FACULTY OF ENGINEERING AND SCIENCE
AALBORG UNIVERSITY

INSTITUTE OF ELECTRONIC SYSTEMS

TITLE:

Volume rendering in 3D Visual Data Mining.

TIME PERIOD:

8th semester,
February 3rd to June 2nd, 2004

PROJECT GROUP:
822 - 2004 CVG8

GROUP MEMBERS:

Mikkel Sandberg Andersen

Sanne Christensen

Tommy Jensen

Rikke Ottesen

SUPERVISOR:

Henrik Rojas Nagel

NO. OF COPIES: 7

NO. OF PAGES IN MAIN REPORT: 93

NO. OF PAGES IN APPENDICES: 41

NO. OF PAGES IN TOTAL: 139

ABSTRACT:

The purpose of this project is to explore the possibilities of displaying advanced visual objects in a 3D visual data mining system in order to explore multi-dimensional data sets in arbitrary view directions and from arbitrary view points.
For visualisation of three different test objects, also called glyphs, volume rendering is employed. The technique used is volume slicing, which samples a series of textures from a volume, perpendicular to the viewing direction, mapping these to a corresponding series of plane surfaces. A number of performance related subjects are also explored. To verify results, an interview with an expert 3D visual data mining user is conducted.
A prototype volume rendering enabled 3D visual data mining system has been developed. With this system, it is possible to visualise multi-dimensional data sets by using advanced visual objects. For each glyph, different mappings of data dimensions to features are possible.
The report concludes that volume rendering used in visual data mining is well suited for visualising a limited number of objects with many features, while surface based rendering is well suited for visualising a large number of objects with few features. Furthermore, it is concluded that further research in the field of perceptual possibilities could be conducted.

Preface

This report is the documentation of the work of group 822, 8th semester specialisation in Computer Vision and Graphics (CVG) 2004, Laboratory of Computer Vision and Media Technology, Aalborg University.

The target audience for this report is 8th semester students on the CVG specialisation.

The report is structured by six parts: Problem domain, system requirements, analysis, design, test, and conclusion.

The appendix is placed last in the report as is the over all class structure of the system.

Source references are on the Harvard-form, an example of this is: [Sherman and Craig, 2003]. Chapters and sections starting with a source reference is based on this source.

The group would like to thank Ryan Rohde Hansen, Rasmus Stenholt. The group would also like to thank Erik Granum for participating in a user interview.

Attached is a CD-rom on which the following can be found:

- Report
- Source code
- Compiled programme
- Documentation of source code
- Class diagram
- Interview

Aalborg University, Denmark. June 2nd 2004.

Mikkel Sandberg Andersen

Sanne Christensen

Tommy Jensen

Rikke Ottesen

Contents

CONTENTS

CHAPTER 1

Introduction

This report is concerned with developing a tool for aiding the analysis of large multi-dimensional data sets. The aim of the report is the area of analysis called Visual Data Mining (VDM), which is the concept of visualising data sets so that features are easily interpreted.

Data Mining

With the development in computer processing and data storage in the last few decades, storing large and complex data sets in databases has become very common. These data sets may describe anything, ranging from sales statistics to registration of results from a scientific experiment. Common for all data sets are that they typically contain a vast number of records, with each record having a large amount of information.

The concept of data mining is to make use of these large, complex, information-rich data sets. This involves various ways of analysing these data sets to discover inter-dataset relations or patterns or anomalies.

The following two sections will briefly describe some of the developments in data mining.

Classical Data Mining

Classical data mining techniques are in some sense an extension of statistics with a few extra additions. Some techniques used are, artificial neural networks, nearest neighbour method and decision trees.

Artificial neural networks Non-linear predictive models, which learn through training and resemble biological neural networks in structure. [Haykin, 1994]

Nearest neighbour method A technique which classifies each record in a data set based on a combination of the classes of the k record(s) most similar to it, in a historical data set. Sometimes called the k-nearest neighbour technique. [Duda et al., 2000]

Decision trees Tree-shaped structures which represent sets of decisions. These decisions generate rules for the classification of a data set. [Jackson, 1990]

Visual Data Mining

Source: [Cox et al., 1997]
The idea of Visual Data Mining (VDM) is to improve the ability to make use of large and complex data sets, building on the fact that people are at the centre of data mining enterprise. Human pattern recognition skills are remarkable, and by far exceed the capability of any existing technology to detect

interesting patterns and relevant anomalies. By properly taking advantage of the human ability to deal with visual presentations, this technique aid the understanding of large amounts of data.

Only within the last decade, with the development of Virtual Reality (VR), has it been possible to fully exploit the possibilities of VDM. This involves enabling the users to immerse in a real-time visual presentation of data sets.

The 3DVDM system (3 Dimensional Visual Data Mining system), developed at Aalborg University by Henrik Nagel, is a system which enables perceptual representation of statistical data. The reference is a visual world where statistical observations are represented in a 3D scatter plot. Each data point in the plot is shown as a visual object, such that statistical variables may also control various visual object properties such as object shape, orientation, colour, surface texture, etc. The purpose is to present data from within, and to allow the human observer as a visual explorer to navigate the visual world, in order to inspect data in arbitrary view directions and from arbitrary viewpoints. [Nagel et al., 2003]

Current technology is also applicable in creating advanced real-time 3D soundscapes which may prove useful. The field of hearing of the human ear is larger than the field of view of the eyes, and thus is able to inform on events happening in areas that cannot be seen. The 3DVDM system provides tools for using 3D sound to perform general and detailed investigation of data visualised in a 3D scatter plot. With this, it is possible to place sampled sounds or synthesised sounds at any point in the virtual world. Thus it is possible to attach sounds representing statistical values to selected objects, or add sound representation to groups of objects, such as density clusters or measurements of density in solid angles around the users current position.

Volume rendering

The current implementation of 3DVDM displays very simple, surface based, visual objects such as tetrahedra shapes, as representations of records in data sets. The goal of this project is to further develop the rendering technique, as to enable the system to display advanced visual objects. Advanced visual objects have the ability of displaying multiple features both exterior and interior in a volume. These features can be visualised in a more complex manner, than those of simple visual objects in terms of shape and texture etc.. To visualise advanced objects, a volume rendering technique is well suited. The technique allows rendering of complex shapes from image volumes (a 3D bitmap image).

Defining the project

The aim of this project is to display advanced objects in a 3D visual data mining system, and allow a human observer as a visual explorer to navigate the system, in order to explore multi-dimensional data sets in arbitrary view directions and from arbitrary view points. In order to develop advanced objects, the conceptual possibilities and the perceptual possibilities of these objects have to be studied. Since the group does not possess expert knowledge in the area of perception psychology, this part of the project will be solely based on available literature.

To explore the possibilities of advanced graphical objects, a volume rendering technique is employed to render these. The implementation of this will be based on the graphic Application Programming Interface (API) OpenGL, as this has been introduced in a semester course.

To ensure a reasonable performance from the system, various compression methods will be studied in regards to volumetric image data. Furthermore, balancing speed and level of detail, when navigating

the 3D visual data mining system will be explored.

To verify the results, user tests involving one or more expert VDM users, will have to be conducted.

Part I

Problem Domain

CHAPTER 2

Visual Data Mining

This chapter describes the general concept of visual data mining. Furthermore, visualisation of data sets in 3D space is described in terms of object properties.

2.1 General concept of visual data mining

Source: [Card et al., 1999]
Data containing various information may be hard to interpret when looking at it in a database. Therefore using another way to visualise data is desirable, so that multiple features in the data can be easily interpreted. This can be done by using Visual Data Mining (VDM). The purpose of VDM is to show, in an easily conceivable way, multiple features, derived from a data set, i.e. to map data to a visible structure.

The typical way to achieve visualisation of data is by following the steps outlined by [Card et al., 1999] in figure 2.1.

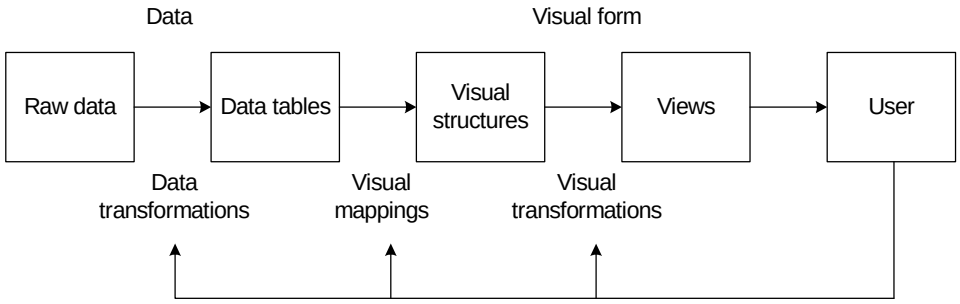

Figure 2.1: Visualisation of data achieved by human interaction. Source: [Card et al., 1999].

A set of data is recorded from any source the user wants to analyse. The data is stored as raw data with no apparent order. The raw data can have many forms such as spreadsheets and text.

In order for this data to be visualised, it has to be transformed into a relation or sets of relations which have been structured. This way the data is easier to map into a visual form. The new structure is stored in data tables.

The data transformation is achieved through human interaction, as all data recordings are individual and cannot be automatically structured into data tables. The data is structured according to the users interaction.

In a data table the data may be stored as three different types of variables:

Nominal An unordered set

Ordinal An ordered set (tuple)

Quantitative A numeric range.

Transformation of raw data into structured data tables typically involves loss or gain of data/information. Raw data may contain errors that must be corrected before the data can be visualised. A data transformation may also contain derived values or structures, as the transformation can incorporate statistical calculations, which again can add information not previously perceivable.

There are multiple ways to map data into a visual representation, such as histograms, curves and plotting of samples. It is important to choose the visual representation that fits best with the data it has to express.

After the data has been formatted into data tables, it has to be mapped to visual structures, which is a spatial substrate that encodes information through marks and graphics. This mapping must preserve the data, and the mapping should be expressive and effective.

The most fundamental aspect of a visual structure is its use of space. Spatial position is the best form of visual coding, that is why the first decision regarding visualisation usually is which variables that gets spatial positions at the expense of others. Spatial position is at the same time the most dominant form of visualisation.

The visual structures are described through axes and their properties, there are three types of axes:

Nominal axis A region is divided into subregions

Ordinal axis An axis with ordered subregions

Quantitative axis A metric axis

All the available information in the raw data is not always visualised. Information can be derived from the raw data, and there are several techniques to increase the amount of information, that can be encoded along with spatial position:

Composition The orthogonal placement of axes.

Alignment The repetition of an axis at a different position in space.

Folding The continuation of an axis in an orthogonal dimension.

Recursion A repeated subdivision of space.

Overloading Reuse of the same space for the same data table.

Marks are the visual things that occur in space, and they normally represent the data that the user requires visualised. There are four elementary types of marks:

Points Zero dimensional point

Lines One dimensional

Areas Two dimensional

Volumes Three dimensional

Points and lines allows graphs and trees to be visualised, and can signify a relation between objects.

There are six standard retinal variables that can be used to visualise data. These can be separated into spatial and object properties:

Spatial Position, size and orientation

Object Gray scale/colour, texture and shape

The separation between spatial and object properties, is due to the fact that these properties are believed to be processed differently by the human brain. These properties are a good basic set for visualisation purposes, but there are other possibilities as well. Other properties include:

Crispness Inverse of blend-distance.

Resolution Grainyness of the displayed object

Transparency How visible the object is

Arrangement If an object is several dots, it is their configuration

Hue/Saturation Colour coding.

Temporal Temporal change in mark position.

Sound Sound may inform on events happening in areas that cannot be seen.

When the visual structure is composed, the user has to decide how to display the visual structure. The view transformation is done through a visualisation engine, that can be in 2D or 3D, and have a distinct way of showing the encoded properties.

2.2 Visual Data Mining in 3D Space

Visual data mining in 3D space is one way of visualising multi-dimensional data sets by letting an object in the space representing a data record. Each object in the 3D space may be described using various feature variables, which represent various dimensions of the record.

2.2.1 Object variables and relations

Source: [Card et al., 1999]
In order to detect visual tendencies in data sets in the 3D space the analyst has to be able to navigate through this space. Also the data has to be represented as objects in the 3D space, where each object corresponds to each data record. These objects have various properties (variables) such as: Spatial position, shape, size, pose, surface properties, etc, which are visual variables.

Spatial position

The three positional variables, x, y, and z, form the location of the objects in the coordinate system of the 3D space. They are continuous variables in the visual space.

Shape

The shape of the objects may make it easier for the analyst to detect visual tendencies in the 3D space, however detailed geometrical visualisations have to be approximated by planar triangles and thus complex shapes are expensive for the visualisation system. An example of shape is seen on figure 2.2.

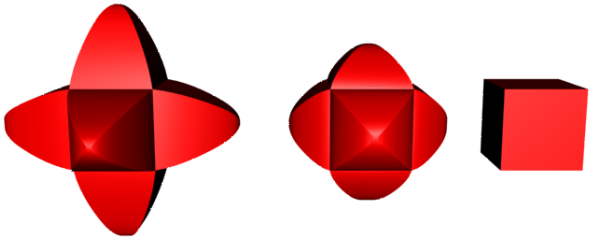

Figure 2.2: An example of a shape morphing between two basic shapes.

Scale
The scale of an object in the 3D space is described as the volume of the object. Scale is a variable that can be used to accentuate visual tendencies, however the scale of an object is often used to describe the distance from the viewer to the object. An example of scale is seen on figure 2.3.

Figure 2.3: An example of shape scaled between 0 and 1.

Pose
The pose of an object in the 3D space can be described in terms of panning, tilting and rotating. A human, when not informed otherwise, will assume that an object is upright. Pose is normally used on objects to allow orientation to be perceptually noticed. Thus pose can only be applied to objects with familiar shapes and the shapes are not allowed to be symmetrical since changes in pose of a symmetrical object cannot be observed. An example of pose is seen in figure 2.4.

Figure 2.4: An example of three different poses.

Surface properties

Surface properties may be described as roughness, general reflectance, and differentiated reflectance patterns. Roughness is a very expensive property in terms of visualisation unless approximated by a texture. Studies show that humans attend to the top right of small objects thus the projected features

should mainly be placed on the top right of the objects. The surface properties colour, texture, and transparency will be described in the following.

Colour
Colour may be described in various ways such as hue, saturation, and brightness; or red, green and blue. Studies show that the analyst detect visual tendencies faster when colour has been applied to the objects, however the use of more than six colours can seem confusing. Applying a black rim on a white background or vice versa also enhance detection of visual tendencies. Generally colour should not be used as a continuous variable due to limits in the human perceptual system in distinguishing accurately between hue, saturation and brightness. An example of colour is seen on figure 2.5.

Figure 2.5: An example of three different colours.

Texture
Texture may be described in terms of granularity, orientation, and pattern. The use of texture should be limited since it is limited to perception at a very close range and with very few objects. An example of texture is seen on figure 2.6.

Figure 2.6: An example of morphing between two basic textures.

Transparency
Transparency may describe a feature through its degree of visibility. An example of transparency is seen on figure 2.7.

Figure 2.7: An example of three levels of transparency.

Specular
Specular may describe a feature through its degree of reflecting light. An example of specular is seen in figure 2.8 on the next page.

Spatial relations

Spatial relations between the objects and the analyst determines the spatial distribution. Two concepts may be used to describe spatial relations: View point dependency, and view point distance.

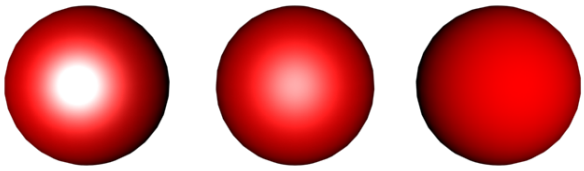

Figure 2.8: An example of three levels of specular.

View point dependency
To detect visual tendencies, the view point and the view direction of the analyst relative to the objects, is of great importance. The spatial distribution will change depending on the view point. Object properties such as shape and pose are also view point dependent.

View point distance
The view point distance is the distance from the analyst to the objects observed. This distance may vary as the analyst navigates in the 3D space. Properties of the objects change as the distance vary. Large distances result in the objects getting smaller and thus details on the objects are reduced and eventually they disappear. According to [Granum and Musaeus,] the distances may be defined as:

Close range Stereoscopic vision is dominating, and details of objects may be distinguished. This extends up to about 5 m.

Intermediate range "Larger" object properties may be noticed. This range is between 5 and 20 m.

Far range Only very prominent object properties can be perceived and may eventually vanish.

In order to distinguish between the three distances the objects may be available as several models. Furthermore, the objects may be collected into clusters of objects when observed at great distance.

Temporal relations

When there are changes over time between the spatial relations between the analyst and the objects, temporal relations occur. Two concepts may be used to describe temporal relations: Navigating in a static 3D space, and temporally varying 3D space.

Navigating in a static 3D space
When navigating in 3D space two situations may occur. Either the analyst is moving around between the objects or the objects are moving around the analyst. The latter situation refers to a stationary analyst and a dynamic 3D space. For both situations three forms of 3D motion may be described according to [Granum and Musaeus,]:

Radial motion refers to motion in depth where the observer is moving away from or toward an object.

Lateral motion refers to the observer moving sideways in the frontal plane. The perspective transformation of the analyst causes the proximal points to have higher relative velocities on retina (image plane) than distant points. This phenomenon is called motion parallax, and gives the observer information about the relative depth of points.

Rotational motion refers to the analyst circling around the object. Perceptually this corresponds to the object being turned around in front of the analyst. The object can refer to a single object or the entire 3D coordinate system with all its objects.

Navigating in a temporally varying 3D space

Objects in 3D space may have individual movements such as rotation and vibration. This can be used on data where temporal sampling is an attribute of the statistical information and therefore should be visualised in order to show temporal relations.

CHAPTER 3

Virtual Reality

The possibilities of Visual Data Mining (VDM) are increasing with the development of Virtual Reality (VR), which enables the user to immerse and interact in a real-time visual presentation of data sets. This chapter will describe the concept of virtual reality and a software framework, VR++, for hosting VR-applications.

3.1 Definition of virtual reality

Source: [Sherman and Craig, 2003]
The word **virtual** is for instance used in computing systems as extensions of the hardware, emulating the real thing through another source. It is also used to simulate virtual worlds [1] where objects exist virtually in that world. These objects are merely images of the physical object they represents. The word **reality** is defined in different ways. [Sherman and Craig, 2003] simplifies it for the purpose of virtual reality, and defines it as a place that exists and that can be experienced.

The two words combined is virtual reality, which is the simulation of a real or imagined environment, that can be experienced visually in the three dimensions of width, height, and depth and that may additionally provide an interactive experience visually in full real-time motion with sound and other forms of feedback. The simplest form of virtual reality is a 3D image that can be explored interactively at a personal computer, usually by manipulating keys or the mouse so that the content of the image moves in some direction or zooms in or out. As the images become larger and interactive controls more complex, the perception of "reality" increases. More sophisticated efforts involve such approaches as wrap-around display screens, actual rooms with wearable computers, and devices that lets the users feel that they are mentally present in the display images, for example joysticks and data gloves. In this way users can receive feedback from computer applications in the form of felt sensations in the hand or other parts of the body. Virtual reality can be described in terms of immersion, interaction, and real-time. This may be illustrated as shown in figure 3.1 on the following page.

Immersion is the feeling of being in the 3D virtual space. There are two states of immersion; mental - and physical immersion. Mental immersion is defined as being deeply engaged. The physical immersion is entering the medium of virtual reality. Stimulation of the body senses via the use of physical feedback devices.

Interaction is the possibility of moving in 3D space and manipulate with virtual objects.

Real-time is action which immediately can alter the virtual space.

[1] An imaginary space often manifested through a medium.

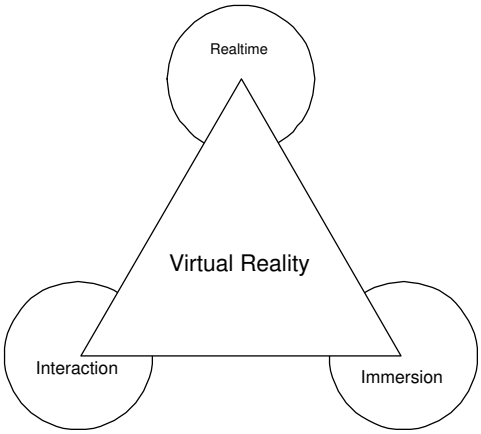

Figure 3.1: Defining virtual reality in a triangle form. Immersion, interaction and real-time are the elements which define virtual reality. [Thalmann, 2004]

3.1.1 Virtual reality features

Source: [Sherman and Craig, 2003]
Virtual reality is a unique medium because of the features. Summarising some of the features:

- The opportunity of being multiple simultaneous users.
- The ability to manipulate with time and space.

These features combined in the same medium gives a dynamic relationship between the participant and the medium. Figure 3.2 on the next page shows a user in a wrap-around display screen, also called a cave, interacting with the medium through stereo glasses. When viewed through lightweight stereo glasses, the left/right stereo images are presented separately to the left and right eyes respectively, producing the illusion of 3D objects appearing both within and beyond the walls of the Cave. The images are presented with reference to the users view point, which is continuously updated via a head-tracking unit; thus even as the user moves around in a Cave the environment displayed will always be in a correct perspective. The Cave shown in the figure is located at Aalborg University and is specially suited for research and design development.

3.2 VR++

Source:[Nagel and Granum, 2002]
In virtual reality many software and hardware parts have to work together, to facilitate different ways of presenting virtual reality and handling user interaction.

VR++ is an open-source framework for creating modular VR applications and consists of a set of libraries which make it easier for the user to make different applications. These libraries, also called add-on packages, contain applications ready to execute as sound-, graphic-, datapackages etc.. VR++ is a distributed system, based on the master/slave principle, which coordinates work amongst multiple processes. To fully exploit the possibilities of distributed computing, these processes can be placed on different machines. In the system a single master distributes the various processes to multiple slaves.

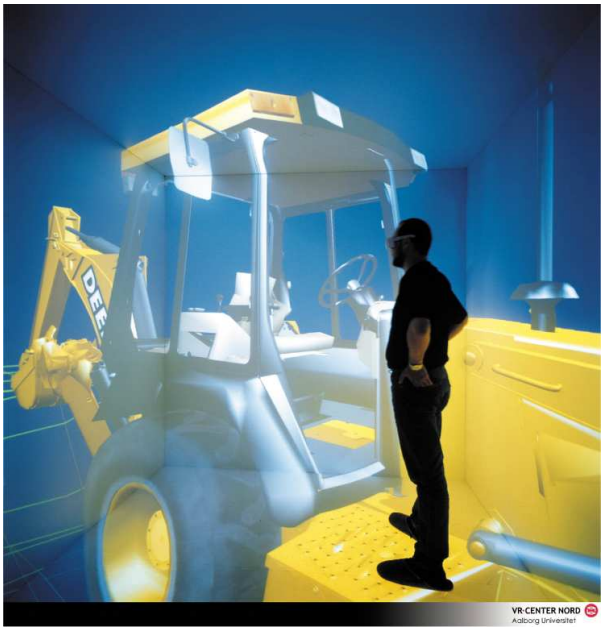

Figure 3.2: A user in a wrap-around display screen interacting with the medium through stereo glasses. Source: [vrl, 2002].

Each slave processes an assignment by the use of a certain task. A task is an implementation of an algorithm and is given processing time by the slave to which it belongs. The tasks communicate with each other through a standard data interface, so that tasks developed for VR++ automatically becomes compatible with each other and thereby make it easier for different users to integrate several projects.

3.3 Surface based rendering

Source: [Shreiner et al., 2003]
As mentioned before 2D geometric primitives is how a 3D object is represented. All geometric primitives are eventually described in terms of their vertices - coordinates that define the points themselves, the endpoints of line segments, or the corners of polygons (triangle).

Three common geometrically (surface) based graphical representations are the polygonal, which is mentioned in appendix A, non-uniform rational B-splines (NURBS), and constructive solid geometry (CSG) schemes.

Polygonal can be used to represent shapes described by NURBS and CSG, although with some loss of information. Many algorithms designed to speed up polygonal rendering methods have been integrated into hardware geometry engines and, as a result, hardware graphical rendering systems almost exclusively make use of the polygonal method.

NURBS are parametrically defined shapes that can be used to describe curved objects.

CSG are objects created by adding and subtracting simple shapes (spheres, cubes, cylinders, etc.).

3.4 Non-surface Based Rendering

Source: [Volumizer, 2004]
Surface based methods work best with solid, non transparent objects. When surfaces are transparent, surface based rendering techniques may not be the best choice; this is particularly true when a space is occupied by varying densities of semi translucent material (e.g., patchy fog or the human body when viewed via X-rays or MRI scans). In such cases, non-surface based methods may offer certain advantages when representing objects in a computer based virtual world.

3.4.1 Volume Rendering

Volume Rendering is well suited for rendering semi transparent objects. Volume Rendering is the process of generating an image directly from the volume data without the generation of an intermediate geometric model. Typically this is done by mapping the data values in the volume to the colour and opacity of an imaginary semi-transparent material, and then rendering an image of this material. Data values of interest can be assigned high opacity values and a specific colour to highlight their location within the volume while other data values can be assigned low opacity values to reduce their visual importance. It is also possible to render geometric and volumetric primitives together, allowing the inclusion of geometric primitives such as coordinate axes and other reference objects. This technique is useful for displaying relationships between areas of interest that are not well defined in a geometric sense. It is also useful for displaying the volume around areas of geometric interest, such as the volume near an isosurface.

In the following, four Volume Rendering techniques are described in terms of Fourier Volume Rendering, Splatting, Shear Warp Factorisation and Volume Slicing.

Fourier Volume Rendering

Source: [Lichtenbelt, 1995]
Fourier Volume Rendering (FVR) computes projection of a three dimensional data set and is faster than a conventional volume rendering method, because the complexity is reduced. This is done by first transforming the data set into the frequency domain by either a 3D Fast Fourier Transform (FFT) or the Fast Harley Transform (FHT). Then the projection image is generated directly from the volume data at any viewing angle by resampling along a plane perpendicular to the viewing direction, and taking the inverse 2D transform of the resampled plane. The projection can be used to visualise 3D data sets. The Fourier projection slice theorem also holds in higher dimensions. For the 3D case it can be stated as follows: The 2D Fourier transform of a 2D projection of a 3D object, at an arbitrary angle, is a 2D plane passing through the origin of the 3D Fourier transform of that 3D object, at the same angle. A projection is a mathematical operation, to be compared with taking an X-ray picture of a 3D object. A ray is cast through the data set, and samples are taken along the ray. Those samples are composited into a pixel value and projected on the screen.

The advantage of FVR is that the 3D Fourier transform only has to be computed once.

Splatting

Source: [Silva, 1995]
Another volume rendering technique is to reconstruct an image from the object space to the image space, by computing for every voxel in the data set, its contribution to the final image. This technique is called Splatting. The projection of each voxel onto the final image leaves a fuzzy impression, called a footprint or a splat. Splatting classifies and shades the voxels prior to projection, and thus each voxel splat is weighted by the assigned voxel colour and opacity. The resulting image is a uniform screen image for homogeneous object regions. However, due to to fuzzy appearance of the splats the edges of the object appears blurry.

Shear Warp Factorisation

Source: [Lacroute and Levoy, 1993]
This algorithm combines the advantages of image-order and object-order algorithms. Image-order algorithms operate by casting rays from each image pixel and processing the voxels along each ray. The advantage is efficient and high quality resampling, but the processing order also has the disadvantage that the spatial data structure must be traversed once for every ray, resulting in redundant computation. In contrast, object-order algorithms operate by splatting voxels into the image while streaming through the volume data in storage order, which speeds up the process. Another advantage is the simpler addressing arithmetic. The disadvantage of the processing order is that it is harder to implement effective optimisation in ray-casting algorithms.

Mapping from object space to image space requires high quality filtering and projection in the object-order algorithm and extra addressing arithmetic in the image-order algorithm. To avoid these problems it is being solved by transforming the volume to an intermediate coordinate system, a sheared object space. This is based on a factorisation of the viewing matrix into a 3D shear parallel to the slices of the volume data, a projection to form a distorted intermediate image, and a 2D warp to produce the final image, as shown in 3.3. For a perspective projection each slice is scaled.

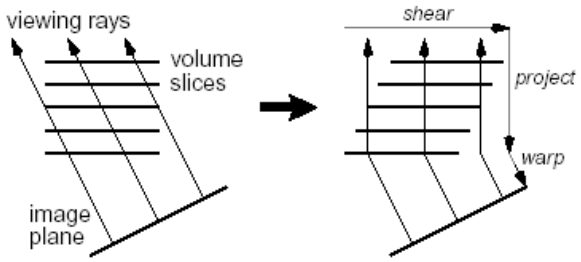

Figure 3.3: To transform a volume into sheared object space for a parallel projection, the slices has to be translated. Source: [Lacroute and Levoy, 1993].

The advantage of Shear Warp Factorisation is that scanlines of the volume data and scanlines of the intermediate image are always aligned, which can be seen in figure 3.4 on the next page.

The Shear Warp Factorisation allows implementation of coherence optimisations for both the volume data and the image with low computational overhead, because both data structures can be traversed simultaneously in scanline order.

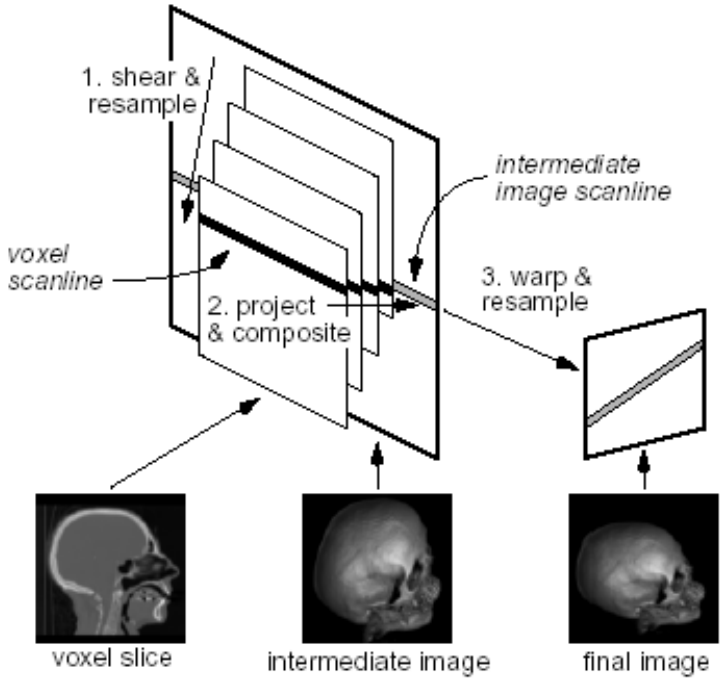

Figure 3.4: First the volume is transformed by translating and resampling each slice. Then voxel scanlines are projected onto the intermediate image scanlines. Finally the intermediate image is warped into the final image. Source: [Lacroute and Levoy, 1993].

CHAPTER 4

Current research

This chapter describes other projects researching 3D visual data mining using volumetric objects in 3D space.

The following sections outlines the methods described in five different research papers. However, the four papers describe the research done on the Stereoscopic Field Analyzer, and thus are all desribed in one section.

Iconic Techniques for Feature Visualization

Source: [Post et al., 1995]

The paper studies a conceptual framework and a process model for feature extraction and iconic visualisation. The visualisation of the icons is based on the pipeline shown in figure 4.1.

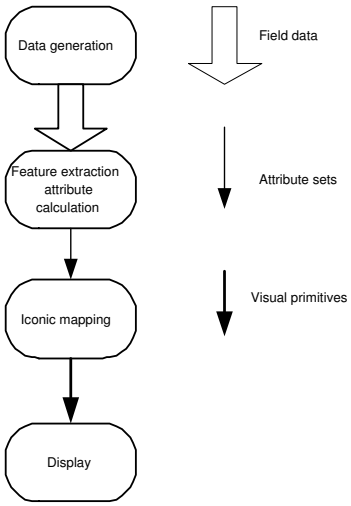

Figure 4.1: The pipeline used to visualise icons. Source: [Post et al., 1995].

For each data set features are extracted, if they are of some relevans for the system. For each of these features attributes are calculated. An attribute is a characteristic parameter of the feature. In the studie the result of the feature extraction and attribute calculation is a set of attributes, which characterize the feature. The iconic mapping of an attribute set is a mapping of the attributes onto the parameters of icons. This is done in order to visualise features by objects in a clear perceivable way, in which there should be some similarity relation between the features and their iconic representations. [Post et al., 1995] divides the attributes into three different categories: Geometric/Morphological attributes, data attributes, and combined morphological/data attributes.

Geometric/morphological attributes describe the spatial propoerties of a feature such as height, width,

volume, centroid, and shape.

Data attributes describe the properties of the data over a feature such as minima and maxima.

Combined morphological/data attributes describe both spatial and data properties of a feature such as center of gravity in a density field.

The modelling of icons is performed by a modelling language developed by [Post et al., 1995] with which the geometrical primitives can be defined and the parameters of these primitives can be bound to the attribute set. The construction of the geometric primitives is performed in three steps, which, by [Post et al., 1995], are:

1. A 2D contour in the x-y plane is designed and colours or colour-maps are bound to this contour. This contour can consist of line-segments or parametric functions. The coordinates of the line-segments, the constants of the parametric functions, and the colours or colour-maps can be bound to the attribute set.

2. This contour is extended to a 3D object by performing a rotation sweep around the x or y axis, performing a translation sweep by moving the contour along an axis over a given length, or performing a general sweep along a arbitrary 3D trajectory. The parameters of the function that defines the 3D trajectory can also be bound to the attributes.

3. This 3D object is scaled, translated, rotated or deformed. In this way the objects can be put on their location in the data set or to their relative location in a complex object. The values defining the rotation, translation, scaling or deformation are also bound to the attribute set.

Stereoscopic Field Analyzer

The studies regarding the stereoscopic Field Analyzer (SFA) have been performed through several years by several different people. The SFA allows the visualisation of both regular and irregular grid of volumetric data and combines glyph based volume rendering with two-handed minimally-immersive interaction metaphor. The SFA is able to encode up to 8 attributes per glyph, based on the glyph's location, 3D size, colour, and opacity. Four papers dealing with the SFA have been analysed. These are:

- Data Visualization Using Automatic, Perceptually-Motivated Shapes, [Shaw et al., 1998].
- Minimally-immersive Interactive Volumetric Information Visualization, [Ebert et al., 1996a].
- Two-handed Interactive Stereoscopic Visualization, [Ebert et al., 1996b].
- Procedural Shape Generation for Multi-dimensional Data Visualization, [Ebert et al., 1999].

In the following the studies regarding volume rendering and visualisation of glyphs in each paper will be described shortly.

Data Visualization Using Automatic, Perceptually-Motivated Shapes

Source: [Shaw et al., 1998]

This paper studies the extension of SFA by glyph rendering with other visually salient features to increase the number of data dimensions to be visualised simultaneously. The feature applied to the SFA is shape in terms of superquadrics. The shape of an object in 3D is formed by the objects curvatures. [Shaw et al., 1998] states that the curvature has a visual order, since a surface of higher curvature look more jagged than a surface of low curvature. Furthermore it is stated that a generation of glyphs, which interpolate between extremes of curvature allow the user to read scalar values from the glyph's shape. Each glyph in the extension uses normalized data in the range [0,1], which is mapped into the domain of the corresponding glyph attribute.

Minimally-immersive Interactive Volumetric Information Visualization

Source: [Ebert et al., 1996a]

This paper studies the general concept of SFA regarding information visualisation. The information visualisation is performed using glyphs's location, size, shape, colour, and opacity. It is stated that the system minimizes the occlusion problems of surface rendering and some volume rendering through the use of glyph rendering, transparency, interactivity, stereo-viewing, interactive manipulation, and interactive volume segmentation and subsetting. Furthermore the SFA allows several different data sets to be visualised at the same time in the same coordinate system, making comparison of trends among related data sets possible.

Two-handed Interactive Stereoscopic Visualization

Source: [Ebert et al., 1996b]

This paper studies the general concept of SFA regarding user control. SFA offers control of the visualisation in numerous ways, including selection of the glyph type and data set mapped to glyph colour. In order to control the colour, three colour tables are used, depending on the properties of the data file and the number of data files being displayed. For grids with only positive or negative scalar values, any pseudocolour table may be used. If both negative and positive values are to be displayed, the scalar value 0.0 is mapped to colour 128 and the maximum magnitude is mapped to colour 255. This results in a symmetric colour table about the 128th entry. The third colour table is used for displaying several data sets a once. It is possible to map one colour to each data set allowing the data sets to be visually separated. [Ebert et al., 1996b] states that the user typically choose a distinguishable hue for each data file, with the colour ramping from low to high saturation. Furthermore, the user is able to choose to concentrate on a particular subset of the volume and to choose only the n^{th} grid points to be displayed, reducing the number of glyphs drawn.

Procedural Shape Generation for Multi-dimensional Data Visualization

Source: [Ebert et al., 1999]

This paper studies the extension of SFA by glyph rendering using shape as a feature. As mentioned in [Shaw et al., 1998], the curvature information of a shape may help distinguish shapes from one another. The paper studies three different procedural techniques for the generation of glyph shapes: Fractal

detail, superquadrics, and implicit surfaces. The main idea of fractal detail is to generate distorted versions of a basic shape e.g. where low data values map the basic shape and high data values map to very perturbed shapes. In order to create the deviation of the basic shape, displacement mapping is used. [Ebert et al., 1999] suggests the use of superquadrics which can allow from one to two data dimensions to be visualised with shape. Superquadrics come in four main families: Hyperboloid of one sheet, hyperboloid of two sheets, ellipsoid, and toroid. [Ebert et al., 1999] chooses superellipses for implementation due to their familiarity. Furthermore [Ebert et al., 1999] suggests that supertoroids could be used to represent negative values of a data set and superellipsoids could represent positive values. Using the superellipses, exponents are assigned to the trigonometric terms of these. The exponents allow continous control over the characteristics of the shape in the two major planes, which intersect to form the shape. In figure 4.2 the varying of the exponents resulting in a smooth, understandable transition in shape, can be seen.

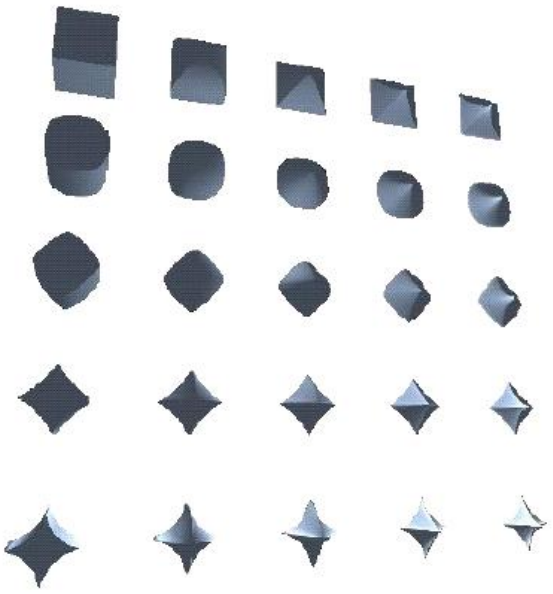

Figure 4.2: Variation of the exponents in superquadrics. Source: [Ebert et al., 1999].

Implicit surfaces use an underlying density field to represent volumetric information. Implicit techniques provide smooth blending functions for individual implicit field sources. Isosurface generation techniques are used to create a geometric representation of this volumetric density field. With the implicit shape visualisation it is possible to map up to 14 data dimensions to uniformly spaced vectors emerging form the center of a sphere. The length of the vectors is scaled based on the data value being visualised.

CHAPTER 5

Current software

This chapter describes the current software available for use in the field of 3D visual data mining systems, and software available for volume visualisation applications.

The following sections describe the 3DVDM system, which holds applications for virtual reality in the field of visual data mining, and the OpenGL volumizer which is an application programming interface designed for volume visualisation applications.

5.1 The 3DVDM system

Source: [Nagel et al.,]
Technology to store and process large amounts of data has during the last few decades improved dramatically. This has led many companies to store ever increasing amounts of customer information in large databases. The hope has been that it would be possible to discover unknown relationships in the data, and thereby obtain a knowledge which can give commercial advantages. However, finding hidden relationships in large amounts of data is not easy. Purely numerical methods have been supplemented with visual methods. This has led to the emergence of Visual Data Mining (VDM).

The 3DVDM system extends the field of VDM with virtual reality techniques to create new methods for exploration of data in VR. The system is able to mine large databases by combining facilities for advanced VR with expertise in database systems, statistical analysis, perception psychology and visualisation. Researchers in the database area are responsible for handling and delivering large amounts of data from databases. Statisticians are responsible for finding interesting problems, creating statistical models for data analysis, and guiding the entire project in experiments of 3DVDM system. Perception psychology researchers are responsible for finding ways of good interaction and human perception of the VR environment.

The 3DVDM system is a collection of VR++ applications, which is a software framework, and is explained in section 3.2 on page 22. The 3DVDM system is based on a visual world, where statistical observations are represented in a 3D scatter plot and shown in either a 6-sided Cave, a 180° Panorama, and on regular computer monitors. Each data point in the plot is shown as a visual object, such that statistical variables may also control various visual object properties such as object shape, orientation, colour, and surface texture. The purpose is to present data from within and to allow the human observer as a visual explorer to navigate the visual world, in order to inspect data in arbitrary view directions and from arbitrary viewpoints.

5.2 SGI OpenGL Volumizer

Source: [Volumizer, 2004]
The OpenGL Volumizer is an application programming interface (API) created by Silicon Graphics Inc. (SGI), enabling a programmer to create hardware optimised applications for several platforms with a

single code base. OpenGL Volumizer is designed for volume visualisation applications and is a toolkit built on top of OpenGL. Volumizer uses a technique similar to ray casting, called volume slicing, to leverage the texture mapping hardware many workstations now have.

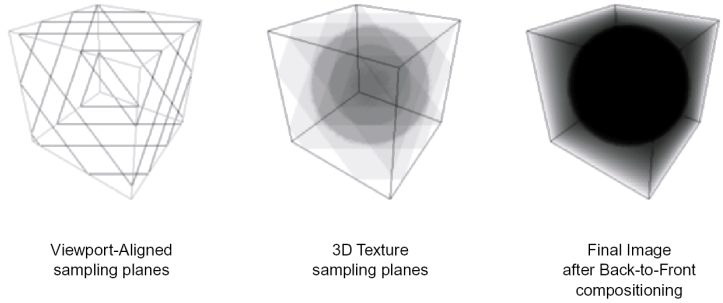

Figure 5.1: Processing a volume through volume slicing. Source: [Volumizer, 2004].

In Volumizer, all points on a plane, orthogonal to the line of sight, are computed sequentially in the texture mapping hardware.

In this technique the volume is sampled in surfaces orthogonal to the viewing direction as shown in figure 5.1. When the rays have intersected the points along one plane the volume is processed. The distance is incremented and the processing occurs again from all points on all the rays in the new plane. Processing the points continues until the plane of points moves beyond the viewing frustum, at which point the processing terminates.

The advantage of volume slicing is that it is faster than ray casting because computations are performed by the dedicated texture mapping hardware, whereas ray casting is performed by the CPU. For further details about SGI OpenGL Volumizer, see appendix B.

Part II

System Requirements

CHAPTER 6

Analysis of Requirements

In this chapter the delimitations, perceptual requirements, and technical requirements of the visual data mining system are defined.

Delimitations of the Project

Due to lack of resources this project can not cover all aspects of the various volume rendering and visual data mining techniques. Therefore, only volume slicing and shear warping will be analysed as volume rendering techniques. Furthermore, the only data mining technique that will be analysed is 3D visual data mining. The analysis of visualisation of objects is founded on earlier research, as perception psychology is beyond the scope of this project. Given the project unit description for 8th semester CVG, virtual reality and the VR++ framework will not be further analysed. Furthermore, due to lack of resources, the graphical user interface and the various techniques used to create a graphical user interface will not be analysed, and thus only a simple graphical user interface will be designed and implemented.

Perceptual Requirements

The perceptual requirements for the system are:

- From a perceptual point of view, the system should express as many feature dimensions as perceptually clear as possible.
- The level of detail of the objects should be high enough not to reveal that the objects are actually simple geometry with transparent textures.
- When navigating through the 3D space, the perceptual requirement is a frame rate, that gives the impression of flying between the objects.

Technical Requirements

Rendering framework

When using hardware accelerated graphics on several different operating systems, only one Application Programming Interface (API) exists, Open Graphics Library (OpenGL). To minimise problems when using the system with different operating systems, using OpenGL in conjunction with the windowing package Graphics Library Toolkit (GLUT) is ideal for ensuring minimal operating system dependent adaptations.

When visualising database records as objects in 3D space, there are two possibilities: Rendering the entire 3D space as one volume, or rendering each object as a volume.

Rendering the entire 3D space as one volume

The approach of rendering the entire 3D space as one volume results in a set of requirements to the visualisation system. The primary requirement is for the resolution of the volume. Each object is a very small part of the entire volume, and if a large amount of details for each object is required, the resolution for the entire volume has to be very high. If the resolution of the volume is very high, the requirements to the memory used to store the volume is proportionally high, as is the CPU time required to render the scene.

The OpenGL-language can only handle textures with a maximum resolution of 256x256. This means that in order to visualise a high resolution volume, with a great level of detail, the system is required to subdivide the rendering of the volume into subregions, in order to maintain an acceptable level of detail.

The advantage of rendering the 3D space as one volume is that once the 3D space volume has been created, the system can handle an infinite amount of objects, as the requirements to the system remain the same no matter how many objects that are to be visualised. This ensures the designer total freedom in the possibilities of expression.

Rendering each object as a volume

When utilising the one volume per object approach, the requirements to the system memory is proportional with the amount of objects and the resolution of each volume. This means that the amount of available system memory sets the limitation to the amount of objects and their resolution.

The rendering time for this approach may, if all objects are rendered with the highest level of detail, be very high. If the level of detail for each object and the level of geometry complexity are reduced, it is possible that the navigation in the 3D space can be optimised to show the objects while navigating.

Discussion

The perceptual requirements are the basis for the selection of method. The following table is a summary of the advantages and disadvantages for the "one volume" approach.

Advantages	Disadvantages
-No limitation in the amount of objects visualised	-With high level of detail the memory requirements are high
-Always uses the same amount of memory independent of the number of objects	-Navigation will be very CPU intensive = low frame rate

A similar table can be set up for the "one volume per object" approach.

Advantages	Disadvantages
-Level of detail can be varied for each object -Navigation may be possible without too much CPU time	-The memory sets a limit to the amount of objects

Both methods are equally effective, in being expressive for each object. However, the "one volume" method may have an infinite amount of objects, whereas the "one volume per object" method has a finite amount in terms of memory usage.
If the "one volume" method is to be utilised with a high level of details, the amount of memory required to visualise the objects is very large.

To exemplify this the following comparison a scenario has been created:

Example: The values used in this example are just for this example, and are not requirements for the system.
If, for each object, a voxel space with the dimensions $64 \times 64 \times 64$ is required, and the ratio between the object size and the 3D space is set to $1 : 128$, the "one volume" approach requires a voxelspace with $(64 \cdot 128)^3 \approx 550 \cdot 10^9$ voxels. If each voxel is a 32-bit value the required memory to store the volume is:

$$\frac{32 \cdot (64 \cdot 128)^3}{8 \cdot 1024 \cdot 1024 \cdot 1024 \cdot 1024} = 2\text{TB}$$

For the same amount of memory the "one volume per object" approach will with the same dimensions for each object, be able to store $128^3 = 2,097,152$ different records.

As the example shows, the only case where the "one volume" approach has its advantage memory-wise, is when the 3D space is filled to the limit with objects. The "one volume" stores a lot of waste space, whereas the "one volume per object" only saves the voxels with a content. Though the "one volume" approach is more flexible as to the amount of objects, the "one volume per object" approach is more efficient in its storage of these.

When the system navigates through the 3D space, the "one volume" method is not very efficient, as the system has to evaluate a large number of voxels for each frame. However the "one volume per object" method has wide possibilities to degrade the level of details on each object when navigating, or simply just displaying a bounding box for each object.

6.0.1 Conclusion

The "one volume per object" approach is through the above comparisons the best method of visualising the 3D space. With this approach each object can have a high level of detail, that expresses many features, dependent on how the visualised objects are designed. The approach allows a creative solution on the navigation problem, as each object can be rendered separately. The "one volume per object" approach is memory-effective, it only stores volumes for the object and not for the space with no objects.

Based on this evaluation, the "one volume per object" approach will be chosen as the method with which the objects are to be visualised. If this method is implemented dynamically, it can also be used

to support the "one volume" approach.
For implementation OpenGL will be used.

CHAPTER 7

Specification of Requirements

Introduction

In the specification of requirements a description of the system, which systematically sets out the requirements, is prepared. The requirements are based on the problem domain, the analysis of requirements. The specification of requirements is based on the SPU method (Struktureret Program Udvikling). Source: [Biering-Sørensen et al., 1988]

Purpose

The initial hypothesis of this project is that the ability to display advanced objects in a 3D visual data mining system will aid the process of exploring large data sets.

The hypothesis entails developing a rendering engine, able to display advanced graphical objects. For this engine an OpenGL based volume rendering engine is the basis.

References

The requirements in this specification are based on the analysis of requirements, see chapter 6 on page 34.

General description

In the following the system will be described in terms of the functionality of the programme, the limitations of the programme, and the future of the programme.

Description of the system

The aim of the project is to develop a system with the ability to display advanced objects in a 3D visual data mining system. In order to do this a volume rendering engine has to be developed. Volume rendering is a technique which allows rendering of complex shapes from image volumes. The appearance of the advanced objects is controlled by the object features to which data dimensions are mapped. The user must be able to remap the data dimensions to other features. Furthermore, the user has to be able to navigate through the 3D space.

The functionality of the programme

The overall structure of the programme may be seen in figure 7.1. The input to the system is normalised data from a database, with values between 0 and 1. The normalised data is mapped to visual structures depending on the various features of each record. The user of the system will be prompted to choose which feature dimensions to be visualised and how and furthermore the user has to be able to remap dimensions of the data set to features. The final structures have to be visualised as volumes in 3D space.

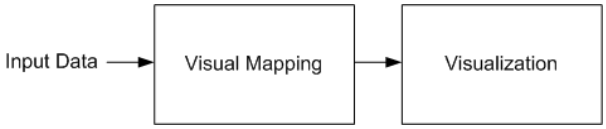

Figure 7.1: The overall structure of the programme.

For each data set a specified number of features must be shown on a specified type of object. As described in the analysis of requirements, chapter 6 on page 34, the system will be made so that each object is rendered as one volume. Due to this, the programme has to be capable of rendering each volume and of producing a frame rate, which gives the impression of flying when navigating through the 3D space. Depending on the placement of the view point, the geometry representing each object must be reduced proportionally to the distance of the view point. The system utilises orthogonal projection as it is not important to get a perspective view of each record in order to get a good perceptual understanding of the entire 3D space.

Based on the description of the functionality of the programme, the overall structure of the programme may be described by the flow diagram seen in figure 7.2

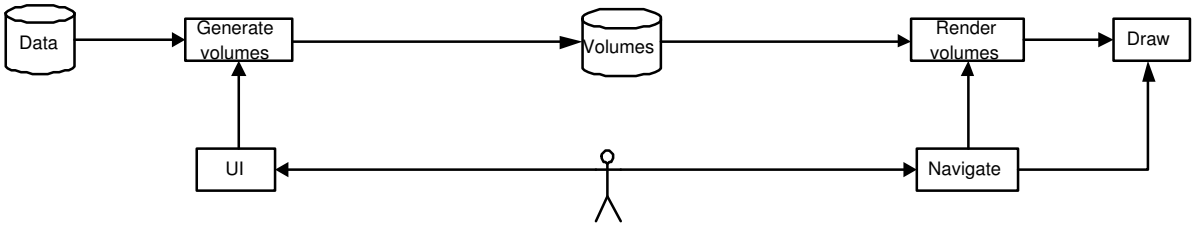

Figure 7.2: The overall flow diagram of the programme.

The limitations of the programme

When navigating through the 3D space, the system is limited to only rendering simple geometry for each volume. This is done to display the volumes in real-time.

The future of the programme

The programme is developed so that it may be used as a general component in the VR++ software framework. This way it may be used to visualise any kind of data. Furthermore, the programme is to be integrated with the 3DVDM system, so this system can visualise more complex shapes.

Specific requirements

Definitions

The following definitions will be used in the programme:

- The resolution of the voxelspaces and the dimensions of each voxelspace in 3D space will be user defined.
- The colour channels applied to the voxelspaces are 32 bit RGBA (Red Green Blue Alpha, channel order).
- The ratio between the object size and the 3D space will be user defined.
- The requested frame rate during navigation is 15 fps.

Functional requirements

Each step of the programme shown in figure 7.2 on the preceding page will be described in the following in terms of input, functionality, and output.

User interface

Input
Interaction from the user of the system.
Functionality
Allows user to determine object type, how data is mapped to object features, and which features that are to be visualised.
Output
A definition of which dimensions that are to be described with which features.

Generate volumes

Input
A definition of which data dimensions that are to be mapped to which object features. Furthermore, the step has normalised data as input.
Functionality
Generates the visual structures for each record and stores them as volumes.
Output
Volumes.

Volume

Input
Volumes generated by the generator.
Functionality

Storage for volumes.
Output
Volumes.

Render volumes

Input
Volumes. View point. Notification of end of navigation. Level of detail.
Functionality
This step makes the view point specific geometry and texture calculations for all volumes in a specified detail level.
Output
Textures and geometric objects.

Navigate

Input
View point. Interaction from the user, indicating movement within the visualisation
Functionality
This step makes new geometry and texture calculations for all volumes in low detail, to maintain a high frame rate during navigation. When navigation stops, the **Render volumes** step is notified to make calculations.
Output
Textures and geometric objects. Notification to **Render volumes** step.

Draw

Input
Textures and geometric objects.
Functionality
Draws the geometry and texturizes it.
Output
A visualisation of the data.

External user interface requirements

User interface

The functional requirements for the user interface are specified in four main functionalities:

Object: User is prompted to choose between various visualisation objects.

Feature mapping: User is prompted to choose which data dimensions are mapped to object features.

Number of records: User is prompted to choose the number of records to be visualised.

Mouse clicks: The system has to be able to detect mouse clicks on objects within the 3D window.

A more specific description of the user interface may be seen in chapter 12, 12.6.

CHAPTER 8

Acceptance test specification

When the programme has been implemented an acceptance test of the system is performed. The purpose of the acceptance test is to control the functionality of the programme specified in the specification of requirements, chapter 7, section 7.

The specific requirements for the system are:

- The structures have to be visualised as volumes in 3D space.
- User is prompted to choose between various visualisation objects.
- User is prompted to choose which data dimensions are mapped to object features.
- User is prompted to choose the amount of records to be visualised.
- The user has to be able to navigate through a visual representation of the a given data set.
- The requested frame rate during navigation is 15 fps.

Test cases

The test cases are divided into three groups: functionality, performance, and object perception.

Functionality

The functionality of the programme is tested through two test cases. The two test cases are:

Mapping is testing a normalised data set, which consists of 50 records with 4 data dimensions for each record. If the data dimensions are mapped correctly to the features the test is successful.

Continous remapping involves testing whether continous remapping of data dimensions to object features is possible. The test is a success if remapping occurs correctly.

Performance

The performance of the programme is tested through two test cases. The two test cases are:

Frame rate the frame rate is tested in both navigational mode and high definition mode, with a scene containing 50 glyphs and a scene containing 1000 glyphs.

Reslicing the reslicing time for a scene containing 50 glyphs and for a scene containing 50 glyphs is tested.

Perception

The perception of the programme is tested through one test cases. The test case is:

Object perception the object perception in the system is tested using select test cases from the article [Raja et al., 2004].

Part III

Analysis

Introduction

The aim of the project is to allow display of advanced objects with a 3D visualdata mining system. Some advantages of combining volume rendering with visual data mining are the ability to display advanced objects both interior and exterior, which increases the possibility of mapping multi-dimensional data sets to features.

When developing a system, which is capable of displaying advanced objects, three main concepts are to be analysed:

- Perceptual possibilities
- Conceptual possibilities
- Performance

In the following the three concepts will be described briefly. Furthermore, a graphical user interface for the project will be described.

Perceptual possibilities

The perceptual possibilities will be analysed by defining three test objects to be used in the project based on various visualisation features. The mapping of the features onto an object is founded on earlier research, as perception psychology is beyond the scope of this project.

Conceptual possibilities

To visualise the objects a volume rendering technique will be used. The fundamentals of this technique is to slice each volume into a number of planes to which a texture is mapped. The volume rendering techniques being analysed are volume rendering using plane geometry, volume rendering using 2D texture stacks and shear warping. Additionally, shading methods will be analysed.

Performance

Using a volume rendering technique for visualising objects demands much CPU processing time due to the fact that each texture has to be re-calculated when the view point is changing. Therefore various methods for reducing the utilisation of memory are analysed.

CHAPTER 9

Visualisation of objects

The perceptual possibilities will be analysed by defining three test objects to be used in the project based on various visualisation features. The mapping of the features onto an object is founded on earlier research, as perception psychology is beyond the scope of this project.

The general concept of visual data mining is to map data from data tables onto a visual structure through human interaction. The concept may be seen in figure 9.1 and may be applied to several dimensions of an object space.

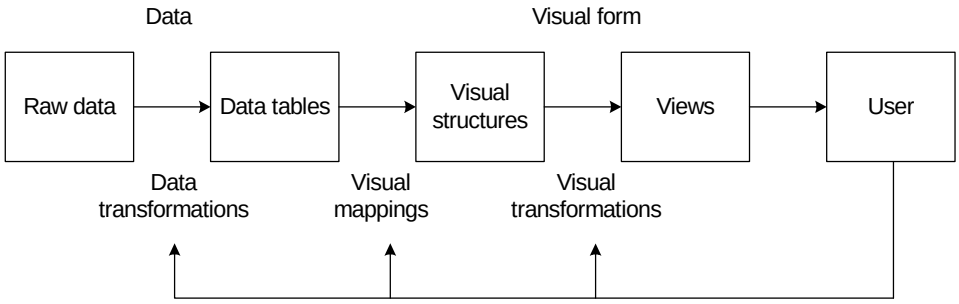

Figure 9.1: Visualisation of data achieved by human interaction. Source: [Card et al., 1999]

9.1 Visual structuring

When a multi-dimensional data set is to be visualised, it is important to have many visualisation parameters available in order to map as many of the dimensions as possible to a visual structure in 3D space. There are some parameters which are more perceivable than others by the user, therefore it is important to map the most significant dimensions with the strongest perceivable parameters. In chapter 4 on page 27 several visualisation techniques are purposed in the various papers. Furthermore, several visualisation parameters are described in chapter 2 on page 14. The techniques and the parameters will be used to construct test objects for the system. The test objects will be constructed based on the visualisation technique of developing distorted versions of a basic shape, proposed by [Ebert et al., 1999]. Furthermore, also is proposed by [Ebert et al., 1999], data values are mapped to the exponents, which controls the shapes overall flavour. This mapping will also be used in the construction of the test objects. For each test object is has been evaluated how many of the visualisation parameters that can be used.

9.2 Test objects

In 3D space objects representing data records are referred to as glyphs. The definition of a glyph is when the volume is rendered and shown in the 3D space. The definition of a volume is when it is still

represented in voxelspace. For each record it is assumed that the data has been scaled so the values lie between 0 and 1.

In the following three different glyphs will be proposed. The main idea for each proposed glyph is to have two different shapes that are significantly different from each other, and make a morphing between them according to a feature value. As specified in the specification of requirements, see chapter 7 on page 38, the three proposed glyphs are examples of how glyphs may be customised for the system.

9.2.1 Cross glyph

The basic shape for this glyph is a cube, and the morphing shape is a cube with the sides grown out as bulges. An example of this can be seen figure 9.2.

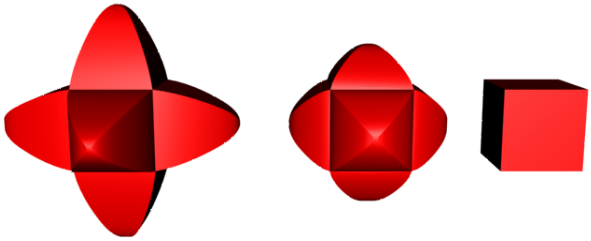

Figure 9.2: The cross glyph with bulges defined by the values of the data.

The cube has been chosen as basic shape upon the assumption that the basic shape should be one that is easily recognisable and simple. As the value of the data increases bulges appear on each side of the cube. These bulges have a set size according to the value of the data. A data value of 1 results in bulges that makes the glyph three times as long as the cube. By using this glyph it is possible to represent features using the following variables:

- Position - x, y, z coordinates, representing three features
- Colour - two dimensions, representing two features
- Transparency, representing one feature
- Shape, representing one feature
- Texture, representing one feature
- Sound, representing one feature
- Specular, representing one feature

For a detailed mathematical description of the cross glyph, see appendix C on page 107.

9.2.2 Diamond glyph

The glyphs's basic shape is a cylinder representing a data value of 1, see figure 9.3 on the next page. As for the cross glyph the basic shape is based on being easily recognisable and simple.

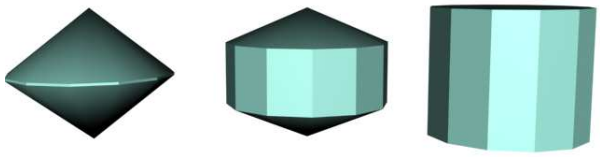

Figure 9.3: The diamond glyph.

As the value of the data increases the cylinder evolves a diamond shape. The diamond shape is based on the research paper done by [Ebert et al., 1999], see figure 4.2 on page 30, where a superquadric shape is used. By using this glyph it is possible to represent features by the following variables:

- Position - x, y, z coordinates, representing three features
- Colour - two dimensions, representing two features
- Transparency, representing one feature
- Rotation, representing one feature
- Shape, representing one feature
- Texture, representing one feature
- Sound, representing one feature
- Specular, representing one feature

For a detailed mathematical description of the diamond glyph, see appendix C on page 107.

9.2.3 Hourglass glyph

The basic shape of the glyph is a cylinder representing a data value of 1, see figure 9.4.

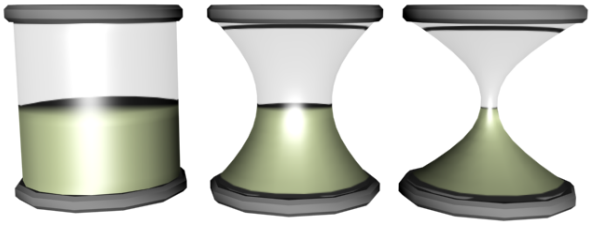

Figure 9.4: The hourglass glyph.

As for the cross glyph the basic shape is based on being easily recognised and simple. As the value of the data increases, the cylinder evolves an hourglass shape. The hourglass shape is an advanced type of shape but yet easily recognisable as an object. By using this object it is possible to represent features by the following variables:

- Position - x, y, z coordinates, representing three features

- Colour - two dimensions, representing two features, can be applied three places, making the total features six.
- Transparency, representing one feature
- Rotation, representing one feature
- Shape - glass-shape and sand level, representing two features
- Texture applied to the sand in the hourglass, representing one feature
- Sound, representing one feature
- Specular, representing one feature

For a detailed mathematical description of the hourglass glyph, see appendix C on page 107.

Discussion

Based on the analysis of the visualisation of objects three test objects have been created. All three objects are generated on the visualisation technique of developing distorted versions of a basic shape. Furthermore, data values are mapped to the exponents, which control the shapes overall flavour. The choice of the object properties used to generate the test objects is based on the assumption that it is possible to achieve a perceptually good visualisation of the data set where it is possible to detect visual tendencies.

CHAPTER 10

Volume Rendering

To visualise the objects a volume rendering technique will be used. The fundamentals of this technique is to slice each volume into a number of planes to which a texture is mapped. The volume rendering techniques being analysed are volume rendering using plane geometry, volume rendering using 2D texture stacks and shear warping. Additionally, shading methods will be analysed.

The method used in this project to represent a volume in 3D space, is volume rendering. Volume slicing is a technique of volume rendering, which utilises simple plane geometry. The fundamental idea is to place simple geometry perpendicular to the viewing angle inside the space in which the volume is represented. Onto these planes a texture is applied, which represents how the volume would look if it was sliced with the plane.

10.1 Volume rendering using plane geometry

The volume rendering technique allows rendering of complex shapes from image volumes, 3D bitmap images. A number of semi-transparent textures sampled from parallel planes in these volumes are generated and mapped onto a series of parallel geometric planes, perpendicular to the viewing angle.

For each glyph in the system, there is an image volumespace, that is represented in the 3D space as a number of planes limited within a bounding cube.

10.1.1 Generation of plane geometry

Source: [Madsen, 1997]
In order to generate the planes, the planes have to be mathematically represented. The mathematics in this section is based upon the assumption that the centre of the volume is situated in $(0, 0, 0)$ in the local coordinate system of the object.

The equation for the plane is:

$$a(x - x_0) + b(y - y_0) + c(z - z_0) + d = 0 \tag{10.1}$$

x_0, y_0 and z_0 are points on the plane, and are used to place the plane in 3D space. The normal to the plane is given by:

$$\vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{10.2}$$

The normal is equivalent to the vector from the viewing angle. Based on these equations, a plane-generator can be made. The centre plane of the volume is placed in $(0, 0, 0)$. This simplifies the equation for the centre plane to:

$$ax + by + cz = 0 \tag{10.3}$$

Since the planes are all being placed along the normal vector, the normal vector coordinates can all be used to place the other planes. If the normal vector is scaled for each placement of a plane, then the rest of the planes can be generated using:

$$ax + by + cz - a^2 - b^2 - c^2 = 0 \tag{10.4}$$

If the normal vector to the plane is set to unit length, the scalar to the centre points of the additional planes can be calculated using:

$$i \cdot \left(\frac{L}{I-1} \right) \cdot \vec{n} \tag{10.5}$$

where I is the total number of planes, i is the current plane number and L is the length from the centre to the edge of the bounding cube in the direction of the normal vector. The side-length of the bounding cube is denoted as ζ. L is calculated using standard Pythagoras:

$$L = \sqrt{(m \cdot a)^2 + (m \cdot b)^2 + (m \cdot c)^2} \tag{10.6}$$

where m is a scalar, that scales a, b and c to the length to the edge of the bounding cube:

$$m = \frac{\zeta}{2 \cdot l}$$

with

$$l = \max(a, b, c)$$

Example For this example the side length of the bounding cube ζ = 10 and the total number of planes, I = 10. The view point is set to $(2, 1, 5)$. With the given viewpoint the plane is given by:

$$2x + y + 5z - 4 - 1 - 25 = 0$$

$$2x + y + 5z - 30 = 0$$

$l = max(2, 1, 5) = 5$ and thus $m = \frac{10}{2 \cdot 5} = 1$. The length from the centre to the edge of the bounding cube in the normal vectors direction, L, is:

$$L = \sqrt{2^2 + 1^2 + 5^2} = \sqrt{30}$$

The normalised vector yeilds $\vec{n} = (\frac{2}{\sqrt{30}}, \frac{1}{\sqrt{30}}, \frac{5}{\sqrt{30}})^T$, and thus for plane number 2 the scalar is

$$2 \cdot (\frac{\sqrt{30}}{10-1}) \begin{pmatrix} \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{30}} \\ \frac{5}{\sqrt{30}} \end{pmatrix} = \begin{pmatrix} \frac{4}{9} \\ \frac{2\sqrt{30}}{135} \\ \frac{2}{27} \end{pmatrix}$$

Inserting the new scalar into equation 10.1 yeilds:

$$\frac{4}{9}x + \frac{2\sqrt{30}}{135}y + \frac{2}{27}z - \frac{764}{3645} = 0$$

10.1.2 Quads or Polygons

When placing geometry in 3D space using OpenGL, there are several primitives that can be utilised, such as triangles, quads and polygons, see appendix A on page 100 for further details on these. Since the textures for the planes have four corners, due to the use of the Cartesian coordinate system in images, the triangle primitive will not be considered any further.

The quad primitive consists of four points, which eases the texture mapping, as the corners of the quad matches the corners of the textured image. The placement of the quad inside the bounding cube, poses a challenge if quads is used creating the planes.

The polygon primitive consists of as many points as the programmer decides, which makes it a very flexible primitive. However, the fact that the number of points creating a plane can vary depending on the planes angle, the texturing of the planes poses a challenge.

A waste of parts of the image that should be textured can not be avoided for neither quads nor polygons.

10.1.3 Fitting the planes to bounding cube using quads

For this project two simple approaches has been utilised to fit the planes to bounding cube using quads: **2D Cutting** and **Cubic Plane Cutting**. The size of the bounding cube, which denotes the limits for the planes, is $\zeta x \zeta x \zeta$. Both methods are based on preliminary testing of creating quads from the bounding cube.

2D Cutting

In this approach the largest component of the normal vector $\vec{n}$ is found and the two remaining components denotes a 2D plane. These two directions is where the vector spans the least, and the thesis is that the plane will likely be oriented, in such a way that the 2D plane is the Cartesian plane that is closest to the viewpoint. When the proper 2D view port has been found, the four 2D corner coordinates of the bounding cube is applied to 10.1, and the result will be four 3D coordinates that make a plane perpendicular to the viewing angle.
This approach makes the best planes, when one vector component is significantly larger than the other two. When the three components are about the same size, the 2D Cutting results in a skewed plane because the cutting angle is far from the viewing angle, an example of this effect can be seen in figure 10.1 on the next page.

When several planes are generated the skewness problem becomes more apparent, as the most forward and backward planes would have more geometry outside of the bounding cube, than inside, an example of this can be seen in figure 10.2 on the following page. All planes in the figure are perpendicular to the viewing angle. When the planes are texturized the parts that are outside of the bounding cube will be invisible.

To solve the skewness problem, each plane could be checked for which corner points that was outside the bounding cube. If two or more adjacent points are outside the space, the planes could be scaled in

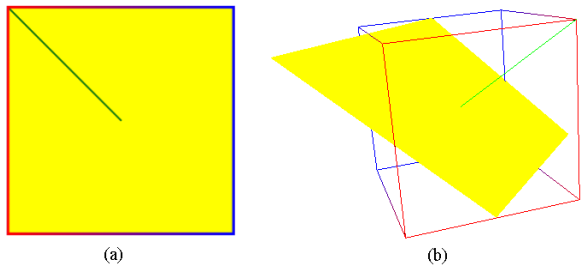

Figure 10.1: (a) The plane viewed from cutting angle. (b) Screenshot of how the cutting angle affects the planes to skew.

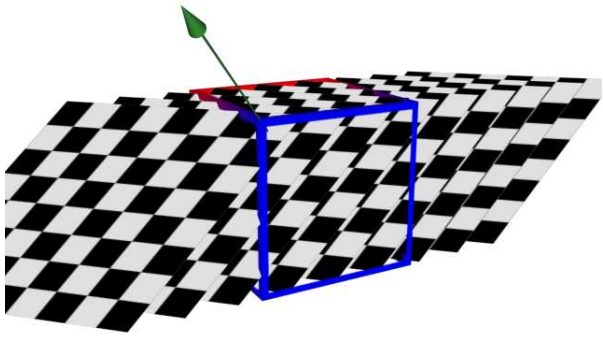

Figure 10.2: An example of how the skewness problem will, in its extreme, look. The green arrow is the vector denoting the viewing angle, all planes are perpendicular to the viewing angle.

regard to the (or those) point(s) inside the bounding cube, until the planes size is exactly to the border of the bounding cube.

Cubic Plane Cutting

The principle of Cubic Plane Cutting can be seen in figure 10.3. The bounding cube can be described with six planes. If the plane generated from the viewing direction is checked for intersection with these bounding planes, it will result in 6 lines. If the view is then limited to a 2D perspective, the generation can be simplified. The 2D perspective is set to be from the axis of the smallest component of the normal vector. That means if the normal vector has the components -20, 140 and 30 (see figure 10.3a) then the 2D perspective will be from the x-axis, ie a yz-coordinate system(figure 10.3b). From the 2D perspective only two lines are interesting, those that cutted the plane that is viewed in the 2D perspective. This means that if the 2D perspective is a yz-coordinate system, the usable lines are those that intersected with the two x-planes (figure 10.3c and d). From the 2D perspective lines can be drawn between the two intersection lines, and if they are drawn perpendicular to the intersection lines (figure 10.3e), the resulting plane will be perpendicular to the viewing angle and quadratic (figure 10.3f).

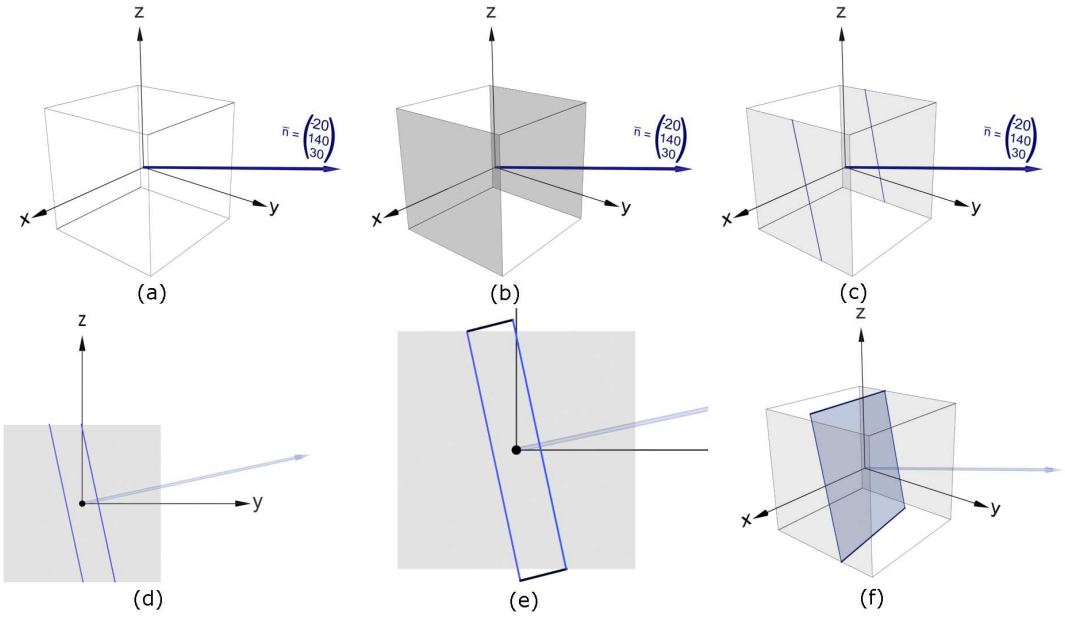

Figure 10.3: The principal of Cubic Plane Cutting. (a) A normal vector (b)with x as the smallest component (c) results in an intersection on the x-planes of the cube, (d) and viewed from an YZ perspective, (e) where the intersection lines are connected perpendicularly, and (f) a plane can be spanned from these lines.

10.1.4 Fitting the planes to bounding cube using polygons

If the planes within the volumes are placed using polygons, then no plane geometry will reach outside the bounding cube. The planes will be strictly confined to the voxelspace. The plane equation, see equation 10.4, is utilised to place the planes. The edge coordinates of the bounding cube can be set in both x, y and z to $\pm\frac{\zeta}{2}$, where ζ is the side length of the bounding cube. This means that all points for the polygon describing a plane can calculated by applying these coordinates to the equation for each plane within the bounding cube. If the resulting coordinate is outside the bounding cube, the coordinate

will not be used in creation of the polygon.

The final step is to sort the coordinates that are within the bounding cube in the order, they should be drawn. This can be done by viewing all the coordinates from a two dimensional perspective, and determine the angle from the centre of the plane to each coordinate. These calculated angles can then be sorted, and the right drawing order is thereby apparent.

10.2 Texturizing plane geometry

When considering the texturizing, one problem is how to slice through the volume, and get the right representation of the slice. Another problem, when considering the generation of textures to the planes, is how to make sure that no important perceptual information is lost in the space between the planes and how to deal with this potential loss. Furthermore, these problems must be evaluated in comparison with the amount of time each method takes.

10.2.1 Coordinate transformation

The problem of finding a proper texture for an arbitrary plane existing within the volume is to remap 3D coordinates into 2D coordinates. This may be solved by coordinate transformation. The principal of coordinate transformation is to have one coordinate system and map it onto another.
In this project the planes are represented by four points in 3D space. An example of a plane may be seen in figure 10.4.

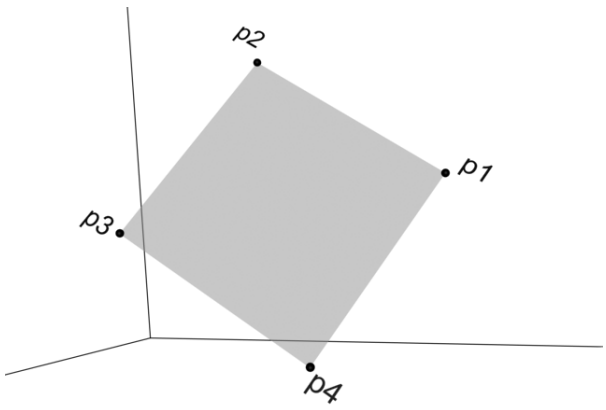

Figure 10.4: A 3D plane represented by four coordinates.

The plane may be described with axes derived from the 3D points of the planes. Furthermore, the two axes may be described by the vectors spanned from point p2 on figure 10.4 where:

$$\vec{v1} = p1 - p2$$

$$\vec{v2} = p3 - p2$$

This is done on the assumption that the plane is symmetrical on the line from p1 to p3. Vector $\vec{v1}$ and $\vec{v2}$ can be seen in figure 10.5 on the next page.

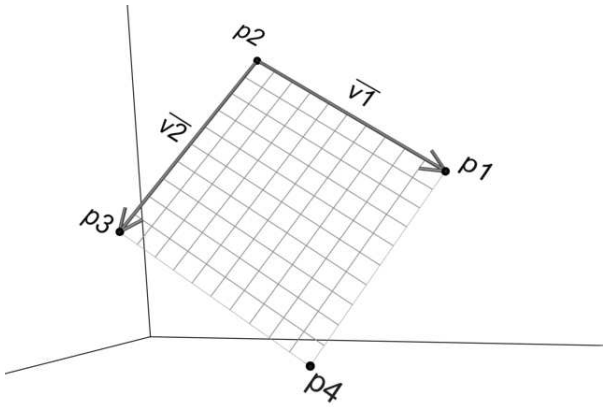

Figure 10.5: Vector $\vec{v1}$ and $\vec{v2}$ derived from points p1,p2 and p3.

The texture has various resolutions, dependent on the requests of the system for that particular plane. The horizontal ($\vec{v1}$) resolution is equivalent to the vertical ($\vec{v2}$), and is denoted s. The half unit length of the vector is calculated in equation 10.8 and equation 10.8. The half unit length is used in order to address the center of the pixel.

$$\vec{vx} = \frac{\vec{v1}}{2 \cdot s} \tag{10.7}$$

$$\vec{vy} = \frac{\vec{v2}}{2 \cdot s} \tag{10.8}$$

The 3D position of the voxels within the volume, which map the x and y position in the texture may be found using $\vec{vx}$ and $\vec{vy}$. If the vertical pixel position on the texture is denoted by x, and the horizontal is denoted by y, then the 3D position p of any pixel on a texture described by the four points may be found with:

$$p(x, y) = (2 \cdot x - 1) \cdot \vec{vx} + (2 \cdot y - 1) \cdot \vec{vy} + p2 \tag{10.9}$$

Example:

In this example the resolution of the texture is 10×10, see figure 10.6 on the following page, therefore s is 10. $\vec{v1}$ and $\vec{v2}$ are scaled down to $\vec{vx}$ and $\vec{vy}$ as seen in the figure. To find the pixel at $(4, 7)$ the vector $p(4, 7)$ is calculated using $(2 \cdot 4 - 1) \cdot \vec{vx} + (2 \cdot 7 - 1) \cdot \vec{vy} + p2$.

With the 3D position for the pixel, the value is the same as the voxel in the volume describing the space.

Another problem is how to make sure that the voxels between the slices are represented on the texture. The representation is needed if there are voxels in the space between the slices, which holds vital perceptual information. In that case the voxels, which are between two slices, must be projected on the texture for representation. The angle of the projection must be equivalent to the viewing angle to create the proper effect. For this the normal vector may be utilised, as it describes the vector from the plane to the users point of view. If the normal vector is scaled to unit length, all voxels between the point on the plane and the point of view may be found by varying a scalar multiplied to the normalised normal vector. If k denotes the scalar and $\vec{n}$ is the normalised normal vector, then the voxel q may be found with:

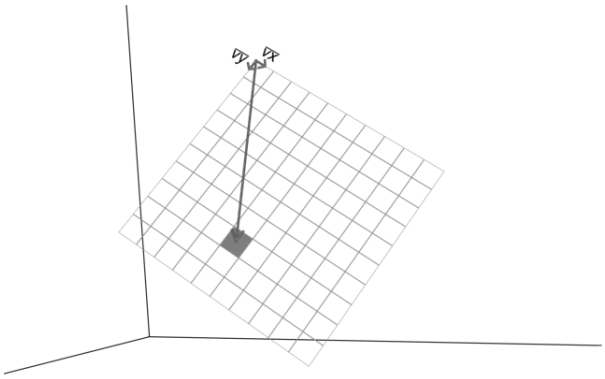

Figure 10.6: Vector $\vec{vx}$, $\vec{vy}$ and the vector to the point x=4 and y=7.

$$q = k \cdot \vec{n} \tag{10.10}$$

This scalar may be used to find all voxels in front and behind of each plane, and may therefore be used to project these onto the texture plane.

10.2.2 OpenGL 3D texture slicing

Source: [Shreiner et al., 2003]
OpenGL extensions allows hardware accelerated texture slicing. This means that all the calculations for slicing the textures will be maintained by the hardware graphics accelerator. This method eases the texture generation to merely telling the hardware points on each plane-geometry and their corresponding points within the voxelspace.

This method is the obvious texture-method, when using polygon slices to represent the volume.

This method is by far the fastest, most effective and easiest. While it is the advantage of the method that the slicing is done within the graphics accelerator, it is at the same time the disadvantage. It is a disadvantage because it is hardware accelerated. When the slicing is hardware accelerated it means that all of the slicing of volumes can only take place on one machine, the machine that draws the graphics for the visualisation. This means that the volume slicing calculations can not be distributed to multiple machines. At the same time in a system that needs to slice multiple volumes, the load on the one machine will be great. At the same time the demands to the memory on the one machine will be very great, as it needs to store all the volumes it is supposed to slice.

10.2.3 Volume rendering using 2D texture stacks

Source: [Rezk-Salama et al., 2000]
The simplest form of volume visualisation utilises only 2D texture mapping which is supported by even the simplest graphics hardware.

To render the volume, the volume is stored as a stack of slices. Three stacks of slices are stored for each volume. Each stack is perpendicular to one the three major axes, and the stack that is most perpendicular to the viewing direction is always chosen as the displayed stack.

Then each slice is rendered back to front.

This method requires only basic OpenGL functionality with no extensions. The principal is illustrated in figure 10.7 with only 9 planes, as it can be seen the hourglass is visible in the volume, even though the viewing direction is far from the planes normal direction. Normally a volume made with this method would be represented with more planes than in the figure, making the gaps between the slices almost invisible.

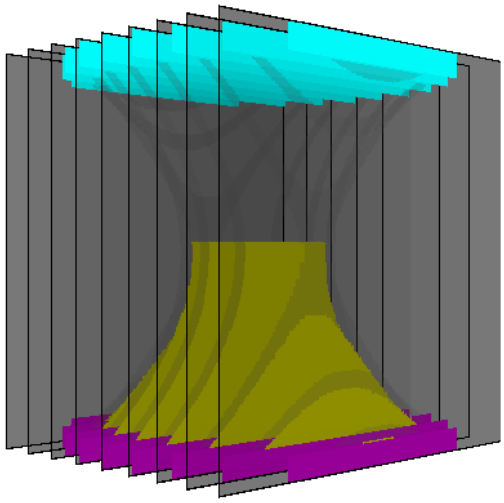

Figure 10.7: The 2D textures Volume rendering method applied with only 9 planes, but still the volume is represented.

There are a number of weaknesses with this simple approach. First, three sets of slices needs to be stored. Secondly, the quality of the image depends on viewing angle. Finally, there is a slight visual pop whenever the viewing direction changes.

The advantage of this method, is that it requires no calculations every time the perspective changes, all necessary calculations are done in advance.

10.2.4 Volume rendering using Shear Warping

Source: [Lacroute and Levoy, 1993]
The general concept of shear warping is a transformation from an object space to a shear object space for a parallel projection. This is done by translating each slice. From the shear object space each voxel slice can be projected into an image simply and efficiently. There are several types of Shear Warp Factorisation methods. The one described in this section is the Affine Factorisation method, see appendix D on page 114 for further details, which includes four transformations: From object coordinates to standard object coordinates, from standard object coordinates to sheared object coordinates, and finally from sheared object coordinates to image coordinates, see figure 10.8 on the following page.

The factorisation of an affine viewing transformation matrix includes a 3D shear, a 2D warp, and a conversion:

$$M_{view} = M_{warp} \cdot M_{shear} P$$

The permutation (conversion) matrix, P is derived from the principal viewing axis and is used to calcu-

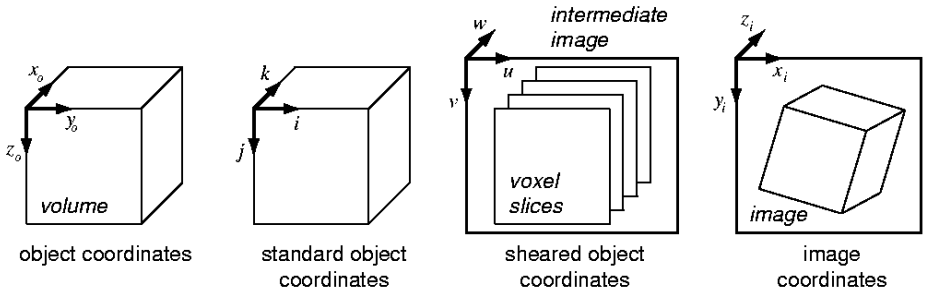

Figure 10.8: The four different coordinate systems used in the derivation of the Affine Factorisation method. Source: [Lacroute and Levoy, 1993]

late the transformation from object coordinates to standard object coordinates. The principal axis is the axis in object space which forms the smallest angle with the viewing direction vector. The transformation from standard object space into image space is made using P and M_{view}. In order to calculate the shear matrix M_{shear} and the warp matrix M_{warp}, the shear coefficients have to be calculated. The shear necessary in the i direction is the negative of the slope, $v_{so,i}/v_{so,k}$ (see figure 10.9), of the projection of the direction vector onto the (i, k) plane. This also holds for the j direction.

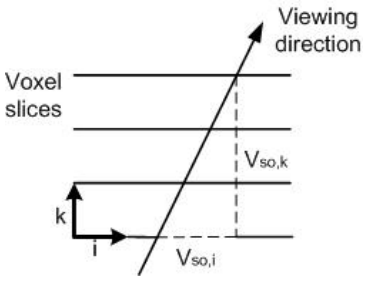

Figure 10.9: The cross-section of the volume and the viewing direction in standard object space. In order to transform the volume to sheared object space the volume must be sheared in the i direction. Source: [Lacroute and Levoy, 1993]

When this transformation from standard object coordinates to sheared object coordinates, the result is a coordinate system where the origin is not located at the top-left corner of the image and thus a translation of the sheared coordinate system is necessary. The four cases of translation is shown in figure 10.10 on the facing page

10.3 Shading

An important aspect of rendering shapes in 3D, is the shading. This applies to volume rendering as well, as the shading is one of the factors that gives objects the effect of appearing in three dimensions. The effect can be seen in figure 10.11 on the next page

The next sections explores ways to apply a shading to a rendered volume. All methods will be used upon the assumption that the light source that shades the volumes is emitting from the user, as if the user in the virtual world is carrying a flashlight.

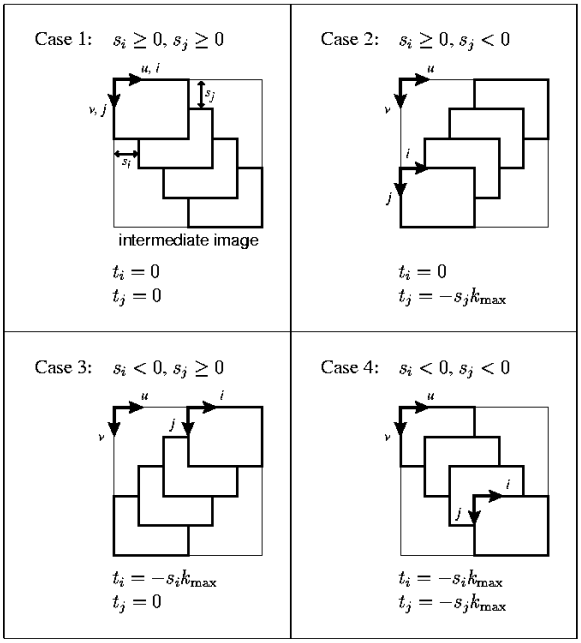

Figure 10.10: The four cases of translation the sheared coordinate system. The translation (t_i, t_j) specifies the displacement from the origin of the standard object coordinate system ($i = 0, j = 0$) to the origin of the intermediate image coordinate system ($u = 0, v = 0$). k_{max} is the maximum voxel slice index in the volume. Source: [Lacroute and Levoy, 1993]

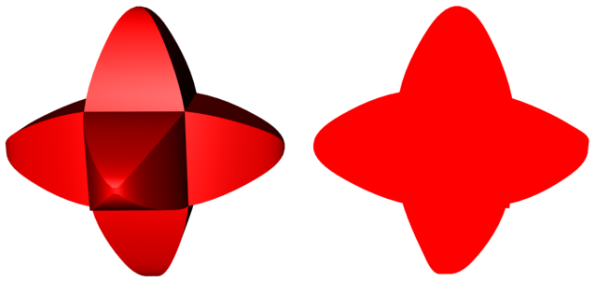

Figure 10.11: The effect shading has on the 3D effect of an object

10.3.1 Plane Depth Grading

This method applies to volume rendering methods using planes to visualise the volume. The approach is very simple, since the light that shades the volume is emitting from the same point as the point of view, the planes nearest to the user must be lighter than the planes farthest away within the volume. With this in mind the concept is to gradually decrease the intensity of the planes as a programme creates them. This method is based on preliminary tests with making a shading for the volume slicing.
Mathematically plane depth grading is utilising:

$$s = \frac{n}{N} \tag{10.11}$$

Where n is the current plane the programme is processing for the current glyph, N is the total number of planes that should be processed for the current glyph, while s is the shading value that should be multiplied to each colour value on each pixel on each plane. The s value is ranging between 1 and 0.
A result with testing this method can be seen in figure 10.12.

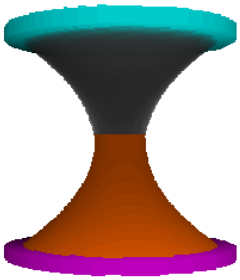

Figure 10.12: An hourglass glyph shaded with Plane Depth Grading.

The weakness of this approach is that it is not a real shading, based upon the curvature of the object. It only helps the user to perceive the depth of the object. Another weakness is that if the number of planes in a volume is low, the intensity difference between the planes can reveal to the viewer that it is merely a row of planes that is visualised, and not a solid object. The advantage of the method is that it is easy, and fast.

10.3.2 Lamberts Reflection Model

Source:[Slater et al., 2001]
One way to create shading upon the surface of a rendered volume, would be by applying Lambert's shading model to the surface calculations of the volume. The model has no base in real-life shading, as it is not synthesised. The model is a simple way to calculate shading of surfaces. Only the most significant affecting lights are taken into calculation of each point, because the peripheral light sources seldom has very large effects on the result. The purpose is not to give a realistic shading of the glyphs, but a shading, that enhances the perception of the shapes of the glyphs. Lamberts shading model can be seen in more details in Appendix E.

First the normal vector of each edge voxel is needed to calculate the shading. This can be found by searching the 26 connected neighbourhood of each edge voxel. The 26 connected neighbourhood can be seen in figure 10.13 on the next page.

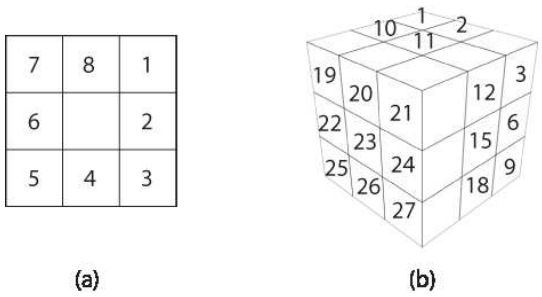

Figure 10.13: The 26 connected neighbourhood.

In the neighbourhood a direction vector is made for each neighbouring voxel that is "free" space, and not solid space. These vectors point in the direction of the free voxels and are all set to unit length. When all vectors has been made, they are all added together, and the resulting vector is the normal vector for the edge voxel, a 2D example of this may be seen in figure 10.14.

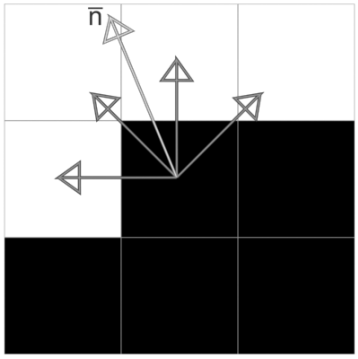

Figure 10.14: 2D example of how vectors to the free voxels, when added together, gives a normal vector for an edge voxel.

The resulting normal vector may be calculated with:

$$\vec{v_T} = \sum_{k=0}^{n-1} \vec{v_k} \tag{10.12}$$

The shading of each voxel, may be divided into 3 categories, diffuse, specular and ambient reflection.
Diffuse reflection is the direct illumination of a surface, which makes the surface visible.
Specular reflection controls how the surface shines on surfaces with direct light.
Ambient controls the illumination of surfaces not affected by light.

Diffuse reflection

When the normal vector and the viewing angle is known, the following calculation can be made to find the diffuse value of the edge voxel:

$$I_d = k_d \cdot I_i \cdot (\vec{n} \bullet \vec{L_i}) \tag{10.13}$$

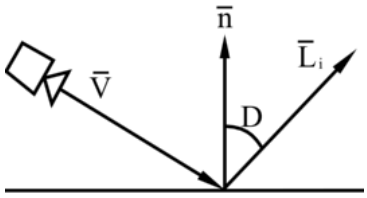

Figure 10.15: The diffuse part of the lambert shading model

Where:
I_d: The resulting intensity value for each edge voxel.
k_d: Diffuse Reflection coefficient. One for each colour channel.
I_i: Intensity of the incoming light.
D: The angle between the normal vector and the incoming light vector.
$\vec{n}$: The normal vector to the surface.
$\vec{L_i}$: The vector for the incoming light.
$\vec{V}$: The viewing vector.
All values are normalised to values between 1 and 0. Equation 10.13 on the page before can be simplified, as there are certain limitations to the shading. Because the light is emitting from the same place as the viewing angle, $\vec{V}$ and $\vec{L_i}$ is the same. The intensity of the incoming light I_i can be set to a constant value of 1. The k_d factor is representing the colours of the voxel, that means k_d is each colour channel represented as a normalised value between 1 and 0;
Hence the diffuse reflection can be reduced to:

$$I_{d,c} = \vec{n} \bullet \vec{V} \tag{10.14}$$

Where I_{d_c} in equation 10.14 is afterwards multiplied with each colour channel, c denoting colour channel.

Specular reflection

To find the specular reflection

$$I_s = k_s \cdot I_i \cdot (n \bullet (\vec{V} + \vec{L_i}))^m \tag{10.15}$$

k_s: Specular Reflection Coefficient.

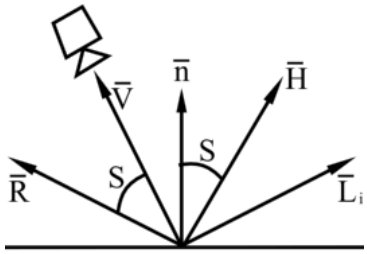

Figure 10.16: The specular part of the lambert shading model

I_i: Intensity of the incoming light.

S: The angle between the perfect reflection angle and the viewing angle.
m: Shininess-factor. Ranging from 1 and to values above, the higher the value, the more concentrated the specular reflection will be.
$\vec{L_i}$: The angle for the incoming light.
$\vec{H}$: The angle between the viewing angle and the incoming light angle. $\vec{H} = \vec{V} + \vec{L_i}$
$\vec{R}$: The perfect light reflection angle.

This calculation applies to all colour channels. As with the diffuse, the specular reflection equation 10.15 may be reduced, by setting the incoming light intensity I_i to 1, while k_s will be set to 1. Since the light emits from the same place as the users point of view, the addition of $\vec{V}$ and $\vec{L_i}$ is irrelevant as they are equal. Hence the specular reflection may be reduced to:

$$I_s = (\vec{n} \bullet \vec{V})^m \tag{10.16}$$

As equation 10.16 bears resemblance to 10.14 this can be utilised to modifying the specular reflection to:

$$I_s = I_{d,c}^m \tag{10.17}$$

Ambient Reflection

Ambient reflection is the simplest of the three, as it does not incorporate angle calculations. Ambient is the light that affects all points.
The ambient reflection I_a may be calculated with:

$$I_a = k_a \cdot I_b \tag{10.18}$$

k_a: Ambient Reflection coefficient
I_b: Ambient Light intensity

These calculations can be omitted by setting I_a to a constant, like 0.3, which means that surfaces on objects that are not affected by the light, will be 30 %.

Total reflection

The three reflection types can be summed to calculate the total reflection:

$$I = I_a + I_d + I_s \tag{10.19}$$

Which can be written as:

$$I = 0.3 + \vec{n} \bullet \vec{V} + (\vec{n} \bullet \vec{V})^m \tag{10.20}$$

Discussion

Based on the analysis of volume rendering it is decided that volume slicing based on simple plane geometry will be utilised in the system. This is done by using the OpenGL primtive quads as it consists of four points and thus matches the corners of the textured image. In order to cut quads out of the planes a 2D cutting method will be used. For navigation the volume rendering technique using 2D texture stacks is utilised, as they do not require calculations of each volume during movement of the view point.

Furthermore, it is decided that in order to find a proper texture for an arbitrary plane the 3D volumes have to be transformed into 2D textures using coordinate transformation, as this process can be distributed, whereas the OpenGL 3D texture slicing can not.

It is decided that the Plane Depth Grading method is to be used. The advantage of the method is that it does not require much processing time. Lamberts Reflection Model needs to project and shade all edge voxels in a volume onto the slices, to have the right effect of the glyph in 3D space. In relation to Plane Depth Grading the Lamberts Reflection Model requires more processing time and is therefore not chosen.

CHAPTER 11

Performance

Using a volume rendering technique for visualising objects demands much CPU processing time due to the fact that each texture has to be re-calculated when the view point is changing. Therefore various methods for reducing the utilisation of memory are analysed.

To exemplify the need for compression, assume that the dimensions of a given voxelspace are $256 \times 256 \times 256$, and thus the resulting size of each voxelspace is:

$$\frac{256 \cdot 256 \cdot 256 \cdot 32}{1024 \cdot 1024 \cdot 8} = 64\text{MB}$$

For instance if there are 1000 objects, each with a unique voxelspace attached, the amount of memory consumed by voxelspaces is 62,5 GB, with 500,000 objects the resulting size is 30,5 TB uncompressed. Therefore, figure 7.2 on page 39 is expanded by a compression before storage and a decompression just before the geometry and texture calculations are performed. The new diagram is shown in figure 11.1.

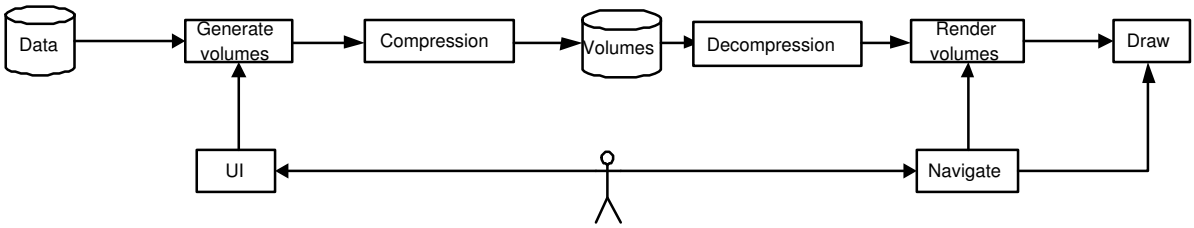

Figure 11.1: The overall flow diagram of the programme expanded with compression and decompression.

In the following sections a case of 1000 objects each with a voxelspace dimension of $256 \times 256 \times 256$ will be utilised to exemplify the memory usage with the various methods.

11.1 General compression methods

A compression of the glyphs is necessary as long as each glyph is projected to take up 64 MB, because the storage of all the glyphs would take up more memory than what is likely available. Several methods may be used for compression such as Joint Picture Expert Group (JPEG) and Portable Network Graphics (PNG). In appendix F on page 124 a more detailed description of the compression methods can be found.

11.1.1 JPEG compression

Source: [Y.Q.Shi and Sun, 2000]
The images is partitioned into 8×8 blocks. On each block a forward Discrete Cosine Transformation

(DCT) is applied, and the resulting coefficients are scanned through a zig zag pattern in order to maintain redundancy. The blocks are then converted into chrominance and luminance effects, which are used in the quantisation process of the image. Based on the quantisation tables for the chrominance and luminance and the DCT coefficients each block is compressed into a bit-stream. The compression ratio for JPEG is 20. The JPEG compression is a lossy compression method. See appendix F on page 124 for further description and testing of the JPEG compression method.

11.1.2 PNG compression

Source: [W3C, 1996]
On glyph-like images the PNG8 compression have a compression ratio of 1:54, which brings the size of each glyph down to 1.18 Mb. This estimate is made on the presumption that each slice of the voxelspace is compressed separately. If the PNG-compression was modified to compressing in the three dimensions instead of two, it is possible that the compression ratio can be improved considerably. The PNG compression is a lossless compression method. See appendix F on page 124 for further description and testing of the PNG compression method.

11.2 Voxelspace reduction

There is an alternative to saving all the images in a voxelspace. Instead of using a standard compression on all the images in the voxelspace, it could save a significantly amount of memory to reduce the image to the minimum amount of information. This may be done by reducing the volume of each image and by reducing the amount of colours representing each glyph. In the case of the three test glyphs described in chapter **??** on page ??, the symmetry of the shapes can be utilised to reduce the voxelspace.

11.2.1 Reduction of volume

If the middle slice of all the voxelspaces is made, this image contains all the information needed to recreate the rest of the voxelspace. The fact that the slice is symmetrical means that the image can be cut to only contain one of the symmetrical parts. This means that the rest of the 255 slices can be recreated from this basic slice. The cross glyph can be recreated with the slice seen in figure 11.2.

Figure 11.2: The basic slice that can recreate the cross glyph voxelspace

The diamond glyph can be recreated by rotating the slice seen in figure 11.3.

Figure 11.3: The basic slice that can recreate the diamond glyph voxelspace

Figure 11.4: The basic slice that can recreate the hourglass glyph voxelspace

The hourglass glyph can be recreated by rotating the slice seen in figure 11.4.

The reduction of data to a basic slice, requires massive calculations in order to be recreated. This can be solved by using a lookup-table for the remapping, where all the voxels points to which pixel in the original image, they are a copy of.

The following example shows how much a volume may be compressed using voxelspace reduction:
Example The resolution of the basic slice of the hourglass is $128x256$, whereas the original size of the voxelspace is $256x256x256$. This is a reduction of size by:

$$\frac{256 \cdot 256 \cdot 256}{128 \cdot 256} = 512 \tag{11.1}$$

The voxelspace, which uncompressed takes up 64 MB, may be described with:

$$\frac{64 \cdot 1024}{512} = 128\text{kB} \tag{11.2}$$

If a 1000 objects are needed for the system, these take up 125 Mb in total, and the images are still compressible. The cross glyphs and diamond glyphs will only take up half the space of the hourglass.

11.2.2 Reduction of colour

The volume of the image is not the only thing that can be reduced. The use of different colours in the voxelspace, is another area where the data size may be reduced by extracting the information. Originally the glyphs contain 32 bit colours, but if the number of different colours is counted, only five different colours are used throughout the basic hourglass image. The cross glyph and the diamond glyph only uses two different colours. If the number of different colours is used as the bit length used to describe them, three bits are enough to describe the five different colours used in the hourglass glyph. The RGB-value of these five colours differ from glyph to glyph, depending on the features used to describe the glyph, therefore a look-up table for the colours is needed for each glyph. If the colour reduction is applied to the hourglass glyph the reduction ratio is: $32/3 = 10.667$. This means that a hourglass glyph may be described using $128/10.667 = 12$kB. With this reduction the total amount of space used to store 1000 hourglass glyphs is approximately 12 MB. In comparison with the original voxelspace, the compression/reduction ratio using colour reduction on the basic image is: $512 \cdot 10.667 = 5461$.

11.2.3 Run length

Source: [Salomon, 2000]
The basic image has large redundant areas, this fact can be used to further reduce the space each image

takes up. If the pixels are described as $start-pixel$, $colour$, and $end-pixel$, instead of the traditional one pixel, one colour structure, huge amounts of data may be saved, as there are many pixels after each other, that have the same colour. This method is called run length encoding and is a powerful way of compressing redundant data.

11.3 Scaling of planes

When several planes are generated, the skewness problem becomes more apparent, as the most forward and backward planes have more geometry outside of the bounding cube than inside. To solve this problem, each plane can be scaled to fit the bounding cube.

11.4 Distance based detail reduction

Another way to utilise the memory is to vary the number of planes for each glyph in regard to the viewing distance. The glyphs that gain the best 3D-effect, are those closest to the point of view, while the glyphs far away gain less by having only a few planes representing them. This way the rendering time is decreased. Additionally, the texture resolution may be degraded, as the distance increases, due to the fact that the representation of textures, far from the view point, is difficult to see.

11.5 Reuse of glyph templates

In order to generate the glyphs, calculations have to be performed for each glyph. If instead templates are created for each glyph with certain intervals, calculations can be minimised. The generation of templates is based on the assumption that the human eye is not able to observe small differences in the dimension values from the data set, which is normalised and lies between 0.0 and 1.0. An example is for the sand element in the hourglass. If the value of the dimension in the data set is 0.1 for one glyph and 0.2 for another glyph, it is difficult to see the sand actually represents two different values. When the observer can not see the difference on the sand element, it is easier to have a glyph representing a larger interval as for instance from 0.0 to 0.2. In that way only five glyphs have to be made and saved in memory as templates, as they may be reused.

Discussion

Based on the analysis of compression it can be concluded that the PNG compression method is the only one of the described compression methods, which is lossless. Based on tests it can be concluded that lossless compression is preferable for this system. On glyph images the PNG8 compression ratio is good in compared to the other compression methods, however a compression ratio of 64 is not sufficient if 500,000 objects are to be represented in the system. The usage of voxelspace reduction is a far more efficient method of compression, due to the fact that each volume is reduced to one 2D image. The method will be used in this system. Furthermore, a reduction of the colours on the 2D image, scaling of the planes, and distance based detail reduction are utilised.

Part IV

Design

Introduction

The system is designed with three main elements. These elements are the Volume Manager handling volumes, the Glyph Manager handling glyphs, and The Volume Slicer handling the translation of volumes to glyphs. A diagram describing the interaction between these elements can be seen in figure 11.5.

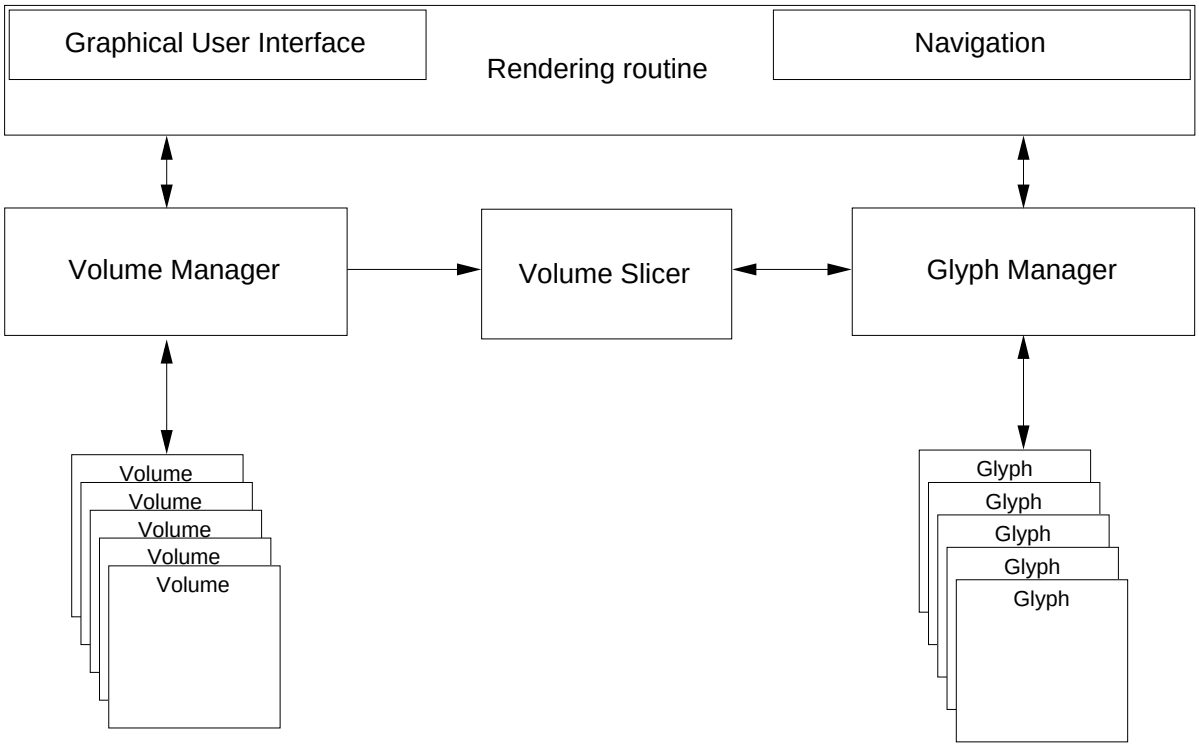

Figure 11.5: A diagram, describing the main elements of the system.

In the following sections a further description of the three main elements will be given.

Volumes

Volumes are the basic data structures where the volumetric data are stored. The system is designed with the ability to store all sorts of image data in these volumes.

When the system is initialised a volume, representing each object to be displayed in the scene will be generated. This means that a large number of objects will demand large amounts of physical data storage. To limit the amount of physical data storage required, each volume type can incorporate a custom compression algorithm, specially designed to minimize that type of volume.

All volumes are managed by a volume manager, that maintains a list of all volumes, with a unique ID.

Glyphs

Glyphs are the abstractions used in the system to describe a renderable element. The glyph is, in the same way as the volume, a data structure containing two main elements namely the textures to be rendered, and the coordinates of the corresponding geometry, on which to place the texture.

All glyphs in a scene is managed by a glyph manager, which maintains at list of glyphs, updating these from the volume they are linked to, when the user requires it.

11.5.1 Overall structure of the system

The overall structure of the system starts with a user choosing a glyph object. The data dimensions are mapped, by the user, to the various features of the object. The first time a user runs the visualisation, the Volume Manager generates all objects as volumes, from the data records. The Glyph Manager utilises the Volume Slicer to generate navigation glyphs and the high density glyphs, which are stored in glyph objects maintained by the Glyph Manager. The Rendering Routine starts drawing the sliced high density glyphs. From here the user can swich to navigation mode, which enables navigation for the user. When the user has navigated to a new point of view, where a high density view ofthe glyphs is desirable, the user commands the system to reslice. The Glyph Manager then utilises the Volume Slicer to generate new high density glyphs from the new point of view. The user has the oppportunity to re-enter the mapping GUI and remap the features or choose a new glyph.

CHAPTER 12

System overview

This chapter describes the functionality of the designed system, in terms of volume, volume slicer, glyph, and graphical user interface.

12.1 Functionality of the system

The object types utilised in the current system are the Hourglass, Diamond, and the Cross glyphs. A further description of these can be found in chapter 9.

Input to the system is a data set clamped between 0-1 from a given database. From the input data, visual objects, of one of the three object types, are generated according to the input from the graphical user interface and stored as volumes. When entering the 3D space in the system, all volumes are converted into glyphs by means of volume slicing.

Volume slicing entails generating a series of planes orthogonal to the viewing direction. This mean that when navigating the 3D world, new calculations will have to be made for each different viewing point. This procedure requires heavy CPU processing, and therefore a series of presliced glyphs are shown when navigating. Each glyph has three stacks of slices along the three global axes. When navigating, one of these is shown, depending on which axis has the largest distance component to the glyph.

A detailed class diagram can be seen in appendix I on page 139.

12.2 Rendering routine

The main rendering routine is the main element, to which all other system part are in some way connected, see figure 11.5. During the initialisation phase, a volume manager and a glyph manager will be created. After this, the GUI is started. Based on the choices made by the user in the GUI, the volume manager and glyph manager will be notified.

12.3 Volume manager

The volume manager is where all volumes are maintained. This includes, generation and removal of volumes. To create volumes, the volume manager is notified by the main rendering routine, when the user has prompted the GUI to start a visualisation. Notification includes information of the number of where input data is located, the number of volumes to create and the volume type is passed to the volume manager. Figure 12.1 is a class diagram of the system part, handling volumes.

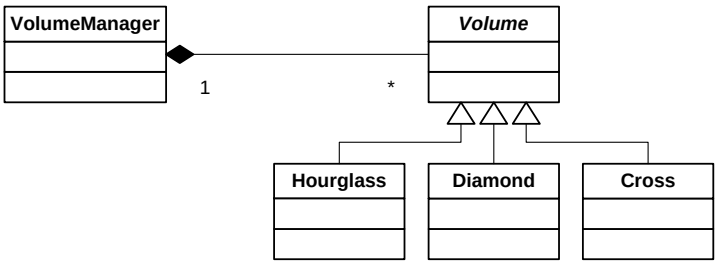

Figure 12.1: A class diagram of the system part handling volumes.

12.3.1 Superclass Volume

The volume is an abstract superclass for the different types of volumes that may be added to the system. Subordinate to the volume class is the different volume-classes, that are used to create the volumes. To function within the system, a volume class must be structured in a specific way, which ensures compatibility. The three test objects, Hourglass, Cross, and Diamond, described in the analysis in chapter 9 on page 47, these are similar in structure, and therefore a general structure will be described.

Setting the data

Creates objects, from normalised data input, as volumes in the system. Based on a list of data and an indication of the desired resolution the glyph is created in voxelspace. The different steps are shown in figure 12.2, where the example is the hourglass volume type. The steps are: the creation of the object, the colour reduction, the compression.

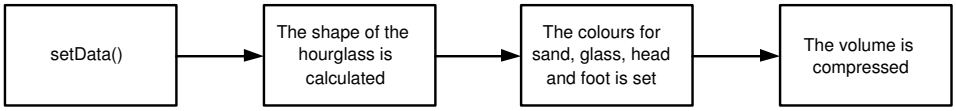

Figure 12.2: The data flow of the method to set the data.

Generation of a look-up table

Some of the volume types require a look-up table with premade calculations, which are utilised for a fast decompression. This allows the data to be decompressed much faster, as the decompression requires less calculations.

Reading the data

The volume is saved internally in the volume object, therefore a general method is needed to read the content of the volume from outside. This method receives a 3D coordinate, and recalculates which voxel the coordinate is similar to. The volume is decompressed using a look-up table, the voxel is found, recoloured and returned to the external function which requested the voxel. An example of this method is shown in figure 12.3 on the following page where the hourglass type is used.

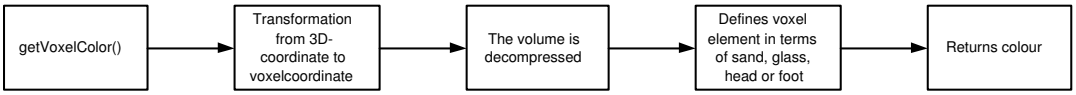

Figure 12.3: The data flow of the method to get the voxel colour.

12.4 Glyph manager

When volumes have been created, the glyph manager will be able to create the elements to render in the 3D scene. To create glyphs, the glyphmanager is initially informed of how many glyphs to create, and which volume each glyph refers to. In this way, the number of glyphs in the scene is not dependent on the number of volumes created. This enables the system to visualise a large number of objects, from a fixed number of template objects.

When initialisation is completed, the glyph manager will be notified when the user changes view point in the system. This requires a reslicing of all glyphs.

To enable a correct back-to-front compositing of all glyphs, the glyph manager will keep track of the rendering order of all glyphs, based on the distance from the view point.

To ensure performance, the system will make use of a distance based degradation scheme for each glyph. This entails reducing the resolution of each glyph based on distance from the view point, and thereby reducing the volume slicing time.

Figure 12.4 is a class diagram of the system part, handling glyphs.

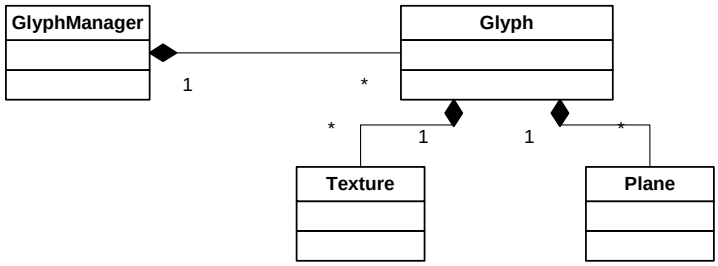

Figure 12.4: A class diagram of the system part handling glyphs.

12.5 Volume Slicing

The purpose of the volume slicing part of the system is to "convert" a volume into a glyph using plane slicing. The volumes are assigned to the slicer by the Glyph manager, which also allocates where the resulting glyphs should be stored. The main argument, the Slicer receives, needed in order to create a sliced glyph, is a normal vector to the planes within the glyph. This vector is the vector from the centre of the glyph to the view point, and thus, because the vector is utilised as normal vector to the planes, these will be perpendicular to the viewing direction, when sliced.
There are two stages to the creation of a glyph from a volume, quad creation and Texture creation.

12.5.1 Quad creation

The quad creation utilises the mathematics for generation of plane geometry, described in chapter 10 on page 51, to calculate the planes from the assigned normal vector. These planes are cut as quads to fit the OpenGL primitive. The cutting of the quads is done from a 2D perspective, chosen from which normal vector component is the largest. Furthermore, the slicer calculates how much of each quad that is contained in the bounding cube and makes a scaling of each slice according to the calculation if parts of the slice are outside the cube. The bounding cube is what limits where the voxels in the volume are situated in 3D space.

When a quad has been created using the 2D perspective cutting, the quad is then evaluated, in regards to how much of it is contained within the bounding cube. Depending on the result of the evaluation the quad will be scaled down in order to make the quad fit best to the bounding cube.

12.5.2 Texture creation

The texture creation stage is the CPU-intensive stage of the two. The method of creating the textures is the coordinate transformation described in section 10.2 on page 56. The method uses the quad-coordinates from the quad creation stage to create the textures. The coordinates are used to create vectors that is used to calculate the different 3D positions where to read a voxel colour for the texture. The volume is accessed to get the voxel colour for the found 3D position, the volume is accessed as though the voxel should be read at the exact 3D position calculated, while the conversion into the volumes voxel coordinates and decompression of the volume is handled internally in the volume. This is done in order to make sure that all volumes, cross hourglass or diamond, can be accessed the same way by the rest of the system, ensuring a flexible system, that can handle all volumes.

When the Texture creation has gone through finding the voxels that should be mapped to all pixels on the textures, the volume slicer writes the calculated quad coordinates and textures into the area allocated by the Glyph Manager.

12.6 Graphical User Interface

According to the external user interface requirements in section 7 on page 41 defined in the specification of requirements in chapter 7 on page 38, the functional requirements, the GUI can be specified in four main functionalities:

Object: User is prompted to choose between various record visualisation objects.

Feature mapping: User is prompted to choose which data dimensions are mapped to object features.

Number of records: User is prompted to choose the number of records to be visualised.

Mouse clicks: The system has to be able to detect mouse clicks on objects within the 3D window.

Before choosing which glyph that is to represent the data set, the data set has to be chosen from a given database. The user then chooses which glyph that is to represent the data. In this system there are three test glyphs available: Hourglass, diamond, and cross. When the glyph has been chosen, the user has to choose which data dimensions are to be mapped to which object features. For each mapping of the

data dimensions, the configuration is shown on the interface. In this way the system becomes more flexible because of the options available for the user on the GUI in contrast to a glyph with predefined features. A predefined glyph can be a problem in some cases, because it might not emphasise what the users intentions are for the given data set. Finally, the user has to choose the number of records to be visualised.

Based on the functional requirements a rough sketch of the GUI can be designed, see figure 12.5, cotaining five panels of buttons reflecting the functional requirements. These are data, features, glyphs, preferences and run visualisation. The glyph, which has been chosen and the data dimensions mapped to the various features, is displayed on the GUI. The rotation, moving, and zooming of the glyph is possible.

The user should at all times be able to switch back and forth between the visualisation of the glyphs and the GUI, to view the perception of the glyph in 3D space. If the user is not pleased, it is possible to return to the previous visualisation again.

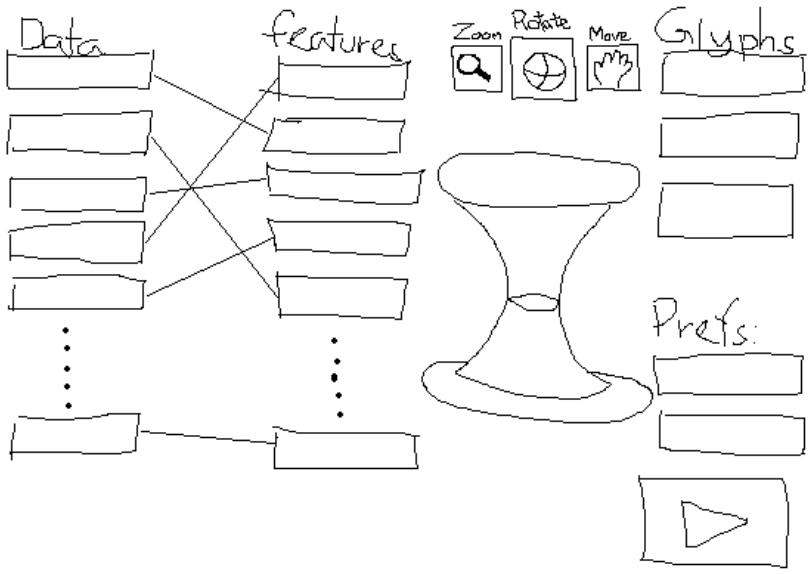

Figure 12.5: A rough sketch of the GUI for the system.

12.6.1 Buttons

The user interface consists only of buttons. These buttons are grouped in panels, and each button in a panel can be assigned an individual function.

The buttons are grouped in Data-buttons, Feature-buttons, Glyph-buttons, a Preferences-button and a Run-button. All buttons are activated using mouse clicks.

Data-buttons

The data-buttons are representing the different dimensions in a data set loaded for the programme. Each dimension is given a button in the panel. The data panel may be seen in figure 12.5 as the leftmost group of buttons. The content and number of buttons in the data panel is changed for each data set loaded. Furthermore, the colour of the buttons change when activated.

Feature-buttons

The Feature-buttons represent each feature that may be mapped to a glyph. The content and number of the buttons changes for every different glyph that is chosen. When the user left-clicks a Feature-button, the data dimension last chosen by the user is mapped to the glyph feature that the button represents. In the GUI this connection is marked by drawing a line from the Data- to the Feature-button. To undo the mapping to a feature, the user right-clicks the Feature-button, and the mapping is removed. In figure 12.5 on the preceding page the Feature-buttons are in the second column from the left, and the data to object-feature-relations are marked with lines from Data-buttons to the Feature-buttons. Furthermore, the colour of the buttons change when activated.

Glyph-buttons

The Glyph-buttons represent the different glyph types that has been added to the system. There is one button per glyph, and the user may, at any time, switch from one glyph to another, and the glyphs in the visualisation will change accordingly. When the user clicks a Glyph-button the Feature-buttons change to the setup in the newly chosen glyph, and the previous mappings are reset. Furthermore, the colour of the buttons change when activated.

Preferences-button

This button allows the user to change attributes in the visualisation. The current version of the system only allows the user to change the number of records to be visualised within the visualisation scene, but other features like the change of various ratios may also be fitted within the preferences category.

Run-button

The Run-button resets all glyphs completely, if they have been initialised earlier, and maps the features, indicated by the data-feature relations set by the user, to the glyphs. The new scene is initialised with the glyph type chosen by the user. Furthermore, the Run-button has a strong colour and is larger than the other buttons, as it is essential for the functionality of the system.

12.6.2 3D space

Another part of the graphical user interface is the 3D space in which the glyphs are visualised. In order to get some aspect in the scene a white grid, encapsulated by the xy, xz, and yz -planes, has to be placed on a black background. Furthermore, the placement of axes, x, y, and z, in the scene, helps the user to keep the orientation.

Part V

Implementation

CHAPTER 13

Implementation

This chapter describes the differences between the designed system and the implemented system. Furthermore, the status of the implemented system is described.

13.1 System Integration

The system has by the end of the project been implemented with all basic functionalities. The system is able to map records from a data set onto the features of the three test glyphs. The system is able to display all feature mapped glyphs inside a 3D scatterplot, and can handle a re-mapping of the features. Furthermore, the system is able switch into a navigation mode, which allows the user to navigate through the system in low detail, but with a high frame rate.

13.2 Functionalities not implemented

The overall functionality of the system has been implemented. However, certain functionalities described in the design have not been used in the system, due to reasons listed below.

13.2.1 Distance based detail reduction and scaling

If the distance based detail reduction utilised when rendering high definition glyphs, the memory usage for the textures is decreased, and thus the frame rate is increased. This is also apparent in scaling of planes.
The distance based detail reduction and scaling has been assigned a lower priority, due to the fact, that it is only used, when rendering high definition glyphs. In high definition mode there are no specific requirements to the animation frame rate, only when navigating. The advantage of these methods is that the slicing time is increased, due to the fact that the resolution on certain textures is decreased. The scaling method has been implemented, but is currently deactivated, as the debugging of the method has been assigned a lower priority.

13.2.2 Projection and shading

The projection of the voxels lying between the slices onto the textures has been implemented, but deactivated. The projection proved to be a CPU heavy operation, which would result in a long slicing time for a scene, when processed on a single computer. This entails that if projection is to be utilised in the system, the calculation process, would need to be distributed.
The lack of projection means that the Lambert's Reflection Model can not be implemented, as this requires all edge voxels to be projected onto the textures for right representation of the shading. As a compensation for the Lambert shading, the Plane Depth Grading has been implemented.

13.2.3 Reuse of glyph templates

The main reason of implementing the method is to reduce the use of memory. Voxelspace reduction has proven to reduce the memory usage for each volume effectively in the system. Therefore the reuse of glyph templates is assigned low priority and has not been implemented.

13.2.4 Graphical User Interface

The five panels, data, features, glyphs, preferences, and run visualisation has been implemented in the GUI. The screenshot of the GUI is shown in figure 13.1. However, the glyph, which should display the data dimensions mapped to various features, is not shown. Furthermore, switching between 3D space and GUI, to view the perception of the glyph, has not been implemented.

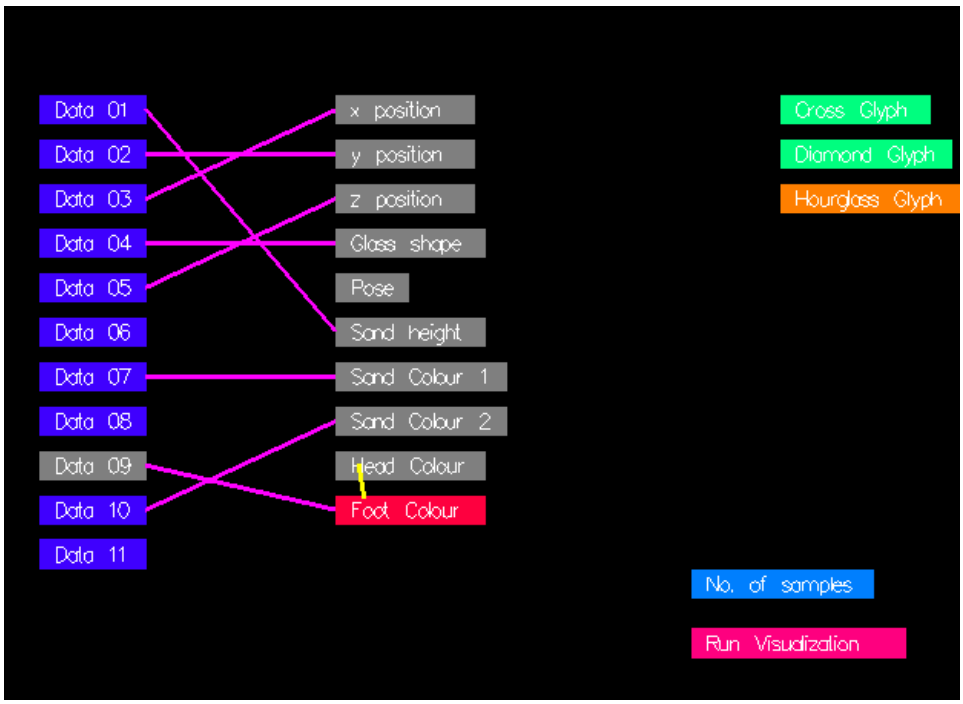

Figure 13.1: The screenshot of the GUI.

13.3 Considerations

This section describes the experiences gained through the implementation of the system.

13.3.1 Detecting mouse clicks using raycasting

The GUI is visualised in 3D, and therefore, normal functionalities such as when a mouse clicks upon a button is not automatically implemented, as the object are now in 3D space. Therefore the mouse clicks on the buttons must be detected, by casting a ray from the users view and find the ray that is cast from the camera to the image plane, a line in 3D space is made. By checking the line for intersections

with the geometry in the scene, it may be determined which object the user clicked upon. Thereby the mouse clicking on 3D buttons may be detected.

13.3.2 Shading during navigation

When a user navigates through the 3D space, the glyphs are shown as 2D stacks for each major axis. A glyph is always shaded from the cutting angle, which means that when the stacks change from one major axis to another, the lighting of the glyph change rapidly. Therefore the navigation slices are made without shading.

13.3.3 Thickness of the glass

In the analysis the hourglass glyph is described to have glass containing the sand. Through the implementation of the system, the glass has proved to be distracting and without any perceptual information. Therefore the hourglass has been modified to a glass thickness of zero.

13.4 System Status

The system has by the end of the project been implemented with all basic functionalities. The system is able to map records from a data set onto the features of the 3 test glyphs. The system is able to display all feature mapped glyphs inside a 3D scatterplot, and can handle a re-mapping of the features. Furthermore, the system is able switch into a navigation mode, which allows the user to navigate through the system in low detail, but with a high frame rate. Screenshots of the system displaying 50 Hourglasses may be seen in figure 13.2 on the next page and 13.3 on the facing page.

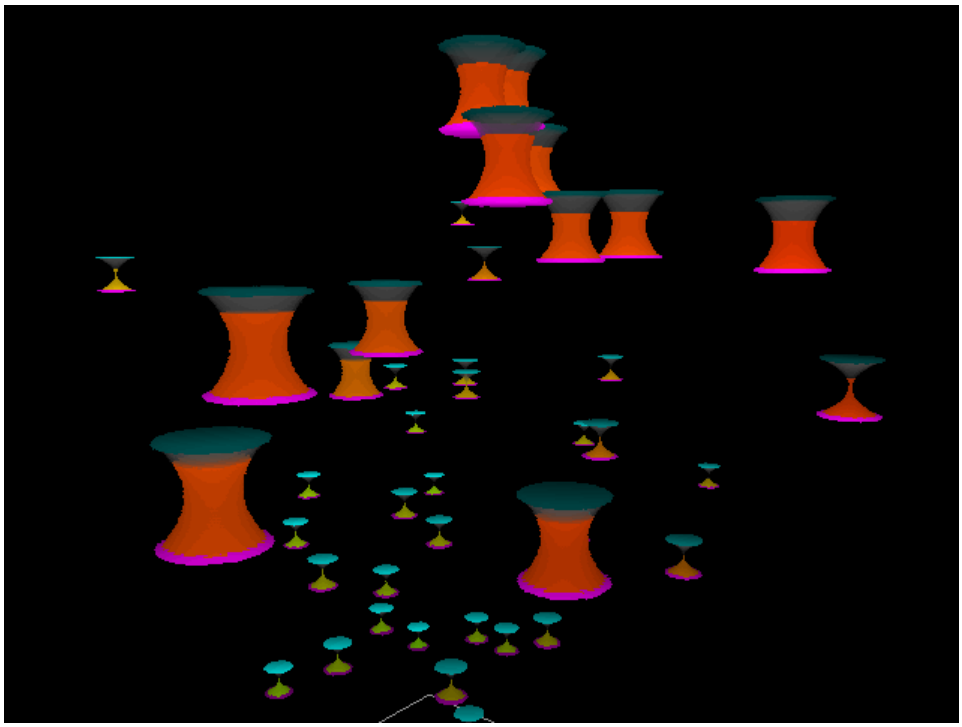

Figure 13.2: A screenshot of the system utilising the hourglass glyph

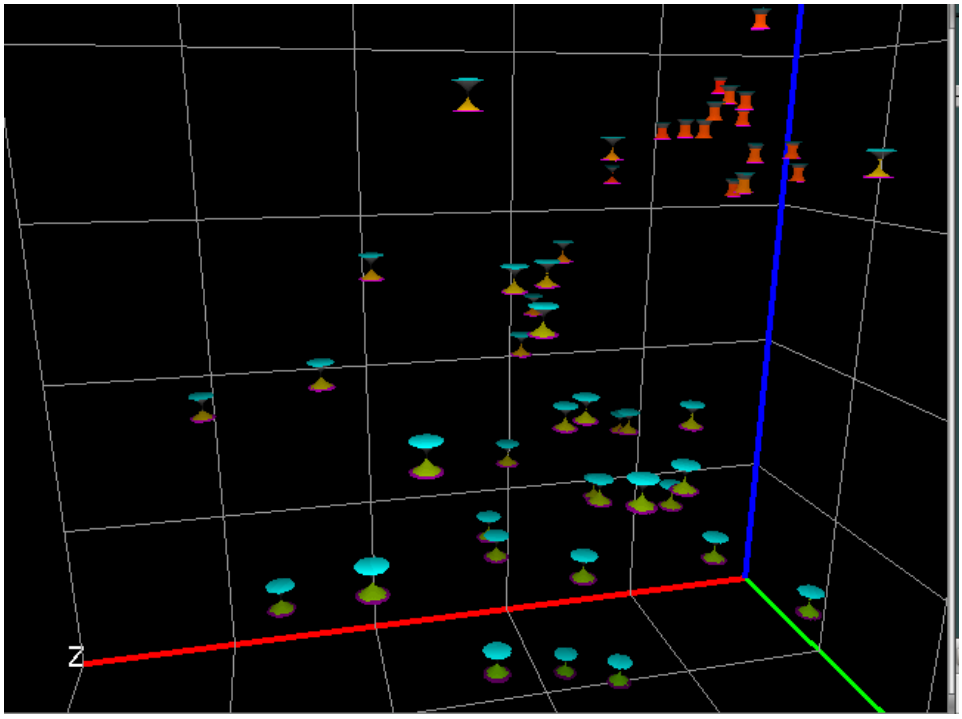

Figure 13.3: A screenshot of the system utilising the hourglass glyph.

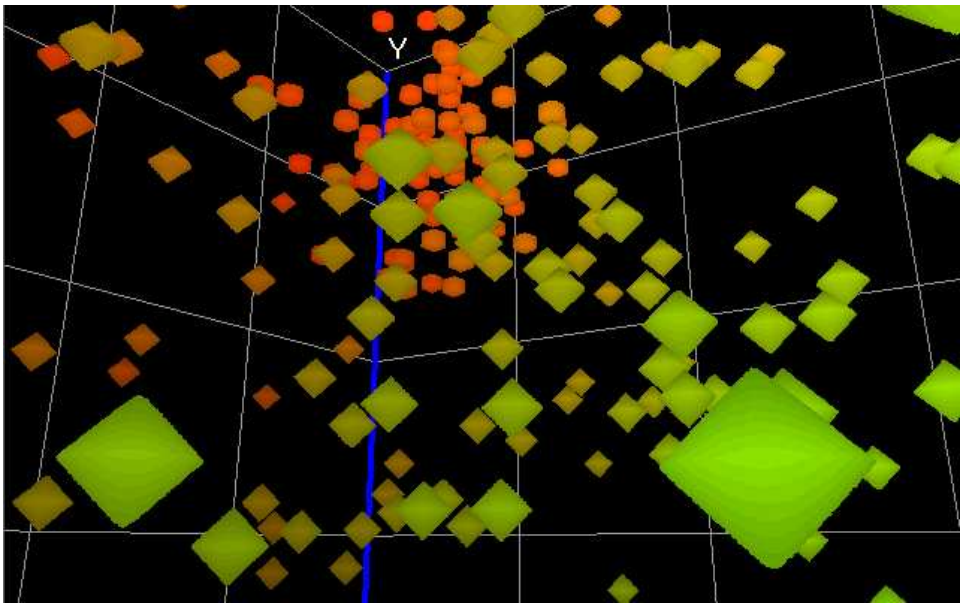

Figure 13.4: A screenshot of the system utilising the diamond glyph.

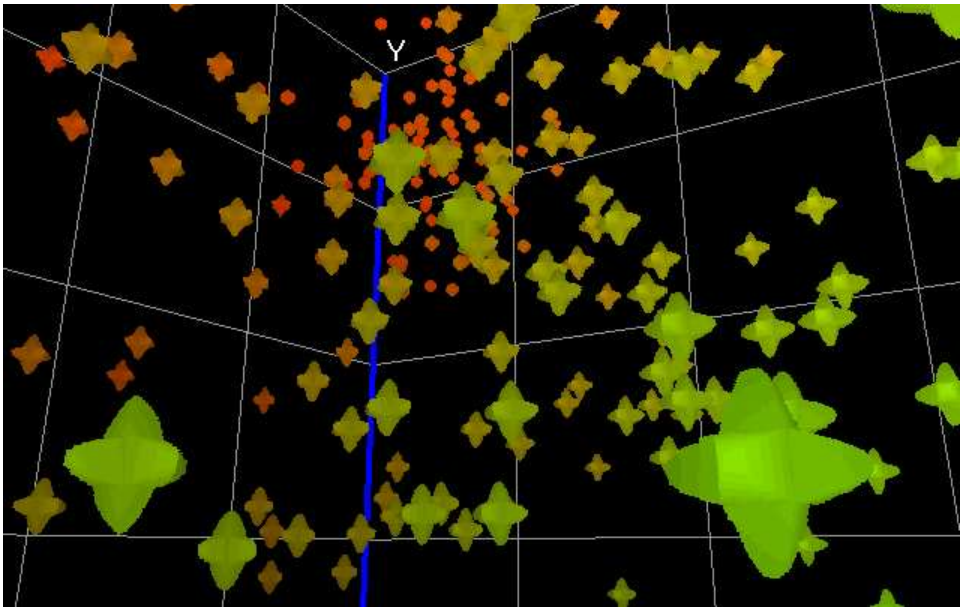

Figure 13.5: A screenshot of the system utilising the cross glyph.

Part VI

Test

CHAPTER 14

Acceptance test

The different procedures for the test cases are described. Furthermore, a discussion of the results is conducted.

14.1 Test of general functionality

This test involves two test cases, described below. A test journal can be seen in appendix H on page 132.

14.1.1 Test description

Mapping is testing a normalised data set, which consists of 50 records with 4 data dimensions for each record. If the data dimensions are mapped correctly to the features the test is successful.

Continous remapping involves testing whether continous remapping of data dimensions to object features is possible. The test is a success if remapping occurs correctly.

14.1.2 Discussion

The system is able to map data set records correctly onto a glyph. Furthermore, the system is able to remap a glyph with new features correctly, and switch between different glyphs.

14.2 Test of performance

This test involves two test cases, described below. A test journal can be seen in appendix H on page 132.

14.2.1 Test description

Frame rate the frame rate is tested in both navigational mode and high definition mode, with a scene containing 50 glyphs and a scene containing 1000 glyphs.

Reslicing the reslicing time for a scene containing 50 glyphs and for a scene containing 50 glyphs is tested.

14.2.2 Discussion

The time to slice the glyphs in a scene is proportional to the number of glyphs in the scene.
The results show, the navigation mode has a significantly higher frame rate than the high density mode,

which means that the implementation of the navigation mode is a necessity for the usability of the system.

14.3 Test of perception

This test involves the test case described below. A test journal can be seen inappendix H on page 132.

14.3.1 Test description

Object perception the object perception in the system is tested using select test cases from the article [Raja et al., 2004].

14.3.2 Discussion

The question of determining the glyph with the largest y component, which demonstrates, that the ability to understand a single data point in the system, is satisfactory. The trends were quickly spotted based on glyph colour, which confirms that colour is one of the most powerful and robust features in 3D visual data mining. The clustering evaluation demonstrated that the shape of the hourglass glyph is not a very conspicuous feature.

CHAPTER 15

User interview

This is a summary of an interview with Erik Granum, conducted on May 27th 2004 , based on the questions in appendix G. The questions are not asked directly, but are only used as a guidelines for the interview. The interview can be divided into four main topics:

- Usability of the system
- Clarity of the 3D space
- Volume rendering
- Graphical user interface

Usability of the system

Displaying many features mapped to one glyph, limits the number of features a person is able to perceive in a scene visualising a large data set. However, large data sets can be explored in other ways. One way of describing a large data set with few multi-featured glyphs could be displaying glyphs representing clusters of data. With a scheme like that, it may also be possible to exploit the fact that volumes can have both an exterior appearance and an interior content, only revealed when entering the object.

Clarity of the 3D space

When mapping data dimensions to various glyph features, the importance of the data dimensions has to be taken into consideration as the priority of the different features is disproportionate. Furthermore, not all features are easily perceived at a far distance, e.g. texture, due to both the distance and high density of the glyphs. This can also result in an overlap of the glyphs. Additionally, high density causes low perception of the features on the glyphs.

Volume rendering

When generating glyphs it should be considered if volume rendering is the right approach in contrast to surface based rendering. For the expert user it seems that the diamond and cross glyphs could be generated by using surface based rendering, as they are not advanced objects. Furthermore, the use of the volume rendering technique enables glyphs with non-symmetrical appearances and hereby increasing the number of features.

Graphical User Interface

The Graphical User Interface (GUI) should be implemented, so the user is able to map data dimensions to different features in real-time while watching the 3D space. This can be done through the existing

3DVDM solution with a separate laptop running a window based GUI. If the GUI is to be implemented in 3D space, it should appear as a sort of tool bar that can freely be moved within the space.

Part VII

Discussion

CHAPTER 16

Conclusion

This chapter is the conclusion of a project made by group 822, 8th semester specialisation in Computer Vision and Graphics 2004, Laboratory of Computer Vision and Media Technology, Aalborg University. The purpose of this project is to display advanced visual objects in a 3D visual data mining system, and allow navigation through this system in order to explore multi-dimensional data sets in arbitrary view directions and from arbitrary view points.

16.1 Purpose of the project

The aim of this project is to display advanced objects in a 3D visual data mining system, and allow a human observer as a visual explorer to navigate the system, in order to explore multi-dimensional data sets in arbitrary view directions and from arbitrary view points. In order to develop advanced objects, the conceptual possibilities and the perceptual possibilities of these objects are studied. Since the group does not possess expert knowledge in the area of perception psychology, this part of the project is solely based on available literature.

To explore the possibilities of advanced graphical objects, a volume rendering technique is employed to render these. The implementation of this is based on the graphic Application Programming Interface (API) OpenGL, as this has been introduced in a semester course.

In order to facilitate user interaction in mapping data properties to object properties, a simple Graphical User Interface (GUI) is employed.

To ensure a reasonable performance from the system, various compression methods are studied in regards to volumetric image data. Furthermore, balancing speed and level of detail, when navigating the 3D visual data mining system is explored.

Verification of results, is performed through an acceptance test. Furthermore, a user interview with an expert VDM user is conducted, to provide a perspective on the usability of the system.

16.2 Methods

In this project, three key elements are to be explored. These are, the perceptual possibilities of using advanced objects to represent multi-dimensional data sets in visual data mining. The conceptual possibilities of how to generate and display advanced objects in 3D space, and performance issues to ensure the users ability to move freely in the 3D space.

16.2.1 Visualisation of multi-dimensional data

Three different test objects have been designed to present multi-dimensional data. The initial assumption for the objects is to be able to map many data dimensions to different object properties in order to

enable a user to detect visual tendencies. Three main properties of the three test objects are, position in 3D space, colours, and basic shape deformation.

16.2.2 Displaying advanced objects in 3D space

Volume rendering is used in order to provide a flexible way of displaying both advanced exteriors and interiors of objects.

For this project the volume rendering technique used is volume slicing. This entails sampling of a number of planes, perpendicular to the viewing direction from volumetric image data. The sample planes are mapped to a series of corresponding, semi-transparent, planes in 3D space, providing a three dimensional impression of the volumetric image data.

To ensure a high level of detail for all rendered objects, they are rendered as individual volumes.

16.2.3 Performance of the system

To ensure that the user of the system is able to move freely in the 3D space, a navigation scheme is employed. This involves an alternation, based on the viewing angle, of three pre-made texture stacks, perpendicular to each of the three major axes. This minimises the need for the CPU-heavy task of slicing all volumes when navigating, thereby increasing the animation frame rate.

To enable the system of displaying large amounts of objects in 3D space, memory usage is considered. Volumetric image data is compressed in order to minimise memory usage. Instead of using a standard compression on all the volumetric image data, a significant amount of memory can be saved by reducing it to the minimum amount of information. This is done individually for each object type, e.g. by taking advantage of symmetry.

16.3 Results

The system is evaluated through both an acceptance test and an interview with an expert 3D visual data mining system user.

16.3.1 Acceptance test

The acceptance test is divided into three main test cases: General functionality, performance, and object perception.

The general functionality of the system is consistent with the requirements defined in the specification of requirements in chapter 7 on page 38. This means that the system is able to map a multi-dimensional data set to properties of a set of corresponding graphical objects. Furthermore, continuous re-mapping of data dimensions to object properties.

The performance of the system is tested for two important factors, the animation frame rate and the re-slicing time. The animation frame rate is measured during navigation mode. With 50 objects ,in the scene, the frame rate is 145 fps. With 1000 objects in the scene, the frame rate is reduced to 7.5 fps. From the test it follows that the system is able to handle a maximum of 500 objects, while maintaining the requested frame rate of 15 fps.

The perception of the hourglass glyph is tested to determine the level of understanding it is able to provide of an underlying multi-dimensional data set. The hourglass glyph is able to provide a relatively fast understanding visual tendencies in the test data.

16.3.2 User interview

The user interview with Erik Granum, professor of Information systems, Aalborg University. The following summarises the essentials of the interview.

Erik Granum points out that the displaying of many features mapped to one glyph, limits the number of objects that can be visualised in a scene while maintaining clear perception of all features. Furthermore, he points out that an application for volume rendering may be letting objects represent clusters of data, e.g. by utilising both the exterior and interior of the objects.

16.3.3 Volume rendering vs. surface based rendering

When evaluating volume rendering and surface based rendering, it is concluded that the two methods have different areas of application within 3D visual data mining. Surface based rendering is well suited for visualising a large number of objects with few features, while volume rendering is well suited for visualising a limited number of objects with many features.

16.4 Prospect

The area of volume rendering has many aspects, of which this report mainly researches the area of conceptual possibilities. The area of perceptual possibilities is primarily based on existing research. This section will describe some aspects that can be further researched.

An approach for improving the perceptual possibilities is to look into the area of structuring the input data set, in order to represent several data records with a single graphical object. A method of doing this is clustering, enabling presentation of a large data set with few objects. With these objects the possibilities of displaying both advanced exteriors and advanced interiors can be used.

Other approaches are to explore the use of different axes, e.g. nominal axes, and possibilities of using non-symmetrical objects to increase the number of available features.

The system developed in this project is a limited self contained prototype system. To increase the possibilities of both user interaction, e.g. a standard graphical user interface and stereo vision, and loading of data, an implementation as a library into the VR++ system may be considered. Furthermore, using VR++ will enable distribution of processing tasks in the system, thereby increasing the performance.

Bibliography

[vrl, 2002] (2002). Fotos fra cave.

[Biering-Sørensen et al., 1988] Biering-Sørensen, S., Hansen, F. O., Klim, S., and Madsen, P. T. (1988). *Håndbog i Struktureret Program-Udvikling*. Ingeniøren|bøger, 1 edition.

[Card et al., 1999] Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). Readings in information visualization: Using vision to think.

[Cox et al., 1997] Cox, K. C., Eick, S. G., and Wills, G. J. (1997). Visual data mining: Recognizing telephone calling fraud.

[Duda et al., 2000] Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. John Wiley and Sons Inc, 1 edition.

[Ebert et al., 1996a] Ebert, D., Shaw, C., Zwa, A., and Miller, E. (1996a). Minimally-immersive interactive volumetric information visualization.

[Ebert et al., 1999] Ebert, D. S., Rohrer, R. M., Shaw, C. D., Panda, P., Kukla, J. M., and Roberts, D. A. (1999). Procedural shape generation for multi-dimensional data visualization. In Gröller, E., Löffelmann, H., and Ribarsky, W., editors, *Data Visualization '99*, pages 3–12. Springer-Verlag Wien.

[Ebert et al., 1996b] Ebert, D. S., Shaw, C. D., Zwa, A., and Starr, C. (1996b). Two-handed interactive stereoscopic visualization. In Yagel, R. and Nielson, G. M., editors, *IEEE Visualization '96*, pages 205–210.

[Gillies, 2004] Gillies, D. F. (2004). Polygon rendering and opengl. `http://www.doc.ic.ac.uk/~dfg/graphics/GraphicsLecture08.pdf`.

[Granum and Musaeus,] Granum, E. and Musaeus, P. Constructing virtual environments for visual explorers.

[Haykin, 1994] Haykin, S. (1994). *Neural Networks - a comphrensive foundation*. Macmillan USA.

[Jackson, 1990] Jackson, P. (1990). *Introduction to Expert Systems*. Hardback.

[Kreyszig, 1999] Kreyszig, E. (1999). *Advanced Engineering Mathematics*. John Wiley and Sons Inc.

[Lacroute and Levoy, 1993] Lacroute, P. and Levoy, M. (1993). Fast volume rendering using a shear-warp factorization of the viewing transformation.

[Lichtenbelt, 1995] Lichtenbelt, B. B. (1995). Fourier volume rendering. Technical report, Hewlett Packard Laboratories.

[Madsen, 1997] Madsen, P. (1997). *Teknisk Matematik Bind 3*. Erhvervsskolernes Forlag, 1 edition.

[Nagel et al.,] Nagel, H., Granum, E., and Musaeus, P. Methods for visual mining of data in virtual reality.

[Nagel and Granum, 2002] Nagel, H. R. and Granum, E. (2002). Vr++ and its application for interactive and dynamic visualization of data in virtual reality. In *Proceedings of the Eleventh Danish Conference on Pattern Recognition and Image Analysis*, Copenhagen, Denmark.

[Nagel et al., 2003] Nagel, H. R., Vittrup, M., Granum, E., and Bovbjerg, S. (2003). Exploring nonlinear data relationships in vr using the 3d visual data mining system. In *Proceedings of the International Workshop on Visual Data Mining, in conjunction with The Third IEEE International Conference on Data Mining*, Melbourne, Florida, USA.

[Post et al., 1995] Post, F., van Walsum, T., Post, F., and Silver, D. (1995). Iconic techniques for feature visualization. pages 288–295.

[Raja et al., 2004] Raja, D., Bowman, D. A., Lucas, J., and North, C. (2004). Exploring the benefits of immersion in abstract information visualization.

[Rezk-Salama et al., 2000] Rezk-Salama, C., Engel, K., Bauer, M., Greiner, G., and Ertl, T. (2000). Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization.

[Roelofs, 2002] Roelofs, G. (2002). Mng specification. `http://www.libpng.org/pub/mng/mngdocs.html`.

[Salomon, 2000] Salomon, D. (2000). *Data Compression - The Complete Reference*. Hardback, 2 edition.

[Shaw et al., 1998] Shaw, C. D., Ebert, D. S., Kukla, J. M., Zwa, A., Soboroff, I., and Roberts, D. A. (1998). Data visualization using automatic, perceptually-motivated shapes.

[Sherman and Craig, 2003] Sherman, W. R. and Craig, A. B. (2003). *Understanding Virtual Reality - Interface, Application, and Design*. Morgan Kaufmann, 1 edition.

[Shreiner et al., 2003] Shreiner, D., Woo, M., Neider, J., and Davis, T. (2003). *OpenGL Programming Guide*. Addison Wesley, 4 edition.

[Silva, 1995] Silva, C. (1995). Parallel processing for volume visualization. `http://www.cs.sunysb.edu/~csilva/papers/rpe/node11.html`.

[Slater et al., 2001] Slater, Steed, and Chrysanthou (2001). *Computer Graphics And Virtual Environments: From Realism To Real-Time*. ADDISON-WESLEY, 1 edition.

[Thalmann, 2004] Thalmann, D. (2004). Fundamentals of virtual reality. `http://vrlab.epfl.ch/~thalmann/VRcourse/Fundamentals.pdf`.

[Volumizer, 2004] Volumizer (2004). *SGI® OpenGL Volumizer™ 2.6 Programmers Guide*. Silicon Graphics Inc.

[W3C, 1996] W3C (1996). Png (portable network graphics) specification. `http://www.w3.org/TR/PNG-Compression.html`.

[Y.Q.Shi and Sun, 2000] Y.Q.Shi and Sun, H. (2000). *Image And Video Compression For Multimedia Engineering*.

Part VIII

Appendix

APPENDIX A

Open Graphics Library - OpenGL

In this appendix, a short introduction to OpenGL is described in relation to both surface based and non-surface based purposes. Furthermore, geometric primitives in OpenGL are described.

All surface based primitives are eventually described in terms of their vertices - coordinates that define the points themselves, the endpoints of line segments, or the corners of polygons.

OpenGL structure

Source:[Gillies, 2004]
Open Graphics Library (OpenGL) is the computer industry's standard application program interface (API) for defining 2D and 3D graphic images. OpenGL specifies a set of commands or immediately executed functions. Each command directs a drawing action or causes special effects. A list of these commands can be created for repetitive effects. OpenGL is independent of the windowing characteristics of each operating system (Microsoft Windows and X Windows), but provides special routines for each operating system that enable OpenGL to work in that system's windowing environment. OpenGL comes with a large number of built-in capabilities requestable through the API. These include hidden surface removal, alpha blending (transparency), anti aliasing, texture mapping, pixel operations, viewing and modelling transformations, and atmospheric effects (fog, smoke, and haze).

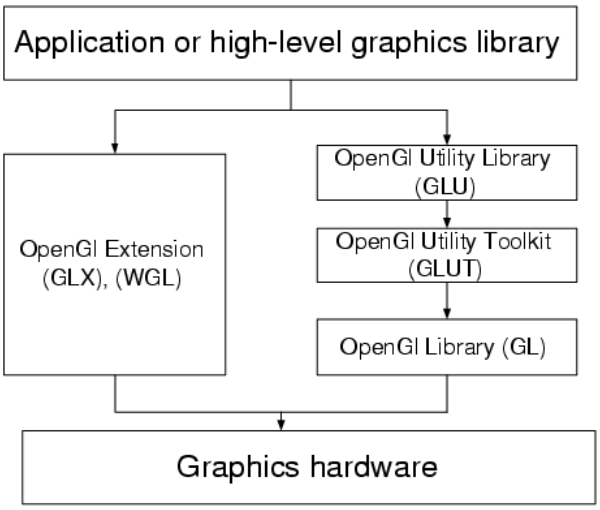

Figure A.1: Overview of the OpenGL API. Source: [Gillies, 2004]

The functionality of the OpenGL library is extended by a number of related libraries: OpenGL Utility library (GLU) contains several convenience routines which are implemented in terms of OpenGL commands. These include routines for setting up viewing transformation. The OpenGL Utility Toolkit (GLUT), a window system independent toolkit for writing OpenGL programs, implements a simple

windowing application programming interface for OpenGL. In addition it provides a portable application programming interface, which facilitates writing a single OpenGL program that works on windows systems such as Microsoft Windows and X Window system environments. Each window system has its own library, which is OpenGL extension (GLX) for X Window system and (WGL) for Microsoft Windows. OpenGL is not a general purpose graphics system. Instead, OpenGL is a polygon-based rendering system which supports a small number of geometric primitives for creating models: Points, lines and polygons.

Points

A point is represented by a set of floating-point numbers called a vertex. All internal calculations are done as if vertices are three-dimensional. Vertices specified by the user as two-dimensional (that is, with only x and y coordinates) are assigned a z-coordinate equal to zero by OpenGL. In figure A.2 five vertices are shown. The GL_POINTS in OpenGL are individual points.

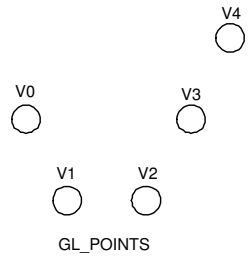

Figure A.2: Individual points called vertices. Source: [Gillies, 2004]

Lines/edges

In OpenGL, line means line segment. There are easy ways to specify a connected series of line segments, or even a closed, connected series of segments. In all cases the line segment type causes successive pairs of vertices to be interpreted as the endpoints of individual segments. Because the interpretation is done on a pairwise basis, successive segments usually are disconnected. In figure A.3 three different ways to show segments may be seen. The GL_LINES are pairs of vertices interpreted as individual line segments, the GL_STRIP is a series of connected line segments and GL_LOOP is the same as GL_STRIP, with a segment added between the last and the first vertices.

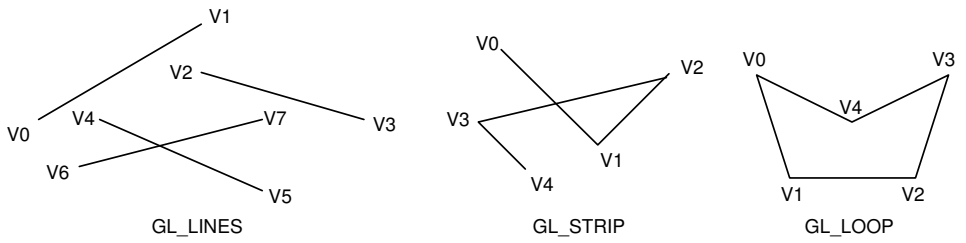

Figure A.3: GL_LINES, pairs of vertices interpreted as individual line segments. GL_STRIP, series of connected line segments and GL_LOOP, the same as GL_STRIP with a segment added between the last and the first vertices. Source: [Gillies, 2004]

Polygons

Polygons are the areas enclosed by single closed loops of line segments, where the line segments are specified by the vertices at their endpoints. A polygon can be displayed in various ways. The interior can be filled with a solid colour, or pattern, and the line segments can optionally be displayed. In general, polygons can be complicated, so OpenGL has restrictions on what constitutes a primitive polygon. Their are three ways to describe polygons, simple, convex and flat:

- Simple polygons are where lines do not intersect.
- Convex polygons are where two points lie inside the polygon and the line joining the points also lie inside the polygon.
- Flat polygons are where polygons lie in a plane.

Since OpenGL vertices are always three-dimensional, the points forming the boundary of a particular polygon does not necessarily lie on the same plane in space. In many cases it lies on a plane, if all the z coordinates are zero, for example, or if the polygon is a triangle. If a polygon's vertices does not lie in the same plane, then after various rotations in space, changes in the viewpoint, and projection onto the display screen, the points might no longer form a simple convex polygon. For example, imagine a four-point quadrilateral where the points are slightly out of plane. You can get a nonsimple polygon that resembles a bow tie as shown in figur A.4, which is not guaranteed to render correctly. This situation is not all that unusual if you approximate surfaces by quadrilaterals made of points lying on the true surface. You can always avoid the problem by using triangles, since any three points always lie on a plane.

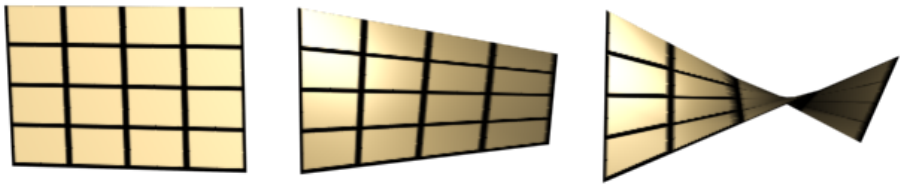

Figure A.4: A non-planar polygon which is transformed into a non-simple polygon. Source: [Gillies, 2004]

For many applications, non-simple polygons are needed, non-convex polygons, or polygons with holes. Since all such polygons can be formed from unions of simple convex polygons, some routines to describe more complex objects are provided in the GLU, for instance a quad, defined in OpenGl as GL_QUAD. These routines take complex descriptions and break them down into groups of the simpler OpenGL polygons that can then be rendered. The reason for OpenGL's restrictions on valid polygon types is that it is simpler to provide fast polygon-rendering hardware for that restricted class of polygons. In figure A.5 on the facing page some examples of valid and invalid polygons GL_POLYGON are presented.

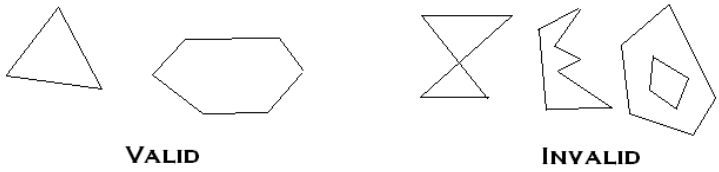

Figure A.5: Valid and invalid polygons. Source: [Gillies, 2004]

APPENDIX B

SGI OpenGL Volumizer

Source:[Volumizer, 2004]
In this appendix a technique called volume slicing in the SGI Volumizer is explained. This technique is used for presenting 3D objects and have separate advantages/disadvantages, which are also mentioned. A short introduction to other features in SGI OpenGL Volumizer are also briefly referred to.

OpenGL Volumizer is a graphics API designed for interactive, high quality, scalable visualisation of large volumetric data sets. The API uses OpenGL for volume rendering and hence allows standard graphics applications to treat volumetric and surface data in a similar fashion.

A 3D object is any object that exists in 3D space. The triangles and other surface elements used to represent such objects are 2D primitives. 2D primitives suffice in many cases because most objects around us are adequately represented by their surface. Objects with interesting interiors are abundant in everyday life. Clouds, smoke, and anatomy are all examples of volumetrically interesting objects. Despite their abundance and importance, volumetric objects are either not handled at all or their treatment is substantially different from that of surface-based models.

OpenGL Volumizer extends the concepts of surface-based models to include volumetric shapes. The result is the OpenGL Volumizer is capable of handling both types of models equally well.

Volume slicing

To render a shape OpenGL Volumizer uses a technique called volume slicing. Volumizer slices a volume along a set of parallel surfaces. This slicing process is called polygonization. The result is a set of polygons, called faces. Textures are mapped with each of these polygons.

The following are advantages of using volume slicing/3D texture mapping:

- Using 3D texture mapping for volume rendering is very fast since all the interpolations for each fragment are done by the OpenGL hardware. Also, the texture data is resident in texture memory, which reduces the data access time considerably.

- Since the volume rendering process generates a polygonal approximation of the data, the technique allows mixing of volumes with other polygonal data.

The orientation of the surfaces is configurable. The simplest case is when the surfaces are orthogonal to the line of sight of the viewer. The surfaces, however, can be aligned arbitrarily. The slices in these figures are planar, but it is possible to slice the the volume in any shape.

Minimal tessellation

In many cases volumes are cubes, also called hexahedral. The minimum tessellation of a hexahedron is five tetrahedra as shown in figure B.1. A tetrahedron, shown in figure B.2, is the simplest and most efficient primitive that can be used to represent volumetric geometry. Any shape can be tessellated into tetrahedra. There are advantages and disadvantages to using minimal tessellations.

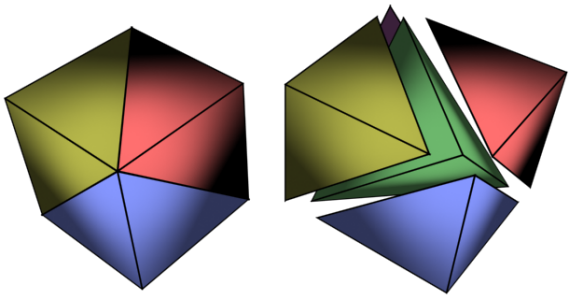

Figure B.1: A cube which consist of five tetrahedra.

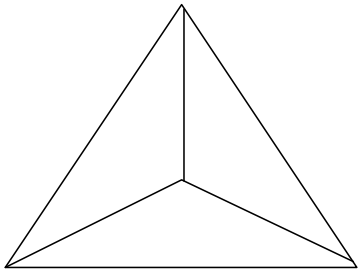

Figure B.2: This is a structure of a tetrahedon.

- The advantage of minimal tessellation for one hexadron is the rendering process, which is increased by a factor of five.
- The disadvantage is the interpolation between the tetrahedra. While different volumes, for instance a cube, often is compact, and therefore consist of several colours, the minimal tessellation is not uniform. This results as mentioned in interpolation between the edges of the shared tetrahedra. A way to minimise the problem is to increase the number of more uniformly distributed tetrahedra.

Bricks

A problem is some image data bases contain more voxel data than can be stored in a machine's texture mapping hardware. This problem can be overcome by taking voxel data and break them up into subsections, called bricks.

A brick is a subset of voxel data that can fit into a machine's texture mapping hardware. Bricks are regular hexahedra (boxes) with non-zero width, height, and depth. Displaying a volume requires paging

the appropriate brick data into texture memory. Anticipating which bricks to page can speed up the performance of the application.

One or more adjacent bricks constitute a brick set. Volumetric appearance is defined as a collection of one or more brick sets. Figure B.3 shows a brick set containing eight bricks, shown as eight cubes within the large cube.

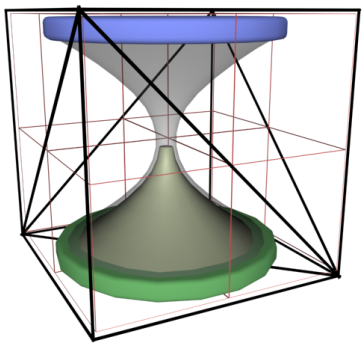

Figure B.3: A brick set containing eight bricks, shown as eight cubes within the large cube.

Typically, a shape can be described by a single brick set. In certain situations, however, more than one brick set is required. For example, on machines that do not support three-dimensional texture mapping, three separate copies of the data set may have to be maintained, one for each major axis to minimise sampling artifacts.

Brick size plays an important role in the overall efficiency. Short data transfers may require frequent interrupts in the data flow and can consequently affect performances. On the other hand, long data transfers optimise the overall bandwidth but are not interruptible. The choice of brick size depends upon the hardware architecture and therefore applications should select values taking system parameters into consideration.

APPENDIX C

Mathematical representations of the glyphs

The purpose of this appendix is to analyse how the three types of glyphs described in chapter 9 on page 47 can be represented mathematically. With a mathematical representation of the glyphs it is possible to generate the glyphs for use in programmes.

For each glyph type a number of features may be represented e.g. through shape, colour, transparency. The mathematical representation of the glyphs mainly deals with the shape of a glyph and how it changes in regards to a given data value.

With a arbitrary voxelspace dimension given, the equations may be utilised to generate the voxelspaces for each glyph needed for the volumetric renderer.

Cross glyph

For the cross glyph one feature is represented by the shape of the glyph. In figure C.1 the glyph is represented geometrically.

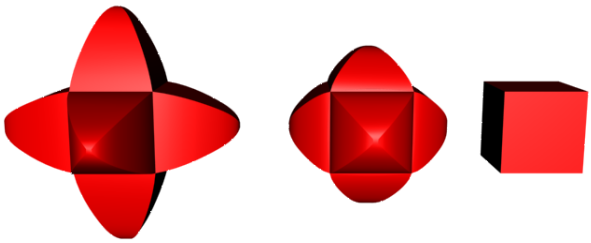

Figure C.1: Geometric representations of the cross glyph

The shape is a cube when the shape controlling feature value is 0. When the value is 1, the bulges of the glyphs cube grow, and the glyph takes on a cross shape. In order to describe the growing bulges of the cross glyph, the equation for a parabola is applicable. The space, in which to describe the parabola, will be limited to the xy-planes 1st quadrant. As the shape is symmetrical, it is only necessary to create the shape within this confined area, as it can be mirrored afterwards. In figure C.2 on the following page a sketch of the cross-glyph in the 1st quadrant is shown.

All equations will be denoted dynamically, with the intended resolution for the volume specified as d.

The range of the quadrant is set to 0 to $\frac{d}{2}$ as a mirroring will give the final glyph a range from $-\frac{d}{2}$ to $\frac{d}{2}$ in both the x and y direction. In figure C.2 on the next page two half parabolas are shown. The parabolas are equal except they are confined to different axes as function axis. The maximum of the parabolas is in the height $\frac{d}{2}$ units, and their roots are in the width $\frac{d}{6}$ units from the symmetry axis. The two parabolas are raised $\frac{d}{6}$ units in both directions, which makes a $\frac{d}{6}$ by $\frac{d}{6}$ square in the lower left corner

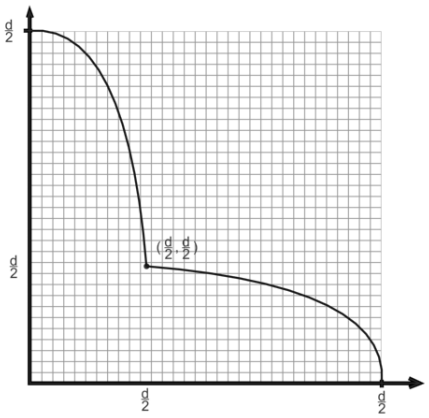

Figure C.2: The cross glyph as two parabolas in the 1st quadrant of the xy-plane.

if the parabolas where to be multiplied by 0. This means, if limiting the equation to the parabola facing upwards, the parabola has a maximum in $(0, \frac{d}{3})$ and roots in $(-\frac{d}{6}, 0)$ and $(\frac{d}{6}, 0)$. The range of the up facing parabola is $-\frac{d}{6} \leq x \leq \frac{d}{6}$. In order to find the equation for the parabola, the two roots are used to find the basic parabola equation:

$$y = (x - \frac{d}{6}) \cdot (x + \frac{d}{6})$$

$$y = x^2 - (\frac{d}{6})^2$$

Since the maximum of the parabola represents the highest value, the parabola equation must be negative:

$$y = -x^2 - (\frac{d}{6})^2 \tag{C.1}$$

The equation has a maximum value of $(\frac{d}{6})^2$, hence the parabola equation must be calibrated so the value of the maximum is $\frac{d}{3}$. To compensate the equation is multiplied by c:

$$c = \frac{\frac{d}{3}}{(\frac{d}{6})^2} = \frac{12}{d} \tag{C.2}$$

Inserting equation C.2 into equation C.1 yeilds

$$y = -\frac{12}{d} \cdot x^2 + \frac{d}{3} \tag{C.3}$$

To obtain the correct shape of the glyph the equation is multiplied by the data value. When multiplied by 0 the shape becomes flat, and when multiplied by one the shape is unchanged thus the shape will range from a cube to a full cross glyph. In order to make the 0-value turn the shape into a cube, $\frac{d}{6}$ is added to expression C.3. This will create a cube ranging from $-\frac{d}{6} - \frac{d}{6}$ in the centre of the space when the parabolas are mirrored on the axes.

The final equation for the up facing parabola is:

$$y = \lambda \cdot (-\frac{12}{d} \cdot x^2 + \frac{d}{3}) + \frac{d}{6} \tag{C.4}$$

Where λ controls the shape feature.

The equation for the parabola facing right in figure C.2 on the facing page is equal to equation C.4, only x and y have switched places:

$$x = \lambda \cdot (-\frac{12}{d} \cdot y^2 + \frac{d}{3}) + \frac{d}{6} \tag{C.5}$$

with the range $-\frac{d}{6} \leq y \leq \frac{d}{6}$.

In figure C.3, equation C.4 and C.5 are utilised to create the mirrored 2D-representation of the equation.

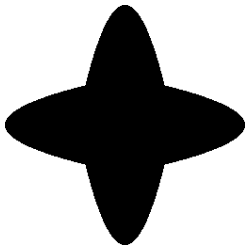

Figure C.3: Mirrored 2D-representation of the equations.

The 2D-representation shown in figure C.3 may be used to create the 3D glyph. The glyph is symmetrical in all directions. To construct the 3D glyph, a voxelspace volume with the arbitrary dimensions $d \times d \times d$ is used. In each direction, x, y and z, the 2D image is applied, this means a xy, xz and yz image. The 3D glyph is constructed by using the binary command AND. For each voxel in the volume a glyph voxel is created if the corresponding xy, xz and yz voxel is a glyph pixel. If this is not the case the voxel is set as a background voxel. The application of this method is illustrated in figure C.4.

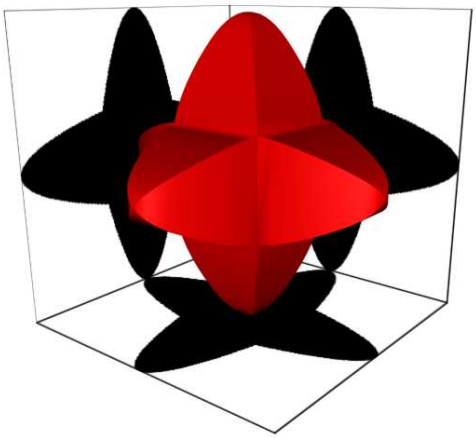

Figure C.4: Creating a 3D glyph from a 2D glyph

Diamond glyph

The shape of the diamond glyph is a morphing between a cylinder and a diamond-like shape, dependent on the data values. Normally, a diamond is edged with straight surfaces, but in order to describe the shape easily by a mathematical representation, the horizontal part of the shape is limited into a circular shape. The circular diamond shape is illustrated in figure C.5.

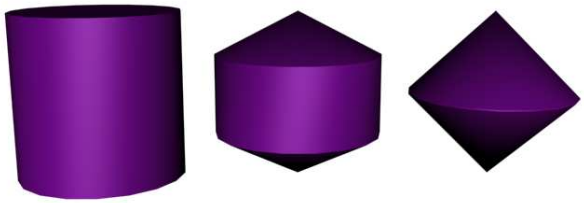

Figure C.5: The circular diamond shape

The shape of the diamond glyph may be described using two equations: The equation of a circle and the equation of a straight line. The circle equation controls the horizontal shape, while the straight line equation controls the skewness of the shape. The shape controlling feature-value is denoted as λ. The straight line must, when λ is 0, intersect the points $(\frac{d}{2}, 0$ and $(0, \frac{d}{2})$ and it must be horizontal if λ is 1. The straight line is illustrated in figure C.6.

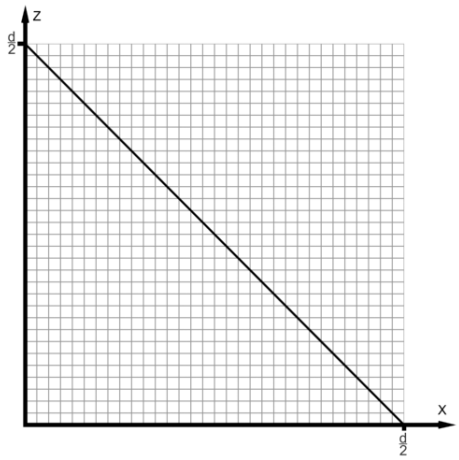

Figure C.6: The straight line controlling the skewness of the diamond glyph

The line may be described through the following equation:

$$y = -(1 - \lambda) \cdot x + \frac{d}{2} \tag{C.6}$$

The way to enable the line to control the skewness of the glyph, is by letting the x value of the line denote the radius of the circles the glyph is constructed from. This is illustrated in figure C.7 on the facing page.

This requires a rewriting of equation C.6, with x replaced by the radius, r:

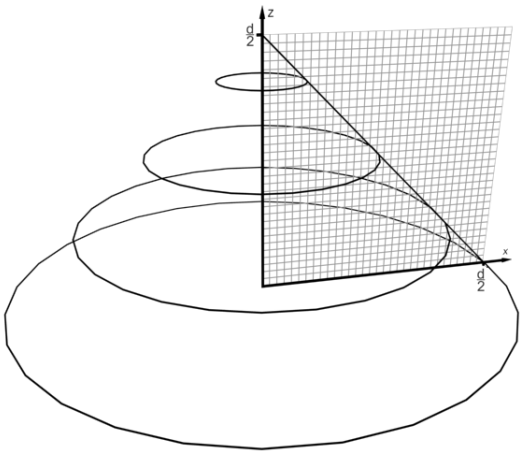

Figure C.7: The straight line controlling the radius of the diamond glyphs circles

$$r = -\frac{y - \frac{d}{2}}{1 - \lambda}$$

The equation of a circle is:

$$x^2 + y^2 = r^2 \tag{C.7}$$

Combining equation C.6 and C.7 equation C.8 is obtained:

$$x^2 + z^2 = (-\frac{y - \frac{d}{2}}{1 - \lambda})^2 \tag{C.8}$$

Hourglass glyph

The hourglass glyph, which is illustrated in figure C.8, is mathematically more complex than the previous two glyphs.

Figure C.8: A geometric visualisation of the hourglass glyph.

While the previous two where solid inside, the hourglass glyph employs morphing of a shell of glass filled with sand in an amount that corresponds to another data value. In addition the glyph has a head and a foot that also have to be modelled. An illustration of these elements may be seen in figure C.9 on the following page.

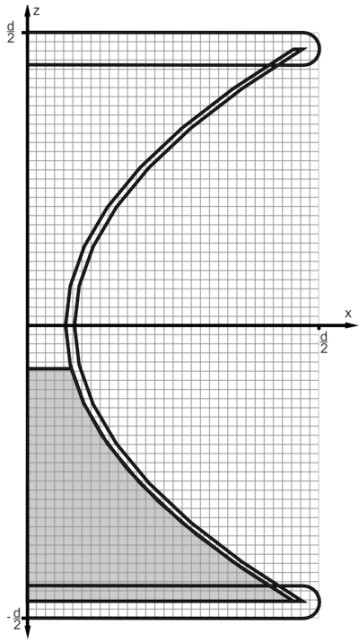

Figure C.9: The different elements of the hourglass glyph, head and foot, sand, and glass.

The three elements will be described in the following.

Glass

The glass of the hourglass glyph is similar to the cross glyph, as it also utilises the parabola equation to describe the shape in 2D space. When the 3D shape is implemented it is done similar to the diamond glyph, by using the x value of the parabola equation as radius for the circle equation.

The thickness of the glass is set to $\frac{d}{32}$ units. The maximum radius of the cylinder is set to $\frac{15 \cdot d}{32}$ to make sure that the glass does not get outside the head and foot of the hourglass. The parabola equation for the glass has the roots $-\frac{15 \cdot d}{32}$ and $\frac{15 \cdot d}{32}$. The maximum denotes how close the glass is to the centre. There should be left $\frac{d}{32}$ units of space for the centre, when the parabola is at it's maximum. The thickness of the glass is $\frac{d}{32}$ units, and therefore the maximum of the parabola must be set to $\frac{d}{2} - \frac{2 \cdot d}{32} = \frac{7 \cdot d}{16}$. This means the equation for the parabola is:

$$x = \lambda \cdot c \cdot (y - (\frac{15 \cdot d}{32})) \cdot (y + (\frac{15 \cdot d}{32})) + k$$

$$x = \lambda \cdot c \cdot (y^2 - d^2 \cdot (\frac{15}{32})^2) + k$$

c is the calibration value that normalises the equation to have the right height in the maximum. k is the constant that places the parabola right vertically when the x-axis is perceived as the vertical axis. The value of c is:

$$c = \frac{\frac{7 \cdot d}{16}}{\frac{15^2}{32^2} \cdot d^2}$$

$$c = \frac{7}{d \cdot \frac{15^2}{64}}$$

The value of k is $d \cdot 15/32$. The values are applied, and the equation yields:

$$x = \lambda \cdot (\frac{7}{d \cdot \frac{15^2}{64}} \cdot y^2 - \frac{7 \cdot d}{16}) + \frac{15 \cdot d}{32} \tag{C.9}$$

As in the diamond glyph, the x-value denotes the radius in the circle equation:

$$x^2 + z^2 = (\lambda \cdot (\frac{7}{d \cdot \frac{15^2}{64}} \cdot y^2 - \frac{7 \cdot d}{16}) + \frac{15 \cdot d}{32})^2 \tag{C.10}$$

The range for equation C.10 is $-\frac{15 \cdot d}{32} \leq y \leq \frac{15 \cdot d}{32}$. The equation may be used to create a solid paraboloid shaped cylinder, but the shape must be tubular because it has to be filled inside. Therefore a similar paraboloid shaped cylinder with a radius of $\frac{d}{32}$ units less must be subtracted from the first cylinder. This will create a paraboloid shaped tube with a radius $\frac{15 \cdot d}{32}$ and a glass thickness of $\frac{d}{32}$ units.

Sand

The sand utilises the same equation as the inner paraboloid shaped cylinder from the glass element to create the shape formed by equation:

$$x^2 + z^2 = (\lambda \cdot (\frac{7}{d \cdot \frac{15^2}{64}} \cdot y^2 - \frac{7 \cdot d}{16}) + \frac{15 \cdot d}{32} - \frac{d}{32})^2 \tag{C.11}$$

The difference is the range of the sand which is dependent on a given data value. When the sand controlling data value has a value of 0 the sand is only filled from $-\frac{15 \cdot d}{32}$ to $-\frac{45}{128} \cdot d$ and when the value is 1, the glass is filled from$-\frac{15 \cdot d}{32}$ to $\frac{45}{128} \cdot d$. The glass is never emptied or filled all the way to the top due to the fact that the colour of the sand may be used to show another feature of the data set, therefore the sand must be visible at all times. If the sand feature is denoted as γ the range of the sand shape is: $-\frac{15 \cdot d}{32} \leq y \leq \frac{d}{2} + \frac{45}{128} \cdot d - \frac{45}{64} \cdot \gamma \cdot d$.

Head and foot

The head and the foot of the hourglass may be described as two circles. The mathematical representations of the circles are the same except for the z-values having opposite signs.

The vertical shape of the head may be described as:

$$(x - \frac{15 \cdot d}{32})^2 + (y - \frac{15 \cdot d}{32})^2 = (\frac{d}{32})^2$$

If x is isolated:

$$x = \sqrt{(\frac{d}{32})^2 - (y - \frac{15 \cdot d}{32})^2} + \frac{15 \cdot d}{32} \tag{C.12}$$

The y value is ranging from $\frac{7 \cdot d}{16} \leq y \leq \frac{d}{2}$. Equation C.12 is used as radius in equation C.13:

$$x^2 + z^2 = (\sqrt{(\frac{d}{32})^2 - (y - \frac{15 \cdot d}{32})^2} + \frac{15 \cdot d}{32})^2 \tag{C.13}$$

APPENDIX D

The affine factorisation method

Source: [Lacroute and Levoy, 1993]
In this appendix the affine factorisation method, based on affine transformations including uniform and non-uniform scales, translations and shears used in parallel projections, is derived. In the derivation, four different coordinate systems, shown in figure D.1 will be used:

Object coordinates are the standard coordinates of the volume data. The unit distance of each axis equals the length of one voxel. The axes are labelled x_0, y_o and z_o.

Standard object coordinates are formed by converting the axes of the object coordinates so that the principal viewing axis becomes the third coordinate axis. The axes are labelled i, j and k, which is the principal viewing axis.

Sheared object coordinates are formed by shearing the standard object coordinates with the shear matrix from the shear warp factorisation. The shear object coordinate system is also the coordinates system for the intermediate image. The axes are labelled u, v and w.

Image coordinates are the coordinates for the final image. The warp matrix of the shear warp factorisation transforms sheared object coordinates into image coordinates, and the original viewing transformation matrix transforms object coordinates into image coordinates. The axes are labelled x_i, y_i and z_i.

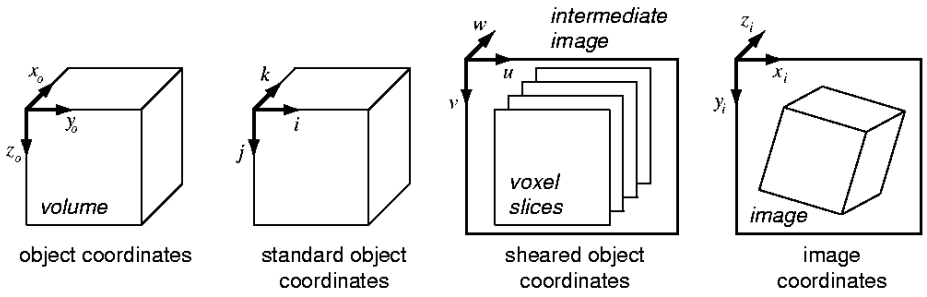

Figure D.1: The four different coordinate systems used in the derivation of the affine factorisation method. Source: [Lacroute and Levoy, 1993]

In the derivation of the affine factorisation method, the viewing transformation matrix is a four-by-four matrix M_{view}, which transforms homogeneous points from object space to image space:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ w_i \end{bmatrix} = M_{view} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix}$$

The goal of the derivation is to factor M_{view} as follows:

$$M_{view} = M_{warp2D} \cdot M_{shear3D} \tag{D.1}$$

where M_{warp2} denotes an affine transformation matrix and $M_{shear3D}$ denotes a shear matrix which transforms object space to sheared object space. This transformation requires shearing the coordinates system until the viewing direction becomes perpendicular to the slices of the volume.

The principal viewing axis

[Lacroute and Levoy, 1993] defines the principal viewing axis as the object space axis that forms the smallest angle with the viewing direction vector. In image space the viewing direction vector is $\vec{v}_i = (0, 0, 1)$. Let $\vec{v}_0$ be the viewing direction vector transformed to object space. It follows that:

$$\vec{v}_i = M_{view,3x3} \cdot \vec{v}_0 \tag{D.2}$$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} v_{o,x} \\ v_{o,y} \\ v_{o,z} \end{bmatrix} \tag{D.3}$$

where m_{ij} denotes the the elements of the upper-left 3x3 sub-matrix of M_{view}. It is possible only to use the upper-left 3x3 sub-matrix as $\vec{v}_i$ and $\vec{v}_0$ are vectors. The linear system of equations, D.2, can be solved analytically for $\vec{v}_{o,x}$, $\vec{v}_{o,y}$, and $\vec{v}_{o,z}$ respectively, using Cramer's rule [Kreyszig, 1999].

$$v_{o,x} = \frac{\begin{vmatrix} 0 & m_{12} & m_{13} \\ 0 & m_{22} & m_{23} \\ 1 & m_{32} & m_{33} \end{vmatrix}}{\mid M_{view,3x3} \mid} = \frac{m_{12}m_{23} - m_{22}m_{13}}{\mid M_{view,3x3} \mid}$$

$$v_{o,y} = \frac{\begin{vmatrix} m_{11} & 0 & m_{13} \\ m_{21} & 0 & m_{23} \\ m_{31} & 1 & m_{33} \end{vmatrix}}{\mid M_{view,3x3} \mid} = \frac{m_{21}m_{13} - m_{11}m_{23}}{\mid M_{view,3x3} \mid}$$

$$v_{o,z} = \frac{\begin{vmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ m_{31} & m_{32} & 1 \end{vmatrix}}{\mid M_{view,3x3} \mid} = \frac{m_{11}m_{22} - m_{21}m_{12}}{\mid M_{view,3x3} \mid}$$

Since the denominator is the same for all three expressions the solution can be written as:

$$\vec{v}_o = \begin{bmatrix} m_{12}m_{23} - m_{22}m_{13} \\ m_{21}m_{13} - m_{11}m_{23} \\ m_{11}m_{22} - m_{21}m_{12} \end{bmatrix}$$

The principal viewing axis is the largest component of $\vec{v}_o$:

$$c = max(\mid v_{o,x} \mid, \mid v_{o,y} \mid, \mid v_{o,z} \mid)$$

Transformation to standard object coordinates

Shear warping is a resampling and composition of the voxel slices which are perpendicular to the principal viewing axis. In order to eliminate special cases for each of the three axes, the volume is transformed into standard object coordinates. Is the principal viewing axis the x_o axis, then the converted matrix is:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Is the principal viewing axis the y_o axis, then the converted matrix is:

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Is the principal viewing axis the z_o axis, then the converted matrix is:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation from object coordinates to standard object coordinates is given by:

$$\begin{bmatrix} i \\ j \\ k \\ 0 \end{bmatrix} = P \cdot \begin{bmatrix} x_o \\ y_o \\ z_o \\ 0 \end{bmatrix}$$

The matrix M'_{view} is a converted viewing transformation matrix, which transforms points from object space into image space:

$$M'_{view} = M_{view} P^{-1}$$

The shear and warp factors

The converted viewing transformation matrix M'_{view} has to be factorised into a shear in the i and j directions. After the shear transformation, the viewing direction must be perpendicular to the (i, j) plane. In standard object space the viewing direction vector is:

$$\vec{v}_{so} = P \cdot \vec{v}_o = \begin{bmatrix} m'_{12}m'_{23} - m'_{22}m'_{13} \\ m'_{21}m'_{13} - m'_{11}m'_{23} \\ m'_{11}m'_{22} - m'_{21}m'_{12} \end{bmatrix}$$

where m'_{ij} are elements of M'_{view}. The projection of the direction vector onto the (i,k) plane has a slope of $v_{so,i}/v_{so,k}$ as shown in figure D.2

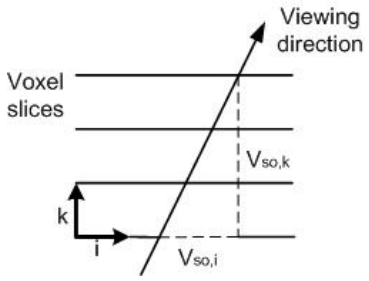

Figure D.2: The cross-section of the volume and the viewing direction in standard object space. In order to transform the volume to sheared object space the volume must be sheared in the i direction. Source: [Lacroute and Levoy, 1993]

In order for the viewing direction to be perpendicular to the k-planes in the i direction, the shear has to be the negative of the slope. This also holds for the shear in the j direction. Thus the shear coefficients are:

$$s_i = -\frac{v_{so,i}}{v_{so,k}} = \frac{m'_{22}m'_{13} - m'_{12}m'_{23}}{m'_{11}m'_{22} - m'_{21}m'_{12}} \tag{D.4}$$

$$s_j = -\frac{v_{so,j}}{v_{so,k}} = \frac{m'_{11}m'_{23} - m'_{21}m'_{13}}{m'_{11}m'_{22} - m'_{21}m'_{12}} \tag{D.5}$$

The shear warp factorisation of the converted viewing transformation can now be written as:

$$M'_{view} = M'_{view} \begin{bmatrix} 1 & 0 & -s_i & 0 \\ 0 & 1 & -s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.6}$$

$$= \begin{bmatrix} m'_{11} & m'_{12} & (m'_{13} - s_i m'_{11} - s_j m'_{12}) & m'_{14} \\ m'_{21} & m'_{22} & (m'_{23} - s_i m'_{21} - s_j m'_{22}) & m'_{24} \\ m'_{31} & m'_{32} & (m'_{33} - s_i m'_{31} - s_j m'_{32}) & m'_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.7}$$

where the left factor is an affine warp matrix and the right factor is a shear matrix.

Projection to the intermediate image

Applying the shear transformation, just derived, to the standard object coordinate system in order to obtain the coordinate system for the intermediate image, results in a coordinate system where the origin

is not located in the top-left corner of the intermediate image. A translation of the sheared coordinate system is therefore necessary. The four possible cases for the translation is illustrated in figure D.3

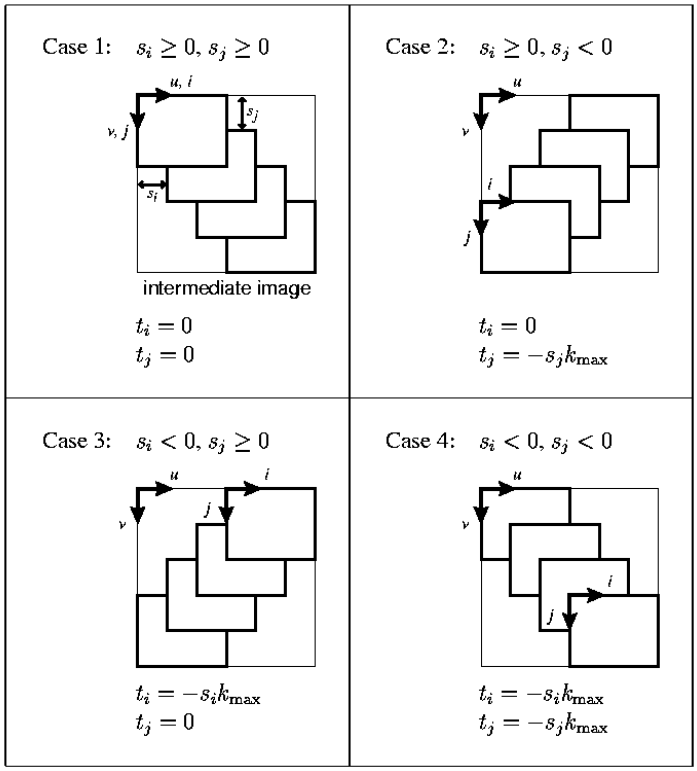

Figure D.3: The four cases of translation the sheared coordinate system. The translation (t_i, t_j) specifies the displacement from the origin of the standard object coordinate system ($i = 0, j = 0$) to the origin of the intermediate image coordinate system (u = 0, v 0 0). k_{max} is the maximum voxel slice index in the volume. Source: [Lacroute and Levoy, 1993]

Furthermore, the stacking order of the voxel slices in the intermediate image has to be calculated. In standard object space the voxel slices are ordered by their k coordinate. The stacking order can be found by examining the component of the viewing direction vector corresponding to the principal viewing axis, $v_{so,k}$. If $v_{so,k} > 0$ then the voxel slice in the $k = 0$ plane is the front slice. Otherwise the slice in the $k = k_{max}$ plane is the front slice.

When the translation has been chosen, the shear and warp matrix factors can be rewritten as seen in D.8 and D.9:

$$M_{shear} = \begin{bmatrix} 1 & 0 & 0 & t_i \\ 0 & 1 & 0 & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & s_i & o \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_i & t_i \\ 0 & 1 & s_j & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.8}$$

$$M_{warp} = \begin{bmatrix} m'_{11} & m'_{12} & (m'_{13} - s_i m'_{11} - s_j m'_{12}) & m'_{14} \\ m'_{21} & m'_{22} & (m'_{23} - s_i m'_{21} - s_j m'_{22}) & m'_{24} \\ m'_{31} & m'_{32} & (m'_{33} - s_i m'_{31} - s_j m'_{32}) & m'_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_i \\ 0 & 1 & 0 & -t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.9}$$

Since the warping is performed in 2D the third row and the third column may be removed from M_{warp}:

$$M_{warp2D} = \begin{bmatrix} m'_{11} & m'_{12} & (m'_{14} - t_i m'_{11} - t_j m'_{12}) \\ m'_{21} & m'_{22} & (m'_{24} - t_i m'_{21} - t_j m'_{22}) \\ 0 & 0 & 1 \end{bmatrix} \tag{D.10}$$

Equation D is the matrix that transforms the 2D intermediate image into the final 2D image:

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = M_{warp2D} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

APPENDIX E

Shading

Source: [Slater et al., 2001]
This appendix contains a description of lighting in a 3D scene, and a model of how light affects 3D surfaces.

Light sources

In computer graphics there are a large number of light sources, where the most common are:

Point light source A single point in 3D space, that emits light equally in all directions.

Spot light source A single point in space, from where light is emitted equally, but only in a limited area and direction. The light may be spread in a cone from the point of origin.

Plane light source A surface in 3D space, with a number of light points spread upon the surface, spreading light equally in a hemisphere.

Global illumination

In computer graphics the ideal shading of surfaces is by utilising a global illumination model, to calculate all lighting in a scene. Global illumination calculates influences from all light sources in all directions at all points of a 3D scene.
To calculate global illumination the following equation is utilised:

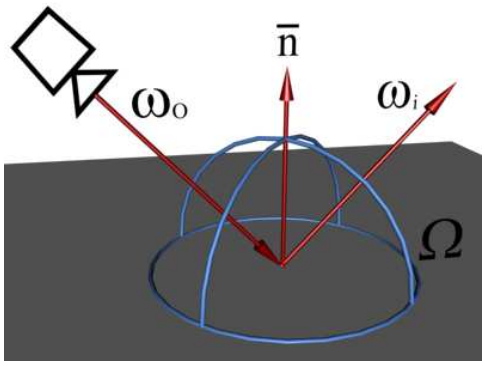

Figure E.1: The global illumination factors.

$$L_(\omega_o) = \int_\Omega BRDF(\omega_o, \omega_i) \cdot L(\omega_i) d\omega_i \tag{E.1}$$

where:
BRDF: Bidirectional Reflection Distribution Function
ω_i: Angle to view point.
ω_o: Exiting angle.
Ω: All angles from calculation point.
An illustration of the factors in the calculation and their placement can be seen in figure E.1 on the facing page.
Calculations of global illumination is a demanding process, which raises a need for a simplified method with less heavy calculations.

Local illumination: Lambert's Reflection Model

A way to simplify illumination calculations is by only calculating the most significant influencing lights, as the peripheral lights has little effect on the illumination. This is called local illumination.

The Lambert Reflection Model is based on local illumination. The model has no base in reality, but is merely a simple way to obtain shading of 3D objects.

The shading of each 3D surface, may be divided into three categories. Diffuse reflection, which is the direct illumination of a surface, which makes the surface visible and specular reflection which controls how the surface shines on surfaces with direct light, see figures E.2 and E.3. Ambient reflection controls the illumination of surfaces not affected by light.

Figure E.2: Diffuse reflection is the direct illumination of a surface, which makes the surface visible.

Figure E.3: Specular reflection controls how the surface shines on surfaces with direct light.

To calculate the total light reflection in a scene the three reflections must be calculated for each point.

Diffuse reflection

Diffuse reflection may be calculated with the following equation. The factors are illustrated in figure E.4 on the next page:

$$I_{r,d} = k_d \cdot I_i \cdot \cos D \tag{E.2}$$

where:
$I_{r,d}$: The resulting intensity value for each edge voxel.
k_d: Diffuse reflection coefficient. One for each colour channel.
I_i: Intensity of the incoming light.

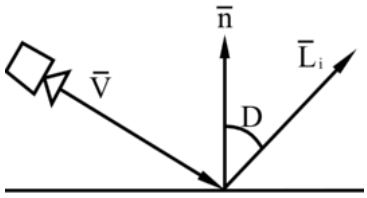

Figure E.4: The diffuse reflection of the Lambert Reflection Model.

D: The angle between the normal vector and the incoming light vector.
$\vec{n}$: The normal vector to the surface.
$\vec{L_i}$: The vector for the incoming light.
$\vec{V}$: The viewing vector.

The diffuse calculations are performed for each colour channel denoted by k_d. The calculations are performed with normalised values, ranging between 0 and 1. With normalised vectors, $\cos D$ may be calculated with the following equation:

$$\cos D = \vec{n} \cdot \vec{L_i} \tag{E.3}$$

Specular reflection

Specular reflection may be calculated with the following equation. The factors are illustrated in figure E.5:

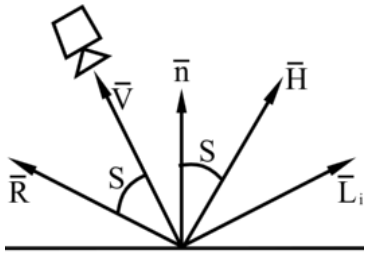

Figure E.5: The specular reflection of the Lambert Reflection Model.

$$I_{r,s} = k_s \cdot Ii \cdot (\cos S)^m \tag{E.4}$$

with:
k_s: Specular reflection coefficient.
I_i: Intensity of the incoming light.
S: The angle between the perfect reflection angle and the viewing angle.
m: Shininess-factor. Ranging from 1 and to values above. The higher the value, the more concentrated the specular reflection will be.
$\vec{L_i}$: The angle for the incoming light.
$\vec{H}$: The angle between the viewing angle and the incoming light angle.

$\vec{R}$: The perfect light reflection angle.

These are the same calculations for each colour channels.
The $\cos S$ may be replaced by:

$$\cos S = \vec{V} \bullet \vec{R} = \vec{n} \bullet \vec{H}$$

Where

$$\vec{H} = \vec{V} + \vec{L_i}$$

By applying the latter replacement the calculations may be simplified, as all factors $\vec{n}$, $\vec{V}$ and $\vec{L_i}$ are known.

Ambient reflection

The ambient reflection is the simplest of the three, as it does not incorporate angle calculations. Ambient is the light that affects all points.

The ambient reflection I_a may be calculated with:

$$I_{r,a} = k_a \cdot I_a \tag{E.5}$$

where:
k_a: Ambient reflection coefficient
I_b: Ambient light intensity

E.0.1 Total reflection

All factors adds to the total reflection like:

$$I_r = I_{r,a} + I_{r,d} + I_{r,s} \tag{E.6}$$

If there are several light sources within the scene, they will have to be calculated one at a time, and the partial results summed at the end. Light sources has no effect on the ambient reflection, hence this can be omitted of the illumination calculations.

$$I_r = k_a + I_a + \sum_{j=1}^{N}(k_d \cdot I_{i,j} \cdot (\vec{n} \bullet L_{i,j})) + \sum_{j=1}^{N}(k_s \cdot I_{i,j} \cdot (\vec{n} \bullet (\vec{V} + \vec{L_{i,j}}))^m) \tag{E.7}$$

This can be reduced to:

$$I_r = k_a + I_a + \sum_{j=1}^{N}(I_{i,j} \cdot (k_d \cdot (\vec{n} \bullet \vec{L_{i,j}}) + k_s \cdot (\vec{n} \bullet (\vec{V} + \vec{L_{i,j}}))^m)) \tag{E.8}$$

The summations ranges from 1 to N, with N denoting the number of light sources.

APPENDIX F

Compression Methods

This appendix investigates how much different types of compression can decrease the amount of required memory used to store the voxelspaces.

Each compression method will partially be evaluated by how well it reproduces the uncompressed images seen in figure F.1 and figure F.2, respectively. Furthermore, the compression methods will be evaluated by their ability to save space. The theory for each method will be analysed.

Figure F.1: The uncompressed photographic evaluation image.

Figure F.2: The uncompressed evaluation image of the hourglass glyph.

JPEG compression

Source: [Y.Q.Shi and Sun, 2000]
JPEG is short for Joint Picture Expert Group. In JPEG compression an image is first partitioned into 8x8 blocks. On each block a forward Discrete Cosine Transformation (DCT) is applied, and the resulting coefficients are scanned through a zig zag pattern. The DCT is a variant of the Fourier transformation:

$$FDCT = S_{uv} = \frac{1}{4} c_u c_v \sum_{i=0}^{7} \sum_{j=0}^{7} s_{ij} \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} \tag{F.1}$$

The zigzag pattern can be seen in figure F.3. The reason for the pattern of the zig zag is to maintain a redundancy from step to step through the path.

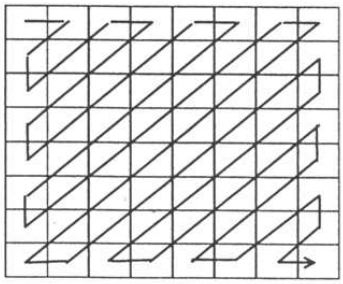

Figure F.3: Zig zag scanning order of the DCT coefficients.

The blocks are then be converted into chrominance and luminance images, which are used in the quantisation-process of the image. In this process the theory is that the first pixel of the block, is the one containing the most information, and that the rest of the pixels in the block, can be derived from the first. With a set of quantisation tables for the luminance and chrominance plus the DCT coefficients it is possible to compress each block into a bit-stream, where the most probable pixels has the shortest bit-sequence, and thus compressing the image. The Huffman tables are used to encode the bit-sequences.

If JPEG-compression is applied to the evaluation image with the optimal quality parameters set, the resulting image can be seen in figure F.4.

Figure F.4: The photographic evaluation image compressed using JPEG.

If the resulting image is subtracted from the original uncompressed image, the errors of the compression, if any, will be visible. In figure F.5 on the next page the result of such a subtraction can be seen.

The computations are also applied to the image of the hourglass. The compressed image and resulting subtracted image can be seen in figure F.6 and in figure F.7 on the following page, respectively.

As the images show, the JPEG compression has problems with areas that have a big contrast, such as the text in image F.1, which is a result of the quantisation-process trying to find a pixel-to-pixel relation in a high contrast area.

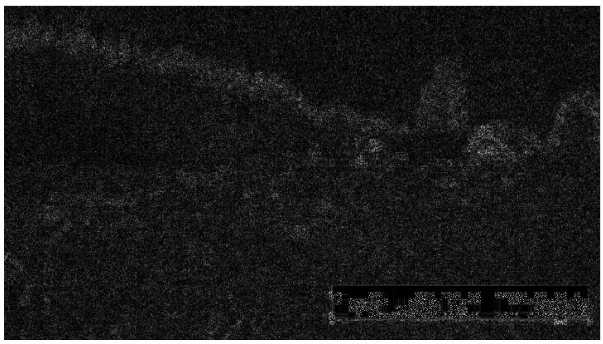

Figure F.5: The errors of the JPEG compression appears as white dots on the background, the whiter the dots, the greater the error

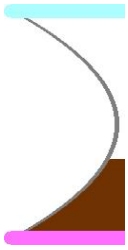

Figure F.6: The evaluation image of the hourglass compressed using JPEG.

Figure F.7: The errors of the JPEG compression appears as white dots on the background, the whiter the dots, the greater the error.

Reduction of file size

The original size of the photographic evaluation image is 848 KB. The JPEG compressed version has a size of 39.2 KB.
The compression ratio for the JPEG compression on photos is then: $848/39.2 = 21.6$.
The original size of the glyph evaluation image is 96 KB. The JPEG compressed version has a size of 2.53 KB.
The compression ratio for the JPEG compression on redundant images is then: $96/2.53 = 37.94$.

Advantages/disadvantages

Advantages: Good compression ratio. When dealing with normal photographs, the use of JPEG compression is invisible in the image.

Disadvantages: The compression is not lossless. When dealing with images with high contrast, the compression is very visible. No alpha channel.

PNG/MNG compression

Source: [W3C, 1996]
PNG is short for Portable Network Graphics. The format has been developed to provide a format with good compression and lossless reproduction. Furthermore, alpha compression is implemented in the codec.

PNG compression specifies deflate/inflate compression. Deflate compression is an LZ77 derivative used in zip, gzip, pkzip and related programs. Deflate-compressed data streams within PNG are stored in the "zlib" format, which has the structure:

Compression method/flags code:	1 byte
Additional flags/check bits:	1 byte
Compressed data blocks:	n bytes
Check value:	4 bytes

The compressed data within the zlib data stream is stored as a series of blocks, each of which can represent raw data, LZ77-compressed data encoded with fixed Huffman codes, or LZ77-compressed data encoded with custom Huffman codes. A marker bit in the final block identifies it as the last block, allowing the decoder to recognise the end of the compressed data stream.

The check value stored at the end of the zlib data stream is calculated on the uncompressed data represented by the data stream. The zlib check value is useful mainly to cross-check that the deflate and inflate algorithms are implemented correctly.

In a PNG file, the concatenation of the contents of all the chunks makes up a zlib data stream as specified above. This data stream decompresses to filtered image data.

There is no required correlation between the structure of the image data (i.e., scanline boundaries) and deflate block boundaries or chunk boundaries. The complete image data is represented by a single zlib data stream that is stored in some number of chunks.

An additional option when using PNG, is to reduce the palette of colours, to describe the image using only 256 (8 bits) different colours. This will result in an image that is flawed in comparison to the original image. However images that already utilises few colours may gain a powerful compression this way.

MNG (Multiple-image Network Graphics) is a variety of PNG, that allows a stream of different images to be compressed in a single MNG-file, [Roelofs, 2002]. The stream of images could be a volumespace.

24 bits PNG compression is applied to the evaluation image, and the result may be seen in figure F.8.

Figure F.8: The PNG compression using 24 bits. Yielding a lossless reproduction.

There are no visible compression artifacts in the resulting PNG-compressed image. Due to the fact that PNG is lossless, there is no reason to check for compression errors using a subtracted image.

8 bits PNG compression is applied to the evaluation image, and the result may be seen in figure F.9.

Figure F.9: The PNG compression using 8 bits. Yielding a lossy but good reproduction.

The original image is not the type of image, that 8 bits PNG (PNG8) is made for. This means that there will occur errors in the resulting compressed image. Used on the right type of images, such as glyph-images, with a limited use of different colours, this will reproduce the image lossless, and highly compressed. The subtracted image in figure F.10 on the next page shows the errors using PNG8 on a 24 bits image. As can be seen, the reproduced PNG8 image is very close to the original image.

The computations are also applied to the image of the hourglass. The compressed image and resulting subtracted image can be seen in figure F.11 on the facing page and in figure F.12 on the next page, respectively.

It can be seen on figure F.12 on the facing page that no errors occur when compressing the hourglass glyph.

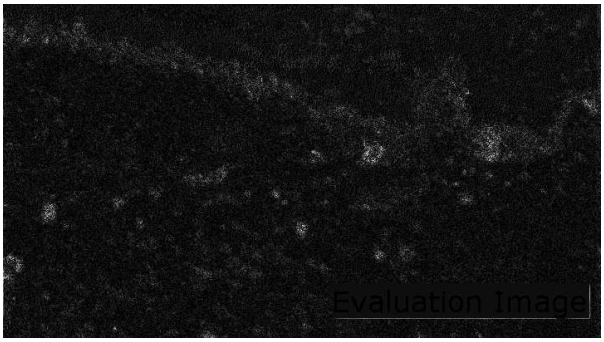

Figure F.10: PNG8 subtracted image.

Figure F.11: The evaluation image of the hourglass compressed using PNG.

Figure F.12: The errors of the PNG compression appears as white dots on the background, the whiter the dots, the greater the error

Reduction of file size

The original size of the photographic evaluation image is 848 KB. The 24 bit PNG compressed version has a size of 352 KB.
The compression ratio for the PNG compression is then: $848/352 = 2.4$.
The original size of the Hourglass evaluation image is 96 KB. The 24 bit PNG compressed version has a size of 3.1 KB.
The compression ratio for the PNG compression is then: $96/3.1 = 30.96$.

The 8 bit PNG compressed version of the photographic evaluation image has a size of 171 KB.
The compression ratio for the PNG compression is then: $848/171 = 5$.
The 8 bit PNG compressed version of the hourglass glyph has a size of 1.79 KB.
The compression ratio for the PNG compression is then: $96/1.79 = 53.63$.

Advantages/disadvantages

Advantages: The compression is lossless. Alpha channel. Good compression ratio, when dealing with images containing large redundancy.

Disadvantages: Bad compression ratio, when dealing with normal photographs.

APPENDIX G

User interview

In this appendix the guideline questions for the expert user interview are described.

To study the perceptual properties and performance of the three advanced objects which have been constructed, a user expert has been selected to comment on these concepts.

Questions concerning the interview are:

- Do the three glyphs look acceptable in their form? One example is how the perception is for an arbitrary number of slices, for the glyph to seem real. Another example is, if the features mapped to the glyph are recognisable.
- Is it possible to observe the differences in the features where the data set is normalised from 0 to 1? Or should the interval from 0 to 1 be divided into subintervals, so that one feature represents more than one data dimension?
- Is the animation frame rate and the glyphs appearance acceptable, when navigating in the 3D space?
- Can the three glyphs be expanded in any way, so the representation of the data set is more adequate and are there other advanced glyphs which can be useful?
- Are there areas of data mining where using advanced glyphs is advantagous?
- How detailed should the information for each glyph be? Should the user be able to select a certain glyph and have the information concerning that glyph, in terms of which data dimensions are mapped to which features?

APPENDIX H

Test

This appendix contains the different tests performed on the system. The tests are: General functionality Test, Performance Test and Immersion Test.

H.1 General functionality Test

H.1.1 Purpose

To test the basic functionalities of the system.

H.1.2 Setup

Computer: Abyss computer at CVMT.
CPU: 2xP4 Xeon 2.4 GHz
Memory: 2 GB
Graphics Card: GeForce 4 dual, driver: 1.0-5336
OS: Mandrake Linux 9.2, kernel 2.6.3-7

Programme: Volumetrics build 0.1
Window resolution: 640×480
Volume resolution: $128 \times 128 \times 128$
Planes per glyph: 16
Plane texture Resolution: 128×128
Data set: 2sphere, an artificial data set with samples placed in two sphere shaped clusters.

Test 1 is testing a normalised data set, which consists of 50 records with 4 data dimensions for each record. If the data dimensions are mapped correctly to the features the test is successful.

Test 2 involves testing whether continuous remapping of data dimensions to object features is possible. The test is a success if remapping occurs correctly.

H.1.3 Results

The hourglass glyph is utilized for the test. The initial mapping of coordinates:

The mapping is done correctly.

The GUI is called, and the mapping is reconfigured to:

Dimension	Feature
X	X
Y	Y
Z	Z
Cluster (W)	Shape
none	Pose
Z	Sandheight
X	Sand colour 1: Green
Y	Sand colour 2: Red
X	Head colour: Cyna
Y	Foot colour: Magenta

Table H.1: The mappings used with the hourglass.

Dimension	Feature
Y	X
Z	Y
X	Z
Z	Shape
none	Pose
Cluster (W)	Sandheight
Z	Sand colour 1: Green
Z	Sand colour 2: Red
Y	Head colour: Cyna
X	Foot colour: Magenta

Table H.2: The new mappings used with the hourglass.

The Initial mapping of coordinates:

And the system displays the new configuration correctly.

Similar tests were performed with cross and diamond glyph, and yielded the same result. The system is also able to switch between different glyphs from one mapping to the other.

H.1.4 Discussion

The system is able to map data set records correctly onto a glyph. Furthermore, the system is able to remap a glyph with new features correctly, and switch between different glyphs.

H.2 Performance Test

H.2.1 Purpose

To evaluate the performance of the developed system, the system will be measured for frame rate, and time to slicing volumes.

H.2.2 Setup

Computer: Abyss computer at CVMT.
CPU: 2xP4 Xeon 2.4 GHz
Memory: 2 GB
Graphics Card: GeForce 4 dual, driver: 1.0-5336
OS: Mandrake Linux 9.2, kernel 2.6.3-7

Programme: Volumetrics build 0.1
Window resolution: 640×480
Volume resolution: $128 \times 128 \times 128$
Planes per glyph: 16
Plane texture Resolution: 128×128
Data set: 2sphere, an artificial data set with samples placed in two sphere shaped clusters.

H.2.3 Results

The test is performed using two setups: a scene with 50 glyphs, and a scene with 1000 glyphs. With 50 glyphs the frame rate is measured for every 100 animation frame. With 1000 glyphs the frame rate is measured for every 10 frame. For both setups the time to reslice all glyphs is measured. All scenes has been tested in both navigation and High Definition (HD) mode.

50 Glyph scene

The scene is tested with two modes, HD mode and navigation mode.

HD mode frame rate
Cross Glyph: 9.06 fps
Diamond Glyph: 9.1 fps
Hourglass Glyph: 9.05 fps

Navigation mode frame rate
Cross Glyph: 143.74 fps
Diamond Glyph: 144.76 fps
Hourglass Glyph: 145.99 fps

Reslicing of 50 glyphs: 3.9 seconds.

250 Glyph scene

The scene is tested with two modes, HD mode and navigation mode.
HD mode frame rate
Cross Glyph: 1.78 fps
Diamond Glyph: 1.82 fps
Hourglass Glyph: 1.8 fps

Navigation mode frame rate
Cross Glyph: 30.14 fps
Diamond Glyph: 30.92 fps
Hourglass Glyph: 30.33 fps

Reslicing of 250 glyphs: 19.55 seconds.

500 Glyph scene

The scene is tested with two modes, HD mode and navigation mode.

HD mode frame rate
Cross Glyph: 0.91 fps
Diamond Glyph: 0.9 fps
Hourglass Glyph: 0.89 fps

Navigation mode frame rate
Cross Glyph: 15.64 fps
Diamond Glyph: 15.25 fps
Hourglass Glyph: 15.13 fps

Reslicing of 500 glyphs: 39.3 seconds.

1000 glyph scene

As seen in the previous tests, there is no significant difference in the performance of each glyph type, and the test with 1000 glyphs has been tested with only the hourglass glyph in both modes.

HD mode frame rate
Hourglass Glyph: 0.405 fps

Navigation mode frame rate
Hourglass Glyph: 7.5 fps

Reslicing of 1000 glyphs: 78.44 seconds.

Calculations

Time to slice each glyph in 50 glyph scene: $\frac{3.9}{50} = 0.078$
Time to slice each glyph in 250 glyph scene: $\frac{19.55}{250} = 0.078$
Time to slice each glyph in 500 glyph scene: $\frac{39.3}{500} = 0.078$
Time to slice each glyph in 1000 glyph scene: $\frac{78.44}{1000} = 0.078$

The relation between the frame rate and the number of glyphs may be seen in figure H.1

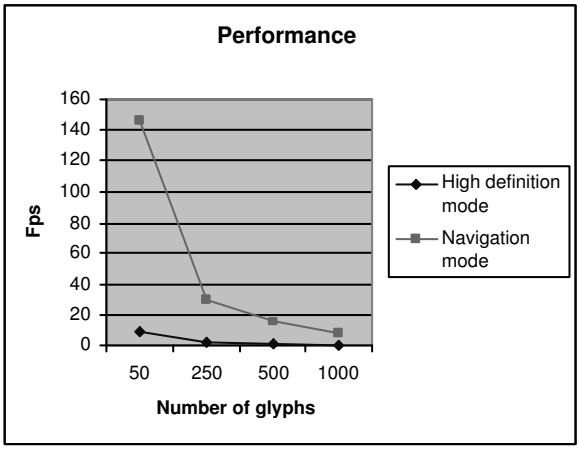

Figure H.1: The relation between the frame rate and the number of glyphs.

H.2.4 Discussion

The time to slice the glyphs in a scene is proportional to the number of glyphs in the scene.
The relation between the frame rate and the number of glyphs displayed in a scene is opposite proportional, when the number of glyphs are doubled, the frame rate is halved.
As the results show, the navigation mode has a significantly higher frame rate than the high definition mode, which means that the implementation of the navigation mode is a necessity for the usability of the system.

H.3 Object perception

H.3.1 Purpose

The purpose of this test is to evaluate the degree of perception with the use of one glyph type, the hourglass glyph.

H.3.2 Setup

Setup is on the hardware system described in the performance test. The display device is a 3 by 4 meter back projection screen.

Dimension	Feature
X	X
Y	Y
Z	Z
Cluster (W)	Shape
none	Pose
Z	Sandheight
X	Sand colour 1: Green
Y	Sand colour 2: Red
X	Head colour: Cyna
Y	Foot colour: Magenta

Table H.3: The mappings used with the hourglass.

Two users with no previous knowledge of the used data set are chosen to participate in the test. The data set used can be found on the accompanying CD-rom, entitled 2sphere.csv. The data set contains 100,000 records, distributed in 2 clusters.

The test is performed with 250 glyphs in the scene.

[Raja et al., 2004] define 6 different evaluations, of which we have chosen to use the five following.

1. One Axis Distance
One Axis Distance asked the subject to find the point with the highest Y value. This task tested their ability to judge distances along one axis in the scatterplot. We felt this was an important basic task in that it is a key component in gaining understanding of a single data point.

2. Two Axis Distance
Two Axis Distance required the subject to locate the point with the both the lowest X and lowest Y value. We were concerned about the subjects ability to orient themselves so that they could compare distances along two axes at once. This proved to be one of the more difficult tasks.

3. Trend Determination
Trend Determination required the subject to get a general sense of the layout of the data in order to spot trends. Subjects were asked to report the trend in this format: as A increases/decreases, B increases/decreases.

4. Clusters
The Cluster finding task asked subjects to locate clusters of data points greater than 20 points.

5. Single Point Search
The Single Point Search task had subjects locate a differently colored point in a densely packed group of points. The point was not visible from the subject s starting position and required them to navigate to find it. This task would be important in a real-world application where a data point is highlighted in another view and must then be located in the VE view (e.g. a brushing-and-linking task).

Mappings

H.3.3 Results

250 glyphs
Question 1
The two users quickly pointed out the glyph.

Question 2
The trend of different colours based on the XY coordinate was quickly picked up.

Question 3
Again, colours based on the XY coordinate were quickly picked up, and the clustering in two groups was seen. After a relatively long time, the it was discovered, that the shape represented the cluster.

H.3.4 Discussion

Determining the glyph with the largest Y component, which demonstrates, that the ability to understand one data point is good. The trends were quickly spotted based on glyph colour, which confirms that colour is one of the most powerful and robust features in 3D visual data mining. The last question demonstrated, that the shape of the hourglass glyph is not a very conspicuous feature.

APPENDIX I

Class diagram