Scalable Model Viewing

Technical Report No. CS03-22-00

Nicholas Appleby
Department of Computer Science
University of Cape Town
South Africa
nappleby@cs.uct.ac.za

Rory Marcussen
Department of Computer Science
University of Cape Town
South Africa
rmarcuss@cs.uct.ac.za

James Mc Millan
Department of Computer Science
University of Cape Town
South Africa
jmcmilla@cs.uct.ac.za

Patrick Marais
Department of Computer Science
University of Cape Town
South Africa
patrick@cs.uct.ac.za

ABSTRACT

In this paper we describe a novel approach to displaying complex 3D scenes on architecture that does not have support for 3D graphical display.

Our system makes use of distributed graphics, employing a standard workstation as a server. This server communicates with a client via UDP sockets over a LAN link. To minimize network traffic, a compression engine is employed to compress and decompress data on either side of the network transfer phase.

By dividing the display into blocks, we effectively reduce the amount of data needing to be transported from client to server and provide an interactive user friendly client.

We choose a PDA as a client, but our approach is extensible and easy to implement on any architecture with at least some 2D graphics drawing capabilities.

Categories and Subject Descriptors

I.3.2 [**Computer Graphics**]: Distributed/Network Graphics.

General Terms

Performance, Design, Human Factors.

Keywords

Compression, Usability, Distributed Graphics, OpenGL.

1. INTRODUCTION

The popularity of 3D graphics has grown considerably in recent years and hardware advances have been frequent. This, however, means that the lifespan of graphics hardware has been shortened considerably. We propose a system that allows complex 3D scenes to be rendered on displays that have only 2D display capabilities.

Our system addresses this problem by employing distributed graphics. As such consists of three main components. These are:

- A server, which is a standard workstation with hardware 3D graphics capabilities.

-A client with only 2D drawing capabilities.

-A compression engine.

We have chosen a PDA as our client as mobility of these devices has made them increasingly popular in recent times.

Our system distributes the rendering of the scenes to the server, which sends the data to the client via a UDP socket over a LAN link. This data is compressed before it is sent to minimize network usage.

The rendered scene is divided into blocks in so that only blocks which change from one frame to the next need be transmitted and thus redrawn.

The client provides a usable interface to the user and sends the user interactions back to the server, which then updates the scene and transmits the next frame.

2. BACKGROUND AND MOTIVATION

2.1 Compression

Two primary restrictions are placed on the system: the bandwidth that is available for the client/server communications is relatively low, and the client is assumed to have low-end hardware and thus has limited resources. This could lead to jittery movement or something that is just not usable. In order to alleviate this problem, the amount of network traffic had to be minimized. Compression is used as the primary means of alleviating this problem.

If a pixel depth of 16bpp is assumed on the PDA, then at the full screen resolution of the PDA one frame will take up 150k. In order for the user to effectively interact with the system, the display must update at least 10 fps (frames per second) [TM]. Updating at this frame-rate results in 2.19MB being transmitted per second. However, we are not likely to use the full PDA display for the 3D scene, since some of the display may be used for displaying user interface components.

Formally, compression reduces the data transmission, processing and storage requirements of a system by reducing the number of bits needed for signal representation. There are two ways in which to accomplish this, **lossless** and **lossy** compression.

Lossless compression chooses an alternative representation of the data which equals the original signal's information content, so no information is lost, and redundancies are removed from the data. Lossy compression, on the other hand, considers the use to which the data will be put and selectively eliminates unneeded or unimportant information. Most compression techniques which are lossless and use statistical data only give a compression ratio from 2:1 to 5:1 on images. A lossy compression scheme such as Jpeg can, however, give compression ratios of 20:1 with minimal image quality degradation [4]. Lossy video compression, which can also exploit temporal coherence, can achieve even higher compression ratios [4].

Having looked at the most important concepts in compression, the key compression techniques used for the compression engine will now be explained.

2.1.1 Lossless entropy encoders

There are many variants of entropy encoders, where the underlying principle is to exploit the statistics of the data to perform compression. Huffman coding [7] is an example of a statistical compression technique, which works by assigning variable-length codes to symbols based on their frequency. By assigning shorter codes to more frequently occurring signals, the average number of bits per symbol is reduced. Substitutional, or dictionary-based compressors such as Lempel-Ziv [14, 15], replace an occurrence of a particular sequence of symbols in a piece of data with a reference to a previous occurrence of that sequence.

2.1.2 Run-length encoding

There are many variants of run-length encoding, which is lossless, but the central idea is to identify strings of adjacent sequences of the same symbol and replace them with a single occurrence along with a count. Once transformed, an entropy encoder, such as Huffman, can be used to code both the symbol values and the counts. This is often done since short lengths are likely to be much more common than long lengths.

2.1.3 Jpeg Compression

Jpeg compression was developed by developed by the Joint Photographic Experts Group (JPEG), part of the International Organization for Standardization (ISO). It is widely accepted as the standard means of image compression.

An overview of the steps for the algorithm is presented in figure x below.

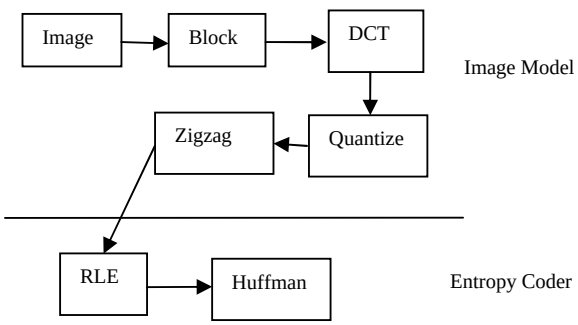

Figure 1. Jpeg compression algorithm

The first step in jpeg compression is the transform step, in which the image is processed in blocks of 8x8 samples. Each block is transformed into a block of 8x8 spatial frequency coefficients by means of a Discrete Cosine Transform (DCT). These coefficients represent weights for each of 64 basis functions (see figure x). These basis functions are effectively patterns that can appear in one block of an image, ranging from lower frequencies in the top left-hand corner to higher frequencies in the lower right-hand corner.

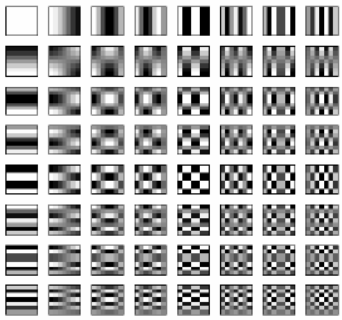

Figure 2. The DCT coefficients

After the DCT step has been done, there tends to be a concentration of a few significant coefficients and the other coefficients are insignificant or close to zero. The motivation for doing the DCT step is that you can now throw away high-frequency information without affecting low-frequency information, since the human visual system is less sensitive to high-frequency information than it is to low frequency information.

The next step is quantization, in which unimportant information (the close to zero coefficients) is discarded. The resulting data is then further compressed using a number of methods, including run-length-encoding and Huffman compression.

Jpeg compresses natural scenes very well [4], but line drawings or images with sharp edges will have noticeable blurring along these edges. Computer generated images often fall somewhere in-between natural and line drawings, depending on how realistically the scene is rendered.

Jpeg compression is useful for this project since it is an extremely efficient image compression technique. However, there can be a large overhead associated with jpeg compression, mostly in the quantization tables which are stored along with the compressed data.

2.1.4. Differential PCM

With DPCM, each pixel is used as a prediction of the next. When doing this the only information that needs to be transmitted is the difference between one pixel and the next. There is no advantage if the number of bits per pixel in the original sample is the same as the number of bits per pixel in the differences we transmit. One possible method for performing DPCM is to assume a maximum change and use that. If a change is encountered that is greater than this maximum, then the actual change is kept internally and modify subsequent differences in order to "catch up" with the actual pixel values. This would result in lossy compression.

DPCM is useful in this project because it is extremely efficient and has minimal overhead associated with it. Extensions to DPCM exist, such as Adaptive DPCM, but these are not as efficient and the compression gain is not significant.

2.1.5. Video Compression

A video is essentially a series of successive images, often with some degree of temporal redundancy.

M-Jpeg compression is a simple video compression technique which ignores this temporal redundancy. Individual frames are compressed using Jpeg (intra-frame compression) and no inter-frame compression is done. Mpeg compression, on the other hand, uses sophisticated inter-frame compression techniques.

Since we do not have advanced information about future frames, Mpeg compression is unsuitable for this project. Video compression is also often very computationally expensive, especially in the case of Mpeg in which the searches for temporal coherence are very expensive.

2.2 Distributed Graphics

Distributed graphics refers to the process of rendering large data-sets or scientific simulations remotely, usually on large clusters of high-powered machines, but visualising the results on local displays [2]. Two types of distributed graphics systems exist [10]:

- A single logical graphics system with distributed components
- Multiple distributed logical graphics systems

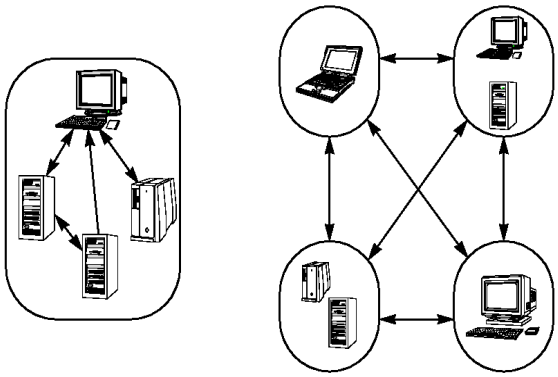

Figure 3. Examples of Single (*left*) and Multiple (*right*) Logical Distributed Graphics Systems

Our approach makes use of the first definition, i.e. we propose one single logical system. Also, our rendering is done on a single workstation, and not on a cluster of workstations or servers.

2.3 HCI Aspects

HCI is the study of all aspects of interaction between users and computers. It focuses on what the user aims to achieve (their *task*) and how difficult it is to achieve this. The aim, according to HCI principles is to allow the user to complete their task without having to focus on the interface. The computer is said to be seen as a tool, and the user's mental focus should naturally be on using the tool, and not on how to use the tool [11].

An important aspect of HCI is usability. Usability focuses on delivering a product that is 'usable'. What this means is that someone can benefit from the product without first having to undergo some sort of training.

Another important notion from the field of HCI is that of an affordance. An affordance is some piece of information that an object imparts upon a user in a subliminal way. This comes from compatibility between the user's perception of the object, and the objects action [6].

An example of an affordance is a large flat metal panel on a door situated where one would normally find a handle. This panel gives the affordance that the door should be pushed to be opened. Conversely, a large bar handle extruded from the door surface, and perhaps even rounded, gives the affordance that it should be gripped and pulled in order to open the door.

With the notion of affordances, comes that of false affordances [6]. This is when an object has an affordance which leads the user to incorrectly perceive how it should be manipulated. An example of a false affordance is a door with a large handle that opens by being pushed.

Affordances should be used by interface designers wherever possible to make the design more intuitive and easier to learn. They can help to reduce what is called cognitive overhead, i.e. the additional effort required to concentrate on several tasks at one time [3]. False affordances should be avoided as they contribute to cognitive overhead, which can build up rapidly resulting in a user feeling "lost".

3. METHOD

3.1 Server

The Server is broken down into two distinct parts, the Renderer and the Front-End Server. These two parts and their basic interaction are shown in Figure .

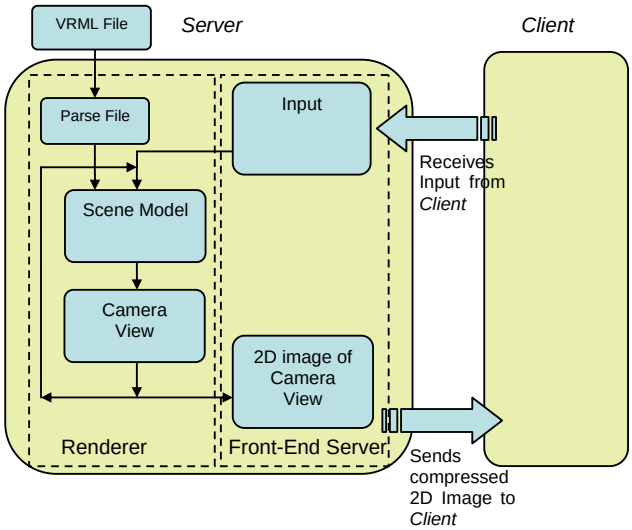

Figure 4. Overview of the Server

3.1.1 The Renderer

Figure shows the Overview of the Renderer in greater detail than previously shown.

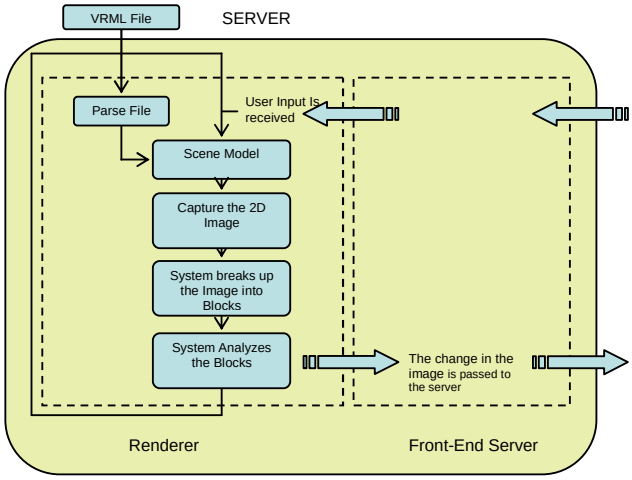

Figure 5. Overview of the Renderer

The initial task of the Renderer is to decide on the 3D environment that the end-user will navigate through. A virtual world is created by using VRML files to specify 3D objects and OpenGL is used to render them to a display. OpenGL renders these VRML model objects using its internal methods and functions to shade and add textures to the 3D models. A scene graph is created when the VRML files are loaded into memory to improve the rendering engine's efficiency. The Server must be able to produce these 2D images in real-time, otherwise the images produced will lag behind even before they have been transferred across the network. There will obviously be a limit on the number of polygons in a scene before the engine will start to fail at producing frames at an interactive rate. This limit will be an attribute of the machine the Server is run on. However, as long as the engine can render scenes with about 50 000 polygons in a scene at interactive rates, the system will be able to deal with fairly complex environments.

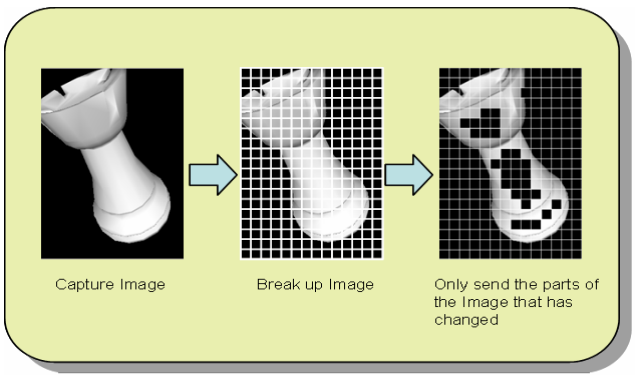

Figure 6. How the Renderer Decomposes the Image

After the 3D environment has been rendered to a 2D buffer, the buffer is decomposed in “blocks”. These blocks are the basic element that is passed across the network to the Client. They are reassembled and displayed on the Client. This is shown in Figure 6.

It is imperative to reduce the amount of information passed between the Server and the Client, and towards this aim only a few selected blocks are sent across the network. The blocks from the current frame are tested against the blocks from the previous frame. If the blocks are equivalent then there is no need to send the frame as the Client has already has that block when it renders the previous frame. If the block from the current frame is different from the previous frame then it is passed on to the Front-End server to be sent client. This is illustrated in Figure 77.

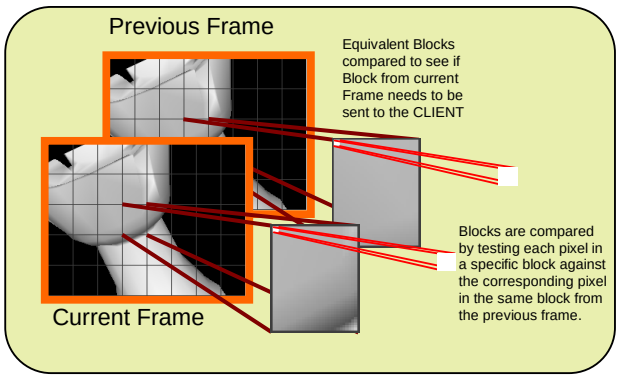

Figure 7. Block Equivalency Test

3.1.2 The Front-End Server

Figure 8 details an overview of the Front-End Server in greater detail than previously shown.

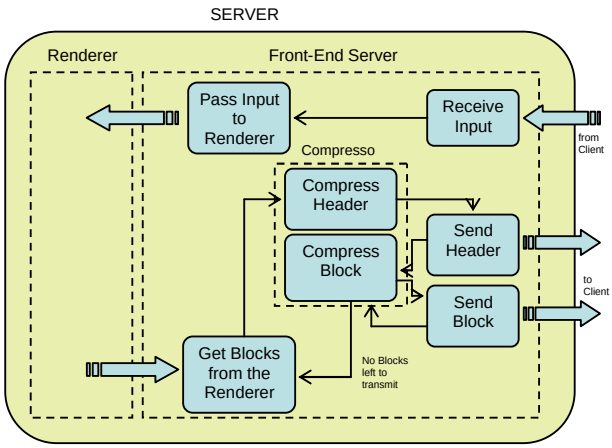

Figure 8. Overview of the Front-End Server

The compressed images are then sent to the Client using the transfer protocol UDP. UDP was decided on instead of TCP/IP as it is faster due to the fact there is less overhead the TCP/IP. However, due to this a protocol was designed to protect the Client from lost of packets and errors in the image it displays.

The first task of the server is to retrieve the blocks of the latest frames from the Renderer. The blocks are then passed to the Compressor.

The compressor first compresses the header. This is created by the Front-End Server and details specifics on how to recreate the image from the blocks. The Header is a vital piece of information and must be passed to the Client, so a protocol is built into the program to ensure that the Header reaches the Client. Once the

header has been created and compressed, it is sent to the Client over the network.

The Front-End Server then sequentially compresses a block from the current frame and sends it to the Client. This is repeated until the Front-End Server has sent the all the blocks designated to be sent. No protocol protects these blocks to ensure that these packets reach the Client. If these packets are not received, the Client will still know that it is missing packets from the information it received in the Header. It will then inform the Server that it is missing the specific block from the Frame and the Server will send that block in the next Frame to be sent. The Front-End Server then retrieves the next sequence of Blocks from the current Frame on the Renderer and starts the sequence again.

The second task of the Front-End Server is to receive user input from the Client. This is done in the Front-End Server by an independent thread as this thread must block while waiting to receive a packet from the Client. This thread listens on another port so in fact there are two connections open between the Client and the Server. The first connection passes all the Frames to the Client while the second is only open to receive input packets from the Client as well as designed packets to give the Server vital feedback. This was done to further separate the two tasks of the Front-End Server and eliminate the need for a more complex protocol between the two.

An independent thread is created so the Server can still run while waiting for the Client's packets as these packets may arrive at random. On receiving these packets the Front-End Server removes the packet header information and passes on the input to the Renderer. Once the thread has passed this information to the Renderer, it goes back to listening for the next input packet.

3.2 Compression

The proposed method for performing compression in this report is a hybrid compression scheme, choosing the best available compression method based on the current circumstances. Since the server first determines which portions of the image have changed, the compression engine receives small blocks of the image of varying sizes. Depending on the size of the block, the most suitable compression method is chosen.

Since Jpeg does not perform well with small image sizes, a lightweight compression algorithm in the form of a variant of DPCM was developed.

When compressing images with DPCM, each colour component of the image is compressed separately. This is because there is no simple means of determining differences in colour which do not favour one colour component over another.

The implementation of the DPCM compressor has several novel features to remove artifacts which arise. The first attempt at a simple DPCM compressor resulted in images with an unacceptably high error, visible in edges that were blurred for long distances. This is because of the inherent lossiness of DPCM. A number of means were used to alleviate this. Firstly, a *scale factor* was introduced, which scaled all the differences represented by DPCM by a constant value, allowing for a more representative sample of image differences. Secondly, a *maximum pixel blur* parameter allows the user of the algorithm to control the maximum length for which a pixel can blur. This is implemented by encoding the absolute value of the pixel when it would otherwise blur for too long. The maximum pixel blur may affect the compression ratio of the image, whereas the scale factor does not do so.

An alternative method for eliminating blur was also introduced, where pixels are scanned in a "zigzag" order (the traditional method is to scan from left to right, for each scanline of the image). This exploits the coherence between pixels which are vertically close together and eliminates blurring at the borders of the image. This is particularly desirable because when the image blocks are placed next to one another, lines at the edges of the blocks show up as unacceptable artifacts.

Various packing structures were devised for when algorithm is in 24, 16 or 12-bit colour mode. For each of the modes, the number of bits used to represent the differences is half that used to represent the pixels.

The header data was compressed using a simplified RLE scheme. Since there are only two possible values, there is no need to indicate the symbol that is repeated, merely the length for which it is repeated. A special code is then necessary to encode values which repeat longer than 255.

3.3 Client

The problem we face in designing the client is to enable, possibly mobile, end-users to view complex 3D models on hardware architectures without native support for 3D rendering. Some of the reasons for doing this are:

- In many places hardware advancements are very slow due to the costs incurred by upgrading. This is particularly true in rural areas which have only older equipment. We aim to facilitate new uses for outdated computing equipment to these areas, without the need to frequently perform expensive upgrades.
- Computing today is becoming more and more mobile. We aim to provide a solution to the rendering of complex 3D scenes that is moved away from the traditional large servers, and introduces "portable rendering".

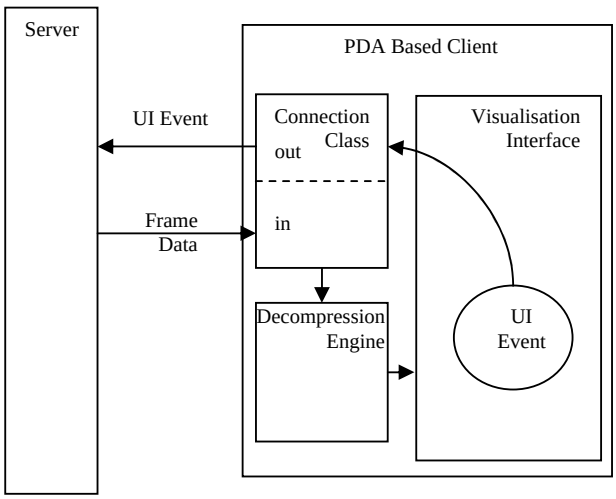

Figure 9. Overview of the Structure of the Client

The scenes must be displayed at a rate of at least 10fps (frames per second). This is accepted as the lowest interactive refresh rate which is necessary to create the illusion of real-time motion [1]. Anything lower than this results in the continuity of the scenes being broken, and the movement feeling "jerky". User input should also be interactive. Ideally, there should be no noticeable latency between user interaction with the system, and the system's response, as this increases cognitive processing. Usability studies have shown that response times under 50ms-100ms do not affect the user's perception [13]. This is consistent with the figure for an interactive frame-rate; 10fps indicates a frame change every 100ms.

We make provision for a network bandwidth of about 19kbps. Although this is relatively slow, it is what could be from older network internet connections common in rural areas.

The complexities of the distributed rendering system are transparent to the user. The user simply starts the client application and instructs it to connect to the server. From here on the system behaves as if it were based locally and can be seen as a single system.

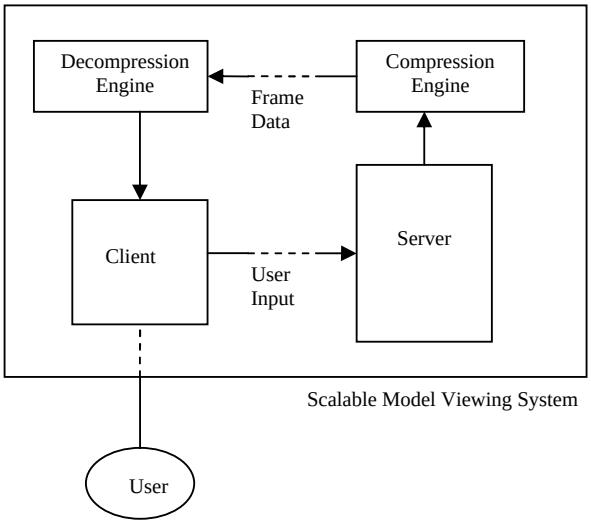

Figure 10. The Scalable Model Viewing System as One Unit

3.3.1 Display

The client display is divided into uniform square blocks. Since only the blocks which have changed since the last frame are transmitted from the server, only these blocks need be updated on the display.

In the case where very little is changing within the scene, having a smaller block size results in better performance, as less data is transported over the network. However, when larger changes are occurring within the scene, larger block sizes are optimal.

The server also sends an array of Boolean values relating to which blocks have been updated. The client receives this array which also tells it how many blocks have changed. It then runs through a loop of receiving a compressed block from the server, decompressing in and displaying it in the appropriate location. This is done until all the blocks have been updated.

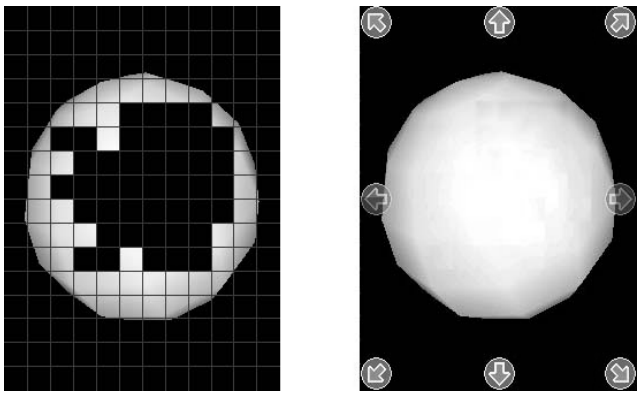

Figure 11. The Incoming Blocks and the Client View

3.3.2 User interface

The user interface constitutes a major part of the design of the client. The poor display resolution of PDAs may lead to poorer performance in information retrieval tasks [9]. Thus it is important to focus on creating a usable interface. It is most important that the user feels he or she is interacting with the environment itself, and not the system. The cognitive overhead introduced by the system should be kept to an absolute minimum, and a relatively inexperienced computer user should have no problems navigating around the environment.

We have tried to make the system as intuitive as possible by using affordances offered by the client device, such as the directional pad, which is used to indicate motion in a given direction. These buttons afford this information both by way of their layout and their design. They are situated closely together and each is noticeable as being either to the top, left bottom or right. They are also marked with arrows pointing in their direction of effect. The left and right buttons are used to move the viewpoint along the horizontal axis, or strafe

The system should allow the user 6 degrees of freedom when viewing models. This means that the user should be able to move left, right, up and down, forwards and backwards. In addition to this the user must be able to rotate the model about both the horizontal and vertical axes.

This poses something of a problem on the PDA itself as it has only the 4 directional buttons and 4 other function buttons. As a result, we extend the metaphor of the stylus as a pen, or pointing device to select items on the screen to convey user input to the system. The stylus is an *absolute*, *direct* and *continuous* input locator device [5]. It is well suited to picking out or placing entities, but not to repetitive tasks such as entering text. Thus, to simplify user input, buttons are placed on the edges of the screen in order to allow the user to interact with the model. The stylus is perfectly suited to touching these buttons in order to relate user input to the system.

This can create an additional problem, since we must either display information on the already small and cluttered PDA screen to somehow inform the users of the functionality of the stylus, or avoiding this, force the users to figure it out for themselves.

Neither of these is a desirable option. However, semi-transparent widgets or buttons are used in order to present more information

on screen [8]. The use of these types of buttons greatly reduces the obscuring of the actual display data whilst still affording the users with sufficient information as to how the interface works. The buttons are small arrows, which affords the user the information that they will have some bearing on the scene, based on the direction in which they are pointing.

3.3.3 Communicating with the Server

The client communicates with the server on two separate ports. One port is used for incoming display data, and the other is used to send the user interaction events and control information. This allows for smoother execution, as the data retrieval from the server is not interrupted by the UI events.

4. RESULTS

We present the results obtained from performing various tests on the components of our system.

4.1 Compression

The DPCM compression method performed consistently better than Jpeg for small image sizes (smaller than 1600 pixels^2), in terms of both speed and artifacts. Both the DPCM compression and RLE header compression are very consistent in terms of speed and compression ratio, whereas other compression methods perform differently depending on the input image.

Using DPCM combined with the method of dividing the display into fixed-size blocks performs better than Jpeg until the portion of blocks in the image which change reaches a certain threshold

4.2 Client

Both user tests and performance tests were used to evaluate the design of the client.

User testing revealed that the design was simple to use and intuitive. Users with little or no prior exposure to computers were able to quickly grasp concepts and make progress.

Performance tests showed that the average frame-rate did not drop below interactive levels in a variety of tests, even when displaying complex, textured scenes.

Figure 12. A Screenshot of the Client

4.1 Server

The aim of the Server was to provide an interactive frame rate in real time while minimising the use of the network. The Server aims to produce a frame rate of 20 frames per second. This specific frame rate is considered a good frame rate for an interactive program but the frame rate can drop to a minimum of 10 frames per second and still remain interactive. Any program's frame rate run on any machine will drop below this lower bound given a complex enough scene.

The Server must have an interactive frame rate for all scenes with 100 000 polygons. The Server was found to be able to produce a frame rate of greater than 15 frames per second for scenes of 25 000 polygons. The target of producing an interactive frame rate for a scene of 100 000 polygons was successful, however, the Server only produced a frame rate just greater than 10 frames per second for this scene.

5. CONCLUSION

The aim of the project was to provide an interactive frame rate on a PDA while minimizing the use of the network. This was considered a success in that the Server was able to produce very good interactive rates for fairly complex scenes and still producing a frame rate above the interactive lower bound for the more complex scenes.

The interface of the client was deemed intuitive and easy to use by the external test users, and the system proved to be responsive enough so as not to introduce cognitive overhead.

We have successfully met the goals set out at the beginning of the year and provided a novel approach to rendering complex 3D scenes on architectures without 3D capabilities.

6. REFERENCES

[1] S. T. Bryson and S. Johan, "Time Management, Simultaneity and Time-Critical Computation in Interactive Unsteady Visualization Environments", In *IEEE Visualization '96*, IEEE, October, 1996.

[2] I. Buck, G. Humphreys and P. Hanrahan, "Tracking Graphics State for Networked Rendering", In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, Interlaken, Switzerland, 2000.

[3] E. J. Conklin, "Hypertext: An Introduction and Survey." *IEEE Computer 20* (September 1987), 17-41.

[4] A. Drozdek. Elements of Data Compression. Brooks/Cole, 2002.

[5] J. Foley, A. van Dam, S. Feiner and R. Philips, "Introduction to Computer Graphics", Addison-Wesley, 1990.

[6] W. Gaver, "Technology Affordances", In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New Orleans, USA, 1991.

[7] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*. Vol. 40, September 1952, pp. 1098-1101.

[8] T. Kamba , S. A. Elson , T. Harpold , T. Stamper and P. Sukaviriya, "Using Small Screen Space More Efficiently", In Proceedings SIGCHI'96, Vancouver, Canada, ACM Press, April, 1996.

[9] L. Kärkkäinen and J. Laarni “Designing for Small Display Screens”, In *Proceedings of the Second Nordic Conference on Human-Computer Interaction*, Aarhus, Denmark, 2002.

[10] B. MacIntyre and S. Feiner, “A Distributed 3D Graphics Library”, In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, July, 1998.

[11] C. Marlin, L. Brown, Ed Jones. “Human-Computer Interface Design Guidelines”, Ablex, 1989.

[12] J. Nielsen, “Heuristic evaluation” In J. Nielsen and R. Mack, *Usability Inspection Methods.*, John Wiley & Sons, New York, NY, 1994.

[13] B. Shneiderman, “Designing the User Interface: Strategies for Effective Human-Computer Interaction”, Addison-Wesley, Reading, MA, 1998.

[14] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*. Vol. IT-23, No. 3, May 1977, pp. 337–343.

[15] J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*. Vol. IT-24, No. 5, September 1978, pp. 530–535.