RELIEF TEXTURE MAPPING

by

Philip West

A thesis submitted to the Department of
Computer Science in partial fulfilment
of the requirements for the degree of

Bachelor of Science Honours

Rhodes University

2003
ABSTRACT

Image-based modelling and rendering techniques have the potential to simplify model geometry and increase realism in a scene. A relief texture mapping system based on the technique described in the paper “Relief Texture Mapping” by Oliveira, Bishop and McAllister [2000] is presented. This system was used to evaluate the possibility of rendering photo-realistic scenes at interactive rates. The technique makes use of textures with displacement values to increase realism in three-dimensional scenes. Pre-warp equations are used to modify the textures so as to support the representation of three dimensional surface details and view motion parallax. Conventional texture mapping is then used to convert from texture coordinates to screen coordinates. A proof that object complexity does not affect the frame rate is presented. Adding multiple relief textures to a scene affects the average frame rate in a linear manner. An average frame rate of 3.9540 frames per second was achieved by pre-warping and image reconstruction. Three optimizations were developed, of which only one affects the pre-warping process. Interactive rates were not recorded, but many optimizations exist that could still be implemented to achieve this goal.
ACKNOWLEDGEMENTS

I’d like to thank the following people for all their help and advice throughout the year:

(a) Shaun for his help with OpenGL and C++. Your assistance was invaluable and this project could not have been done without you.

(b) Adele and George for sharing their experience in project management and advice on writing up. Special thanks to
Adele for proof reading all my work so efficiently!

(c) The members of the Virtual Reality Special Interest Group (VRSIG) for their comments and suggestions.

(d) Olivia Norton for her support and advice.

(e) Anyone else who helped me in any way. Sorry I forgot to mention you.
# TABLE OF CONTENTS

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv</td>
</tr>
</tbody>
</table>

## LIST OF TABLES AND EQUATIONS

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi</td>
</tr>
</tbody>
</table>

## (a)

### Chapter 1 – INTRODUCTION

1.1 Motivation for the Study ......................................................... 2

1.2 Thesis Statement ................................................................. 2

1.3 Structure of the Study ......................................................... 3

1.4 Results ................................................................................ 3

1.5 Discussion .......................................................................... 4

### Chapter 2 - RELATED WORK

2.1 An Introduction to Image-based Modelling and Rendering .......... 5

2.2 Image-based Rendering Techniques ....................................... 7

2.2.1 Texture Mapping .............................................................. 7

2.2.2 Bump Mapping ............................................................... 8

2.2.3 Image Morphing .............................................................. 8

2.2.4 3D Image Warping ......................................................... 9
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.5 Trilinear Tensor</td>
<td>9</td>
</tr>
<tr>
<td>2.2.6 Sprites with Depth</td>
<td>10</td>
</tr>
<tr>
<td>2.2.7 Layered Depth Images</td>
<td>10</td>
</tr>
<tr>
<td>2.2.8 Multiple-Centre-of-Projection Images</td>
<td>11</td>
</tr>
<tr>
<td>2.2.9 Plenoptic Functions</td>
<td>12</td>
</tr>
<tr>
<td>2.2.9.1 Plenoptic Modelling</td>
<td>12</td>
</tr>
<tr>
<td>2.2.9.2 The Lightfield and the Lumigraph</td>
<td>13</td>
</tr>
<tr>
<td>2.2.9.3 Concentric Mosaics</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Discussion</td>
<td>14</td>
</tr>
<tr>
<td>Chapter 3 - RELIEF TEXTURE MAPPING</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Relief Textures</td>
<td>16</td>
</tr>
<tr>
<td>3.2 Pre-Warping Equations</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Folding</td>
<td>19</td>
</tr>
<tr>
<td>3.4 Hole-Filling</td>
<td>21</td>
</tr>
<tr>
<td>3.5 Capturing Samples Outside of the Source View Plane</td>
<td>23</td>
</tr>
<tr>
<td>3.6 Optimizations</td>
<td>25</td>
</tr>
</tbody>
</table>
Chapter 4 - PROGRAM DESIGN

4.1 Phases of Implementation

4.2 Development Environment and Hardware

4.3 System Architecture

4.4 User Interface

4.4.1 Frame Rate

4.4.2 Pre-warping and Image Reconstruction

4.4.3 Optimizations

4.4.4 Miscellaneous

4.5 Discussion

Chapter 5 - SYSTEM IMPLEMENTATION

5.1 Objects

5.2 Implementation Issues - Converting Texture Coordinates to Screen Coordinates

5.3 Phase 1 - Relief Textures

5.4 Phase 2 - Pre-warp Equations

5.5 Phase 3 - Image Reconstruction

5.5.1 Folding

5.5.2 Hole-Filling
6.5.3 Tolerance

Levels ........................................................................... 50

6.5.4 Discussion ................................................................... 51

6.6 Limitations ...................................................................... 52

6.7 Discussion ...................................................................... 52

Chapter 7 - CONCLUSIONS AND FUTURE WORK .................................................. 54

7.1 Review ........................................................................ 54

7.2 Findings ........................................................................ 54

7.3 Extensions and Future Research ................................................... 54

LIST OF REFERENCES ............................................................................... 56

APPENDIX ONE – RELIEF TEXTURES AND SAMPLE IMAGES ......................... 61

APPENDIX TWO – EQUATIONS AND SAMPLE CODE ....................................... 65

LIST OF FIGURES

Figure 1-a.1. The relief texture mapping process ......................................................... 1

Figure 2-1. Geometry-based rendering ........................................................................ 5

Figure 2-2. The image-based rendering process ......................................................... 6

Figure 2-3. Bump Mapping ....................................................................................... 8
Figure 2-4. Example multiple centre of projection image........................................11

Figure 2-5. The plenoptic function..............................................................................13

Figure 2-6. A concentric Mosaic..............................................................................14

Figure 3-1. Illustration of the relief texture mapping algorithm.................................16

Figure 3-2. Example relief texture map....................................................................17

Figure 3-3. Six relief textures representing a complex object....................................17

Figure 3-4. Image view plane......................................................................................18

Figure 3-5. Image divisions for the painter’s algorithm............................................20

Figure 3-6. Source image showing correct warping order........................................20

Figure 3-7. Hole artefacts.........................................................................................21

Figure 3-8. Pseudocode for pre-warp equations.......................................................22

Figure 3-9. Splatting.................................................................................................23

Figure 3-10. Over flow............................................................................................24

Figure 4-1. Basic class diagram of the relief texture mapping system......................30

Figure 4-2. The user interface of the relief texture mapping system.........................31

Figure 4-3. The system output window displaying a relief texture map..................33

Figure 5-1. Example relief texture............................................................................35
LIST OF TABLES AND EQUATIONS

Page

Table 2-1. A taxonomy of plenoptic functions......................................................12
Table 6-1. Average frame rates of pre-warping and image reconstruction..............45
Table 6-2. Average frame rates of relief textures of varying complexity...............47
Table 6-3. Back-face culling optimization..............................................................48
Table 6-4. Zero displacement optimization............................................................49
Table 6-5. Tolerance levels.......................................................................................50
Equation 3-1a. Next x value....................................................................................18
Relief Texture Mapping

Table of Contents

Equation 3-1b. Next y value ................................................................. 18
Equation 3-2a. K1 constant ................................................................. 18
Equation 3-2b. K2 constant ................................................................. 18
Equation 3-2c. K3 constant ................................................................. 18
Virtual reality scenes are designed to be as realistic and interactive as possible. Funkhouser and Séquin [1993] define an interactive frame rate to be a constant 10 frames per second (fps), slower frame rates may appear “jerky” and greatly diminish the interactive feel of the system. The measure of quality used to evaluate the realism of images generated is whether the image is distinguishable from a photograph or not.

Texture mapping allows for increased realism in scenes while maintaining an interactive frame rate. It is limited because a two-dimensional (2D) image is added to a surface and this absence of parallax may become evident when the surface is viewed from certain angles. Relief texture mapping, a form of image-based modelling and rendering (IBMR), is an extension to conventional texture mapping that allows for the representation of three-dimensional (3D) objects while achieving an interactive rate of display. Oliveira, Bishop and McAllister [2000] describe a technique that uses a two-pass, pre-warp, forward transform process followed by conventional texture mapping (Figure 1-1). The pre-warp equations are used to manipulate relief textures, textures with a per texel displacement value. This process handles the parallax effects that result from viewing 3D objects from differing angles. Conventional texture mapping is then used to convert the texture coordinates to screen coordinates, as well as handling scaling, rotation and any remaining perspective transformations [Oliveira, 2000].
1.1 Motivation for the Study

Geometry-based modelling has been used to create photorealistic scenes but this approach is difficult and computationally expensive [McMillan, 1997]. Attempts to create realistic scenes have increased geometric complexity, and therefore the number of polygons used in a scene. This increase in complexity increases the computation time required to render these scenes. McMillan [1997] suggests that photographs can be used to represent model complexity and can achieve photo-realism that is beyond any modelling techniques currently available. Images are also easy to acquire and this simplifies the model building process.

Numerous relief texture mapping systems have been implemented, such as those by Oliveira [2000] and Parilov [2002]. A maximum of six polygons, which have a matching relief texture for every face, are required to render a complex object. This greatly reduces the number of polygons required to represent an object as some object models may contain tens of thousands of polygons. The time required to compute each relief texture is constant, meaning that the complexity of an object becomes insignificant [Oliveira, 2000]. These IBMR systems have achieved rates of display that are interactive or close to it. There are various possible extensions to this technique which have yet to be implemented. Fujita [2002] developed a hardware implementation of the technique. This implementation allows for faster rates of display as well as for lighting and shading effects. IBMR, and in particular relief texture mapping, is fast replacing conventional methods of representing 3D objects owing to the computational speedup and increase in realism that it can provide.
1.2 Thesis Statement
This project aims to investigate IBMR techniques as a method of generating photorealistic scenes at an interactive frame rate. A relief texture mapping system, as described by Oliveira et al., [2000], will be implemented as a platform to test the possibility of achieving these desired outcomes. The system should be able to render scenes from arbitrary view points by using images with displacement as input combined with the pre-warp equations developed by Oliveira [2000]. Solutions to the various problems associated with re-projecting images will also be implemented.

1.3 Structure of the Study
This is a study of IBMR techniques which focuses on the relief texture mapping process. The first chapter provides an introduction to relief texture mapping and an overview of the study. Project aims are stated and a summary of results is provided. IBMR techniques are described in chapter two. Work which has been carried out in related fields is summarised and any relevance to this project is discussed. Chapter three provides a detailed description of the relief texture mapping algorithm and various extensions and optimizations that have been implemented by other researchers. Chapters four and five build on this by assessing the design of a relief texture mapping system and its implementation. Various issues are mentioned and the solutions developed. Results obtained during this project are presented and analysed in chapter six and any benefits and limitations of the system are discussed. The final chapter provides conclusions on the work done and a summary of all findings. Possible extensions to the system are also presented.

1.4 Results
A summary of the results obtained during this project is provided below.

(f) A method of creating relief textures.
(g) A one-pass forward mapping implementation of the pre-warping equations developed by Oliveira [2000].

(h) The development and implementation of equations that allow for relief textures to be correctly displayed on all OFF objects.

(i) An algorithm that makes use of a depth buffer to ensure correct visibility.

(j) Filling of holes created by pre-
Chapter 1 - Introduction

(k) A method for catching texels warped outside of the destination images view plane by attaching auxiliary polygons to extend the plane.

(l) Proof that the complexity of an object represented using relief texture mapping techniques does not affect the time required to compute...
Relief Texture Mapping

Chapter 1 - Introduction

(m) The implementation of a back-face culling algorithm that reduces computation time.

(n) A zero displacement clipping algorithm as described by Oliveira [2000].

(o) A frame rate tolerance speedup method based on the techniques developed by Parilov [2002] which allows for interactive rates of display.

1.5 Discussion
Relief Texture Mapping

Chapter 1 - Introduction

This chapter provides an introduction to IBMR and specifically relief texture mapping. Project aims are stated, the structure of the study is described and motivation for the study is also presented. A summary of the results achieved during the project is listed. Chapter two gives an overview of the IBMR process and describes related techniques.
2.1 An Introduction to Image-based Modelling and Rendering

Computer graphics has conventionally involved synthesizing images from geometric object models (Figure 2-1). A 3D model of a scene is created and then a 2D image of the scene is rendered. The models are assigned surface descriptions detailing such properties as reflectivity and opacity. In this type of system the interaction of light, described by a global illumination model, with a scene is calculated and this is used to render the final image. Objects can be shaded in order to reduce optimal illusions that emphasize boundaries between polygons [Bangay, 2003].

IBMR (Figure 2-2) is an alternative technique to conventional geometric approaches that generates synthesized images, which represent 3D objects, directly from other images [McMillan, 1997]. This allows for novel views, which are views from an arbitrary position, to be created. In this way the need for geometric models of objects is removed. The technique involves a projective mapping of texels from a source image to their correct position in the destination image [McMillan, 1997]. Rendering of new images is thus simplified to a 2D problem. McMillan [1997] suggests the possibility of synthesizing images using image-based methods instead of geometry-based methods. These synthesized images can be generated using fewer computational operations than images that are generated based on geometric models, and “are at least as realistic as those images synthesized from geometric models” [McMillan, 1997].
IBMR techniques have two main benefits; realism and speed. Creating realistic scenes using conventional computer graphics techniques is difficult since object models are used [McMillan and Gortler, 1998]. When IBMR techniques are used it is possible to use real world images, which are realistic by definition, and images generated from these source images will also be realistic. The time required to render scenes when using conventional methods increases with the complexity of the scene, so in order to achieve interactive rates the complexity of the scene must be restricted. With IBMR methods the rendering time depends only on the sampling density of the image and thus no bounds need be placed on the complexity of the scene.

IBMR techniques also exhibit various disadvantages [Parilov, 2002]. It is difficult to render moving objects as the new image is synthesized from only a few given images. View-dependant effects, such as fog and transparent surfaces, are difficult to render since they require view position information which is not available in reference images. Techniques to simulate correct illumination have been implemented but have large computational costs.

Currently the triangle is the predominant modelling primitive used in geometric techniques. Triangles have various advantages. They can be used to describe any shape, can be easily manipulated using linear methods and are convex in both 2D and 3D space. Regions may be represented compactly when using triangles as only three vertices are needed to describe a plane. The advantages associated with polygons are reduced as the size of the triangle decreases, this occurs as the realism of scenes is
Relief Texture Mapping
Chapter 2 – Related Work

increased. McMillan and Gortler [1998] suggest that IBMR methods could replace the use of triangles as the predominant modelling primitive as images are a more efficient method of sampling a scene from an arbitrary viewpoint. IBMR methods can be used for synthetic images as well as photographic ones. Other factors contributing towards the move to IBMR include the availability of inexpensive digital image-acquisition hardware as well as recent graphics accelerator architecture developments [McMillan and Gortler, 1998].

IBMR methods have numerous applications. Images can be used to replace 3D representations of scenes and thereby increase realism and speed of rendering. Such methods can be used as a form of video compression. Key frames are stored at set intervals, and the required images can be generated from these source images. When using such a technique it needs to be determined if the given reference images contain enough information to recreate a scene from arbitrary viewpoints.

2.2 Image-based Rendering Techniques

2.2.1 Texture Mapping

Standard 2D texture mapping is commonly used to increase realism in virtual reality scenes. A stored image file is superimposed onto a smooth surface in order to add colour detail to a scene [Bangay, 2003]. For example, a picture of bricks can be added to a polygon to give the appearance of a wall. Texture mapping is the most commonly used IBMR technique in computer graphics due to the small computational cost of implementation and the large increase in realism [McMillan and Gortler, 1998]. This technique is restricted in the sense that it does not support the representation of 3D surfaces. When a textured object is viewed from certain angles the absence of parallax may reveal that the surface is actually flat [Oliveira et al, 2000].

Projective texture mapping is an extension to texture mapping that allows for collections of primitives to share a texture defined from a particular viewpoint. Heckbert and Moreton [1991] introduced such a system that provided for the interpolation of texture coordinates and shading parameters for polygons in a scene.
View-Dependant Texture Mapping is another extension to standard texture mapping that involves generating novel views of a scene with only approximately known geometry. Debevec et al. [1998] implemented a three-pass method that firstly calculates, for each polygon in a scene, the set of source images for which are visible. These values are then used to create a view map which describes the best texturemap to use for various possible view points. Images are then blended together and mapped to appropriate polygons in a scene using projective texture mapping. Although it allows for the correct view from arbitrary directions, view-dependant texture mapping does not allow for 3D representations of texture images.

2.2.2 Bump Mapping

A method for simulating bumps or wrinkles on a surface without the need for a geometric object model was developed by James Blinn in 1978 [Fusiello, 2003]. The surface normal of a polygon, which is used to determine the orientation of that surface, is perturbed at each vertex according to a stored set of values known as a bump map (Figure 2-3) [Bangay, 2003]. The intensity at each point is then calculated using a global illumination model. When a shading model is applied to the scene, the effects created by this technique become visible.

![Figure 2-3. (a) Sphere with no bump mapping. (b) Bump mapped sphere. [Bangay, 2003].](image)

2.2.3 Image Morphing

The aim of image morphing is to generate intermediate images that represent the fluid transition between two source images. Colour and position of texels in the reference images are interpolated to generate these intermediate images. Image morphing is
mainly used as an animation technique. A 2D function is used to transform images. The first image is gradually faded out while the second image begins to become less distorted [Beier and Neely, 1992]. Different object positions or view points can cause distortions that may appear unnatural. Seitz and Dyer [1996] present a variation of image morphing, known as view morphing, which eliminates the artefacts related with conventional image morphing methods by preserving the shape of an object that is being morphed. Two source images are pre-warped before a morph is computed. Interpolated images are then post-warped to create the final image set. View morphing correctly handles 3D camera and scene transformations without the need for 3D object information.

Chen and Williams [1993] introduced view interpolation which extends view morphing by adding image displacement information to generate novel views of a scene. It allows for the interpolation between images with displacement values. They use a forward mapping technique to move each texel in the source image to its destination in the desired image. 3D screen coordinates of each texel are used to calculate the new position of that texel. The depth buffer is used to ensure correct visibility. Generated view points are constrained to within a ninety degree angle of the original view point.

**2.2.4 3D Image Warping**

McMillan [1997] introduced a method to correctly reproject displacement images onto different view planes. A warp equation maps texels in a source image to the correct position in an arbitrary desired view. A visibility algorithm, known as a painter’s algorithm, determines a drawing order which ensures correct visibility of the desired image. This painter’s algorithm can be used in relief texture mapping and is therefore discussed in greater detail in Chapter 3. There are certain cases after the image has been warped that the remaining transformations cannot be handled by a texture mapping operation. Oliveira [2000] observed that 3D image warping can be simplified to a 2D problem by factoring out rotation operations. The remaining operations can then be handled by standard texture mapping. This is the basis of relief texture mapping.

**2.2.5 Trilinear Tensor**
Avidan and Shashua [1998] developed a technique to generate novel views of a scene from three source images. Three matching measurements across three views satisfy linear constraints. These constraints are known as the trilinear tensor and enable novel views to be synthesized accurately. The relationships between the images for a given point in the scene are examined. The trilinear tensor is a set of coefficients that solve the geometric relations between three uncalibrated projective cameras. This allows for the geometry of the images to be captured. Tensor operators modify the tensor coefficients correctly reflecting the motion of one of the cameras.

The algorithm follows the following form. First the optic-flow between the reference images is calculated. Optic flow describes the movement from one source image to another. The trilinear tensor is then constructed. Any movement of a camera is used along with the tensor to construct a new tensor. This novel image is then reprojected onto the scene. Avidan and Shashua [1998] implement a general warping function that does not require an associated displacement map or the complete reconstruction of camera parameters from reference cameras.

2.2.6 Sprites with Depth

Sprites are images that have transparent texels which are rendered onto surfaces [Shade et al., 1998]. They can be used to cache results of rendering; these cached results are then used to generate new images using an inverse transform method. Sprites can also be used as drawing primitives. A 3D view-matrix is used to project a portion of the scene onto the image plane. Sprites are enhanced by adding a displacement value per texel. This provides greater realism but can only be used as a forward transform which can cause holes and other artefacts.

A two-step algorithm is used to create a desired image from a source image. The first step creates an intermediate displacement map by forward mapping the source image’s associated displacement map. In the second pass, coordinates are calculated which are used to index the colours of destination texels. Sprites with depth differ from relief texture mapping in that they are an approximation of the 3D warping process, whereas relief texture mapping is an exact factorization. Relief texture mapping also takes
advantage of texture mapping hardware and provides a more efficient image reconstruction method [Oliveira et al., 2000].

2.2.7 Layered Depth Images

When using conventional reprojection methods information which is occluded in the source image may be required in the destination view. This information is lost when the desired image is created [Fusiello, 2003]. Layered depth images (LDIs) solve this occlusion problem. Like sprites with depth, these images have an associated displacement value per texel along with a colour value. Layered depth images can have multiple displacement values associated with the same texel. As the view point is moved away from the centre of the LDI, texels in the first depth layer may be pulled apart causing holes in the image. The displacement texels that are farther away from the view point will fill in the holes created by warping the image. LDIs can be created by a number of depth images into a certain camera view. Only the nearest depth texels in the same texel position are displayed [Shade et al., 1998].

2.2.8 Multiple-Centre-of-Projection Images

A multiple-centre-of-projection (MCOP) image (Figure 2-4) is an image that has been acquired from multiple locations. The technique, introduced by Rademacher and Bishop [1998], solves many of the problems associated with images which have a displacement value per texel. Quality of images generated using IBMR techniques is limited by the number and quality of sample images. In order to recreate every possible view of a scene perfectly it is necessary to sample from every viewing position and in every direction. Combining information from different image samples may decrease representation efficiency [Rademacher and Bishop, 1998]. An MCOP image (Figure 2-4) solves this problem by combining multiple samples into a single image which contains the complete dataset necessary for computing any desired viewpoint.
MCOP images have the advantage of having greater connectivity information when compared with a collection of standard range images which results in improved quality of the rendered images. The ability to sample different portions of a scene at different resolutions provides greater flexibility in acquiring image datasets. An internal epipolar geometry defines how multiple viewings of scene points relate to one another. Benefits associated with conventional range images, such as moderate storage requirements, are maintained [Rademacher and Bishop, 1998].

### 2.2.9 Plenoptic Functions

Adelson and Bergen [1991] introduced the concept of the plenoptic function which describes the light information visible at any point in space and time and from any direction. The way in which materials in the world are arranged determines the structure of light and the plenoptic function formally describes this structure. A set of light rays passing through such a point is termed a pencil. Each ray of light in a pencil has an associated intensity and a collection of rays forms a panoramic image which varies with time, view point and wavelength [Adelson and Bergen, 1991]. Alternatively, the plenoptic function can be considered as all the possible environment maps for a specific scene [McMillan, 1997]. Various IBMR techniques have been developed that make use of the plenoptic function. Some of these techniques are described below and their properties are summarised in Table 2-1.
## 2.2.9.1 Plenoptic Modelling

McMillan and Bishop [1995] used the plenoptic function (Figure 2-5), simplified to five dimensions (5D), to implement various IBMR techniques such as morphing and view interpolation. They claimed that all IBMR approaches are attempts to reconstruct the plenoptic function from a subset of that function. The approach was to sample, reconstruct and then resample the plenoptic function in order to generate novel views of source images. A scene is sampled by taking measurements in a cylindrical path about a point. Reconstruction of the plenoptic function is done by estimating the optic flow of points when the image is translated. The reconstructed function is then resampled to create images from arbitrary view points. Plenoptic modelling has no restrictions on viewing space (Table 2-1).

\[
p = P(\theta, \phi, V_x, V_y, V_z)
\]

![Figure 2-5. The plenoptic function [McMillan and Bishop, 1995].](image)
2.2.9.2 The Lightfield and the Lumigraph

A database of images is created and used to generate new views of a scene. The Lumigraph and the Lightfield simplify the plenoptic function to a four-dimensional (4D) problem by only considering a subset of light rays [Levoy and Hanrahan, 97]. This form of IBMR generates new views from arbitrary viewpoints without the use of displacement values. Source images are combined and resampled in order to create the desired image. Each image has a light field, which is the radiance at a point in a given direction. The 4D Lightfield function describes the flow of light through space in a static scene with fixed illumination. Lightfields are created by inserting source images into the 4D plenoptic function. New images are generated by resampling sections of the Lightfield. Texels are assigned colour values by querying the database for the particular ray. If the ray at that position is not in the database the texel is coloured by interpolating between the rays closest to that point.

Although minimal interpretation of the data is required, these methods require high sampling densities. Many images must therefore be rendered and this may use hundreds of megabytes of memory [Fusiello, 2003]. The method is also restricted to space that is free of obstructions and the light source must be fixed. Fields of view are restricted to a 3D box (Table 2-1) and the reconstruction algorithms are computationally expensive [McMillan and Gortler, 1998]. The Lumigraph differs from the Lightfield in that it considers the geometry of the model that it is mapped onto when generating desired views. The Lumigraph requires considerably more pre-processing than the Lightfield [McMillan and Gortler, 1998].

2.2.9.3 Concentric Mosaics

Shum and He [1999] present a 3D plenoptic function known as concentric mosaics. Mosaics are composed of images taken from different viewpoints of a scene (Figure 2-6). The camera follows a continuous circular path around the scene, capturing all rays passing through the circular region. In this way the plenoptic function is derived. Any novel view of the scene can be rendered from a collection of concentric mosaics since they store most of the rays in the image plane. Rays in the desired view are matched to the rays stored in the original mosaic, or they can be interpolated from neighbouring
mosaics. Viewing space is restricted to a 2D circle (Table 2-1). Concentric mosaics have a much smaller file size than a Lightfield or Lumigraph of the same scene [Shum and He, 1999].

![Figure 2-6. A concentric Mosaic [Shum and He, 1999].](image)

### 2.3 Discussion

McMillan and Gortler [1998] suggest that images could replace geometric models as the primary form of rendering primitive. IBMR techniques offer increased realism with reduced computational times. While these methods suffer from various disadvantages, such as holes and folding, techniques have been developed that solve these problems. Early IBMR methods greatly improved the realism of scenes with relatively small computational cost. Extensions to these techniques allow for the representation of complete complex objects, which can be viewed from any position, and exhibit at least the same rendering quality as conventional methods. The complexity of an object does not affect the rendering time required when IBMR techniques are used. Images with displacement can be used to accurately recalculate the position of texels in an image to allow for correct visibility from novel view points. Database IBMR methods, such as the Lumigraph, index light values and these are used to resample images. Relief texture mapping extends the 3D image warping method introduced by McMillan [1997]. Oliveira [2000] simplified the pre-warp equations to a 2D problem followed by conventional texture mapping. This approach is used as the basis of the project. The next chapter describes the relief texture mapping algorithm in detail and discusses the problems associated with pre-warping images and their various solutions.

This chapter introduced various IBMR methods and related work. Chapter three describes the relief texture mapping process in detail. The pre-warp functions are defined and various methods of image reconstruction are described. Optimization techniques are investigated as a method of achieving interactive rates of display.
Chapter 2 – Related Work
The relief texture mapping algorithm was first implemented by Oliveira [2000] (Figure 3-1). Intermediate images are created by applying pre-warp equations to relief textures. Image reconstruction fills the holes created during the pre-warp process as well as ensuring correct visibility. These pre-warped images are then used to create a final view using standard texture mapping operations.

![Figure 3-1. Illustration of the relief texture mapping algorithm. The pre-warping ensures correct visibility and fills holes. The resulting image is used as input for standard texture mapping which produces the final image. [Oliveira, 2000].](image)

We examine implementations by Oliveira [2000], Policarpo [2002] and Parilov [2002] in order to describe the process in detail. The pre-warp and image reconstruction processes are described first and then optimizations and extensions are examined.

### 3.1 Relief Textures

Conventional images store a colour value for each texel in the image. A relief texture is a parallel projection image with an associated displacement value per texel [Oliveira, 2000]. Relief textures are used as modelling and rendering primitives for the relief texture mapping process. The displacement value is measured as the distance between a reference point and the point that is being sampled. Oliveira [2000] stores the
displacement value in the form of a displacement map where darker areas represent points that are further away from the original reference point than lighter areas (Figure 3-2).

![Figure 3-2. (a) Image storing colour values. (b) Associated displacement map. Darker texels represent greater distance. [Oliveira, 2000].](image)

Policarpo [2002] implements a method that stores displacement values in the alpha channel of an image. 3D models are designed and then converted to a set of six images (Figure 3-2). This method is more compact than the method implemented by Oliveira because only one image is required to represent a relief texture.

![Figure 3-3. The six images exported using the 3D Studio Max plug-in developed by Policarpo [2002]. When mapped to a cube these images can be used to represent a skull.](image)

3.2 Pre-Warping Equations
The pre-warping equations used in relief texture mapping developed by [Oliveira, 2000] are derived by factoring the 3D image warping equation defined by McMillan [1997] into a serial warp followed by conventional texture mapping. Hence the 3D warp is reduced to a 2D problem. The speed of the pre-warping equations is the greatest advantage. Pre-warping a point requires a few simple arithmetic operations [Parilov, 2002]. The pre-warping equations are as follows:

\[
\begin{align*}
    u_i &= \frac{u_s + k_1 \text{displ}(u_s, v_s)}{1 + k_3 \text{displ}(u_s, v_s)} \\
    v_i &= \frac{v_s + k_2 \text{displ}(u_s, v_s)}{1 + k_3 \text{displ}(u_s, v_s)}
\end{align*}
\]

(3-1a)

(3-1b)

Where \((u_s, v_s)\) is the position of a texel on the source plane and \((u_i, v_i)\) is the destination point of the corresponding texel, \(u\) represents the horizontal axis and \(v\) the vertical axis. \(\text{displ}(u_s, v_s)\) gives the displacement value of the texel. \(k_1, k_2\) and \(k_3\) are constants for the configuration of the source and target viewpoint. The equations used to calculate these values are given below:

\[
k_1 = \frac{f . (b \times c)}{a . (b \times c)} \\
k_2 = \frac{f . (c \times a)}{a . (b \times c)} \\
k_3 = \frac{1}{c . f}
\]

(3-2a)

(3-2b)

(3-2c)
Figure 3-4 illustrates the vectors needed in the pre-warping calculations. Vectors \( \mathbf{a} \) and \( \mathbf{b} \) are the basis of the plane of the source image. \( \mathbf{f} \) is a unit vector perpendicular to the source plane. \( \mathbf{C}_s \) is the origin of the source image plane. \( \mathbf{C}_t \) is the target centre of projection (COP) and \( \mathbf{c} \) is a vector from \( \mathbf{C}_t \) to \( \mathbf{C}_s \).

![Image View Plane with Values Required for Pre-Warping Equations](image)

**Figure 3-4. Image view plane with the values required for the pre-warping equations [Oliveira, 2000].**

### 3.3 Folding

During the pre-warping process, texels are warped from a source image to new positions on a destination image. It is possible for two texels to be warped to the same position on this new image. If a texel is warped to a position already occupied by a texel with greater displacement, the incorrect texel will be stored at this position in the image and visibility of the image will not be correct. This problem is known as folding [Fusiello, 2003].

Z-buffering is a well known hidden surface removal method [Parilov, 2002]. It is an image space method that examines each texel in the image in order to determine which face of which object should be displayed at that texel [Bangay, 2003]. The complexity of an image space method depends on the size of the image being considered. Initially all displacements on the destination image are set to a maximum value. Then texels are warped from the source image to the destination image. A comparison is done to determine if the new displacement value is closer to the view point than the one currently at that position. If this test passes, the colour and displacement values at that position replace the values currently stored. If the test fails, the colour and displacement
information are discarded and can thus increase performance. Z-buffering is fast when implemented in hardware, but slow when implemented in software [Parilov, 2002].

To ensure that the correct texel is stored at each position in an image, Oliveira [2000] suggests using a painter’s algorithm, which was developed by McMillan [1997]. The algorithm describes the correct order to warp the texels in so as to ensure correct visibility. An image is divided into nine regions (Figure 3-5).

![Figure 3-5. The nine regions of an image [McMillan 1997].](image)

The desired centre of projection (COP) is projected onto the reference image. The reference image is then subdivided depending on where the desired COP lies. One, two or four subdivisions are possible depending on the position of the desired COP. Each subdivision is then warped in a specified order to ensure correct visibility. Four subdivisions are necessary when the desired COP falls into region E, two subdivisions for regions B, D, F and H and one subdivision for regions A, C, G and I. Texels are warped from the borders of the image towards the epipole (Figure 3-6). This method is faster than the depth-buffer technique since the number of comparisons required is greatly reduced.

<table>
<thead>
<tr>
<th>Region A</th>
<th>Region B</th>
<th>Region C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(u_{min}, v_{min})$</td>
<td>$(u_{max}, v_{min})$</td>
<td></td>
</tr>
<tr>
<td>Region D</td>
<td>Reference Image</td>
<td>Region F</td>
</tr>
<tr>
<td>$(u_{min}, v_{max})$</td>
<td></td>
<td>$(u_{max}, v_{min})$</td>
</tr>
<tr>
<td>Region G</td>
<td>Region H</td>
<td>Region I</td>
</tr>
</tbody>
</table>

*Figure 3-5. The nine regions of an image [McMillan 1997].*
3.4 Hole-Filling

Texels cover 2D images with an even distribution. Holes arise in destination images because texels covering a 3D object have an uneven distribution [Parilov, 2002]. When a source image is warped to create a new image, the resulting texels may be unevenly distributed. This means that holes will occur if only one texel is drawn on the destination image for every texel on the source image [Parilov, 2002]. Figure 3-7 shows the source image with an even distribution of texels and the resulting warped image with holes.

Oliveira [2000] implements a two-pass, one-dimensional (1D) resampling approach in order to fill holes in the destination image. In the horizontal pass, each row of the source image is considered individually. Each texel in the source image row is
Relief Texture Mapping
Chapter 3 – Relief Texture Mapping

transferred to a new position in the corresponding row of the destination image. In the vertical pass, the same is done for each column of the image. The order in which these passes are implemented is insignificant as the resulting destination image will be the same for either order. During each pass any holes are filled by linearly interpolating between adjacent texels (Figure 3-8). Interpolation errors may occur across sharp displacement discontinuities where there is no information about the surface [Oliveira, 2000].

\[
\begin{align*}
&\text{get } U_{in}, V_{in}, D_{in} \\
&U_{next} = \text{Equation}_3\text{-}1a(U_{in}, D_{in}) \\
&V_{next} = \text{Equation}_3\text{-}1b(V_{in}, D_{in}) \\
&\text{for } (U_{out} = \text{integer}(U_{prev} + 1); U_{out} \leq U_{next}; U_{out}++ ) \\
&\quad \text{linearly interpolate } C_{out} \text{ between } C_{prev} \text{ and } C_{in}; \\
&\quad \text{linearly interpolate } V_{out} \text{ between } V_{prev} \text{ and } V_{in}; \\
&\quad \text{put } C_{out}, V_{out} \text{ at } U_{out}; \\
&U_{prev} = U_{next}; V_{prev} = V_{next}; C_{prev} = C_{in};
\end{align*}
\]

Figure 3-8. Pseudocode for a first-pass left-to-right horizontal warp and resampling of one texel with coordinates \((U, V)\), color \(C\) and displacement \(D\). No anti-aliasing computed for simplicity. [Oliveira et al., 2000].

An alternative to this method, described by Oliveira [2000], replaces 1D reconstruction with a mesh-based resampling approach while still using the painter’s algorithm introduced by McMillan [1997]. The image is warped and reconstructed as a set of triangles. The coordinates used for rasterizing the triangles are the coordinates of the
pre-warped texels. Approximately two triangles are required for each of the texels in the source image. Mesh-based reconstruction does not suffer from linear distortions or colour interpolation errors caused by displacement discontinuities.

Inverse warping is used to find, for each desired image texel, an associated source image texel [Parilov, 2002]. Conventional texture mapping is done in this way. When inverse warping is used with relief texture mapping there are no holes and the resulting image quality is very high. According to Parilov [2002] inverse warping techniques are relatively slow and thus cannot be used for interactive systems. The pre-warp equations (Equations 3-1a and 3-1b) define many-to-one mappings and are therefore not invertible [Oliveira, 2000]. The problem arises since the values of displ(u, v) are not known and thus the function will accept multiple solutions. For this reason an inverse approach has to search for the closest sample from the view point along the target ray. It is this search that is computationally expensive. Oliveira [2000] describes a technique for searching along epipolar lines.

Splatting is a hole-filling method that draws a texel as a cloud instead of a single texel in the desired image (Figure 3-9). The texel cloud is opaque in the centre and becomes increasingly transparent towards its edges. In a generated image, splats will overlay one another giving the correct colour at each texel. Parilov and Stuerzlinger [2002] implement an algorithm for determining the size of the splat based on the viewing angle. This is necessary since holes vary in size depending on the view point. The algorithm ensures that no holes are left in an image and that the splats do not cause artefacts by overlapping each other.
Parilov [2002] also uses LDIs to fill-holes created by warping images. LDIs are described in detail in Chapter 2. When texels are warped apart the underlying image layers are used to fill in the holes.

3.5 Capturing Samples Outside of the Source View Plane

It is possible for a texel to be warped to an area that is outside the view plane of the destination image. Oliveira [2000] has termed this overflow. Data is lost and the destination image is drawn incomplete (Figure 3-10a and 3-10d). The restriction placed on the viewing angle by this artefact is dependent on the relief texture as greater displacement values will result in a larger warping distance.

To correct this effect, Oliveira [2000] attaches auxiliary polygons perpendicular to the view plane in order to accumulate these texels (Figure 3-10b). This method requires the use of a complex occlusion compatible algorithm [Parilov, 2002]. Although this method requires rendering extra polygons a maximum of two polygons is needed for any view. Adjacent faces are pre-warped to the desired faces. Because the auxiliary polygons are attached perpendicularly the mappings can be performed using the same pre-warping equations.

Parilov [2002] suggests an alternative approach where the size of the destination view plane is dynamically resized and repositioned to allow the entire destination image to be displayed (Figure 3-10c). This method does not require additional polygons, but does rely on a more complex algorithm. A bounding box is used to achieve this. The vertices of the bounding box are pre-warped before the texture is. This process ensures that all texels will translate to within view plane defined by the bounding box. The new values associated with the projected bounding box are then used to rescale and shift the image view plane to allow for correct visibility of the destination image.
3.6 Optimizations

When objects are rendered certain parts of these objects may have no influence on the appearance of the objects themselves. Clipping is the process of identifying these areas and then not rendering them so as to decrease computation time. This clipping algorithm is similar to Warnock’s Area Subdivision Algorithm, which is described in detail by Bangay [2003]. Parilov [2002] provides an example to better illustrate this concept. A building is represented using IBMR techniques. If a room is viewed through a doorway, part of that room will be obscured by the outer walls. The entire building can be rendered and correct visibility will be obtained since the room and then the walls will be rendered over the rest of the building. Not rendering the occluded parts would save a lot of time, which is the aim of clipping.
Parilov [2002] makes use of a conservative approach which estimates the visible set of relief textures. This is done because determining the exact visible set is computationally expensive. The conservative approach may overestimate the set but will never underestimate it. A quad tree is built where the root of this tree represents a rectangle enclosing all source image points and the nodes of the tree represent axis-aligned rectangles in the source image. Maximal and minimal displacements of the points located inside each of these rectangles are stored at each node. Clipping is done during a depth-first traversal of the tree by using these displacement values to determine if a rectangle of texels will be visible in the destination image. Tree building can be done as a pre-process and can increase texture resolution if the size of the destination view plane is smaller than that of the source image view plane.

Oliveira [2000] describes a 1D clipping algorithm which is applied during the pre-warp phase. Any samples which have zero displacement are not pre-warped, as they will be projected to the same position in the destination image. This optimization is dependant on the number of texels with zero displacement within a particular relief texture.

Parilov [2002] suggests using a tolerance approach to relief texture mapping. This method is a compromise between realism and speed. Parilov [2002] questions whether observers will notice slight changes in parallax, and thus if it is necessary to perform pre-warp operations for every frame rendered. Since texture mapping is decoupled from pre-warping, it is possible to slightly reduce realism in order to increase speed even further by only pre-warping every few frames. By only pre-warping images every two or three frames, the frame rate will be greatly increased, but parallax effects will be slightly incorrect.

Image pyramids are used to represent differing levels of detail at fixed per-pixel cost [Oliveira, 2000]. If a fixed resolution is used then a constant number of calculations must be carried out during the pre-warp stage, regardless of the number of pixels that the image is represented by on the screen. In this way the computation time of the pre-
warp is kept in proportion to the final image size. Image pyramids also reduce aliasing affects.

Fujita and Kanai [2002] developed a relief texture mapping system that replaced part of the pre-warp operations with per-texel shader functions. Instead of generating a pre-warped image from a source image, an offset map is created. Offset maps are generated by writing position information as a RGB value instead of copying source image texels. A normal map stores the normal of each texel in the source image as an RGB value. Complex lighting operations, such as specular highlights and reflection mapping, require normal values. Offset maps can be applied to normal maps to allow for complex shading operations. This system speeds up the relief texture mapping process as well as providing the ability to increase realism even further by allowing for correct lighting and shading of objects.

Oliveira [2000] implemented a shading technique that makes use of normal maps and has extended this to include a shadow map. A shadow map algorithm is used where the scene is rendered from both the desired view point and from each light source. Each pixel of the desired view point is reprojected onto the image planes of the light sources using camera and depth buffer information. This allows for shadows to be cast from and onto relief textures.

3.7 Discussion

Relief texture mapping is a form of IBMR that uses textures with displacement values to allow for changes in parallax. Artefacts may result from using the pre-warping equations described but various methods are available for correcting these problems. Implementations in hardware have allowed for decreased computation time as well as adding complex lighting effects. This chapter describes in detail the algorithm developed by Oliveira [2000] which forms the basis for the project. Various other relief texture mapping techniques and extensions are introduced. Chapter four will describe the design phase of our relief texture mapping system.
4.1 Phases of Implementation

The implementation of the system is divided into four phases, with each phase building on the previous one.

(p) Phase 1 – Develop data structures used to represent relief textures.

(q) Phase 2 – Implement the pre-warping equations developed by Oliveira [2000].

(r) Phase 3 – Develop and implement algorithms for reconstructing pre-warped images.

(s) Phase 4 – Optimize the
Relief Texture Mapping
Chapter 4 – Program Design

4.2 Development Environment and Hardware

Red Hat Linux 8.0 was the operating system used to create and test the system. This platform was suggested by the author’s supervisors owing to project constraints.

The system was written using the C++ programming language. C++ is a widely used and understood programming language. Although C++ is a high-level language it allows for a large degree of low level access. This flexibility provides opportunities for code optimization. According to Schmidt [2003], C++ provides support for object-oriented programming features, such as abstract classes and inheritance which allow for the building of flexible and extensible software.

OpenGL was selected as the graphics application programming interface (API). It is the most widely used and supported graphics API and supplies a broad set of graphics rendering methods including a texture mapping function [Unknown, 2003]. OpenGL is portable across many platforms and is a standardised language. The API is well documented and a large quantity of sample code is available.

The computer used for the development of this system was a Pentium IV 1.8GHz processor with 512MB of RAM. An nvidia Quadro4 900XGL graphics accelerator card with 128MB DDR SDRAM was provided by the Computer Science Department of Rhodes University for use with this project.

4.3 System Architecture

As can be seen from the class diagram (Figure 4-1), the system consists of five main classes. The **system interface** allows the user to interact with the system by manipulating objects on the screen. A detailed description of the user interface is
provided in the next section. The \texttt{GL World} class provides the manipulation operations and is used to render the desired scene. A \texttt{GL World} object can contain zero or many \texttt{object models}. Each object consists of at least one \texttt{triangle}, which can be textured using an associated \texttt{relief texture}.

An \texttt{object model} provides a \texttt{render relief textures} method which is used to calculate the values required for the pre-warp equations. \texttt{Vertices} from each associated triangle are transformed to screen coordinates to provide the required vectors. The \texttt{render textures} method allows for standard OpenGL texture mapping. The \texttt{surface normal} of each triangle is used by the back-face culling algorithm to determine whether or not the pre-warp equations should be calculated and the triangle rendered. \texttt{Relief texture} objects store the colour and displacement values of each texel in separate buffers. The pre-warp method performs the actual pre-warp operations, using the displacement values and the view plane values calculated by the object class. Hole-filling and over flow operations are performed by the \texttt{reconstruct} function.

![Basic class diagram of the relief texture mapping system.](image)

**Figure 4-1. Basic class diagram of the relief texture mapping system.**

### 4.4 User Interface

The system is represented by two windows. A control panel (Figure 4-2) is used to apply different methods to an object which is displayed in the output window (Figure 4-
3). All rotations, transformations and scaling operations are performed on the output window. This is can be achieved using the mouse. Holding down the left mouse button and moving the mouse will rotate an object, and holding down the right mouse button and moving the mouse will result in a translation in that direction. Scaling is achieved by holding down the scroll button, if one exists, and moving the mouse. Alternatively these operations can be performed using the keyboard:

(t) Up Arrow key: Positive rotation about the x-axis.

(u) Down Arrow key: Negative rotation about the x-axis.

(v) Left Arrow key: Positive rotation about the z-axis.

(w) Right Arrow key: Positive rotation about the z-axis.

(x) 'm' key: Positive rotation about the y-axis.

(y) 'n' key: Negative rotation about the z-axis.
The graphical user interface (GUI) has been designed to allow a user to manipulate an object displayed in the output window. The buttons and labels have been implemented to assist in illustrating the results of using different relief texture mapping operations. The control panel and its major functions are described in detail next.
4.4.1 Frame Rate

Label 6 displays the frame rate which is calculated as the number of frames rendered per second. It is used to determine whether or not an interactive rate is being achieved. Button 13 can be used to adjust the number of frames over which the average frame rate is calculated. Measuring the frame rate over multiple frames provides an indication of
average frame rate, whereas single frame times can be used to illustrate variations in the rate.

4.4.2 Pre-warping and Image Reconstruction
The number of relief textures rendered in the scene is displayed at label 7. This is used to test the effectiveness of the back-face culling method (button 5), and to measure the effect of rendering multiple relief textures in a scene. Button 1 toggles between standard texture mapping and relief texture mapping. The aim is to illustrate the difference in parallax changes that the pre-warping equations allow for. By pressing button 11 the user can either display or hide the intermediate image created during the pre-warp stage. Clicking button 2 toggles between not filling holes created during pre-warping and using linear interpolation to fill these holes. This illustrates the extent of the artefacts and the effectiveness of the method. The buttons shown at 16 are used to set the sample resolution. Increasing the resolution will improve image quality but decrease the frame rate. Label 17 displays the value of the current sample resolution. A user can view the effectiveness of the overflow solution by using button 3.

4.4.3 Optimizations
Zero displacement optimization is implemented by button 4. This will not affect image quality but will increase the frame rate. The changes in rate are shown by label 6. Button 5 toggles back-face culling on and off which may also affect the frame rate. Tolerance can be toggled on and off using button 10, and the buttons at 14 allow a user to set the tolerance level n, which is displayed in label 15. Pre-warp calculations are performed every n frames when tolerance is enabled.

4.4.4 Miscellaneous
The methods mentioned here have been added as they provide extra functionality to the system, but are not directly related to the relief texture mapping process. Button 8 toggles between negative and positive displacement values. The term indented refers to negative displacement values which conceptually extend below the view plane, protruding refers to positive displacement values which extend above the view plane.
The buttons at 18 increments and decrement the current displacement value by 0.01. Label 19 is used to display the current displacement value. It is possible to cycle through the collection of relief textures provided with the system using the buttons at 12. This can be used to test the effects of the process on different objects. Clicking button 9 captures each rendered scene. The frames are stored as numbered image files which can be used to create video clips. Button 20 exits the application and closes both the control and the output windows.

![Image](image.png)

**Figure 4-3. The output window displaying a rendered relief texture.**

### 4.5 Discussion

This chapter describes the four phases of project implementation. An overview of the development environment and hardware used is given as well as motivation for their selection. The main classes used in the system are detailed and the operation of the system is discussed. A detailed description of the user interface and its associated functions is provided. Chapter five details the implementation stage of this project by describing the functions of the system in detail.
5.1 Objects
Object file format (OFF) objects are textual descriptions of objects. The geometric shape of objects is stored, along with the texture coordinates of each polygon in the object. Triangles are used to build objects as they can conceptually represent any shape. OFF objects define the image view plane, which relief textures are mapped onto. This file format was chosen because it is a standardised format which most other 3D object formats can be converted to [Bangay, 2003]. Data is represented as ASCII characters, which makes it readable by humans and easy to parse or create files. The project makes use of OFF files that are zero-indexed, uncoloured and triangulated. There are variations of the format and a detailed description of OFF objects is provided by Bangay [2003].

5.2 Implementation Issues - Converting Texture Coordinates to Screen Coordinates
The texture coordinates defined by an OFF object are not necessarily convenient for the coordinate system of the texture map being used. Certain special cases of texture coordinates worked correctly, but other faces produce artefacts. Images are mapped incorrectly as a result of the objects having inconvenient texture coordinates. All the relief textures tested are rectangular in shape. Two triangles are used to create an image view plane, which textures are mapped onto. This divides the image in two sections. Inconvenient texture coordinates cause a break between the sections of the image. A solution to this problem was developed by Bangay [2003] (Appendix Two, 2.1).

5.3 Phase 1 - Relief Textures
Relief textures are represented by a colour image and an associated displacement map as described by [Oliveira, 2000]. The colour image is a 256 x 256 texel portable pixmap (PPM) image (Figure 5-1a). This format is simple to parse, which makes reading and writing operations a fast process. Each colour value is stored in the red, green and blue (RGB) colour format provided by the OpenGL API. A displacement map (Figure 5-1b)
stores the displacement values associated with each texel of the colour image. Displacement maps are stored in the portable grey map (PGM) image file format. A maximum displacement value is specified. Black represents zero displacement, white represents maximum displacement. Black is defined as (0) and the maximum value is a number less than (65536).

![Image](image1.png) ![Image](image2.png)

**Figure 5-1.** (a) An image giving colour values. (b) A displacement map giving displacement values. Black (0,0,0) represents zero displacement and white (1,1,1) represents maximum displacement.

Displacement maps were created manually in the graphic editing program Gimp. The 3D Studio Max plug-in developed by Policarpo [2002] was also used. This plug-in allows object models to be exported from 3D Studio Max as a set of six images, with the displacement values stored in the alpha channel of each image. A displacement map for each of these images is then created using the values in the alpha channel of the colour image.

A problem which originates from using displacement maps is that of maximum value. Here white would represent the maximum displacement possible, but it is unclear what maximum displacement is. Currently this has not been standardised and thus the scale of displacement values is adjusted for different textures. Another problem associated with displacement values that has not been solved is deciding on a unit of measurement.

5.4 Phase 2 - Pre-warp Equations
The equations developed by [Oliveira, 2000] have been implemented using a forward transform, 2D one-pass approach (Appendix Two, 2.2.1). For each texel in the source image, the destination position is calculated and the texel is moved to that position. This is in contrast to the two-pass approach implemented by Oliveira [2000] which performs a horizontal warp followed by a vertical warp. The function performs the pre-warp operations on the source image, and then texture maps the intermediate image onto the specified triangle. Image reconstruction is done during in this function and is described in detail in the next section.

5.5 Phase 3 - Image Reconstruction

5.5.1 Folding

A depth buffer has been used to ensure correct visibility. The method is based on the algorithms described by Bangay [2003] and Parilov [2002]. First all displacement values in the buffer are set to minimum values. When texels are warped to a certain position their displacement values are compared to the value currently at that location. The old value is replaced if the new value is closer to the view point. Computational cost is increased as an additional comparison operation is required for each texel in the source image. The painter’s algorithm developed by McMillan [1997] is a more efficient method of achieving correct visibility, but a depth buffer has been implemented owing to its simplicity and effectiveness.

5.5.2 Hole-Filling

A two-pass, 2D linear interpolation method is used to fill holes. This method is based on the two-pass, 1D resampling approach implemented by Oliveira [2000]. The function determines, for each texel in the destination image, if a hole has been created between it and the texel to its left. The texel directly above it is also considered. When a hole exists the difference between the colours of the two texels is calculated and this difference is averaged over the number of texels that constitute the hole. The same is done for the displacement values at each position. This process is illustrated in Figure 5-2.
Figure 5-2. (a) Texels X and Y are adjacent to one another in the source image. (b) In the destination image X and Y have been warped apart leaving a hole. (c) Linear interpolation is used to fill the hole between the texels.

Since the hole-filling algorithm is a 2D function, a line algorithm is required to find a path between two texels in an image (Appendix Two, 2.2). First the difference between the x coordinates and the y coordinates of the two texels is calculated. The number of texels that must be interpolated across is equal to the x or y difference, depending on which difference is the greatest. The algorithm reduces the size of the holes but will not necessarily fill them completely (Figure 5-3). Holes will be filled correctly if there is only a rotation about the x-axis or the y-axis. If there is an x and y rotation, holes will be reduced in size but will still exist.

Figure 5-3. (a) Source texels with an even distribution. (b) Texels warped to a destination image displaying uneven distribution. (c) Texels in the destination image after the hole-filling algorithm.

Sampling at higher resolutions effectively increases the number of texels in the source image and this reduces the size of the holes in the destination image. Although this method produces high quality images it has a large computational cost. For an image with m texels, an increase in resolution will result in m more texels and thus m more pre-warsps. Increasing resolution is an effective and simple implementation of hole-filling but will not allow for interactive rates of display.

5.5.3 Over flow
Oliveira [2000] attaches auxiliary polygons at right angles to the source view plane in order to catch texels warped to positions outside of this plane. The method used in this project extends the image view plane by attaching polygons parallel to it (Figure 5-4). At most three polygons are required to render a view, whereas the method implemented by Oliveira [2000] requires at most two extra polygons. It is not possible to view the image from an oblique angle since the extended view plane could never catch polygons warped to positions outside of the source image at this angle. The restriction placed on the viewing angle is dependant on the displacement values of the relief texture being used.

![Figure 5-4. Eight auxiliary polygons are attached in order to extend the view plane. Texels warped off the source view plane are texture mapped to the appropriate polygon. At most three auxiliary polygons are required for any view.](image)

5.6 Code Optimization

When a closed polyhedral object is rendered the faces of the object which point away from the view point are not visible [Bangay, 2003]. In the default OpenGL situation the viewer looks straight down the negative z-axis. In this case the surface normals of polygons facing away from the viewer will have a negative z value. Computation time of a scene can be reduced by not rendering these hidden surfaces. OpenGL provides an image space hidden-surface removal method known as back-face culling which implements this automatically.
Although back-face culling ensures that relief textures which face away from the view point will not be rendered, the images are still pre-warped. Since the pre-warp operations are computationally expensive, back-face culling has been extended to remove unnecessary pre-warps. A surface normal is added to each face of the object being rendered. A vector is then calculated from the origin to a point represented by the surface normal. The OpenGL model view matrix is then applied to this vector in order to obtain the surface normal of the face after any rotations or translations. If the z value of this surface normal greater than zero, the pre-warp calculations are done, otherwise they are omitted. The speedup achieved is dependant on the view point and the design of the object being rendered.

The zero displacement clipping method described by Oliveira [2000] has been implemented. Image warping equations do not alter the positions of texels which have zero displacement. Therefore the pre-warp equations can be omitted for any texels with zero displacement without affecting the realism of a scene. The speedup achieved is dependant on the texture used. A texture with a greater number of texels which have zero displacement will result in a greater speed up as this reduces the number of pre-warp operations required.

By setting a tolerance level n, pre-warp equations are only recomputed every n frames. In the intermediate frames the last pre-warped image is mapped to an object using standard texture mapping operations. Using this approach reduces the realism of a scene since parallax effects are not accounted for in every new frame. This implementation is based on the approach suggested by Parilov [2002]. The speed up achieved will depend on the tolerance value n. A greater value of n will increase the level of speed up but will simultaneously decrease realism in the scene.

5.7 Discussion

A one-pass, 2D forward transform approach to relief texture mapping is presented. A solution to finding the correct texture coordinates for all OFF objects has been developed by Bangay [2003]. Displacement maps and colour images are used to represent relief texture maps. The pre-warping equations implemented are identical to
those developed by Oliveira [2000]. A linear interpolation method has been implemented for image reconstruction, which can also be achieved by sampling at a higher resolution. The method for capturing overflow is based on the technique described by Oliveira [2000], but attaches polygons to extend the view plane rather than attaching polygons perpendicular to the view plane. This method has a restricted viewing angle dependant on the displacement values of the relief texture being used. Three optimizations have been developed. The back-face culling method is based on the standard OpenGL back-face culling method, but has been extended to determine visible polygons during the pre-warp phase. Texels with zero displacement need not be pre-warped and can thus be omitted. A tolerance value is used to adjust the level of speed and realism in a scene. Chapter six details and discusses the results achieved using this system.
Chapter 6

RESULTS AND ANALYSIS

6.1 Method
Objects were rendered from different viewpoints by applying random rotations. Two hundred samples were rendered for each experiment and the frame rates were recorded and averaged. These average rates form the basis for all comparisons and deductions made. The aim of the experiments is to evaluate how different methods affect the frame rate and realism of a scene.

6.2 Pre-warping and Image Reconstruction
The same relief texture (Appendix One, 1.1 Texture 1) is used to illustrate the results achieved during the pre-warping and image reconstruction phase. This is done to better illustrate the effects of the various techniques. For pre-warping and hole-filling a displacement value of -0.1 was used. The value was inverted to 0.1 for the fold over solution.

6.2.1 Standard Texture Mapping
An average frame rate of 80.6500fps was recorded. This is far above the required interactive rate of 10fps, despite the relatively high standard deviation of 2.6536. Although interactive rates were achieved, it was evident from certain angles that the object was 2D (Figure 6-1).
Figure 6-1. Standard texture mapping. (a) Final image from new view point after the texture mapping process illustrating lack of parallax. (b) The same image viewed from a different position.

6.2.2 Pre-Warping

Relief texture mapping was tested with no image reconstruction. Pre-warping resulted in an average frame rate of 3.9540fps. This is a 95.0974% decrease in frame rate from standard texture mapping, and a 60.46% decrease from an interactive rate. Although the frame rate was reduced by a large amount, the realism of the scene was increased. Figure 6-2(a) shows the intermediate image created by the pre-warp phase. Figure 6-2(b) shows the final image produced, after pre-warping and texture mapping, displaying correct parallax changes. Figure 6-2 shows results from a different view point. The black artefacts are holes created during the pre-warping phase.

Figure 6-2. The pre-warp process. (a) Intermediate warped image. (c) Final image after the texture mapping process using (b) as input. (c) and (d) illustrate the same process from a different view point.

6.2.3 Image Reconstruction
6.2.3.1 Linear Interpolation

This process reduces the average frame rate achieved using only relief texture mapping by 12.04% to 3.4783fps. The reduction in frame rate is countered by an increase in image quality. Figure 6-3(a) and (b) show the intermediate and final images produced using the hole-filling technique. Artefacts created during the pre-warp stage, visible in Figure 6-2, have been largely eliminated. Figure 6-3(c) and (d) illustrate an alternative view point, which has larger holes because an x and y rotation has occurred.

![Figure 6-3. Hole-filling. (a) Intermediate warped image with holes filled. (b) Final image after the texture mapping process using (b) as input. (c) and (d) illustrate the same process from a different view point. Note the larger holes.](image)

6.2.3.2 Sample Resolution

Images were sampled at double the original resolution. The average frame rate was reduced by 74.01% to 0.9922fps from the rate achieved using relief texture mapping with no image reconstruction. As with the hole-filling process, the slow frame rates achieved are countered by increased image quality (Figure 6-4). Sampling at triple resolution reduced the frame rate by 87.88% to 0.4752fps. The final image quality was further improved (Figure 6-5).
Figure 6-4. Double Resolution. (a) Intermediate warped image with reduced holes. (b) Final image after the texture mapping process using (b) as input. (c) and (d) illustrate the same process from a different view point.
6.2.3.3 Over flow

An image illustrating the artefacts created by over flow can be seen in Figure 6-6(a). Figure 6-6(b) shows the object viewed from the same point, but with the over flow solution. An average frame rate of 3.6005fps was achieved when only a single polygon was required to catch over flow. The rate decreased slightly to 3.5539fps when a further two polygons were required (Figure 6-6(c) and (d)). An overall frame rate of 3.5772fps was recorded, which is a 9.53% reduction from the rate achieved using relief texture mapping with no image reconstruction.
caught and attached at the top of the image using an extra polygon. (c) and (d) show the same process using three extra polygons.

6.2.3 Discussion

Table 6-1 and Figure 6-7 summarize the frame rates recorded during the various phases of pre-warping and image reconstruction and compare them to the interactive rate of a constant 10fps. These results have shown that it is possible to render objects viewed from arbitrary points using the system implemented. Pre-warping adjusts the image to allow for parallax changes, hole-filling and the over flow process are then used to reconstruct the image after the artefacts produced during the pre-warp stage. Each reconstruction method further reduces the average frame rate (Figure 6-7). The results recorded exhibit a low standard deviation. This is illustrated by the white bars in Figure 6-7. Low standard deviation is an indication of a constant rate, this is a requirement of an interactive rate of display.

![Figure 6-7](image)

Figure 6-7. Average frame rates achieved during pre-warping and image reconstruction. The white bars indicate standard deviation.
Relief Texture Mapping
Chapter 6 – Results and Analysis

<table>
<thead>
<tr>
<th>Method</th>
<th>Frame Rate</th>
<th>Percentage Reduction</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Texture Mapping</td>
<td>80.650</td>
<td>0</td>
<td>2.6536</td>
</tr>
<tr>
<td>Interactive rate</td>
<td>10.000</td>
<td>87.60</td>
<td>0.0000</td>
</tr>
<tr>
<td>Pre-warp</td>
<td>3.9540</td>
<td>60.46</td>
<td>0.0728</td>
</tr>
<tr>
<td>Over flow</td>
<td>3.5772</td>
<td>64.23</td>
<td>0.0966</td>
</tr>
<tr>
<td>Hole-filling</td>
<td>3.4783</td>
<td>65.22</td>
<td>0.0553</td>
</tr>
<tr>
<td>Resolution 2</td>
<td>0.9922</td>
<td>90.07</td>
<td>0.0093</td>
</tr>
<tr>
<td>Resolution 3</td>
<td>0.4752</td>
<td>95.25</td>
<td>0.0083</td>
</tr>
</tbody>
</table>

Table 6-1. Average frame rates achieved during pre-warping and image reconstruction.

High quality images have been produced but the frame rates achieved are not interactive ones. Figure 6-8 illustrates the gap between interactive frame rates and the best rate achieved during our experiments.

![Interactive Frame Rates](image)

Figure 6-8. Interactive frame rate compared to the frame rate achieved during relief texture mapping.
6.3 Rendering Multiple Relief Textures

All results have been recorded using a single relief texture. The results are almost linear. Adding multiple relief textures to a scene reduces the average frame rate in a linear manner (Figure 6-9). The rate is approximately halved when the number of textures rendered is doubled. Theoretically the results should be perfectly linear but the measurements are not since the rotations applied to the scene are selected randomly.

![Multiple Relief Textures](image)

**Figure 6-9.** The effect on frame rate when adding multiple relief textures to a scene.

6.4 Complexity proof

An object can be considered more complex if it has a greater frequency of change in displacement values than another object. Five objects represented by relief textures of varying complexities were tested (Appendix One, 1.1 Textures 1 to 5). Only the displacement maps affect the complexity of an object, the colour values used are irrelevant. The results of this experiment are summarized in Table 6-2.
Table 6-2. Average frame rates (200 samples) for relief textures of varying complexity.

<table>
<thead>
<tr>
<th>Texture</th>
<th>Frame Rate</th>
<th>Overall Average</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>3.9429</td>
<td>3.9549</td>
<td>0.0728</td>
</tr>
<tr>
<td>Two</td>
<td>3.9291</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Three</td>
<td>3.9620</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Four</td>
<td>3.9918</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Five</td>
<td>3.9443</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average frame rates achieved for each texture only vary by an average standard deviation of 0.0728fps. This slight variation can be attributed to the random rotations applied to the object. The viewpoint may affect speed of rendering. These results support the proposition by Oliveira [2000] that the complexity of an object does not affect the computation time of IBMR methods.

6.5 Optimizations

6.5.1 Back-Face Culling

A cube object was used to test the effectiveness of this optimization. The results obtained reflect the observation that the number of relief textures in a scene affects the frame rate in a linear manner. Since the number of relief textures directly affects the frame rate, the speedup achieved depends on the shape of the OFF object and the viewing position. A best speed up of almost 6 times and a worst speed of about 2 times were recorded. These results are summarized in Table 6-3.

Table 6-3. Results of back-face culling optimization tested on a cube object.

<table>
<thead>
<tr>
<th>Visible Faces</th>
<th>Frame Rate</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (no optimization)</td>
<td>0.6734</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1.3482</td>
<td>2.0021</td>
</tr>
<tr>
<td>2</td>
<td>2.0001</td>
<td>2.9702</td>
</tr>
<tr>
<td>1</td>
<td>3.9540</td>
<td>5.8717</td>
</tr>
</tbody>
</table>

This method does not affect the speed of the pre-warp process and thus will only assist when relief textures in a scene are not visible.
6.5.2 Zero Displacement Clipping

Five textures, each containing a different number of texels with zero displacement, were used to test this optimization. The textures can be seen in Appendix One, 1.1 Textures 1,3,4,5 and 6. The results of the experiment are summarized in Table 6-4 and Figure 6-10.

![Zero Displacement Optimization](image)

**Figure 6-10. Results of the zero displacement optimization function.**

<table>
<thead>
<tr>
<th>Texture</th>
<th>Texels with Zero Displacement</th>
<th>Frame Rate Without Optimization</th>
<th>Frame Rate With Optimization</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>59</td>
<td>3.9620</td>
<td>3.9636</td>
<td>1.0004</td>
</tr>
<tr>
<td>5</td>
<td>662</td>
<td>3.9918</td>
<td>4.0111</td>
<td>1.0050</td>
</tr>
<tr>
<td>3</td>
<td>13829</td>
<td>3.9291</td>
<td>4.2386</td>
<td>1.0788</td>
</tr>
<tr>
<td>1</td>
<td>18565</td>
<td>3.9429</td>
<td>4.3223</td>
<td>1.0962</td>
</tr>
<tr>
<td>6</td>
<td>42256</td>
<td>3.9443</td>
<td>4.9627</td>
<td>1.2422</td>
</tr>
</tbody>
</table>

**Table 6-4. Comparison of frame rates with and without the zero displacement optimization.**
As the number of texels with a displacement value of zero increases, so does the frame rate. The speedup is dependant on the texture being used. This method directly affects the speed of pre-warping an image and thus assists in achieving our goal of an interactive rate.

6.5.3 Tolerance Levels

A tolerance of one is the default setting in the system. Using this setting, an image is pre-warped for every frame that is rendered. The system was tested using tolerance levels of one up to five (Figure 6-11 and Table 6-5). Three relief textures were tested for each level of tolerance and the results were averaged.

![Pre-warp Tolerance Levels](image)

**Figure 6-11.** Average frame rates of the relief texture mapping system using different tolerance levels.

<table>
<thead>
<tr>
<th>Tolerance Level</th>
<th>Frame Rate</th>
<th>Standard Deviation</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>57</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Increasing the tolerance level from one to two causes a very large speedup. Subsequent increases in tolerance level do not produce such a dramatic increase in frame rate, but standard deviation does decrease. The levels of standard deviation are shown in Figure 6-11 as white lines on every bar of the graph. All the levels of tolerance exhibit a high standard deviation because of the large gap in frame rates between standard texture mapping and relief texture mapping. Although the average frame rate has been greatly increased, the actual relief texture mapping process has not been increased at all. This causes the visual output to be jerky. Higher levels of tolerance produce smoother output but realism is reduced further. By calculating pre-warp equations every second frame we increased the average frame rate to a level way above that required for interactivity. The method is unsuccessful however, since the rate is not constant.

6.5.4 Discussion

Back-face culling provided a method of hidden surface removal. Despite providing a speedup for certain objects and view points, the method did not increase the speed of the pre-warp equations. By not pre-warping texels with zero displacement the frame rate of the relief texture mapping process was increased from an average of 3.9540fps to 4.2997fps. This optimization affects the speed of pre-warping and thus reduces the gap between the current frame rate and an interactive rate. Adjusting tolerance levels improved the average frame rate dramatically, but the method suffers from large standard deviations in frame rate. This technique does not allow interactivity but provides a method for finding a compromise between speed and reality. Figure 6-12 shows how the various speedups compare to the target of a constant 10fps.
6.6 Limitations
The methods of filling holes that have been implemented are incomplete solutions. In various cases the holes are not entirely filled. Linear interpolation cannot correctly add colour to areas where a sharp displacement discontinuity is associated with a colour change [Oliveira, 2000]. The relief textures cannot be used to represent layered objects. These errors could be solved by implementing an approach which uses LDIs, as described by Parilov [2002]. Extending the view plane by attaching additional polygons still restricts the viewing angle. Oliveira [2000] and Parilov [2002] suggest methods for implementing a complete solution.

6.7 Discussion
The results obtained reveal that pre-warping is the most expensive function of the relief texture mapping process. This suggests that optimization efforts should focus on this.
function. Image reconstruction further reduces the average frame rate but is not of major concern. Of the three optimizations introduced, only the zero displacement clipping method affected the actual pre-warp computation time, but not by enough to allow for interactive rates of display. Back-face culling is useful when multiple relief textures are used in a scene. Tolerance levels can be used to find a compromise between speed of computation and the level of realism in a scene. Appendix One (1.2) provides a collection of example images rendered using the system. Chapter seven will review the work carried out during this project and form some conclusions based on the results obtained.
Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Review
This project investigated the possibility of producing photo-realistic images at interactive frame rates. A relief texture mapping system was implemented to analyse this. An introduction to IBMR techniques and a summary of the major work carried out in this field was provided. The relief texture mapping process is discussed. A detailed description of the relief texture mapping system developed during this project has been provided. The results obtained from this system were summarized and analysed.

7.2 Findings
IBMR techniques exhibit two advantages over conventional geometric techniques. Since real world images can be used as input these methods can generate photo-realistic images. The complexity of an object does not affect the computation time of the IBMR methods. Increasingly complex scenes can be represented without reducing the speed of the system. This was proved by testing textures of varying complexity. The pre-warp phase of the relief texture mapping process is computationally the most costly. Optimizations should therefore focus on this function. Only zero displacement optimization reduced the time required to pre-warp an image. Adding relief textures to a scene affects the frame rate in a linear manner.

Oliveira et al., [2000] achieved an average frame rate of 9.42fps, whereas this system achieved an average rate of 3.9540fps. Funkhouser and Séquin [1993] define an interactive rate as being a constant 10fps. The rate recorded during this project is only 40% of this. However, our rate could be increased with further optimizations.

7.3 Extensions and Future Research
Better and more complete methods of hole-filling exist and need to be investigated in order to make this system complete. The over flow solution currently used in our system suffers from a restricted viewing angle. This can be improved using the
technique described by Oliveira [2000]. This would also allow for complex 3D objects to be represented. Oliveira [2000] and Fujita and Kanai [2002] developed methods of displaying correct lighting and shadows on relief textures. Normal maps and shadow maps can be implemented to achieve this and thus further enhance the realism of a scene. Policarpo [2002] developed a method for extracting relief textures from 3D objects. A system such as this allows for the easy creation of relief textures. There are numerous optimizations that could be implemented, such as the clipping algorithm described by Parilov [2002]. Fujita and Kanai [2002] present a hardware implementation of a relief texture mapping system. Performing the pre-warp operations in hardware would greatly increase the speed of the system. Texture pyramids allow the computational cost of a relief texture to be proportional to its size on the screen, eliminating unnecessary samples and assisting with anti-aliasing.
LIST OF REFERENCES


[Debevec et al., 98] Debevec, P., Yu, Y., Borshukov, G. “Efficient View-Dependent Image-Based Rendering with Projective...
List of References


List of References


List of References


Appendix One

RELIEF TEXTURES AND SAMPLE IMAGES

a. Relief Textures

These relief textures are the images used to conduct experiments with during the course of this project. The colour image is shown on the left and the associated displacement map is shown on the right.

Texture 1 [Oliveira, 2000]

Texture 2

Texture 3

Texture 4 [Policarpo, 2002]
Relief Texture Mapping
Appendix One – Relief Textures and Sample Images

- Texture 5 [Oliveira, 2000]
- Texture 6 [Fujita and Kanai, 2002]
- Texture 7 [Policarpo, 2002]
- Texture 8 [Policarpo, 2002]
b. Sample Rendered Images

These images were created using the implemented system and the above relief textures. They are meant to illustrate the capabilities of the system. Note that they were not rendered at an interactive frame rate as image quality was the focus. The images on the left hand side are flat textures. The images on the right hand side are the results of the relief texture mapping process. Any artefacts present are a result of the incomplete hole-filling algorithm used.

Texture 5

Texture 7
Texture 7. Greater displacement values used.

Texture 8.
Relief Texture Mapping
Appendix One – Relief Textures and Sample Images
Appendix Two

EQUATIONS AND EXAMPLE CODE

2.1 Texture Coordinates

Texture maps have a coordinate system, given by (s, t), where s is the horizontal axis and t is the vertical axis. Triangles are described by three coordinates: O, A and B. Each triangle has texture coordinates represented by the values (os, ot), (as, at) and (bs, bt).

The problem arises since the points defining the coordinate system of the corresponding triangle do not have convenient texture coordinates of O= (0,0), A=(1,0) and B=(0,1). Instead the coordinates are at O=(os,ot), A=(as,at) and B=(bs,bt). The vector AO, and BO can define a coordinate system (u,v) both in 3D space and in texture space. The coordinates (u,v) are used in the calculation of the relief textures, so we need to reorient the coordinate systems in order to calculate these coordinates. The (u, v) coordinate for any corresponding (s, t) coordinate can be calculated and thus we can solve this artefact for any triangle.

(s, t) are given by:

\[
\begin{align*}
    s &= os + u (as - os) + v (bs - os) \\
    t &= ot + u (at - ot) + v (bt - ot)
\end{align*}
\]

The position of O, A and B in the (s, t) texture space is known and so we can solve for u and v. This is inverted to find (u, v):

\[
\begin{align*}
    (bt - ot) s &= (bt - ot) os + u (as - os) (bt - ot) + v (bs - os) (bt - ot) \\
    (bs - os) t &= (bs - os) ot + u (at - ot) (bs - os) + v (bt - ot) (bs - os)
\end{align*}
\]

u is given by:

\[
\begin{align*}
    u &= [(bt - ot) s - (bt - ot) os] - [(bs - os) t - (bs - os) ot] = u [(as - os) (bt - ot) - (at - ot) (bs - os)]
\end{align*}
\]
unless:

$$\frac{(as - os) (bt - ot) - (at - ot) (bs - os)}{(as - os) (bt - ot) - (at - ot) (bs - os)} = 0$$

or

$$\frac{(as - os)}{(at - ot)} = \frac{(bs - os)}{(bt - ot)}$$

In this case the two texture vectors from the same point are parallel.

Then:

$$s - (os + u (as - os)) / (bs - os) = v$$

or

$$t - (ot + u (at - ot)) / (bt - ot) = v$$

This will be fine unless the BO vector is zero, so alternatively, and finally we have:

$$(at - ot) s = (at - ot) os + u (as - os) (at - ot) + v (bs - os) (at - ot)$$
$$(as - os) t = (as - os) ot + u (at - ot) (as - os) + v (bt - ot) (as - os)$$

giving:

$$(at - ot) (s - os) - (as - os) (t - ot) = v [(bs - os) (at - ot) - (bt - ot) (as - os)]$$
2.2 This code is provided to illustrate the implementation of various methods used in this project. It has been simplified for this purpose and is meant more as an explanation of how functions were implemented than as actual code.

2.2.1 Pre-Warp Function
/*Performs the pre-warp equations for an image. Speedups omitted for simplicity*/

//values used to calculate correct texture coordinates
double denu = ((as-os)*(bt-ot))-((at-ot)*(bs-os));
if (denu == 0.0) {
    //flat texture coordinates
    return;
}
double denv = ((bs-os)*(at-ot))-((bt-ot)*(as-os));
if (denv == 0.0) {
    //flat texture coordinates
    return;
}

//pre-warp for each texel in the image
for (int y=0; y<ysize; y++) {
    for (int x=0; x<xsize; x++) {
        //s,t,u and v coordinates
        double s = ((double)x)/((double)(xsize-1));
        double t = ((double)y)/((double)(ysize-1));
        double u = (((s-os)*(bt-ot))-((t-ot)*(bs-os)))/denu;
        double v = (((s-os)*(at-ot))-((t-ot)*(as-os)))/denv;
        //displacement value
```
int disp = d->getHeight(x,y);
double fdisp = ((double)disp)*(-0.4/256.0);
double den = lookupden[disp];

//pre-warp equations
double ui = (u + lookupu[disp])*den;
double vi = (v + lookupv[disp])*den;
double si = os + (ui*(as-os))+(vi*(bs-os));
double ti = ot + (ui*(at-ot))+(vi*(bt-ot));

//destination x and y values
int xi = (int)(si*xsize);
int yi = (int)(ti*ysize);

//if the destination texel is in the viewplane
if ((xi>=0) && (xi<xsize) && (yi>=0) && (yi<ysize)) {
    //destination buffer index value
    indi = (xi+(xsize*yi));
    //source buffer index value
    inds = (x+(xsize*y))*3;
    if (zi>zbuffer[indi]) {
        colours[indi*3+0] = c->colours[inds+0];
        colours[indi*3+1] = c->colours[inds+1];
        colours[indi*3+2] = c->colours[inds+2];
        zbuffer[indi] = zi;
    }
}
```

### 2.2.2 Linear Interpolation Function

/*draws a line between two texels in 2D, interpolating colour and height values*/
for (int i = 1; i < numPixels; i++) {
    tmpi = i;
    yMove = (int)(collectiveInc);
    if (xDiff >= yDiff) {
        //if we need to move left
        if (nextX < prevX)
            tmpi = 0 - i;
    }
}
//if we need to move down
if (nextY < prevY) {
    yMove = 0 - yMove;
    //new index position of the image buffer
    pos = tmpi + (yMove * xsize);
}
else  //(yDiff > xDiff) {
    if (nextY < prevY)
        tmpi = 0 - i;
    if (nextX < prevX)
        yMove = 0 - yMove;
    pos = yMove + (tmpi * xsize);
}

//ensure correct visibility
if ((prevHeight + incHeight) > zbuffer[prevIndi + pos]) {
    colours[(prevIndi+pos)*3+0]=colours[prevIndi*3+0]+incRed;
    colours[(prevIndi+pos)*3+1]=colours[prevIndi*3+1]+incGreen;
    colours[(prevIndi+pos)*3+2]=colours[prevIndi*3+2]+incBlue;
    zbuffer[prevIndi+pos]=prevHeight+incHeight;
}

//value used to move along the line
collectiveInc += inc;

//value used to interpolate height values
incHeight += heightDiff;

//value used to interpolate the red value of RGB
incRed += colDiffRed;

//value used to interpolate the green value of RGB
incGreen += colDiffGreen;

//value used to interpolate the blue value of RGB
incBlue += colDiffBlue;