 3, then line 5 etc. to the bottom of the scan from field 1, and then repositioning the electron beam so that lines 2, 4, 6 etc. are scanned from the next field, thus shortening the interval between each successive scan. [35]
In raster graphics architecture a primitive is formed by scan conversion where each scan line intersects the primitive at two ends, P left and P right. A contiguous sequence of pixels on the scan line between P left and P right is called a Span. Each pixel within the span contains the z, R, G, and B data values. [11]

4.6.4 Representation

In Figure 20 the process of how light is converted to digital representation of the real life scene focused through the optical lens is illustrated. Light rays are focused onto a photosensitive surface, in this instance a CCD, which "converts" the light rays into an electrical signal. The ADC (Analogue Digital Converter) converts the electrical (analogue) signal into a digital one. In the chip marked DSP (Digital Signal Processing) in Figure 20 the digital signal received from the ADC chip is converted (generally this conversion process involves compressing the) into the desired image format.
Based on [38].

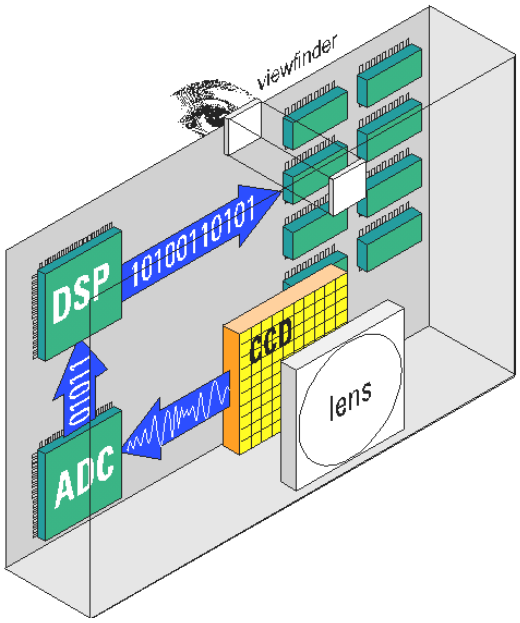
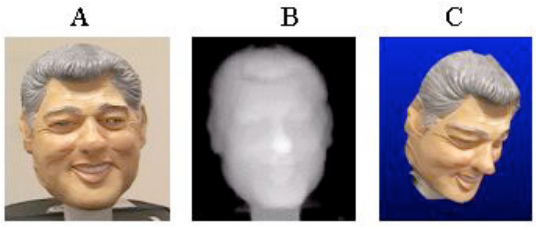


Figure 20 from [38]

4.6.5 Video Cameras

Originally video cameras was designed to create a two-dimensional representation of a scene, as described in chapter *4.6.4 Representation.* However technology is constantly on the move, searching for bigger and better solutions. The latest addition to the DV-camera family is the Z-cam, which allow user to create record both image and depth.

The ZCam™ captures the depth value of each pixel in the scene in addition to the color value, thus creating a depth map of the object by grey level scaling the distances [39] And so by using the PRS (Parallel Range Sensing - invented and patented by 3DV) technology, every video frame is supplemented with an additional frame known as the Z-Buffer [39]. The depth image supplemented by the ZCam has an image (Z-Buffer) corresponding to every frame in the video, and supplements the color information found in the video with the depth information extracted by the ZCam, all done in real-time [39]



Capturing the color image (A) and depth map (B) in order to generate a 3D Model (C).

Figure 21 [39]

The optical system of the sensor itself performs the high-speed parallel calculations for each pixel range. And so information such as the object boundaries; object surface elevations and object distances is provided by a ZCam™ output [39].

4.6.6 3D Display

There are several possible approaches for labeling 3D displays, but basically they are only classified after three distinctive criteria's.

1) In the first approach 3D displays are divided into three categories; Real 3D, true 3D and not true 3D. Real 3D is analogue to the real world and the display device resembles the pattern of light rays emitted from real 3D space. True 3D recreates real life by creating two sets of images from the same scene, where the only difference is the slight difference in the angle of view. By presenting one of the image sets only to the left eye and the other set only to the right, the effect of 3D is achieved. In not true 3D the same image is presented to both eyes. [35]

2) The second approach labels 3D displays either as auto-stereoscopic or stereoscopic. With a stereoscopic 3D display device the viewer requires optical means like glasses or special viewing to observe in 3D. Auto-stereoscopic 3D displays the viewer does not require the use of special viewing or any other optical means for viewing in 3D. [35]

3) The third and last approach labels 3D displays according to the type of technology the 3D display construction is based on. Current techniques include; holographic, volumetric, lenticular, parallax based or stereoscopic [7].

Currently autostereoscopic LCD 3D (15") displays can be acquired for less than $1700 [40]. 3D display devices are currently most useful in context with computer applications, allowing the viewer to experience 3D models the way they where meant. Affordable 3D display devices could prove an interesting alternative for mainstream consumers looking for an enhanced viewing experience.

4.7 MPEG-4

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group), the committee that also developed the Emmy Award winning standards known as MPEG-1 and MPEG-2. These standards made interactive video on CD-ROM, DVD and Digital Television possible. MPEG-4 is the result of another international effort involving hundreds of researchers and engineers from all over the world. MPEG-4, with formal as its ISO/IEC designation 'ISO/IEC 14496', was finalized in October 1998 and became an International Standard in the first months of 1999. The fully backward compatible extensions under the title of MPEG-4 Version 2 were frozen at the end of 1999, to acquire the formal International Standard Status early in 2000. Several extensions were added since and work on some specific work-items work is still in progress. [41]

One of the main objectives with the MPEG standards was to avoid a multitude of proprietary non-inter-working media formats. The basic idea is similar to most standardization project, to achieve global interoperability. MPEG-4 is backward compatible with all prior releases, and seeks to achieve interoperability by providing standardized ways to:

[7] For more information on these 3D specific technologies [35] is recommended.

- Represent media content like visual, auditory or audiovisual content, often referred to as media objects. These media objects can be of synthetic or natural origin (recorded with a camera or microphone, or be computer generated).
- Describe the composition of the media objects in a scene, in order to create complex audiovisual scenes.
- Multiplex and synchronize data associated with these media objects. In order to assure that the media objects are transported across the network with the appropriate QoS (Quality of Service) parameters considered necessary for the specific media objects.
- Interact with the audiovisual scene generated by the user equipment at the receiver's end [41].

The MPEG-4 specification organizes scene graph (the scene graph is composed of several media objects) in a hierarchical fashion. At the leaves of the scene graph we find primitive objects such as

- Still images (e.g. as a fixed backgrounds like landscape, buildings and so on);
- Video objects (e.g. a talking person - without the background;
- Audio objects (e.g. the voice associated with that person, background music);

These primitive media objects are coded according to the MPEG-4 specification, which is designed to be as efficient as possible while still attending to desired functionalities like: Error robustness, easy extraction and editing of a media object, or having an object available in a scalable form. The coding allows all media objects to be represented independent of its surroundings or background, thus elements in a scene can be extracted and inserted dynamically. Based on [41].

4.7.1 Scene composition in MPEG-4

In Figure 22 from [41] an example of an MPEG-4 scene is illustrated. The figure (Figure 22 **[41]**) is composed of several different types of primitive objects such as; the person, the globe and sound (voice). By combining primitive objects like the visual object corresponding to the person with the corresponding voice, a new compound media object is formed. The new media object will then contain both the aural and visual component of a talking person. [41]

The hierarchical structure of a scene graph specified in MPEG-4, allows authors to construct complex scenes. The MPEG-4 specification is highly versatile and offers a wide set of function to manipulate both primitive objects and compound media objects. MPEG-4 provides the possibility to:

- Place media objects anywhere in a given coordinate system.
- Change both the geometrical and acoustical appearance of a media object, through the use of transforms.
- Apply streamed data to media objects.
- Interactively change the viewpoint and the listening points in a scene.

Based on text from [41].

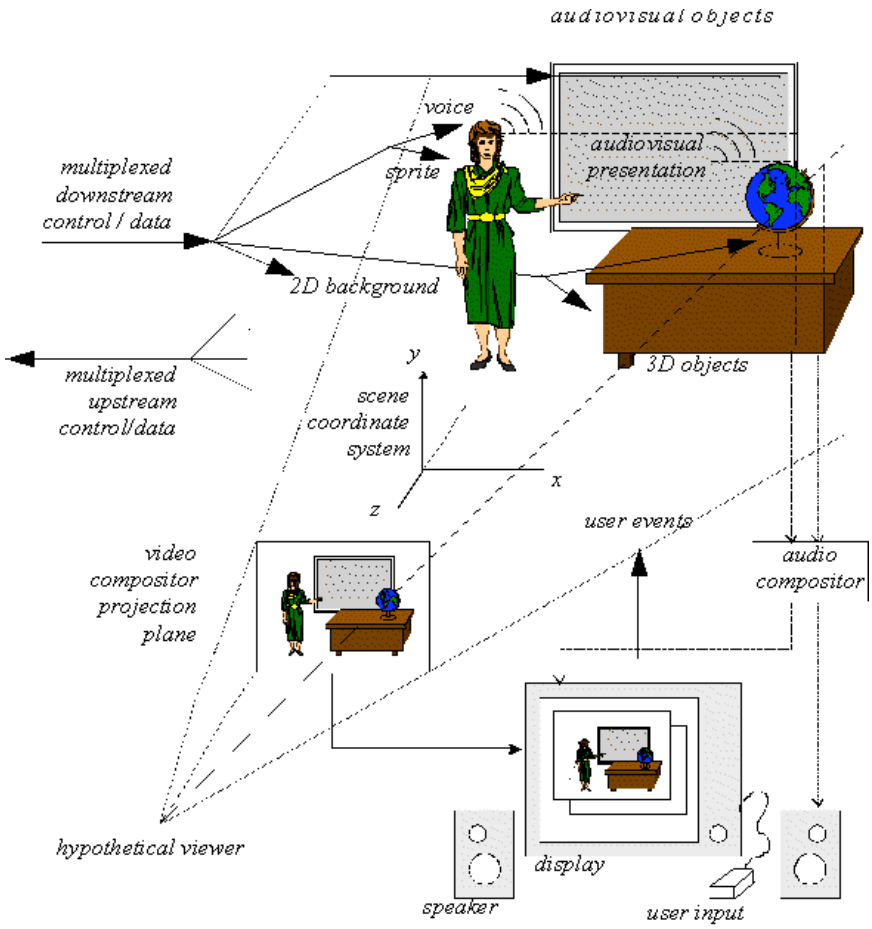


Figure 22 [41]

Figure 22 **[41]** illustrates one of the core principles associated with MPEG-4, it is object oriented. A complete scene is created from primitive media objects. In Figure 22 **[41]** primitive media objects like the 2D background and 3D objects (person, chair and the desk) are combined with audio objects (voice) to produce a complete multimedia experience.

4.7.2 Major Functionalities in MPEG-4 systems

The MPEG-4 specification is an extensive framework, which basically defines an advanced toolbox for compression algorithms and coding of media content. MPEG-4 specification seeks to offer a complete framework that address all major aspects associated with multimedia content. Therefore the MPEG-4 specification is relatively large and complex. Major functionalities described in the MPEG-4 specification include:

- **Transport**, MPEG-4 does not define transport layers, but instead offers adaptation to a specific existing transport layer.
- **DMIF**, or Delivery Multimedia Integration Framework, is an interface between the application and the transport layer. DMIF supports a wide variety of transport protocols including; IP, ATM, mobile, PSTN, narrowband ISDN and offers transparent communication between peers.
- **Systems**, this part addresses description of the relationship between the audio-visual components that constitute a scene, and defines an extensive set of tools that can be used

to code, compress, transmit[8] and generate multimedia scenes. For more details see 4.7.3 MPEG-4 Systems.

- **Audio**, supports coding of general audio signals, ranging from very low bit rates to high quality. MPEG-4 also supports speech and synthetic audio coding. Speech coding can be done using bit rates from 2 kbit/s up to 24 kbit/s, averaging as low 1.2 kbit/s when utilizing the possibility for variable rate coding.
- **Visual**, MPEG-4 allows hybrid coding of natural (pixel based) images together with computer generated 3D models and graphics in a scene. For more details see 4.7.4 MPEG-4 Visual.
- **Extensions**, MPEG is currently working on several possible extension to MPEG-4 specification.

4.7.2 Major Functionalities in MPEG-4 systems is based on text from [41].

4.7.3 MPEG-4 Systems

4.7.3 MPEG-4 Systems is copied from [41].

As explained above, MPEG-4 defines a toolbox of advanced compression algorithms for audio and visual information. The data streams (Elementary Streams, ES) that result from the coding process can be transmitted or stored separately, and need to be composed so as to create the actual multimedia presentation at the receiver side.

The systems part of the MPEG-4 addresses the description of the relationship between the audio-visual components that constitute a scene. The relationship is described at two main levels.

- The Binary Format for Scenes (BIFS) describes the spatio-temporal arrangements of the objects in the scene. Viewers may have the possibility of interacting with the objects, e.g. by rearranging them on the scene or by changing their own point of view in a 3D virtual environment. The scene description provides a rich set of nodes for 2-D and 3-D composition operators and graphics primitives.
- At a lower level, Object Descriptors (ODs) define the relationship between the Elementary Streams pertinent to each object (e.g the audio and the video stream of a participant to a videoconference) ODs also provide additional information such as the URL needed to access the Elementary Steams, the characteristics of the decoders needed to parse them, intellectual property and others.

Other issues addressed by the MPEG-4 Systems part:

- A standard file format supports the exchange and authoring of MPEG-4 content
- Interactivity, including: client and server-based interaction; a general event model for triggering events or routing user actions; general event handling and routing between objects in the scene, upon user or scene triggered events.
- Java (MPEG-J) is used to be able to query to terminal and its environment support and there is also a Java application engine to code 'MPEGlets'.
- A tool for interleaving of multiple streams into a single stream, including timing information (FlexMux tool).
- A tool for storing MPEG-4 data in a file (the MPEG-4 File Format, ‘MP4’)

[8] MPEG-4 provides a transparency between the MPEG-4 application and a variety of transport protocols.

- Transport layer independence. Mappings to relevant transport protocol stacks, like (RTP)/UDP/IP or MPEG-2 transport stream can be or are being defined jointly with the responsible standardization bodies.
- Text representation with international language support, font and font style selection, timing and synchronization.
- The initialization and continuous management of the receiving terminal's buffers.
- Timing identification, synchronization and recovery mechanisms.
- The initialization and continuous management of the receiving terminal's buffers.
- Timing identification, synchronization and recovery mechanisms.
- Datasets covering identification of Intellectual Property Rights relating to media objects

---

4.7.4 MPEG-4 Visual

The MPEG-4 Visual standard allows the hybrid coding of natural (pixel based) images and video together with synthetic (computer generated) scenes [41]. This enables, for example, the virtual presence of videoconferencing participants [41]. To this end, the Visual standard comprises tools and algorithms supporting the coding of natural (pixel based) still images and video sequences as well as tools to support the compression of synthetic 2-D and 3-D graphic geometry parameters (i.e. compression of wire grid parameters, synthetic text) [41]. MPEG-4 provides a relatively powerful set of tools that can be used to combine computer graphics with video. The framework supports streaming and allows different encoding of objects in the same scene; accordingly visual objects in focus can be coded with higher resolution than objects out of focus. Hence limiting the amount of transferred information. Based on text from [41].

The MPEG-4 visual Part supports a vide variety of bit rates (from 5kbit/s to more than 1 Gbit/s), formats and resolutions (from sub-QCIF to 4000x4000 pixels). It offers state of the art compression algorithms for all bit rates including textures for texture mapping on both 2D and 3D meshes. [40]

MPEG-4 visual Part is embedded with content based functionality, which include:

- *Content-based* coding of images and video allows separate decoding and reconstruction of arbitrarily shaped video objects [41].
- *Random access* of content in video sequences allows functionalities such as pause, fast forward and fast reverse of stored video objects [41].
- *Extended manipulation* of content in video sequences allows functionalities such as warping of synthetic or natural text, textures, image and video overlays on reconstructed video content. An example is the mapping of text in front of a moving video object where the text moves coherently with the object [41].

Scalability

The scalability of MPEG-4 is highly versatile in that it can adjust its coding and decoding complexity, in order to adapt to both the available bit rate, delay and the display resolution. MPEG-4 compression scalability can be accomplished both on the receiver and the sender side. [41]

On the sender side the level of compression is determined by two factors, delay and available bit rate. A high level of compression is time consuming (involves a lot of processing), but allows high quality video objects to be transferred over a connection with relatively low available bit rate. On the other hand by keeping the compression level at a minimum (limited need for processing) the delay is reduced extensively while increasing the demand for bandwidth. [41]

On the receiver side similar options as those on the sender side is available. The decoding processes available in MPEG-4 are also able to compromise between delay (measured in needed processing power) and quality. In practice there are four available decoding techniques in MPEG-4, which are:

- *Spatial scalability,* only a subset of the encoded bit stream is decoded, resulting in a reduced spatial resolution.
- *Temporal scalability*, only a subset of the received bit-stream is decoded resulting in reduced temporal resolution.
- *Quality scalability*, the received bit-stream is translated into several layers of different bit rates, where a combination of the subset of the layers can be decoded into a meaningful signal. The quality of the reconstructed signal relates to the number of layers used for decoding and reconstruction.
- *Complexity scalability,* quality of the decoded signal relates directly to the complexity of the decoder used. The complexity of the decoder might be directly related to the available processing power, so that less powerful decoders only decode parts of the bit stream.

Since MPEG-4 is scene graph is object oriented, scene objects can be encoded and decoded according to desired bit rate, delay and resolution. This means that essential visual objects in focus can be coded or decoded with higher resolution than other “less important” visual objects in the scene. The complexity scalability is available to all visual objects including texture, image and video. [41]

MPEG-4 coding of 2D and 3D meshes

MPEG-4 is embedded with a set of tools for coding of 3D polygonal meshes or TINs (4.2.20 Tessellated Models), which include coding of properties like shading, colors and texture coordinates for polygons. Coding of 2D meshes include; motion tracking of animated objects, 2D mesh and motion vector compression and suspended texture transmission with dynamic meshes. [41]

4.8 OpenGL

The main goal of designing the OpenGL framework was to provide programmers with a standard programming language that could utilize the full power of the graphical processing unit (GPU) of the system. By using the OpenGL framework graphics developers no longer had to concern themselves with hardware specific problems, but instead focus on program specific problems. Today OpenGL is the foremost used application-programming interface (API) for developing 2D and 3D graphics; it is supported by every major operating system (Windows 95/98/2000/XP/NT, Unix, Linux, Mac OS, OPENStep and BeOS), and is callable from Ada, C, C++, Fortran, Python, Pearl and Java. OpenGL offers complete independence from network protocols and topologies. [42] [43]

OpenGL encourages innovation and speeds application development by offering the developer a broad set of embedded rendering techniques, which include:
Viewing, Color, Lighting, Fog (4.2.6 Fog), Blending (4.2.1Alpha-Blending), Texture Mapping (

4.5.1 2D Texture Mapping, 4.5.2 3D Texture Mapping). [44] [45]

4.8.1 OpenGL graphics restrictions

The OpenGL interface consists of about 150 distinct commands that can be used to specify objects and the operations needed to produce interactive 3D applications. Although OpenGL is a extensive hardware independent interface, it has certain limitations. OpenGL does not provide the programmer with high level commands that can be used to describe complex 3D models, such as; cars, trains and airplanes. Instead the programmer is required to describe/build the desired 3D models using a set of geometric primitives. [46]

In order for OpenGL to remain a hardware and platform independent interface, no commands for performing windowing tasks or handle user input are included in the OpenGL API.

4.8.2 OpenGL graphics operations

OpenGL is a large and complex graphic system embedded with an extensive set of tools that can be used to manipulate and create stunning graphics[9]. However, to thoroughly describe the OpenGL graphics system is a considerable task and beyond the scope of this report. Therefore only a brief description of major graphics operations is listed:

1. OpenGL construct shapes from geometric primitives, and in so doing it creates a mathematical description of objects (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.).
2. Arrange the constructed objects in a 3D space with basis in a given viewpoint, and then extract the “projected view frustum ” of the composed 3D scene.
3. Calculate the effect of specific lighting conditions for the color of tall objects in a 3D scene.
4. Convert the mathematical description of 3D objects and their associated color

[9] For more detailed information on OpenGL rendering see [46].

information into a 2D pixel array that can be displayed on the screen. This process is called *rasterization*.

During these four stages OpenGL can also perform occlusion culling, thus removing hidden objects surfaces. In practice OpenGL performs its task analogous to the steps described in 4.2.21 Graphics Pipeline.

Since OpenGL is platform and hardware independent platform, the command format is identical on both the server and client side. Thus OpenGL programs can work across networks even if the server and client are running on completely different platforms (Unix and Windows). [44**Error! Bookmark not defined.**]

4.8.3 OpenGL Architecture

When OpenGL was created the designers wanted the architecture to be flexible, in order to make sure that vendors could tailor the OpenGL implementation to meet unique system and performance objectives. The solution was an architecture that could be executed on both dedicated hardware, run as software routines on the standard system CPU, or a combination of both. By introducing the OpenGL extension mechanism, hardware developers where able to differentiate their products by developing extensions. In the context of OpenGL extensions are vendor specific API that allows software developers to access additional performance and functionality. [44]

Several different OpenGL extensions are available through the OpenGL Extension Registry, which also contains naming conventions as well as guidelines for crating; new extensions, extension specifications and other related documentation. [44]

4.8.4 LWJGL (Lightweight Java Game Library)

The Lightweight Java Game Library (LWJGL) is a solution aimed directly at professional and amateur Java programmers alike to enable commercial quality games to be written in Java [47]. LWJGL provides developers access to high performance cross-platform libraries such as OpenGL (Open Graphics Library) and OpenAL (Open Audio Library) allowing for state of the art 3D games and 3D sound [47]. Additionally LWJGL provides access to controllers such as Gamepads, Steering wheel and Joysticks. All in a simple and straightforward API. The LWJGL project also has bindings for FMOD and DevIL, aka OpenIL [47].

---

5 Specification

Propose and specify a distributed game using graphics and video

The idea in this thesis is to create a distributed online game, similar to existing games where you can play against other people on the Internet. However I wanted to incorporate some new ideas into the game that similar games does not have at the current time.
The idea for the distributed game application presented in this paper appeared as result of relentless focus on designing a game application within area **D** in Figure 1. Considering that the arrival of massive multiplayer online games have been astonishingly successful, the idea was to build further on some of the same concepts and ideas that constitute the bedrock of these games, which are:

- Player to player interaction
- Game content are largely produced by participating players

The concepts mentioned above are associated with network effects so that the "value" of the game is proportional to the number of players that are linked to the game.

5.1 The Game: Avatars-Online

Avatars-Online, the name is set to reflect the idea of numerous players who plays an online version (the avatar) of them selves in a virtual world that only can be accessed through the Internet, thus the name Avatars-Online (and slightly inspired from the success of the Funcom success anarchy-online).

In order to participate in the game players need to create an online account. Once an online account has been created players need to log onto a game server. Since the game is real-time based (meaning that players need to respond immediately to the constantly shifting virtual environment), players will automatically be directed to servers within near proximity.

The setting of the virtual world could be anything from medieval to futuristic, but the setting of the game described in this report is set to a futuristic virtual world. Detailed description and thematic for this futuristic world is not a essential part of this report, and therefore a world description has been limited to the following expressions; daunting, incredible, innovative and amazing. Most important is that the world can offer the players a world where they can escape from boring real life everyday routines. Instead players can enter a virtual world where the everyday routine is daring adventures packed with frightening monsters that needs to be defeated.

5.2 Interaction in Avatars-Online

Communication
The game is played in a similar fashion as most multiplayer online games are played today, the only major difference between Avatars-Online and other games is that a headset equipped with a microphone (and support for 3D audio) is needed in order to play the game. The head set is an essential piece of equipment that is vital for communication with other participating players. Once a player has logged on to a game server communication with all other players logged on to the same server is possible. Every avatar is surrounded by a fixed communication edge. When other player's avatar enters this edge, communication between these two avatars is possible. The game application dynamically updates which avatars that can talk to each other. If there are more

than two avatars within each other's communication edge, all avatars can communicate to each other simultaneously.

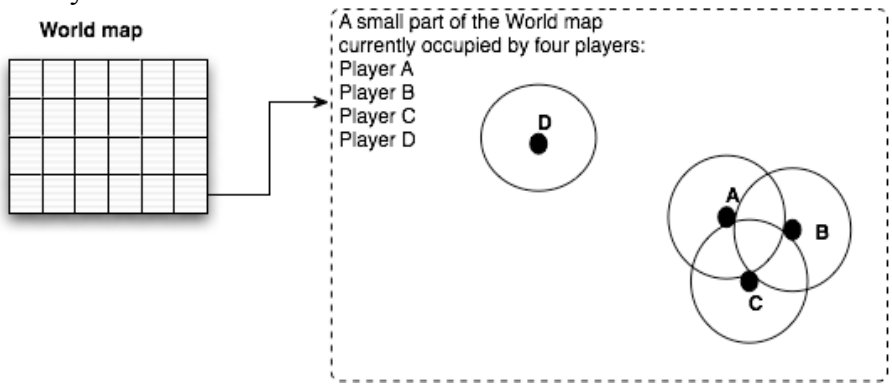


Figure 23

Figure 23 the position of four players player *A*, *B*, *C* and *D* in a virtual three-dimensional world, and their communication edge marked by the surrounding circle. In Figure 23 the player *A, B* and *C* are able to communicate with each other because their position in the virtual world places them within the communication edge of each other. Players within each others communication edge are

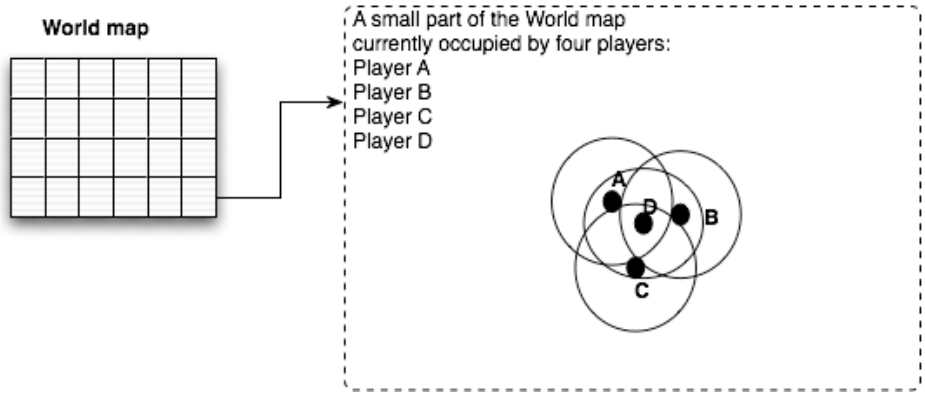


Figure 24

In Figure 24 player *D* has changed his position in the virtual space allowing him to join in on the conversation between player *A, B* and *C.* The game application continuously monitors and updates the communication list for each player in the game. In order to enhance the feeling of a virtual reality audio is appended to the player's avatars, which is equivalent to 3D-audio. In practice this allows players with the right user-equipment (6.1 surround) will be able to match the sound with positions in the 3D virtual world. Thus players are able to separate players on voice characteristics and by positioning the origin of the voice in the virtual 3D world.

In order to hinder players from verbally abusing each other, the game offers an ignore function that allows players to ignore specific players. Once a specific avatar is added to the ignore list of a player, the avatar becomes invisible and soundless from that players perspective. The same happens to the players who are being ignored, they are no longer able to see or hear the ignoring avatar. Thus no direct contact between the two players is possible. Otherwise the virtual world appear the same to both players (ignored and the ignoring).

5.3 Avatar Creation

Conversation alone is not enough to create a virtual environment where players can escape from boring everyday routines. Players need freedom of movement and a set of useful skills that can be helpful when embarking on daring adventures in the virtual world. The virtual world should offer an extensive set of skills players can choose for their avatar, in order to let players create and mold their avatar individually.

In the avatar creation phase players can choose voice filters that will modify the voice of the player to suit the appearance of the avatar.

Once a player has chosen skills for his avatar, the player is granted a fixed amount of money that he can buy equipment to his character for. Equipment can vary from clothing to spaceships, but at the start of the game players should only have enough money to buy the most basic equipment. Consequently encouraging players to go out in the virtual world to generate wealth and resources.

5.4 Movement

Players should be able to move freely around the virtual world. However in order to travel between planets and over vast distances equipment like cars and spaceships must be used, if the avatar cannot afford the equipment he can hitch a ride or use public transport (low cost).

5.5 Actions

The following actions are basic actions that must

- **Trade,** Players are allowed to trade or transfer equipment between each other.
- **Battle** is a major part of the game, and there are always monsters somewhere within the virtual world that needs to be defeated. Players can defeat monsters enemies/monsters together or individually. Avatars-Online allows players to fight, ambush and rob each other at certain areas. Players are not allowed to put other avatars whom they are battling on the ignore list once the battle has started, instead they must fight the battle to end or flee. However in most areas players are not allowed to battle each other, but instead encouraged to go adventuring together.
- **Movement,** avatars have different movement rate in the three-dimensional virtual world. The movement rate reflects the amount of equipment carried and the specific attributes of each avatar. Thus avatars carrying a small amount of weight can move faster than avatars carrying loads of equipment.
- **View**: players can choose to view the world through the eyes of their avatar or from a bird's perspective.
- **Collect,** players must be able to collect gear that they find in the virtual world.
- **Use equipment,** Players are allowed to use a wide variety of equipment including; weapons, transport and more. At all times players have access to their inventory and able to rearrange the way they use their equipment.

5.5 Audio

Audio plays an important part in Avatars-Online, since all communication with other players are verbal. More precisely it's the demand for 3D audio that is vital for the game to achieve its full potential. 3D audio strange as it might sound in this context addresses the need for players to identify where in the virtual world the audio signal originated.

In the game the virtual position of the players in the game can be viewed nodes in a continuously changing topology that will consist of separate nodes (players with no other players within their communication edge) and clusters of nodes (players within each others communication edge). The game application must be able to reconfigure the routing of the received audio stream from every player, to match the current network topology at all times.

5.6 Graphics

Excellent graphics is one of the bedrocks for designing a commercially successful computer game and excellent graphics demand both loads of memory and processing power. In order to limit the amount of transferred data, graphics are generated locally from primitive commands. Accordingly every player needs to have a copy of all graphical objects used in the game stored on the local machine that they are using to play Avatars-Online.

Avatars-Online shall be compatible with 3D display devices.

5.7 system requirements

In order to play the game players needs an Internet connection (preferably 2Mbit+) and a relatively new computer with a graphics card that should be able to handle a 3D display device. The computer must be equipped with support for at least 5.1 surround, but preferably 7.1 surround, in order to exploit the full effect of 3D-sound.

5.8 Derived functionality

The following list of functionality has been derived from the description of Avatars-Online:

5.8.1 Create Avatar

Use case name	**Create Avatar**
Summary	In order to play the game, players must create an avatar that he can direct in the virtual world of Avatars-Online.
Main success scenario:	**1.** The player decides how his particular avatar is designed by combining several pre defined parameters (hair and eye color, gender, skills and more) **2.** After the avatar has been designed the player provides the avatar with a name.
Exceptions:	The player chooses to abort the create avatar procedure, in which case the procedure must be finished at some other time.
Trigger	**1.** The player decides to create an avatar.
Pre condition	Avatars-Online has been successfully installed on the player's

	computer.
Post condition	• The player has successfully created an avatar. • The same as in *Pre condition.*

5.8.2 Buy equipment

Use case name	**Buy equipment**
Summary	The player wishes to equip his avatar from predefined list of available equipment.
Main success scenario:	**1.** The player chooses to abort the buy equipment procedure, in which case the procedure must be finished at some other time. **2.** In case the player tries to buy more equipment than he can carry a dialog box will pop up, forcing the player to either drop or sell a piece of equipment. **3.** The player tries to buy equipment he cannot afford, in which case a dialog window pops up to inform the player
Exceptions:	NIL
Trigger	**1.** The player decides to equip his avatar (buy items from a predefined available at avatar creation phase). **2.** The player has entered a shop in the virtual world
Pre condition	• The player has successfully created an avatar • The avatar enters a shop within the virtual store
Post condition	• The player has successfully equipped his avatar. • The same as in *Pre condition.*
Author	Marcus Vangli

5.8.3 Access inventory

Use case name	**Access inventory**
Summary	Once the avatar has been created players can access their avatars inventory in order to: • Specify exactly what sort of equipment the avatar is currently using (weapons, clothing...). Equipment not used can either be carried for later use, or it can be dropped/discarded.
Main success scenario:	**1.** The player accesses the inventory list of the avatar. **2.** The player decides what kind of equipment the avatar is going to use, by dragging the piece of equipment to correct place (this should be a straight forward and intuitive operation, due to self explaining GUI). **3.** The player can access equipment information by double-clicking on the item, which causes a window with information on the specified item to pop up. **4.** Items can be discarded or dropped by right clicking on

	the specified item. This causes a dialog box to pop up that requires the player to confirm that he wants to discard the selected item.
Exceptions:	**2a:** The player tries to place a piece of equipment at a illegal position (boots on the hands), which causes a dialog message to appear informing the player that the piece of equipment be placed elsewhere.
Exceptional paths	NIL
Trigger	**1.** The player pushes the view inventory button.
Pre condition	The player has successfully created an avatar.
Post condition	• The player has successfully configured the inventory of his avatar. • The same as in *Pre condition.*
Author	Marcus Vangli

5.8.4 Create user account

Use case name	**Create user account**
Summary	In order for a player to participate in Avatars-online he needs to create a user-account (the user-account is created for billing and customer support purposes).
Main success scenario:	**1.** When the game application is installed on the local machine, the user is required to register and create a user-account in order to play the game. Once the game has been properly installed a dialog box pops up asking the player if he wants to create a user-account now or later. **2.** If the player chooses to create a user account, he must fill out the Avatars-Online registration form (the form is used to verify the installed software and for billing). **3.** The player confirms that the form has been properly filled out, before pressing commit.
Exceptions:	**A.** The player has missed to fill out one or more essential parts of the form. In which case the program notifies the player of what is missing. **B.** The player has filled in a invalid product, in which case the player has to redo **2** and **3** in *main success scenario*. **C.** In case the player has an inactive Internet connection, he may choose to fulfill **2** and **3** in *main success scenario* later. But a user account must be created in order for a player to use the online game servers.
Trigger	**1.** A version of Avatars-Online has been successfully installed on the player's computer. **2.** The player decides to go online (play the game for the first time).

Pre condition	A proper version of Avatars-Online has been successfully installed on the player's computer.
Post condition	• The player has created an active user-account. • The same as in *Pre condition.*
Author	Marcus Vangli

5.8.5 Log on

Use case name	**Log on**
Summary	The player has successfully created an avatar and wishes to start playing the game.
Main success scenario:	**1.** The player chooses the game server he wishes to log onto from a list of available game servers.
Exceptions:	The player's user-account is no longer active, and thus the player is not allowed to access a game server.
Trigger	**1.** The player pushes the join game button to a particular server
Pre condition	• The player is in possession of a ready (fully created and equipped) avatar. • The player has an active user-account. Active in this context, denote that the system can verify that the monthly user fee has been paid.
Post condition	• The same as in *Pre condition.* • The player is logged onto a game server
Author	Marcus Vangli

5.8.6 Movement

Use case name	**Movement**
Main success scenario:	Players control the movement of their avatars by using both the keyboard and the mouse. Control of movement should be analogue to similar games, thus making it easier for new player to adapt to the avatars-Online. The movement message consist of the following three different parameters that the player needs to be able to control efficiently: • X-movement/speed • Z-movement/speed • Y-movement/speed (the z-movement is a passive parameter dependant of the terrain.) **1.** The player specifies a point in the virtual world by a left mouse click, the avatar automatically moves towards that point. **2.** Players can also manually control the movement by

	pressing keyboard hot keys that directly control the movement (north, south, east, west and speed) of the avatar.
Exceptions:	• At certain areas movement can be blocked by virtual hinders like; buildings, trees, walls and more. • Other avatars or fiends that tries to obstruct the avatar from continuing on his path can affect movement.
Trigger	• Key_ events: north, south, east, west and speed. • Left mouse click on point in the virtual world guides the avatar on a direct path towards that point.
Pre condition	The player has successfully logged onto the game server of his choice.
Post condition	The same as in *Pre Condition.*
Special Requirements:	Local game application must be synchronized with game application (since Avatars-online it is a "real-time" game) running on the server, and so must all of the other local game application. In practice the application nodes are plesiochronous. Thus allowing some lag between the different nodes and but still keeping a consistent view of the virtual world on all application nodes over time.
Author	Marcus Vangli

5.8.7 Change view

Use case name	**Change view**
Summary	The player changes the viewpoint, from through the avatars eyes to a bird's perspective or the other way around.
Main success scenario:	**4.** The player changes the viewpoint
Exceptions:	NIL
Trigger	**1.** The player pushes the change view button (available on the screen) or the keyboard change view hot-key is pressed
Pre condition	• The player is logged on to a game server
Post condition	• The player is stilled logged onto the game server but the viewpoint is shifted.
Author	Marcus Vangli

5.8.8 Communication

Use case name	Communication
Summary	Communication in Avatars-Online is designed to be very dynamic. The game application automatically checks which avatars that is your avatars communication edge and automatically play the received speech from the other avatars on your sound system. When a player chooses to talk the speech is digitally recorded and sent to the avatars that are within the communication edge. Thus all the avatars players are able to hear what is said.
Main success scenario:	1. Similar to a basic multiple cal session.
Exceptions:	NIL
Trigger	NIL: Once logged on to a game server this process stays active.
Pre condition	• The player is logged onto a game server
Post condition	• The same as in *Pre condition.*
Author	Marcus Vangli

5.8.9 Fire weapon

Use case name	Fire weapon (battle)
Summary	When the player decides to fire the weapon of his choice (the one his avatar is currently equipped with), he places the mouse pointer on thee target and presses the right mouse button. The avatar then fires the weapon.
Main success scenario:	The player right clicks on the target
Exceptions:	The avatar is out of ammunition and consequently the weapon cannot be fired.
Trigger	Key event: Right mouse click fires the weapon;
Special requirements:	Since the game is played in real-time, certain measurement 1. The event is time-stamped. 2. Mouse position is recorded and used as the direction the weapon was fired (x-parameter, y-parameter and z-parameter). 3. Each weapon has parameters as range and damage, which is included in the message to the game server or host.
Pre condition	1. The player needs to be logged on to an active game-session and the avatar needs to have at least one bullet left in the ammunition storage/inventory.
Post condition	1. No ammunition left 2. Ammunition minus fired shell left.
Author	Marcus Vangli

5.8.10 Ignore Avatar

Use case name	Ignore Avatar
Summary	Because of the very nature of Avatars-Online players might end be bothered and annoyed verbally by other players. Therefore Avatars-Online offers players the possibility to ignore other avatars. Once a specific avatar is added to a players ignore list, the player can no longer see or hear the selected avatar. Consequently the virtual world will appear the same to both players (ignored and ignoring) except that they can no longer observe each other avatars directly in any way.
Main success scenario:	**1.** The player left clicks on the target avatar. **2.** The player chooses ignore avatar from the dialog box that pops up
Exceptions:	Nil
Trigger	Player adds target avatar to his ignore list
Special requirements:	Nil
Pre condition	Target avatar is within the player's avatars communication edge
Post condition	Player can no longer observe the ignored avatar.
Author	Marcus Vangli

5.8.11 Gather equipment/gear

Use case name	Gather equipment/gear
Summary	The player comes across items in the virtual world that he wishes to add to his avatars inventory.
Main success scenario:	**1.** The player left clicks on the target piece of equipment **2.** The player chooses to add the target to his inventory list from the dialog box that pops up
Exceptions:	**2a:** The avatars inventory list is full. Thus forcing the player is forced to discard/drop the least desired piece of equipment from the inventory in order to make room for the new piece of equipment.
Trigger	The player chooses: add item to the inventory list from the dialog box.
Special requirements:	Nil
Pre condition	Target avatar is within reach (close distance within the virtual world) of a new piece of equipment.
Post condition	**1.** The avatars inventory list has changed as a result of added equipment **2.** The player has chosen to leave the newly discovered piece of equipment; consequently the post condition is identical to the pre conditions.
Author	Marcus Vangli

5.8.12 Leave the game

Use case name	Leave the game
Summary	The player wishes to end an ongoing game session
Main success scenario:	**1.** The player left clicks on exit game button in the dialog window **2.** A new dialog box appears requiring the player to confirm that he wishes to end the game. **3.** The application automatically saves that current status of all avatar parameters, thus making sure the avatar is ready for subsequently game sessions.
Exceptions:	NIL
Trigger	NIL
Pre condition	The player is logged onto an active game server
Post condition	The player is no longer logged onto an active game server
Author	Marcus Vangli

5.9 Derived non-functional requirements

In order to perform certain aspects of the given description of Avatars-Online successfully according to the expectations of current gamers, certain non-functional requirements have to be fulfilled.

1. **Performance,** the performance of the game application is considered as the time from the player utters a command until the system has processed and displayed the result of that command, which is commonly referred to as the system response-time. The response-time of Avatars-Online should be no more than the average response time of human nerve system.
2. **Graphics,** today computer graphics has reached a level of detail that is close to viewing objects in real life, and current gamers expect nothing but the best. Accordingly graphics in Avatars-online should be of high quality and as close to real life as possible.
3. **Audio,** due to the very nature of Avatars-Online (5.2 Interaction in Avatars-Online) players should have equipment that supports 3D sound, to get fully pleasure from Avatars-Online.
4. **Scalability (load).** The game application need to be able to handle a growing workload gracefully, meaning that an increasing number of active players should only lead to a proportional increase in the game servers workload. A strict definition would be that the workload should be growth of the workload should be O(n) where n is the number of players logged onto the server.
5. **Scalability (geographic).** Players should not feel that the performance (**1**) of the game is related to their geographical whereabouts.
6. **Availability:**
7. **Dependability:**
8. **Extensibility** is an important success factor of Avatars-Online, because it permit rapid introduction of new game features. Consequently players and their avatars can grow with the game, instead of out of it.
9. **Platform compatibility.** In order to maximize the number of potential players/customers Avatars-Online should be platform compatible with the most commonly used gaming-platforms, which include: Windows 2000/XP/Vista, Mac OS X, Playstation 3 and X-box 360.

6 Design

Design parts of the game[10]

In part 5 Specification Avatars-Online was described and evaluated from the perspective of potential users of the system, which resulted in a list of specific requirement. This chapter (

6 Design) is devoted to describing the resulting system and how it should be constructed to best accommodate the requirements in 5.8 Derived functionality and 5.9 Derived non-functional requirements.

6.1 Hardware Architecture

Basically there is only one type of network topology that can be used to meet the demands of Avatars-Online, which is the server-client architecture described in Figure 25 below. A prerequisite for using the Avatars-Online application is that all clients are in possession of an Internet connection with a minimum bandwidth of +2Mb/s, typically an ADSL connection.

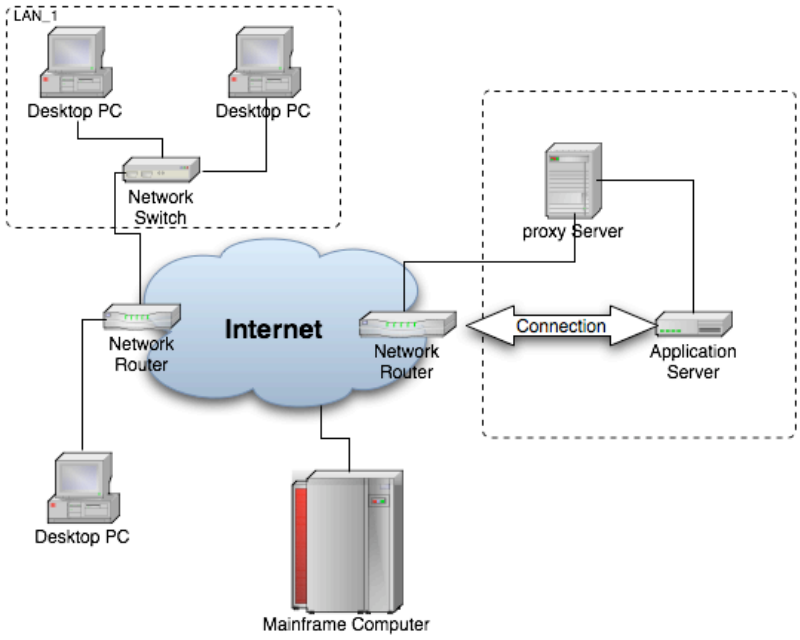


Figure 25

The proxy server is used to manage incoming requests. Request originating from clients with a valid game-key (indicating that the player has paid the monthly subscription fee) are directed to the game server. Requests originating from players without a valid game-key are redirected to a billing server where the potential player can buy a valid game-key.

When a player logs on to a game server for the very first time, the proxy needs to send a request to the mainframe computer to validate that the player has installed a valid copy of Avatars-online on his computer, if the request checks out the proxy issues a valid game-key to the requesting player. The player is then offered to log onto the server

[10] Due to a fault the task originally was "*Design the game",* however it has later been corrected to its intended form.

Once a player is granted access to the game server a direct connection between the game server and the client is established. This process is illustrated in Figure 26 below.

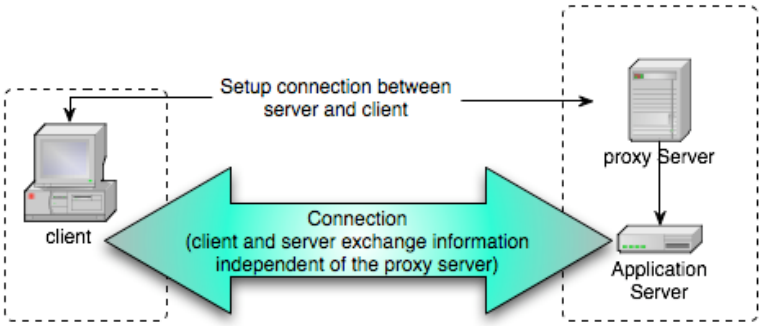


Figure 26

6.1.1 Geographic scalability

In 5.9 Derived non-functional requirements geographic scalability was listed as one of the non-functional requirements that had to be fulfilled. To achieve the desired level of real-time (geographical) scalability the game servers and the clients needs to be in relatively close proximity, in order to prohibit a considerable increase in signal propagation delay. Thus Avatars-Online need to offer players available application servers situated within certain geographic vicinity (i.e. a player geographically situated in Norway wishes to log onto a game server physically located in Sydney, would most likely experience that the signal propagation delay between the two cities as annoying). The concept is illustrated in Figure 27 where each city is represented with a local application server allowing players to log onto the application server with the least propagation delay. All customer data is contained in the Avatars-Online mainframe computer and since communication with the mainframe computer does not impose real-time constraints, only one mainframe computer is needed [11].

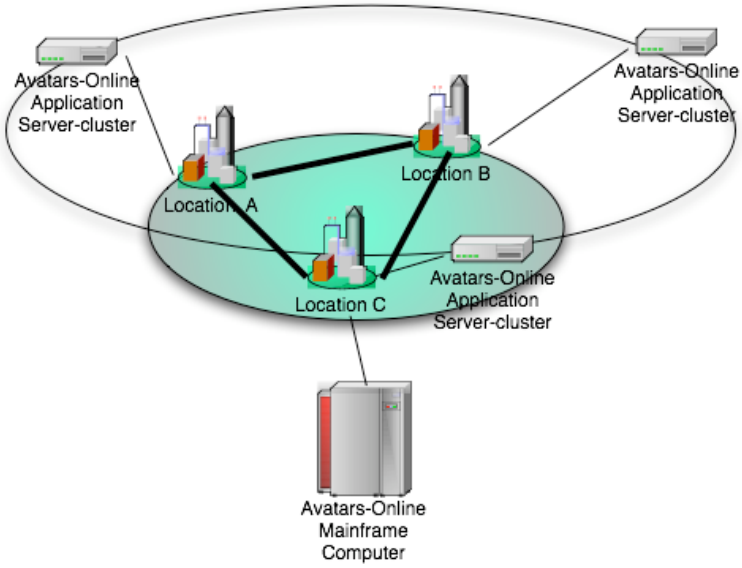


Figure 27

[11] In Figure 27 the Avatars-Online Application servers refers to a cluster of servers, the composition (number of servers and type. The specific configuration will differ from cluster to cluster due to variations in workload from location to location).

6.1.2 Operational non-functional requirements

Operational non-functional requirements are qualities concerning the operational system, such as availability, reliability and fault-tolerance (requirements are specified in 5.9 Derived non-functional requirements). The operational non-functional requirements (availability, reliability and fault-tolerance**)** all have a relatively clear and standardized definition, and they are therefore relatively easy to measure[12].

To ensure that the fault-tolerance requirement is met, a redundant hardware architecture like the one illustrated in Figure 28 is necessary. The scheme is to operate with two sets of network routers where one set is active while the other contains hot reserves. In case one of the active routers in set A fails, a hot reserve from set B is ready to take over, thus reducing the downtime considerably. The size of the two sets should adequately large to accommodate the operational non-functional requirements (availability, reliability and fault-tolerance**)**

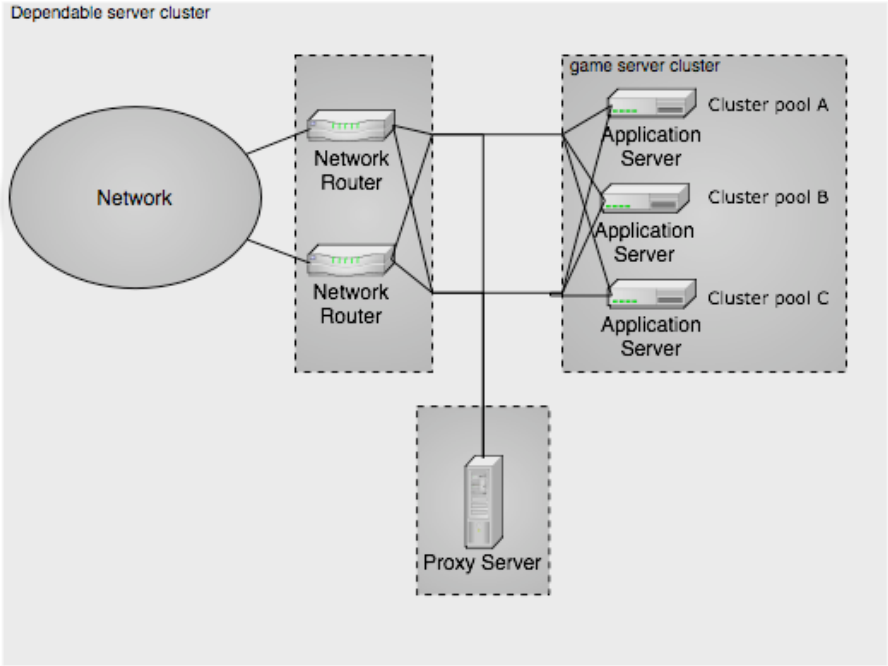


Figure 28

The game-server-cluster contains a pool of active servers (type 1 in Figure 28) and a pool of hot reserves (type 2 in Figure 28). In case of a sudden peak in the workload one of the hot reserves can "step in" and relieve the active servers of some of their workload. The last type (type 3 in Figure 28) is a pool of cold reserves that are used to replace failed servers from the active or hot pool. However since the workload on the server-clusters will differ from geographic location to geographic location, the exact composition of the server-clusters should be configured according to the experienced workload on that specific server-cluster. Figure 28 is only designed to illustrate the concept of how hardware can be configured to increase; reliability, availability, dependability and fault-tolerance.

[12] For detailed description of how to measure and design dependable systems [TTM 4120 dependable computing systems and communication, Department of Telematics, NTNU 2003] is recommended].

6.1 Software Architecture

Avatars-Online is described as a massive multiplayer online game, the

6.2.1 Client Software Components

In Figure 29 the software structure running on a client machine is illustrated. On top of the software stack is the game application, which controls the other software (OpenGL and Audio) components during an active game session. The Avatars-Online communication component is an integrated part of the game application, but has been placed among the standardized components because it contains logic that operates independently of the *Game-Application*. The *Game-Application* should be implemented partly in C/C++ and partly in assembler in order to achieve maximum application speed.

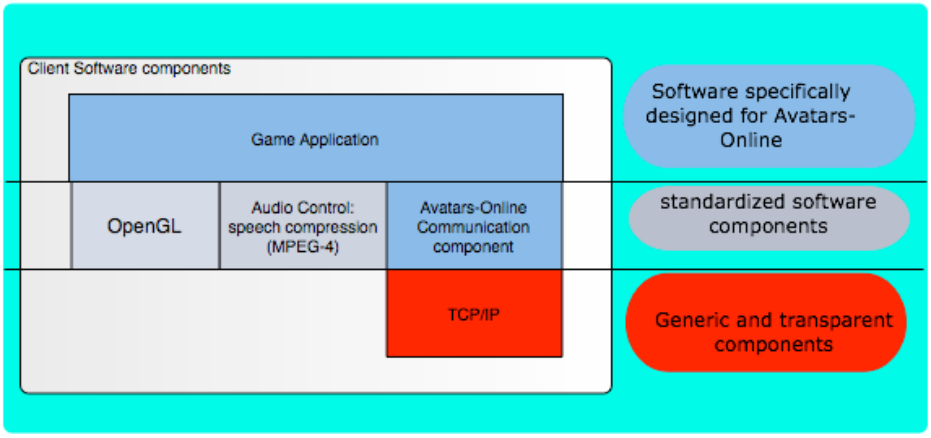


Figure 29

6.2.2 Local Processing

The two processes in Figure 30 (the *LocalGameEngine* and the *LocalGameAgent*) that have to share the only CPU. The *LocalGameAgent* is equivalent to the communication component in Figure 29. Once the game has been initiated the *LocalGameEngine* is given control of the CPU and keeps it as long as the game application is running, or until the underlying operative-system grants it to another process. The *LocalGameAgent* basically works as organizer of incoming data from the server and data sent to the server.

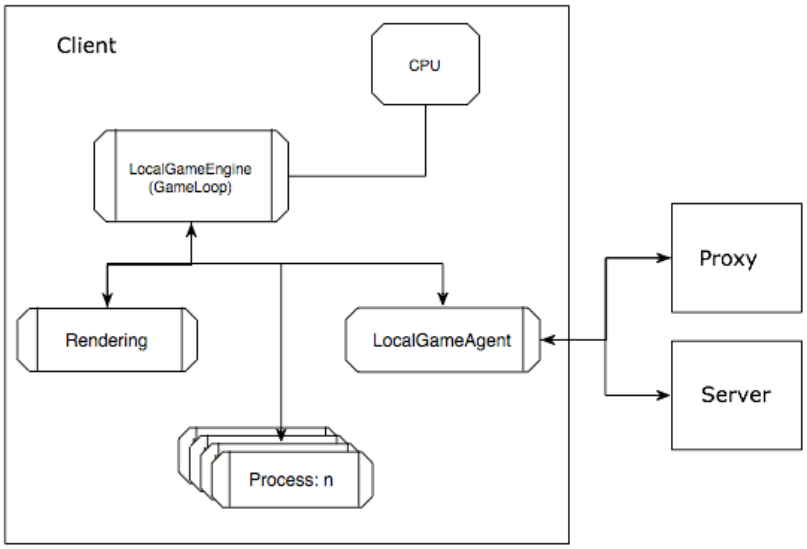


Figure 30

In order to reduce the traffic between the game server and the clients, all clients are in possession of an identical file-system that contains all the graphical objects that are used to build the virtual

World. Data transferred between the server and client is limited to primitive commands that are used to manipulate the graphical objects. Consequently the graphical objects level of quality will not affect the amount of traffic between the server and the client. The level of quality of the graphical content displayed is thus only limited to the processing power of the client's computers.

Avatars-Online is a real-time massive multiplayer online game, which in practice means that performance is of vital importance. If the player's actions are not immediately reflected in the rendered scene, the application will give the player the impression it has a life of its own. Consequently user input has to be monitored and processed quickly enough to give the player a feeling of instant control of the displayed scene. The solution to this problem is described in 6.2.3 Processes.

In Figure 30 only three processes are named *LocalGameLoop*, *Rendering* and *LocalGameAgent* and a reference to a number (n) of unidentified processes are included. Once the game application is up and running the process named *LocalGameLoop* is running the show. All other process that are called, are called from this process.

6.2.3 Processes

In order to construct a real-time game application that will work successfully, it is imperative to display the effect of the user input in the displayed scene, to accommodate this performance requirement the local game application is controlled through a continuously iteration of the *LocalGameLoop* (described in Figure 31, Figure 33 and Figure 32).

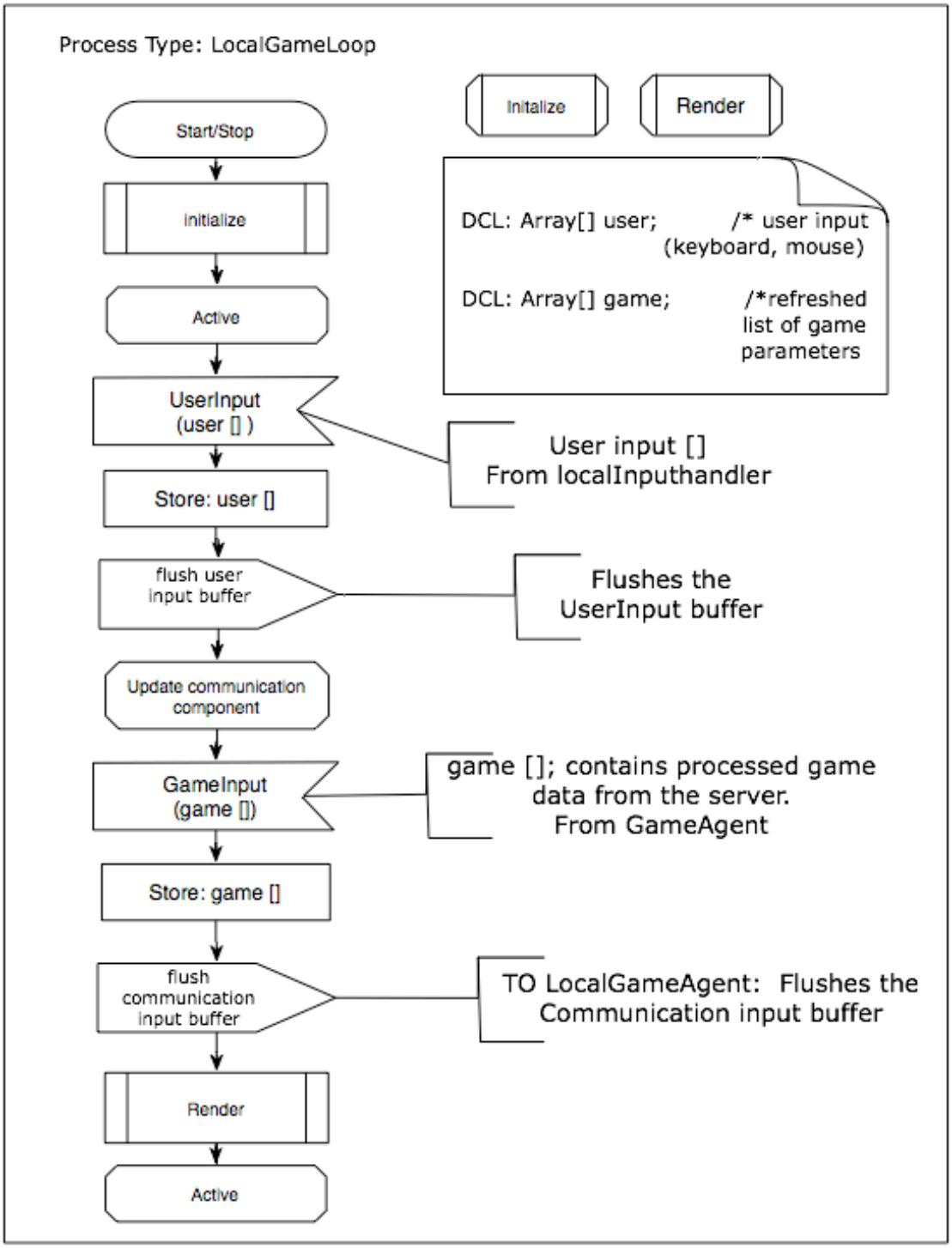


Figure 31

The *LocalGameLoop* is designed to be the "brain" of the application running on client machines. Within an iteration of the *LocalGameLoop* both input from the user and the game server is gathered and processed. The *LocalGameLoop* continue (repeat itself) to execute its logic, until it receives the endgame command (either from the server or the player), described in Figure 32.

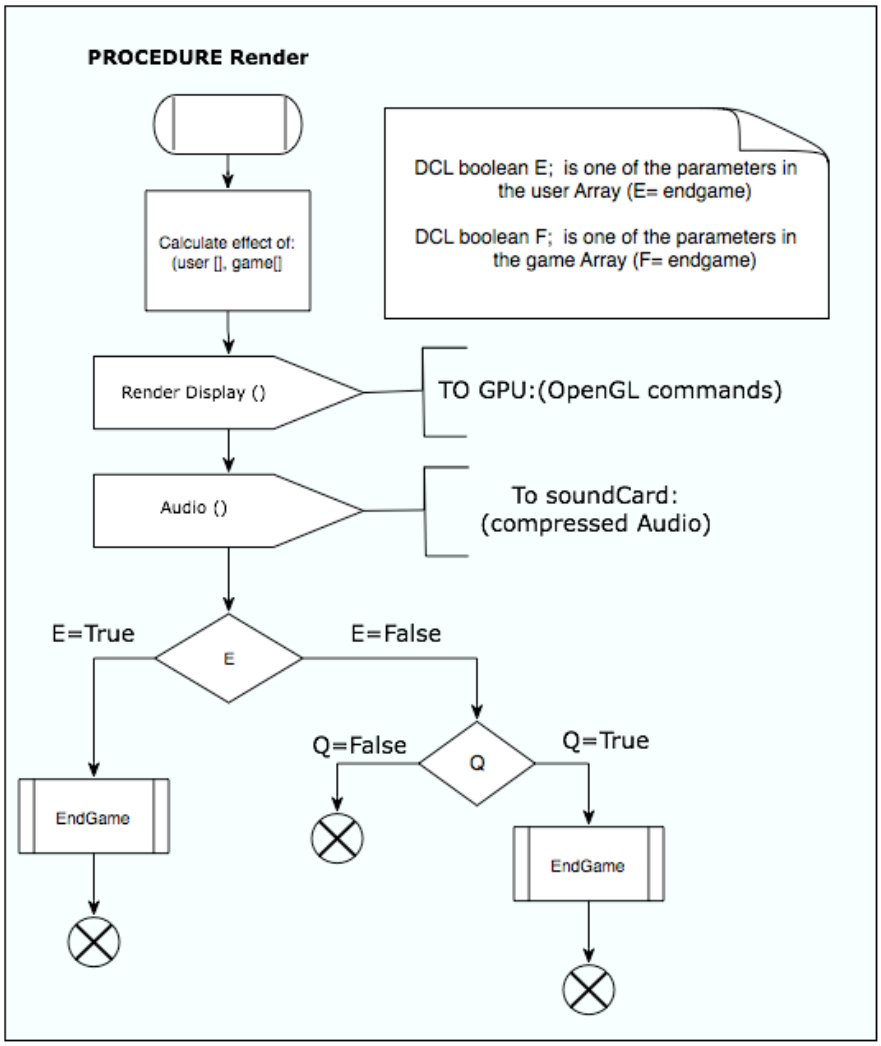


Figure 32

The *EndGame* procedure is not thoroughly described, since all it really does is to end the game. Ending the game is analogue to ending the *LocalGameLoop*, thus once the end game procedure is called the *LocalGameLoop* process is stopped. Consequently the other processes are allowed to take control of available machine resources.

In the box named *calculate effect of: (user [], game [])* in Figure 32 the effect of both user-input and game-input is processed. The Rendering procedure is the most process demanding part of the game application, it involves a lot of computing since it has to calculate the effect of the player and the effect the other players actions has on the avatars that are within certain range (visual range in the virtual world) of the player. Once the effect has been calculated it is sent to the GPU controller in form of OpenGL statements.

In order for the *LocalGameLoop* to start a direct connection with the game server needs to be established. The *Initialize* procedure instructs the *LocalGameAgent* to establish a connection with the game server. Specific details on the construction of the *LocalGameAgent* have been left out, since the goal was to illustrate how the *LocalGameLoop* is structured.

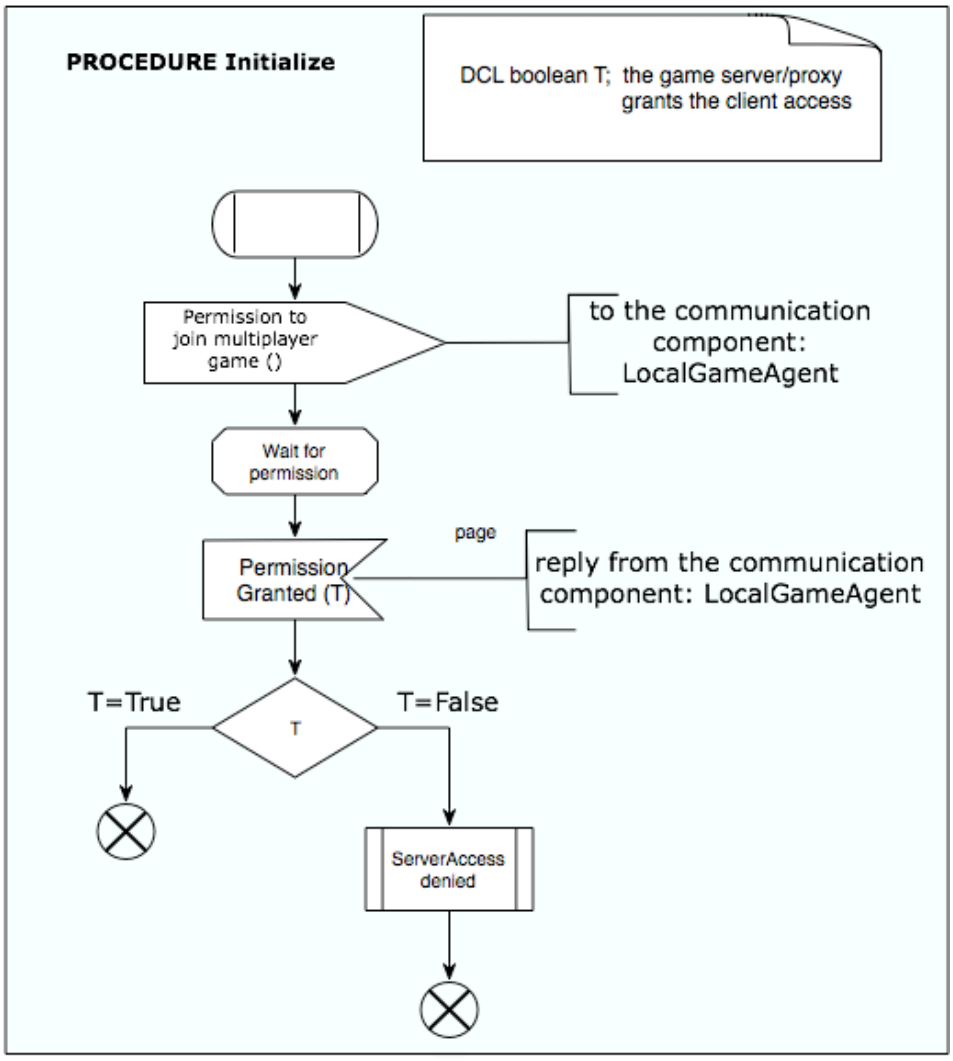


Figure 33

If the client is denied access to the server (T=false) for some reason (server is down, the player does not have a valid game key, network failure and so on) the *LocalGameAgent* process cannot be activated. Instead other processes are started depending on the reason of the access denied message.

6.3 Realization of the functional requirements

There are some terms used in the UML-sequence charts that need some degree of clarifying, so that they are interpreted correctly. The term *GameLoop* and *GameEngine* is used as extensively used as references in the MSCs below. They are closely related but there is subtle difference; *GameEngine* is a software class that contains the process *GameLoop*. There are two different version of the *GameLoop*.

- The *LocalGameLoop*, which is the version of the *GameLoop* clients are equipped with.
- The *MainGameLoop,* which is the version servers are equipped with.

Once a game-server is active, the *MainGameLoop* continuously monitor events and calculate the effect these events have on the game globally. The *MainGameLoop* is responsible for keeping the players

The *GameAgent* is a software object situated on the server/host. For every player participating, the game-server/host creates a *GameAgent* that represent the respective player. All communication between the client and server/host is conducted through the related *GameAgent.* Clients are equipped with a process called *LocalGameAgent,* which is responsible for communication with the server. The *LocalGameAgent* will in practice work as a passive container/buffer, which the *LocalGameLoop* regularly checks (and flushes), in order to continuously monitor the continuously transmits game-data from the server. However the Clients are only equipped with one *LocalGameAgent* and for that reason it is not necessary to refer to it in the MSCs, instead the term client is used.

The MSC has been limited to only procedures that involve direct communication between the server and the client.

6.3.1 FR: Create Avatar

The create avatar process does not involve direct communication between the server and the client, since the process is executed on locally and the result stored in as an (avatar) object on the client machine. However once the player logs onto a server the specific avatar-object parameters are transferred to the according to 6.3.5 FR: Configure avatar.

6.3.2 FR: Buy Equipment

The buy equipment process can partly be viewed as an integrated part of the create avatar process since it is executed locally. In this case the server will receive the necessary data on the avatars equipment once the player logs onto the server.

However if the player is logged onto a game server and has entered a shop in the virtual world and the player chooses to buy an item, the associated process will involve communication with the server. Before a message is sent the client will reconfigure the avatar object according to the purchased item. In this case the MSC will be identical to
6.3.5 FR: Configure avatar. The same procedure is used when accessing the avatars inventory, since the effect of rearranging the inventory basically is the same as reconfiguring the avatar. Thus the MSC in
6.3.5 FR: Configure avatar will be valid for both, 6.3.2 FR: Buy Equipment and

6.3.3 FR: Access Inventory

See 6.3.2 FR: Buy Equipment

6.3.4 FR: Create user account

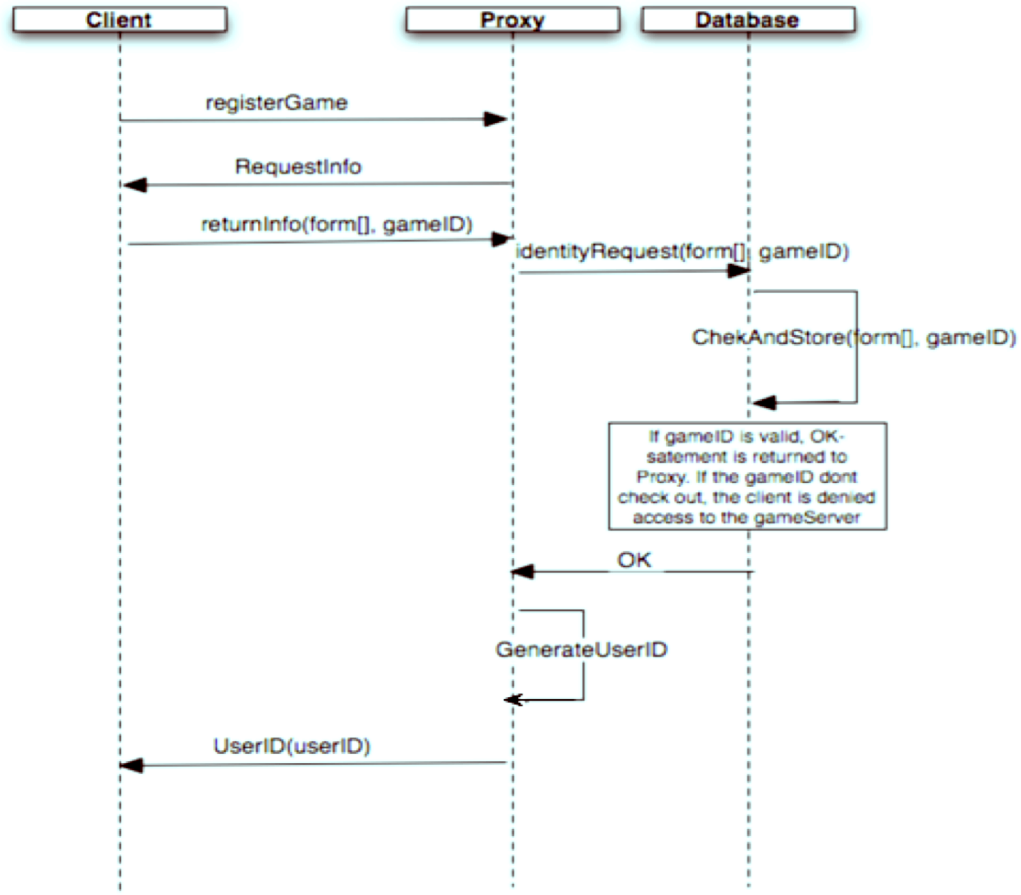


Figure 34

The database access is needed to check weather the game has been purchased or if it is an illegal copy. If desired the customers can be forced to fill out registration form, so that customer information can be stored for future use.

Once the proxy has generated a *userID* the *userID* is stored locally in the clients files-system so that this process don't have to be repeated. The proxy also stores a copy of the *userID* in order to avoid unnecessary database accesses.

6.3.5 FR: Configure avatar

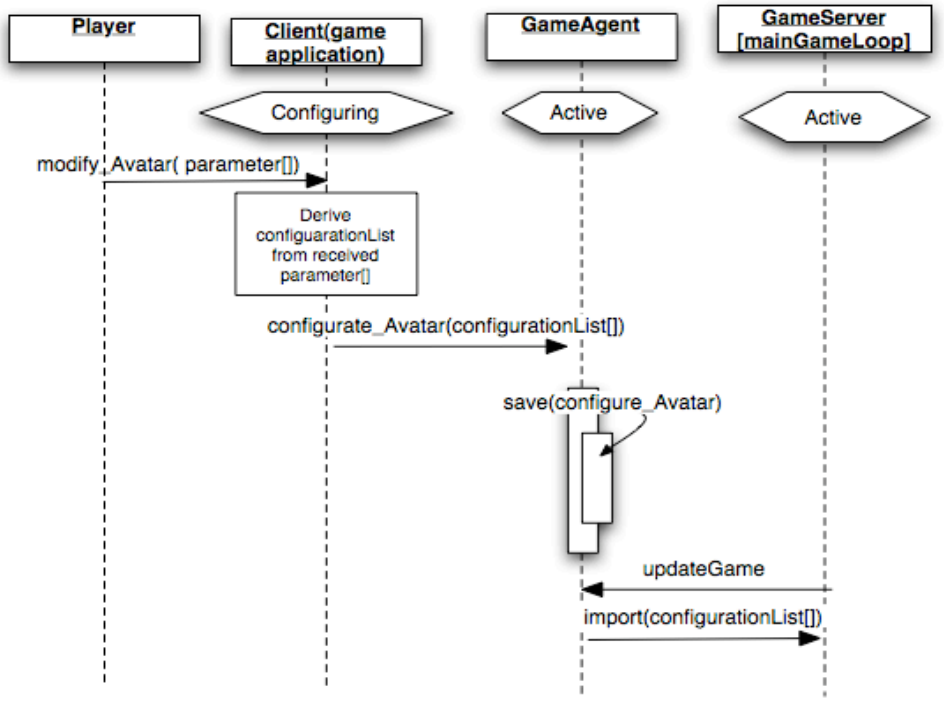


Figure 35

The game application on running on the client will reconfigure the avatar object according to the instructions contained in the *parameter []* array. Once the avatar object has reconfigured it self the derived result is stored and appended to the *configurationList []* and passed on to the server.

6.3.6 FR: Logon

In order for a player to logon to a game server, the player needs to have a valid game-key. The only way to acquire a valid game key is to register as a customer of Avatars-Online (Figure 36).

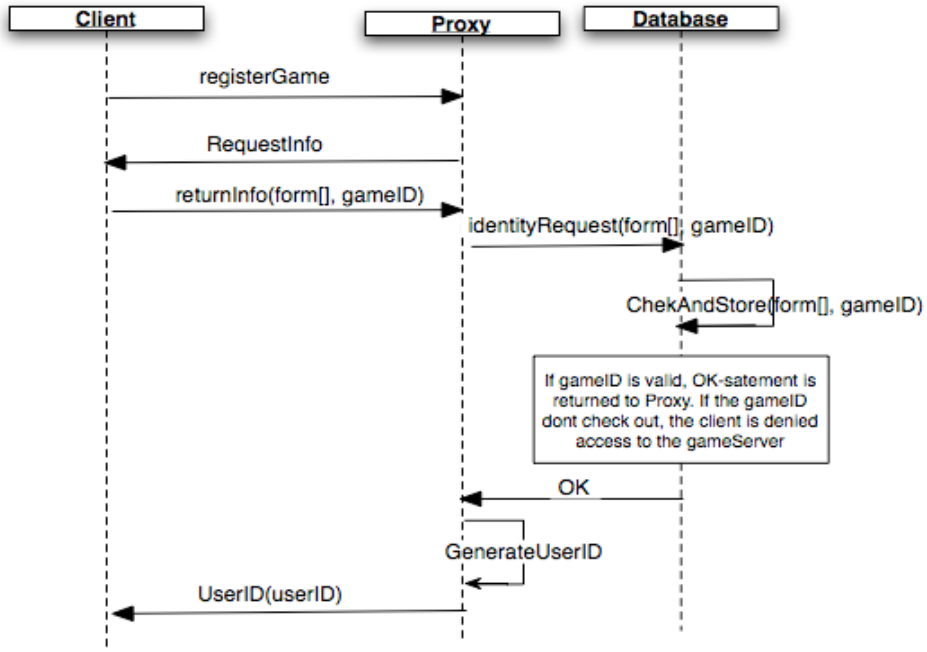


Figure 36

The database access in Figure 36 is needed to check whether the game has been purchased legally or if it is an illegal copy.

Once the player has received a valid game-key (game-key is equivalent to the *userID* in Figure 36 and Figure 37) the player is ready to log onto a game server.

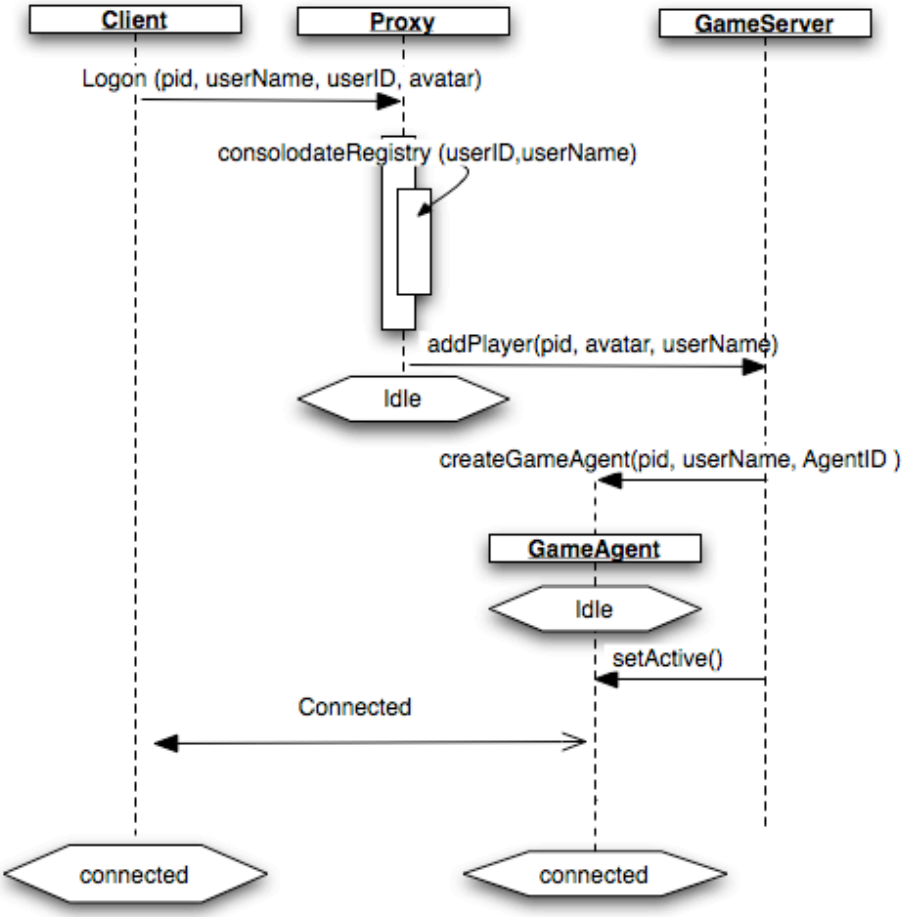


Figure 37

The reader should notice that the logon process in Figure 37 that eliminates the proxy server once a connection between the game server and the client has been established. Elimination of the proxy is done in order to cut down the propagation delay to a bare minimum and relieve the proxy from unnecessary work.

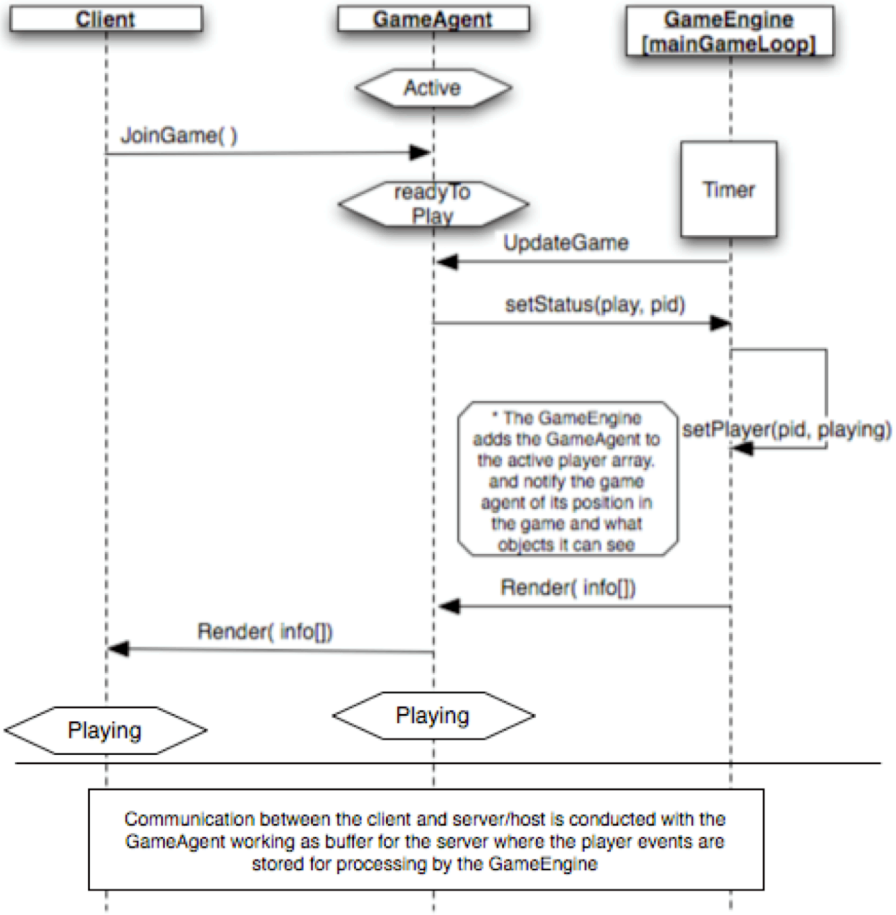


Figure 38

Once a player has joined the states are changed in the *GameAgent* and on the client side. Once the *GameEngine* noticeses the state change in the *GameAgent,* the *GameEngine* adds the respective *GameAgent* to the array containing the address of clients currently in play. By doing this we achieve a higher level of game-response since the *GameEngine* now will check the *GameAgent* for data-updates once in every iteration of the main *GameLoop*.

6.3.7 FR: Movement

Movement in the virtual environment is one of the time critical functions in the game. Ideally movement should be mirrored perfectly on all clients, so that the players have a consistent view of the virtual world. In practice this means that the map position of player A should be identical on client **A** and client **B**, and that the speed and angle of movement also should be identical. However when considering the information flow between client-server/host-client, some propagation delay is always associated, and will to some extent influence the human perception of the game. Therefore some lag between the players position from client to client must be tolerated.

To compensate for this delay, clients render the screen at a higher pace than the sending interval of *Movement (pid, x, y, z, speed, position)* messages in Figure 38. As a consequence of this, the position of a player at a given moment in time may vary from client to server/host, but since the server doesn't need to display graphics it is considered adequate to monitor the position in the virtual environment.

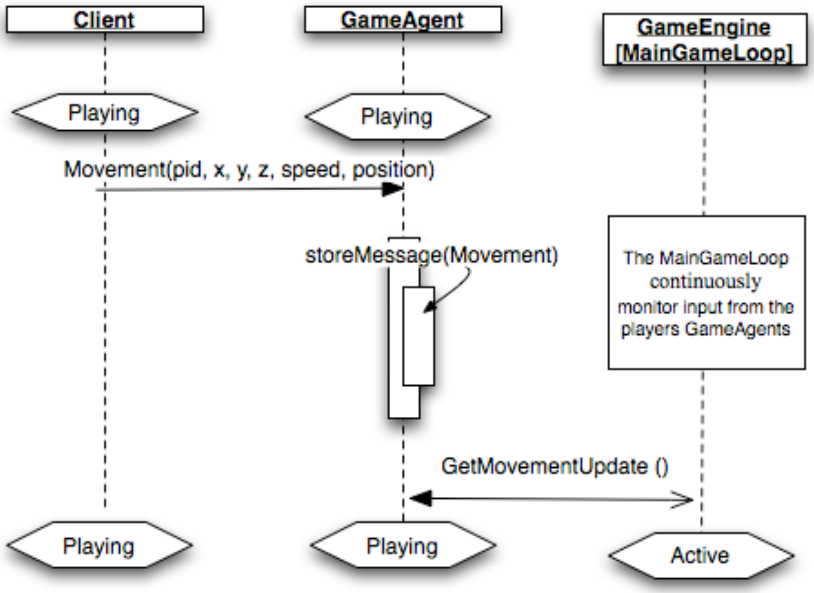


Figure 39

However it is not sufficient for the server alone to know the position of all players. The clients need to know the movement of other players as well (only player within the avatars visual range) in order to display them on the screen. To keep the traffic load between the server and the clients as low as possible, only primitive parameters that can be used to manipulate the media objects stored in the file-system of the clients are sent (also included is the compressed of the other players that are within the communication edge of the avatar[13]).

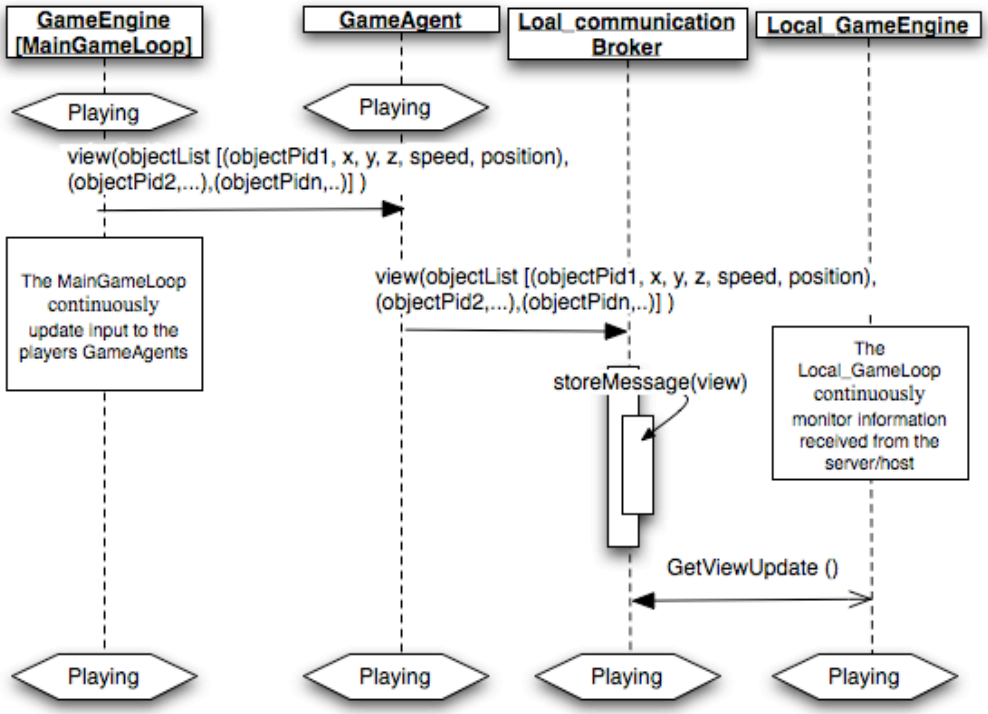


Figure 40[14]

[13] See 5.2 Interaction in Avatars-Online for explanation.
[14] Local_communication Broker is identical to a LocalGameAgent

If the clients rendered the screen only when the *GetViewUpdate ()* message reply was received, the movement of the other players could appear blurry and fragmented. To avoid this effect, a client renders every frame with the parameters currently in the *objectArray* (the *objectArray* different is where the reference and parameters to the objects, which is to be displayed on the screen is stored) once for every iteration of the *LocalGameLoop*. In theory this should result in a slight degree of inconsistency between the positions in the virtual environment, which the server and the clients have registered.

To compensate for this inconsistency the exact position of the object is included in the message *view*, the server or host uses the position parameter to correct small inconsistency between the server/host and the clients. On the client side, inconsistency between the calculated position and received is compensated by increasing the objects movement speed to the correct position (correct position is always the position received in the latest reply of the *GetViewUpdate* message). Using this approach prevents the objects movement from appearing detached.

6.3.8 FR: Change view

The change-view process is solely executed on the client and does not involve any communication with server, thus a MSC for this functional requirement is not included.

6.3.9 FR: Communication

- Outgoing communication should be included in the *movement* message in Figure 39[15].
- Incoming communication should be included in the *view* message in Figure 40.

6.3.9 FR: Fire weapon

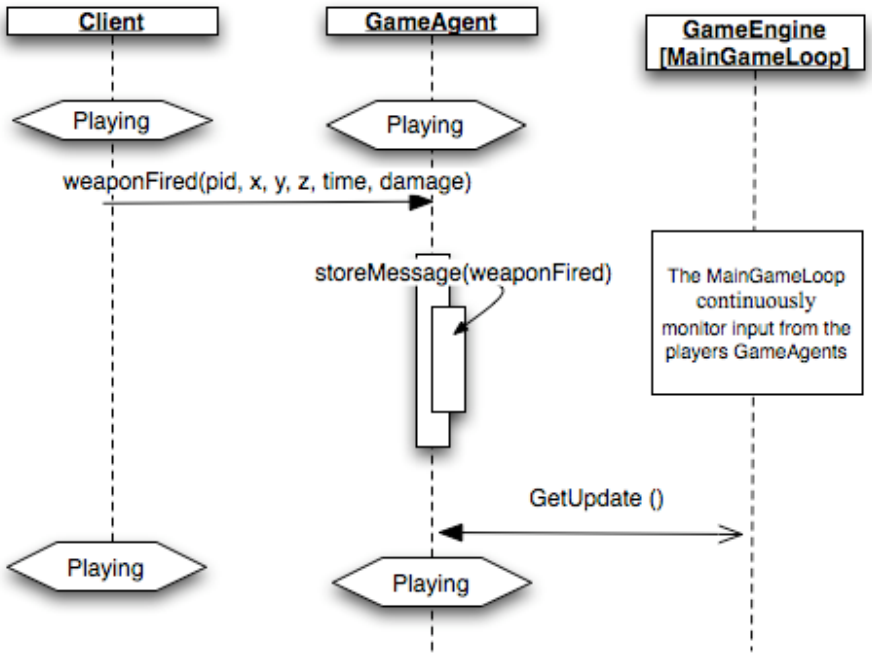


Figure 41

[15] In times when an avatar is not moving, both the client and the server will send the view and movement message, since the network resource is already reserved (connection oriented server-client communication).

The *weaponFired* event and its parameters are stored in the *GameAgent* object, which is situated on the server or host. Each player has one *GameAgent* object that represents him or her on the server. In *Figure 41* above the game agent stores the message received from the client in an *eventArray*, the main *GameLoop* regularly (once every iteration of the main *GameLoop*) access each of the players *GameAgents* to monitor movement and battle parameters in the game.

Since most of the calculations are done in the periphery nodes (the clients) the *weaponFired* event is generated. The *GetUpdate* message replies are handled/processed in a round-robin fashion. Since this process is repeated so frequently (several times pr second) that it is impossible for human perception to register the time between each iteration, the game will not concern itself with sorting events successively with respect to the exact time they where generated. Events recorded within one iteration of the main *GameLoop* are treated as events occurring at the exact same time.

Firing a weapon can be dangerous in the real world, and this applies to the virtual world in Avatars-online as well. Therefore it is possible for avatars to die from wounds caused by firearms. In Figure 43 a description of how Avatars-Online design handles damage done to avatars is provided in the form of a MSC.

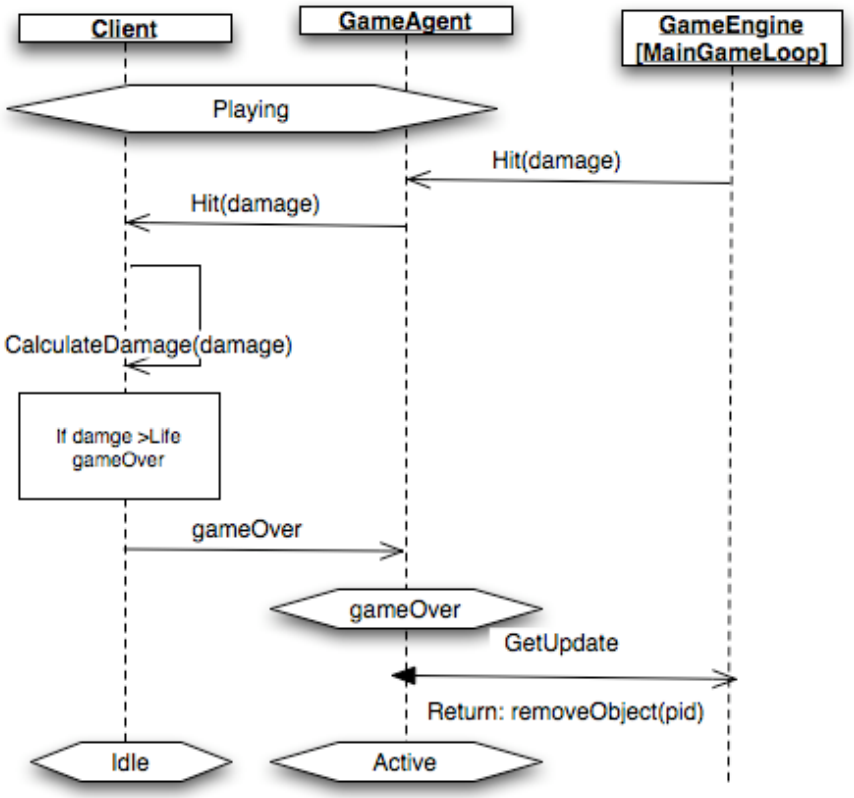


Figure 42

To ease the workload of the server, the design seeks to leave most of the workload to the clients. The message *Hit (damage)* is checked first for every iteration of the local *GameLoop*. In practice this means that once it is received the local *GameLoop* calculates the damage, if the damage doesn't exceed current life the local *GameLoop* continuous at the point it was interrupted, else the local *GameLoop* issues the message *gameOver* to the server/host before terminating letting the player know the game is over.

Once the server or host register that a player has been “killed” it removes the *GameAgent* representing the player is removed from the list of active *GameAgents* and transferred to the list of idle *GameAgents*. The *MainGameLoop* only occasionally accesses the list of idle *GameAgents*.

6.3.10 FR: Ignore avatar

It has been considered that the best way to implement this feature is to keep the list of ignored avatars contained in the avatar object. Thus the MSC for adding a avatar to the ignore list is best described by the MSC in 6.3.5 FR: Configure avatar.

6.3.11 FR: Leave Game

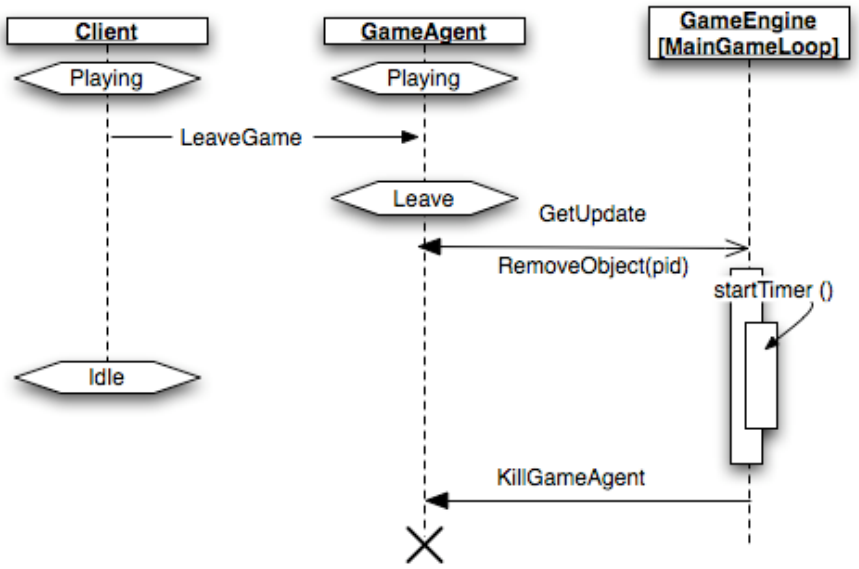


Figure 43

The *Leave_Game* procedure is almost identical to the *Remove_Object* procedure, the only real difference is that the player initiates the Leave_Game procedure. The *LeaveGame* message is in very similar to the *gameOver* message in *Figure 42*, except that it leaves the *GameAgent* in state *Leave*. Once the main *GameEngin*e registers that the *GameAgent* is in state *Leave* it initiates a timer, which eventually will remove the *GameAgent* object.

---

7 IMPLEMENTATION

Implement, test and demonstrate parts of the game on a single PC with a graphics card (e.g., a NVIDIA GeForce card and NVSG development kit)

7.1 Architecture

In this thesis Java was chosen on the basis that the author are familiar with the language and solely on that basis. In a complete realization of Avatars-Online, the application should be implemented in a combination of C/C++ and assembler, instead of Java, due to its efficiency in speed (a decision based on the conclusion of [48]). The reason for implementing the demo in Java was not based on the fact that the author was proficient in this language and had no experience with either C/C++ or assembler. The software modules used in the demo is illustrated in Figure 44 below.

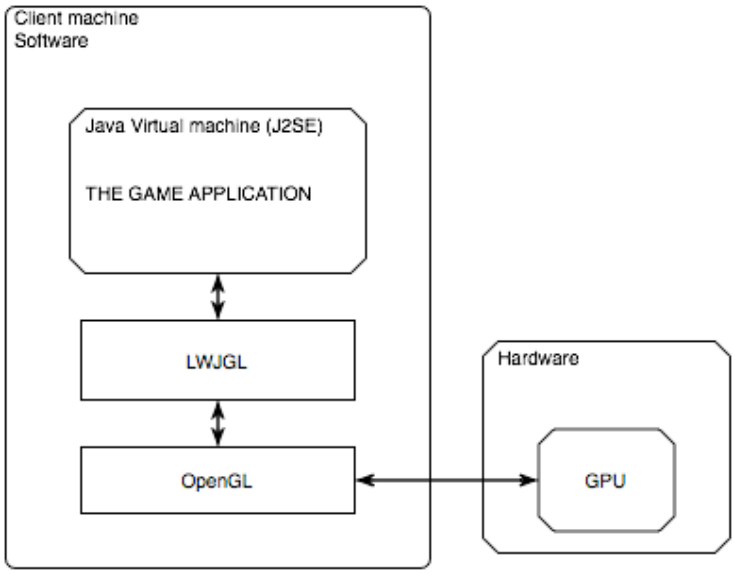


Figure 44

In *Figure 44* the different building blocks of the game design is described conceptually. The core of the game is the Java Virtual Machine (JVM) where all processes are executed. The local JVM is granted access to the graphical processing unit (GPU) through the LWJGL library, which allows the JVM to utilize the power of the GPU. Both the JVM and the OpenGL are allowed to access media files stored on the local hard-drive.

There are several other libraries which the JVM utilizes in the realization of the game, but the essence of the design is how the JVM is can utilize the GPU through the use of LWJGL and OpenGL. However the reason for choosing OpenGL instead of DirectX is based on the report "3D rendering in Java" that concluded that OpenGL performs better than DirectX. [48]

7.2 UML Class diagram

The software classes used in the demo and their relationship is described in Figure 45.

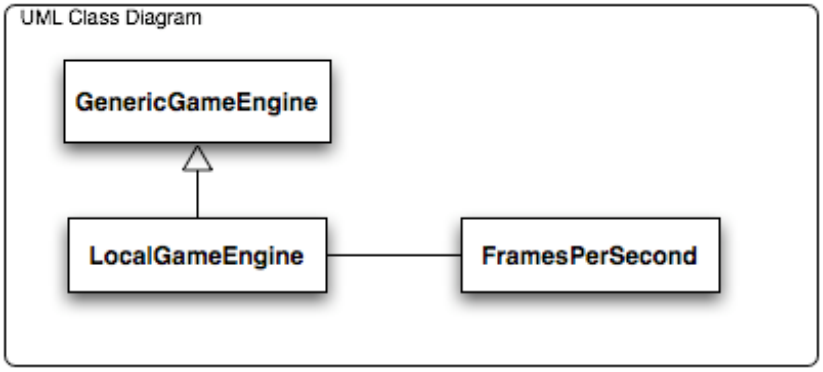


Figure 45

7.3 Screenshots

The program creates a cube and rotates it in 3D space. In Image 16 and Image 17 the screenshots from the demo pictures the cube from different angles in the 3D space.

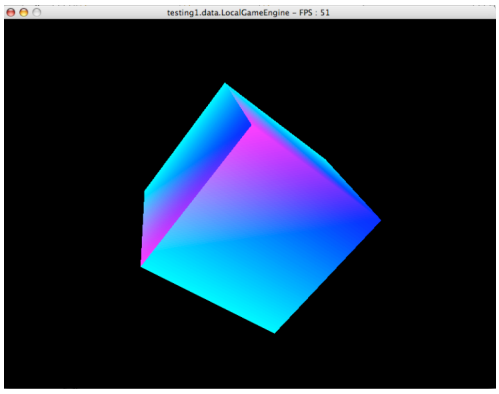


Image 17

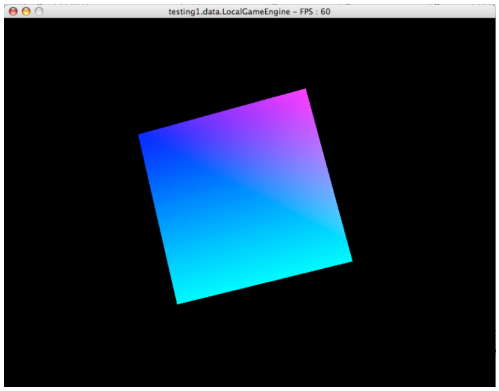


Image 16

7.3 Source code

The source code of the program can be found in the Appendix. In order to successfully run the demo on a computer the LWJGL framework has to be installed. The LWJGL framework works on the following platforms; Windows, Mac OSX and Linux.

At the LWJGL homepage (http://www.lwjgl.org/installation.php), both the framework and a installation manual can be found.

8 Discussion

8.1 Reflection

After carrying out the tasks presented, some reflection around the choices that where made is necessary in order discover where the “mistakes” were made, and why they were made, in order to gain knowledge from them and thus avoid making the same “mistakes ” in the future. In the next chapter a quick evaluation of the four parts (introduction to graphics and video, specification, design and implementation) is presented.

8.1.1 Introduction to graphics and video

In the beginning of this report, the author had little to no experience with in either the area of graphics or the area of computer graphics. However the vastness of the two areas soon became apparent, and with it the realization that either the scope of the task or the level of detail had to be reduced. The decision ended on a compromise between the two paths (broadness or detail). Subjects relating to the areas described in Figure 46 where given more attention than the other areas. This Decision was based on the idea that this area would be interesting in the context of quick production of quality 3D models. The target in Figure 46 is pictured to be a piece of hardware equipment that could be used to create close to reality 3D models that could be used to shorten the time spent to development stunning graphics. Looking back on the choices that where made, the impression is that the compromise was the right thing to do.

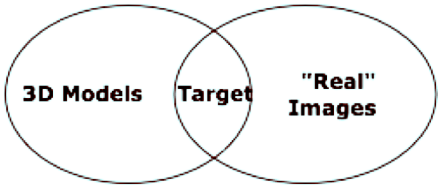


Figure 46

8.1.2 Specification

Specification is all about the good ideas, and it soon became apparent that all the good ideas “I” developed, someone else had developed before me or the ideas turned out to be everything else than good. However the task in the assignment (Propose and specify a distributed game using graphics and video) was not to specific to kill all creativity, and not to wide to lead creativity on a directionless path.

The idea for the game Avatars-Online is inspired by similar games like, World of Warcraft and AnarchyOnline. However the idea of incorporating a way for player to interact with other players in the most natural way (using Mpeg4 compressed speech and appending it onto graphical objects using 3D-audio), I still believe is quit original (until I discover that someone else has created a similar feature). With out the interaction feature Avatars-Online would be a reproduction of most other massive online multiplayer games.

8.1.3 Design

Due to the scope of the task the design had to be incomplete, the question was to find the correct tasks to focus on. The focus was directed to the client and specifically the *LocalGameEngine* in order to clarify the common structure for all real-time based computer games. Looking back on the design phase the focus could also have been directed towards process concerning player interaction and then specifically how to achieve the combine audio with the graphical Objects. Considering that the interaction process is original (as far as I know), I believe that it positively could have been the major focus of the design process.

8.1.4 Implementation

Originally the idea was to add texture onto the cube (7 IMPLEMENTATION) and move it around in 3D-space with the use of the keyboard or the mouse, unfortunately I did not succeed in doing so. However the demo provides a good example on how a real-time game application could be designed and has contributed to increase the author knowledge on OpenGL programming.

8.2 Convergence

Continues development within computer graphics and the digitalization of the movie industry, has resulted in closer relationship between the two industries. There is no longer a clear border between computer graphics and the movie industry. The movie industry now extensively uses computer graphics as a tool to create spectacular scenes, and some movies are made using only computer graphics (Toy Story). Likewise movie clips are used to add reality to computer games. Thus computer graphics and the motion picture industry can no longer be seen as two distinct branches. A fair statement would be that there is an ongoing convergence process between the computer industry and the motion picture industry.

It is in my opinion likely that the increased demand for lifelike graphics in the computer industry will result in new ways to create three-dimensional graphical objects, and that hardware like the Z-cam [39] will play a more significant role in the creation of interactive entertainment.

8.3 Derived ideas

The author is convinced that two ideas presented in this chapter should be further explored in the future

Interactive meeting rooms using graphics and 3D sound
Combination of an online meeting place with the speech feature described in 5.2 Interaction in Avatars-Online, could be killer application.
Imagine the possibility to log onto a German server and practice your German, or to "Walk" into a virtual café and mingle with other people from all around the world

Rapid construction of high quality 3D models
A possible course of action is illustrated in Figure 47. The idea is to use Z-cameras that are able to record the three-dimensional properties of an object. In order to get the necessary information the object has to be shoot from a total of six angles (north, south, west, east, top and bottom). Using heuristics the six images can be synthesized into a complete digital representation of the three-dimensional object. It is important that the distance from the center of the object is equal from all the six angles, so that the scale of the different images is fitted for the synthesizing process. Once a complete digital representation of the object is finished, it can be manipulated by standard GPU operations and methods already available through open-GL. Through the use of this scheme spectacular three-dimensional graphics can be created quickly.

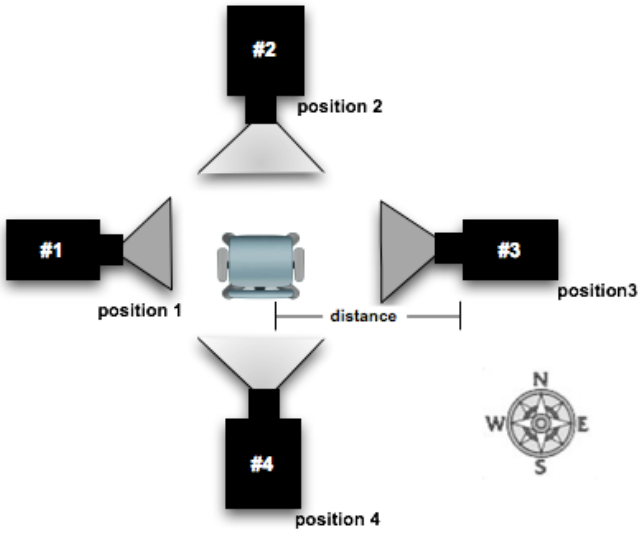
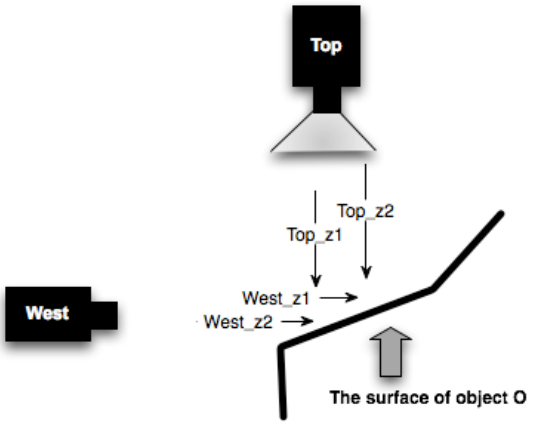


Figure 47

- Many real surfaces have a rather smooth surface (cars, books, furniture and so on) therefore a large number of points in the z-buffer will have the same value. Practically this means that the z-grid is unnecessary tight for a large smooth area, and thus large parts of the TIN can be replaced by single triangles.
- In the synthesize process some redundant information can be deleted in order to reduce the amount of data needed in a digital representation of the object, *Figure 48* illustrates the scheme.
- When the image is captured, the object in most cases does not occupy all the pixels in the image. The idea is to "carve" out the desired object in the image and deleting the rest of the image in order to delete unnecessary information from the image.
- By using Huffman-coding and other smart compression algorithms the data size can be further reduced.



The synthesizing algorithm must be able to identify triangles that cover the same are in each of the two images TINs. Once identified only one of them is used in the final digital representation of the three-dimensional object.

Figure 48

The technique described above could also be utilized effectively in other industries than the interactive gaming industry. Other suggested areas of use could be; as an effective technique to document different aspects in the construction industry and in the manufacturing industry.

9 Conclusion

The writing of this report has greatly enhanced the authors understanding of computer graphics and video technology, and improved his familiarity different technologies within the area of both graphics and video. Furthermore the author has gained valuable experience in writing scientific papers.

All tasks presented in the assignment have been answered, and all the goals in presented in chapter 3 has been achieved. The author is also convinced that someone will find the two derived ideas presented in chapter 8.3 interesting.

10 Bibliography

1 www.dn.no/esa Last visited 15 February 2006

2 World of Warcraft: http://en.wikipedia.org/wiki/World_of_Warcraft#_note-sixmillion. Last visited 31 May 2006.

3 FAQ (official World of Warcraft site): http://wowvault.ign.com/faq/index.php?category=1#1_0_6. Last visited 31 May 2006.

4 U.S. Industry Status and Projected Growth: http://www.coloradofilmschool.net/cgi-bin/disp_help.cgi?subpage=stats. Last visited 29 May 2006.

5 Web-games; http://www.ebaumsworld.com/pacman.html. Last visited 29 May 2006.

6 http://www.doom3.com/

7Alpha Blending: http://www.visionengineer.com/comp/alpha_blending.shtml, Last visited 12 June 2006.

8 Chroma-Key; http://en.wikipedia.org/wiki/Chroma_key. Last visited 12 June 2006

9 Depth Cuing: https://www.mcell.psc.edu/DReAMM/reference/03-Depth%20Cueing.html, Last visited 12 June 2006.

10 Depth Cueing: http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node278.html. Last Visited 13 June 2006.

11 Directron.org: http://www.directron.com/videoglossary.html. Last visited Monday, July 3, 2006

12 Double buffering: http://en.wikipedia.org/wiki/Double_buffering. Last visited 13 June 2006.

13 Real-Time Color-Based Depth Cueing: http://www.vis.uni-stuttgart.de/depthcue/. Last visited 14 June 2006.

14 PNG (Portable Network Graphics) Specification Version 1.0: http://www.w3.org/TR/PNG-GammaAppendix. Last visited 15 June 2006.

15 Gamma Correction in Computer Graphics: http://www.teamten.com/lawrence/graphics/gamma/index.html. Last visited 15 June 2006.

16 Chapter 6 of the book: *A Technical Introduction to Digital Video*, by Charles Poynton, published in 1996 by John Wiley & Sons.

17 Perspective-Correct Interpolation, Kok-Lim Low, Department of Computer Science, University of North Carolina at Chapel Hill, March 12, 2002.

18 Perspective-Correct Interpolation**,** Kok-Lim Low, Department of Computer Science, University of North Carolina at Chapel Hill, March 12, 2002.

19 Perspective Correction: http://www.mediachance.com/pbrush/help/perspectc.html. Last visited 15 June 2006.

20 Perspective Correction: http://www.mediachance.com/pbrush/help/perspectc.html. Last visited 15 June 2006.

21 Rendering (computer graphics): http://en.wikipedia.org/wiki/Rendering_(computer_graphics)#Features. Last visited 18 June 2006.

22 Z-Buffer: http://www.tyan.com/support/html/graphics.html#d. Last visited 19 June 2006.

23 Z-Buffer: http://www.tyan.com/support/html/graphics.html#d. Last visited 19 June 2006.

24 Glossary - Computer Video and Graphics, http://www.directron.com/videoglossary.html. Last visited 29 May 2006.

25 3D digital corp: http://www.3ddigitalcorp.com/home.htm. Last visited 19 June 2006.

26 The Graphics Pipeline: http://www.devhardware.com/c/a/Video-Cards/The-Graphics-Pipeline/1/. Last visited 19 June 2006.

27 View Frustum: http://astronomy.swin.edu.au/~pbourke/stereographics/HET409_2003/frustum.html. Last visited 20 June 2006.

28 Teaching Texture Mapping Visually, http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm. From _1.htm to _10.htm. Last visited 29 May 2006.

30 Fundamentals of Computer graphics, Peter Shirley, School of computing University Utah

31 http://www.graphics.com/modules.php?name=News&file=print&sid=2802

32 Displacement Mapping, Michael Doggett, ATI Research, January 13, 2003. Complete text in PDF is provided on source -CD.

33 Cube mapped reflection: http://en.wikipedia.org/wiki/Cube_mapped_reflection. Last visited 29 May 2006.

34 ATTEST: Advanced Three-dimensional Television System Technologies, André Redert1, Marc Op de Beeck, Christopher Fehn, Wijnand Jsselsteijn, Marc Pollefeys, Luc Van Gool, Eyal Ofek, Ian Sexton, Philip Surman, The complete report is included in the source-CD.

35 Digital Video: An introduction to mpeg-2, Barry G.Haskell, Atul Puri and Arun N. Netravali.

36 IVAR ’94 Tutorial, Image Acquisition and Display, by PatrickWambacq. Complete text in PDF is provided on source -CD.

37 Raster Scan:http://computing-dictionary.thefreedictionary.com/raster+scan. Last visited June 8, 2006.

38 Technology CCD: http://www.answers.com/topic/ccd. Last visited June 10 June 2006

39 3DV Systems: http://www.3dvsystems.com/products/zcam.html. Last visited June 10 June 2006

40 3D Displays: http://www.3dcgi.com/cooltech/displays/displays.htm. Last visited June 10 June 2006

41 Overview of the MPEG-4 standard: http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm. Last visited 29 May 2006.

42 Official online OpenGL overview: http://www.opengl.org/about/overview/#8. Last visited 23 June 2006.

43 http://lwjgl.org/wiki/doku.php/lwjgl/tutorials/opengl/index

44 Official online OpenGL overview: http://www.opengl.org/about/overview/#8. Last visited 23 June 2006.

45 http://lwjgl.org/wiki/doku.php/lwjgl/tutorials/opengl/index

46 The Official Guide to Learning OpenGL, Version 1.1, OpenGL Programming Guide (Addison-Wesley Publishing Company), Second Edition. Complete guide is included in the source CD.

47 Lightweight Java Game, Libraryhttp://en.wikipedia.org/wiki/Light_Weight_Java_Game_Library. Last visited 29 May 2006.

48 3D-rendering in Java, report from university of Roskilde. The complete report is included in the source-CD.

11 Appendix

```
package demo.data;

 import testing1.utils.GenericGameEngine;
 import testing1.utils.FramesPerSecond;
 import org.lwjgl.opengl.GL11;
 import org.lwjgl.input.Keyboard;
 import org.lwjgl.opengl.glu.GLU;
 import org.lwjgl.opengl.Display;

 public class LocalGameEngine extends GenericGameEngine {

   private static float rotX = 0.0f;
   private static float rotZ = 0.0f;
   private static float rotY = 0.0f;

   public LocalGameEngine() {
     super();
   }
 //---------------------------------------------------------------------------

 protected void render() {
         Display.setTitle(this.getClass().getName() + " - FPS : " +
         FramesPerSecond.getFps());

         GL11.glClear(GL11.GL_COLOR_BUFFER_BIT |
         GL11.GL_DEPTH_BUFFER_BIT);              //clears color and depth buffer
         GL11.glLoadIdentity();

         GLU.gluLookAt(0, 0, 6,    0, 0, 0,    0, 1, 0);   // This determines the viewpoint
                                                           // in the 3D virtual space

         GL11.glRotatef(rotX, 1.0f, 0.0f, 0.0f);
         GL11.glRotatef(rotY, 0.0f, 1.0f, 0.0f);
          GL11.glRotatef(rotZ, 0.0f, 0.0f, 1.0f);

         CreateCube(-1, -1, -1, 2);

         rotX += 0.3f;                          //  rotation speed along the X axis
         rotY += 0.3f;                          //  rotation speed along the Y axis
         rotZ += 0.8f;                          //   rotation speed along the X axis

    GL11.glMatrixMode(GL11.GL_PROJECTION);    // Select The Projection Matrix
    GL11.glPushMatrix();                       // Store The Projection Matrix
    GL11.glLoadIdentity();                     // Reset The Projection Matrix
    GL11.glOrtho(0, 800, 0, 600, -1, 1);
    GL11.glMatrixMode(GL11.GL_MODELVIEW); // Select The Modelview Matrix
    GL11.glPushMatrix();                       // Store The Modelview Matrix
```

```
    GL11.glLoadIdentity();                                  // Reset The Modelview Matrix



          GL11.glColor3f(1.0f, 0.0f, 0.0f);
          GL11.glMatrixMode(GL11.GL_PROJECTION);    // Select The Projection
                                                                  // matrix
          GL11.glPopMatrix();                              // Restore The Old Projection Matrix
          GL11.glMatrixMode(GL11.GL_MODELVIEW);     // Select The Modelview
                                                                  // Matrix
          GL11.glPopMatrix();                  // Restore The Old Projection Matrix
  }

   protected void gamelogic() {
     if (Keyboard.isKeyDown(Keyboard.KEY_ESCAPE)) {
       // end the "game"/DEMEO
       exitApplication = true;
     }
   }

   protected void initGL() {
     super.initGL();

     GL11.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);                // Black Background
     GL11.glEnable(GL11.GL_DEPTH_TEST);                  // Enable depth testing

     GL11.glMatrixMode(GL11.GL_PROJECTION);              // Select The Projection
                                                                  // Matrix
     GL11.glLoadIdentity();                  // Reset The Projection Matrix

     //set the preffered screen mode parameters
     GLU.gluPerspective(45.0f, (float)SCREEN_LENGTH_Y /
(float)SCREEN_LENGTH_X, 1.0f, 300.0f);

     GL11.glMatrixMode(GL11.GL_MODELVIEW);  // Select The Modelview Matrix

   }

   private void CreateCube(float x, float y, float z, int radius) {
      // The following part creates a 3D-cube (quad).
      // the commands used in this part is identical to the commands
      //used in "pure2 OpenGL. The six sides of the cube are created separately.

     GL11.glBegin(GL11.GL_QUADS);

        // These vertices create the Back Side
      GL11.glColor3f(0, 0, 1); GL11.glVertex3f(x, y, z);
      GL11.glColor3f(1, 0, 1); GL11.glVertex3f(x, y + radius, z);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y + radius, z);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y, z);
```

```
        // These vertices create the Front Side
      GL11.glColor3f(0, 0, 1); GL11.glVertex3f(x, y, z + radius);
      GL11.glColor3f(1, 0, 1); GL11.glVertex3f(x, y + radius, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y + radius, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y, z + radius);

        // These vertices create the Bottom Face
      GL11.glColor3f(0, 0, 1); GL11.glVertex3f(x, y, z);
      GL11.glColor3f(1, 0, 1); GL11.glVertex3f(x, y, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y, z);

        // These vertices create the Top Face
      GL11.glColor3f(0, 0, 1); GL11.glVertex3f(x, y + radius, z);
      GL11.glColor3f(1, 0, 1); GL11.glVertex3f(x, y + radius, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y + radius, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y + radius, z);

        // These vertices create the Left Face
      GL11.glColor3f(0, 0, 1); GL11.glVertex3f(x, y, z);
      GL11.glColor3f(1, 0, 1); GL11.glVertex3f(x, y, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x, y + radius, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x, y + radius, z);

        // These vertices create the Right Face
      GL11.glColor3f(0, 0, 1); GL11.glVertex3f(x + radius, y, z);
      GL11.glColor3f(1, 0, 1); GL11.glVertex3f(x + radius, y, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y + radius, z + radius);
      GL11.glColor3f(0, 1, 1); GL11.glVertex3f(x + radius, y + radius, z);

    GL11.glEnd();
  }

  public static void main(String[] args) {
                LocalGameEngine avatarGame = new LocalGameEngine();
             avatarGame.init(800, 600);
             avatarGame.localGameLoop();
            }

}
```

```
package demo.utils;

 import org.lwjgl.opengl.*;
 import org.lwjgl.input.Keyboard;
 import org.lwjgl.input.Mouse;
 import org.lwjgl.opengl.Display;
 import org.lwjgl.opengl.DisplayMode;
 import org.lwjgl.opengl.GL11;
 import org.lwjgl.opengl.glu.GLU;

 public abstract class GenericGameEngine {

   protected static final int SCREEN_COLOUR_DEPTH = 16;
   protected static final float VIEW_DISTANCE = 3500;

   protected boolean exitApplication = false;
   protected long timerRes = 0;

   protected int fps = 0;

   protected int SCREEN_LENGTH_X;
   protected int SCREEN_LENGTH_Y;

   protected int mouseX;
   protected int mouseY;
//_____________________________________________________________________
   // STANDARD LWJGL METHOD USED TO CREATE THE DISPLAY SCREEN

   public void init(int pixelWidth, int pixelHeight) {
            SCREEN_LENGTH_X = pixelHeight;
            SCREEN_LENGTH_Y  = pixelWidth;

     try {

          DisplayMode[] modes = Display.getAvailableDisplayModes();
          if (modes.length == 0) {
                   throw new Exception("Cannot find any available display modes.");
           }
          DisplayMode chosenDisplay = modes[0];    // Default to the first screen mode
          for (int i=0; i<modes.length; i++) {
          // Try and find a screen mode that fits the preffered  screensize.
               if ((modes[i].getWidth() == SCREEN_LENGTH_X) &&
               (modes[i].getHeight() == SCREEN_LENGTH_Y) &&
              (modes[i].getBitsPerPixel() == SCREEN_COLOUR_DEPTH)){
                     chosenDisplay = modes[i];
                      break;
                    }
```

```
        }


        Display.setDisplayMode(chosenDisplay);
        Display.setFullscreen(false);
        Display.setVSyncEnabled(true);
        Display.setTitle(this.getClass().getName());
        Display.create(new PixelFormat(0, 8, 0));

        Mouse.create();
        Keyboard.create();

        mouseX = SCREEN_LENGTH_X / 2;
        mouseY = SCREEN_LENGTH_Y / 2;

        initGL();
  }
  catch (Exception e) {
        System.out.println("Failed to initialise : " + e);
        }
}

//_____________________________________________________________________

protected void resizeGLScene(int width, int height) {

        GL11.glViewport(0, 0, width, height)
        GL11.glMatrixMode(GL11.GL_PROJECTION);
        GL11.glLoadIdentity();

         GLU.gluPerspective(45.0f, ((float) width) / ((float) height), 1.0f,
        VIEW_DISTANCE);

         GL11.glMatrixMode(GL11.GL_MODELVIEW);

        GL11.glLoadIdentity();
        }

protected void initGL() {
            resizeGLScene(SCREEN_LENGTH_X, SCREEN_LENGTH_Y);

            GL11.glShadeModel(GL11.GL_SMOOTH);

            //Black Background
            GL11.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);


            GL11.glClearDepth(1.0f);

            GL11.glEnable(GL11.GL_DEPTH_TEST);
```

```
            GL11.glDepthFunc(GL11.GL_LEQUAL);

            GL11.glHint(GL11.GL_PERSPECTIVE_CORRECTION_HINT,
            GL11.GL_NICEST);
          }

  // --------------------------------------------------------------------------
 // ONCE THE LOCALGAMELOOP HAS BEEN STARTED IT WILL RUN
//UNTILL "THE PLAYER" PRESSES
// THE ESCAPE BUTTON OR ENDS THE APPLICATION
  public void localGameLoop() {
                try {
                        while (!exitApplication) {
                                if (!Display.isVisible()) {
                                        Thread.sleep(200);
                                } else if (Display.isCloseRequested()) {
                                        exitApplication = true;
                                } else {
                                        gamelogic();
                                        render();
                                }
                                Display.update();
                        }
                } catch (Throwable t) {
                        t.printStackTrace();
                } finally {
                        destroy();
                }
  }

  protected abstract void gamelogic();

  protected abstract void render();

  private void destroy() {
    Keyboard.destroy();
    Mouse.destroy();
    Display.destroy();
  }

 }
```

```
package demo.utils;

 import org.lwjgl.Sys;

 public class FramesPerSecond  {

          // Hold the time from the last frame
         private static long lastTime = 0;

          // Old fps number
         private static int old_fps = 0;

          // Frame interval.
         public static float frameInterval = 0;

          //Last frame's time.
         private static long frameTime = 0;

          // Fps = number of frames pr second
         private static int fps = 0;


         // Calculates the frame rate (the number of times the frame is rendered pr
second).
         public static int getFps() {
                  long currentTime = Sys.getTime();

                  frameInterval = ((float)(currentTime - frameTime)) / 1000;
                  frameTime = currentTime;
                  fps++;

                  if(currentTime - lastTime > Sys.getTimerResolution()) {
                           lastTime = currentTime;
                           old_fps = fps;
                           fps = 0;
                           return fps;
                  }
                  else {
                           return old_fps;
                  }
         }
 }
```