

# **Displacement Mapping and Volume Rendering Graphics Hardware**

by  
**Michael C. Doggett**

ISBN: 1-58112-166-0

**DISSERTATION.COM**



USA • 2002

*Displacement Mapping and Volume Rendering Graphics Hardware*

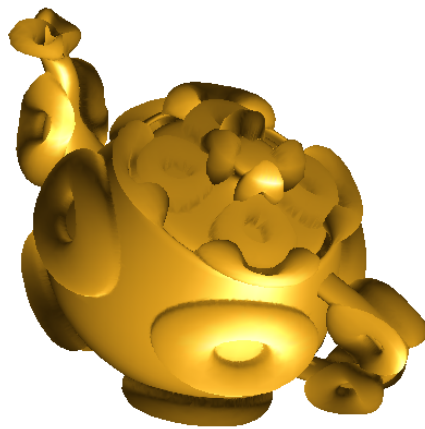
Copyright © 2001 Michael C. Doggett  
All rights reserved.

Dissertation.com  
USA • 2002

ISBN: 1-58112-166-0

[www.dissertation.com/library/1121660a.htm](http://www.dissertation.com/library/1121660a.htm)

**Displacement Mapping  
and  
Volume Rendering  
Graphics Hardware**



Michael Doggett  
Tübingen  
2001

*The work documented in this dissertation was done at the G<sub>R</sub>aphisch-Interaktive Systeme (Computer Graphics Laboratory) in the Wilhelm Schickard Institut für Informatik, Universität Tübingen from March 1998 to January 2001 towards the German degree of Habilitation under the supervision of Prof. Dr.-Ing Wolfgang Straßer.*



# Abstract

This dissertation introduces new hardware architectures for more realistic surface rendering of three dimensional objects and the rendering of volumetric datasets. Surface rendering is dealt with in the first part of the dissertation where the architectures for displacement map rendering in hardware are proposed. This work represents the first to appear in scientific literature on displacement map hardware rendering. Where possible these architectures propose components that integrate into currently available pipelines and make use of existing units in those pipelines. Displacement map rendering in hardware is a desired feature currently under development by most graphics hardware vendors. The first architecture is scan-line based and works just before rasterization and the second adaptively retessellates a triangle mesh using additional hardware on either side of the geometry transformation stage in the graphics pipeline.

The VIZARDII architecture and several hardware based performance improvements for any ray casting architecture are presented in the second part titled Volume Rendering. VIZARDII is an interactive programmable hardware accelerator for Volume Rendering implemented on a PCI Card. The main pipeline is implemented on a Xilinx FPGA allowing new features to be added relatively quickly. A memory interface is presented and discussed with its final implementation appearing in the VIZARDII system. Novel architectures for ray queuing and sorting, sub-cube based space leaping are also presented which improve the performance of ray casting based hardware architectures. Antialiasing that occurs when ray casting volume data is also discussed and possible solutions are presented using multiresolution volume datasets.



# Acknowledgements

Thanks to Professor Strasser for inviting me to Tübingen to further my research in Volume Rendering hardware with the GRIS research lab. I am also thank him for his wisdom and guidance during my time here. Thanks to Andreas Schilling for his brilliant insights into my work, willingness to help and associated suggestions.

Thanks to Johannes Hirche, Anders Kugler, Michael Meißner, Urs Kanus, Dirk Bartz, Gregor Wetekam, Olaf Eitzmuß, Rainer Jäger, Helga Mayer, Edelhard Becker and all the members of the GRIS lab that have made my time at GRIS both rewarding and enjoyable.

Thanks to Roland Proksa, and our partners at Philips Research Laboratories in Hamburg for our work together on the VIZARDII and DynCT projects.

Parts of this work were funded by the Commission of the European Communities (CEC) and the SFB grant 382 of the German Research Council (DFG).

Thanks to Gaby, Iliana and Cassandra for always being there for me.





# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Overview . . . . .	3
<b>2</b>	<b>Graphics Hardware</b>	<b>5</b>
2.1	Surface Rendering . . . . .	6
2.1.1	Texture and Bump Mapping . . . . .	6
2.1.2	Displacement Mapping . . . . .	8
2.1.3	Higher Order Primitive Rendering . . . . .	11
2.2	Volume Rendering . . . . .	12
<b>I</b>	<b>Displacement Mapping</b>	<b>17</b>
<b>3</b>	<b>A Scan Conversion Hardware Architecture</b>	<b>19</b>
3.1	Scan Conversion Displacement Mapping Algorithm . . . . .	21
3.1.1	Scan Conversion Algorithm Overview . . . . .	24
3.1.2	Recomputing Normal Vectors . . . . .	28
3.1.3	Generating Bump Maps from Height Fields . . . . .	29
3.1.4	Bilinear Interpolation of Vertex Coordinates . . . . .	30
3.2	Hardware Architecture . . . . .	31
3.2.1	Displacement Mapping Architecture . . . . .	32
3.2.2	A Cheaper Architecture . . . . .	35
3.2.3	Dual Pipelines with a Displacement Mapping Unit . . . . .	37
3.2.4	Performance . . . . .	37
3.3	Results . . . . .	37
3.4	Summary . . . . .	38

<b>4</b>	<b>Adaptive Displacement Mapping using Tessellation</b>	<b>41</b>
4.1	Adaptive View Dependent Tessellation . . . . .	42
4.1.1	Tessellation based only on edge information . . . . .	42
4.1.2	Surface Normal Variance Test . . . . .	44
4.1.3	Local Area Average Height Test . . . . .	44
4.1.4	View Dependent Resampling . . . . .	46
4.1.5	Refinement Limit Test . . . . .	47
4.1.6	Tessellation . . . . .	48
4.1.7	Curved Surface rendering using Displacement Maps . . . . .	48
4.2	Hardware Architecture . . . . .	49
4.2.1	Tessellation Tests Architecture . . . . .	50
4.3	Results . . . . .	53
4.4	Summary . . . . .	58
<b>II</b>	<b>Volume Rendering</b>	<b>61</b>
<b>5</b>	<b>An Efficient Low-Cost Memory Architecture for Interactive Ray Casting</b>	<b>63</b>
5.1	Memory Access . . . . .	64
5.1.1	SDRAM terminology . . . . .	65
5.1.2	Cubic Addressing Scheme . . . . .	67
5.1.3	Parallel Memory Access . . . . .	68
5.1.4	Memory Access Buffering . . . . .	69
5.2	Memory Architectures . . . . .	69
5.2.1	Sixty Four Memories . . . . .	70
5.2.2	Eight Logical Memories . . . . .	70
5.2.3	Four DIMMs . . . . .	71
5.3	Results . . . . .	72
5.4	Summary . . . . .	74
<b>6</b>	<b>Ray Queueing and Sorting for Ray Casting Hardware</b>	<b>81</b>
6.1	Multiple Rays . . . . .	82
6.2	Overtaking FIFOs . . . . .	85
6.3	Compositing Buffer . . . . .	86
6.4	Results . . . . .	87
6.5	Summary . . . . .	87

---

<b>7</b>	<b>Sub-cube based Space Leaping for Ray Casting Hardware</b>	<b>89</b>
7.1	Space leaping algorithm . . . . .	90
7.1.1	Space Leaping Algorithm . . . . .	90
7.2	Hardware implications . . . . .	92
7.2.1	Space Leaping Hardware . . . . .	92
7.2.2	Multiple Rays . . . . .	94
7.2.3	Compositing Latency . . . . .	95
7.3	Results . . . . .	96
7.3.1	Dataset characteristics . . . . .	97
7.3.2	Experiments and Discussion . . . . .	98
7.4	Summary . . . . .	100
<b>8</b>	<b>Ray Casting Antialiasing using Multiresolution Datasets</b>	<b>105</b>
8.1	Multiresolution Datasets . . . . .	106
8.1.1	Calculating lower resolutions . . . . .	106
8.1.2	Level switching . . . . .	107
8.2	Results . . . . .	110
8.2.1	Lower resolution datasets . . . . .	111
8.2.2	Sample frequency matching . . . . .	111
8.3	Summary . . . . .	112
<b>9</b>	<b>VIZARDII: A Programmable and Interactive Volume Rendering PCI Card</b>	<b>115</b>
9.1	The Ray Processing Unit . . . . .	117
9.1.1	Ray-caster . . . . .	118
9.1.2	Address Calculation . . . . .	119
9.1.3	Interpolation . . . . .	120
9.1.4	Illumination . . . . .	120
9.1.5	Classification . . . . .	121
9.1.6	Combiner . . . . .	121
9.1.7	Compositing . . . . .	121
9.2	Complex Gradient Filters . . . . .	122
9.3	Voxel Memory . . . . .	123
9.3.1	Sub-cube Memory Organization . . . . .	123
9.4	VIZARDII PCI Card . . . . .	124
9.4.1	DSP . . . . .	125
9.4.2	Xilinx Virtex FPGA . . . . .	125
9.4.3	Control Unit . . . . .	126

9.4.4	Logic Usage and Layout	126
9.4.5	Rendering Performance	127
9.5	Summary	127

# List of Figures

2.1	A cross section of a displaced surface. . . . .	9
2.2	A plane displaced with a text displacement map. . . . .	9
2.3	The viewer, viewplane and Volume dataset as used in Ray Casting. . . . .	13
3.1	The pixel, its maximally displaced position and its new position on the displaced surface. . . . .	21
3.2	The two new triangles introduced at each pixel to cover pixels not rendered. . . . .	23
3.3	Remeshing points are snapped to the new edge remesh points which ensure matching sampling points along triangle edges. . . . .	27
3.4	A close up of (a) a grey scale and (b) an analytically computed bump map. . . . .	30
3.5	The calculation of the fractional values at the pixel. . . . .	31
3.6	Displacement mapping pipeline using bilinear interpolation and a dual pipeline. . . . .	34
3.7	A cheaper displacement mapping triangle raster pipeline. . . . .	36
3.8	A sphere with (a) bump mapping only and (b) bump mapping and displacement mapping. . . . .	38
3.9	A close up of the sphere with (a) bump mapping only and (b) bump mapping and displacement mapping. . . . .	39
3.10	Several examples of a displacement mapped sphere using only two displacements maps but with varying parameters. . . . .	40
4.1	A triangle from the base surface is recursively tessellated to create a displaced surface. . . . .	43
4.2	Using the texture coordinates of the vertices to calculate the summed height. . . . .	45
4.3	Tessellation edge tests. In (a) only the Summed Height Test succeeds. In (b) only the Normal Test succeeds. . . . .	46

4.4	The three triangulations used for tessellation (a,b,c). (d) Two other possibilities for splitting two edges. . . . .	48
4.5	A graphics pipeline with new units added to perform displacement map rendering. . . . .	50
4.6	Architecture for the Tessellation Tests unit. . . . .	51
4.7	(a) Normal Test only. (b) Summed Height Test only. (c) Both tests. . . . .	55
4.8	Volker's head dataset. (a), (b) and (c) show distance from view plane, (d), (e) and (f) are mesh close-ups for corresponding images.	56
4.9	The Crater Lake . . . . .	57
4.10	(a) A half donut displaced teapot (b) The half donut with texture and more surface detail added using another displacement map. . .	57
5.1	Simplified SDRAM State Diagram for continuous reads with burst length 1 and absolute values based on NEC SDRAM. . . . .	75
5.2	The positioning of voxels in eight parallel memory banks and two sample points along a ray. . . . .	76
5.3	The FIFO buffers for addresses and voxel values. . . . .	77
5.4	Memory access across page boundaries with 64 SDRAMs. Interpolation neighborhoods consist of $2 \times 2$ squares, the numbers denote SDRAM delivering the voxel. . . . .	78
5.5	Memory architecture using 8 logical memories. . . . .	78
5.6	Memory architecture using 4 DIMMs. . . . .	79
5.7	The addressing values for voxels in the four DIMMs memory architecture. Address value $M_n$ is sent to DIMM $(n \bmod 4)$ . . . . .	79
5.8	(a) A CT scan of a human food. (b) A CT angiography scan of human brain vessels. (c) The foot from a statue casting. . . . .	80
6.1	Four rays and associated voxels showing relevant memories. . . . .	82
6.2	A group of four rays showing the sample points along each ray (a), sample processing order when each ray is processed individually (b) and when tracing four interleaved rays (c). . . . .	83
6.3	The calculation stages in a ray casting pipeline. . . . .	84
6.4	The architecture of the Overtaking FIFO. . . . .	86
7.1	The Space Leap Volume for the 2D case. . . . .	91
7.2	Calculation of the skipping . . . . .	91
7.3	Pipeline for the skip calculator. . . . .	93

---

7.4	Four rays and associated voxels showing relevant memories. . . .	95
7.5	The stages in a ray casting pipeline. (a) Standard pipeline stages, (b) Extra units required for multiple rays and improved memory performance, (c) Further improving memory performance by sorting rays. . . . .	101
7.6	Test datasets: Rendered applying the associated transfer functions.	102
7.7	Percentage of skipable sub-cubes. Only empty voxels have been exploited. . . . .	103
7.8	Percentage of skipable sub-cubes exploiting the given classification.	103
7.9	Cycles per frame, with classification applied. . . . .	104
8.1	Rays missing a voxel when there are 3 voxels between them. . . .	108
8.2	The distances used to calculate when to change resolutions in the multiresolution dataset. . . . .	109
8.3	A voxel cube rendered without (a), (b) and with multiresolution (c).	110
8.4	A CT dataset rendered (a) without and (b),(c) with multiresolution. (c) shows the second level in red and (d) is the difference between images (a) and (b). . . . .	113
8.5	A Confocal Microscopy dataset rendered using matched sampling frequencies. (a) original, (b) level 0 grey and level 1 red (c) level 1 red and level 2 green, (d) level 2 green and level 3 blue. . . . .	114
9.1	VIZARDII from below (a) and above (b). . . . .	116
9.2	Architecture of the Ray Processing Unit (RPU). . . . .	118
9.3	The interleaving of the entries of one face of the cube-map to ensure four values are read in one cycle. . . . .	121
9.4	Architecture of the PCI Card. The main data and address bus is shown in red and the optional DSP and DIMM are shown using dashed lines. . . . .	124
9.5	Architecture of the Xilinx Virtex FPGA. The data download buses are indicated in Green. . . . .	125
9.6	Colored Floorplan of the Virtex FPGA. . . . .	127
9.7	Images generated by VIZARDII showing (a) a ventricle fly-through, (b) a lobster, (c) an aneurism and (d) a transparent engine block. .	128





# List of Tables

3.1	Coordinate System Terminology. . . . .	20
3.2	Algorithm notation and rasterisation pipeline variables. . . . .	22
5.1	Characteristics for NEC SDRAM . . . . .	66
5.2	Memory timing for three different datasets. . . . .	73
7.1	Set of datasets which have been used to evaluate the space leaping approach. . . . .	96
7.2	Storage needed for the space leap map. . . . .	98
7.3	Frame-rates for the five datasets when skipping empty voxels only ('0') and when exploiting the given classification (class). . . . .	99



# Chapter 1

## Introduction

While the amount of photorealism in real-time rendering systems has increased to an impressive level in recent years, the demand for even more realistic images has not subsided. To meet these demands the feature lists for currently available three dimensional graphics cards has grown to include techniques such as bump mapping. This dissertation presents novel work into the next step in fulfilling this ever increasing demand in the form of hardware based displacement map rendering.

Along with these demands for photorealism is the growing interest in real-time visualization of volume data. Several novel architectures to improve the performance of these visualization systems in general are presented along with the architecture of the VIZARDII system.

### 1.1 Overview

This dissertation starts with a brief overview of graphics hardware starting with the basic concepts used in surface rendering including displacement mapping in Chapter 2. Chapter 2 also presents the fundamental algorithms used for Volume Rendering using ray casting and some previous hardware architectures.

The dissertation is then broken into two parts. The first presents two approaches to displacement mapping in hardware. Chapter 3 presents a scan line based approach where triangles are processed in screen space and a new displaced mesh generated and rendered. This work was first presented in [18] and the final version on which the chapter will be published in a journal next year[19].

Chapter 4 presents a different approach to displacement mapping hardware

that takes advantage of geometry engines which are becoming common in PC based graphics hardware. This work was first presented in sketch form[17]with a final full paper on which the chapter is based appearing in[16]. Further work following on from the adaptive architecture for rendering of subdivision surfaces is presented in [2].

The second part of the dissertation deals with Volume Rendering and in particular ray casting hardware architectures. Chapter 5 presents a low-cost memory interface that takes advantage of SDRAM technology and removable Dual Inline Memory Modules (DIMMs). This work was first presented in [20] and the final version on which the chapter is based is presented in in [21]. To further improve the performance of ray casting based Volume Rendering a ray queueing and sorting design is described in Chapter 6 which is based on the work from [15]. Ray casting is can be further improved by avoiding the wasting of cycles on empty space and a design to achieve this improvement is presented in Chapter 7 based on work that will be published in [62].

Chapter 8 presents some of the problems encountered by ray casting algorithms in general that can appear as aliasing artifacts. Proposals to solve this problem are presented in the chapter which is based on work from [22].

The VIZARDII interactive volume rendering system is presented in Chapter 9. This ray casting, image-order based hardware accelerator employs the memory interface from Chapter 5 and the ray queueing from Chapter 6 into a fully featured PCI Card for interactive Volume Rendering.

# Chapter 2

## Graphics Hardware

Graphics Hardware includes the algorithms, systems, electronic computational and storage devices that are responsible for digital image synthesis on modern computer systems. The design that places many digital devices, from the system level down to the gate level elaborating a hierarchy of structure and order is often referred to as the *Architecture*. The objective of graphics hardware architectures over the last 15 years has been to find a careful balance between performance and features in what has become one of the most competitive markets in the computing industry today. It is this architectural design from top level issues such as parallelisation right down to low level issues of high performance arithmetic operations that determines whether the computer user can let go of their real world and become totally immersed inside the world generated inside the computer.

3D graphics architectures have focused on the rendering of triangles which are used to model objects in three dimensions. But three dimensional objects can be represented using a large range of representations. The basic representation of objects in three dimensional space used has determined many of the approaches to architectural design over the last 15 years, but now new methods of object representation and their sampling for image rendering are being investigated leading to a profound influence on the way graphics architectures will be designed in the future. Currently systems capable of rendering millions of flat shaded triangles per second are readily available resulting in a focus shift away from performance towards features and high quality rendering[36].

We can categorise graphics hardware architectures based on the fundamental type of geometric primitive they use to generate 2D images. This dissertation deals the two most common types of representation, surface and volume based systems. While surface rendering has traditionally been driven by triangle render-

ing pipelines, higher order primitives are of increasing importance and the rendering of these primitives in hardware is expected this year. Also with the advent of new concepts of the image synthesise process completely new approaches to architectural design become possible marking the beginning of a whole new era in Graphics Hardware design and research.

## 2.1 Surface Rendering

Standard rendering hardware contains a rasteriser that scan converts triangles or polygons into pixels and then sends the pixels to the output display. Traditional shaded triangle scan conversion is typically performed by a pipeline of an edge-walking phase followed by the span interpolation. An edge processor decomposes triangles into horizontal spans. Spans are further decomposed into pixels by a span processor. Span interpolation forms the inner loop of the triangle shading pipeline; it interpolates the colour, depth, texture coordinate and normal vector along the current span. During edge interpolation, a triangle is scanned horizontally from top to bottom, delivering the boundaries of the triangle, the starting and ending values of texture coordinates,  $(u, v, w)$ , normal,  $(N_x, N_y, N_z)$ , colour  $(R, G, B)$ , and depth,  $Z$ , for the span interpolation. The span processor generates one pixel per cycle in the  $X$ -direction. Alternatives to scan-line rasterisation such as stamp based rasterisation are also used for rasterization[61, 60].

### 2.1.1 Texture and Bump Mapping

Texture and bump mapping are rendering techniques for adding more surface detail to computer-generated objects. Today's standard for texture filtering is mipmapping[89], where textures are stored according to their level of detail.

Bump mapping, introduced by Blinn[5], perturbs surface normals on a surface as if a height field displaced the surface in the direction of the original surface normal. Bump mapping does not change the underlying geometry of the model, but fools the shading to produce an interesting surface. Since bump mapping only changes the appearance of an object, it makes certain approximations. Standard techniques for bump mapping assume that the bumpiness is only microdisplacements and hence assume the magnitude of the height field is negligible. Blinn[5] presented two methods for computing bump mapping the first used Offset Vector bump maps and the second Vector Rotation bump maps.

Bump mapping is applied to a surface by taking the surface normal vector  $\hat{\mathbf{N}}$  at a point  $\mathbf{P}$  which is perturbed by a perturbation vector  $\mathbf{B}$  dependent on a perturbation function  $\mathbf{F}(u, v)$  of the surface parameters, stored as a two-dimensional table indexed by the texture coordinate  $(u, v)$ . For a point  $\mathbf{P}$  on a surface  $\mathbf{S}(u, v)$ , the normal vector  $\mathbf{N}$  at that point is expressed as:

$$\mathbf{N}_p = \mathbf{S}_u \times \mathbf{S}_v = \frac{\partial \mathbf{S}}{\partial u} \times \frac{\partial \mathbf{S}}{\partial v} \quad (2.1)$$

where  $\mathbf{S}_u$  and  $\mathbf{S}_v$  are the partial derivatives in the parameter directions  $u, v$ . The new normal to the perturbed surface is given by:

$$\mathbf{N}' = \mathbf{N} + \underbrace{\frac{\mathbf{F}_u(\mathbf{N} \times \mathbf{S}_v)}{|\mathbf{N}|} + \frac{\mathbf{F}_v(\mathbf{N} \times \mathbf{S}_u)}{|\mathbf{N}|}}_{\mathbf{B}'} \quad (2.2)$$

Since bump mapping modifies the original normal vectors it requires the normal vector to be interpolated across the current triangle. Different types of hardware architectures were described in the past for texture and bump mapping support [25, 71, 50].

Bump mapping requires interpolating the Cartesian coordinates of surface normals and applying the perturbation in a local coordinate system  $\mathbf{E}_u, \mathbf{E}_v, \mathbf{E}_w$  tangent to the surface, defined by the normalised surface normal vector  $\hat{\mathbf{N}}$  and two vectors perpendicular to  $\hat{\mathbf{N}}$ . A local and orthonormal coordinate system can be built from the interpolated surface normal  $\mathbf{N}$  and a constant main direction  $\mathbf{M}$ , such as the polar axis, as shown by Schilling [79] :

$$\mathbf{E}_w = \frac{\mathbf{N}}{\|\mathbf{N}\|} = \hat{\mathbf{N}} \quad \mathbf{E}_u = \frac{\mathbf{M} \times \mathbf{N}}{\|\mathbf{M} \times \mathbf{N}\|} \quad \mathbf{E}_v = \mathbf{E}_w \times \mathbf{E}_u \quad (2.3)$$

$$\mathbf{A} = [\mathbf{E}_u \mathbf{E}_v \mathbf{E}_w]^T \quad (2.4)$$

and the new normal  $\mathbf{N}'$  to the surface becomes:

$$\mathbf{N}' = \mathbf{N} + \mathbf{A}\mathbf{B} \quad (2.5)$$

Bump mapping best approximates an embossed surface when the heights of the bumps are not too great and when the surface is viewed from a direction close to the surface normal. When looking at a rendered object, bump mapping can be



recognised immediately, in the sense that the bumps do not "pop out" of the surface at silhouettes or edges.

Several hardware based approaches to implementing bump mapping have been presented in recent years[25, 71, 50]. Peercy[71] presents an implementation for high-end 3D graphics hardware that supports bump mapping within the context of per-fragment lighting operations. Kilgard[42] presents an approach to bump mapping that takes advantage of currently available low-cost graphics hardware.

The limitation of bump mapping becomes obvious when the surface is parallel to the viewer and the bump does not create a silhouette. Also as a surface moves in perspective space the shape created in the viewers mind by the bump map will not occlude other objects. Bump mapping can only simulate the roughness of natural surfaces with only small deformations well, it cannot be used to alter the geometry of a surface. To alter the geometry of the surface displacement mapping is needed.

## 2.1.2 Displacement Mapping

To add real geometric detail to a flat surface, *displacement mapping*, first introduced by Cook[10], can be used.

Displacement mapping uses a base surface defined by a bivariate vector function  $\mathbf{P}(u, v)$ <sup>1</sup> that defines 3D points  $(x, y, z)$  on the surface. Displacement values for the surface of the object are determined from the displacement map in a similar fashion to determining the colours of a surface from a texture map. The displacements from the surface are defined by a bivariate scalar function  $d(u, v)$  and the normals on the base surface  $\mathbf{P}(u, v)$  by  $\hat{\mathbf{N}}(u, v)$ . The points on the new displaced surface  $\mathbf{P}'(u, v)$  are defined as follows:

$$\mathbf{P}'(u, v) = \mathbf{P}(u, v) + d(u, v)\hat{\mathbf{N}}(u, v) \quad (2.6)$$

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

A cross section of an example displacement mapped surface is shown in Figure 2.1, where  $\mathbf{N}'(u, v)$  is the normal to the displaced surface.

As an example Figure 2.2(a) shows a flat plane with a colour texture applied, which is displaced using the displacement map in Figure 2.2(b), where white represents a high displacement and black represents no displacement. The application of the displacement map to the plane results in the the displaced surface shown in Figure 2.2(c).

<sup>1</sup>A bivariate vector function is a function of two variables where the result is a vector.