Supporting Multiple Output Devices on an Ad-hoc Basis in Visualisation

A thesis
submitted in partial fulfilment
of the requirements for the Degree of
Master of Software and Information Technology
at
Lincoln University

By

Xi Zha

Lincoln University

February 2010

Abstract of a thesis submitted in partial fulfillment of the requirements for the Degree of Master of Software and Information Technology

Supporting Multiple Output Devices on an Ad-hoc Basis in Visualisation

By Xi Zha

In recent years, new visualisation techniques and devices, such as remote visualisation and stereoscopic displays, have been developed to help researchers. In a remote visualisation environment the user may want to see visualisation on a different device, such as a PDA or stereo device, and in different circumstances. Each device needs to be configured correctly, otherwise it may lead to an incorrect rendering of the output. For end users, however, it can be difficult to configure each device without a knowledge of the device property and rendering. Therefore, in a multiple user and multiple display environment, to obtain the correct display for each device can be a challenge.

In this project, the focus on investigating a solution that can support end users to use different display devices easily. The proposed solution is to develop an application that can support the ad-hoc use of any display device without the system being preconfigured in advance. Thus, end users can obtain the correct visualisation output without any complex rendering configuration.

We develop a client-server based approach to this problem. The client application can detect the properties of a device and the server application can use these properties to configure the rendering software to generate the correct image for subsequent display on the device.

The approach has been evaluated through many tests and the results show that using the application is a useful in helping end users use different display devices in visualisation.

Keywords: Visualisation, Multiple Display Devices, Client-Server based application

Acknowledgements

I would like to express my deepest appreciation to my supervisors, Stuart Charters and Keith Unsworth, for their support and guidance. Without their contributions and persistent help this thesis would not have been possible.

I would like to thank the TLS(Teaching and Learning Services)staff, Caitriona Cameron, for her help and advice in the use of the English language.

Finally, special thanks go to my parents and my partner Yan Zhang for their love and support during my study.

Table of Contents

Tables

Figures

Chapter 1 Introduction

Visualisation is an important technique used in many areas to help people understand and solve problems. “Visualisation is the process of presenting data in a form that allows rapid understanding of relationships and findings that are not readily evident from raw data” (Schroeder, Martin, & Lorensen, 1998).

As computing power has increased, complex scientific simulations have become possible and the sizes of scientific data sets have grown to terabyte or petabyte size in areas such as the geosciences and meteorology. Visualisation and rendering face challenges as the data sets size increases.

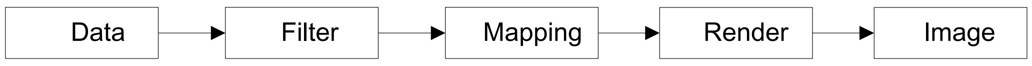


Figure 1-1Haber-McNabb Visualisation Pipeline

Figure 1-1 shows the typical visualisation and rendering pipeline. The first step provides the visualisation data to filter stage; at this stage the data may be processed to obtain the format, structure, and types that the map stage requires. The map stage generates a visual representation of the input data which is the input to the render stage. Lastly, the data will be rendered to generate the final image. However, for a large visualisation model, data sets of this size will not fit within the storage of a typical desktop workstation environment. Also, there is a limited amount of CPU power to process data sets of this size. For that reason, to use large data sets in visualisation, high-end workstation or supercomputers are often good solutions. Thus, for low-end client platforms, such as PCs and PDAs, a server that maintains large-scale data sets must handle the majority of computing and storage needs of the whole visualisation process (Ding, Huang, Beck, Liu, Moore, & Soltesz, 2003).

In a visualisation system, if the data of pipeline on one machine and the users want to view results on another machine then we need remote visualisation approaches to help the user to process the data and image.

Several remote visualisation techniques have been developed to solve large data problems; these are discussed below.

One remote visualisation technique is a client-server based system for visualising environments (Deb & Narayanan, 2004). This visualisation approach uses a high performance platform to store and process large data sets and allow different clients to connect and receive the visualisation output.

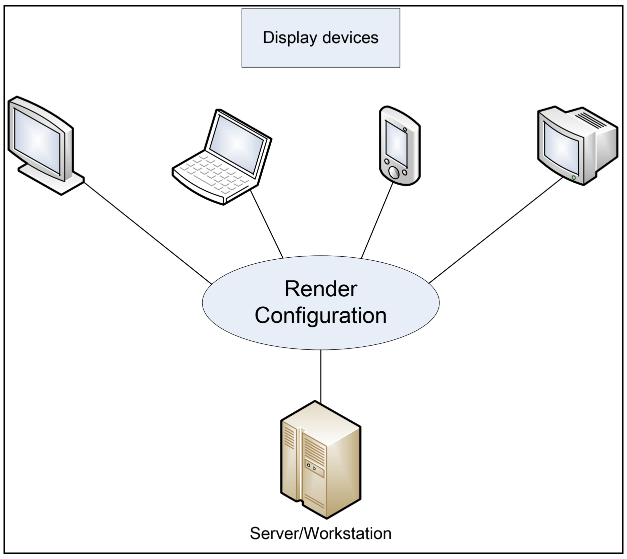


Figure 1-2 Remote visualisation environment

Normally, in a remote visualisation system, many users may use different display devices to view the visualisation result. These devices may include standard 2D monitor, small screen and stereoscopic display. Figure 1-2 shows a multiple display device environment in a remote visualisation system. Within the system, different users may use different display devices to view the visualisation, one may use a standard monitor and one a stereoscopic monitor. In the remote visualisation system, if the user changes the display device they are using to view the image, the rendering needs to be reconfigured to provide the correct output. A scenario is discussed which explains this in more detail. A scientist normally views the visualisation on a 15” desktop monitor with 1024*768 resolution. In some cases, the scientist needs to receive the image on a mobile device such as a PDA which has a screen size of 3.5” and resolution of 320*240. Before the scientist uses the PDA, he needs to give the new device’s properties to the rendering server to reconfigure the image generation. Then the PDA can receive the correct visualisation output from the server. In addition, if the scientist needs to use the desktop monitor again, the rendering must be reconfigured again.

From this example, we can see that if the user wants to display the correct image on different display devices, the rendering must be reconfigured each time. This situation is not always convenient, because many users may not know enough about the properties of the display device they are using or how to change the rendering configuration.

1.1 Aim

In this project, we will investigate how to simplify supporting multiple display devices in a remote visualisation environment. To support multiple display devices, we should consider these situations: one user using different devices to view the visualisation at different times, multiple users using different devices concurrently and also at different times.

To succeed, the solution must not require users to work with a complex rendering system or to have a depth of knowledge of the properties of their display devices. The aim of this project is to provide a solution that allows users to use different types of display devices with automatic configuration of rendering services to provide the correct image for display on their connected device.

Chapter 2 Literature Review

Visualisation techniques can be used to represent complex scientific images. They allow users to study experimental data or try experiments that may be difficult in the real world. In this chapter, a general discussion of visualisation and rendering techniques will be introduced in section 2.1. Section 2.2 will describe different types of display devices and discuss possible issues for rendering for these devices. Section 2.3 will discuss approaches to remote visualisation.

2.1 Visualisation and Rendering

Visualisation is a technique for using the data and computer graphics to create images to communicate a message. Visualisation today has been widely applied in the area of science, education, engineering, interactive multimedia, and medicine (Haber & McNabb, 1990) (Ware, 2004). In a visualisation system, the final image is generated by rendering.

Rendering is the process of generating an image from a model by means of a computer program. The model is a description of the objects in a computer programming language or data structure. It contains geometry, viewpoints, texture, lighting, and shading information. The image produced by the rendering system from the model may be a raster graphics image. Many rendering algorithms have been researched, and software used for rendering may implement a number of different algorithms to produce the final image. This section provides a discussion about rendering algorithms and their uses.

2.1.1 Rendering algorithms

Many rendering algorithms have been developed. These algorithms can be grouped into four loose families: rasterisation, ray casting, ray tracing, and radiosity; these are discussed below.

a) Rasterisation

Rasterisation is the process of converting graphics to a bitmap to output on a display device. A high-level representation of an image should contain many elements not only the pixels. These elements are referred to as primitives. For instance, in a graphical user interface, windows and buttons might be the primitives. In 3D scene, triangles and polygons in space might be primitives. In many cases, a pixel by pixel rendering is too slow for the task, then a primitive by primitive approach of rendering may helpful. This approach uses one loop through each of the primitives to determine which pixels in the image are affected, and then modify those pixels. This rendering method is called rasterisation, and all current computer graphics cards use this method.

b) Ray casting

Normally, ray casting is used for real-time simulations, such as in 3D computer games and cartoon animations, where detail is not important, or where there is a need for better performance in the computational stage. This is usually the case when a large number of frames need to be animated. Basically, the ray casting can be broken down into three steps. For each object in a scene, firstly, construct ray from eye position through view plane. Secondly, find first surface intersected by ray through pixel. Thirdly, compute colour based on surface radiance. More details are introduced in (Watt & Policarpo, 1998).

c) Ray tracing

In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane (Watt & Policarpo, 1998). Ray tracing is an extension of the same technique developed in ray casting. This method can handle complicated objects very well, and the objects may be described mathematically. This makes ray tracing best suited for applications where the image can be rendered slowly, such as in still images or television special effects. This method is poorly suited for real-time applications, such as computer games, where speed is very important.

d) Radiosity

Radiosity is a method used to simulate how directly illuminated surfaces act as indirect light sources that light up other surfaces. The surfaces of a scene are divided up into one or more small surfaces. A view factor for each small surface is generated. After then, the view factors are used for the rendering equation to render the image (Watt & Policarpo, 1998). This approach produces more realistic shading and is good when used to represent an indoor scene. A classic example is the way that shadows in the corners of rooms are rendered.

2.1.2 Volume rendering

Volume rendering techniques have been developed to display a 2D projection of an object with in a 3D data set. Meißner, Pfister, Westermann, and Wittenbrink (2000) describe volume rendering. Volume rendering is a key technology with increasing importance for the visualisation of 3D sampled, computed, or modelled datasets. A typical 3D data set is a group of 2D slice images acquired scanning techniques, such as MRI, CT, PET, or ultrasound. For volume data, one key goal is to convey the structure of the data distribution, for example the shape of the liver. By using this technique, the volumetric data can be displayed as a two-dimensional image to the user. Medical imaging is one area where volume rendering is frequently used.

Compared with other rendering techniques, such as ray casting, the volume rendering technique has the advantage of presenting the structure of the volume, rather than selected boundary surfaces of variable value or coordinate value.

2.1.3 Stereoscopic rendering

In today's world, with the ongoing development of 3D display technology, people can show an image with the illusion of depth on a 2D screen. The illusion of depth using photographs, movies, or other two-dimensional image is created by presenting a slightly different image to each eye. Three dimensional display technology holds great promise for the future of television, virtual reality, entertainment, and visualisation.

Unlike rendering 2D images, stereoscopic rendering needs to calculate stereo pairs to create a perception of depth, so the renderer can generate 3D images. Before discussing how to calculate stereo pairs, the difference in characteristics between 2D images and 3D images will be explained. Bourke (1999) introduces a number of cues that the human visual system uses to see in stereo. Some cues are present even in 2D images.

Table 2-1Cues to the human visual system present in 2D and 3D images

Cue Name	**Explanation**
Perspective	Objects get smaller further away.
Sizes of known objects	An object appears smaller than other objects. For example, if a car and a teacup appear the same size then we expect the car to be further away.
Detail	Close objects appear in more detail, distant objects appear in less detail.
Occlusion	An object that blocks another is assumed to be in the foreground.
Lighting, shadows	Closer objects are brighter, distant ones darker. There are a number of other slight cues implied by lighting: the

	way a curved surface reflects light suggests the rate of curvature, shadows are a form of occlusion.
Relative motion	Objects further away seem to move more slowly than objects in the foreground.

Table 2-1provides details about the cues which are present in both 2D images and 3D images. There are other cues that are only present in 3D images. These cues are listed in Table 2-2.

Table 2-2 Cues to the human visual system present in 3D images

Cue Name	**Explanation**
Binocular disparity	This is the difference in the images projected onto the back the eye, because the eyes are separated horizontally by the interocular distance.
Accommodation	This is the muscle tension needed to change the focal length of the eye lens in order to focus at a particular depth.
Convergence	This is the muscle tension required to rotate each eye so that it is facing the focal point.

In the real world, binocular disparity (the difference in image location of an object seen by the left and right eyes) is considered to be the most important depth cue in the human visual system. In order to create a stereo pair, two images need to be generated, one for each eye. Creating stereo pairs involves rendering a left and a right eye view from positions separated by the interocular distance (the distance between the center of rotation of the eyeballs). The view directions of the eyes are parallel to each other, so the frustum from each eye to each corner of the projection plane is asymmetric, as shown in Figure 2-1(a).

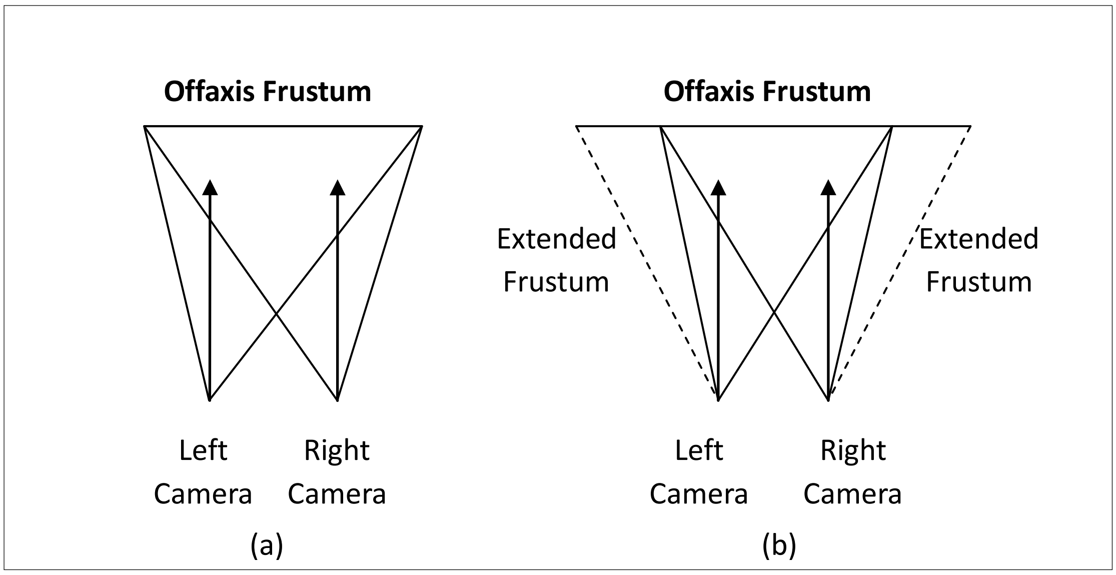


Figure 2-1 Camera geometry of stereo pairs (Mackay, 2006)

There are several methods of setting up and rendering the stereo pairs. To generate correct stereo pairs, the frustum needs to be extended horizontally for each eye or camera making it symmetric (Figure 2-1(b)). After rendering, the image from the extended frustum will not be displayed, thus the user can obtain the correct stereo effect from the screen.

There are a number of methods for setting up the camera and rendering the stereo pairs. Mackay (2006) explains how to calculate and generate the stereo pair. The basic principle for generating a stereoscopic image is to use two cameras within a scene to generate slightly different images for each eye, so that the user can obtain a perception of depth from a three dimensional image.

2.1.4 Summary

The main rendering techniques introduced in this section are rasterisation, ray casting, ray tracing, radiosity, volume rendering, and stereoscopic rendering. Most visualisation systems use one or more of these rendering techniques. This section also discussed stereoscopic rendering. Unlike other rendering techniques, stereoscopic rendering is based on other rendering techniques, such as ray tracing; with stereoscopic rendering, the focus is on how to obtain the correct camera parameters for generating a stereo pair to produce the final image, rather than an algorithm.

2.2 Display Devices

A display device is an output device for the presentation of information for visual or tactile reception, acquired, stored, or transmitted in various forms. In visualisation environments, the display device is an essential component.

The properties of different display devices are quite varied. The following table shows common display properties for some different types of display devices.

Table 2-3 Common display devices and properties

Display Device	Resolution	Number of Images Required
PDA	240x160 to 640x480	One
24’ Standard Monitor	1920x1280	One
Projector	800x600 to 1920x1080	One
24’ 2 View Stereoscopic	1920x600 （per eye）	Two
Multiview (9) Stereoscopic	1080x600 to 1920x1080	Multiple(18)
Volumetric Display	768x768x198	Multiple(198)
PowerWall	High Resolution	Single or Multiple
CAVE	Ultra High Resolution	Multiple

From Table 2-3 we note that the display device type determines both the resolution and the number of images required. The resolution typically increases with screen size. In addition, for anormal2D monitor and small screen devices, only one image is required for the 2D display effect but the stereoscopic display devices require multiple images to achieve the stereo display effect. A visualisation environment enables users to access images using different types of display devices. Therefore, to generate the correct image, only providing the visualisation data is not enough, the display device properties also need to be given to configure the renderer so it can create the correct image for display.

2.2.1 Standard display device

In general, the standard display device can only display 2D images: most people use standard display devices to view visualisations. The standard devices only require single image input to display. Therefore, when a visualisation system generates the image for different standard devices, the main issue of concern is the current resolution of the monitor. Most current visualisation systems can support standard display devices directly. The rendering server does not need to make a special change, it only needs to adjust the resolution for the new display device to generate the correct image. Two of the most common standard display devices are CRT and LCD.

a) Cathode Ray Tube (CRT)

The cathode ray tube display device is usually called CRT monitor and is one of the most widely used display devices.

The CRT monitor has two important properties that need to concern us when we view the visualisation: they are resolution and refresh rate. A CRT monitor usually supports a range of resolution. Thus, to obtain the correct image display, the resolution property is important.

b) Liquid Crystal Display (LCD)

Liquid Crystal Displays have become more and more popular as display devices in today’s world. Compared with traditional CRTs, LCD devices have advantages in size, weight, and electric power consumption.

Unlike with CRT monitors, there is an optimal resolution of a LCD screen. The image will be displayed correctly and provides the best view under the optimal resolution. The resolution is the only property of concern when rendering an image.

c) Projector

Projectors are one of the common ways of displaying visualisation. A projector takes a video signal and projects the corresponding image on a project screen using a lens system. A common projector for a visualisation system is the portable projector, and the display resolutions include 800x600 pixels, 1024x768 pixels, 1280x720 pixels and 1920x1080 pixels.

Like the LCD monitors, a projector has an optimal resolution. The image size should be set as the optimal resolution so that the image can be displayed correctly for the projector. Hence, resolution is the key property when rendering an image for a projector.

2.2.2 Stereoscopic display device

In recent years, stereoscopic display devices have become increasingly popular and practical in the computer visualisation community. A stereoscopic display device is a display device capable of conveying 3D images to the viewer. There are few basic types of 3D display devices. A stereoscopic device separately sends two views of a 3D scene onto the screen for the viewer, one for each eye. Usually the viewer needs to use some optical or electric equipment, such as special glasses or headgear, to obtain the 3D effect from the display device.

Autostereoscopic display devices can present a 3D image to a viewer without the use of 3D glasses or other special viewing equipment. There are three classes of autostereoscopic displays: reimaging displays, volumetric displays, and parallax displays(Halle, 1997). Reimaging displays can represent an existing three dimensional object in a new location. Volumetric displays can illuminate an object in three dimensions in a spatial volume. Parallax displays emit directionally-varying image information into the viewing zone. Parallax displays are the most common autostereoscopic displays and are most compatible with computer graphics. Therefore, the information about parallax display devices will be introduced in this section.

Halle (1997) describes the working principles of parallax devices. The surface of a parallax display is covered with a special display element that can emit light of varying intensity in different directions. Figure 2-2 shows the different surfaces of parallax displays.

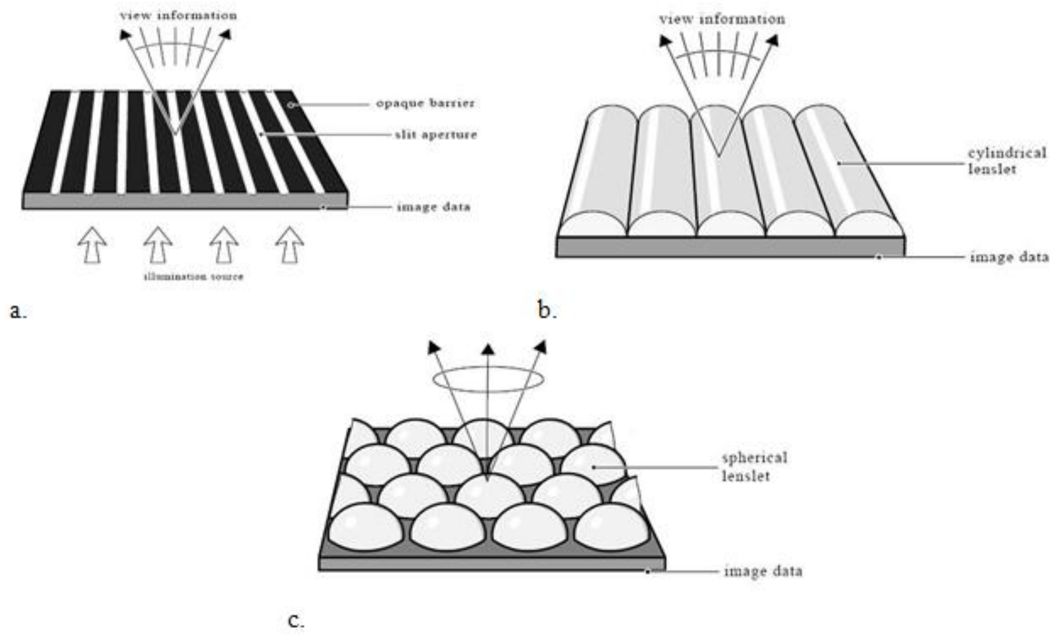


Figure 2-2 a) Parallax Barrier Displays b) Lenticular sheet displays c) Lenticulardisplays

When rendering for an autostereoscopic display device, some camera and image parameters are required. These are:

- Position of the camera.

- The camera parameters, such as eye separation and focal length.
- The number of images that must be rendered.
- Pixel resolution and image format of the image data.

Annen, Matusik, Pfister, Seidel, and Zwicker (2006) introduce a multiview autostereoscopic display device. Multiview parallax displays allow the user to obtain stereoscopic views without glasses, and to view the image at arbitrary positions within the viewing zone. In Annen et al.'s work, the developers used 16 projectors each with 1024*768 output resolution to implement the multiview 3D effect display. In addition, the authors introduce two different approaches to generating multiview 3D displays: optical multiplexing for lenticular screens and software multiplexing for parallax-barrier displays. In optical multiplexing, each view is projected as a whole to the display surface. Optical multiplexing is easy to implement in a distributed rendering environment, but there are some weaknesses with this approach. The viewing angles are limited, and flat-screen display devices, such as a high resolution LCD, are not feasible with optical multiplexing.

In contrast to optical multiplexing, software multiplexing allows for more flexible and compact display designs. In software multiplexing, the rendering software will use the parameters, such as the number of projectors and view angles, to generate the corresponding multiview pixels for the projectors and then compose these pixels to the screen to obtain the 3D display.

2.2.3 Summary

Users may use various types of display devices to view visualisations. In order to obtain a correct image display, the visualisation system needs to know which type of display is being used and also the corresponding property information. This section discussed the characteristics of different kinds of displays. For the standard display device, the resolution is important property. For stereoscopic devices, not only is the resolution information important but also other property information, such as eye separation and focal length.

2.3 Remote Visualisation

Remote visualisation is used to solve large visualisation model problems. Normally, a low-end computer is unable to store and process large data sets. Therefore, users need a high-performance computer to process large data sets. Generally, two strategies have been developed to visualise large data sets. The first approach, usually called render remote, generates images on a high-end workstation which stores the data sets, then transmits the image to the user who is connected with the workstation. The link between the workstation and client would be over a network connection. In the second strategy, usually called render local, smaller portions of data are sent to the client on demand. The local client then uses the data to do the rendering. The network connection between the data source and visualisation is shown in Figure 2-3

In a remote visualisation environment, the networks which connect the client and server are very important. The transfer speed of the network is a major factor that determines the remote visualisation performance. In recent years, two key developments have allowed users to explore different approaches. The first development is a network data cache that is tuned for wide-area network access, called the Distributed Parallel Storage System or DPSS (Tierney, Crowley, Holding, Hylton, & Drake, 1999). The DPSS is a data block server, built using low-cost hardware components and custom software to provide parallelism at the disk, server, and network level. Current performance results of this technology are 980Mbps across a LAN and 570 Mbps across a WAN.

The other key development is a proliferation of high-speed networks. There are currently a number of next generation Internet networks whose goal is to provide network speeds of 100 or more times the current Internet network speed. These include NSF’s Abilene, DARPA’s Supernet, ESnet testbeds, and KAREN. Typically, the WAN connection speed of these networks could achieve 622Mbps of OC12 or 2.4Gbps of OC48 (Bethel, Tierney, Lee, Gunter, & Lau, 2000). In addition, KAREN (Kiwi Advanced Research and Education Network) can provide up to 10 gigabits a second connection speed. Along with the developing of high-speed network technologies, many approaches to remote visualisation have been developed.

2.3.1 Remote visualisation framework

The principle of remote visualisation is to use a high-end workstation as a server to handle large data sets and allow remote users to access the visualisation output. Therefore, the remote visualisation approach is a client-server based method. The server and client communicate via a network environment.

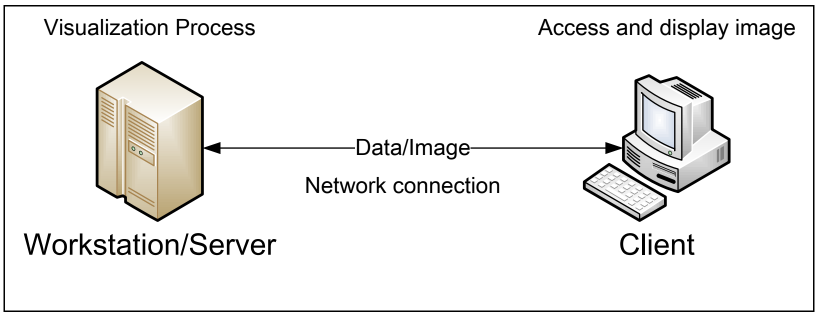


Figure 2-3Remote visualisation structure

Figure 2-3 presents the architecture of the remote visualisation environment. A majority of current remote visualisation projects use this basic structure. However, the implementation of the three main components, Client, Server and Network, can be quite dissimilar in different projects.

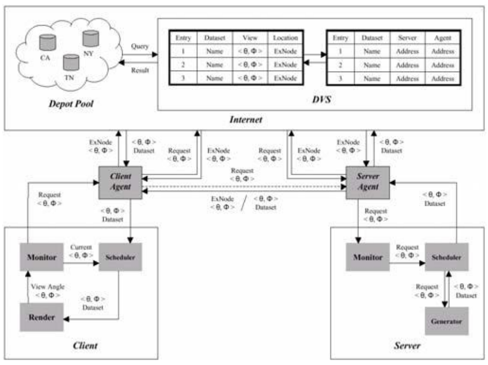


Figure 2-4Pipeline of a remote visualisation system over logistical networking infrastructure (Ding et al., 2003)

Ding et al. (2003) explain how to implement remote visualisation over a logistical networking infrastructure. In Ding's work, the developer uses a light-field rendering and light-field database for remote visualisation. A light-field database is a small database that stores part of the data sets of objects; light-field rendering is rendering using the data from the light-field database to generate the image of particular view sets of an object. To allow a user to navigate different volumes, multiple light-field databases are needed. The reason light-field rendering and databases are used is for large complex data sets, as to visualise a scene with raw geometries is too inefficient. A streaming model over Logistical Networking infrastructure is introduced in Ding et al.'s paper, and the working process of server, server agent, client, and client agent are shown in Figure 2-4. In this system, the server stores the light-field database of all view sets, which is then uploaded to a depot pool. After that, a dictionary service listing each view set, as well as its exNode pointer, is created and maintained on a server-based Dictionary of View Sets (DVS server). View set is accessed via a corresponding exNode. The exNode table stores the index of view sets and corresponding exNode pointers. When a request is received, it consults the exNode table. If it finds the requested view set, it returns the corresponding exNode. The retrieved exNode is then used to obtain a copy of the view set from the pool.

The characteristic technique of this approach is that it uses the concept of Logistical Networking to build up the network connection between the client and server. In addition, this method needs to build a light-field database and dictionary table to improve the efficiency of the visualisation.

Yuen, Garbow and Erlebacher, (2004) introduce an approach to remote visualisation using Internet and off-screen rendering techniques. This system is based on the client-server model, and the developer needs to create a software application acting as middleware to allow the client access to the data. Figure 2-5 demonstrates how this middleware connects the client with the server to obtain the data. The complex details of accessing the data is hidden in the middleware so the client can obtain the data simply, while the middleware takes care of network communication and security.

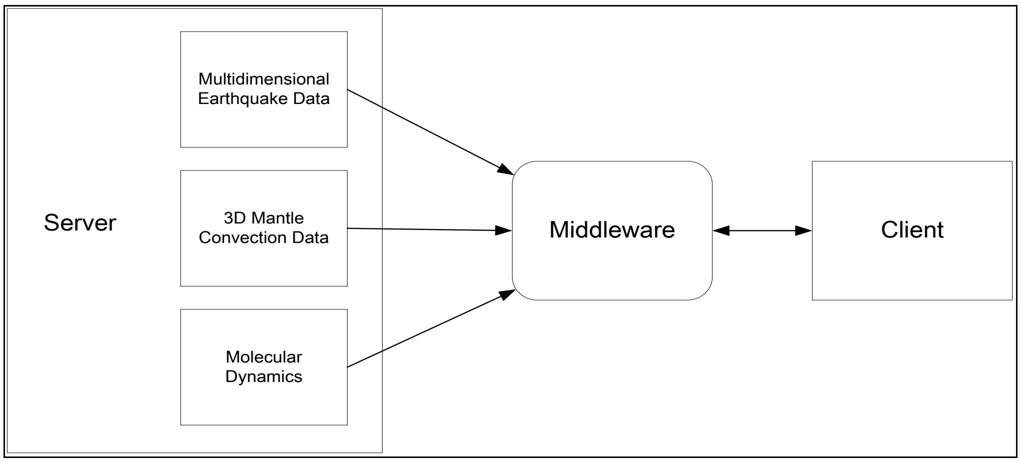


Figure 2-5Themiddleware allows the client to access an assortment of geophysical data from the server

This project uses a combination of CORBA, C++, Java, Python, and OpenGL to provide the middleware and client application. Figure 2-6 illustrates the implementation of the application using these programming tools. The client application provides a graphical user interface so the user is able to change and view the visualisation interactively. After the server receives the request from the client, it begins data analysis and rendering the image, and then returns the resulting image to the client. The system uses CORBA to connect and transfer information between the client and the server.

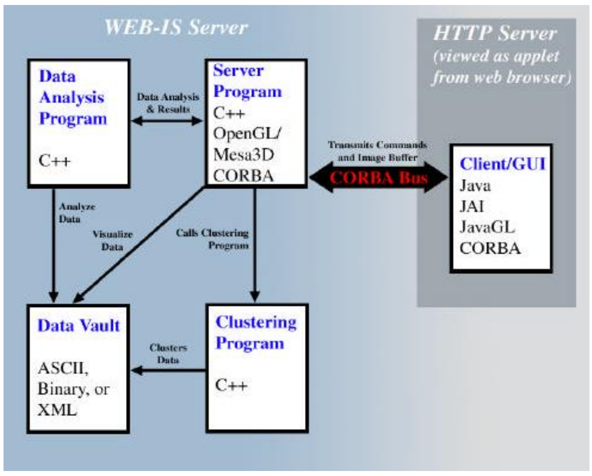


Figure 2-6 Implementation of middleware and client application (Yuen et al., 2004)

In this approach, the client application allows the user to view the visualisation remotely using a web browser and provides a Java applet as the front-end interface through which users can interactively select subsets of their data to analyse and to view the results of the analysis. As an example to demonstrate how it can be used, a user who wishes to interrogate a large cluster of earthquakes to identify patterns in the interrelationships of small earthquake events first loads the client applet over the Internet into a web browser. He then inputs the clustering parameters into the applet to produce the desired output. After that the server will produce the visualisation based on the parameters and pass the image back to the user.

This project uses the CORBA bus to translate commands and the image buffer. This protocol can provide interoperability between objects on different machines, among multiple different programming languages, regardless of the machine's platform (Yuen et al., 2004). The other feature of this approach is off-screen rendering. This function is implemented on the server side and enables multiple clients to render different data simultaneously.

The Resource-Aware Visualisation Environment (RAVE) uses Grid/Web Services to advertise data sources and recruit rendering assistance from remote resources (Grimstead, Avis, Walker, & Philp, 2005). The RAVE architecture includes a Data Service, Render Service, and Thin Client. Figure 2-7 shows the architecture of the RAVE system.

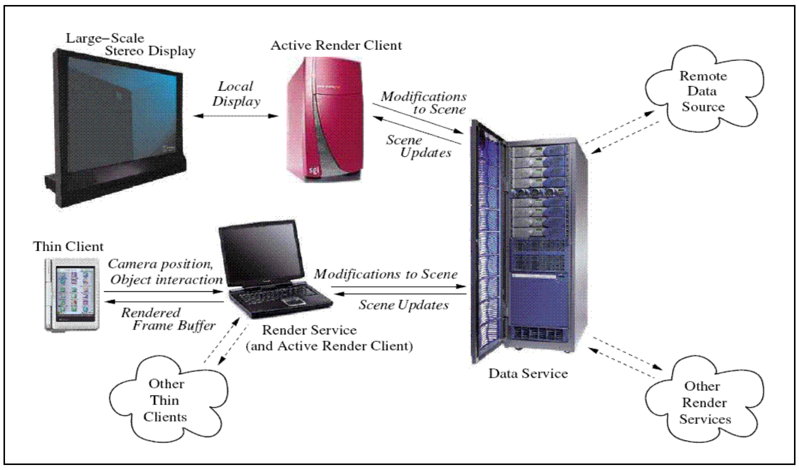


Figure 2-7 RAVE architecture (Grimstead et al., 2005)

In the RAVE system, the data service imports data from either a static file or a live feed from an external program, either of which may be locally or remotely hosted. The render services connect to the data service to obtain the latest data and generate an image. The render service can be exposed to the local console, so the user can interactively modify the visualisation output. However, if a local user does not have the facility to install a render service on their machine, an active client can be installed instead—this is a stand-alone copy of the render service that can only render to the screen and does not support off-screen rendering. From the architecture chart (Figure 2-7), we see that the active render client on the top can only support a local stereoscopic display, but the laptop console is a render service and an active render client. Thus, it can generate a local display on its own screen as an active render client and also support remote display to another thin client as a render service.

In modern medical science, visualisation makes a significant improvement on research methods, especially in anatomy and neuroradiology. The increasing capabilities of magnetic resonance (MR) imaging and multisection spiral computed tomography (CT) can represent the structure of human brain as shown in Figure 2-8.

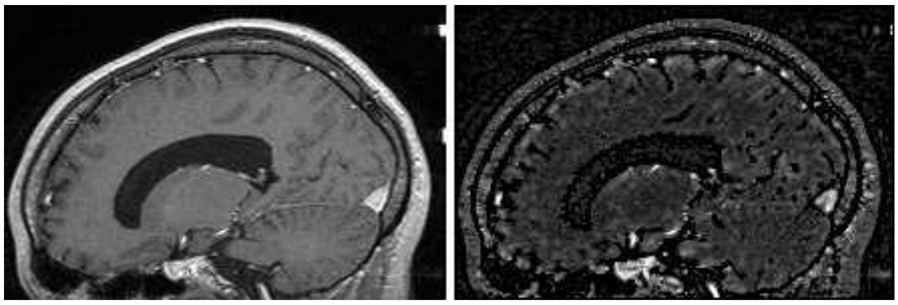


Figure 2-8 CT and MR image of human brain (Sato et al., 1997)

In addition, the CT and MR scanner can acquire volumetric data with near-isotropic voxels, using these data to make three dimensional images is necessary, especially in studies of complex structures like intracranial vessels. Most modern CT and MRI, however, provide limited 3D image processing capabilities, so to implement 3D visualisation with interactive operations requires a high-end performance graphics workstation that is not available at many medical institutions. In this situation, remote visualisation is a good solution to the problem. Bethel et al. (2000) discuss an approach that combines fast visualisation on a normal PC system with high-quality visualisation on a high-end graphics workstation that is directly accessed and remotely controlled from the PC environment via the Internet using a Java client.

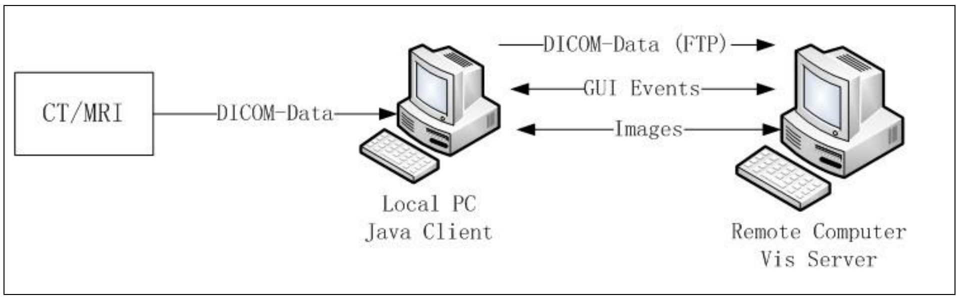


Figure 2-9Diagram of the data stream in a local network

Figure 2-9shows the data stream in this remote visualisation system, the DICOM (Digital Imaging and Communications in Medicine) image data obtained from the CT scanner or MR imager. The image data are transferred to the local PC first and then by FTP to the visualisation server, which is remotely controlled from the local PC by use of a Java client application. In this system, the server can process the data, generate the images and send them back to the local PC; users are able to change and view the visualisation interactively.

To evaluate this system, researchers used CT angiography data of five patients with intracranial aneurysms transferred to the workstation directly after the investigation. The average time for the complete visualisation process including data transfer was 15-25 minutes depending on the complexity of the structure. That result is fast enough for this method to be used in the clinical situation of patients with subarachnoid hemorrhage under emergency conditions(Bethel et al., 2000).

2.3.2 Support visualisation output on different displays

In the RAVE system, the render service connects to the data service and requests a copy of the latest data. As render services have a full scene data, an image may be rendered as required. In this system, the developers focused on the data transfer and command transfer between the clients and servers. There are two kinds of client within the RAVE system, the active client and thin client. The active client collects the required datasets from the data server and renders the image locally, and the thin client uses a remote render server to generate the image.

The solution provides the client user application to the users. The application allows the users to select different datasets from the data server to render and provides interaction, such as move and rotate, with the visualisation.

The users must use the RAVE application to view the visualisation result. The RAVE application renders the image with a fixed size and colour depth. The datasets discussed by Grimstead et al (2005). were rendered at 400x400 resolution with 24 bits. The image size and colour depth cannot be changed. Details of the rendering process, such as how to set the resolution and colour depth of the image, are not provided.

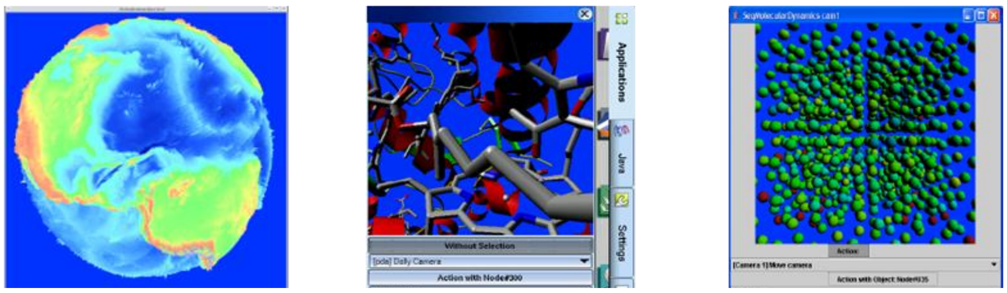


Figure 2-10RAVE client interface run on Linux, PDA and Windows platform

In RAVE, the clients view the visualisation at a fixed size (Figure 2-10). This solution works well for the standard 2D display client. Grimstead et al. (2005) mention that the client may view the visualisation on a stereoscopic device; however, they do not provide details to explain how to support devices such stereoscopic displays to obtain the 3D effect.

The solution suggested by Garbow et al. (2003) provides a Java applet for the client. This applet is accessed through a web browser and provides functionality, such as selecting datasets, rendering data, interactive data mining, and analysis of the data.

The visualisation is displayed in a window with a fixed size in the applet user interface. Therefore, the servers do not need to collect the display properties from the client to render the image. The client only needs to access the applet and then view the visualisation result. This solution allows multiple users to access and view the visualisation. However, Garbow et al. do not mention if solution can support different types of display such as a stereo display device. The special display device may not obtain the correct image from the server.

Engel, Sommer, Ernst, and Ertl (1999) introduce an approach for web-based remote 3D visualisation techniques. This approach uses the typical remote visualisation structure, the image generated by a high-end visualisation server, and the clients access and view visualisation by the network connection. This solution has developed into a web-based Java application. The application provides a window to display the visualisation result. The users can select different image quality with different resolution. However, support for the special display device is not provided.

Remote visualisation approaches provide a client application. The client application could be web-based or stand alone. Most of the applications display the visualisation result in a window with a fixed size. The approach, such as RAVE, mentions that the client can use stereoscopic display to view the visualisation, but does not give the detail of the rendering process. All these approaches can support multiple users but most of them cannot support multiple types of display devices properly.

2.3.3 Summary

Remote visualisation provides a solution so that users are able to access high-end workstations to help with complex or large visualisation models. There are a number of visualisation approaches that have been developed. All these solutions are client-server based systems. However, the platform, program language, and network protocol may be different. Therefore, we need to consider how to support multiple platform environments when we develop a solution. Furthermore, most remote visualisation systems cannot support special devices, such as stereoscopic devices, directly. RAVE can support the stereoscopic device but it still needs an active render to generate the image for it.

2.4 Literature Review Summary

A visualisation system combines many different components. For the end users the display device is an essential component. Recently, many visualisation models have become large and complex, and standard computer environments, such as personal computers, are not able to process this kind of visualisation model. Therefore, remote visualisation has been created to solve the problem. In a remote visualisation system, users can use different types of display devices to access the visualisation; however, many systems do not correctly support some devices. From section 2.2, we understand that key properties for different types of devices are not the same. In addition, for special devices, such as stereoscopic devices, we need to use stereoscopic rendering techniques to calculate the stereo pairs and display the right image.

Chapter 3 Analysis and Design

This chapter presents an analysis of the requirements of this project and introduces the proposed design of the system.

3.1 Requirements

In recent year, a number of new display technologies have been used in visualisation, such as autostereoscopic and volumetric display devices. At the same time, remote visualisation has begun to become widely used to help the researcher deal with large, complex visualisation projects for scientific research. With this technique, users can easily use a high-end workstation to process the complex data sets and obtain the final image from the workstation. One of the most important characteristic of remote visualisation technique is that it provides a multi-user environment, which means a number of clients may use several different types of display to view the visualisation result. The diversity of display devices requires the renderer to be configured for each device so that the device displays the correct image. The requirements for a system to generate the correct output for a display include:

1. **Multiple user environment**

   Current remote visualisation systems are designed with a multiple user environment. For different types of visualisation system, the solution should be able to support both single and multiple user environments.

2. **Multiple display devices**

   A number of new display devices have been developed and begun to be used in the visualisation area. In a multiple user environment, different users may use different types of display devices in their local system to view the visualisation output. In addition, a single cliental so can use many different display devices in a local visualisation environment. The diversity of display device properties has been discussed in the literature review section, and this diversity may cause the incorrect display of images on a device if not correctly configured. To obtain and display the correct image is a basic requirement of a visualisation system. Therefore, this situation requires a solution to support various types of displays to ensure the user can obtain the correct image.

3. **Platform independence**

Multiple user environments result in multiple computer platform environments. For instance, in a typical remote visualisation system, the operating system of the server/workstation is Linux or UNIX. For local PC clients, the operating system may include Microsoft Windows, Linux, and Macintosh OS. In addition, a mobile device client usually uses Symbian, Palm OS, or Windows Mobile as the operating system. The solution requires a good level of compatibility so that it can be used with a variety of computer platforms.

4. **Ad-hoc display support**

The solution should support various display devices in an ad-hoc way: that is, if a new device is used, the system should be able to detect the properties of the device “on the fly”, in a similar way to how “Plug n Play” works for USB devices. This feature means the end users can easily use different types of display devices in visualisation.

3.2 Objectives of the Project

The overall objective of this project is to develop a solution that allows a user to use a visualisation system with different types of display devices without explicit configuration of rendering services. In order to achieve this overall objective, there are several key objectives, as follows:

1. **Develop a solution which can support multiple display devices.**

   To support multiple display devices, the key task is to obtain the display device properties, and use those properties to configure the rendering so that the correct image is generated. Thus, a method which can detect the properties of the display device should be developed. In addition, the properties data should be described using a data format that can be transferred to the rendering software for display configuration. Therefore, the solution should implement a method to describe device properties and transfer the property information from client to server.

2. **Develop a solution which can support multiple simultaneous users.**

   The solution should be able to support multiple users, a stand-alone approach cannot satisfy this requirement, as it can only support a single user. A client-server structure allows many users to connect with one server. For this reason, a client-server based approach is appropriate for this project.

3. **Develop a solution which allows automatic configuration.**

   In order to obtain the correct visualisation result for the display device, the solution should be able to use the display properties to configure the rendering software automaticallyso that the render can generate the correct image.

4. **Develop a solution which can run on various platforms.**

   To satisfy different users, the solution should be able to run on various platforms, such as Linux, Windows series. Thus, a platform independent solution will be developed.

3.3 System Model

The above sections outline the requirements and objectives of this project. The proposed solution is a client-server based application. A simple structure of this design is presented in Figure 3-1.

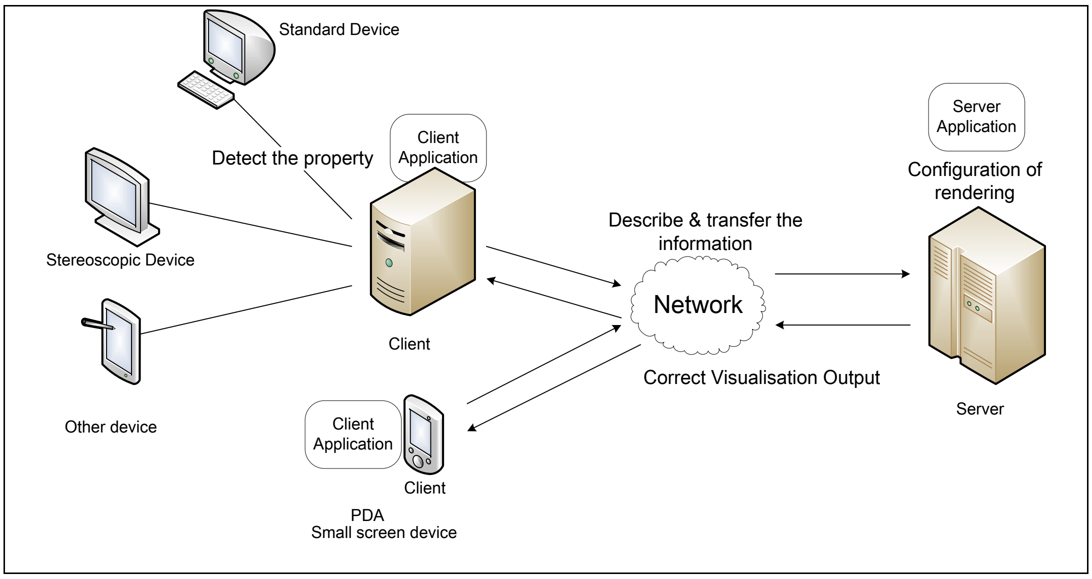


Figure 3-1 System model

In this system, the client application runs on a client machine to detect or gather the properties of the display device. After the properties are detected, the property data will be encoded and transferred to the server via the network. The server application receives the property data from a client and parses the data to extract the display properties used to configure the rendering software. Lastly, it renders an image based on the display properties and returns to the client application an image to display.

3.4 System Architecture

The system has two main parts: the client application and the server application.

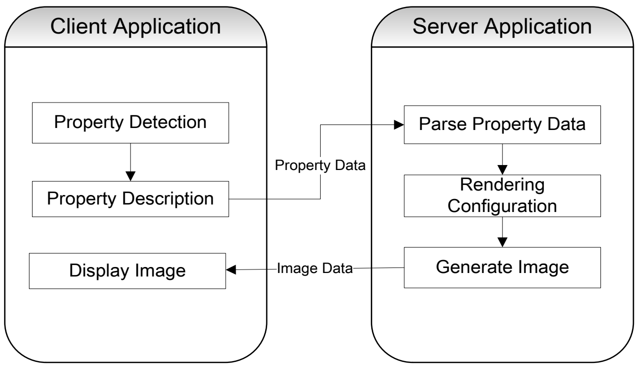


Figure 3-2 Flowchart of system

The main functions of this project are displayed in Figure 3-2. These are property detection, property description, property data parse, rendering configuration, and network communication.

3.4.1 Property detection

Property detection is one of the key functions in this project. In this project, the application should be able to support multiple display devices to obtain the correct visualisation output. To achieve this goal, the correct device property information must be found. There are two points that need to be considered when designing this function:

1. **Detect the properties automatically.**

   The reason for automatic property detection is to make the client application as easy to use as possible. Users do not need to setup many parameters on the client application in order to use it. The application will automatically detect the property information as much as possible.

2. **Allow user to manually enter or modify property information.**

In some case, users may use special display devices, such as stereoscopic displays. For special displays, the computer may not be able to automatically detect all property information from the display device. Some properties, such as eye separation, may not be stored in the device itself, so users will have to manually enter property information into the application. Another situation is that the application may detect incorrect properties from the display device. For example, the property information which is detected by the application is not the same as the value recorded in the device manual. Thus, the user may need to manually modify incorrect property information.

3.4.2 Property description

After the application obtains the device property information, we need to find a way to describe this data so that the property data can be stored and transferred easily. There are a number of data formats that can be used to describe the property data. The most appropriate data format should be chosen by considering the following points:

1. **Platform independence**

The application may run on different computer platforms, so that property data must be able to be transferred between different platforms correctly.

2. **Extensible for new device**

There are a number of different types of display devices, and we need to consider how to use an appropriate structure to store the property data for all devices. This structure should also be able to describe new types of device. When we need to describe new types of property information, the structure should be able to be easily extended.

3. Compact format

The data should be in a compact format so that data size can be minimised and the transfer time of the network reduced. The reason for this requirement is that the client application may be used on mobile device and the data may be transferred by mobile network connection.

3.4.3 Parse property data

After the server application receives the property data from the client application, it should be parsed by a data parse function. Thereby, the server application can receive the property information about the target display device.

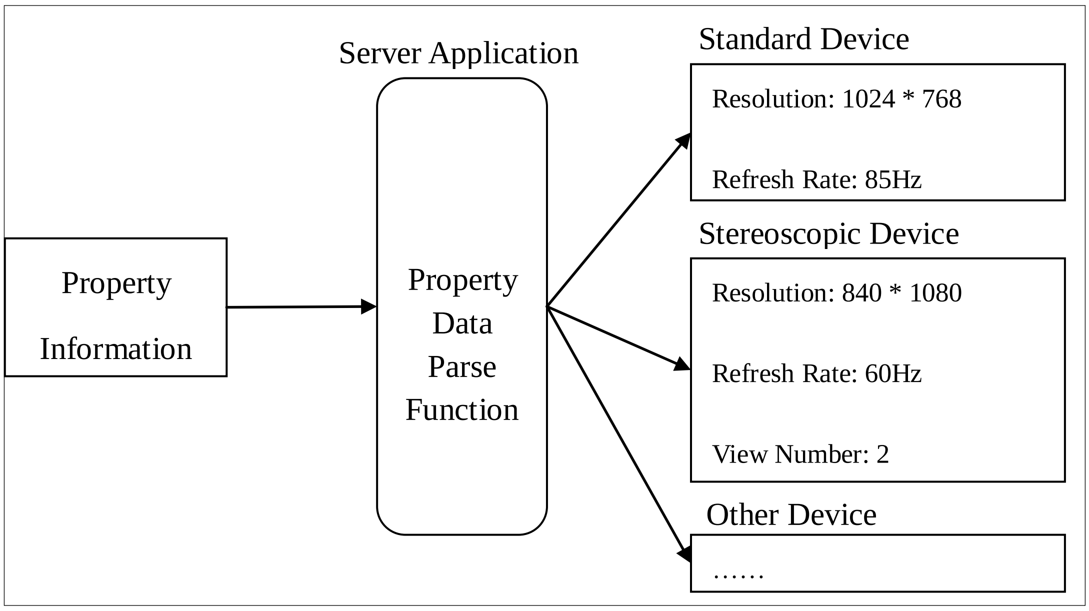


Figure 3-3 Property Data Parse Function

Figure 3-3 shows the output of the property data parse function. The property information is passed from client to the server. The server application will parse the property data so that the application can obtain the property information. After that, the property information will be transformed to parameters to configure the rendering system, such as the size of the windows, focal length.

3.4.4 Rendering system configuration

After the application parses the property data of the target display device, the server can obtain the rendering parameters. The application uses these parameters to configure the rendering system, such as change the size of the image, calculate and set the camera's position.

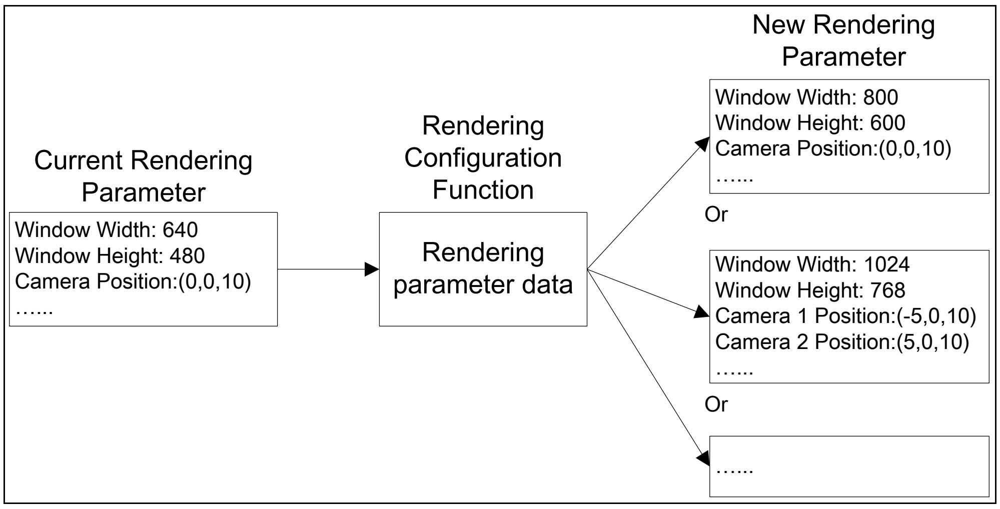


Figure 3-4Rendering reconfiguration function

Figure 3-4 gives an example showing the output from this function. In this example, the rendering parameters are window size 640x480 and camera position (0, 0, 10). This function will then change the parameters of the rendering system based on the parameter data which is passed by the property data parse function.

3.4.5 Network communication design

This project is to investigate the development of a client-server application. The client and server communicate via a network connection.

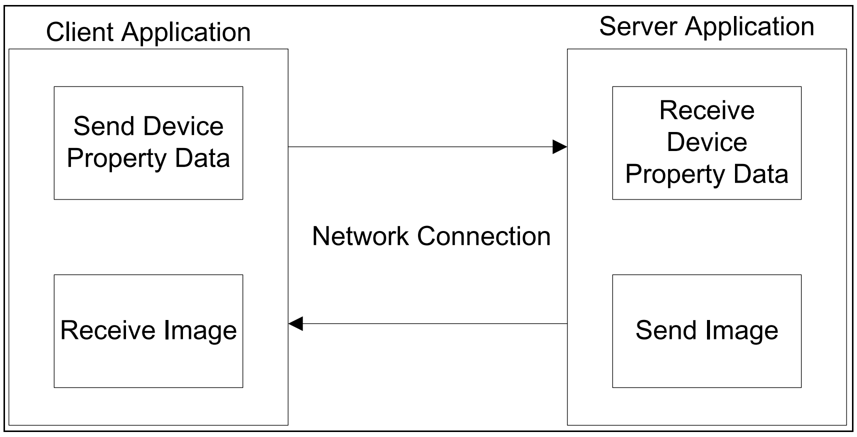


Figure 3-5Network communication

In this project, we focus on transfer of the property data from the client to the server application, and the final image from the server to the client application, see Figure 3-5. When designing the network communication part, the following requirements need to be considered:

1. **Reliable transfer**

   Correct display device property information is essential for rendering configuration. The data must be transferred correctly and completely when the client sends the property data to the server. Therefore, the network protocol for this project should provide a reliable transfer environment to ensure correct data transfer.

2. **Appropriate transfer efficiency**

   The properties and image data will be transferred via a network. The network communication protocol should have a low overhead to maintain the transfer efficiency.

3.5 User Perspective of Application Flow

This solution should easily be used by end users. Therefore, the application needs to be made so that it is as easy as possible to use. The users only need to provide a few inputs to obtain the appropriate output. Figure 3-6 gives a possible user perspective application flow.

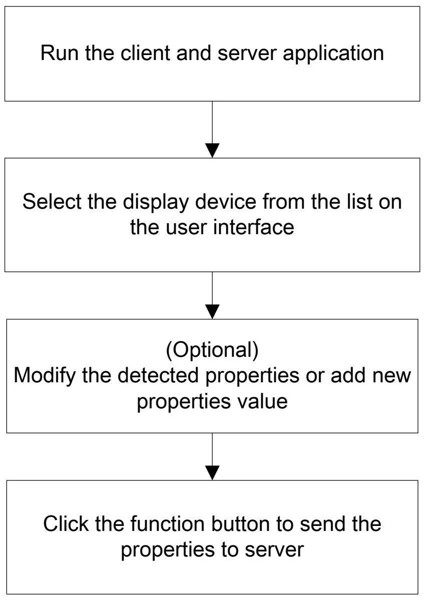

Run the client and server application
Select the display device from the list on the user interface
(Optional) Modify the detected properties or add new properties value
Click the function button to send the properties to server

Figure 3-6Logical structure of system

First, the users need to run the client and server applications. Second, the users need to select a display device from the available device list. Then the users can modify or add display properties if it is necessary. Lastly, the users simply need to click a button to create and send the profile, and then the server application will automatic configure the render. From this flow, we can see that end users only need to follow a few steps in order to configure the render for a display device.

3.6 Analysis and Design Summary

This chapter briefly introduces the objectives and how to design the application to achieve the requirements. The main functions of the client and server applications are discussed. The client application focuses on display device property detection and description, and the server application has responsibility for reconfiguration and generation of the final image.

Chapter 4 Implementation

This chapter discusses the implementation of the design described in chapter 3 including the technical choices made and the reasons for those decisions. This system is named ADAM, Ad-hoc Display Adaptation (for) Multiple (device). The chapter begins with an overview of ADAM and this is followed by a detailed description of each of its components.

4.1 System Overview

In chapter 3, the design chapter, the system model was introduced. The two main components of ADAM are: the client application and the server application. The client application is used to detect and describe the properties of the display device and display the resulting image returned by the server. The server application is used to parse the property data, configure the rendering software, generate the image and transfer it to the client. Communication between the client and server is via a network connection. The display device properties and the generated image will be transferred between the client and server applications. The logical structure of the system has been presented in Figure 3-2. This figure provides an overview of how the system is structured and at what stage communication occurs between the applications.

4.2 Programming Language

In order to implement the system, we need to select appropriate programming languages for the client and server applications. In the last chapter, we discussed the requirements of the project. One requirement is that the solution should have a good level of compatibility so that it is able to be used in variety computer platforms. Table 4-1 shows the common operating systems on different platforms.

Table 4-1 Operating system on different platforms

Platform	**Operating System**
SuperComputer	Linux, Unix
Apple Mac	OS X
PC	Windows XP, Windows Vista, Linux, Unix
PDA	Symbian, Windows Mobile

However, the requirements for the server and client applications are different. The user may use the client application on different platforms, but the server application normally runs on a server machine, where the platform is usually Linux, UNIX, or Windows. In addition, the user may use the client application on a mobile device, such as a PDA. Thus, the size of the client application should be kept small, but the size of the server application is not as important.

Based on the requirements, we compared the compatibility of different support platforms for common programming languages, as shown in Table 4-2.

Table 4-2Compatibility comparison of common program language

	VB.Net	C#	Java	C++
Windows Series platforms (98, XP, Vista)	Yes	Yes	Yes	Yes
Macintosh OS platform	No	No	Yes	Yes
Linux, UNIX platforms	No	No	Yes	Yes
Symbian platform	No	No	Yes	Yes
Windows Mobile platform	Yes	Yes	Yes	Yes

From this table, we can see that VB.net and C# language can support all Windows platforms; however, there is a low compatibility of these two languages for other platforms.

The C++ program can run on all platforms but it may not run across different platforms. In addition, it requires a different compiler for each platform. For example, a C++

program compiled for a Windows OS can only be used on a Windows platform, they cannot run on other platforms. If we need a C++ program for another platform, such as Linux, we need to use a compiler that can support Linux to create the program. This requires using a different C++ program for each platform.

Unlike C++, a Java application can run across multiple platforms. The most important character of the Java platform is that Java applications are typically compiled to bytecode that runs on any Java virtual machine (JVM) which itself is implemented for a variety of computer architectures. The Java platform provides various JVM versions for a platform. The user only needs to install the corresponding Java running environment for different platforms and then the Java applications are able to run on it. Therefore, we can use one Java application for all platforms. In addition, the Java platform provides a micro edition (Java ME) for the mobile device environment. Java ME has been widely used for creating applications for mobile devices, such as PDAs and cell phones. Accordingly, the Java programming language will be used to implement both the client and server applications.

4.3 Client Application Implementation

The client application is designed to detect device properties and display the resulting image. This section will discuss the implementation of the client application.

4.3.1 Client user interface

In order to allow users to use the functions of the client application, we used a GUI including lists, buttons, text fields, and combo boxes to implement the client application user interface. Figure 4-1 shows a screenshot of the client user interface.

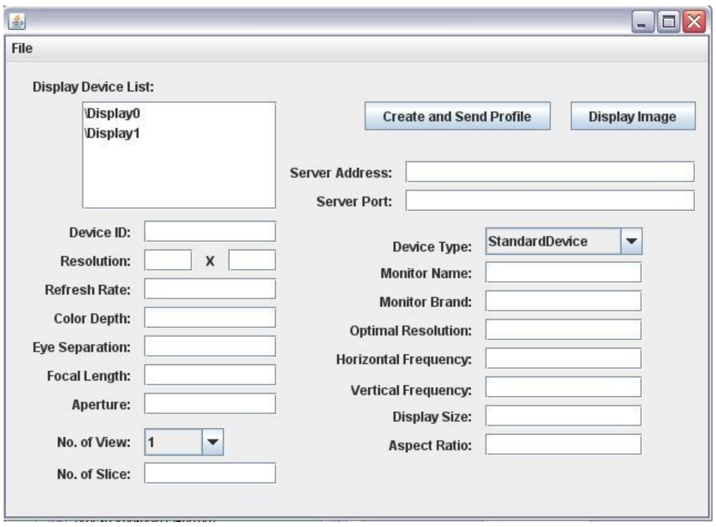


Figure 4-1 The client application user interface

The Display Device List includes all available display devices which are connected to the client machine. The users need to select a device from this list in order to collect the display properties. The text fields are used to display the properties and also allow the users to enter and modify the properties. There are two buttons on the interface, the Create and Send Profile button is used to generate the device profile and send it to the server application. After the resulting image is sent back, the image will be displayed in a new window automatically. In addition, the users can use the Display Image button to display the resulting image at any time.

The common screen size of PDA devices is 2” to 4”, and the resolution is much lower than the desktop and laptop device. Hence, compared with desktop and laptop displays, the small screen has a limitation in that it cannot display a large, complex user interface. In this project, we developed a client application user interface. This user interface works correctly on desktop and laptop displays. However, the interface is too big for small screen devices and the contents of the interface may not be displayed correctly. In addition, the required display properties for the small screen device may not be the same as the desktop displays. For the desktop system, the users may use different types of display devices, such as a standard device and stereoscopic device. This means that the client application must provide the functions to allow the users to input the corresponding display properties. For the small screen device, most use 2D screen and cannot extend to connect with other display devices. Furthermore, the small screen usually is a standard LCD screen, and in section 2.2.1 we noted that the most important display property for the standard LCD screen is resolution. Hence, the user interface for the PDAs' client needs to contain the basic display property such as the resolution. PDAs may have a stereoscopic screen. For those special PDAs, we need to consider entering extra display properties.

The desktop or laptop system may connect with multiple displays, so we need to provide a device list on the client application user interface to allow the users to select the connected device. PDAs, however, have only one screen. Hence, we can remove the device list from the user interface to make it fit more easily within the small screen. Some of the latest PDAs have a built-in projector, thus the PDA can be used as a portable projector. For this kind of device, we need consider how the application can support multiple displays.

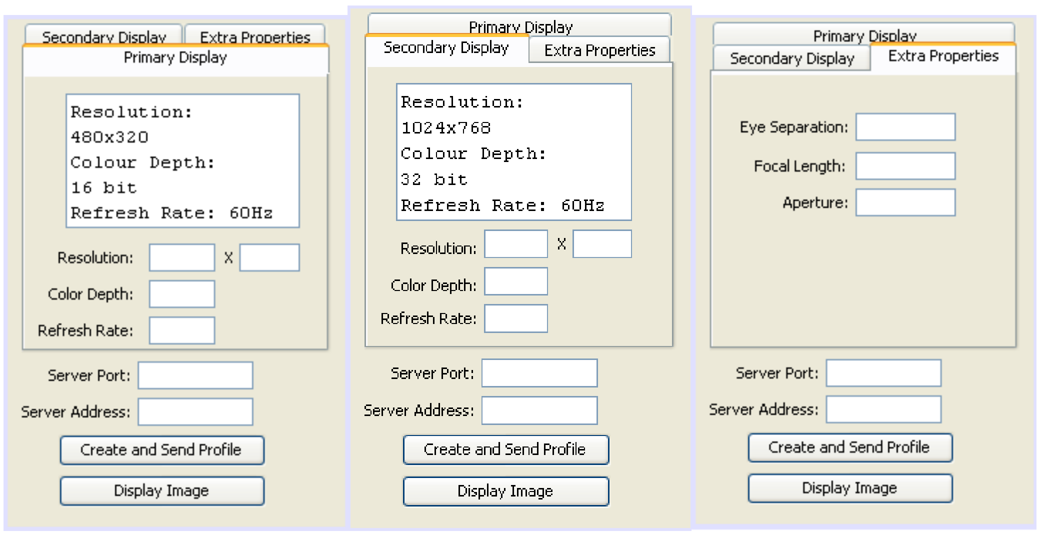

```
Resolution:
480x320
Colour Depth:
16 bit
Refresh Rate: 60Hz
```
```
Resolution:
1024x768
Colour Depth:
32 bit
Refresh Rate: 60Hz
```

Figure 4-2 Prototype client application interface for small screen device

Figure 4-2 gives a prototype client application interface design for a small screen device. The interface provides the basic display properties for the current device. The users can modify the properties if there is an error. The application also provides more options to help the users to support some extra or special displays. Compared with the desktop version of the user interface, this one only provides the essential properties and functions for the small screen device. The size of this interface can easily fit within a common PDA device screen.

4.3.2 Property detection

The first step of the system process is to detect the properties of the display device. A detailed implementation of this function will be presented in this section.

Before we discuss the implementation, we need to decide which display properties need to be found.

1. Basic display properties

Generally, there are three basic properties for every display device. They are the resolution, the refresh rate, and colour depth or bit depth. The resolution is the number of pixels contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and the number on the vertical axis. To display the appropriate image on a device, the correct resolution is necessary. The refresh rate is the number of times a display's image is repainted or refreshed per second. An incorrect refresh rate will cause a blank display, or the image may become distorted. The colour depth, also called bit depth, is the number of distinct colours that can be represented by a pixel. The higher colour depth can show truer colours. If the colour depth is too low, the image will lose colour and become distorted. Therefore, the basic display properties must be correctly found by the client application.

2. Special display properties

Unlike the standard display device, in addition to the basic display properties, special devices need other property information to display the image correctly.

A stereoscopic display device can present the image with a 3D effect on a 2D screen. This kind of display requires three special properties: eye separation, view distance, and aperture. These properties are used to configure the rendering software and to generate the stereo image.

Volumetric 3D display devices can present real, volume-filling 3D images. The resolution of this device has three dimensions so that we also need a slice number property for volumetric displays.

In addition to the above display properties, many devices also have properties, such as the manufacturer and the display size. This is called the extended display identification data (EDID). The EDID Standard (VESA, 2007) is created by VESA[1] to support Plug n Play. It defines a data structure used to carry configuration information for optimal use of a display. The EDID data structure resides in the display device and is stored in 128 byte blocks.

Table 4-3Basic structure of EDID data

EDID data	Description
Header	Exact match of defined header data
Vendor/Product ID	Product information such as manufacture, product ID, serial number
EDID Structure Version	EDID version number
Basic Display Parameters and Features	Display parameters such as maximum

[1] VESA, the Video Electronics Standards Association, is an international non-profit organisation representing hardware, software, PC, display and component manufacturers, cable and telephone companies, and service providers.

	horizontal and vertical image size.

Table 4-3 presents the basic structure of EDID data. The detailed EDID data structure is described in the EDID implementation guide, which is provided by VESA. This guide document provides guidance for both writing and interpreting EDID data. Table 4-4shows a summary of the EDID data reported by an Envision EN-775e monitor:

Table 4-4 Example of EDID data

EDID Data	**Information**	
Monitor Name	EnVision EN-775e	
Monitor ID	EPID775	
Model	EN-775e	
Manufacture Date	Week 26 / 2002	
Serial Number	1226764172	
Max. Visible Display Size	32 cm × 24 cm (15.7 in)	
Picture Aspect Ratio	4:3	
Horizontal Frequency	30–72 kHz	
Vertical Frequency	50–160 Hz	
Maximum Resolution	1280×1024	
Gamma	2.20	
DPMS Mode Support	Active-Off	
Supported Video Modes:	640×480	140 Hz
	800×600	110 Hz
	1024×768	85 Hz
	1152×864	75 Hz
	1280×1024	65 Hz
Monitor Manufacturer	Envision, Inc.	

The EDID properties identify and describe capabilities of a display device. Compared with basic display properties, EDID properties are optional display properties for the rendering configuration.

Table 4-5Required display properties for correct image display

	Standard Display Device	**Stereoscopic Display Device**	**Volumetric 3D Display Device**
Resolution	Yes	Yes	Yes
Colour Depth	Yes	Yes	Yes

Refresh Rate	Yes	Yes	Yes
Eye Separation	No	Yes	No
View Distance	No	Yes	No
Aperture	No	Yes	No
Slice Number	No	No	Yes
EDID data	Optional	Optional	Optional

Table 4-5 lists the required display properties for different types of device. The basic properties, such as resolution, colour depth, and refresh rate, are key properties for every display device. For stereoscopic devices, the eye separation, view distance, and aperture properties are essential, and for the volumetric display, the slice number is essential. In addition, EDID data can help the system identify the display device, but it is not the decisive factor for the resulting image display.

4.3.3 Detection of display properties in Java

Generally, a display device is able to support several modes of resolution, refresh rate, and colour depth. We need to detect the current properties of the device, such as the current resolution, to configure the renderer so that the correct image can be generated. To obtain the current property information, we need use a computer program to detect and return the properties. The Java language provides an interface to help a developer obtain the display device information. In this case, we need use *GraphicsEnvironment*,*GraphicsDevice* and *DisplayMode* classes to implement the detection function. The *GraphicsEnvironment*class describes the collection of *GraphicsDevice*objects available to a Java application. The *GraphicsDevice*class describes the available graphics devices in a particular graphics environment. The *DisplayMode*class encapsulates the current resolution, the colour depth, and refresh rate of an available *GraphicsDevice*. Therefore, in order to implement a detection function, we need create a new *GraphicsEnvironment*first, and then find all available graphics devices in this collection. After that, we can obtain the display properties for each graphics device. The properties will be displayed after detection,as show inFigure 4-3.

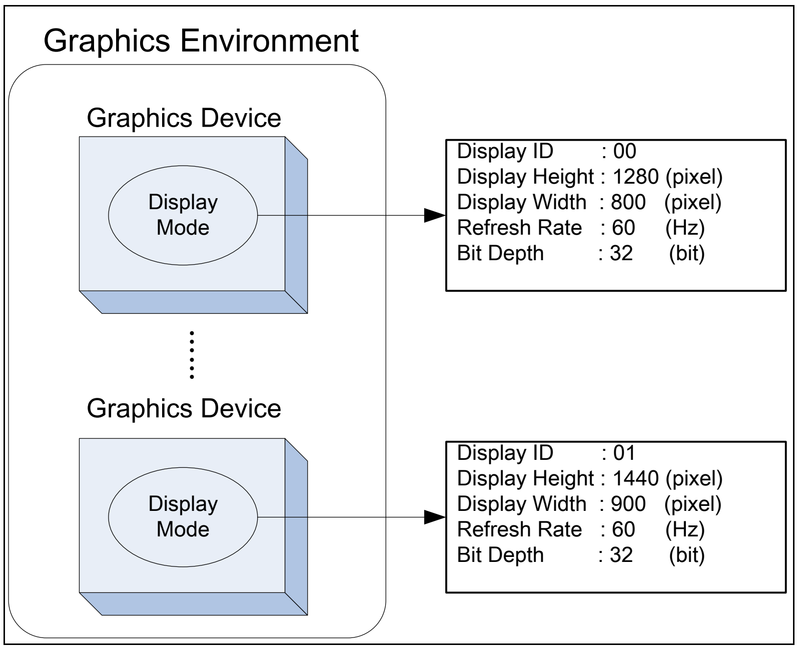
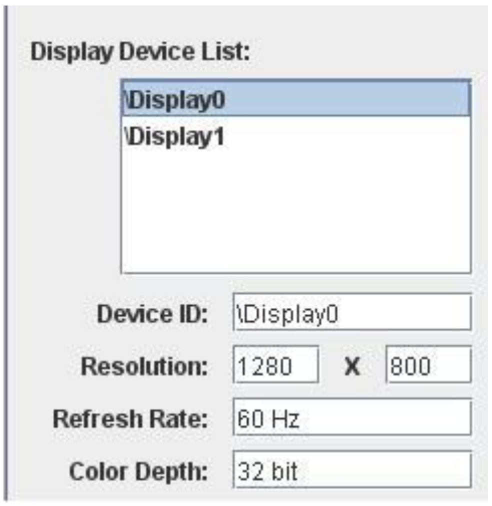


Figure 4-3 Graphics environment structure in Java

When the client application starts, the detection function will be called on automatically. In a multiple screen environment, all available devices will be displayed in a list and the user can select a device and view the corresponding properties as shown in Figure 4-4.


Display Device List:

\Display0
\Display1

Device ID: \Display0
Resolution: 1280 X 800
Refresh Rate: 60 Hz
Color Depth: 32 bit

Figure 4-4 Detect properties in multiple devices environment

Many solutions have been developed to detect the EDID information. The Nicomsoft Company provides an API called WinI2C/DDC, which allows a user to read EDID data from the display device. To use this API, we need the client application to call the WinI2C/DDC dll file. The Java platform cannot call WinI2C/DDC dll file directly, we must design a new dll file called Project DLL to wrap it. Figure 4-5 presents the process for reading EDID data from the displays.

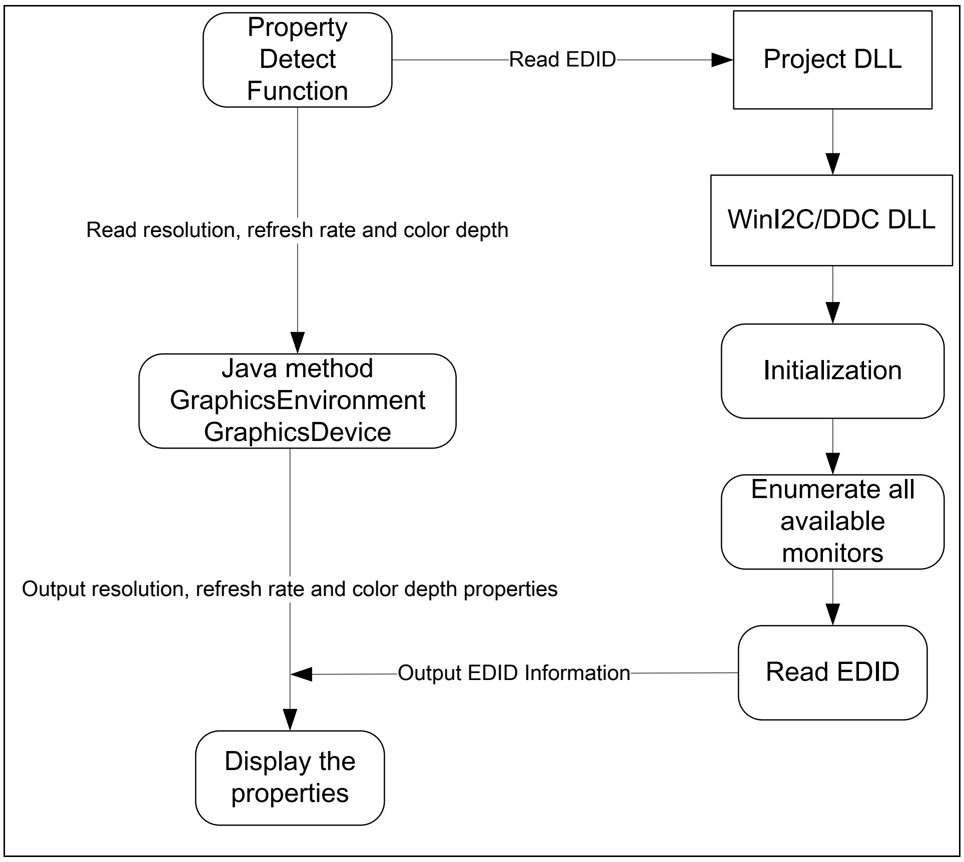

Property Detect Function
Read EDID
Project DLL
WinI2C/DDC DLL
Read resolution, refresh rate and color depth
Java method GraphicsEnvironment GraphicsDevice
Initialization
Enumerate all available monitors
Output resolution, refresh rate and color depth properties
Output EDID Information
Read EDID
Display the properties

Figure 4-5 Read EDID information flow

The client application starts the property detection function. This function will use a Java method to return the basic properties immediately, meanwhile the function calls the Project DLL file to detect and return EDID information.

Although WinI2C/DDC API provides a powerful ability to read the EDID information, it is purchased software, so this may increase the cost of the system. There is another open-source EDID application, read-edid, which was also evaluated. However, this application did not provide an interface to the developer. By using this application, the program will take more time to be implemented. Currently, we use a trial version of WinI2C/DDC API to implement the read-edid data method. For future development, the open-source application may be considered.

4.3.4 Manual entry of other properties

Unlike the basic display properties, for special displays, such as the stereoscopic display device, the Java application may not be able to automatically detect the properties such as eye separation and view distance. Therefore, users have to manually enter this information. Another possibility is that the application may detect incorrect properties, thus the user needs to be able to manually modify incorrect properties.

In this project, we use the *TextField* control in the Java platform to display property information. This allows the user to manually enter or modify the information. In addition, the application will evaluate the properties before generating the device profile. For example, if the user enters text information into the resolution property field, the application will give an error message that the field can only accept integer values. Some properties, such as resolution, are essential for every display device, thus the resolution field cannot remain empty if the user needs to generate the device profile. Another example is that of special displays,such as a stereoscopic device, where the eye separation and view distance are required fields. The system will check the required field before creating the profile, and if the user omits an essential property the application will send an error message. Figure 4-6 gives the process steps between property detection and creating a profile.

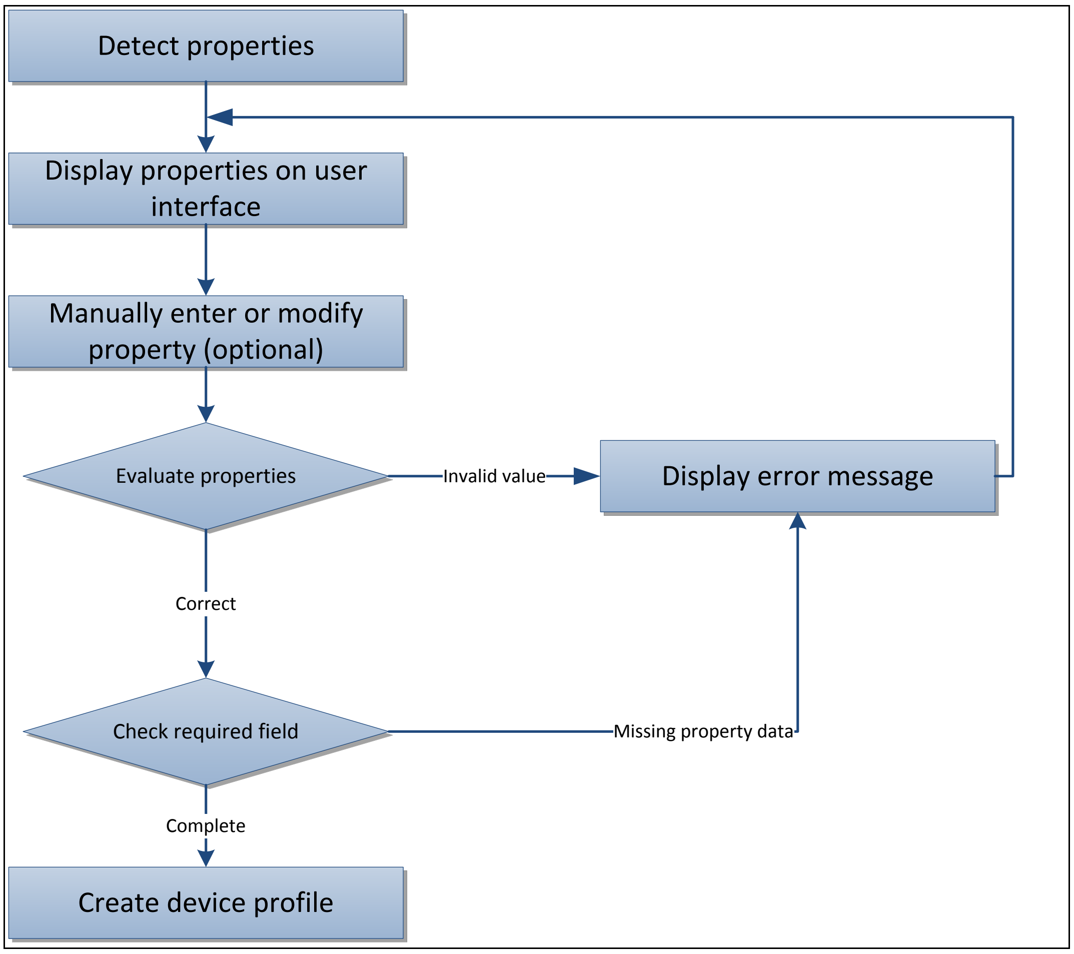


Figure 4-6 Process steps from detection to description

4.3.5 Property description function

Before we can use and transfer the display properties, the data need to be described in a format that can be stored and transferred easily. This application is designed to support a multiple user, multiplatform environment, and the three requirements of the data format which were discussed in the design chapter. These requirements are platform independence, extensible for new devices and compact format. The XML(W3C, 2008) format will be used as the data format to describe the display properties in this project. The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages. It is a simple and flexible text format language and supports Unicode, allowing almost any information in any written human readable language to be communicated. Thus, XML is widely used as the format for information storage and transfer between various platforms. Thus, it satisfies the requirement of platform independence. Furthermore, XML is classified as an extensible language because it allows its users to define their own elements, such as structure, field names, and specific values. This is very useful when we need to extend the element to describe a new type of display device. Nevertheless, there is a shortcoming for XML format. Compared with other data formats, such as binary, XML format is redundant and large. This disadvantage may affect application efficiency when storing, transmitting, and processing large data sets. In this project, however, we only use XML format to describe the display device property data, and the size of data will not be large and complex. For that reason, this limitation, relative to other advantages, is acceptable.

4.3.6 XML schema and file example

An XML schema is a description of a type of XML document and it provides a view of the structure of the document. In this case, the property information of a display device needs to be described using an XML format file. Therefore, it is necessary to create an XML schema to define the structure of this XML.

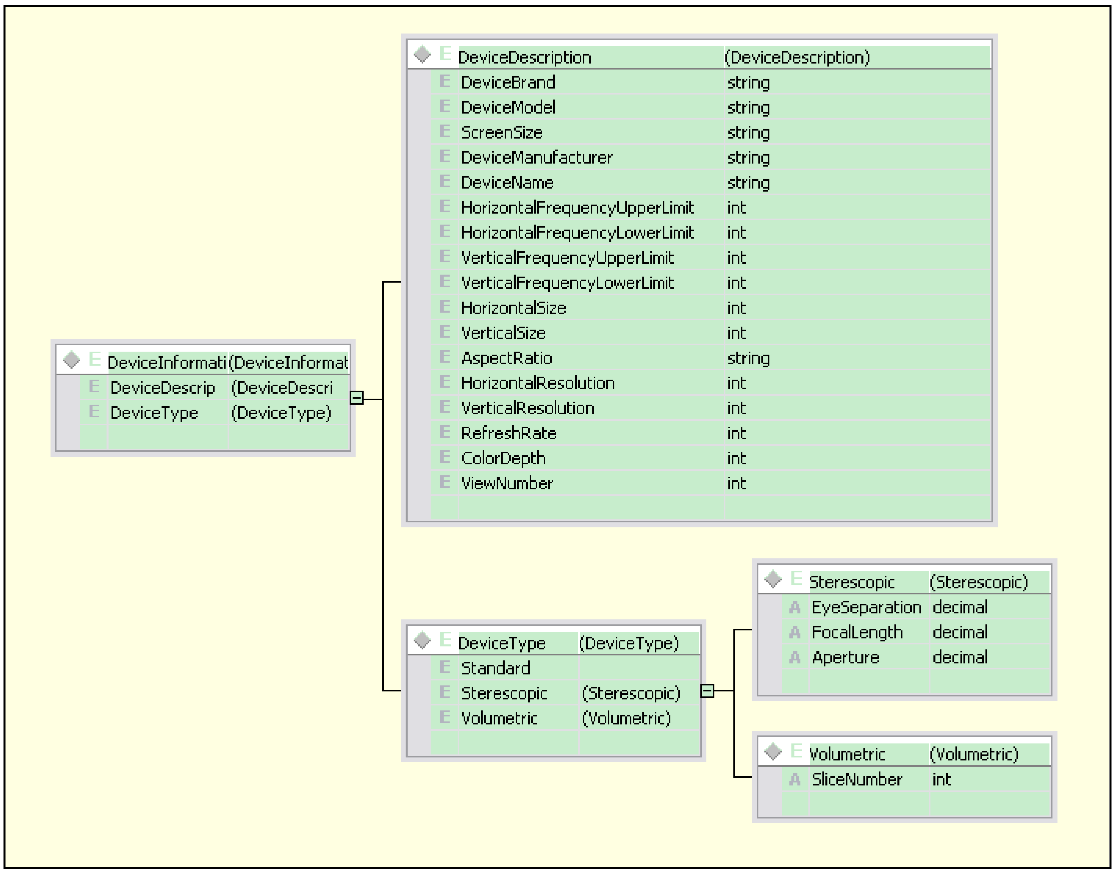


Figure 4-7 Device information XML schema

Figure 4-7 presents the structure of this XML schema. This schema is called a Device Information schema. It is used to describe and record the properties for a display device. The detailed code of this XML schema is provided in Appendix A.

The device properties are stored under two main categories, Device Description and Device Type. The Device Description category contains general information of the display device, such as brand and model. This category also records the hardware properties, such as resolution, colour depth, refresh rate, screen size, view number. The last category, Device Type, records the type of display device, with corresponding attributes. For the detailed content of these two categories, the reader should refer to the Appendix B.

```
<?xml version="1.0" encoding="ISO8859-1" ?>
 - <DisplayDevice>

    <DeviceDescription>
      <DeviceName>ViewSonic</DeviceName>
      <DeviceModel>E71f</DeviceModel>
      <DeviceScreenSize>24</DeviceScreenSize>
    </DeviceDescription>

    <HardwareDescription>
      <CurrentResolution>960*1200</CurrentResolution>
      <CurrentRefresh_Rate>80Hz</CurrentRefreshRate>
      <CurrentColor>32bit</CurrentColor>
      <ViewNumber>2</ViewNumber>
    </HardwareDescription>

    <DeviceType>
      <StereoscopicDevice EyeSeparation="0.3" FocalLength="2" Aperture="60">
    </DeviceType>

  </DisplayDevice>
```

Figure 4-8 Example of device profile

Figure 4-8provides an example of a device profile. This profile stores the properties of a stereoscopic display device. From this profile, we know the device information such as the brand and model. We also can get the display properties, such as the resolution, and eye separation, from this profile.

4.3.7 Create device profile

We need a method that can create an XML device profile by using the detected display properties. In this project, we used JDOM[2] as a toolkit to generate the XML file in Java. JDOM provides a method, *BuildXMLDoc()*, to generate an XML file. To create the file, we need to set the display property type as the *Element* first, such as resolution and refresh rate. The *Element* and structure of the file must be based on the XML schema. Then we pass the property value to the corresponding *Element*. After all properties are passed we used *XMLOutputter()* method to write the file to the disk.

[2] JDOM is a Java-based "document object model" for XML files. JDOM provides a way to represent that document for easy and efficient reading, manipulation, and writing.

4.4 Network Communication Implementation

In chapter 2, we introduced different remote visualisation approaches. These approaches each use a different network protocol to connect the client and server. Table 4-6 lists the approaches and corresponding network protocol.

Table 4-6 Network protocol of remote visualisation approaches

Approach Name	Protocol	Function
Resource Aware Visualisation Environment	TCP	Direct transfer of live data or continuous stream
	FTP	Transfer data file
Remote Visualisation over Internet	HTTP	Transfer web client application
	CORBA bus	Transfer commands and image buffer
CT/MRI Remote Visualisation	FTP	Transfer raw CT/MRI data
Remote 3D Visualisation using Image-Streaming Techniques	UDP	Transfer image buffer data
	CORBA bus	Transfer commands and events

From Table 4-6 we note that the CORBA bus is usually used to transfer commands and event information. The CORBA (Common Object Request Broker Architecture) technique works more like an architectural rather than a network protocol. However, in this project we need to focus on data transfer and other protocols, such as FTP, UDP and TCP, which are commonly used to transfer the data.

The FTP protocol is usually used to transfer data files, the transfer reliability of this protocol is good. However, the overhead of FTP consists of commands to log in, the user name and password, specify the files or directories to upload or download, and what ports need to be used. As a result, the overhead of FTP is large and that may increase the transfer time. In addition, the users need to use some information, such as the server address, server port, user name, password, directory and file name, in order to transfer the file between the client and server application. This situation requires the users to make more inputs to use the applications and may increase the probability of misoperation, such as giving the incorrect password or file name.

Unlike FTP, UDP and TCP allow a network connection of the client and server application without much extra information.

The advantage of UDP protocol is that the overhead is small and that can reduce the transfer time. However, a big disadvantage for this protocol is that a data packet may not be delivered. In addition, UDP gives no indication of the packet missing so the application at the receiver side must includes checking method for this. This will increase the working time.

TCP is a reliable network protocol, it has a significant advantage in that it can ensure all data packets are delivered. But this feature adds an overhead, because every packet will receive a reply or acknowledgement from the receiver, this can slow down the transmission time.

In this project, the device property data file is important, since the rendering reconfiguration function may not work correctly if we lose property data. However, the UDP protocol has some risk of data loss, so we use TCP protocol to implement the network communication. In addition, we can use a built-in library with the Java platform to implement the TCP connection without any extra toolkit or library.

Implementation of a TCP connection requires two parts: the client side and server side. To implement communication between the client and the server, the server application needs to create a listening socket on a TCP port and wait for the client to connect. Once a client connects to the port, the server accepts the connection and starts to receive and respond to the data with the client. The following sections explain how each part works in this project.

4.4.1 Data transfer and receive function on client side

This function is used to send the device profile to the server application and receive the resulting image from the server. For the transfer and receipt of data, the client needs to enter the IP address and port of the server on the user interface.

After the client obtains the server address and port, the following steps are used to create the connection.

1. Create a new socket

We use the *Socket(address, port)* function to create a new socket to connect with server application.

2. Send the profile via the socket

Java provides the *OutputStream()* function to transfer the data to the server side via the socket. When the *OutputStream()* process is finished, the profile is transferred. The socket begins to listen and waits for the data from server side.

3. Receive the data from server

When the server application begins to send the image file,the*InputStream()* function is used to receive data from the socket.

4.4.2 Data receive and transfer function on server side

The goal of this function is to receive the device profile from the client and return the final image to the client. For implementation of this function, the server needs to create a new listening socket on a port and wait for the client to connect. When the client is connected, the server begins to receive and transfer the data. In this project, the following steps are used to program a Java application using the corresponding functions.

1. Create a new server socket

We use the function *ServerSocket(int port)* to create a server socket and bind it to the specified port. The parameter *port* defines the port number.

2. Listen for the client

In the Java platform, we need to use an infinite loop to achieve this step. An example is presented below:

```
While (true)
{
Socket Socket = ServerSocket.accept();
}
```

In this loop the *ServerSocket.accept()* function listens for the client.

3. Accept a client connection

In order to accept the connection of a client, the *ServerSocket.accept()* function is used. This function listens for a connection on a particular port and returns a new socket.

The server application should be able to support multiple clients. In order to implement this capacity, we use an array to store all connected sockets, and add each new socket into the array. Thus, the server application can get and process each socket from the array based on the connection order.

4. Receive and respond request

After accepting a connection and getting a socket, the server should be able to receive the data from the socket by using *save()* function and then call the other functions to generate the resulting image for this display device. After that, the image will be returned to the client application to display.

4.5 Server Application Implementation

This section will discuss the implementation of the server application and corresponding functions.

4.5.1 Overview of server application and functions

The server application is designed to process the display device properties profile, configure the rendering software, generate and return the resulting image to the client side.

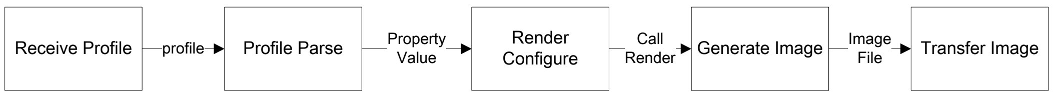


Figure 4-9 Process steps of server application

Figure 4-9 presents the process steps of the server application and the functions run by the server. The implementations of the data receive and transfer functions are discussed in the network section. The detailed implementation of the other functions will be introduced in the following sections.

4.5.2 Render configuration

The main function of the server application is the render configuration function. This function is the key function that allows the application to support various types of displays. The server application will configure the renderer based on different display properties so that a particular display device will obtain the correct visualisation output. In this section, we will discuss rendering software and introduce the implementation of the profile parse and render configure function.

4.5.2.1 Render software

There are a number of render solutions that are used in visualisation, such as POV-Ray, OpenGL, and VTK. In this project, we focus on the rendering configuration, to obtain the correct image output. Therefore, we need to select a render solution which can easily be used to generate and output the image. There are some requirements that need to be considered in choosing the renderer:

- Java support, this application is created using the Java language so that the render solution should be able to support the Java platform.
- The render software should output the image file directly after rendering has finished.

Table 4-7 Render solution comparison

	POV-Ray	OpenGL	VTK
Java Support	Yes	Yes	Yes
Stand-alone render application	Provides its own render application. Allows user to render the POV-Ray image	Need to use other programming language platform to use it.	Need to use other program language platform to use it.
Image File Output	Support to output image file to the disk automatically after rendering is finished.	Users have to add functions into the application to output image file to disk.	Users have to add functions into the application to output image file to disk.

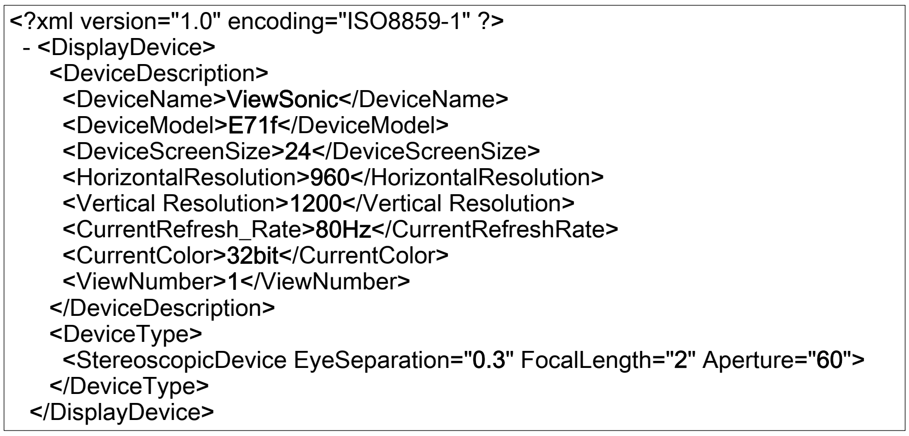
Table 4-7 compares possible render applications. All render solutions support Java. However, the configuration of the OpenGL and VTK environments are not straightforward to set up. Before the developer can use these two render toolkits, they need to configure the render toolkit with a particular programming platform such VB, C++, or Java. After that, developers can use the same programming platform as the compiler to use the render toolkits.

POV-Ray provides its own stand-alone render application so that users can use any program language to call POV-Ray to render an image directly. In addition, after POV-Ray finishes rendering, the application can output the final image file to the disk immediately. Hence, we chose POV-Ray as the rendering software in this project.

4.5.2.2 Profile parse and storage

The display device profile stores the properties of a device and is described in an XML format. The profile is transferred from client to server. The server application should be able to parses the XML profile so that the application can use the property information for further processing. In order to achieve this goal, we use the JDOM toolkit to implement the parse function.

The JDOM toolkit provides APIs that can parse an XML file easily. The following is a simple example to explain how JDOM is used to parse an XML file, in this project.



```
<?xml version="1.0" encoding="ISO8859-1" ?>
 - <DisplayDevice>
     <DeviceDescription>
       <DeviceName>ViewSonic</DeviceName>
       <DeviceModel>E71f</DeviceModel>
       <DeviceScreenSize>24</DeviceScreenSize>
       <HorizontalResolution>960</HorizontalResolution>
       <Vertical Resolution>1200</Vertical Resolution>
       <CurrentRefresh_Rate>80Hz</CurrentRefreshRate>
       <CurrentColor>32bit</CurrentColor>
       <ViewNumber>1</ViewNumber>
     </DeviceDescription>
     <DeviceType>
       <StereoscopicDevice EyeSeparation="0.3" FocalLength="2" Aperture="60">
     </DeviceType>
  </DisplayDevice>
```

Figure 4-10 Stereoscopic display device XML profile

Figure 4-10 shows an example XML profile, this profile records properties of a stereoscopic display device. We use JDOM to parse this file and obtain the properties, resolution, eye separation, focal length, and aperture. The following steps show how to use JDOM to obtain the properties from this profile:

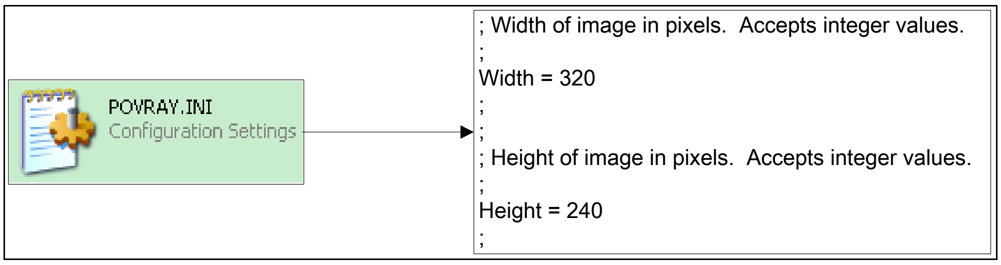
First, we need create variables such as resolution and aperture to store property values. Then we use the *Document doc=builder.build(name)*function to obtain the XML profile, property *name* is the profile name and location. After that, the *getRootElement()* function is used to take the root element. In this example, the root element is “DisplayDevice”, as shown in Figure 4-10. Then, we use a loop to find each property value such as Horizontal Resolution, Eye Separation, and pass the values to corresponding variables.

4.5.2.3 Render configure

In POV-Ray, there are several ways to use property value to configure the render. When POV-Ray starts the rendering process, the program looks for the configuration settings. We can either store the properties in an INI file, or POV-Ray also provides another option, to use the command line to configure the render. Therefore, there are two options we can choose when we need to use property values to configure a render:

1. Use POVRAY.INI file

There is a special INI file in the POV-Ray’s render directory called POVRAY.INI. This file contains configuration information about the render such as the width and height of the image as shown in Figure 4-11. We can store display device properties in the POVRAY.INI file. Whenever we start up POV-Ray to render an image for a particular display device, we need to store the current properties in the file, to override any previous settings.



```
; Width of image in pixels.  Accepts integer values.
;
Width = 320
;
;
; Height of image in pixels.  Accepts integer values.
;
Height = 240
;
```

Figure 4-11 POVRAY. INI file

2. Command line option

This option allows the user to use the command line to configure the render directly. When we use POV-Ray to render an image, we can enter the device properties on the command line. The property value in the command line will override corresponding values in the POVRAY.INI file. For example, we can use *+w320* and *+h240* commands to set the resolution of the render to 320 by 240 instead of using the INI file.

Comparing these two methods, the command line solution offers the following advantages:

- Flexibility for changing the render properties

Using the command line to change the render properties is straightforward. The POV-Ray command line option allows users to set any render properties by using the command.

- Reduces the complexity of the server application implementation

By using the POVRAY.INI file to change the render parameters we need several steps: open the file, parse the content, rewrite the properties, and save the new file. After that, POV-Ray will generate the image with the new render properties. To implement the steps, we need to create methods and functions for the server application. This will increase the complexity of the server application implementation. Using the command line, however, we only need one method to achieve the same requirement.

4.5.3 Generate image

After the render is configured, we need to generate the image. Generally, POV-Ray provides two ways to generate an image, using a GUI or command line.

1. Generate an image using aGUI.

POV-Ray provides a compiler with a graphical user interface (GUI). By using this compiler, the developer can easily process and design the scene. Figure 4-12 gives an example showing how to use GUI to generate an image.

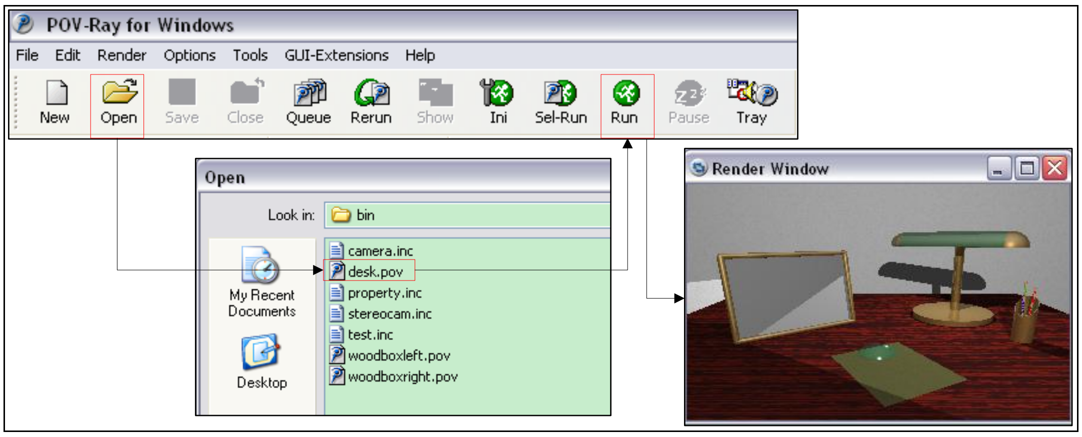


Figure 4-12 Image generation in typical way

In the POV-Ray render system, a POV-Ray scene is recorded as a *pov* file. This file stores all scene data such as the shape, size, and position of the object, the type and position of the light source and so on. Thus, we need a *pov* file to generate an image usingthe POV-Ray render system.

- First, we need to choose a *pov* file and then open it.
- Second, we call the run function to render this scene file.

- Lastly, we can see the resulting image in the render window and the system will store the scene as a bitmap file.

2. Generate an image using the command line

The POV-Ray program provides a set of command line switches that are used to set the options of POV-Ray. Thus, we can use the command line to achieve the same objective as the GUI. Figure 4-13 shows the syntax of the user command line to call POV-Ray to generate an image, and gives an example to display of how to use a command line to render a scene.

```
:\\POV-Ray application     \\function   \\pov file       \\other options
```

```
povray.exe   /render   desk.pov   +w640   +h480   /exit
```

Figure 4-13 Image generation in command line

In this example, we need to call the POV-Ray application first. Then we specify the function, in this case we use the *render* function, and specify the *pov* file used to generate the scene. Lastly, we give the resolution of the image and use the*exit* function to terminate the POV-Ray application after the rendering is finished.

The advantage of using the GUI is that when the users use it to process the POV-Ray scene, it is convenient and they can see the resulting image instantly. However, in this project we need to use the server application to configure and render to obtain the resulting image automatically. Thus, the GUI method is not an appropriate solution for this project.

On the other hand, by using command line option, we can configurethe render and generate the image at the same time. The implementation of the command line method for the server application is straightforward.

4.5.3.1 Generate stereoscopic pairs

For some special display devices, such as a stereoscopic display, we need to generate the stereo pairs image separately and combine the pair to obtain the final image. In chapter 2, we discussed how to calculate the camera parameters for stereo pairs. In POV-Ray, we developed a camera .inc file to do the calculation of the left and right camera parameters. The camera .inc file should be included in a *pov* file, so that when this *pov* file is rendered, the camera parameters can be loaded automatically. We developed a method that can modify the content of the *pov* file in order to include the camera .inc file when the device type is stereoscopic. In addition, we need to call the POV-Ray application twice to generate the left eye image and right eye image separately. Figure 4-14 shows the steps for generating the stereoscopic pairs.

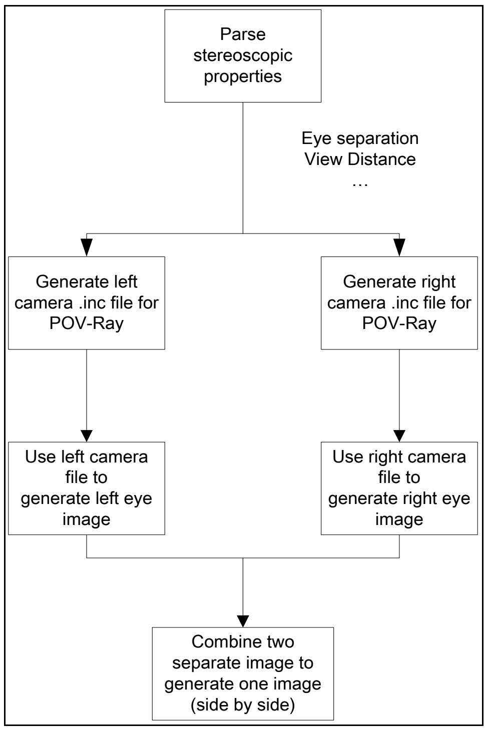

Parse stereoscopic properties
Eye separation
View Distance
…
Generate left camera .inc file for POV-Ray
Generate right camera .inc file for POV-Ray
Use left camera file to generate left eye image
Use right camera file to generate right eye image
Combine two separate image to generate one image (side by side)

Figure 4-14 Stereoscopic pairs generation steps

The device profile records the type of device. We know a device is either a standard display or a stereoscopic display after parsing the device profile. If it is a stereoscopic display, then the server application starts to use the stereoscopic display properties, such as eye separation and view distance, to calculate the left and right camera parameters and create two camera .inc files. Then the server application will call POV-Ray twice to generate the left and right eye images separately. When the individual images have been generated, the server application should combine the two separate images to one side-by-side image. We created a method to combine the pictures. First, we passed these two images into the function in this order: left eye image first and right eye image second. Then, we used the *BufferedImage()* function to read each image. Lastly, we generated a new combined image by using the *ImageIO.write()* function and wrote this image as a bitmap file to a local disk, which means we can then transfer this image file to the client.

4.5.4 Call POV-Ray in Java

In these last two sections, we discuss how to use the command line option to configure the renderer and generate the image in the POV-Ray system. This section describes how Java was used to implement the methods.

For Java to call POV-Ray, Java provides a class called *Runtime*. It provides a method *getRuntime.exec()*which allows the user to call a standard application with a command line.

```
Runtime.getRuntime().exec("/povray.exe  /render  desk.pov +width +height /exit")
```

Figure 4-15 Call POV-Ray in Java

Figure 4-15provides an example showing how to use this method to call POV-Ray. In this example, we call POV-Ray to render the desk.pov file and configure the render with specific size. After the image has been generated, POV-Ray will output the image as a bitmap file to local disk, normally in the same place as where the POV-Ray application is stored.

After the resulting image has been generated, the server application starts to transfer the image to the client application. For the standard device, when the image has been created, the server application will load the image to the *Socket* which connects with the client application by using the *OutputStream()* function. For the special displays such as the stereoscopic device, the server application will wait while the final image is combined, and then start to transfer the combined image to the client application.

4.6 Summary

This chapter presents the techniques that have been used to implement the client and the server applications. It discusses different program platforms in section 4.2, essential display properties and client application function implementation in section 4.3, the network protocol and connection in section 4.4, and the rendering software and the server application function implementation in section 4.5.

Chapter 5 Evaluation and Discussion

This chapter discusses the evaluation of ADAM, and examines the functions of the client and server applications. It discusses the successes and limitations of ADAM. First, the evaluation environment is introduced. The following sections then discuss the evaluations, and the limitations of ADAM. The chapter concludes with a summary of the findings.

5.1 Evaluation Environment

The requirements of this project are discussed in the design chapter. To evaluate the requirements, a number of tests were designed and performed. These tests are discussed in detail in section 5.2. In conducting the tests, a number of environments were used and these are discussed below. All tests were conducted during normal working hours and under normal network conditions. Four different evaluation environments were used, as in Table 5-1:

Table 5-1 Evaluation environment

Machine	**Platform**	**Monitor**	**Rendering Software**
PC desktop Processor: AMD 4600+ RAM: 2GB Hard Disk Capacity: 320GB	Windows Vista Home Premium	17” standard desktop monitor 20” wide screen desktop monitor	POV-Ray version 3.6
PC desktop Processor: AMD 4600+ RAM: 2GB Hard Disk Capacity: 320GB	Ubuntu version 9.04	17” standard desktop monitor 20” wide screen desktop monitor	POV-Ray version 3.6

Laptop Processor: Intel T2400 RAM: 2GB Hard Disk Capacity: 100GB	Windows XP SP3 version 2002	15.4” wide screen laptop monitor 17” stereoscopic desktop monitor.	POV-Ray version 3.6
Laptop Processor: Intel T2400 RAM: 2GB Hard Disk Capacity: 100GB	Windows Mobile 6.1 Emulator	15.4” wide screen laptop monitor	POV-Ray version 3.6

5.2 System Evaluation

To conduct the evaluation we developed three test types:

- Client Side tests
- Server Side tests
- Exceptional/Abnormal Condition tests

The client and server application tests were designed to exercise the critical functions of the solution. Sections5.2.2 and 5.2.3 present the details of these tests. The final part of the evaluation was to test the client and server applications in exceptional and abnormal conditions, such as network failure and software misconfiguration.

5.2.1 Platform independence

A key requirement of this project is that the system can be run on various platforms without reconfiguration so that it can support different users using different display devices. To test this capability, we ran ADAM on different operating systems before running the test ADAM on each operating system. Prior to running the test, we setup Java suite (version 1.5.0.09) and installed POV-Ray for each platform. Table 5-2 lists the outcomes of testing ADAM on different operating systems.

Table 5-2 Test ADAM on various platforms

Test Platform	Client Application	Server Application	POV-Ray
Windows XP SP3 version 2002	Working	Working	Working
Windows Vista Home Premium	Working	Working	Working
Ubuntu version 9.04	Working	Working	Not test
Windows Mobile 6.1(emulator)	Not Working	Working	Not Available

Normally, the server application runs on the desktop or server device. Thus, we only needed to test the server application on Windows desktop series and Ubuntu platforms; for the Windows Mobile platform we only needed to test the client application.

Table 5-2 shows the outcome of the test for ADAM on different platforms. The client and server applications can be used on Windows XP, Windows Vista, and Ubuntu platforms. However, the test of the client application on the Windows Mobile operating system failed. The following steps were used to test the client application on Windows Mobile platform. First, we ran the Windows Mobile emulator as it gave an emulator interface. Second, we used a Java running environment emulator application, MIDP Java Emulator version2.3, to load and run the client application. When we tried to run the client application an error message displayed, “The client application is not a valid Windows CE application”.

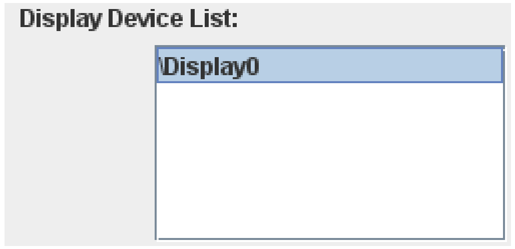
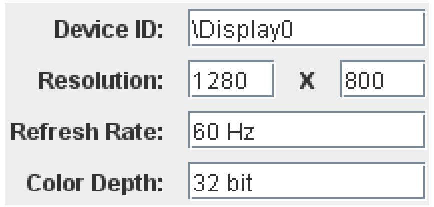
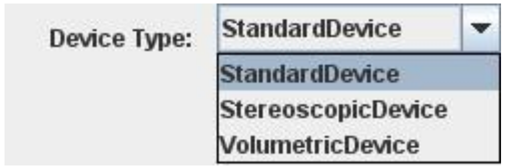
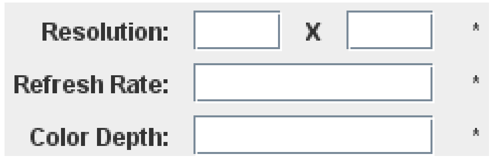
5.2.2 Client application evaluation

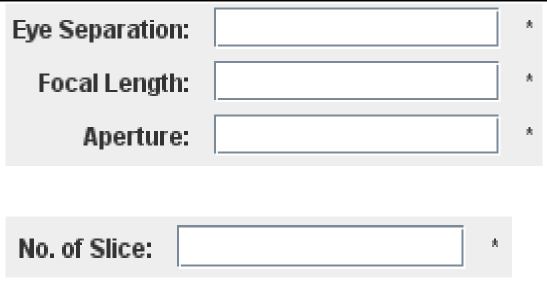
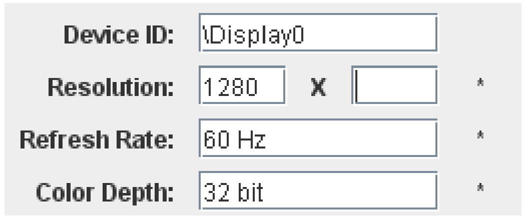
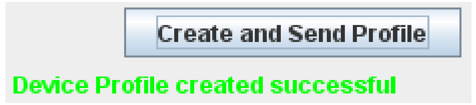
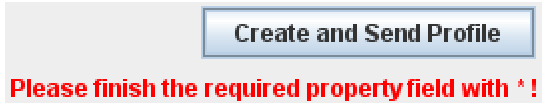
The aim of this evaluation was to test the client application. The main functions of the client application are:

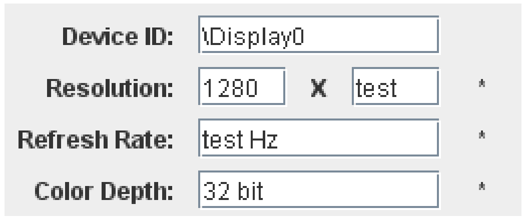
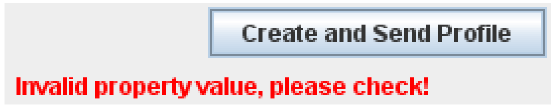
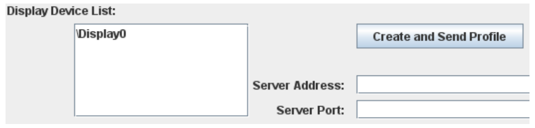
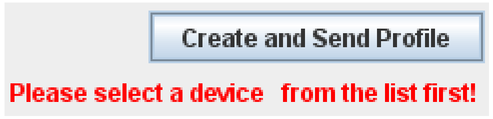
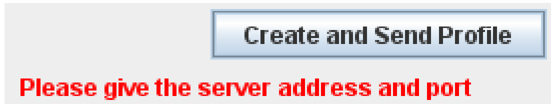
- Display properties detection
- Display properties description(XML format)
- Transfer the XML profile to the server application
- Receive the image from the server application
- Display the resulting image.

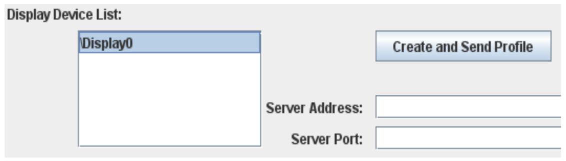
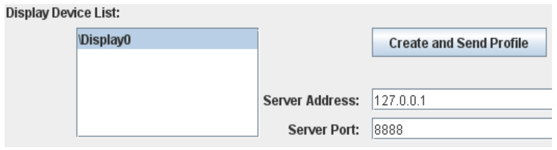
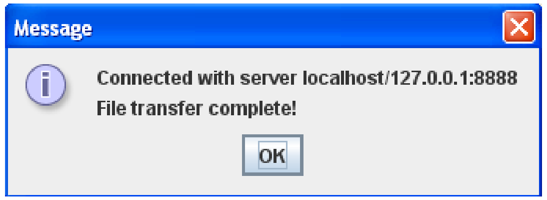
For each function above, we needed to test if it worked correctly or not. In addition to those main functions, the client application also has other functions. These functions are used by the major function and were not evaluated explicitly. The following table provides the detail of the tests and the result of those tests.

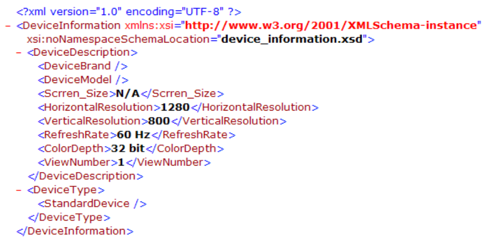
Table 5-3 The client application functions tests

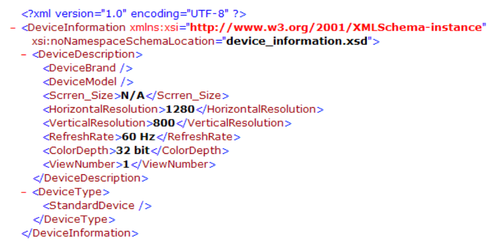
Test Number	Test Item	Test Method	Expect Result	Test Result
1	Automatically detect display properties	Run the client application, and select the display device from the display list.  Display Device List: \Display0	The client application should detect the basic display properties, the resolution, the colour depth, and the refresh rate. If the device has EDID data, the client application should detect the EDID information.	The client application detected the resolution, the colour depth, and the refresh rate. The current client application could not detect the EDID information because of the library license. The license is valid for 30 days trial, and it was expired when the evaluation was conducted.  Device ID: \Display0; Resolution: 1280 X 800; Refresh Rate: 60 Hz; Color Depth: 32 bit
2	Display the required display properties for different types of device	Select different device types from the Device Type list on the client application.  Device Type: StandardDevice; StandardDevice, StereoscopicDevice, VolumetricDevice	There is a star that will appear beside the property field to show which property is required for the selected device type.	Displayed the required display properties for different types of device.  Resolution: X *; Refresh Rate: *; Color Depth: *

				 Eye Separation: * Focal Length: * Aperture: * No. of Slice: *
3	Check the required display properties for different types of device	a) After the client application detects the properties, click the “Create and Send Profile” button on the client application. b) Delete some required properties value and then click the “Create and Send Profile” button on the client application.  Device ID: \Display0 Resolution: 1280 X * Refresh Rate: 60 Hz * Color Depth: 32 bit *	If all required properties are complete, the client application will give a message showing that the profile has been created successfully. If missing a required property, the client application should give a message to the user that some required properties are missing.	In test a, the client application created and sent the profile to the server, and showed a successful message.  Create and Send Profile Device Profile created successful In test b, the client gave an error message to the users to finish the required property fields and to create and send profile.  Create and Send Profile Please finish the required property field with * !
4	Valid property value check	a) After the client application detects the properties, click the “Create and Send Profile” button on the client application. b) Add some invalid property values, for example, add some text in the resolution field, and then click	The client application should create the profile successfully if there is no invalid value. If there is an invalid property value, the client	In test a, the client application successfully created and sent the profile. In test b, an error message about invalid property values was displayed. The client application to create and send profile stopped.

		the “Create and Send Profile” button on the client application.  Device ID: \Display0 Resolution: 1280 X test * Refresh Rate: test Hz * Color Depth: 32 bit *	application should give a message to the user.	 Invalid property value, please check!
5	Create and send profile	Click the “Create and Send Profile” button in the following situations: a) Click the button without selecting any item from the Display Device List  Display Device List: \Display0 Server Address: Server Port: b) Select the display device but do not give the Server Address and Port value, then click the button.	For the first situation, the client application should give a message notifying the user to select a device from the Display Device List. For the second situation, the client application should give a successful message to the user to create the profile. It should give another message to the user that the Server Address and Port value is missing,	In test a, the client application gave an error message notifying the users to select a device from the list.  Please select a device from the list first! In test b, the client application gave an error message notifying the users to input the server address and port.  Please give the server address and port In test c, the client application gave a successful message to create and send the profile.

		 Display Device List: Display0 Create and Send Profile Server Address: Server Port: c) Select the display device and give the Server Address and Port value, then click the button.  Display Device List: Display0 Create and Send Profile Server Address: 127.0.0.1 Server Port: 8888	and profile transfer has failed. For the third situation, the client application should give the successful message to the user to create and send the profile.	 Message Connected with server localhost/127.0.0.1:8888 File transfer complete! OK

6	Check the profile content	Run the client application, select a display item and click the "Create and Send Profile" button.	After clicking the button, the client application should create a XML file to store the display properties. The XML file should be stored at the same location as the client application, and the content should be correct.	We found the XML file after we clicked the button. We also checked the content of the XML, and the content was correct for the select display device.
				 XML file (1280_800.xml)create by client application:
				  `<?xml version="1.0" encoding="UTF-8" ?>`
				`- <DeviceInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="device_information.xsd">`
				`- <DeviceDescription>`
				`<DeviceBrand />`
				`<DeviceModel />`
				`<Scrren_Size>N/A</Scrren_Size>`
				`<HorizontalResolution>1280</HorizontalResolution>`
				`<VerticalResolution>800</VerticalResolution>`
				`<RefreshRate>60 Hz</RefreshRate>`
				`<ColorDepth>32 bit</ColorDepth>`
				`<ViewNumber>1</ViewNumber>`
				`</DeviceDescription>`
				`- <DeviceType>`
				`<StandardDevice />`
				`</DeviceType>`
				`</DeviceInformation>`
				 XML file (5268497.xml) on server side:

				`<?xml version="1.0" encoding="UTF-8" ?>`
				`- <DeviceInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
				`  xsi:noNamespaceSchemaLocation="device_information.xsd">`
				`  - <DeviceDescription>`
				`      <DeviceBrand />`
				`      <DeviceModel />`
				`      <Scrren_Size>N/A</Scrren_Size>`
				`      <HorizontalResolution>1280</HorizontalResolution>`
				`      <VerticalResolution>800</VerticalResolution>`
				`      <RefreshRate>60 Hz</RefreshRate>`
				`      <ColorDepth>32 bit</ColorDepth>`
				`      <ViewNumber>1</ViewNumber>`
				`    </DeviceDescription>`
				`  - <DeviceType>`
				`      <StandardDevice />`
				`    </DeviceType>`
				`</DeviceInformation>`

7	Display the resulting image	Run the client and server application, create and send a profile to the server application.	The server application should generate the resulting image base on the profile and send it back to the client application. The client application should display the resulting image.	After we sent the profile to the server application, the client application received and displayed an image. The image was created based on the device properties. Resulting image:

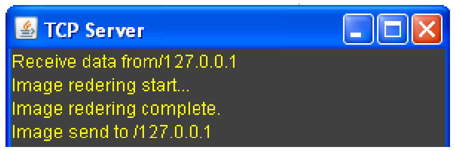
5.2.3 Server application evaluation

The main task of the server application is discussed in chapter 3, the design chapter. The server application should be able to configure the rendering system based on the device properties. The server application includes the following functions:

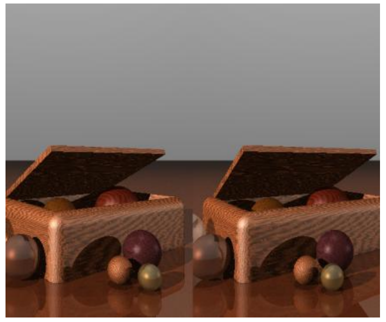
- Receive the device profile from the client
- Parse the XML profile to obtain the display properties
- Configure the rendering software and generate the resulting image
- Send the resulting image back to the client application.

The following table provides the detail of the tests for each function of the server application.

Table 5-4The server application functions tests

Test Number	Test Item	Test Method	Expect Result	Test Result
8	Receive the XML profile from the client application	Run the client and server application. Select the display device, give the server address and port value and then click the "Create and Send Profile" button.	The client application should give a message that the transfer of the profile has been successful. The server application should display "Receive data from "*client IP address*"" on its form. Lastly, the server application should display "Image send to "*client IP address*"", and the client application should receive the resulting image data from the server and display it.	The server application received the XML file from the client application and stored the file to the disk at same location as the server application.  TCP Server Receive data from/127.0.0.1 Image redering start... Image redering complete. Image send to /127.0.0.1
9	Check the device profile content	After the device profile has been transferred from the client application, open the file and check the content.	The XML file on the server side should be same as the client side. The content of the profile should be same, otherwise the data transfer between the client application and server application is incorrect.	Opened the XML file and compared the content with the original file on the client side. Where the content was the same, the data transfer function was working correctly.
10	Check the XML parse function	Run the server application as debug mode, and use the Java compiler to monitor the data parse function.	The data parse function should be able to obtain the property value from the XML file, and pass the value to corresponding variables.	The data parse function found the property value from the XML profile, and then passed the value to the variables so that the application could use it for further processing.

11	Check the resulting image	Use the rendering software to generate an image based on a display device's properties used as a reference image. Then, use the client and server applications to detect the properties and create an image. Compare these two images.	Two images should be same.	The resulting image and the reference image were same. The result proved the render configure function and image generation function were working correctly.
12	Check the multiple user	Use multiple clients (4) applications to connect with one server application. The client applications use different display devices.	The server application should be able to connect with the client applications and generate the correct image for each client.	The server application generated the correct image for each client application. The images were generated separately and based on the connection order.
13	Check the image for the stereoscopic display	Run the client application, and select the display type as stereoscopic, then send the profile to obtain the image.	The server application should create a combined image for the stereoscopic display.	The server application generated two images, one for each eye, and combined these two images. The client application received a combined 2-view stereoscopic image.

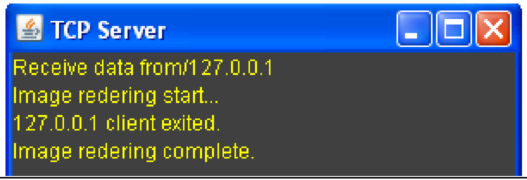
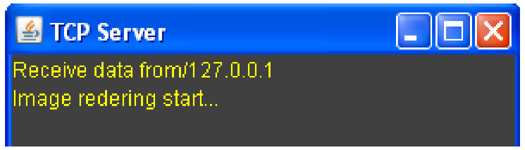
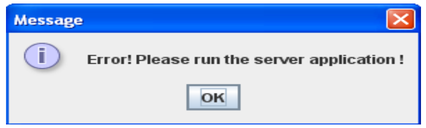
5.2.4 Exceptional and abnormal conditions

When we conducted the evaluation of ADAM described in 5.2.1, 5.2.2 and 5.2.3, we assumed ADAM was in a working environment. All components, such as render software and network, were functioning correctly. However, in real world use of ADAM, the working environment may encounter abnormal situations. The user may not follow expected or desired usage patterns. This may cause an unexpected result for the client and the server applications. In this section, we describe the test conducted on ADAM under some special situation, and discuss the test result.

In ADAM, after the client application transfer of the device profile to the server application, the render software will be called and start to generate an image. In this section, we discuss how we tested ADAM under the following situations:

- After the profile is transferred to the server application, and before the resulting image is sent back, shut down the client application.
- The server application needs to use POV-Ray to generate the resulting image. If the POV-Ray is missing, what situation will occur?
- The server application is shut down, and the client application still attempts to transfers the profile to the server.

Table 5-5Exceptional and abnormal conditions tests

Test Number	Test Item	Test Method	Expect Result	Test Result
14	Check the system status when the client application is shut down	Shut down the client application when the server application is generating the image.	The client application gives a message to remind user that image is rendering on server side.	The server application continued to generate the image until finished. The server displayed "*IP address* client exited".  Receive data from/127.0.0.1 Image redering start... 127.0.0.1 client exited. Image redering complete.
15	Check the system status if the render software is missed	Run the server application in a non-POV-Ray working environment.	The client application gives error message to notify the users that the render has a problem.	The server application continued to run but no image was generated.  Receive data from/127.0.0.1 Image redering start...
16	Check the system status when the server application is shut down	Run the client but not the server application, and then transfer the profile to the server.	The client application gives the message to notify the users that server is closed.	The client application sent a message to notify the user that the server application was not running and that file transfer had failed.  Error! Please run the server application !
17	Change the screen	Run the client application, and	The client application can	If the user had already selected the display device

	properties while the client application is running	then change the resolution of the screen.	detect the latest resolution of the screen.	from the list before the resolution changed, the user had to rerun the client application to detect it again. If the user did not select the device, the application detected the latest resolution after the change.

5.3 Results and Discussion

Section 5.2 discussed the evaluation of the functions for the client and server applications. The design chapter lists some objectives for the system, and in this section, we discuss the test results in relation to these four objectives.

1. Support multiple display devices

To support different display devices we need to find the display properties from a display device so that the render software can generate the resulting correct image based on the properties. In ADAM, the client application provides functions that can detect the display properties from the display device. In the test, we used the client application to detect the display properties from different display devices. Test 1 showed that the detect method automatically detects the basic display properties from the connect display device accurately.

The client application also provides other functions, such as the required property check and valid property value check, to help end users to ensure the property data is completed and data format is correct. In tests 2, 3, and 4 the functions were evaluated and the test outcomes matched the expected result.

Although the property detect method can obtain the basic property data from the device, there are other properties that the users of special devices may need to enter manually, for example, the eye separation value for the stereoscopic device. Currently, at the client application user interface, there are the default display device types and property fields. However, the users may want to use other types of display devices or add some new display property. This would require modification of the client application code. This is not a flexible way to add the display device and property type for end users.

We developed a method that can use the display property data to generate XML file as a device profile. This profile can be used to configure the render software on the server side in order to generate the correct image. As tests 5 and 6 showed, this method is currently working well but it faces the same challenge as the client user interface. If the user needs to record some new type of display device and property, the method code needs to be changed.

2. Support multiple users

We developed a client server structure system. The server application is able to connect with multiple users. In the test, we opened four client applications and connected to the server application, and then we tested the capability for each function of the server application. Test 8 showed that the device profile can be transferred from the client applications to the server application. We checked the profile content on both client and server side. The profile content was the same, so that the data transfer function on client application and data receive function on server side was working. In addition, the resulting image was transferred between the server application and each client application correctly.

3. Automatic configuration

It is important that the server application can use the display properties to configure the render software automatically, so that the users can obtain the resulting correct image without manually changing the render parameters each time the users change to a different display device. To use the property data in the profile, we developed a method that can parse the XML profile file in order to obtain the property data. In test 10, we checked the profile parse function and the outcome showed this function can parse the device profile to get the properties. In the trial, we used the render software to generate an image based on the specific display device’s properties and used it as the reference image. After that, we used the client application to detect the properties from this display device and generate the resulting image. We compared the two images to test the configuration function. The resulting image was the same as the reference image, as shown in test11, so that the configuration function was working.

4. Platform independence

We have developed a Java solution to achieve the platform independence requirement. In the trial, we tested the client and the server application on different platforms. For most platforms, the client application and server application were running well, but the client application failed to run on the Windows Mobile platform. We used a Windows Mobile emulator to test if the client application can be used on a mobile device. In addition to using the Java application on the Windows mobile platform, we needed to use a Java virtual machine application to load and run the Java application. Although we installed the Java virtual machine application, the client application still could not run.

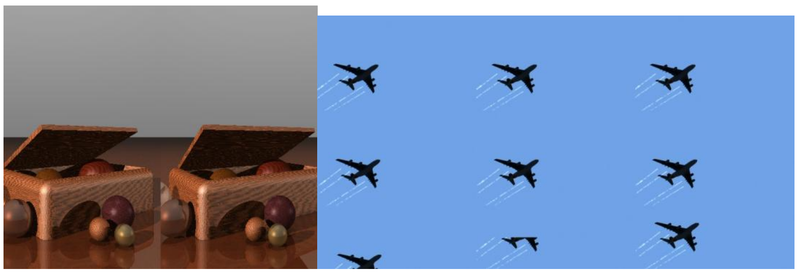
Another issue is the render software. Currently, we use the POV-Ray Windows version as the render toolkit. Thus, the configuration method is based on a Windows platform and cannot be used on other platforms directly. We can modify this function to solve this issue and the detail will be discussed in chapter 6, the future work chapter.

5.3.1 Limitations

According to this evaluation and some other feedback, some limitations of ADAM have been found.

1. Support the multiview stereoscopic device

Currently, the ADAM can generate the 2 view stereoscopic image. In test 13, we successfully generated a 2 view stereoscopic image by using ADAM. However, if the device is a multiple view stereoscopic device, for example an 8 view, the ADAM cannot generate the corresponding stereo images. The following figures show the stereo images of 2 view and 8 view stereoscopic display device.



(a) (b)

Figure 5-1 (a) 2 view stereo image (b) 8 view stereo image

As we can see in Figure 5-1, the 8 view stereo image is more complex. The calculation of this image is different compared with the 2 view stereo image.

2. Support other special display devices

The ADAM can support the standard and 2 view stereoscopic display device currently. The system needs to be further developed so that it can support other special display devices, such as the volumetric display device.

3. The extension of user interface and device profile

The user interface needs to be further developed. For example, how can the client user interface allow the users to add new display property types or device types to the interface?

The XML device profile is used to describe and store the property information for a device. However, if there are new types of properties, we need to add the new property to the profile. Currently, if we need change the property type in XML profile, we have to recode the client application function.

4. Support more platforms

In this evaluation, the client application can work on many platforms except the Windows mobile platform. The current version of the server software is configured to make use of the Windows version of the POV-Ray software. POV-Ray is available for platforms other than Windows, including Linux and MAC OS X, however to make use of these versions the server application would need to be recompiled with a platform specific command line. At present therefore the server application is limited to running in a Windows environment.

5.4 Summary

To evaluate the functionality of ADAM, we conducted several different tests to test each function. The test outcomes showed that the main functions, such as the property detect function, the property describe function, the data transfer function, and the render configuration function, can achieve the design requirements. Thus, ADAM can detect the display properties from the connect display device and configure the render software correctly based on the properties. This means that ADAM enables the users to use different display devices to obtain the correct image output without requiring any render configuration in advance.

During these tests, the evaluation also shows some limitations of the current solution. The solution needs to be further developed in order to support more special display devices. There is a problem, for example, when the client application runs on the Windows Mobile platform. The functions such as property description and the user interface also need to be considered as to how to allow the users to add new type of properties in a flexible way.

This section presented the evaluation of ADAM and discussed the test results in relation to the design requests. During the tests, some limitations were found, and the future work on these limitations will be discussed in next chapter.

Chapter 6 Conclusion and Future Work

This chapter draws this project to a conclusion and discusses its success and achievements. Lastly, the future work is presented, highlighting avenues for further development and refinement.

6.1 Conclusion

Currently, in a visualisation system, a user may use a variety of display devices to view the visualisation result, such as the standard 2D monitor and stereoscopic monitor. More especially, in a remote visualisation environment, there are multiple users who may access the resulting visualisation and the users may use different devices. The visualisation result may be displayed correctly on some displays but distorted on others due to the different display properties of the device. Therefore, the end user needs to carefully configure the rendering parameters for different devices based on the device’s properties each time in order to obtain the correct visualisation output. As a result, in multiple types of display environments, end users can find it very difficult to correctly configure the renderer for different devices without assistance from technical experts. This situation highlights the complexity of using the visualisation system, and also may lead to the user receiving the incorrect resulting image for a particular display device. In order to solve this problem, the solution needs to met these requirements:

- Support different types of display devices to show a visualisation without manual configuration of the renderer, so that end users can use different displays easily.
- Support a multiple user environment, so that the solution can be used in a remote visualisation system.

To develop an appropriate solution for end users, we discussed the design of the system in chapter 3. We indicated that the application should have the ability to detect and collect the display properties from the display device. In addition, the solution should also have the ability to use the detected display properties to automatically configure the renderer in order to generate the correct image.

To support a multiple user environment, we pointed out that the system should be a client-server based approach. The solution should use a platform independent programming language so that it can support users on different platforms. In order to meet the requirements, we developed a client application and a server application system. This system is implemented in Java. We evaluated how well the approach met the requirements via several tests and discussed the results in chapter 5, the evaluation chapter. The achievements of the approach are summarised below for each of the requirements.

- Support multiple display devices

To support different display devices, we developed an approach to detect the display properties of a device, and used the properties to automatically configure the renderer.

We have developed a client application which can automatically detect the display properties, such as the resolution, refresh rate, and colour depth, from the display device. In addition, the users can manually enter extra properties, such as the eye separation and viewing distance of a stereoscopic display. The display device properties are stored as a profile using an XML format.

We have developed a server application which can automatically configure the renderer by using the display properties. After the server application receives the device profile from the client application, the profile is parsed and the server application obtains the property data, and then the automatic configuration method uses the properties to configure the renderer. As a result, the renderer can render the appropriate image for that display device.

By using the applications, end users are able to obtain the correct visualisation output for different displays without manually configuring the render.

- Support multiple user environment

A multiple user environment may lead to a situation with various computer platforms. To support multiple user environments, we used Java language to implement the client and the server applications, so that the solution can be used on different platforms.

In addition, in a multiple-user environment, such as a remote visualisation system, a server should allow multiple clients to access and obtain the resulting image. Thus, we use the client-server base structure to implement the approach. The server application of ADAM allows multiple client applications to connect and transfer the device profile and resulting image data.

This study looked at the issues involved in multiple display devices and multiple user environments in visualisation. The primary goal of this project is to create a solution for users to easily use the different types of display device in visualisation. In this project, we developed a system named ADAM to achieve this goal. The result is that ADAM is able to configure the render to help users to obtain the correct visualisation output for some display devices. Although ADAM is a useful tool, it currently has some limitations. Future improvements are discussed in section 6.2.

6.2 Future Developments

This study provides a useful tool for the users to use different display devices in visualisation. However, there are some limitations in the current system. In order to improve ADAM and overcome the limitations discussed in the evaluation chapter, this section discusses future enhancements to support the multiview stereoscopic device, support special display device, refinement of the client user interface, the extension of the device profile language and support more platforms and render software.

1. Support the multiview stereoscopic device and other special devices

Currently, the system can support all standard display devices. For the stereoscopic display device, it can support 2 view stereoscopic display devices but cannot support multiple view stereoscopic devices such as an8 view device. In order to support this type of display device, we need to further develop the camera parameters calculation and image compositing methods.

Although the volumetric device type is included in the current device type list in the user interface, the system does not currently support this type of device. Volumetric displays use multiple image slices of a visualisation to display a 3D image. Each slice is different. To support volumetric devices we need to undertake more complex calculations to determine the images to render.

2. The client-user interface

The client application provides many display device types and display property types on the user interface. Currently, we use a combo box to list all display device types, the users can only select the device type from the list. In order to improve this design that allows the user to add new device types from the user interface, we plan to add a new method that allows the users to add customised device types by themselves.

Additionally, we put all display properties fields on the current client application user interface. The users can enter the property data in these fields, but the property type cannot be changed. We need to upgrade the interface that should allow the users to add new display property types, and the users to set the data type of the new property.

3. The extension of display device and property

Another method that we need to improve is the property description method. The current method can only record existing display properties, if the users need to add a new display property, the describe function has to be recoded. As we improve, the user interface will allow the users to add new display property types. We plan to modify the existing property description function to make it more flexible and capable of adding or removing the display property.

4. Support more platforms and render software

Currently, the client application does not run on the Windows Mobile platform. We need to determine the reason for this. In the future, we plan to support more platforms In addition, the POV-Ray windows version is used as the render software in the current system. In the future, we plan to use multiple versions of POV-Ray, such as Linux or Mac version, to help the users use this system in different platforms.

Additionally, we can make the system support more render software, such as the OpenGL, and VTK, so that the users have more options. A possible design for supporting other render software is to use a case structure on the server side. For each case, we can create a method to call a render software such as POV-Ray, OpenGL, and VTK. In addition, we need a list to display all available render software on the user interface so that the users are able to select the needed renderer.

6.3 Summary

This section presents the conclusion and future work of this project. In the conclusion, we represent the problem and the aim of the project, discuss the achievements and contributions of the project. Future work of the system is also discussed in this section. This is based on the current limitations and includes three main parts to support more display devices, extend display properties, and support more platforms.

This study investigated supporting different display devices in a multiple user environment. The approach uses a client-server based structure. It provides a client application for collecting display properties and creating a device profile, and a server application for automatic configuration of the renderer and generation of the image. The approach was evaluated with several tests and the results show that this solution was useful to help users to use different display devices in visualisation.

References

Annen, T., Matusik, W., Pfister, H., Seidel, H.-P., & Zwicker, M. (2006). *Distributed rendering for Multiview Parallax Displays*. Paper presented at the SPIE Conference Stereoscopic Displays and Virtual Reality Systems XIII. Retrieved from http://www.merl.com/papers/docs/TR2006-031.pdf

Bethel, W., Tierney, B., Lee, J., Gunter, D., & Lau, S. (2000, November).*Using high-speed WANs and network data caches to enable remote and distributed visualization.* Paper presented at the IEEE Supercomputing Conference, Dallas, TX. Retrieved fromhttp://doi.ieeecomputersociety.org/10.1109/SC.2000.10002

Bourke, P. (1999). *Calculating stereo pairs*. Retrievedfrom http://local.wasp.uwa.edu.au/~pbourke/projection/stereorender/

Deb, S., & Narayanan, P. J. (2004).*Aremote visualization system for large environments* (No. 2004-41). Hyderabad, India: International Institute of Information Technology. Retrieved from http://www.iiit.ac.in/techreports/2004_41.pdf

Ding, J., Huang, J., Beck, M., Liu, S., Moore, T., & Soltesz, S. (2003). *Remote visualization by browsing image based databases with logistical networking.* Paper presented at the Supercomputing, 2003 ACM/IEEE Conference. Retrievedfrom http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=--1592937

Engel, K., Sommer, O., Ernst, C., &Ertl, T. (1999). Remote 3D visualization using image-streaming techniques.*Proceedings of the International Symposium on Intelligent Multimedia and Distance Education*. Retrieved from http://citeseer.ist.psu.edu/394248.html

Grimstead, I. J., Avis, N. J., Walker, D. W., & Philp, R. N. (2005, September). *Resource-aware visualization usingweb services*. Paper presented at the UK e-Science All Hands Meeting, Nottingham, England. Retrieved from http://www.allhands.org.uk/2005/proceedings/papers/306.pdf

Haber, R. B., & McNabb, D. A. (1990). *Visualization idioms: A conceptual model for scientific visualization systems*. Visualization in Scientific Computing, 74-93.

Halle, M. (1997). Autostereoscopic displays and computer graphics.*Computer Graphics, 31*(2), 58-62.

Mackay, D. (2006). *Generating synthetic stereo pairs and a depth map with PoVRay.* (DRDC-Suffield-TM-2006-197).Ottawa, Canada:Defence Research and Development. Retrieved fromhttp://www.drdc-rddc.gc.ca/index1-eng.asp

Meißner, M., Pfister, H., Westermann, R., & Wittenbrink, C. M. (2000).*Volume visualization and volume rendering techniques*. Retrieved from www.labri.fr/perso/preuter/imageSynthesis/02-03/papers/volvistut.pdf

Sato, Y., Nakajima, S., Atsumi, H., Koller, T., Gerig, G., Yoshida, S., et al. (1997). *3D multi-scale line filter for segmentation and visualization of curvilinear structures in medical images* In Lecture Notes in Computer Science (Vol. 1205/1997): Springer Berlin / Heidelberg.

Schroeder, W., Martin, M. K., & Lorensen, E. W. (1998). *The visualization toolkit: An object-oriented approach to 3D graphics*(2nd ed.)Englewood Cliffs, NJ: Prentice-Hall.

Tierney, B., Lee, J., Crowley, B., Holding, M., Hylton, J., & Drake, F. (1999). A network-aware distributed storage cache for data intensive environments. *Proceedings of IEEE High Performance Distributed Computing Conference.* Retrieved from http://www-didc.lbl.gov/DPSS/

VESA. (2007). *E-EDID verification guide*. Retrieved from http://www.vesa.org/Standards/summary/2007_3b.htm

W3C. (2008). *Extensible Markup Language (XML).*Retrieved from http://www.w3.org/XML/

Ware, C. (2004). *Information Visualization: Perception for Design*. San Francisco: Morgan Kaufmann.

Watt, A., & Policarpo, F. (1998). *The Computer Image.* Harlow: Addison Wesley Longman Limited.

Yuen, D. A., Garbow, Z. A., & Erlebacher, G. (2004). *Remote data analysis, visualization and problem solving environment (PSE) based on wavelets in the geosciences*. Visual Geosciences, 9(January, 2004). Retrieved from http://www.springerlink.com/content/6klcc6px6tk80xnq/fulltext.pdf

Appendix A

Device Information XML Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="DeviceInformation">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="DeviceDescription">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="DeviceBrand" type="xsd:string">
              </xsd:element>
              <xsd:element name="DeviceModel" type="xsd:string">
              </xsd:element>
              <xsd:element name="ScreenSize" type="xsd:string">
              </xsd:element>
              <xsd:element name="DeviceManufacturer" type="xsd:string">
              </xsd:element>
              <xsd:element name="DeviceName" type="xsd:string">
              </xsd:element>
              <xsd:element name="HorizontalFrequencyUpperLimit" type="xsd:int">
              </xsd:element>
              <xsd:element name="HorizontalFrequencyLowerLimit" type="xsd:int">
              </xsd:element>
              <xsd:element name="VerticalFrequencyUpperLimit" type="xsd:int">
              </xsd:element>
              <xsd:element name="VerticalFrequencyLowerLimit" type="xsd:int">
              </xsd:element>
              <xsd:element name="HorizontalSize" type="xsd:int">
              </xsd:element>
              <xsd:element name="VerticalSize" type="xsd:int">
              </xsd:element>
              <xsd:element name="AspectRatio" type="xsd:string">
              </xsd:element>
              <xsd:element name="HorizontalResolution" type="xsd:int">
              </xsd:element>
              <xsd:element name="VerticalResolution" type="xsd:int">
              </xsd:element>
```

```
                    <xsd:element name="RefreshRate" type="xsd:int">
                    </xsd:element>
                    <xsd:element name="ColorDepth" type="xsd:int">
                    </xsd:element>
                    <xsd:element name="ViewNumber" type="xsd:int">
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="DeviceType">
            <xsd:complexType>
                <xsd:choice>
                    <xsd:element name="Standard">
                    </xsd:element>
                    <xsd:element name="Sterescopic">
                        <xsd:complexType>
                            <xsd:attribute name="EyeSeparation" type="xsd:decimal"
use="required">
                            </xsd:attribute>
                            <xsd:attribute name="FocalLength" type="xsd:decimal"
use="required">
                            </xsd:attribute>
                            <xsd:attribute name="Aperture" type="xsd:decimal" use="required">
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="Volumetric">
                        <xsd:complexType>
                            <xsd:attribute name="SliceNumber" type="xsd:int" use="required">
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:choice>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Appendix B

Device Description Category

Element Name	Data Type	Content	Required Property
Device Brand	String	The brand of display device	Optional
Device Model	String	The model of display device	Optional
Screen Size	String	The diagonal length of display device	Optional
Device Name	String	The display device name in operation system	Optional
Horizontal Frequency	Integer	The range of horizontal frequency of device	Optional
Vertical Frequency	Integer	The range of vertical frequency of device	Optional
Horizontal Size	Integer	The horizontal physical length of device screen	Optional
Vertical Size	Integer	The vertical physical length of device screen	Optional
Aspect Ratio	String	The aspect ratio of device screen	Optional
Horizontal Resolution	Integer	The current horizontal resolution of the device	Yes, for all devices
Vertical Resolution	Integer	The current vertical resolution of the device	Yes, for all devices
Refresh rate	Integer	The current refresh rate of device	Yes, for all devices
Colour Depth	Integer	The current colour depth of device	Yes, for all devices
View Number	Integer	The number of view angle for a display device. This property is special for stereoscopic device, the view number of different autostereoscopic device may not same. For example, if the view number is 2, the device allows people to view the screen in two	Yes, only for stereoscopic devices

		different angle with 3D effect.	

Device Type Category

Element with Attribute	Data Type	Content	Required Property
Standard		No special attributes	Yes, for all devices
Stereoscopic (Element)			
Eye Separation	Decimal	Eye separation value for a stereoscopic device	Yes, only for stereoscopic devices
Focal Length	Decimal	Focal length for a stereoscopic device	Yes, only for stereoscopic devices
Aperture	Decimal	Aperture value for a stereoscopic device	Yes, only for stereoscopic devices
Volumetric (Element)			
Slice Number	Integer	The number of slice for a volumetric device	Yes, only for volumetric devices