Displacement Mapping and Volume Rendering Graphics Hardware

by

Michael C. Doggett

ISBN: 1-58112-166-0

DISSERTATION.COM

USA • 2002

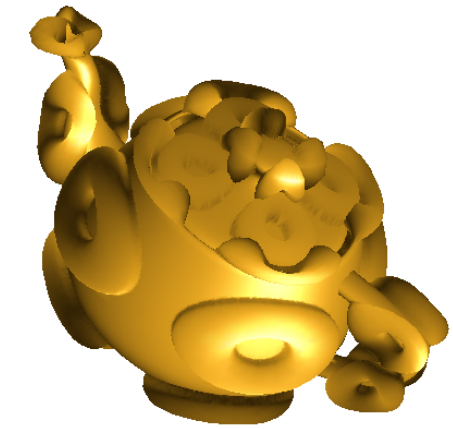
Displacement Mapping and Volume Rendering Graphics Hardware

Dissertation.com
USA • 2002

ISBN: 1-58112-166-0

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 GRaphisch-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 interactve programmable hardware acelerator 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 Etzmuß, 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 Communites (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

List of Figures

List of Tables

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 encounted 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 imploys 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 reponsible 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 approches 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 profond 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:

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

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:

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

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] :

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

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

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

$$\mathbf{N}' = \mathbf{N} + \mathbf{AB} \tag{2.5}$$

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 $\mathbf{P}(u,v)$[1] 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:

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

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).

[1] A bivariate vector function is a function of two variables where the result is a vector.