Comprehensive Summaries of Uppsala Dissertations
from the Faculty of Science and Technology 955

Improved Algorithms for Fast Shading and Lighting

BY

ANDERS HAST

ACTA UNIVERSITATIS UPSALIENSIS
UPPSALA 2004

Dissertation presented at Uppsala University to be publicly examined in Häggsalen, Ångströmslaboratoriet, Uppsala, Thursday, April 29, 2004 at 10:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract

Hast, A. 2004. Improved Algorithms for Fast Shading and Lighting. Acta Universitatis Upsaliensis. *Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 955. 116 pp. Uppsala. ISBN 91-554-5916-1

Shading is a technique that is used to make faceted objects appear smooth. Gouraud and Phong shading are the most well known shading techniques and Gouraud is the simplest of the two. Phong is not that often used for software rendering since it is not as computationally fast as Gouraud. The Phong illumination equation is computed per pixel instead of per vertex as in the case for Gouraud. Moreover, it is necessary to normalize the interpolated normals and this is a time consuming task since it is done per pixel. Phong shading is well suited for modern GPU's. However, Gouraud is more often chosen for software rendering since it is so much faster. Quadratic shading has been proposed as a fast alternative to Phong. Nonetheless, it is not often used since the setup is rather complex.

We have investigated how the mathematics behind shading algorithms can be reformulated in order to make shading faster. Bump mapping is closely related to shading since vector interpolation is necessary for fast bump mapping. Therefore, fast interpolation methods are valuable for bum mapping too.

A distinction between scanline and polygon setup algorithms is don and improvements for both are discussed. Especially the polygon setup for quadratic shading has been improved in several steps. Moreover, the scanline setup algorithm could be improved by using slerp. It is shown how slerp can be evaluated very efficiently along a scanline.

The Phong illumination equation has also been investigated and especially the highlights can be computed faster. Furthermore, it is shown how the proposed model can be used more to decide the size of highlights more efficiently. The warn spotlight model is also improved so that stage lighting terminology can be applied for spotlights.

Keywords: Computer Graphics

Anders Hast, Centre for Image Analysis, Lägerhyddsv. 17, Uppsala University, SE-75237 Uppsala, Sweden

ISSN 1104-232X
ISBN 91-554-5916-1
urn:nbn:se:uu:diva-4135 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-4135)

Printed in Sweden by University Printers, Uppsala 2004

I dedicate this thesis to all those of you who have supported me in different ways over these years of hard work.

M. I♥U

List of enclosed papers

The thesis is based on the following publications, which will be referred to in the text by their Roman numerals:

I. Improved Shading Performance by avoiding Vector Normalization, A. Hast, T. Barrera, E. Bengtsson, WSCG'01, short paper pp. 1-8. 2001.

II. Approximated Phong Shading by using the Euler Method, A. Hast, T. Barrera, E. Bengtsson, Eurographics01, short paper pp. 43-48, 2001.

III. Shading by Spherical Linear Interpolation using De Moivre's Formula A. Hast, T. Barrera, E. Bengtsson, WSCG'03, Short Paper, pp. 57-60, 2003.

IV. Faster shading by equal angle interpolation of vectors T. Barrera, A. Hast, E. Bengtsson, IEEE Transactions on Visualization and Computer Graphics, pp. 217-223, 2004.

V. Faster Computer Graphics - by Reformulation and Simplification of Mathematical Formulas and Algorithms, A. Hast, T. Barrera, E. Bengtsson, IMAGINE2001, 2001.

VI. Improved Bump Mapping by using Quadratic Vector Interpolation, A. Hast, T. Barrera, E. Bengtsson, Eurographics02, short paper/Poster 2002.

VII. Surface Construction with Near Least Square Acceleration based on Vertex Normals on Triangular Meshes, Tony Barrera, Anders Hast, Ewert Bengtsson pp. 43-48, Sigrad 2002.

VIII. Fast Quadratic Shading by using a Mid-edge Vector Approximation, Tony Barrera, Anders Hast, Ewert Bengtsson, *Submitted for publication*.

IX. Reconstruction Filters for Bump Mapping, A. Hast, T. Barrera, E. Bengtsson, WSCG'02, Poster, pp. 9-12, 2002.

X. Fast Intensity Distribution Functions for Soft and Hard Edged Spotlights A. Hast, T. Barrera, E. Bengtsson, WSCG'04, Short Paper, pp. 95-99, 2004.

XI. Fast Specular Highlights by modifying the Phong-Blinn Model, A. Hast, T. Barrera, E. Bengtsson, Siggraph, Sketches and applications 2003.

All papers published or accepted for publication are reproduced with permission from the publisher.

The author has significantly contributed to the work performed in all the papers. The author has been deeply involved in algorithm discussions, method development, and algorithm implementations which have taken place in connection to the work.

Faculty opponent is Dr. Jos Stam, Alias, Toronto, Canada.

Related work

In addition to the papers included in this thesis, the author has also written or contributed to the following publications:

i. Fast Setup for Bilinear and Biquadratic Interpolation over Triangles, A. Hast, T. Barrera, E. Bengtsson, Graphics Programming Methods, Editor Jeff Lander, Charles River Media, pp. 299-314, 2003.

ii. A Modified Phong-Blinn Light Model for Shadowed Areas A. Hast, T. Barrera, E. Bengtsson, Graphics Programming Methods, Editor Jeff Lander, Charles River Media, pp. 231-236, 2003.

iii. Antialiasing for Bump Maps and a Fast Normalization Trick A. Hast, T. Barrera, E. Bengtsson, Graphics Programming Methods, Editor Jeff Lander, Charles River Media, pp. 291-298, 2003.

iv. A Fast and Simple All-Integer Parametric Line T. Barrera, A. Hast, E. Bengtsson, Graphics Programming Methods, Editor Jeff Lander, Charles River Media, pp. 133-138, 2003.

v. A Fast All-Integer Ellipse Discretization Algorithm T. Barrera, A. Hast, E. Bengtsson, Graphics Programming Methods, Editor Jeff Lander, Charles River Media, pp. 121-132, 2003.

vi. Swedish Universities Prepare Students for Entry Into Industry, V. Zdravkovic, E. Carling, S. Seipel, A. Hast, Computer Graphics, newsletter of ACM SIGGAPH. 36(2) May, 2002.

vii. Faster Algorithms for Interactive Computer Graphics, A. Hast, T. Barrera, E. Bengtsson, Working Paper No 19, Proceedings from the first Seminar on Computer Graphics and Visualization, pp. 8-25, 2001.

viii. Licentiate Thesis: Improved Fundamental Algorithms for fast Computer Graphics, Anders Hast, 2002.

ix. Snow Accumulation in Real-time Håkan Haglund, Anders Hast, pp. 11-15, Sigrad 2002.

x. Animation of Water Droplet Flow on Structured Surfaces Malin Jonsson, Anders Hast, pp. 17-22, Sigrad 2002.

xi. Reconstruction Filters for Bump Mapping, A. Hast, T. Barrera, E. Bengtsson, Second Conference for Promotion of Research in IT at New Universities and University Colleges in Sweden. pp 244-256, 2002.

xii. A Modified Phong-Blinn Light Model for Shadowed Areas Third Conference for Promotion of Research in IT at New Universities and University Colleges in Sweden. 2003.

Contents

Introduction and Objectives

Computers graphics has emerged as an useful tool for many disciplines within science. Medical imaging, scientific visualization and applications of image analysis would be impossible without the use of computer graphics. With the advent of home computers, the general public is now familiar with computer graphics. For instance, graphical user interfaces are now standard on most operating systems, and today, most computers have graphics acceleration cards. It is not the case that modern operating systems need them. Instead, the most popular applications used for home computers have driven the evolution of graphics cards. And those applications are of course games, not at least 3D games.

Since the first computer animated special effects in the early eighties, computer animated movies have become increasingly popular. Many of the special effects seen in movies today are computer graphics generated. One proof of this can be seen in the list of nominees for the academy award in fields like animated movies and special effects.

When hand held devices like cell phones started to have a graphical display a new field for computer graphics was opened. Since most such devices do not have a graphics card, fast and simple software algorithms are once again useful.

1.1 Goal and focus

The aim of our research has been to come up with new ideas and to develop new and faster algorithms, methods and techniques as well as improved techniques giving a more realistic result, in the field of computer graphics. The papers presented here will discuss different software implementations. However, some of the algorithms themselves should be possible to realize in hardware. Moreover, some can be implemented on a graphics card with a programmable processing unit (GPU).

The focus for our research has been on shading algorithms including bump mapping and on the local lighting model used for shading. Many of these algorithms have been around since the dawn of computer graphics and it seems like they have been considered as final by many, in the sense that there are not much that can be done to improve them. Nonetheless, some have come up with

interesting improvements lately, and we hope that our contributions described in this thesis will be useful for the computer graphics community in the future.

Even though over thirty years have past since Henri Gouraud [32] published his article, which showed how faceted objects could be rendered smoothly, it is still one of the most used algorithms for shading. The reason why the algorithm by Phong [57] is not often used, even though it gives much better quality, is probably due to the normalization process needed for Phong's method. The normalization includes, a division and a square root, which are computationally expensive compared to additions and multiplications. Even bump mapping suffers from this problem. The normalization is sometimes substituted with table look-up algorithms, which are fast in hardware, but rather slow in software. One of the overall goals for our research has therefore been to remove the need for normalization by using alternative methods, which give a result that is not noticeable different from previous techniques. Several of the papers in this thesis propose different ways of avoiding vector normalization. This is important for making software implementations faster but it also seems to be something hardware manufacturers actively avoid in their hardware.

We will make a distinction between two basic approaches when it comes to shading. These approaches differ in how the setup is done. One approach is hereby called polygon setup and the reason for this is that most of the setup is done before the rasterization of the polygon. Therefore, the setup needed for each scanline is incrementally computed per scanline and thus, the computationally effort needed per scanline is minimized.

The other approach will be referred to as scanline setup algorithms, which perform a minimal setup for each polygon, and a second setup is needed for each scanline. This setup typically computes the variables needed in the forward difference that is evaluated over the scanline.

It will be discussed in this thesis which approach is the best when it comes to quality and speed. As usual we can not have both, therefore a trade off is necessary.

1.2 About this thesis

Chapter two describes the fundamentals of shading algorithms and the local lighting model often used for shading. Moreover, bump mapping, different types of luminaires and filtering techniques are briefly described. Hopefully, these introductory overviews will familiarize the reader with the concepts that are the base for this thesis. Furthermore, some insights not easily found in literature, which were gained during the research, are also explained.

In chapter three, the included papers are summarized, and the main research contributions and results are pointed out and discussed. Paper i, ii and iii in the related work section should have been included among these papers as

well. However, due to copyright problems these papers could not be included and had to be put into the related work list. Still, they are briefly discussed in chapter three, since they are related to the other papers in many ways. In chapter four our results are discussed.

Background

This chapter describes the fundamentals of shading algorithms and the local lighting model often used for shading. Moreover, bump mapping, different types of luminaires and filtering techniques are briefly described.

2.1 The Phong Illumination Model

Shading is a technique used to make faceted objects appearing smoothly curved. An illumination model is applied in the shading process in order to achieve this effect. Usually Phong's [57] illumination model is used for interactive applications, since it is simple and fast. On the other hand, when realism is important, a global illumination model like ray tracing or radiosity is used, which take inter-reflections between facets into account. Usually more sophisticated ways of computing light than Phong's model is used for global models. These include micro facet (BRDF) approaches [74], which increase realism since they allow a wider spectrum of materials to be shaded.

Phong's illumination model tend to produce too perfect plastic like materials. Whereas BRDF's can produce metallic surfaces better. However, they are often too computationally expensive to use for real time graphics.

In its simplest form the Phong illumination model can be described as

$$I = I_d + I_s + I_a \tag{2.1}$$

where the intensity of the light I is computed as the sum of the diffuse light I_d, the specular light I_s and the ambient light I_a. The diffuse light is the light coming from a perfect matte material. Such material will scatter light equally in all directions and is therefore perceived as a non shiny surface. The diffuse light is computed using Lambert's Law [26]

$$I_d = cos\,\aleph = \mathbf{n} \cdot \mathbf{l} \tag{2.2}$$

where $\mathbf{n}$ is the normal vector of the surface and $\mathbf{l}$ is the vector in the direction to the light source as shown in figure 2.1

The specular light is the light reflected by a shiny surface and is visible as a so-called highlight on an object. The specular light is view dependent in contrast to the diffuse light that is equally bright in all directions. Therefore,

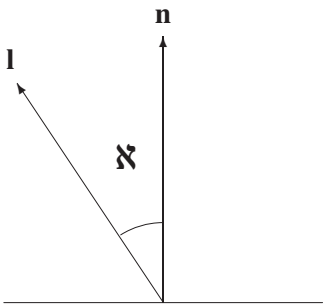

Figure 2.1: The normal vector **n** and the vector in the direction to the light source **l** is needed in order to compute the diffuse light using Lambert's law.

the computation involves a vector in the direction to the viewer **v**. Moreover, the reflection of the light vector **l** around the normal is needed. This reflection vector **r** is usually computed as

$$\mathbf{r} = 2\mathbf{n}(\mathbf{n}\cdot\mathbf{l}) - \mathbf{l} \tag{2.3}$$

Phong [57] proposed that the specular light should be computed as

$$I_s = \cos^s \forall = (\mathbf{r}\cdot\mathbf{v})^s \tag{2.4}$$

where $\forall$is the angle between **r** and **v** as shown in figure 2.2. Furthermore, s is a constant that will decide the size of the highlight. This model is not based on physics but produces a physically plausible result. The power function makes the band width of the highlight smaller if s is increased.

Another, and sometimes faster way of computing the specular light is by using the half-way vector introduced by Blinn [13]. The half-way vector is computed as

$$\mathbf{h} = \frac{\mathbf{l}+\mathbf{v}}{\| \mathbf{l}+\mathbf{v} \|} \tag{2.5}$$

It can be proven that the normal giving a maximal highlight is equal to **h**. Therefore, the specular light can be computed as

$$I_s = (\mathbf{n}\cdot\mathbf{h})^{\mathbf{s}} \tag{2.6}$$

This equation is faster than Phong's original computation since it is not necessary to compute the reflection vector, especially if parallel light is used and

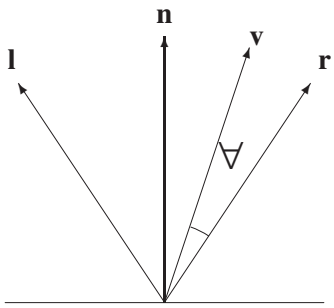

Figure 2.2: The vector **r** in a perfect reflection around the normal **n** and the vector in the direction to the viewer **v** is needed in order to compute the specular light in the Phong model.

not a point light source. Moreover, this approach is faster if the vector **v** is assumed being constant. This is often the case for games. On the other hand it is more efficient to use the original formulation when several light sources are used as shown in [25]. It can be shown that **v** can be reflected around the normal instead of the different light vectors $\mathbf{l_i}$. Subsequently the specular light is computed by taking the dot product between the reflected view vector and each light source direction vector $\mathbf{l_i}$.

Finally, the ambient light is an approximation of the surrounding light that is the result of inter-reflections between objects. As an example, we know that even if there is no direct light under the table in our kitchen, it still will not be completely dark there. The reason for this is that some reflected light reaches the floor via the wall, etc.

The Phong illumination equation also includes the color of the material that is being lit as well as the color of the light source. It could also be argued whether the highlight is affected by the color of the material or if it should always be white. In the micro facet approach the color of the highlight varies depending on the angle between the light source and the viewer.

There are also a number of different types of light sources that could be used, as shown in figure 2.3. A parallel light source has a constant light vector, e.g. the sun. A point light source or a spot light will have a variable **l**. Moreover, point light sources emit light in all directions, while a spotlight emit light within an specific angle from the main direction. This light will be visible as a spot if directed on a planar surface, thereof the name.

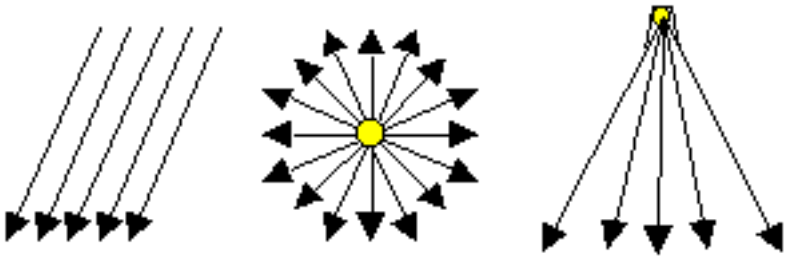

Figure 2.3: From left to right: A parallel light source, a point light source and a spotlight.

Figure 2.4: A Gouraud shaded torus.

2.2 Shading

Gouraud [32] introduced bi-linear interpolation for shading of polygons. He proposed that the normal vector for each vertex could be computed as the normalized average of the normals of each of the polygons sharing that vertex. The intensity could then be computed for each vertex using the Phong illumination equation, even though it should be mentioned that Gouraud only focused on diffuse light in his paper. A Gouraud shaded torus is shown in figure 2.4

Later, Phong [57] proposed that the normals should be bi-linearly interpolated over the polygon. The illumination equation he proposed in the same paper could be used per pixel, instead of per vertex as it is used with Gouraud shading in for instance OpenGL. There are several advantages by using Phong shading, but also disadvantages. One advantage is that Mach-bands [26] are generally less visible when Phong shading is used instead of Gouraud shading.

Figure 2.5: A Phong shaded torus.

Mach-bands are a visual illusion that comes from the fact that the human eye tries to find edges on objects. Since the intensity generally is not C^1continuous over the edges from one polygon to another, these will appear as white bands in the shaded images.

Another advantage is that a highlight that is close to one vertex is not smeared out over the polygon as is the case for Gouraud. Furthermore, it will not miss a highlight that is visible in the middle of a polygon as Gouraud will. In conclusion, Phong shading produces much better highlights than Gouraud shading and the Mach-band effect will generally be less noticeable. A Phong shaded torus is shown in figure 2.5

However, there are also some disadvantages. One is that Phong shading is much more computationally expensive than Gouraud shading, since the illumination model is computed per pixel instead of per vertex. Moreover, the normalization necessary per pixel includes a division and a square root. Both are computationally complex and they also take a lot of space in a hardware implementation. It should be noted that Phong shading is not implemented in OpenGL. There is also a very simple type of shading often called flat shading. The intensity is not interpolated when flat shading is used. Instead one intensity per polygon is computed and the normal of each polygon is used for this computation. A flat shaded torus is shown in figure 2.6.

2.3 Bump mapping

Blinn [12] introduced bump mapping as a technique that makes a surface appear rough or wrinkled. This affect is achieved by perturbing the normals used

Figure 2.6: A flat shaded torus.

in the illumination computation. Hence, it only affects the shading, not the underlying geometry. The bump map can contain height values that will affect the normal of each pixel. However, it can as an alternative contain normals that have been computed from such a bump map and this map is then called a normal map [42].

Especially moving frame bump mapping[56] [42] is closely related to shading since it includes vector interpolation, which is the essence of Phong shading. Bump mapping is also closely related to texture mapping [34]. It will also suffer from similar aliasing problems when bumps are magnified or minified.

Blinn used an approximation for the perturbed normal, which depends on the surface normal and the height information in the bump map. The perturbed normal is calculated as

$$\mathbf{n}' = \mathbf{n} + \frac{F_u(\mathbf{n} \times \mathbf{P_v}) - F_v(\mathbf{n} \times \mathbf{P_u})}{\mathbf{n}} \tag{2.7}$$

where $\mathbf{n}$ is the surface normal, $\mathbf{P_u}$ and $\mathbf{P_v}$ are the partial derivatives of the surface in the u and v directions respectively. F_u and F_v are the gradients of the bump map. Peercy et al. [56] use another approach for computing the normal. An orthonormal frame on the surface is used to rotate the vector in the direction to the light source $\mathbf{l}$ into that local frame. As an alternative, the bump normal could be rotated into the world by the same frame, i.e. the inverse of the same matrix. However, it is more efficient to rotate the light vector so it will be correct compared to the flat normal map. This can be done per vertex and then the rotated light vectors are linearly interpolated over the polygon. Hence, the diffuse intensity is computed as $I_d = \mathbf{n}' \cdot \mathbf{l}'$, where $\mathbf{n}'$ is the normal obtained from the bump map, and $\mathbf{l}'$ is the light vector rotated by the frame

$(\mathbf{t}, \mathbf{b}, \mathbf{n})$, where $\mathbf{t}$ is the tangent vector of the surface at the point with the surface normal $\mathbf{n}$ and finally $\mathbf{b}$ is the bi-normal, computed as $\mathbf{n} \times \mathbf{t}$. They are taking this approach a step further by precomputing bump normals over the whole object. Nonetheless, this is the main idea of moving frame bump mapping. The perturbed normal is

$$\mathbf{n}' = (-F_u, -F_v, 1) \tag{2.8}$$

which is obtained by computing the cross product of the gradients as shown by Doggett et al. [18]. They also use Prewitt masks, which is a kind of convolution kernel, to obtain F_u and F_v. Kugler [45] uses a different representation of the normal perturbation based on spherical coordinates. Sung Kim et al. [67] elaborates this idea further by directly computing the inner products for the diffuse and specular light from the perturbed normal in this representation. A hardware implementation of a bump mapping chip is proposed by Ikedo and Ohbuchi [39], where vector normalization is avoided by using vectors in angular form. Miller and Halstead [50] discuss how hardware accelerated bump mapping using standard texture-mapping hardware can be performed. An overview of other bump map approaches is given by Ernst et al. [21] and Kilgard [42].

In this type of bump mapping, the frame is used to rotate the vector in the direction to the light source. This can be done since a rotation matrix is an orthonormal frame. That is, each column in a rotation matrix is a vector and each vector has unit length and they are orthogonal to each other. This fact is seldom mentioned in computer graphics text books, but it certainly helps understanding the concept of rotation matrices. A rotation of $\aleph$degrees around the z-axis is defined as

$$R_z(\aleph) = \begin{bmatrix} \cos\aleph & -\sin\aleph & 0 \\ \sin\aleph & \cos\aleph & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

It is clear that the last column is a vector aligned with the z-axis. It is also easy to see that this last vector is orthogonal to both the two first vectors since the dot product between the last column and any of the two first is zero. The two first vectors are also orthogonal since the dot product once again is zero: $\cos\aleph \cdot \sin\aleph + (-\sin\aleph \cdot \cos\aleph) + 0 \cdot 0 = 0$. Similarly it can be shown that each vector has unit length. The length of the first vector is $\sqrt{\cos^2\aleph + \sin^2\aleph + 0^2}$. The trigonometric unity gives that this is equal to one. The same applies for the other two columns. Furthermore, each row is also a vector and they also constitute a frame.

The interpretation of this concept is that if this frame is the same as the world coordinate system, i.e. the unity matrix, then the object is not rotated. However, if we rotate the coordinate system (or frame) the object is defined in,

then the object itself is rotated in the same way as the frame is rotated. The same concept applies for both vectors and objects.

The conclusion is that if we want to rotate an object or a vector to a certain position, we use a rotation matrix that contains a local frame that defines this rotation. Moreover, this frame can be regarded as the local coordinate system where the object or vector is defined.

If we use equation (2.9) to rotate a position vector $\mathbf{p}$ with

$$\mathbf{p}' = \mathbf{p}R_z(\aleph) \tag{2.10}$$

then the frame will be found on the rows of $R_z(\aleph)$. Note that the row vectors are the inverse of the column vectors since the inverse of a rotation matrix is the same as its transpose. Thus, in the moving frame bump mapping approach, the inverse of the frame is used, i.e. the vectors in the frame are stored in the columns.

Homogenous coordinates [26] expands this concept by allowing the frame to be translated to any position in the world coordinate system. This is achieved by using an extra coordinate telling whether the x, y and z coordinates define a point or a vector.

2.4 Lighting in Object Space

In interactive computer graphics as well as animations, objects are moved around and rotated in the scene. Similar transformations are used on the objects if the camera is moved around in the scene. Objects are rotated by applying a matrix multiplication with a rotation matrix M_R for both the vertices coordinates P_m and the vertex normals $\mathbf{n_m}$. If back face culling [26] is used then the normals of the polygon must also be rotated. They are needed in order to determine if a polygon is facing away from the viewer and therefore can be discarded in the rasterization process.

Although, translations are applied for P_m, they are not used for $\mathbf{n_m}$, since they are vectors. If the object has m vertices, then the rotation of the normals is said to be $O(m)$. This cost could be reduced to $O(1)$, when local shading or moving frame bump mapping is applied for the object, assuming that parallel light sources are used. This concept is known as lighting in object space [22] or lighting in model space [69]. Especially the concept of bump mapping will be easier to explain by using the following transformations

$$\mathbf{v}' = M_R\mathbf{v} \tag{2.11}$$

where $\mathbf{v}$ is a vector and M_r is the rotation matrix. Using the form $\mathbf{v}' = \mathbf{v}M_R$ would just rotate $\mathbf{v}$ in the opposite direction as shown later.

2.4.1 Shading

The Phong illumination model propose that the diffuse intensity is computed as

$$I_d = \mathbf{l} \cdot \mathbf{n}' \tag{2.12}$$

where $\mathbf{l}$ is the vector in the direction to the light source, and $\mathbf{n}'$ is the rotated normal of the surface at this point. The specular intensity can be computed as

$$I_s = (\mathbf{v} \cdot \mathbf{r})^s \tag{2.13}$$

where $\mathbf{r}$ is the vector in the direction of the perfect reflection from the light source and $\mathbf{v}$ is the vector in the direction to the viewer, and finally s is the shininess value which affects the size of the highlighted area. The reflection vector is computed as

$$\mathbf{r} = 2\mathbf{n}'(\mathbf{l} \cdot \mathbf{n}') - \mathbf{l} \tag{2.14}$$

Blinn [13] formulated the specular intensity a bit differently as

$$I_s = (\mathbf{h} \cdot \mathbf{n}')^{\mathbf{s}} \tag{2.15}$$

How are these equations affected when the object is rotated by the rotation matrix M_R? First the diffuse intensity is investigated. Remember that

$$\mathbf{n}' = M_R\mathbf{n} \tag{2.16}$$

Thus, the diffuse intensity is

$$I_d = \mathbf{l} \cdot (M_R\mathbf{n}) \tag{2.17}$$

Note, that matrix multiplications are associative, and therefore

$$I_d = (\mathbf{l}M_R) \cdot \mathbf{n} \tag{2.18}$$

This implies that the diffuse intensity could be computed by rotating the vector in the direction of the light source instead of rotating all the normals of the object.

The specular intensity for a rotated normal is

$$I_s = \mathbf{v} \cdot \mathbf{r} = 2\mathbf{v} \cdot \mathbf{n}'(\mathbf{l} \cdot \mathbf{n}') - \mathbf{v} \cdot \mathbf{l} \tag{2.19}$$

Taking the rotation into account yields

$$I_s = \mathbf{v} \cdot \mathbf{r} = 2\mathbf{v} \cdot (M_R\mathbf{n})(\mathbf{l} \cdot (M_R\mathbf{n})) - \mathbf{v} \cdot \mathbf{l} \tag{2.20}$$

thus

$$I_s = 2(\mathbf{v}M_R) \cdot \mathbf{n}((\mathbf{l}M_R) \cdot \mathbf{n}) - \mathbf{v} \cdot \mathbf{l} \tag{2.21}$$

Here, the vectors $\mathbf{l}$ and $\mathbf{v}$ have to be rotated by M_R, but not the dot product $\mathbf{v} \cdot \mathbf{l}$.

The specular intensity according to Blinn is computed as

$$I_d = \mathbf{h} \cdot (M_R \mathbf{n}) \tag{2.22}$$

thus

$$I_d = (\mathbf{h} M_R) \cdot \mathbf{n} \tag{2.23}$$

In conclusion, all components of the intensity in Phong's illumination model can be computed without having to rotate the normal. Instead the $\mathbf{l}$ vector can be rotated for the diffuse light. For the specular light, either $\mathbf{v}$ or $\mathbf{h}$ is rotated, depending on which model is used.

2.4.2 Bump Mapping

Bump mapping use an orthonormal frame on the surface to rotate $\mathbf{l}$ into that local frame. Thus, the diffuse intensity is computed as $I_d = \mathbf{n_B} \cdot \mathbf{l}'$, where $\mathbf{n_B}$ is the normal obtained from the bump map, and $\mathbf{l}'$ is the vector in the direction to the light source, rotated by the frame $M_F = (\mathbf{t}, \mathbf{b}, \mathbf{n})$, where $\mathbf{t}$ is the tangent vector to the surface, $\mathbf{b}$ is the bi-normal, and finally $\mathbf{n}$ is the normal to the surface. Note, that the components of the frame are all rotated by the rotation matrix M_R. Thus

$$\mathbf{l}' = \mathbf{l} M_F \tag{2.24}$$

and

$$M_F = (M_R \mathbf{t}, M_R \mathbf{b}, M_R \mathbf{n}) \tag{2.25}$$

The previous equation can be reformulated by breaking out $\mathbf{M_R}$

$$M_F = M_R(\mathbf{t}, \mathbf{b}, \mathbf{n}) \tag{2.26}$$

Thus, $\mathbf{l}'$ can be computed as

$$\mathbf{l}' = \mathbf{l} M_R(\mathbf{t}, \mathbf{b}, \mathbf{n}) \tag{2.27}$$

The associative law gives

$$\mathbf{l}' = (\mathbf{l} M_R)(\mathbf{t}, \mathbf{b}, \mathbf{n}) \tag{2.28}$$

It is obvious that for moving frame bump mapping only the $\mathbf{l}$ vector needs to be rotated, while the frame does not need to be rotated.

It can be argued that the bi-normal is computed from the normal and the tangent in order to save computations since a rotation of the bi-normal takes more time. However, by rotating $\mathbf{l}$ by M_R instead, it is possible to precompute the bi-normal and store it together with the normal and the tangent.

This method only works for parallel light sources. If a point light source is used then the vector in the direction at the light source must be computed for every shaded pixel on the polygon. It is not efficient to rotate all these vectors instead of rotating the three vertex normals.

It can be noted, that the rotation for the method is applied in a reversed order compared to rotating $\mathbf{n}$. This makes perfect sense, since this implies that, instead of rotating $\mathbf{n}$ by M_R, $\mathbf{l}$ is rotated by the inverse of M_R. This can easily be proved by

$$(\mathbf{l}M_R)^T = (M_R^T\mathbf{l}^T) \tag{2.29}$$

This is known as the product of the transposes in reverse order [5]. Note, that the transpose of $(\mathbf{l}M_R)$ and the transpose $\mathbf{l}$, both are transpose of vectors, and will not change the content of them.

It can be shown that M_R is an orthonormal vector space, and therefore the following relation holds [26]

$$M_R^T = M_R^{-1} \tag{2.30}$$

thus

$$\mathbf{l}M_R = M_R^{-1}\mathbf{l} \tag{2.31}$$

Hence, this can be viewed as rotating $\mathbf{l}$ with the inverse of M_R.

If back face culling is used, then the normal of the polygon has to be rotated separately. This can be done in the same way as usual.

The computational cost of rotating one single object with m vertices is $O(m)$. This applies for both ordinary shaded objects and bump mapped objects, including back face culling. However, this approach is $O(1)$, since only the $\mathbf{l}$ vector and the $\mathbf{h}$ or $\mathbf{v}$ vector has to be rotated.

2.5 Bi-linear Interpolation

Bi-linear interpolation is often used in software shading algorithms and requires an explanation of its own. Figure 2.7 shows how a linear interpolation of normals can be done. The normal on the left edge is interpolated linearly from $\mathbf{n_0}$ to $\mathbf{n_2}$ yielding $\mathbf{n_a}$. Likewise $\mathbf{n_b}$ is obtained by interpolating from $\mathbf{n_0}$ to $\mathbf{n_1}$. Finally $\mathbf{n}$ is obtained by a linear interpolation over the scanline by interpolating from $\mathbf{n_a}$ to $\mathbf{n_b}$. Linear interpolation of normals requires an interpolation of each component of the normal vector. Moreover, it has to be normalized when Phong shading is used. If the intensity is interpolated, then only one interpolation is necessary. However, sometimes it can be preferable to interpolate RGB values instead, which requires three interpolations. On the other hand, if the intensity is interpolated, then it has to be multiplied with the color for each pixel. Nonetheless, intensity interpolation is often used in papers when algorithms are explained.

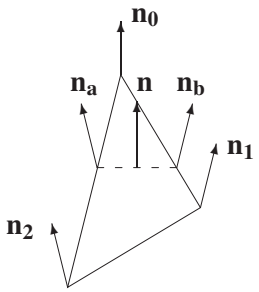

Figure 2.7: Linear interpolation is done on the edges as well as over the scanline.

A simple linear interpolation of intensities is defined as

$$I_a = I_0(1-u) + I_1 u \tag{2.32}$$

It is easy to see that $u = 0$ gives $I_a = I_0$, $u = 1$ gives $I_a = I_1$ and $u = 1/2$ gives $I_a = (I_0 + I_1)/2$. Hence, a *linear* interpolation between I_0 and I_1 is obtained. In Gouraud shading the intensities on the edges can be obtained by linear interpolation. The interpolation on the other edge can be obtained by a similar interpolation

$$I_b = I_0(1-u) + I_2 u \tag{2.33}$$

The intensity over the scanline is then obtained as a linear interpolation between I_a and I_b

$$I = I_a(1-v) + I_b v \tag{2.34}$$

This type of interpolation is called a bi-linear interpolation since it depends on two variables. Hence, the previously explained interpolation of normals is also a bi-linear interpolation. We can rewrite the last equation as

$$I = I_a + v(I_b - I_a) \tag{2.35}$$

This is the well known equation for a straight line. For each scanline we start with $I = I_a$ and for each pixel $(I_b - I_a)/k$ is added to this value, where k is the number of pixels over the scanline. Hence, a small setup is needed for each scanline that involves a division. This is a typical scanline setup approach and is often used as an example in text books [7]. The division can be avoided by using a bi-linear interpolation, which uses a polygon setup approach instead. Here we can compute the gradients in the scan line direction and in the direction along the edges as shown by for example Narayanaswami [52].

Let $\forall$ be a bi-variate linear function

$$\ldots = ax + by + c \tag{2.36}$$

The coefficients for a bi-variate polynomial over this triangle is first computed by setting up the following system of equations

$$\ldots = M\mathbf{c} \tag{2.37}$$

where

$$\ldots = [I_0, I_1, I_2]^T \tag{2.38}$$

$$\mathbf{c} = [a, b, c]^T \tag{2.39}$$

The matrix M is

$$M = \begin{bmatrix} 0 & 0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \tag{2.40}$$

Here, relative coordinates are used in order to transform the upper vertex to $p = (0,0)$, which will simplify the following computation drastically.

The solution for $\mathbf{c} = M^{-1}\ldots$is

$$a = I_0 \frac{y_1 - y_2}{x_1 y_2 - x_2 y_1} + I_1 \frac{y_2}{x_1 y_2 - x_2 y_1} - I_2 \frac{y_1}{x_1 y_2 - x_2 y_1} \tag{2.41}$$

$$b = I_0 \frac{x_2 - x_1}{x_1 y_2 - x_2 y_1} - I_1 \frac{x_2}{x_1 y_2 - x_2 y_1} + I_2 \frac{x_1}{x_1 y_2 - x_2 y_1} \tag{2.42}$$

$$c = I_2 \tag{2.43}$$

This approach requires one addition per scanline only for the setup. Thus, the setup is computed incrementally and fulfills the criteria for polygon setup approaches.

Pseudo code for a Gouraud shader using the described approach:

```
p = I0;
kr = x2/y2;
kl = x1/y1;
dr = a;
dp = a * kl + b;
for (y = y0; y < y2; y++)
    r = p;
    for (x = xs; x < xe; x++)
        intensity = r;
        r+ = dr;
    end
    p+ = dp;
    xs+ = kl;
    xe+ = kr;
end
```

Duff [20] showed that the computation of Phong shading could be done in a more efficient way. The computation over a scanline is

$$\frac{\mathbf{n}\cdot\mathbf{l}}{\|\,\mathbf{n}\,\|} = \frac{Ax+B}{\sqrt{Cx^2+Dx+E}} \tag{2.44}$$

The numerator is a linear equation. As Duff points out, it would be the same as Gouraud shading if the denominator is not used, i.e. normalization is not performed. Furthermore, the denominator is the square root of a quadratic bi-variate function.

2.6 Scanline setup algorithms

Scanline setup algorithms, perform a minimal setup for each polygon, and a second setup is needed for each scanline. A typical example of this kind of setup is fence shading [10] where the normal of the vertices are interpolated on the edges. Then, Gouraud shading is done over the scanline. Hence, a setup computing the variables needed for the scanlines is performed.

Kuijk and Blake [46] introduced an approach based on angular interpolation that will give approximations to the Phong curve. Ouyang and Maynard [55] propose an approach that will produce correct Phong shading. However, by extending scanline temporarily to a length that is a power of two, the number of square roots will be less, and therefore it is faster than Phong shading.

2.7 Polygon setup algorithms

There are many examples of polygon setup algorithms. Duff [20] shows how the computation can be implemented in an efficient way, requiring only three additions, one division and one square root per pixel for Phong shading. Bishop and Weimer [11] use a Taylor series expansion of Duff's equation to obtain a second order bi-variate polynomial. It could be evaluated by only two additions per pixel along a scanline. Moreover, it is evaluated by only three additions along the edge. This is the main advantage, by using polygon setup algorithms. A scanline setup algorithm would need a normalized normal per scanline. Either by some kind of approximation or true normalization. If the latter is used the normal must be interpolated, which is done by three additions. Then a division and a square root is needed.

2.8 Illumination and Lighting

Computer graphics is in many ways a question about handling and computing light in a proper way. When a lighting artist is lighting a scene for a computer

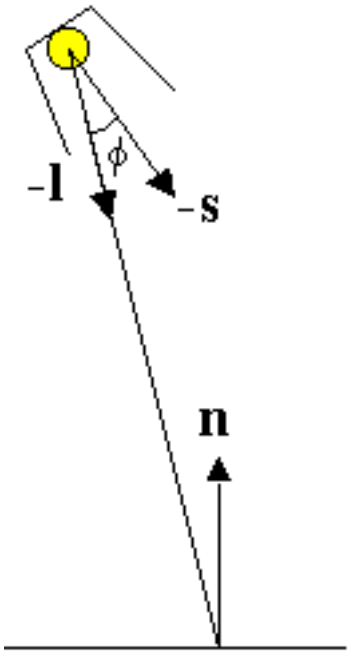

Figure 2.8: The light intensity is modeled by the directional multiplier, which is computed as $D(\forall) = (\mathbf{s} \cdot \mathbf{l})^c$.

animated movie, he usually makes use of several light sources, like key, fill and back light, in order to produce lighting that looks right [66]. Often some kind of spotlight is used in this process. Warn [73] showed how the light could be concentrated in the primary direction of the spotlight by using a power function. If the primary direction is denoted s and the direction to the pixel currently shaded is denoted $\mathbf{l}$ as shown in figure 2.8, then the diffuse intensity is modeled by a directional multiplier $D(\forall) = \cos^c = (\mathbf{s} \cdot \mathbf{l})^{\mathbf{c}}$ where the exponent c provides control over the concentration of the light. The concentration can be delimited by a cone, where the intensity is set to zero outside the cone. This factor is then used when computing the diffuse and specular light using the Phong illumination model [57]. As an example, the diffuse intensity of a pixel can be computed as

$$I_d = D(\forall)(\mathbf{n} \cdot \mathbf{l}) \tag{2.45}$$

The specular intensity is computed in a similar way, by multiplying the factor with the specular intensity. Even though the power function is relatively slow, this model is still used for several API's [76],[71]. Another model is often used by professional modeling tools [1] and in graphics hardware [24]. The basic idea is that inside an inner cone, or hotspot, the concentration of light is constant. While outside this cone the intensity drops off smoothly down to

zero where it reaches the outer cone. This ramp function is either a linear or cubic function. However, the cubic function produces more visually appealing results than the linear drop off.

New Algorithms and Models

The included papers are summarized, and the main research contributions and results are pointed out and discussed in this chapter. Paper i, ii and iii in the related work section are also briefly discussed.

3.1 Papers I and II: Scanline setup algorithms

Paper I and II propose several scanline setup algorithms. However, it should be pointed out that some of these could be extended to work for the polygon setup approach as well. Nevertheless, this is not shown in the papers. How this can be done is discussed in the end of this section.

Paper I makes use of the known fact that the equation for calculating the vector in a perfect reflection of the vector pointing at the light source around the normal often written as

$$\mathbf{r} = 2\mathbf{n}'(\mathbf{l} \cdot \mathbf{n}') - \mathbf{l} \tag{3.1}$$

also can be rewritten as

$$\mathbf{r} = 2\mathbf{n}\frac{\mathbf{n} \cdot \mathbf{l}}{\mathbf{n} \cdot \mathbf{n}} - \mathbf{l} \tag{3.2}$$

The only difference is that equation (3.1) assumes the normal to be normalized, whereas equation (3.2) can be used for a normal with arbitrary non zero length.

The paper proposes that this fact can be used to compute specular highlights for Gouraud shaded objects. The highlight is computed as

$$I_s = (\mathbf{r} \cdot \mathbf{v})^s \tag{3.3}$$

where $\mathbf{v}$ is the vector pointing at the center of projection and s is used to model the specularity of the object, i.e. the size of the specular reflection on the surface.

This hybrid shading therefore uses Gouraud for the diffuse light, which implies that no vector interpolation is done. Instead intensities are linearly interpolated over the polygon. The diffuse light is computed as

$$I_d = \mathbf{n} \cdot \mathbf{l} \tag{3.4}$$

Figure 3.1: The Venus de Milo statue, shaded by the proposed hybrid of Gouraud for the diffuse light and accurate Phong specular light.

where **l** is the vector in the direction at the light source.

For the specular light the normal is linearly interpolated, but not normalized. This normal is then used for equation (3.2). If we assume that $\mathbf{v} = [0,0,-1]$ then we have

$$\mathbf{r} \cdot \mathbf{v} = l_z - 2n_z \frac{\mathbf{n} \cdot \mathbf{l}}{\mathbf{n} \cdot \mathbf{n}} \tag{3.5}$$

The result of using the hybrid shading approach is shown in figure 3.1.

It can be noted that the computation necessary is less expensive than the computation of Blinn's [13] formulation of specular light

$$I_s = (\mathbf{n} \cdot \mathbf{h})^s \tag{3.6}$$

The reason is that no normalization of the interpolated normal is necessary.

A new shading technique that avoids vector normalization that will produce Phong like diffuse light is also proposed in paper I. This technique also utilizes equation (3.2), but in another way. The idea is that if we could find a vector **v** which is exactly halfway between our interpolated, and un-normalized normal **n** and a vector **u**, which is normalized, then we could use equation(3.2) to reflect **u** around **v** to get $\mathbf{n}'$ which will be in the same directions as **n**, but it will have unit length. Hence, we have avoided vector normalization with the time consuming square root. The problem is that we will not find **v** without

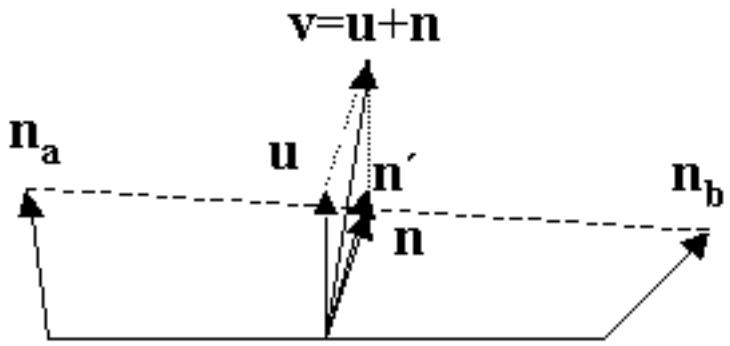

Figure 3.2: Construction of the new normal.

using a square root. However, we could obtain approximations of $\mathbf{v}$ in several ways. One easy and often sufficiently good way is to let $\mathbf{u}$ be the normal of the polygon, and to let $\mathbf{v} = \mathbf{u} + \mathbf{n}$. If $\mathbf{u}$ is close to $\mathbf{n}$, then $\mathbf{v}$ will be close to halfway in between. This approach is visualized in figure 3.2 The diffuse light can be computed by

$$\mathbf{n}' \cdot \mathbf{l} = \frac{2(\mathbf{v} \cdot \mathbf{l})(\mathbf{v} \cdot \mathbf{u})}{\mathbf{v} \cdot \mathbf{v}} - \mathbf{u} \cdot \mathbf{l} \tag{3.7}$$

Figure 3.3 shows the Venus de Milo statue shading using the proposed technique and figure 3.4 is shaded using phong shading. Both use Blinn's formulation of specular light. The renderings reveal that the visual difference between using the new method and Phong shading is neglectable. Moreover, the hybrid shading approach has identical highlights as Phong has. However, the diffuse light is identical to Gouraud shading.

It should be noted that both the normalization trick used for reflection shading as well as the hybrid shading approach could be implemented on a modern GPU. Moreover, both reflection shading and hybrid shading could be implemented using a polygon setup instead of a scanline setup. It is easy to see that equation (3.5) is quite similar to the equation (2.44) for computing the diffuse light using Phong shading.

3.1.1 An quadratic approximation of the Phong curve

Paper II proposes a quadratic interpolation scheme of the diffuse and specular intensity for each scan line. The main idea is to use Euler's method. It can be used to approximate a curve with a known start point and a given derivative at each point. The method works as follows: We begin with the intensity at the start point. In the next step the derivative is added to the start value. This is our new approximation, and as we proceed along the scan line the derivative is added to this new intensity value.

Figure 3.3: The Venus de Milo statue, shaded by the proposed reflection shading technique, for both the diffuse and specular light.

Figure 3.4: A Phong shaded Venus de Milo statue.

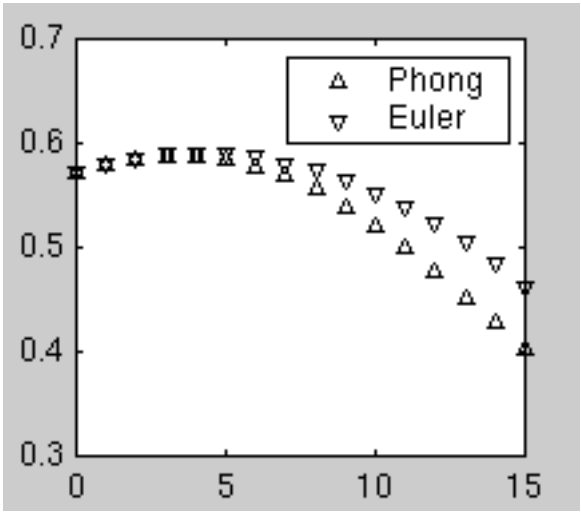

Figure 3.5: Difference in intensity between Phong and Euler shading along a scan line.

The equation for Phong shading on a scan line

$$\ldots = \frac{\mathbf{n} \cdot \mathbf{l}}{\| \mathbf{n} \|} = \frac{Ax + B}{\sqrt{Cx^2 + Dx + E}} = \frac{p}{\sqrt{q}} \tag{3.8}$$

can be differentiated

$$d\ldots = \frac{d}{dx}\frac{p}{\sqrt{q}} = \frac{1}{\sqrt{q}}\left(\frac{dp}{dx} - \frac{p}{2q}\frac{dq}{dx}\right) \tag{3.9}$$

This equation still contains a square root so it will not be efficient to compute it for each step. The paper proposes that the derivative is approximated by, a linear interpolation of the derivatives at the edges, over the scan line. Since $q = 1$ on the edges the reciprocal square root can be avoided.

The drawback using this method is that we will end up with another value than expected on the other edge as shown in figure 3.5. This occurs since two approximations are used and this will produce Mach-bands and must therefore be avoided.

The proposed solution is to linearly compensate for this error. It is shown that since colors are discrete values, the color obtained with Phong shading will not be much different than the color obtained by using this method as shown by figure 3.6. In fact the difference for reasonably highly tessellated objects is not more than one gray level when the corrected Euler interpolation is used. It is also shown that this approach also works for computing the specular light using Blinn's formulation. Figure 3.7 shows the Venus de Milo statue again, shaded using the proposed technique.

It is also shown that we could decide in advance whether Gouraud shading would produce the same discrete color value as Euler shading, and this therefore gives the opportunity to chose the appropriate method. For a software implementation we could save one addition in the inner loop. Nonetheless the cost for computing the difference is rather high.

The setup is given in the paper. It was shown in [33] that the setup could be simplified further. However, We have found that this approach can be simplified even further and the result is interesting since we will come up with the same result in a quite different way in a paper VII. This other way of doing the setup is by constructing a quadratic curve from three sample points, the intensities at the edges as well as the intensity in the middle. The intensity in the middle can be computed by evaluating the curve at $n = 1/2$.

In the paper it is shown that the derivatives at the edges are

$$d..a = \left(\frac{dp_a}{dx} - \frac{p_a}{2}\frac{dq_a}{dx}\right) \tag{3.10}$$

$$d..b = \left(\frac{dp_b}{dx} - \frac{p_b}{2}\frac{dq_b}{dx}\right) \tag{3.11}$$

where $..a = \mathbf{n}_a \cdot \mathbf{l}$, $..b = \mathbf{n}_b \cdot \mathbf{l}$, $dp_a = dp_b = ((\mathbf{n_b} - \mathbf{n_a})/n) \cdot \mathbf{l}$ and $dq_a = 2((\mathbf{n_a} \cdot \mathbf{n_b} - 1)/n)$. At the other edge we will have $dq_b = -dq_a$ and n is the number of pixels over the scanline. Moreover $ds = (d..b - d..a)/n$. However, a compensation for the error is used and $d© = (..n - ..b)/n$ and $t_0 = d..a - d©$

By using the equations above the intensity at the middle can be computed as

$$..n/2 = ..a + t_0(n/2) + ds((n/2)^2 - (n/2))/2 \tag{3.12}$$

If this euqation is simplified we end up with this equation

$$..n/2 = \left(\frac{5 - \mathbf{n}_a \cdot \mathbf{n}_b}{8}(\mathbf{n}_a + \mathbf{n}_b)\right) \cdot \mathbf{l} \tag{3.13}$$

The coefficients for the quadratic curve is found by solving the following system of equations

$$... = M\mathbf{c} \tag{3.14}$$

where

$$... = [I_L, I_M, I_N]^T \tag{3.15}$$

$$\mathbf{c} = [a, b, c]^T \tag{3.16}$$

The solution for $\mathbf{c} = M^{-1}...$is

$$a = 2I_L n^2 - 4I_M n^2 + 2I_N n^2 \tag{3.17}$$

$$b = -3I_L n + 4I_M n - I_N n \tag{3.18}$$

$$c = I_L \tag{3.19}$$

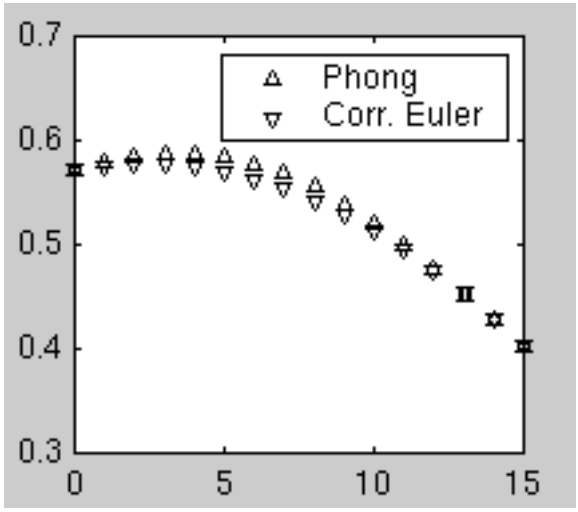

Figure 3.6: Difference in intensity between Phong and corrected Euler shading along a scan line.

where

$$\ldots = a\left(\frac{x}{n}\right)^2 + b\frac{x}{n} + c \tag{3.20}$$

This setup will be much more straightforward than previously proposed setups for quadratic interpolation along the scanline.

Another interesting result presented in paper II, which has not to our knowledge been presented before is that the normalization function q in equation (3.8) is symmetric, if we have normalized normals on the edges. Therefore, we do not have to compute q for the whole scan line since we can reuse these values for the second part of the scan line. Moreover, this means that we only need to compute $1/\sqrt{q}$ for the first half of the scan line and then use the same values for the second part. We could do that by shading the scan line from both ends simultaneously.

The quadratic interpolation approach can easily be extended to a polygon setup algorithm as shown in paper V, VI and VIII.

3.2 Papers III and IV: Improved Spherical Linear Interpolation

Paper III and IV propose two new ways of computing the intermediate normal using spherical linear interpolation (slerp). These two methods are typical scanline setup approaches and can not easily be extended to a polygon setup approach. Still, some attempts in that direction will be discussed in this thesis, which were not included in the original papers.

Figure 3.7: The Venus de Milo statue, shaded by the quadratic Euler shading technique, for both the diffuse and specular light.

3.2.1 Spherical Linear Interpolation

Spherical interpolation for shading was introduced by Kuijk and Blake [46]. Also Abbas et al. [2] have discussed this approach for hardware implementation. These approaches will not be fast without using table look ups. They both make use of the fact that the angle between **n** and **l**, where both are interpolated, can be computed by using trigonometric formulas.

The theory behind spherical interpolation or equal angle interpolation as it is sometimes called, was developed by Shoemake [64] and an interesting proof is given by Glassner [29]. The formula used for the interpolation of the normal over a scanline as shown in figure 3.8 is

$$\mathbf{n}(t) = \mathbf{n_a}\frac{\sin((1-t)\aleph)}{\sin(\aleph)} + \mathbf{n_b}\frac{\sin(t\aleph)}{\sin(\aleph)} \tag{3.21}$$

where $t \in [0,1]$, and $\aleph$is the angle between $\mathbf{n_a}$ and $\mathbf{n_b}$. It is shown in paper III that this equation can be rewritten as

$$\begin{aligned}\mathbf{n}(t) &= \mathbf{n_a}\cos(t\aleph) \\ &+ \frac{\mathbf{n_b} - \mathbf{n_a}(\mathbf{n_a}\cdot\mathbf{n_b})}{\sqrt{1-(\mathbf{n_a}\cdot\mathbf{n_b})^2}}\sin(t\aleph)\end{aligned} \tag{3.22}$$

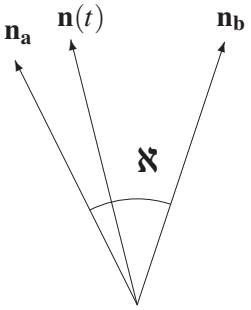

Figure 3.8: Spherical interpolation of the vector $\mathbf{n}(t)$ between the vectors $\mathbf{n_a}$ and $\mathbf{n_b}$.

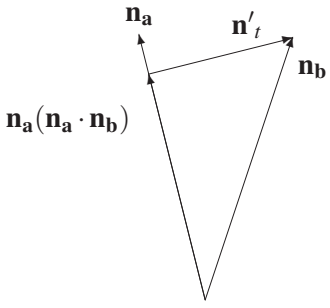

Figure 3.9: Gram-Schmidt orthogonalization.

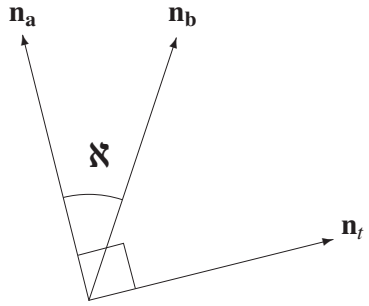

Figure 3.10: Construction of the tangent vector $\mathbf{n_t}$ orthogonal to $\mathbf{n_a}$ lying in the plane spanned by $\mathbf{n_a}$ and $\mathbf{n_b}$.

The second vector has an interesting interpretation. The numerator is the same as the vector we obtain by applying one step of Gram-Schmidt's orthogonalization algorithm [53], as shown in figure 3.9

$$\mathbf{n}'_t = \mathbf{n_b} - \frac{\mathbf{n_b} \cdot \mathbf{n_a}}{\| \mathbf{n_a} \|^2} \mathbf{n_a} \tag{3.23}$$

where $\| \mathbf{n_a} \|^2 = 1$, since $\mathbf{n_a}$ has unit length. Moreover, the denominator is in fact the norm of the numerator, thus giving a vector with unit length orthogonal to $\mathbf{n_a}$. The norm is

$$\begin{aligned} &\| \mathbf{n}'_t \| = \\ &\sqrt{(\mathbf{n_b} - \mathbf{n_a}(\mathbf{n_a} \cdot \mathbf{n_b})) \cdot (\mathbf{n_b} - \mathbf{n_a}(\mathbf{n_a} \cdot \mathbf{n_b}))} \\ &= \sqrt{1 - 2(\mathbf{n_a} \cdot \mathbf{n_b})^2 + 1(\mathbf{n_a} \cdot \mathbf{n_b})^2} \\ &= \sqrt{1 - (\mathbf{n_a} \cdot \mathbf{n_b})^2} \end{aligned} \tag{3.24}$$

The tangent vector $\mathbf{n_t}$ is orthogonal to $\mathbf{n_a}$ as shown in figure 3.10 and lies in the plane spanned by $\mathbf{n_a}$ and $\mathbf{n_b}$.

This result implies that we can rewrite the spherical interpolation formula as

$$\mathbf{n}(t) = \mathbf{n_a} \cos(t\,\text{א}) + \mathbf{n_t} \sin(t\,\text{א}) \tag{3.25}$$

This comes as no surprise since we know that for a unit circle we have: $x = \mathbf{e_1} \cos(\text{א})$ and $y = \mathbf{e_2} \sin(\text{א})$, where $\mathbf{e_1}$ and $\mathbf{e_2}$ are the basis vectors for the Carte-

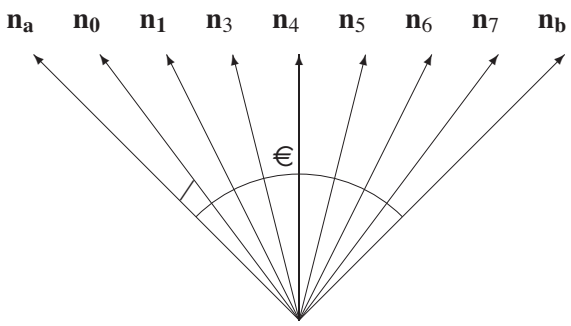

Figure 3.11: Linear normal interpolation yields different angels between the normals.

sian system. This well known result gives an interesting formulation of shading over a scanline using complex computation, which will be explained later.

3.2.2 Shading by Angular Interpolation

One of the mentioned drawbacks with Phong shading is that we have to normalize the normal at each pixel. Another effect of linear interpolation is that the angle between each normal on the scan line will not be the same. Clearly ⁄ is not equal to €in figure 3.11. The new method will solve these two problems.

First we need the angle between $\mathbf{n_a}$ and $\mathbf{n_b}$, which we denote $\aleph$

$$\aleph = \cos^{-1}(\mathbf{n_a} \cdot \mathbf{n_b}) \tag{3.26}$$

If a scan line has k pixels then the angle between each new normal denoted $K_\aleph$ is

$$K_\aleph = \frac{\aleph}{k} \tag{3.27}$$

Subsequently we rewrite equation (3.25) into a form which suites incremental scanline shading

$$\mathbf{n}(n) = \mathbf{n_a}\cos(nK_\aleph) + \mathbf{n_t}\sin(nK_\aleph) \tag{3.28}$$

where n is the n'th pixel along the scanline currently being shaded. As noted by Duff [20], it is possible to make the computation more efficiently by computing the dot product $\mathbf{n} \cdot \mathbf{l}$ directly instead of computing the interpolated normal and then computing the dot product.

This is also possible for spherical interpolation shading since

$$\begin{aligned} I_d(n) &= \mathbf{n_a} \cdot \mathbf{l}\cos(nK_\aleph) + \mathbf{n_t} \cdot \mathbf{l}\sin(nK_\aleph) \\ &= I_a\cos(nK_\aleph) + I_t\sin(nK_\aleph) \end{aligned} \tag{3.29}$$

where

$$I_a = \mathbf{n_a} \cdot \mathbf{l} \tag{3.30}$$

$$I_t = \mathbf{n_t} \cdot \mathbf{l} \tag{3.31}$$

This will make the computation noticeably faster since we will compute scalars instead if vectors. However, it will still be rather slow since the sin and cos functions are even more computationally expensive than the original division and square root in ordinary normalization. Obviously, this approach will only be useful if we can derive a faster way of evaluating equation (3.30) along a scanline.

3.2.3 De Moivre's Formula

Paper III shows that it is possible to compute spherical linear interpolation shading by using De Moivre's formula [49]. Complex numbers are defined in an orthonormal system where the basis vectors are $\mathbf{e_1} = 1$ and $\mathbf{e_2} = i$. The De Moivre's formula states that

$$(\cos(\aleph) + i\sin(\aleph))^n = \cos(n\aleph) + i\sin(n\aleph) \tag{3.32}$$

The right part of the formula looks very much like what we are trying to compute. We shall show that we can compute the left part instead, by using one complex multiplication. However, there are still some things to arrange before we can compute intensities instead of complex numbers.

Let Z be a complex number computed by

$$Z = \cos(K_\aleph) + i\sin(K_\aleph) \tag{3.33}$$

Furthermore, we treat complex numbers as if they were vectors in 2D-space, by defining a product which is similar to the ordinary dot product

$$(I_a, I_t) \bullet (\lfloor\ (Z), \rbrace\ (Z)) = I_a \lfloor\ (Z) + I_t \rbrace\ (Z) \tag{3.34}$$

Thus, we compute the diffuse intensity as:

$$I_d(n) = (I_a, I_t) \bullet Z^n \tag{3.35}$$

Finally, we prove that this will yield the desired result. Rewrite equation (3.35) by using equation 3.32

$$\begin{aligned} I_d(n) &= (I_a, I_t) \bullet Z^n \\ &= (I_a, I_t) \bullet (\cos(n\aleph) + i\sin(n\aleph)) \end{aligned} \tag{3.36}$$

Then, expand this equation using by equation (3.34)

$$I_d(n) = I_a \cos(nK_\aleph) + I_t \sin(nK_\aleph) \tag{3.37}$$

The cost for computing the intensity will be one complex multiplication and one dot product. Nevertheless, we will show that we can formulate the problem in another way which will be even faster to solve.

3.2.4 Chebyshev's Recurrence Relation

Paper IV shows that it is possible to solve equation (3.25) by using Chebyshev's recurrence relation [15]. This will make the computation in the inner loop much more effective than using the approach described in paper III. Nonetheless, the similarities are many and it is still a scanline setup approach using the angular interpolation method.

The Chebyshev's recurrence relation is

$$T_{n+1}(u) = 2uT_n(u) - T_{n-1}(u) \tag{3.38}$$

where $T_n(u)$ are Chebyshev polynomials [15] of the first kind that can be written as

$$T_n(u) = \cos(n\forall) = \cos(n\cos^{-1} u) \tag{3.39}$$

In order to obtain the Chebyshev recurrence for solving equation (3.30) let

$$u = \cos(K_{\aleph}) \tag{3.40}$$

$$\forall = K_{\aleph} \tag{3.41}$$

Then equation 3.39 gives

$$\cos(nK_{\aleph}) = \cos(n\cos^{-1} u) \tag{3.42}$$

This equation can be solved by equation (3.38). Since sin is just a phase shifted cos it will also be solved by the proposed equation as long as the angle is the same for both. Let

$$I_n = I_a\cos(nK_{\aleph}) + I_t\sin(nK_{\aleph}) \tag{3.43}$$

Starting with $n = 0$, we subsequently have

$$\begin{aligned} I_{n-1} &= I_{-1} \\ &= I_a\cos(-K_{\aleph}) + I_t\sin(-K_{\aleph}) \\ &= I_a\cos(K_{\aleph}) - I_t\sin(K_{\aleph}) \end{aligned} \tag{3.44}$$

and

$$I_0 = I_a\cos(0) + I_t\sin(0) = I_a \tag{3.45}$$

Now, we can put together our algorithm. We can also optimize it somewhat by removing the factor 2 from the loop.

$K_{\aleph} = \cos^{-1}(\mathbf{n_a}\cdot\mathbf{n_b})/k$
$I_{-1} = I_a\cos(K_{\aleph}) - I_t\sin(K_{\aleph})$
$I_0 = I_a$
$U = 2\cos(K_{\aleph})$
for $n := 1$ **to** $k-1$
    $I_{+1} = UI_0 - I_{-1}$
    $I_{-1} = I_0$
    $I_0 = I_{+1}$
end

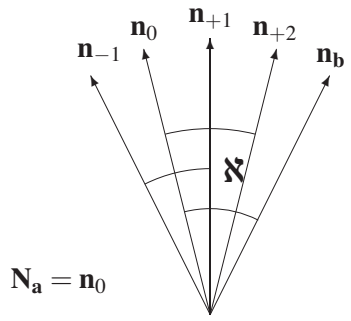

Figure 3.12: Geometrical interpretation of Chebyshev interpolation, where $\mathbf{n}_{-1}$ is reflected around $\mathbf{n}_0$ in order to obtain $\mathbf{n}_{+1}$ and $\mathbf{n}_{+2}$ is obtained by reflecting $\mathbf{n}_0$ around $\mathbf{n}_{+1}$.

To summarize our development so far; we have shown how to generate high quality shading along the scanlines in a polygon in only four basic operations per pixel, one multiplication, one add and two moves.

It should also be mentioned that it is discussed how this approach can be modified to step on pixel centers [35]. This is often not mentioned in papers and the modification of the algorithm is often rather straight forward.

3.2.5 Geometric Interpretation

The proposed recurrence has a very interesting geometric interpretation. In paper I, "reflection shading" was proposed, where normalization was avoided by reflecting a normalized vector around a vector half way between this normalized vector and a linearly interpolated vector. The reflected vector will thus be in the same direction as the linearly interpolated vector, but will have unit length. However, the vector in between was obtained by an approximation. Chebyshev's recurrence relation uses the same principle scheme, as we shall show. However, the vectors in between are computed exactly.

Let the first in between vector be $\mathbf{n}_0 = \mathbf{n_a}$. The vector to be reflected around this vector in order to obtain $\mathbf{n}_{+1}$ is $\mathbf{n_{-1}} = \mathbf{n_a}\cos(K_\aleph) - \mathbf{n_t}\sin(K_\aleph)$, as shown in figure 3.12. The equation for the reflected vector is

$$\mathbf{n}_{+1} = 2\mathbf{n}_0(\mathbf{n_0} \cdot \mathbf{n_{-1}}) - \mathbf{n}_{-1} \tag{3.46}$$

Note that $\mathbf{n_0} \cdot \mathbf{n_{-1}} = \cos(K_\aleph)$ so this equation is equivalent to equation (3.38). Hence, this is essentially what the Chebyshev's recurrence developed in paper IV does. The two moves in the algorithm comes from the fact that in the next

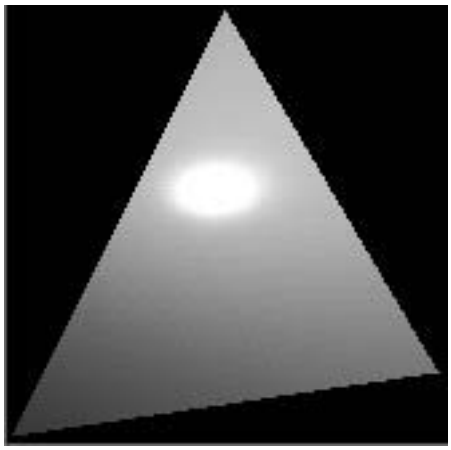

Figure 3.13: Phong shaded polygon.

step, as shown in figure 3.12, the vector $\mathbf{n}_{+2}$ is obtained by reflecting $\mathbf{n}_0$ around $\mathbf{n}_{+1}$. This process is then repeated for each new normal.

The similarity between the proposed shading method and Phong shading is shown in figure 3.13 and 3.14 for a very large polygon in order to make the small differences visible. It can be discussed which algorithm produces the best result, since neither method is based on a correct physical model. In our opinion the results from our method is at least as good as those from Phong's method. It can be noted that the highlight has different forms in the images. This is due to two different reasons. To some extent, it is due to the two different interpolation approaches that are used: angular interpolation and linear interpolation. The second reason is the fact that Phong in this case use a polygon setup approach, while the other use a scanline setup approach.

When the polygons are smaller, like in figure 3.15 and 3.16 for a 800 polygon torus, the difference is barely noticeable.

We have studied the computational efficiency of the proposed algorithm in an experiment. The timing was done in such a way that the efficiency of the two algorithms was measured on an average scanline length basis. This was done by increasing the size of the polygon mesh linearly. The average scanline length was computed for each size.

Both Phong and the proposed method computes the diffuse and specular light separately. This yields two Chebyshev recurrences along the scanline for the proposed method. For Phong shading we need two linear interpolations of the two dot products and one quadratic interpolation of the normalization, as well as the division and square root included in the normalization. These interpolations are done by using bi-variate polynomials, i.e. the interpolation is done incrementally over the whole polygon.

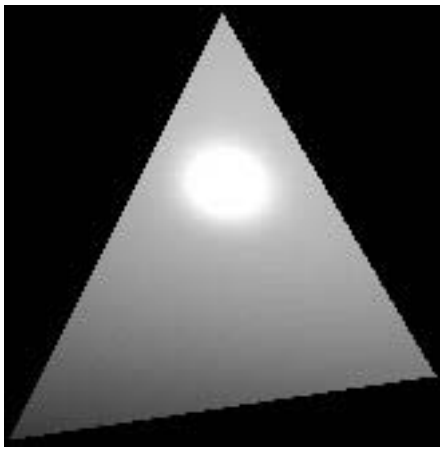

Figure 3.14: Polygon shaded using the proposed technique for both the diffuse and specular light.

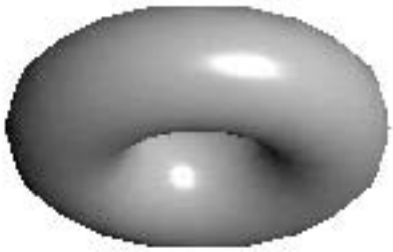

Figure 3.15: Phong shaded Torus.

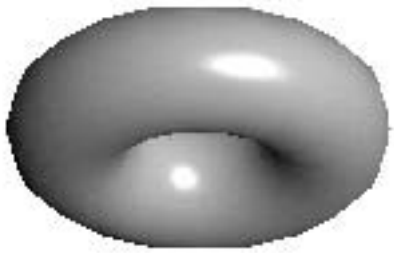

Figure 3.16: A Torus shaded using the proposed technique for both the diffuse and specular light.

Two dimensional tables were used to speed up setup for the proposed method. In paper III it is shown that equation (3.28) can be written as

$$\begin{aligned}\mathbf{n}(t) = \mathbf{n_a}\cos(n\aleph/k) \\ + \frac{\mathbf{n_b} - \mathbf{n_a}(\mathbf{n_a}\cdot\mathbf{n_b})}{\sin\aleph}\sin(n\aleph/k)\end{aligned} \tag{3.47}$$

Therefore, one table for $\cos(n\aleph/k)$ was used and another table was used for $\sin(n\aleph/k)/\sin\aleph$ The variables k and $\aleph$ were used as look up variables for the table.

A number of 200 equal sized triangles were rotated clockwise in 5000 steps. The total rendering time was measured. Furthermore, the rendering time of a rasterizer doing nothing but stepping over the pixels was also measured. By taking the difference between this fake rasterizer and the two algorithms in question, we were able to get a fairly good timing of the shading algorithms themselves. The result is shown in figure 3.17. The average scanline length can be less than one since two small polygons can occupy the same pixel. However, only one of them will set the color of the pixel. For very small polygons, the proposed method is very efficient, since we do not need to do the setup when only one pixel is rasterized. Therefore, it will be slightly faster than Phong. For small polygons the result is more or less equal. However, for larger polygons, the proposed method becomes noticeably faster, as shown in the same figure.

Vector interpolation also becomes very fast using the proposed technique and look-up tables. Figure 3.18 shows that the proposed technique quickly becomes much faster than ordinary vector interpolation with normalization. A

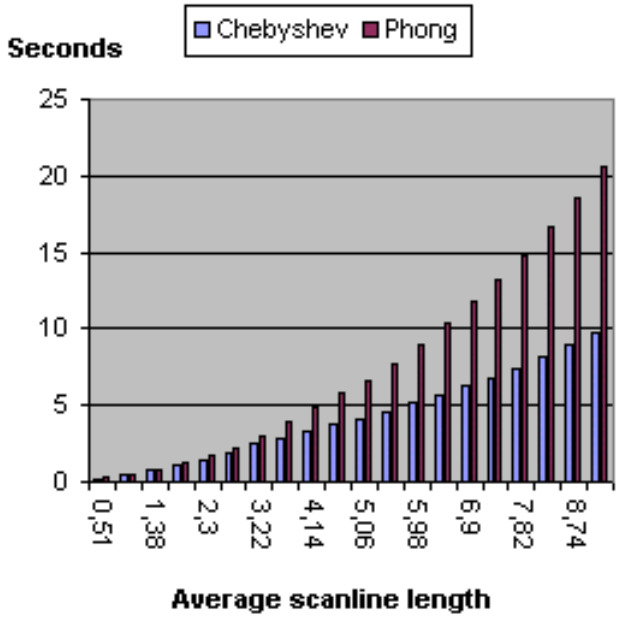

Figure 3.17: Comparison between Phong shading and the proposed method using Chebyshev's recurrence relation.

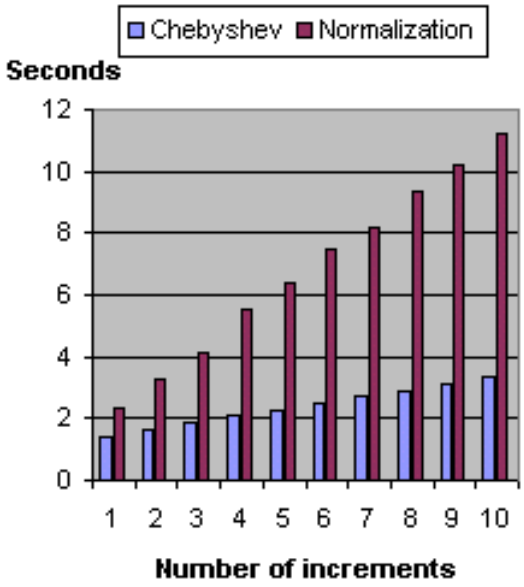

Figure 3.18: Comparison between slerp vector interpolation using Chebyshev's recurrence relation and ordinary vector interpolation using normalization.

number of ten million setups and interpolations were done for different number of increments.

3.2.6 Simplifications

There are several more simplifications that could be done in order to speed up the computation. First of all it can be noted that the angles between the normals at the vertices are not changed when the normals are rotated. These can therefore be precomputed. Moreover, the normalization of the tangent vectors on the edges can be precomputed since the length of the tangent will not be affected by rotation. For example, the norm of the tangent on the edge between $\mathbf{n_0}$ and $\mathbf{n_1}$ can be precomputed as

$$\Leftarrow = \mathbf{n_0} \cdot \mathbf{n_1} \tag{3.48}$$

$$\rightarrow = \frac{1}{\sqrt{1 - \Leftarrow^2}} \tag{3.49}$$

Subsequently the tangent vector is computed as

$$\mathbf{n}_t = \rightarrow(\mathbf{n_1} - \mathbf{n_0}\Leftarrow) \tag{3.50}$$

3.2.7 When the angle is small

There will be a problem, when $\mathbf{n_0}$ and $\mathbf{n_1}$ are equal or close to being equal. Then the tangent vector will have a $\rightarrow$close to infinity, this will lead to problems. In this case $\Leftarrow$will be close to one. However, this problem can easily be handled. If $\Leftarrow$is over a certain threshold value the setup is changed to

$T_{-1} = I_a$
$T = I_a$
$U = 2$

This will make sure that $T = I_a$ in the inner loop, since: $T_{+1} = UT - T_{-1} = 2I_a - I_a = I_a$.

3.2.8 Evaluating the dot products for setup along edges

Some attempts were done to move the work from the scanline setup to the polygon setup. These are not included in paper IV, but we show here some possible changes that can be done .

Unfortunately, the setup for each scanline requires that some trigonometric functions are evaluated. However, tables were used in the paper in order to speed up the setup. Moreover, it turns out that it is possible also to compute most of this scanline setup by using Chebyshev's recurrence relation. This will

yield a more complex polygon setup, but the setup for each scan line will be very fast.

The dot product $\mathbf{n_a} \cdot \mathbf{l}$ can be evaluated using Chebyshev's recurrence relation on the edges. This is also true for the $\mathbf{n}_t$ vector. The equation (3.47) shows that it is necessary to also compute $\mathbf{n_b} \cdot \mathbf{l}$ on the edge as well as $\mathbf{n_a} \cdot \mathbf{n_b}$. Unfortunately this dot product depends on two variables and must be treated differently.

3.2.9 Upper part of the polygon

First we need the angles at both edges for each step. For a triangle oriented as in figure 3.19 with upper vertex on relative scan line 0, middle vertex on relative scan line y_1 and bottom vertex on relative scan line y_2 the angles are computed as

$$K_{\diagup} = \frac{\cos^{-1}(\mathbf{n_0} \cdot \mathbf{n}_2)}{y_2} \tag{3.51}$$

$$K_{\in} = \frac{\cos^{-1}(\mathbf{n_0} \cdot \mathbf{n_1})}{y_1} \tag{3.52}$$

For the orientation in figure 3.20 the angles are computed as

$$K_{\diagup} = \frac{\cos^{-1}(\mathbf{n_0} \cdot \mathbf{n_1})}{y_1} \tag{3.53}$$

$$K_{\in} = \frac{\cos^{-1}(\mathbf{n_0} \cdot \mathbf{n}_2)}{y_2} \tag{3.54}$$

The dot product between the normals on the edges can be evaluated as

$$\mathbf{n_a}(n) \cdot \mathbf{n_b}(n) \tag{3.55}$$

where

$$\mathbf{n_a}(n) = \mathbf{n_0}\cos(nK_{\diagup}) + \mathbf{n_{t_a}}\sin(nK_{\diagup}) \tag{3.56}$$

$$\mathbf{n_b}(n) = \mathbf{n_0}\cos(nK_{\in}) + \mathbf{n_{t_b}}\sin(nK_{\in}) \tag{3.57}$$

Note, that $\mathbf{n_0} \cdot \mathbf{n_0} = 1$ and $\mathbf{n_0} \cdot \mathbf{n_{t_a}} = 0$ as well as $\mathbf{n_0} \cdot \mathbf{n_{t_b}} = 0$, therefore the dot product becomes

$$\begin{aligned}\mathbf{n_a}(n) \cdot \mathbf{n_b}(n) = &\cos(nK_{\diagup})\cos(nK_{\in}) + \\ &(\mathbf{n_{t_a}} \cdot \mathbf{n_{t_b}})\sin(nK_{\diagup})\sin(nK_{\in})\end{aligned} \tag{3.58}$$

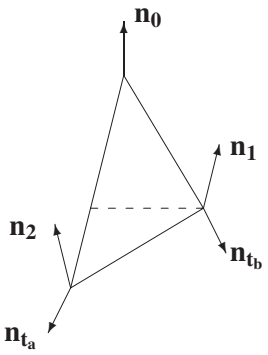

Figure 3.19: Normals and tangent vectors for computing the dot product of the normals at the edges for the upper part of a triangle.

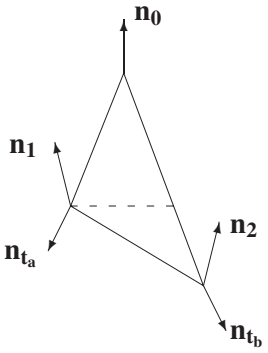

Figure 3.20: Normals and tangent vectors for computing the dot product of the normals at the edges for the upper part of a triangle with another orientation.

Equation (3.58) can be rewritten using the following trigonometric identities

$$\cos(\aleph)\cos(\forall) = (\cos(\aleph+\forall)+\cos(\aleph-\forall))/2 \tag{3.59}$$
$$\cos(\aleph)\sin(\forall) = (\sin(\aleph+\forall)-\sin(\aleph-\forall))/2 \tag{3.60}$$
$$\sin(\aleph)\cos(\forall) = (\sin(\aleph+\forall)+\sin(\aleph-\forall))/2 \tag{3.61}$$
$$\sin(\aleph)\sin(\forall) = (\cos(\aleph-\forall)-\cos(\aleph+\forall))/2 \tag{3.62}$$

By using equations (3.59) through (3.62) we obtain

$$\begin{aligned}\mathbf{n_a}(n)\cdot\mathbf{n_b}(n) = \\ 1/2(\cos(nK_{\Im})+\cos(nK_{\wp}))+ \\ (\mathbf{n_{t_a}}\cdot\mathbf{n_{t_b}})/2(\cos(nK_{\wp})-\cos(nK_{\Im}))\end{aligned} \tag{3.63}$$

where

$$K_{\Im} = K_{\diagup} + K_{\in} \tag{3.64}$$
$$K_{\wp} = K_{\diagup} - K_{\in} \tag{3.65}$$

Rearrange the equation in the following way

$$\begin{aligned}\mathbf{n_a}(n)\cdot\mathbf{n_b}(n) = \\ \cos(nK_{\Im})(1-\mathbf{n_{t_a}}\cdot\mathbf{n_{t_b}})/2+ \\ \cos(nK_{\wp})(1+\mathbf{n_{t_a}}\cdot\mathbf{n_{t_b}})/2\end{aligned} \tag{3.66}$$

This equation can be separated into two equations which each can be evaluated by Chebyshev's recurrence equation, and the sum of them will be equal to the dot product in question

$$\mathbf{n_a}(n)\cdot\mathbf{n_b}(n) = N_A(n)+N_B(n) \tag{3.67}$$

where

$$N_A(n) = \cos(nK_{\Im})(1-\mathbf{n_{t_a}}\cdot\mathbf{n_{t_b}})/2 \tag{3.68}$$
$$N_B(n) = \cos(nK_{\wp})(1+\mathbf{n_{t_a}}\cdot\mathbf{n_{t_b}})/2 \tag{3.69}$$

For the setup it is necessary to evaluate $\cos(K_{\Im})$, $\sin(K_{\Im})$, $\cos(K_{\wp})$ and $\sin(K_{\wp})$. Since we already have evaluated $\cos(K_{\diagup})$, $\sin(K_{\diagup})$, $\cos(K_{\in})$ and $\sin(K_{\in})$, we can easily compute them by using the following trigonometric identities

$$\cos(\aleph+\forall) = \cos(\aleph)\cos(\forall)-\sin(\aleph)\sin(\forall) \tag{3.70}$$
$$\cos(\aleph-\forall) = \cos(\aleph)\cos(\forall)+\sin(\aleph)\sin(\forall) \tag{3.71}$$
$$\sin(\aleph+\forall) = \sin(\aleph)\cos(\forall)+\cos(\aleph)\sin(\forall) \tag{3.72}$$
$$\sin(\aleph-\forall) = \sin(\aleph)\cos(\forall)-\cos(\aleph)\sin(\forall) \tag{3.73}$$

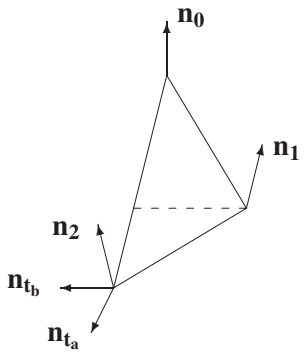

Figure 3.21: Normals and tangent vectors for computing the dot product of the normals at the edges for the lower part of a triangle.

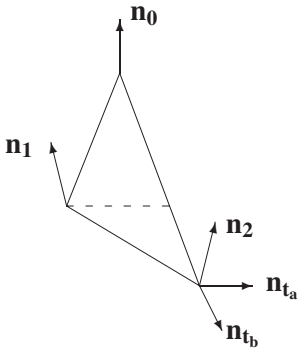

Figure 3.22: Normals and tangent vectors for computing the dot product of the normals at the edges for the lower part of a triangle with another orientation.

3.2.10 Lower part of the polygon

For the lower part of the polygon another setup is needed. If the polygon is oriented as in figure 3.21, then the left edge must be evaluated from scanline y_1 in the following way

$$\begin{aligned}\mathbf{n_a}(n) =& \mathbf{n_0}\cos((n+y_1)K_{\diagup}) + \\ & \mathbf{n_{t_a}}\sin((n+y_1)K_{\diagup})\end{aligned} \tag{3.74}$$

$$\mathbf{n_b}(n) = \mathbf{n_1}\cos(nK_{\in}) + \mathbf{N_{t_b}}\sin(nK_{\in}) \tag{3.75}$$

where

$$K_{\diagup} = \frac{\cos^{-1}(\mathbf{n_0}\cdot\mathbf{n_1})}{y_1} \tag{3.76}$$

$$K_{\in} = \frac{\cos^{-1}(\mathbf{n_1}\cdot\mathbf{n}_2)}{(y_2-y_1)} \tag{3.77}$$

This yields the following equations

$$\begin{aligned}&\mathbf{n_a}(n)\cdot\mathbf{n_b}(n) = \\ &\mathbf{n_0}\cdot\mathbf{n_1}\cos(K_{\diagup}\,y_1)\cos(nK_{\diagup})\cos(nK_{\in}) + \\ &\mathbf{n_0}\cdot\mathbf{n_{t_b}}\cos(K_{\diagup}\,y_1)\cos(nK_{\diagup})\sin(nK_{\in}) - \\ &\mathbf{n_0}\cdot\mathbf{n_1}\sin(K_{\diagup}\,y_1)\sin(nK_{\diagup})\cos(nK_{\in}) - \\ &\mathbf{n_0}\cdot\mathbf{n_{t_b}}\sin(K_{\diagup}\,y_1)\sin(nK_{\diagup})\sin(nK_{\in}) + \\ &\mathbf{n_{t_a}}\cdot\mathbf{n_1}\cos(K_{\diagup}\,y_1)\sin(nK_{\diagup})\cos(nK_{\in}) + \\ &\mathbf{n_{t_a}}\cdot\mathbf{n_{t_b}}\cos(K_{\diagup}\,y_1)\sin(nK_{\diagup})\sin(nK_{\in}) + \\ &\mathbf{n_{t_a}}\cdot\mathbf{n_1}\sin(K_{\diagup}\,y_1)\cos(nK_{\diagup})\cos(nK_{\in}) + \\ &\mathbf{n_{t_a}}\cdot\mathbf{n_{t_b}}\sin(K_{\diagup}\,y_1)\cos(nK_{\diagup})\sin(nK_{\in})\end{aligned} \tag{3.78}$$

Once again equation (3.78) can be rewritten using equations (3.59) through (3.62). A similar approach can be used if the polygon is oriented the other way as in figure 3.22. However, as can be seen already, these computations are a lot messier than for the upper part of the triangle. An attractive alternative may be to treat the lower part as an upside down upper part. Hence, the lower part should be rasterized in the reversed scanline order, assuming that the application allows arbitrary scanline order in the rendering. Another solution would be to precompute the involved dot products as they are rotation invariant. One reviewer commented this setup as “ghastly” and therefore it was not included in the paper.

3.2.11 Approximation of trigonometric functions in the setup

The trigonometric functions $\cos^{-1}$, cos and sin that are necessary in the setup for each polygon and each scanline will more or less eat up the gain in speed

obtained by using Chebyshev's recurrence relation unless we find efficient ways to compute them too.

In paper IV table look-up's where used. However, it is also possible to obtain satisfactory approximations of the trigonometric functions involved in the setup. Good approximations using Maclaurin polynomials is one way to speed up the setup noticeably. It turns out that just a few terms in the approximations are necessary depending on the shape of the object. If the object is tessellated without large angles between polygons, then the angle between the normals of the polygon will be relatively small. And the angles between each interpolated normal will be even smaller.

Note, that the error will be propagated to each pixel, due to the reflection as explained in the geometric interpretation section. This might seem to be a great disadvantage when using these approximations. However, if the edges of the polygons have a limited number of pixels then the error will not become that great before the end of the line. Moreover, if the edge has many pixels, the accuracy of the approximation of the angle between each pixel will be better. Thus, the error will, even if duplicated many times, not become that great. This can be seen in figure 3.24 and 3.25.

Maclaurin series of the trigonometric functions in question are

$$\cos^{-1}(\Leftrightarrow) = \frac{\Theta}{2} - (\Leftarrow + \frac{\Leftarrow^3}{6} + \frac{3\Leftarrow^5}{40} + \frac{15\Leftarrow^7}{336}...) \tag{3.79}$$

$$\cos(\aleph) = 1 - \frac{\aleph^2}{2} + \frac{\aleph^4}{24} - \frac{\aleph^6}{720} + ... \tag{3.80}$$

$$\sin(\aleph) = \aleph - \frac{\aleph^3}{6} + \frac{\aleph^5}{120} - \frac{\aleph^7}{5040} + ... \tag{3.81}$$

Over the scan line we have

$$\Leftarrow = \mathbf{n_a} \cdot \mathbf{n_b} \tag{3.82}$$

If the angle between the normal is small then $\Leftarrow$ is close to one, i.e. $\aleph$ is close to zero. This implies that only the few first terms will have an noticeable impact on the result for equation (3.80), depending on how small the angle $\aleph$ is.

If $\aleph$ is sufficiently small, i.e. only the first two terms have an impact on the result, then $\Leftarrow$ is approximately equal to the Macluarin expansion of $\cos(\aleph)$

$$\Leftarrow = 1 - \frac{\aleph^2}{2} \Rightarrow \tag{3.83}$$

$$\frac{\aleph^2}{2} = 1 - \Longleftrightarrow \tag{3.84}$$

$$\aleph = \sqrt{2 - 2\Leftarrow} \tag{3.85}$$

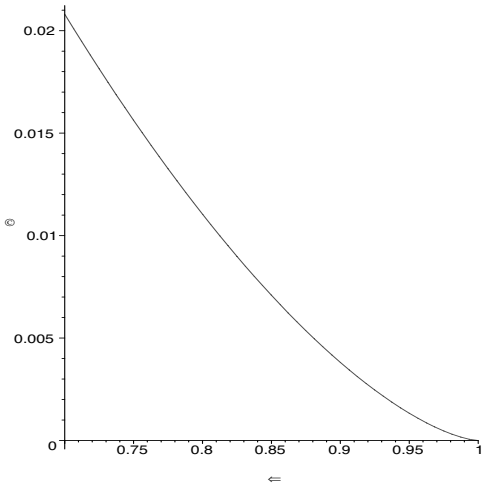

Figure 3.23: Difference between $\cos^{-1}$ and the proposed approximation.

This gives an excellent approximation for our needs. Moreover it will be faster than evaluating $\cos^{-1}$ and the square root can also be avoided, improving the efficiency even further.

The difference between $\cos^{-1}$ and the proposed approximation is shown in figure 3.23

$$\text{©} = \cos^{-1}(\Leftrightarrow) - \sqrt{2-2\Leftarrow} \tag{3.86}$$

In the algorithm, both sin and *cos* of $\aleph/k$ are computed. We derive approximating equations for them as well, using their Maclaurin expansions.

As stated earlier, if $\aleph$is mall we can use the two first terms in equation 3.80 in order to compute cos

$$\cos\frac{\aleph}{k} \approx 1 - \frac{\aleph^2}{2k^2} = 1 + \frac{\Leftarrow 1}{k^2} \tag{3.87}$$

Hence, the square root disappears. The difference ©between $\cos(\cos^{-1}\Leftrightarrow/k)$ and $1+\frac{\Leftarrow 1}{k^2}$ is shown in figure 3.24.

If the angle is small we only need the first two or even a single term in equation (3.81). In equation (3.47) there is a sin both in the numerator and the denominator. We will use two terms of the Maclaurin expansion for both, yielding

$$\frac{\sin(\cos^{-1}(\Leftrightarrow/k)}{\sin(\cos^{-1}(\Leftrightarrow)} \approx \frac{(3k^2 - 1 + \Leftrightarrow}{k^3(2+\Leftrightarrow} \tag{3.88}$$

The difference between the left and right side of the equation is shown in figure 3.25.

As can be seen from these figures the error is very small. Therefore, the proposed approximations will be sufficient for computing a correct setup as

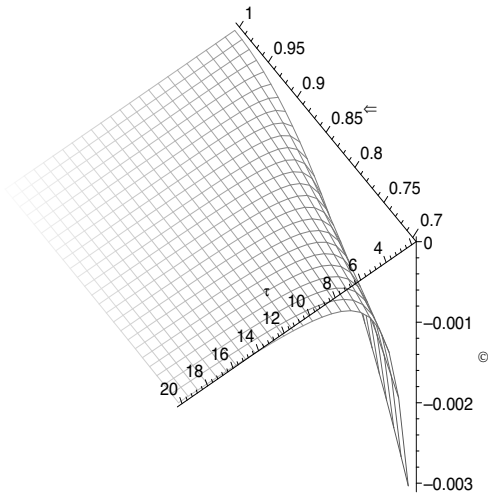

Figure 3.24: Difference between $\cos(\cos^{-1} \Leftrightarrows k)$ and the proposed approximation in equation (3.87).

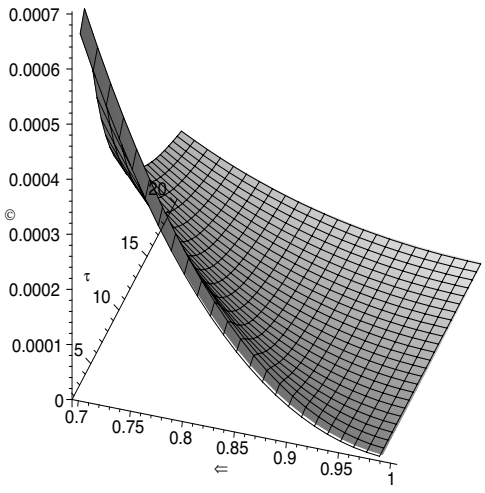

Figure 3.25: Difference between the trigonometric functions and approximation in equation (3.88).

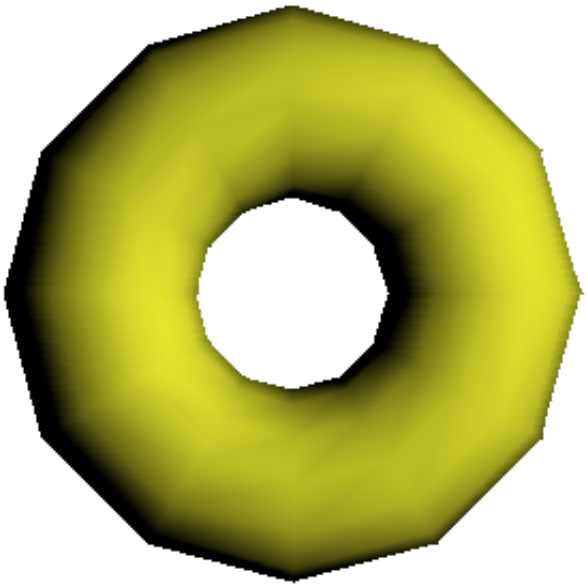

Figure 3.26: A torus shaded using quadratic interpolation of the diffuse intensity.

long as the normals on the vertices have a small angle between them, i.e. $\measuredangle$is not too small. In the figures it is clear that the error is rather small even for angles as large as 45^o degrees.

3.3 Papers V through VIII: Polygon setup algorithms

Polygon setup algorithms have a setup that computes all variables needed in the forward differences that are computed incrementally along the edges as well as along the scanlines.

Gouraud [32] used bilinear interpolation of the colors at the vertices. If the intensities are interpolated, then only one addition per pixel is necessary to achieve this type of shading. The intensity over the polygon in screen space can be considered as a plane in $(x, y, \ldots)$-space, where x and y are the screen coordinates, and $\ldots$is the intensity. The bi-variate quadratic polynomial, on the other hand, will form a smooth surface in $(x, y, \ldots)$-space. The result is shown in figure 3.26. The torus in the figure has relatively few polygons in order to enhance the Mach bands. Nevertheless, the Mach bands are less visible than for linear interpolation (Gouraud) shown in figure 3.27.

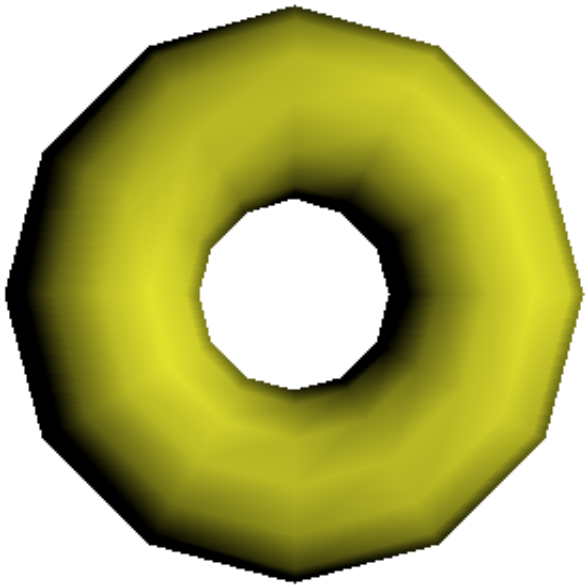

Figure 3.27: A torus shaded using linear interpolation of the diffuse intensity.

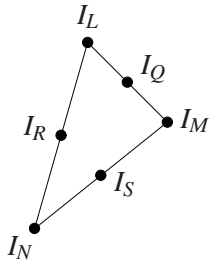

Figure 3.28: Sample points needed for quadratic shading.

3.3.1 Quadratic Shading

One type of quadratic shading was introduced by Bishop and Weimer [11]. They use a Taylor series expansion of Duff's equation to obtain a second order bi-variate polynomial. It could be evaluated by only two additions per pixel along a scanline. This quadratic interpolation scheme will produce a second order polynomial surface in $(x, y, \ldots)$-space.

Kappel [41] uses a surface fitting technique based on Powell-Sabin [58] quadratic interpolation using the Cendes-Wong [17] formulae. The purpose of this approach is to eliminate Mach bands by generating a C^1 continuous surface over the polygon. This is possible by subdividing the triangle into smaller triangles. Kirk et al. [43] solves this problem by fitting a surface to a triangle, where the intensity of the vertices of adjacent triangles are the same, and the first derivative of the intensity at the edges tend toward being continuous.

Kirk and Voorhies [44] adopt another approach to quadratic shading which is somewhat different from previous approaches. They show that quadratic interpolation could be setup by fitting a second order surface to six points, yielding a polynomial with six unknown coefficients which must be determined. The sample points are the vertices and edge mid points of a triangle, as shown in Fig 3.28. A quadratic shading function is defined by

$$\ldots = Ax^2 + By^2 + Cxy + Dx + Ey + F \tag{3.89}$$

Kirk and Voorhies do not show how these coefficients are computed, neither do Saxe et al. [59] who use this type of quadratic interpolation in Pixel-Planes 5 to display radiosity illuminated models [28].

Seiler [62] proposes a simple and fast way to setup the coefficients for this type of quadratic shading. How the computation can be done is also shown by Abbas et al. [3], [4] but they do not try to make the computation as efficient as Seiler does. Both methods do not cover special cases which will lead to division by zero. This problem however, is solved by Lee and Jen [47] who have simplified Seiler's approach.

In paper V it is shown how the methods by Seiler, Abbas et al. and Lee and Jen can be simplified further in order to make the computation of the coefficients as efficient as possible.

In general the coefficients can be obtained by setting up a system of equa-

tions using equation (3.89) and the relative coordinates of the sample points

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 & 1 \\ x_2^2 & y_2^2 & x_2 y_2 & x_2 & y_2 & 1 \\ \left(\frac{x_1}{2}\right)^2 & \left(\frac{y_1}{2}\right)^2 & \frac{x_1}{2}\frac{y_1}{2} & \frac{x_1}{2} & \frac{y_1}{2} & 1 \\ \left(\frac{x_2}{2}\right)^2 & \left(\frac{y_2}{2}\right)^2 & \frac{x_2}{2}\frac{y_2}{2} & \frac{x_2}{2} & \frac{y_2}{2} & 1 \\ \left(\frac{x_1+x_2}{2}\right)^2 & \left(\frac{y_1+y_2}{2}\right)^2 & \frac{x_1+x_2}{2}\frac{y_1+y_2}{2} & \frac{x_1+x_2}{2} & \frac{y_1+y_2}{2} & 1 \end{bmatrix} \tag{3.90}$$

Relative coordinates are preferable since it will yield a system which is easier to solve, as it will contain more zeroes. The relative coordinates are obtained by shifting the triangle so that the top vertex has coordinate $(0,0)$. The middle vertex will have coordinate (x_1, y_1), and the bottom most vertex will be (x_2, y_2). This implies that the vertices has to be sorted in the y direction. Note that, these operations are done in screen space and it is here presumed that the top left corner is coordinate $(0,0)$. The transformation is shown in Fig. 3.29.

The following substitutions is used in order to shift the triangle and to obtain the relative coordinates

$$x_1 = x_b - x_a \tag{3.91}$$
$$y_1 = y_b - y_a \tag{3.92}$$
$$x_2 = x_c - x_a \tag{3.93}$$
$$y_2 = y_c - y_a \tag{3.94}$$

Finally, the system of equations to solve is

$$M\mathbf{c} = \ldots \tag{3.95}$$

where

$$\ldots = [I_L, I_M, I_N, I_Q, I_R, I_S]^T \tag{3.96}$$
$$\mathbf{c} = [A, B, C, D, E, F]^T \tag{3.97}$$

In order to speed up the computation of coefficients the following intermediate values are used by Seiler

$$T = I_L + I_M - 2I_Q \tag{3.98}$$
$$U = I_L + I_N - 2I_R \tag{3.99}$$
$$V = I_L + I_S - I_R - I_Q \tag{3.100}$$
$$G = 2I_Q - 2I_L - T \tag{3.101}$$
$$H = 2I_R - 2I_L - U \tag{3.102}$$
$$i = x_2/x_1 \tag{3.103}$$
$$j = y_1/y_2 \tag{3.104}$$
$$k = 1 - ij \tag{3.105}$$

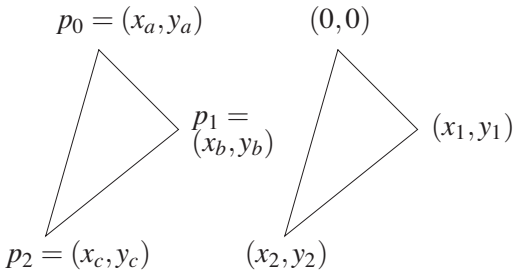

Figure 3.29: A polygon with sorted vertices at the left and the same polygon with shifted vertices at the right.

Then the coefficients are computed as follows

$$A = 2(T + Uj^2 - 2Vj)/(k^2 x_1 x_1) \tag{3.106}$$
$$B = 2(U + Ti^2 - 2Vi)/(k^2 y_2 y_2) \tag{3.107}$$
$$C = 4(2V - Ti - Uj - Vk)/(k^2 x_1 y_2) \tag{3.108}$$
$$D = (G - Hj)/(kx_1) \tag{3.109}$$
$$E = (H - Gi)/(ky_2) \tag{3.110}$$
$$F = I_L \tag{3.111}$$

It should be pointed out that x_1 could be zero. Then, the correct solution for that case must be used in order to compute the coefficients correctly. Also, note that it is possible to break out the constant 2 from equation 3.101 and equation 3.102, which will save two multiplications.

3.3.2 The method by Abbas et al.

If the same notation used by Seiler for the vertices is also used for the equations proposed by Abbas et al. then their equations becomes

$$T = I_M + I_L - 2I_Q \tag{3.112}$$
$$U = I_N + I_L - 2I_R \tag{3.113}$$
$$V = 4(I_L + I_S - I_Q - I_R) \tag{3.114}$$
$$G = 4I_Q - 3I_L - I_M \tag{3.115}$$
$$H = 4I_R - 3I_L - I_N \tag{3.116}$$

Note that if (3.98) is substituted into (3.101) and (3.99) is substituted into (3.102) as proposed by Seiler, then (3.115) and (3.116) are obtained. This is computationally more efficient, and was therefore used later in the comparison

of methods. The coefficients according to Abbas et al. are:

$$A = \frac{(4Ty_2 - Vy_1)E_{32} - (4Uy_1 - Vy_2)E_{23}}{E_{32}D_{23} - E_{32}D_{32}} \tag{3.117}$$

$$C = \frac{(4Uy_1 - Vy_2)D_{23} - (4Ty_2 - Vy_1)D_{32}}{E_{32}D_{23} - E_{32}D_{32}} \tag{3.118}$$

$$B = \frac{2T - Ax_1^2 - Cx_1y_1}{y_1^2} \tag{3.119}$$

$$D = \frac{Gy_2 - Hy_1}{x_1y_2 - y_1x_2} \tag{3.120}$$

$$E = \frac{Hx_1 - Gx_2}{x_1y_2 - y_1x_2} \tag{3.121}$$

$$F = I_L \tag{3.122}$$

where they use the following notation for

$$E_{23} = x_1y_1y_2 - y_1^2x_2 \tag{3.123}$$

$$E_{32} = y_1x_2y_2 - x_1y_2^2 \tag{3.124}$$

$$D_{23} = 2x_1^2y_2 - 2x_1y_1x_2 \tag{3.125}$$

$$D_{32} = 2y_1x_2^2 - 2x_1x_2y_2 \tag{3.126}$$

Some simplifications are possible. First note, that the denominators in (3.117) and (3.118) are equal, and the denominators in (3.120) and (3.121) are equal as well. Let

$$F_1 = \frac{1}{x_1y_2 - x_2y_1} \tag{3.127}$$

$$F_2 = \frac{1}{E_{32}D_{23} - E_{32}D_{32}} \tag{3.128}$$

then it can be shown that $F_2 = -1/2F_1^3$. Also note, that the factors $(4Ty_2 - Vy_1)$ and $(4Uy_1 - Vy_2)$ in (3.117) also appears in (3.118). There are, however, still many multiplications that could be saved by the following substitution

$$\mathrm{K} = x_1y_2 - x_2y_1 \tag{3.129}$$

This equation has got an interesting interpretation. It is essential to determine the orientation of the triangle in order to be able to rasterize it properly. The orientation of a polygon could be determined by computing the cross product, or by computing the double area of the polygon

$$\mathrm{K} = (x_a - x_c)(y_b - y_c) - (y_a - y_c)(x_b - x_c) \tag{3.130}$$

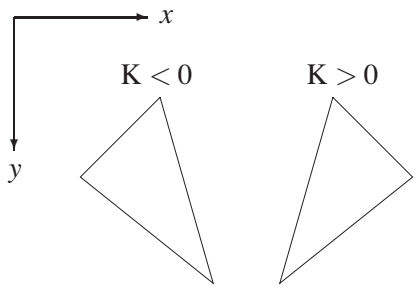

Figure 3.30: A left and right oriented triangle.

Using the substitutions in (3.91) through (3.94) yields equation (3.129). Hence, the sign of K will tell how the triangle is oriented as shown in Fig. 3.30. If $\mathrm{K} = 0$ then the vertices are on a straight line, and the area is equal to zero.

The equations for the coefficients can, by using this substitution, be rewritten in the following simplified form

$$A = (C_1E_{32} - C_2E_{23})F_2 \tag{3.131}$$
$$C = (C_2D_{23} - C_1D_{32})F_2 \tag{3.132}$$
$$B = (2T - Ax_1^2 - Cx_1y_1)/y_1^2 \tag{3.133}$$
$$D = (Gy_2 - Hy_1)F_1 \tag{3.134}$$
$$E = (Hx_1 - Gx_2)F_1 \tag{3.135}$$
$$F = I_L \tag{3.136}$$

where

$$C_1 = 4Ty_2 - Vy_1 \tag{3.137}$$
$$C_2 = 4Uy_1 - Vy_2 \tag{3.138}$$
$$D_{23} = 2x_1\mathrm{K} \tag{3.139}$$
$$D_{32} = -2x_2\mathrm{K} \tag{3.140}$$
$$E_{23} = y_1\mathrm{K} \tag{3.141}$$
$$E_{32} = -y_2\mathrm{K} \tag{3.142}$$
$$F_1 = \frac{1}{\mathrm{K}} \tag{3.143}$$
$$F_2 = -1/2F_1^3 \tag{3.144}$$

Also here it should be noted that this method also suffers from the problem of division by zero when $y_1 = 0$.

3.3.3 The method by Lee and Jen

Lee and Jen has modified Seiler's method slightly by modifying the denominators, and the problem of division by zero is hereby removed. The intermediate terms are as follows

$$w_1 = x_1 y_2 \tag{3.145}$$
$$w_2 = x_2 y_1 \tag{3.146}$$
$$w_3 = w_1 - w_2 \tag{3.147}$$
$$T = I_L + I_M - 2I_Q \tag{3.148}$$
$$U = I_L + I_N - 2I_R \tag{3.149}$$
$$V = I_L + I_S - I_R - I_Q \tag{3.150}$$
$$G = 2I_Q - 2I_L - T \tag{3.151}$$
$$H = 2I_R - 2I_L - U \tag{3.152}$$

Note that, $w_3 = \mathrm{K}$. Once again, the constant term in the last two equations should only appear once in each equation, in order to save multiplications. The coefficients are then computed as follows

$$A = 2(Ty_2^2 + Uy_1^2 - 2Vy_1y_2)/w_3^2 \tag{3.153}$$
$$B = 2(Ux_1^2 + Tx_2^2 - 2Vx_1x_2)/w_3^2 \tag{3.154}$$
$$C = 4(V(w_1 + w_2) - Ux_1y_1 - Tx_2y_2)/w_3^2 \tag{3.155}$$
$$D = (Gy_2 - Hy_1)/w_3 \tag{3.156}$$
$$E = (Hx_1 - Gx_2)/w_3 \tag{3.157}$$
$$F = I_L \tag{3.158}$$

Besides breaking out the constant, the only simplifications made in the comparison was that the divisors $1/w_3$ and $1/w_3^2$ were pre computed in order to save divisions.

3.3.4 Comparison

Table 3.1 shows the time in seconds for the computation of the coefficients of equation (3.89) for one million triangles. The method by Abbas et al. is obviously faster than the fast method proposed by Seiler after the proposed simplifications in paper V are done. However, the method by Lee and Jen is still the fastest and more robust since no special cases needs to be handled as in the other two methods.

3.3.5 Alternative Derivation

An alternative and ultimately faster approach was derived in [33] and in paper VII and related work paper i. It was also shown how this could be used for

Table 3.1: Timing of one million triangles, comparing the computation of the coefficients.

Seiler	Abbas et al.	Lee and Jen
9.17	8.24	7.24

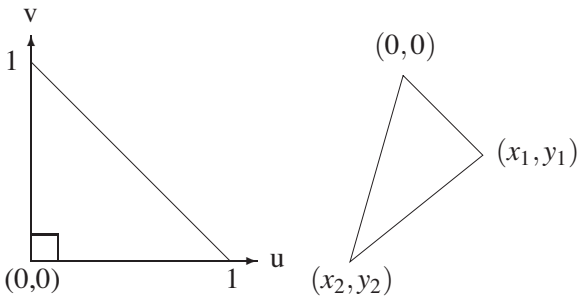

Figure 3.31: To the left: Ortho-normalized triangle. To the right: a polygon with relative vertices.

bump mapping in paper VI. This approach computes the coefficients for a trivial triangle and the setup variables needed are computed from these directly.

The triangle is called an ortho-normalized triangle with $p_0 = (0,0)$, $p_1 = (1,0)$ and $p_2 = (0,1)$, as shown to the left in figure 3.31. It has been obtained by first sorting the vertices, and then shifting them as shown in the same figure to the right. Screen space coordinates were used where the top left corner is $(0,0)$. The coefficients for a bi-variate polynomial over this triangle is first computed by setting up the following system of equations

$$\ldots = M\mathbf{c} \tag{3.159}$$

where

$$\ldots = [I_L, I_M, I_N, I_Q, I_R, I_S]^T \tag{3.160}$$

$$\mathbf{c} = [a, b, c, d, e, f]^T \tag{3.161}$$

The solution for $\mathbf{c} = M^{-1}\ldots$is

$$a = 2(I_L + I_M - 2I_Q) \tag{3.162}$$
$$b = 2(I_L + I_N - 2I_R) \tag{3.163}$$
$$c = 4(I_L + I_S - I_R - I_Q) \tag{3.164}$$
$$d = I_M - I_L - a \tag{3.165}$$
$$e = I_N - I_L - b \tag{3.166}$$
$$f = I_L \tag{3.167}$$

where

$$\ldots = au^2 + bv^2 + cuv + du + ev + f \tag{3.168}$$

It is shown in related work paper i and [33] how the setup variables can be computed directly using equation 3.168 instead of using equation 3.89. Moreover, it is shown that this approach is faster.

Next a transformation is needed from the actual triangle (the triangle to the right in figure 3.31) into the so called ortho-normalized triangle. For any point (x,y) on the actual triangle, a corresponding point (u,v) on the normalized triangle can be obtained by the following transformation

$$u = x\not{/}_u + y\not{\in}_u \tag{3.169}$$
$$v = x\not{/}_v + y\not{\in}_v \tag{3.170}$$

where

$$\not{/}_u = y_2\not{\varnothing} \tag{3.171}$$
$$\not{\in}_u = -x_2\not{\varnothing} \tag{3.172}$$
$$\not{/}_v = -y_1\not{\varnothing} \tag{3.173}$$
$$\not{\in}_v = x_1\not{\varnothing} \tag{3.174}$$

and

$$\not{\varnothing} = \frac{1}{x_1y_2 - x_2y_1} = \frac{1}{\mathrm{K}} \tag{3.175}$$

The reason why the ortho-normalized triangle is used is that the coordinates for the vertices are $(0,0)$, $(1,0)$ and $(0,1)$ and this yields a much more simple matrix inverse to solve. The three solutions explained in paper IV are actually three approaches for solving this inverse. The inverse of this new matrix formulation becomes much more straightforward to solve. The proposed approach is faster than the approaches described in the previous papers.

It is also shown in this paper that the set up variables needed in the rasterization of the polygon can be computed directly from the coefficients used for the ortho-normalized triangle in equation (3.168) instead of computing them from

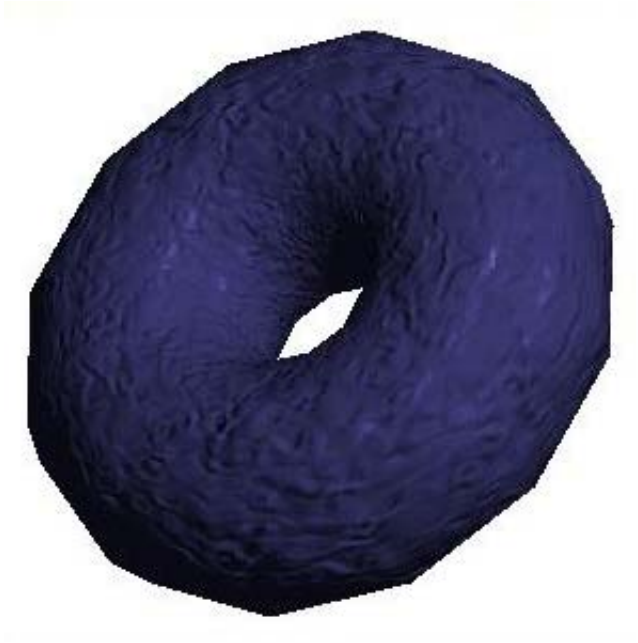

Figure 3.32: A torus shaded using linear interpolation of both the diffuse and specular intensity.

the coefficients of the actual triangle using equation (3.89). Therefore, it is not necessary to compute the latter coefficients which include the transformation. This approach is even faster.

Moreover, it is discussed how the polygons can be rasterized using the proposed set up computation.

3.3.6 Improved Bump Mapping by using Quadratic Vector Interpolation

Paper VI shows how the quadratic shading approach can be used also for bump mapping. Standard graphics hardware can be sued to linearly interpolate rotated vectors in the direction to the light source as well as rotated half-way vectors over the polygon. A cube map [42] is used to normalize these vectors. Figure 3.32 shows that normalization is necessary. The specular reflection almost disappears when the object has got relatively high specularity, whereas they are clearly visible in figure 3.33 where ordinary normalization is performed.

Here it is proposed that quadratic interpolation is used instead. This has to our knowledge not previously been used for bump mapping. This will make the normalization stage unnecessary. The result is faster bump mapping with

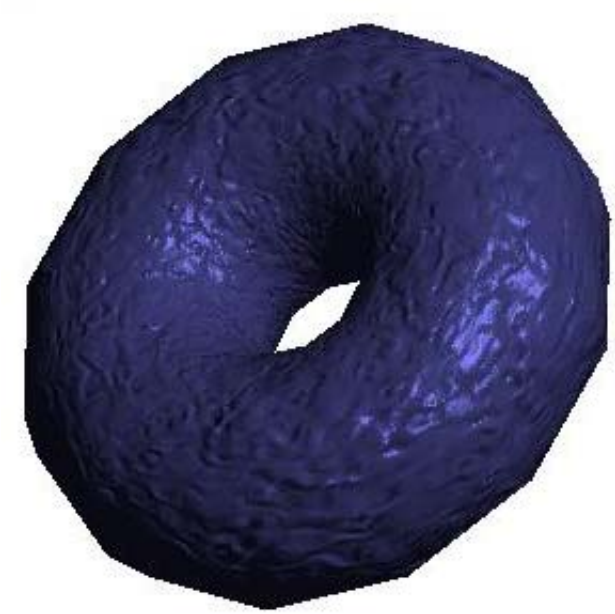

Figure 3.33: A torus shaded using true vector normalization for both the diffuse and specular intensity.

quality near the quality obtained by using vector interpolation with normalization. Figure 3.34 shows that quadratic interpolation produces a result very similar to what is obtained when true normalization is performed. However, to a lower cost, as shown in the paper.

The fast setup derived in the previous paper is used for the new bump mapping interpolation. Furthermore, it is shown that this approach also can be applied to Phong shading resulting in increased performance. Similarly the same approach can be used for efficiently setting up the linear interpolation used for Gouraud shading and Z-buffering.

Let $\forall$be a bi-variate linear function

$$\ldots = au + bv + c \tag{3.176}$$

The coefficients for a bi-variate polynomial over this triangle is first computed by setting up the following system of equations

$$\ldots = M\mathbf{c} \tag{3.177}$$

where

$$\ldots = [I_L, I_M, I_N]^T \tag{3.178}$$

$$\mathbf{c} = [a, b, c]^T \tag{3.179}$$

Figure 3.34: A torus shaded using quadratic interpolation of both the diffuse and specular intensity.

The matrix M is

$$M = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{3.180}$$

The solution for $\mathbf{c} = M^{-1}\ldots$is

$$a = I_M - I_L \tag{3.181}$$
$$b = I_N - I_L \tag{3.182}$$
$$c = I_L \tag{3.183}$$

Use the basis vectors in order to do the transformation from (x,y) to (u,v) when using equation (3.176)

$$\begin{aligned} \forall &= a(\not{} _u x + \in_u y) + b(\not{} _v x + \in_v y) + c \\ &= (a\not{} _u + b\not{} _v)x + (a\in_u + b\in_v)y + c \end{aligned} \tag{3.184}$$

When moving down along the edges , equation (3.176) only requires one additions per scan line. While the y-value is increased by 1 for each new scan line, the x value must be increased by the slope of the edge. For the polygon to the right in figure 3.31 the slope of the left edge is $k_l = x_2/y_2$. The slope

of the upper right side is $k_r = x_1/y_1$ and the slope for the lower right side is $k_r = (x_2 - x_1)/(y_2 - y_1)$. For the edge of the n'th scan line the bi-variate polynomial is

$$\begin{aligned} \ldots_n &= (a\not{}_u + b\not{}_v)nk + (a\in_u + b\in_v)n + c \\ &= n(k(a\not{}_u + b\not{}_v) + (a\in_u + b\in_v)) + c \end{aligned} \tag{3.185}$$

The value of the polynomial for the m'th pixel along the n'th scan line beginning at nk is

$$\begin{aligned} \ldots_{(n,m)} &= a(\not{}_u(nk+m) + \in_u n) + & (3.186) \\ &\quad b(\not{}_v(nk+m) + \in_v n) + c \\ &= a(\not{}_u nk + \in_u n) + & (3.187) \\ &\quad b(\not{}_v nk + \in_v n) + c + a\not{}_u m + b\not{}_v m \\ &= \forall_n + m(a\not{}_u + b\not{}_v) \end{aligned}$$

Paper I shows that for a general quadratic function

$$\ldots = Ax^2 + Bx + C \tag{3.188}$$

a forward difference can be used that only requires two additions per pixel for a scanline

$$p_{i+1} = p_i + dp_i \tag{3.189}$$

$$dp_{i+1} = dp_i + d^2p \tag{3.190}$$

where $p_0 = C$, $dp_0 = A + B$ and $d^2p = 2A$.

When moving down the scanlines, equation (3.89) only requires two additions per scanline. While the y-value is increased by 1 for each new scanline, the x value must be increased by the slope of the edge. For a right oriented triangle with K > 0 the slope of the left edge is $k = x_2/y_2$. For a left oriented triangle with K < 0 the slope of the upper part is $k = x_1/y_1$. The lower part will be treated later. For the edge of the n'th scanline the bi-variate polynomial is

$$\begin{aligned} \ldots_n &= A(nk)^2 + Bn^2 + C(n^2k) + D(nk) + En + F \\ &= (Ak^2 + Ck + B)n^2 + (Dk + E)n + F \end{aligned} \tag{3.191}$$

The forward difference in equation (3.189) and (3.190) is used for the edge where

$$p_0 = F \tag{3.192}$$

$$dp_0 = Ak^2 + Ck + B + Dk + E \tag{3.193}$$

$$d^2p = 2(Ak^2 + Ck + B) \tag{3.194}$$

Note that the factor inside the parentheses of equation (3.194) also occurs in equation (3.193). Therefore, computations could be saved by first computing that factor, and then using it for both equations.

The value of the polynomial for the m'th pixel along the n'th scan line beginning at nk is

$$\begin{aligned} \cdot\cdot_{(n,m)} =& A(nk+m)^2 + Bn^2 + C(nk+m)n + \\ & D(nk+m) + En + F \end{aligned} \tag{3.195}$$

Expanding the equation with (3.191) in mind yields

$$\cdot\cdot_{(n,m)} = \cdot\cdot_n + Am^2 + (2Ank + Cn + D)m \tag{3.196}$$

The term inside the parentheses does not only depend on m but also on n. How does this term change when moving down the scanlines? Let $q = (2Ank + Cn + D)$ then

$$q_n = (2Ak + C)n + D \tag{3.197}$$

Hence, this is a linear recurrence in the y direction with

$$q_0 = D \tag{3.198}$$

$$dq = 2Ak + C \tag{3.199}$$

The following setup is needed for the evaluation over the scanline

$$r = p \tag{3.200}$$

$$dr = A + q \tag{3.201}$$

$$d^2 r = 2A \tag{3.202}$$

It is possible to simplify the setup by changing equation (3.201) to $dr = q$, by setting equation (3.198) to $q_0 = A + D$.

For the lower triangle $k = (x_2 - x_1)/(y_2 - y_1)$ and the computation from (x_1, y_1) down to (x_2, y_2) for the n'th scanline after (x_1, y_1) is

$$\begin{aligned} \cdot\cdot_{y_1+n} =& A(x_1 + nk)^2 + B(y_1 + n)^2 + \\ & C(x_1 + nk)(y_1 + n) + \\ & D(x_1 + nk) + E(y_1 + n) + F \end{aligned} \tag{3.203}$$

After expanding the expression with (3.191) in mind, we have

$$\begin{aligned} \cdot\cdot_{y_1+n} =& n^2(Ak^2 + B + Ck) + \\ & n(2Ax_1k + 2By_1 + Cx_1 + \\ & Cky_1 + Dk + E) + \cdot\cdot_{y_1} \end{aligned} \tag{3.204}$$

The setup variables for this equation are

$$p_0 = I_M \tag{3.205}$$

$$\begin{aligned} dp_0 =& Ak^2 + B + Ck + 2Ax_1k + 2By_1 + \\ & Cx_1 + Cky_1 + Dk + E \end{aligned} \tag{3.206}$$

$$d^2p = 2(Ak^2 + B + Ck) \tag{3.207}$$

Once again, the factor inside the parentheses in equation(3.207) also occurs in equation (3.206). Thus, it can be computed once and used in both equations.

The value of the polynomial for the m'th pixel along the n'th scan line beginning at nk for this case is

$$\begin{aligned} \ldots_{(n,m)} =& A(x_1 + m + nk)^2 + B(y_1 + n)^2 + \\ & C((x_1 + m + nk)(y_1 + n)) + \\ & D(x_1 + m + nk) + E(y_1 + n) + F \end{aligned} \tag{3.208}$$

Expanding the equation with (3.191) in mind yields

$$\begin{aligned} \ldots_{(n,m)} =& \ldots_n + Am^2 + \\ & (2Ax_1 + 2Ank + Cy_1 + Cn + D)m \end{aligned} \tag{3.209}$$

The term inside the parentheses for this case depends on n as

$$q_n = (2Ak + C)n + 2Ax_1 + Cy_1 + D \tag{3.210}$$

Once again q is a linear equation, that requires only one addition per scanline

$$q_0 = 2Ax_1 + Cy_1 + D \tag{3.211}$$

$$dq = 2Ak + C \tag{3.212}$$

The same setup and forward difference with r, dr and d^2r for the evaluation over the scanline used for the upper part can be used for the lower part. Although, a similar simplification used for the upper part should be used here also. Therefore, let $q_0 = 2Ax_1 + Cy_1 + D + A$.

Note that the term $2Ax_1$ occurs both in equation (3.206) and (3.211). Thus, it can be precomputed, in order to save two multiplications. Another two multiplications could be saved by breaking out the factor k from equation (3.206). These simplifications have been used for the pseudo code in a later section, which shows how the forward differences explained here are set up using the coefficients. However, the computation of the coefficients are left out since they are explained in detail in the previous section.

To sum up, we have so far demonstrated how the setup and the computation along each edge and along each scanline can be done efficiently. Now we will show how the setup can be simplified.

3.3.7 Simplified Setup

For each polygon, the setup is a combination of the coefficients of equation (3.89). Examining for example equations (3.193), (3.194) and (3.199) reveals this.

By expressing the setup in terms of the coefficients in equation (3.168) instead of (3.89), it is not necessary to compute the latter coefficients. This saves a substantial amount of computations. Note that for this case, the setup for p, r and dr do not change. For K > 0 with $k = x_2/y_2$ and $j = 1/y2$ the setup variables are

$$dp = K_1 + ej \tag{3.213}$$
$$d^2p = 2K_1 \tag{3.214}$$
$$q = K_2 + (dy_2 - ey_1)\varnothing \tag{3.215}$$
$$dq = (c - 2by_1 j)\varnothing \tag{3.216}$$
$$d^2r = 2K_2 \tag{3.217}$$

where

$$K_1 = bj^2 \tag{3.218}$$
$$K_2 = (y_2^2 a + y_1^2 b - y_1 y_2 c)\varnothing^2 \tag{3.219}$$

For K < 0, the upper partial triangle of the right oriented triangle have $k = x_1/y_1$ and $j = 1/y1$ and

$$dp = K_1 + dj \tag{3.220}$$
$$d^2p = 2K_1 \tag{3.221}$$
$$q = K_2 + (dy_2 - ey_1)\varnothing \tag{3.222}$$
$$dq = (2ay_2 j - c)\varnothing \tag{3.223}$$
$$d^2r = 2K_2 \tag{3.224}$$

where

$$K_1 = aj^2 \tag{3.225}$$
$$K_2 = (y_2^2 a + y_1^2 b - y_1 y_2 c)\varnothing^2 \tag{3.226}$$

Note that if $y1 = 0$, then there is no upper partial triangle, that is, the triangle has an upper edge which is aligned with the scanline. Hence, the computation above is not necessary and division by zero can be avoided.

For the lower triangle with $k = (x_2 - x_1)/(y_2 - y_1)$ and $j = 1/(y_2 - y_1)$ the setup variables are

$$dp = K_1 + (c - 2a - d + e)\, j \tag{3.227}$$
$$d^2 p = 2K_1 \tag{3.228}$$
$$q = K_2 + (y_2(2a + d) - y_1(c + e))\varnothing \tag{3.229}$$
$$dq = (y1(c - 2b) + y2(c - 2a))\, j\varnothing \tag{3.230}$$
$$d^2 r = 2K_2 \tag{3.231}$$

where

$$K_1 = (a + b - c)\, j^2 \tag{3.232}$$
$$K_2 = (y_2^2 a + y_1^2 b - y_1 y_2 c)\varnothing^2 \tag{3.233}$$

Note that $p = I_M$, since the computation starts from the middle vertex. If $y2 - y1 = 0$ then there is no lower partial triangle, and hence, the code for the lower triangle could be skipped for that triangle.

Paper VI states that the method described in [33] makes the setup for Phong shading faster. Once again we do not need to make the transformation, instead we compute the setup variables directly, using the ortho-normalized triangle.

The Phong coefficients can generally by derived as follows. If $\mathbf{n}$ is linearly interpolated by $\mathbf{A}u + \mathbf{B}v + \mathbf{C}$, then $\mathbf{n} \cdot \mathbf{n}$ is

$$(\mathbf{A}u + \mathbf{B}v + \mathbf{C})^2 = \tag{3.234}$$
$$\mathbf{A}^2u^2 + \mathbf{B}^2v^2 + 2\mathbf{AB}uv + 2\mathbf{AC}u + 2\mathbf{BC}v + \mathbf{C}^2 \tag{3.235}$$

If the original setup is used the transformations described in 2.41 through 2.43 must be used for the normals at the vertices. However if the new setup is used then the simple transformations described in 3.181 through 3.183 are used instead. Anyhow, the coefficients for Phong shading become

$$a = \mathbf{A}^2 \tag{3.236}$$
$$b = \mathbf{B}^2 \tag{3.237}$$
$$c = 2\mathbf{AB} \tag{3.238}$$
$$d = 2\mathbf{AC} \tag{3.239}$$
$$e = 2\mathbf{BC} \tag{3.240}$$
$$f = 1 \tag{3.241}$$

Table 3.2 shows that the proposed approach is faster than the original approach, which need about 96.4% of the time that the original setup need.

The result is thus a minor improvement compared to the original Phong shading setup.

Table 3.2: Timing of one hundred million triangles, comparing the computation of the setup variables for both orientations of triangles, on an 1.8 GHz Pentium 4.

	Phong	New approach
$K > 0$	12.94	12.47
$K < 0$	13.80	13.31

3.3.8 Pseudo code for quadratic shading

The following code is only an example of how the proposed equations for the setup could be used in order compute the forward differences in the right way.

```
FUNCTIONrasterize()
begin
  for y := y_s to y_e − 1
      r = p;
      dr = q;
      Z = z;
      for x := x_s to x_e − 1
          if (Z < zBuffer[x][y])
            Zbuffer[x][y] = Z;
            frameBuffer[x][y] = r;
          end
          Z = Z + dZ;
          r = r + dr;
          dr = dr + d^2r;
      end
      x_s = x_s + k_l;
      x_e = x_e + k_r;
      z = z + dz;
      q = q + dq;
      p = p + dp;
      dp = dp + d2p;
  end
end
```

$FUNCTIONshade()$
begin

$x_1 = x_b - x_a;$
$y_1 = y_b - y_a;$
$x_2 = x_c - x_a;$
$y_2 = y_c - y_a;$
$\varnothing = 1.0/(x_1 * y_2 - x_2 * y_1);$
$/_u = y_2 * \varnothing;$
$/_v = -y_1 * \varnothing;$
$€_u = -x_2 * \varnothing;$
$€_v = x_1 * \varnothing;$
$z = z_a;$
$Az = z_b - z_a;$
$Bz = z_c - z_a;$
$a = 2 * (I_M + I_L - 2 * I_Q);$
$b = 2 * (I_N + I_L - 2 * I_R);$
$c = 4 * (I_L + I_S - I_Q - I_R);$
$d = I_M - I_L - a;$
$e = I_N - I_L - b;$
$f = I_L;$

if $\varnothing > 0$

begin

$x_s = x_e = x_a;$

$y_s = y_a;$

$y_e = y_b;$

$k_l = x_2/y_2;$

$k_r = x_1/y_1;$

$dZ = Az */_u + Bz */_v;$

$dz = k_l * dZ + Az * €_u + Bz * €_v;$

$p = f;$

$j = 1/y_2;$

$K_1 = b * j * j;$

$K_2 = (y_2 * y_2 * a + y_1 * y_1 * b - y_1 * y_2 * c) * \varnothing * \varnothing;$

$dp = (K_1 + e * j);$

$d2p = 2 * K_1;$

$q = (K_2 + (d * y_2 - e * y_1) * \varnothing);$

$dq = (c - 2 * b * y_1 * j) * \varnothing;$

$d^2r = 2 * K_2;$

$rasterize();$

$x_e = x_b;$

$y_s = y_b;$

$y_e = y_c;$

$k_r = (x_2 - x_1)/(y_2 - y_1);$

$rasterize();$

end

else $IF\varnothing < 0$

begin

$x_s = x_e = x_a;$

$y_s = y_a;$

$y_e = y_b;$

$k_l = x_1/y_1;$

$k_r = x_2/y_2;$

$dZ = Az*/_u + Bz*/_v;$

$dz = k_l * dZ + Az * €_u + Bz * €_v;$

$p = f;$

$j = 1/y_1;$

$K_1 = a * j * j;$

$K_2 = (y_2 * y_2 * a + y_1 * y_1 * b - y_1 * y_2 * c) * \varnothing * \varnothing;$

$dp = (K_1 + d * j);$

$d^2p = 2 * K_1;$

$q = (K_2 + (d * y_2 - e * y_1) * \varnothing);$

$dq = (2 * a * y_2 * j - c) * \varnothing;$

$d2r = 2 * K_2;$

$rasterize();$

$p = I_M;$

$x_s = x_b;$

$y_s = y_b;$

$y_e = y_c;$

$k_l = (x_2 - x_1)/(y_2 - y_1);$

$z = zb;$

$dz = k_l * dZ + Az * €_u + Bz * €_v;$

$j = 1/(y_2 - y_1);$

$K_1 = (a + b - c) * j * j;$

$K_2 = (y_2 * y_2 * a + y_1 * y_1 * b - y_1 * y_2 * c) * \varnothing * \varnothing;$

$dp = (K_1 + (c - 2 * a - d + e) * j);$

$d^2p = 2 * K_1;$

$q = (K_2 + (y_2 * (2 * a + d) - y_1 * (c + e)) * \varnothing);$

$dq = (y_1 * (c - 2 * b) + y_2 * (c - 2 * a)) * j * \varnothing;$

$d^2r = 2 * K_2;$

$rasterize();$

end

end

3.3.9 Faster Quadratic Shading

Paper VIII shows how a four times faster setup can be derived and it will be referred to as X-shading in the subsequent sections, since that was our working

name for this approach.

Some results from paper VII was used in the derivation of this approach and this paper shows how a curvilinear mesh can be constructed from vertex points and vertex normals. Tangents can be constructed in the same way as shown in figure 3.10 by using the so-called Gram-Schmidt orthogonalization algorithm[53]

$$\mathbf{t}_1 = \mathbf{n}_2 - \mathbf{n}_1(\mathbf{n}_1 \cdot \mathbf{n}_2) \tag{3.242}$$

$$\mathbf{t}_2 = -\mathbf{n}_1 + \mathbf{n}_2(\mathbf{n}_1 \cdot \mathbf{n}_2) \tag{3.243}$$

Note, that the normals are assumed to be normalized. However it is shown in the paper that the resulting formula is invariant to tangent length.

A curve spanned by the tangent vectors and vertex points which is as relaxed as possible is derived. This curve can be obtained by minimizing the integral which is the total sum of the square acceleration

$$\int_0^1 \|f''(s)\|^2 ds \tag{3.244}$$

on some variable that controls the tangent length and apply it on the boundary curve. This defines the least square acceleration. However, a second degree curve can only have one derivative determined when it is also determined to pass through two end points. Therefore, we have to find a second degree curve spanned by the tangent vectors and vertex points which have optimal fit between both these tangent vectors and the derivatives. Such a curve can be derived by finding an optimal fit for the first tangent, then we can minimize the integral in equation (3.244).

The final formula is

$$\mathbf{f}(t) = \left(\frac{/'€'\mathbf{t}_1 - / €\mathbf{t}_0}{2}\right)s^2 + \left(\mathbf{P} + \frac{/ €\mathbf{t}_0 - /'€'\mathbf{t}_1}{2}\right)s + P_1 \tag{3.245}$$

where

$$/ = \frac{\mathbf{t}_0 \cdot \mathbf{t}_1}{\mathbf{t}_0 \cdot \mathbf{t}_0} \tag{3.246}$$

$$€ = \frac{\mathbf{P} \cdot \mathbf{t}_0}{\mathbf{t}_0 \cdot \mathbf{t}_1} \tag{3.247}$$

$$/' = \frac{\mathbf{t}_0 \cdot \mathbf{t}_1}{\mathbf{t}_1 \cdot \mathbf{t}_1} \tag{3.248}$$

$$€' = \frac{\mathbf{P} \cdot \mathbf{t}_1}{\mathbf{t}_0 \cdot \mathbf{t}_1} \tag{3.249}$$

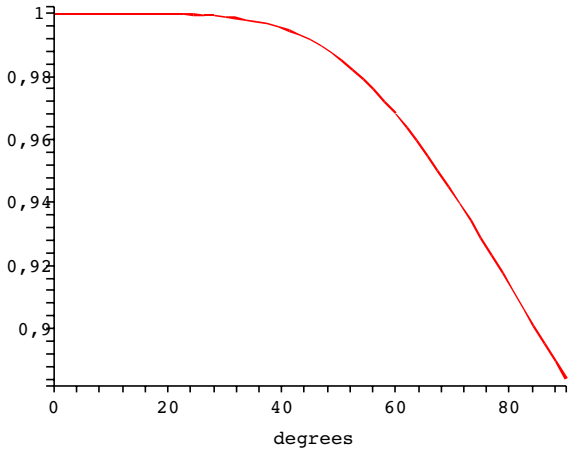

Figure 3.35: The vector norm of the approximation for different angles between the vectors.

and

$$\mathbf{P} = P_2 - P_1 \tag{3.250}$$

where P_1 and P_2 are the vertices on a polygon. However, we are not interested in the vertices. Instead we want an approximation of the vector half way between two vectors. We can get such an approximation by treating the end points of the two normals as the two points in the formula. Thus we will obtain a point on a quadratic curve between the end points. This new point will be treated as the new normal. It can easily be shown that for this case we get

$$\not{\mathbf{t}}_0 = \mathbf{t}_0 \tag{3.251}$$

$$\not{\mathbf{t}}_1 = \mathbf{t}_1 \tag{3.252}$$

Let $s = 1/2$ then we get

$$\mathbf{n}_{1/2} = \left(\frac{\mathbf{t}_1 - \mathbf{t}_0}{8}\right) + \left(\frac{\mathbf{n}_1 - \mathbf{n}_0}{2} + \frac{\mathbf{t}_0 - \mathbf{t}_1}{4}\right) + \mathbf{n}_0 \tag{3.253}$$

This approximation of the mid edge vector works quite well for normals with an angle that are less than 45^o as shown in figure 3.35 where the vector norm of the mid-edge vector is computed for different angles. The norm is close to one for angles less than 45^o. However, the norm starts to decrease rather rapidly after that. It should be noted that the scale goes from 0.88 to 1.0.

Hence, the approximation is still not so bad for angles over 45^o. Moreover, this approximation will always do better than a linear approximation, which assures that the quadratic approach will always be better than Gouraud shading. However, the quality of the highlight will be affected for larger angles. It could be argued though, that such large angles means that the polygons should be subdivided further.

Expanding equation (3.253) by using equations (3.242) and (3.243) yeilds

$$\mathbf{n}_{1/2} = \frac{5 - \mathbf{n}_0 \cdot \mathbf{n}_1}{8}(\mathbf{n}_0 + \mathbf{n}_1) \tag{3.254}$$

This is exactly the same as equation (3.13), which was derived for a faster variant of Euler shading. Hence, X-shading is in some sense a polygon setup variant of Euler shading, which was proposed in paper II.

In the next step we use this approximation for computing the mid egde vectors and put this into equation (3.159). The solution now becomes

$$a = (I_L + I_M)(\mathbf{n}_0 \cdot \mathbf{n}_1 - 1)/2 \tag{3.255}$$
$$b = (I_L + I_N)(\mathbf{n}_0 \cdot \mathbf{n}_2 - 1)/2 \tag{3.256}$$
$$c = a + b - (I_M + I_N)(\mathbf{n}_1 \cdot \mathbf{n}_2 - 1)/2 \tag{3.257}$$
$$d = I_M - I_L - a \tag{3.258}$$
$$e = I_N - I_L - b \tag{3.259}$$
$$f = I_L \tag{3.260}$$

Note that this solution only depends on the vertex normals and the reason for this is of course that the approximation of the mid-edge vectors only depends on the vertex normals. The effect of this is that the extra normalization of the mid-edge vectors disappears as well as the extra light computation. Hence, a substantial amount of computation time is saved by using the proposed method. Table 3.3 shows the timing for the complete setup for the method proposed by Lee and Jen, the fast version proposed in paper VI, paper i and in [33] and the method proposed in this paper for 100 million complete setup's. The new method is about four times faster than previous methods. The K represents the two possible orientations for a polygon as shown in paper V.

Figure 3.36 shows the back of the famous Venus de Milo statue that has been shaded by Phong shading using a polygon setup approach. This image can be compared to the same object in figure 3.37 shaded by the proposed approach. The difference is neglectable.

A torus with only 288 triangles where every two adjacent triangles constitute a planar surface, was shaded in figure 3.38 and 3.39. The maximum angle between two normals are as high as 41.4^o and obviously it should have been subdivided further since the contour is not looking good. Anyway, the shading and especially the highlights are indistinguishable from each other.

Figure 3.36: A Phong shaded Venus de Milo statue.

Figure 3.37: A shaded Venus de Milo statue using the proposed appraoch (X-shading)

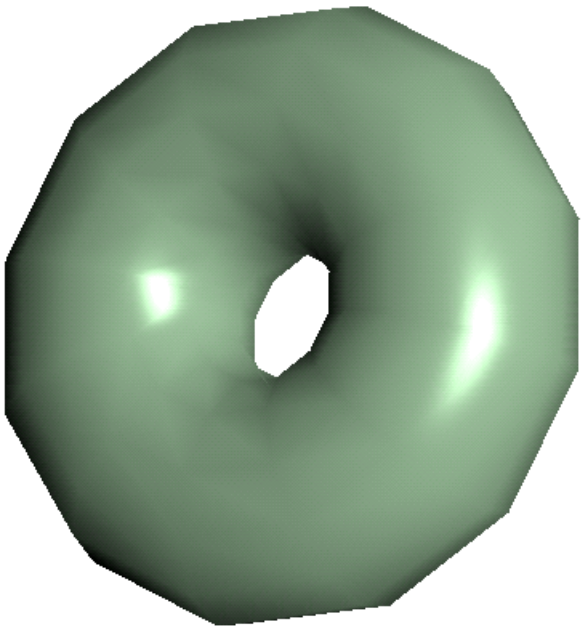

Figure 3.38: A Phong shaded torus with 288 polygons.

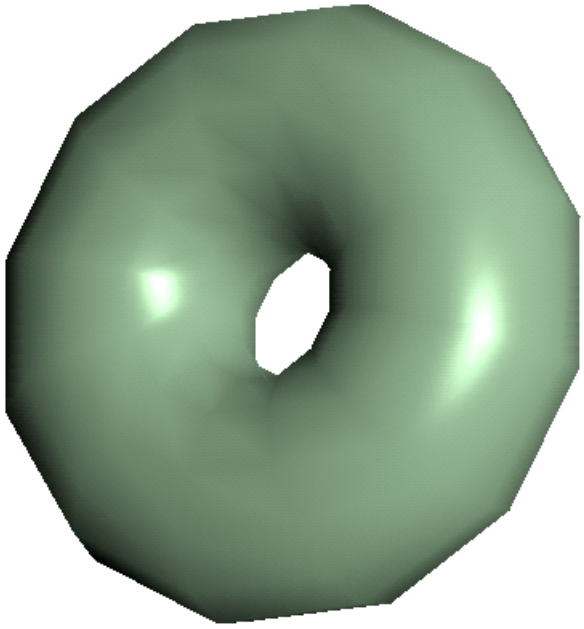

Figure 3.39: A shaded 288 polygon torus using the proposed approach (X-shading)

Table 3.3: Timing of one hundred million triangles, comparing the computation of the setup variables for both orientations of triangles, on an 1.8 GHz Pentium 4.

	Lee & Jen	Quadratic	New approach
$K > 0$	51.9	49.4	12.5
$K < 0$	54.2	50.8	13.0

The conclusion is that X-shading, even though an approximation is used for the mid-edge vectors, still will produce satisfactory results. Moreover, it is much faster than Phong and even faster than previous quadratic shading approaches. This will make X-shading attractive for software rendering. Especially since all square roots and several divisions have been removed from the setup.

3.4 Paper IX: Bump map Antialiasing

Paper IX deals with the problem that appears for bump mapped surfaces that are magnified. Minification can be treated by using prefiltering and Mip-Maps [75]. However, prefiltering will reduce the bumps, which is undesirable. The goal is to produce non jagged bumps, without removing high frequency details by low-pass filtering. Textures that are magnified appears jagged, and therefore reconstruction filters using linear or cubic interpolation polynomials are often used to reduce aliasing. Bump mapped objects will also appear jagged if no filtering is used. In figure 3.40 a bump mapped torus is sampled using nearest neighbor sampling (no filtering). The jaggedness is clearly visible.

The Shannon sampling theorem [63] say that a signal could be reconstructed from its samples if the sampling frequency is greater than twice the highest frequency in the original signal. This lower bound on the sampling frequency is known as the Nyquist limit. This theorem also applies for digital images, where the size of the sampling interval should be chosen so that it is less than half of the smallest interesting detail in the image [65]. If the sampling rate is below the Nyquist rate, aliasing will occur since only parts of the high frequency signal will be sampled and the resulting signal will have a lower frequency. A comprehensive overview of antialiasing and the sampling theorem is given by Foley et al. [26].

Texture mapping is the procedure of mapping a 2D texture onto a surface, for the purpose of making objects appear more realistic. Low-pass filtering

Figure 3.40: A bump-mapped torus sampled using the nearest neighbor method.

will remove high frequencies in a texture, and could be used to assure that all frequencies will be below the Nyquist limit. Therefore, prefiltering is often used for antialiasing for texture mapping. Heckbert [34] gives an overview of these techniques. Prefiltering will also remove some of the jaggedness that could be visible in an image, especially on the silhouette of objects, where the gradients are high.

When the texture is sampled, several techniques could be used, as described in the standard textbook in image processing by Gonzales and Woods [31]. The *nearest neighbor* method is the fastest but gives the worst quality, since the u and v values are truncated and then these integer values are used to sample the texture map. *Bilinear interpolation* will use the fraction of the u and v coordinates in order to obtain an interpolation between four neighboring texels. The approach of taking more than one sample and then combining them is known as *super sampling*.

Bilinear interpolations is often used together with Mip-maps, which was introduced by Williams [75] as a method to handle the minification problem. He uses several texture maps, where each map is a quarter of the size of the previous one. Each map is down sampled by filtering, typically by averaging four neighboring texels into one texel. Depending on how small the texture mapped object is, two maps are chosen and bilinear interpolation is done in both of them and these values are weighted together depending on how far away the object is. This is known as *trilinear interpolation*.

When an image is magnified the texels will appear as angular squares in the image, i.e. the image appears jagged. Bilinear interpolation could be used to diminish this effect. The individual texels are themselves regular samples of

some texture, maybe a photograph of a real life object. Bi-cubic interpolation will do a better job to reconstruct the details in the image that are missing, which lies between the texels. Mitchell and Netravali [51] shows how piecewise cubic polynomials could be used as reconstruction filters. An overview of different spline filters is given by Unser [70]. Horbelt et al. [38] shows how cubic interpolation could be used for resizing images with arbitrary scaling factors and Bongjun et al. [48] uses cubic splines for CCD color filters, to mention just a few uses for cubic splines as reconstruction filters.

In general, a continuous representation $F(x)$ of a discrete signal $f(x)$ is obtained by convolving the discrete signal with a reconstruction filter kernel $\uparrow(x)$

$$\begin{aligned} F(x) &= f(x) * \uparrow(x) \\ &= \sum_{n=-\div}^{\div} f(n)\uparrow(x-n) \end{aligned} \tag{3.261}$$

To assure that the output signal will be constant if the input signal is constant, it is required that

$$\sum_{n=-\div}^{\div} \uparrow(x-n) = 1 \tag{3.262}$$

For the family of cubic splines it can be shown that [51]

$$\uparrow(x) = \frac{1}{6}\begin{cases} (12-9B-6C)x^3+ \\ (-18+12B+6C)x^2+ \\ (6-2B) & \text{if } x<1 \\ (-B-6C)x^3+ \\ (6B+30C)x^2+ \\ (-12B-48C)x+ \\ (8B+24C) & \text{if } 1\le x<2 \\ 0 & \text{otherwise} \end{cases} \tag{3.263}$$

where B-splines have $B=1$ and $C=0$, Catmull-Rom splines [16] have $B=0$ and $C=\frac{1}{2}$. Other values for B and C, will give cubic splines with different properties. However, B-splines and Catmull-Rom splines were chosen for further investigation in paper IX.

3.4.1 Antialiasing of Bump Maps

Some efforts, trying to remove aliasing from bump mapped objects has previously been described. Schilling [60] uses a roughness pyramid that contain roughness information for the texels. The roughness is computed from a covariance matrix containing squares of the standard deviation of the gradients F_u and F_v. The distribution of the normal vectors could be obtained from

this matrix. This roughness information is used to change the exponent in the Phong-Blinn model [13]

$$I_s = (\mathbf{n} \cdot \mathbf{h})^s \tag{3.264}$$

The effect will be a less specular surface for pixels that are more rough.

Becker and Max [9] use the derivative of a cubic B-spline to reconstruct the gradients for bump mapping. Eight values are needed for each gradient. Two quadratic interpolations are done in each direction, which then are linearly weighted together. This approach is taken a step further in this paper, by showing how a general reconstruction filter that will give intermediate gradients, can be formulated.

It could be shown by Fourier analysis [14] that a square shaped wave will contain an infinite number of sine shaped waves, with higher frequency than the square shaped wave itself. Since reconstruction filters will smooth the curve, they will perform low-pass filtering. If the reconstruction filter is sampled at integer points, the original samples are obtained. If the original samples are not obtained, then the reconstruction filter will not reconstruct the original signal properly. Such filter is said to perform low-pass filtering below the Nyquist limit, and could be useful if we want to smoothen the original signal.

For image reconstruction a suitable family of the cubic curves can be chosen from Eq.(3.263). Then Eq.(3.261) is used in two dimensions in a tensor product form

$$f_c(x,y) = \sum_{n=0}^{3} \sum_{m=0}^{3} f(n,m) \uparrow (x-n) \uparrow (y-m) \tag{3.265}$$

This equation is separable, which means that we could compute Eq.(3.261) for each row to obtain four values that are then used in one computation for the columns. This is principally the same equation used for spline surfaces by Hill [37] and Hearn and Baker [36]. However, the matrix form will be used subsequently

$$F = \mathbf{u} M G M^T \mathbf{v}^T \tag{3.266}$$

where $\mathbf{u} = (u^3, u^2, u, 1)$, $\mathbf{v} = (v^3, v^2, v, 1)$, M is the basis matrix that tells which family of splines that are used, and G is the geometry matrix which contains the control points, or as for the case of bump mapping, the height values. If u' and v' are the coordinates in the height map then $u = u' - \lfloor u' \rfloor$ and $v = v' - \lfloor v' \rfloor$.

Reconstruction filters can be used to obtain intermediate height values. However, they can also, as proposed in the paper, be used to obtain intermediate gradients immediately.

Paper IX propose that the reconstruction filters for gradients are

$$F_u = \frac{\mathrm{E}\mathbf{u}}{\mathrm{E}u} M G M^T \mathbf{v}^T \tag{3.267}$$

$$F_v = \mathbf{u} M G M^T \frac{\mathrm{E}\mathbf{v}}{\mathrm{E}v}^T \tag{3.268}$$

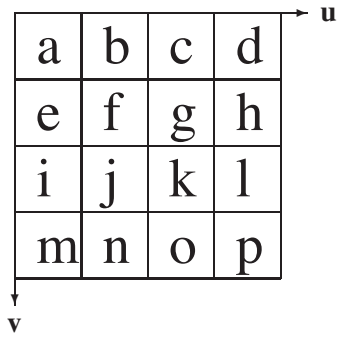

Figure 3.41: Sixteen sample points in a local neighborhood of the height map

Table 3.4: Partial derivative F_u of Cubic B-splines for different values of u and v.

u	v	F_u
0	0	$\frac{1}{12}((c-a)+3(g-e)+(k-i))$
1	0	$\frac{1}{12}((d-b)+3(h-f)+(l-j))$
0	1	$\frac{1}{12}((g-e)+3(k-i)+(o-m))$
1	1	$\frac{1}{12}((h-f)+3(l-j)+(p-n)$

Different filters are compared and it is shown that Catmull-Rom [16] spline filters will not suppress bumps as B-spline filters will. If height values are obtained in a local neighborhood as in figure 3.41 and the normal is computed as $\mathbf{n}' = (-F_u, -F_v, 1)$, then we can compare what the gradients will turn out to be. The difference is shown in table 3.4 and 3.5. The gradients for B-spline filters are combinations of three gradients and this will blur the result as shown in figure 3.42. However, the gradients for Catmull-Rom filters are not a combination and therefore the result is better quality bump mapping. This can be noticed in figure 3.43.

Furthermore, it is shown how linear interpolation of gradients can be used to remove jaggedness. The result is shown in figure 3.44. Linear interpolation of gradients will produce results comparable to Catmull-Rom filtering.

3.4.2 Fast Normalization of Bump Map Normals

Normalization is a computationally costly process, mainly because of the square root and division that are involved. When moving frame bump mapping is used

Figure 3.42: A bump-mapped torus sampled using B-spline filtering.

Figure 3.43: A bump-mapped torus sampled using Catmull-Rom filtering.

Table 3.5: Partial derivative F_u of Catmull-Rom splines for different values of u and v.

u	v	F_u
0	0	$\frac{1}{2}(g-e)$
1	0	$\frac{1}{2}(h-f)$
0	1	$\frac{1}{2}(k-i)$
1	1	$\frac{1}{2}(l-j)$

Figure 3.44: A bump-mapped torus sampled using interpolation of Gradients.

it is necessary to normalize the perturbed normal for each pixel, unless these normals are computed in advance [56].

There are at least two interesting approaches for approximations of the normalization. Both lends themselves to normalization of bump map normals. Related work paper iii shows how the computation can be efficiently implemented. A bump map normal can be retrieved as $\mathbf{n} = (-F_u, -F_v, 1)$, where F_u and F_v are the gradients in the u and v directions respectively. These are obtained from the height map and of course we can compute them so that we get the negative sign directly, without having to negate them before normalization. One of the reasons why bump map normalization can easily be approximated is the fact that F_u and F_v are relatively small. Therefore $\mathbf{n}$ will be rather close to the vector $(0,0,1)$. The necessary code is:

```
float N_z=1/sqrt(F_u*F_u+F_v*F_v+1);
float N_x=F_u*N_z;
float N_y=F_v*N_z;
```

In Paper I a normalization trick is proposed, where a unit length vector is reflected around a vector that is halfway between this unit vector and the vector we will normalize. After reflection the resulting vector will point in the same direction as the normal. The unit vector in this case is preferably $\mathbf{u} = (0,0,1)$ and the problem is to find the vector halfway between $\mathbf{u}$ and $\mathbf{n}$. An approximation of this vector would be to let $\mathbf{v} = \mathbf{n} + \mathbf{u}$, which will be a reasonably good approximation. Since the angle between $\mathbf{n}$ and $\mathbf{u}$ is small, $\mathbf{v}$ will be relatively close to the vector that is halfway between. The approximated vector is

$$\mathbf{n}' = 2\mathbf{v}\frac{\mathbf{v}\cdot\mathbf{u}}{\mathbf{v}\cdot\mathbf{v}} - \mathbf{u} \tag{3.269}$$

The nice thing about this formulation is that we get rid of the square root. However, the division will still slow down the computation. After simplification the code becomes

```
float t=4/(F_u*F_u+F_v*F_v+4);
float N_x=D_u*t;
float N_y=D_v*t;
float N_z=2*t-1;
```

In a presentation by nVIDIA [77] an interesting approximation of the normalization is proposed, which is based on one step of the Newton-Raphson method. This method is discussed in paper V. The proposed equation is

$$\mathbf{n}' = \frac{\mathbf{n}}{2}(3 - \mathbf{n}\cdot\mathbf{n}) \tag{3.270}$$

This normalization method is used by nVIDIA for normalizing the interpolated vectors in their bump mapping approach. However, it is not used for normalizing the perturbed normal. We have found that it can be used for this purpose and it will be very fast, since the third element in the normal vector is always 1 for bump map normals. This approach is developed in related work paper iii. Since it does not include any division it is faster than our normalization trick and of course faster than true normalization.

Equation (3.270) can for bump map normals be rewritten as

$$\mathbf{n}' = \mathbf{n}(3/2 - (F_u^2 + F_v^2 + 1)/2) = \mathbf{n}(1 - (F_u^2 + F_v^2)/2)) \tag{3.271}$$

This equation can be realized in code as:

```
float N_z=1-(F_u*F_u+F_v*F_v)*0.5;
float N_x=F_u*N_z;
float N_y=F_v*N_z;
```

Table 3.6: Timing of different methods as well as an empty loop.

Empty loop	Normalization	Reflection	Newton
0.21	8.24	2.45	1.37

3.4.3 Comparison

A loop where the normalization of one hundred million normals was timed on a 1.8 G Hz Pentium 4 machine. The table shows that the reflection normalization trick is about 3.5 times faster and the Newton-Raphson method is about 6.9 times faster than ordinary normalization, taking into account the timing of an empty loop. A software table look-up normalization would include several multiplications, additions and float to integer conversions as well as fetches and could hardly compete with the Newton-Raphson based normalization approximation.

3.5 Papers X and XI: Illumination and Lighting

Related work paper ii present a modification of how ambient light is computed. The diffuse term $I_d = n \cdot l$ cannot be used when it is negative. Therefore, polygons facing away from the light source are usually rendered using ambient light only, when using the Phong-Blinn model. If we could use the diffuse light for these polygons as well, the back sides would not appear that flat, since the diffuse light gives the viewer information about the form of the object. Actually this is possible with some modification. If the polygon is not facing away then use the Phong-Blinn model as usual. However, if it is back facing, i.e. $I_d < 0$ then use

$$I_d = I_a(1 + qI_d) \tag{3.272}$$

where $q \in [0,1]$, which determines the range of ambient light. The equation tells, that we shall use ambient light I_a as usual, but then some part of the diffuse light should be added. Whatever value for q we choose, I_d will be equal to zero on the contour of the shadow. This is just the same as the ordinary model would yield and it assures that the intensity will be continuous over the shadow contour (if no specular light is visible there). However, inside the shadowed area the intensity depends on the normal at each point. Therefore q will determine how much the shadowed area will be affected by the underlying surface. Since I_d is negative on the backside the intensity varies from $I_a(1-q)$ to I_a.

Figure 3.45 shows a bump mapped torus where the inside is facing away from the light source and thus appear flat. In igure 3.46 the same torus has

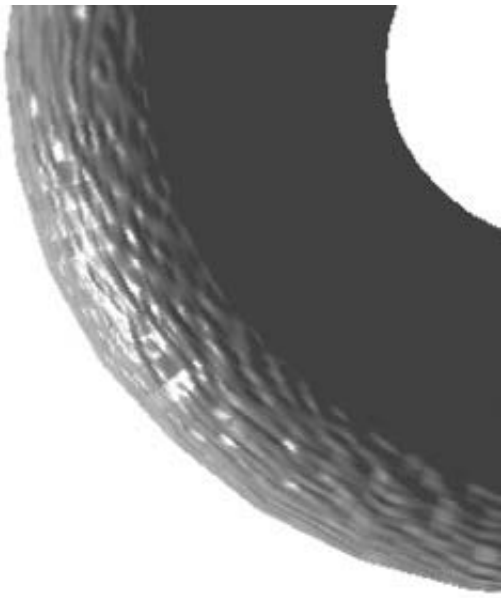

Figure 3.45: A bump mapped torus where the inside is facing away from the light source and thus appear flat.

been rendered with $q = 50$. The underlying geometry is clearly visible and the shadowed area looks more realistic.

We later found that this idea has been proposed before but not for local illumination using the Phong Illumination model. It was proposed in [23] that the ambient light should be replaced by directional light for radiosity. Any direction could be chosen. However, they propose that $\mathbf{n} \cdot \mathbf{v}$ should be used since the whole visible object will be lit in this way. If the view vector is constant, i.e. $\mathbf{v} = [0,0,1]$ then this computation is very fast. On the other hand, I_d is always computed so no extra computation is necessary using our approach.

3.5.1 Spotlights

The purpose of paper X is twofold: to propose two fast distribution functions for spotlights and to use terminology used in stage lighting to model these luminaries. In OpenGL and other API s the original Warn model is used where the light distribution is computed using a power function. In professional modeling tools, a linear or a cubic function is often used. We propose the use of two different quadratic functions instead that will make the computation involved faster than using the power function or a cubic function. Moreover it will be more flexible than using a linear function. These functions can be used

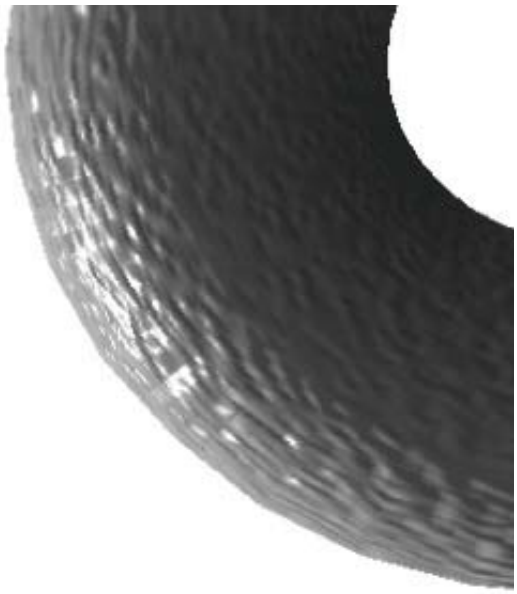

Figure 3.46: A torus where the inside has been rendered using the new method and the underlying geometry is clearly visible.

to model both hard and soft edged spotlights.

The Warn model is quite simple and the type of spotlights that can be simulated is rather limited. By using different functions it is possible to obtain light distributions that mimics both soft and hard edge types of spotlights. In stage lighting, different types of luminaries are used in the lighting for theaters and such [27]. Flood-lights usually have no lens and spreads light in a broad angle. Other luminaries have a lens, which makes it possible to focus the light when gobos are used to make patterns in the beam. Some spotlights have very hard edges and can be useful for specials. This is when a single luminaire is brought up on a solo actor for a monologue or similar. Such a spotlight is sometimes called a follow spot. Profile spotlights can be used for these and they can produce both hard and soft edges. The Fresnel spotlight on the other hand produces only soft edges. These two are probably the most common types of spotlights. There are also a number of other spotlights that have characteristics similar to the Fresnel spotlight, but differ in some properties, as explained in the paper.

More sophisticated light models have been developed by others. Barzel [8] shows how lighting using the Warn model can be modified and controlled so that the shape of the light could be changed. Verbeck and Greenberg [72] showed how the light distribution of a real light source could be measured and used for a virtual light source. The measured distribution is showed in a go-

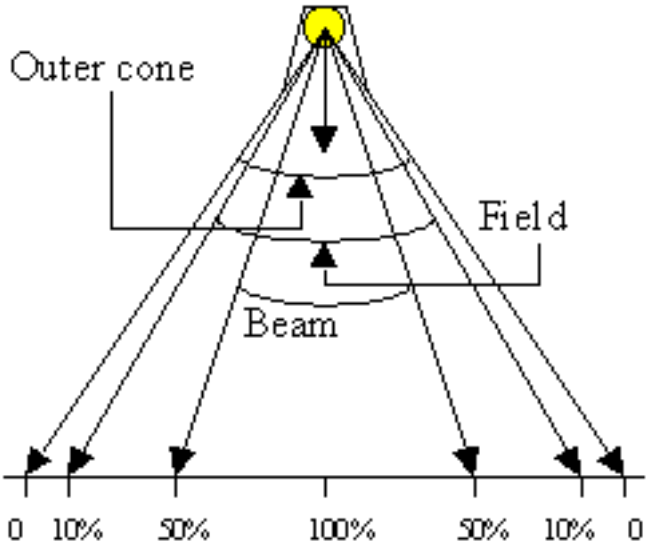

Figure 3.47: Terminology used in stage lighting

niometric diagram. Goesele et al. [30] showed how the luminous intensity distribution of flash lights and similar could be measured and accurately rendered. The problem with accurate measurements is also discussed by Albin and Peroche [6]. Qing and Jizhou [78] propose how interactive editing of light source intensity distribution and rendering using this luminaire could be done. Even though a physically measured light distribution is far better than a simple approximation, it will certainly take more time to compute. Nishita [54] propose interpolation in a table where the luminous intensity data are given, for example, every ten degrees. However, for many real-time applications this will still not be fast enough, especially if the rendering is done in software.

3.5.2 Terminology

The characteristics of spotlights, i.e. the luminous intensity distribution function, is often described using angles measuring the spread of light within the cone. Two important measures are: beam angle and field angle [19]. The beam angle is the angle where the intensity has dropped off to 50% of the intensity along the beam axis, i.e. the main direction. The field angle is the angle where the intensity has dropped off to 10% of the intensity along the axis. These angles are shown in figure 3.47

3.5.3 A soft edge spotlight model

The original Warn model produces a very soft edge spot similar to a Fresnel spotlight. The reason for this is that a power function drops off to close to zero rather quickly. A quadratic function can approximate this behavior quite well. A soft edge spotlight can be modeled by the quadratic function

$$D(\Leftrightarrow) = \frac{(\Leftarrow - \Downarrow)^2}{(1 - \Downarrow)^2} \tag{3.273}$$

where $\Leftarrow$is the cosine of the angle between the main direction, or the beam axis, and the pixel in question. Note that the corresponding angle is only half the angle we describe with the beam and field angles. In order to make the formulas easier to read, we have chosen to use variables for the cosines of the half of the angles in questions. Furthermore, we obtain the cosine of half the angles by the dot product and it is therefore easier to use this notation. Thus, $\Downarrow$is the cosine of half the outer cone angle. The equation will yield zero when $\Leftarrow$is equal to $\Downarrow$and it will be one when $\Leftarrow$is equal to one. We can calculate a value for $\Downarrow$depending on the properties we would like our spotlight to have. This function flattens out when $\Leftarrow$comes closer to $\Downarrow$and therefore it will model a soft edge type of spotlight.

This simple model only allows us to set one of the mentioned angles. Either we set a predefined field angle or beam angle. The other can easily be computed as shown in the paper. Figure 3.48 shows the result and it is clear that the edges are very soft as they are using the original Warn model, which is shown in figure 3.49.

3.5.4 A hard edge spotlight model

Hard or crisp edges can be obtained by a function that does not fade out and yield a function value close to zero when $\Leftrightarrow$becomes smaller. Instead the function has a rather high slope for $\Leftarrow$equal to $\Downarrow$. This can be obtained by the quadratic function

$$D(\Leftrightarrow) = 1 - \frac{(\Leftarrow - 1)^2}{(\Downarrow - 1)^2} \tag{3.274}$$

This distribution will be equal to one when $\Leftrightarrow$is equal to one and it will be equal to zero when $\Leftarrow$is equal to $\Downarrow$. Figure 3.50 shows a scene rendered using this approach. The edges are much harder than using the soft edge model.

The proposed models are much faster than using a power function, which is used in the original Warn model. The scenes in this paper were rendered by using a fragment shader and the Cg language. However it can be mentioned that 1 Billion calls to the power function in the C language (math.h) and the

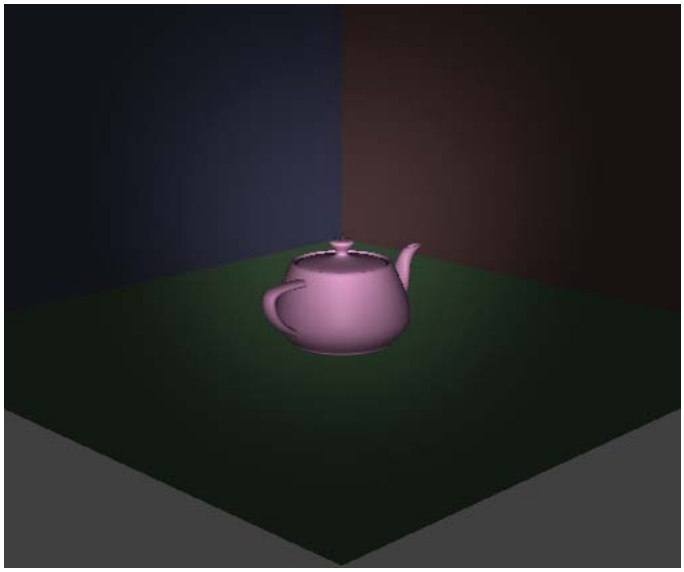

Figure 3.48: A teapot lit by a spotlight using the soft spotlight model.

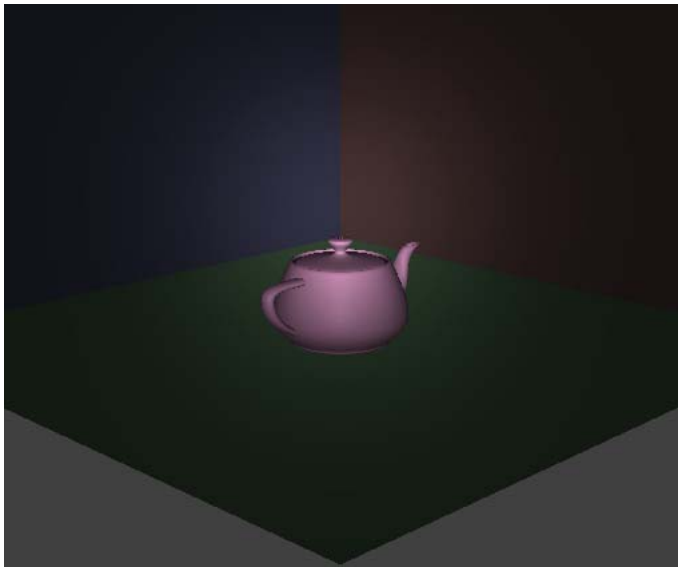

Figure 3.49: A teapot lit by a spotlight using the Warn model.

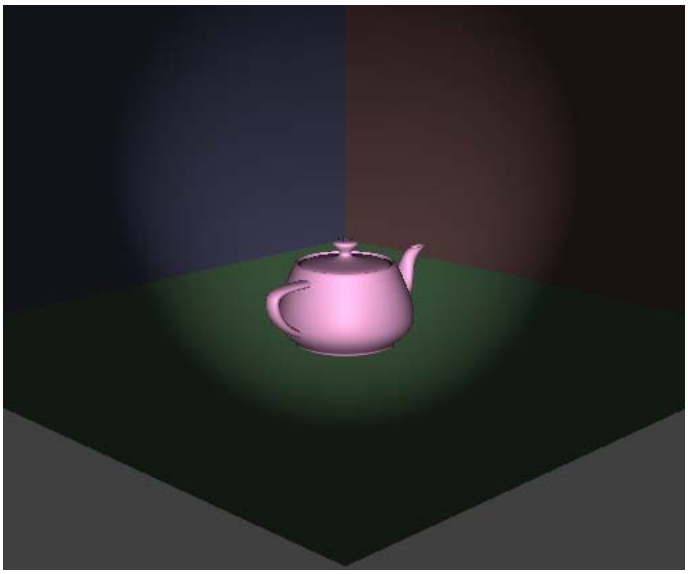

Figure 3.50: A teapot lit by a spotlight using the hard edge model.

proposed method was conducted on a Pentium 4, 1.8 GHz and the power function took 300.27 seconds to execute, while the proposed method for soft edges took 6.29 seconds.

Figure 3.51 shows a Cartesian diagram over the Warn distribution function as well as the proposed functions used in the renderings. A field angle of 15^o was used. Note that the Warn model and the proposed model for soft edges are quite similar. It can also be noted that the hard edge model does not flat out as much as the others do, which gives it its characteristic hard edge.

3.5.5 Highlights

Paper XI propose a new way of computing highlights. Specular highlights are usually modeled with the Phong-Blinn illumination model, as

$$I_s = (\mathbf{n} \cdot \mathbf{h})^s \tag{3.275}$$

where $\mathbf{n}$ is the normal, $\mathbf{h}$ is the halfway vector and s is the shininess coefficient. The evaluation of the power function is computationally expensive and therefore some alternatives have been proposed. Schlick [61] tries to produce the very same result as the power function. However, neither the power function nor the dot product are based on physical behavior of highlights. Therefore, we could as well try some other physically plausible function. We propose such a function which is faster to compute than the power function.

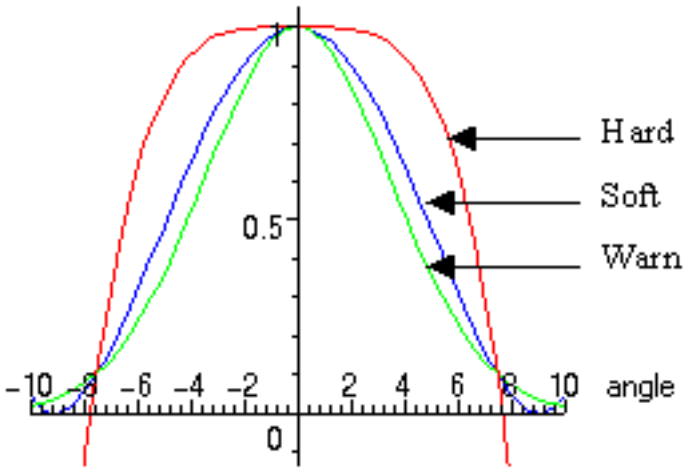

Figure 3.51: Comparison of the Warn model and the new models

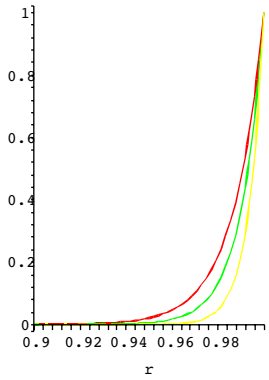

Figure 3.52: Specular intensity for s=75.

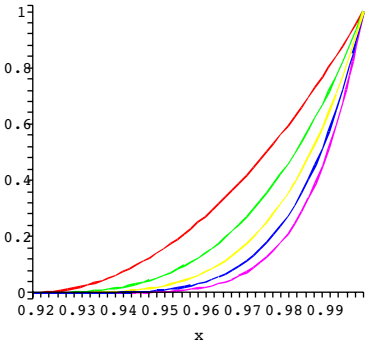

Figure 3.53: The proposed function with d={2,3,4,5,6}

3.5.6 A new Approach

Let $\Leftarrow = (\mathbf{n} \cdot \mathbf{h})$. The result of varying $\Leftarrow$from 0.90 to 1.0 is shown in figure 3.52 for $s = 75$, $s = 100$ and $s = 125$. It is quite clear that there will be no highlight if $\Leftarrow$is less than 0.92 for $s = 75$. For larger s the highlight will be smaller. We could utilize this fact and produce a similar curve with a polynomial of order d, with all roots at $\Downarrow = 0.92$. Let $\uparrow = 1/(1 - \Downarrow)^d$, then our function is

$$f(\Leftrightarrow = \uparrow(\Leftarrow \Downarrow)^d \tag{3.276}$$

where $\uparrow$ assures that $f(1) = 1$.

Figure 3.53 shows four different functions where d varies from 2 to 6. A second order curve will of course be faster than higher order curves. However, a higher order curve will give a highlight with other properties, that might be desirable for certain kind of lighting conditions and materials.

The function $f(\Leftrightarrow$ will be negative for values of $\Leftarrow < \Downarrow$. However, this is no problem because we can simply skip the whole specular computation if $\Leftarrow < \Downarrow$, since this means that we do not have any highlight at all.

The new approach is faster than using a power function. However, it means that we have to define the specular light with $\Downarrow$or as an alternative with both $\Downarrow$and d. Figure 3.54 shows an ordinary highlight where s is varied from 5 to 485. The size of the highlight does not change much in the three last columns. Hence, the size will not vary linearly when s is increased and therefore it is not possible to get very small highlights. These problems are solved with the new approach.

Figure 3.55 and 3.56 shows that the highlight can be varied much more with the new technique.

The presented method for computing highlights in paper XI makes the computation a lot faster by not using the time consuming power function. This is probably replaced by a table look-up by many software shaders. Nonetheless,

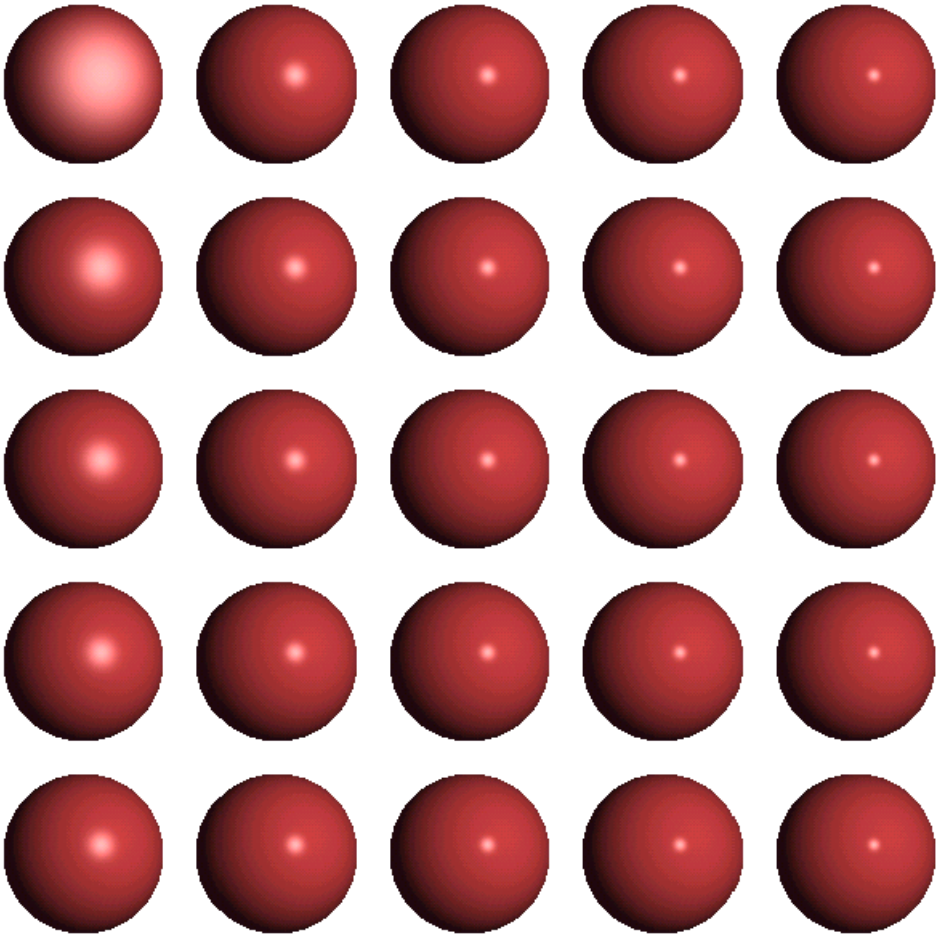

Figure 3.54: Varying s from 5 to 485 in the Phong model

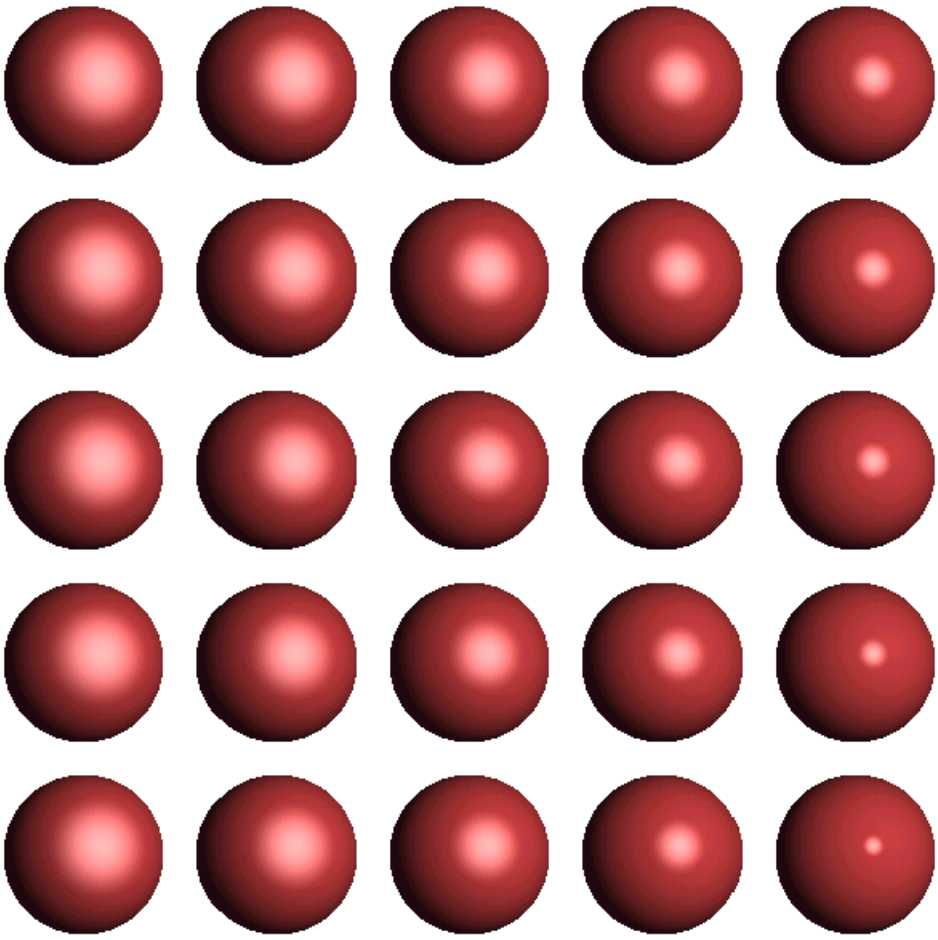

Figure 3.55: Varying ⇓from 0.7 to 0.988

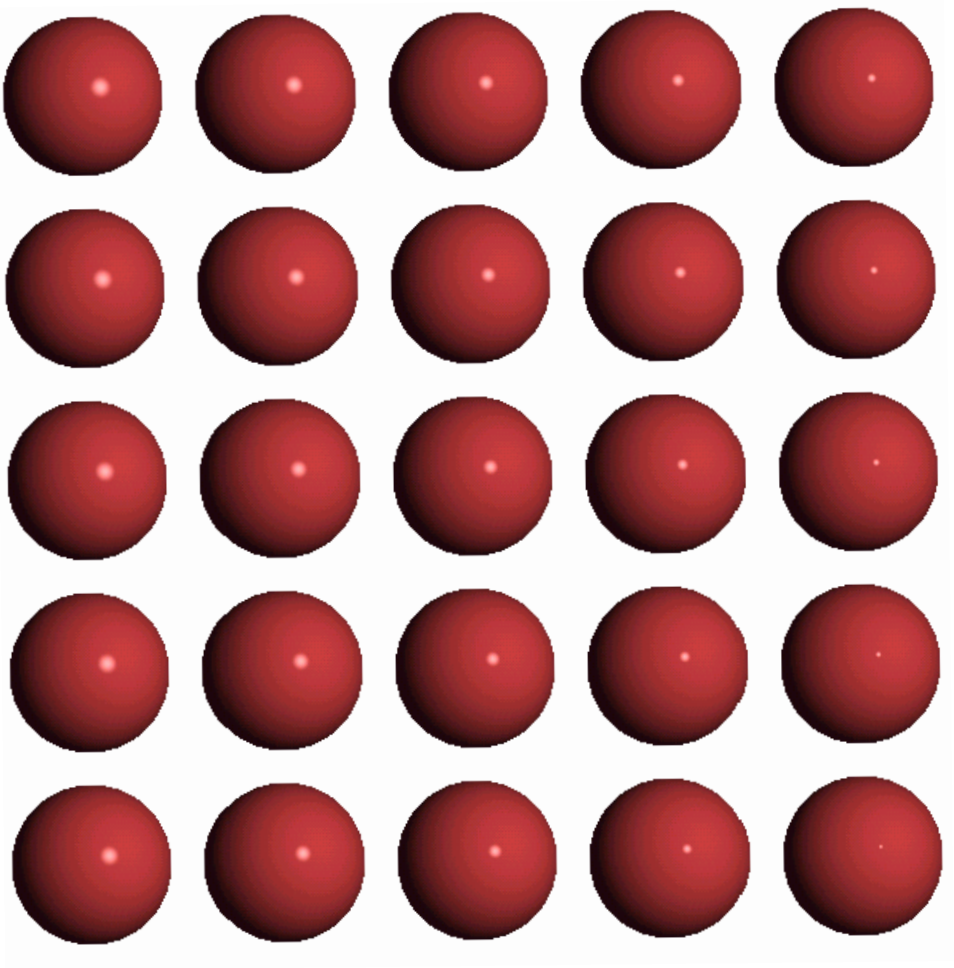

Figure 3.56: Varying ⇓from 0.988 to 0.99952

the new approach will make highlight computation feasible for software implementation due to its simplicity. However, it should also be mentioned that it could also be implemented on a GPU.

Discussion

Computers graphics is today well known to the general public, much thanks to the fact that many families today have a PC and many of those have advanced graphics cards. Moreover, the most popular applications used for home computers have driven the evolution of graphics cards. And those applications are of course, games, not the least 3D games. It is therefore no surprise that some of the most interesting cutting edge research in the area of computer graphics is done by graphics cards developers. Games need to be fast and interactive but also realistic. Therefore, the hardware industry, have put in as much high quality graphics as possible in their cards without compromising speed. Even though there is much to do when it comes to photo-realism, we can expect that as hardware becomes faster, games will include more photo-realism as well.

Another area where computer graphics have been used extensively lately is on the big screen. One of the goals, for many movies that are using computer animation techniques only, is photo-realism. However, non photo-realism includes a number of various techniques, which are also used by artists. Even though graphics cards can do more when it comes to photo-realism, the big screen format is different from the screen format used on home computers. Nonetheless, rendering of computer animation is a time consuming task. A single frame can take hours or even days to render. Therefore, clusters of computers, so called rendering farms, are used by film makers to a non neglectable cost. It is obvious that faster algorithms would be appreciated in this particular field too, since money will be saved, and this reason speaks for itself.

Another emerging field that will include more and more graphics is hand held personal devices, such as palm pilots and cell-phones. Any graphics card on such a device will be much simpler than the card used for gaming on a home computer. The reason for this is that graphics cards are big energy consumers. And energy is crucial for these types of devices since they use some kind of batteries and should be able to operate without recharging for several days. Even though some attempts to construct energy saving graphics cards have been done [40], it is reasonable to presume that software rendering will be used for 3D graphics on many of these devices, at least in the near future.

Some of the current research focus is on how standard graphics hardware could be utilized to speed up algorithms. Whereas our intention is to find new algorithms that could change the way hardware is constructed, as well as be

used on programmable hardware or just be implemented in software. Modern graphics cards are programmable, and some may therefore think that efficient algorithms are not an issue anymore, since the hardware can do it for you, and even do it better. However, the need for research on faster algorithms is still justified. We do not know today what new paradigms might change the way graphics is produced. In the future, faster processors on the mother board, might even make GPU's redundant. This might seem hard to believe today. However, GPU's cost extra money and if money can be saved, they will be removed, when or if that day comes. Nevertheless, better algorithms are also important for software implementations. Especially off-line rendering make use of algorithms implemented in software when high quality renderings are produced. Since current hardware is designed mainly to be fast rather than to produce time consuming renderings, it cannot always be used for off-line rendering. The conclusion is that faster fundamental algorithms are needed either way.

4.1 The fastest type of shading

In our research we have studied different types of algorithms for shading based on scanline as well as polygon setup. We have shown some timings as we have discussed the different methods. But for a final comparison we have timed a number of different approaches on a Pentium machine in order to determine which setup approach is the fastest. Two hundred equal size polygons were rotated ten thousand times. The rotation was done around the z-axis so that the polygons kept their size. The polygons are initially right triangles with the same base and height counted in pixels. The sizes was changed from one to seven and the timing is shown in figure 4.1. Note, that this timing shows both the setup and the following rasterization. However, a fake rasterizer was also timed, which did nothing but traverse all pixels and the time for this was reduced from the time in the figure. Hence, a measure of what each shading algorithm actually does, was obtained.

It is clear that the scanline setup for Phong is slower than the polygon setup for Phong. The polygon setup has been optimized according to the proposed method in Paper VI. Perhaps there still could be some optimizations that we have missed for the scanline setup. Anyway, the trend is clear; scanline setup approaches are generally slower than polygon setup approaches of the same kind.

It was shown in paper IV that a scanline setup approach can be faster than a polygon setup approach. However, they are of different kinds. The scanline setup approach uses slerp evaluated using Chebyshev's recurrence relation where no square roots are used and hence it is faster than the Phong shader using a polygon setup.

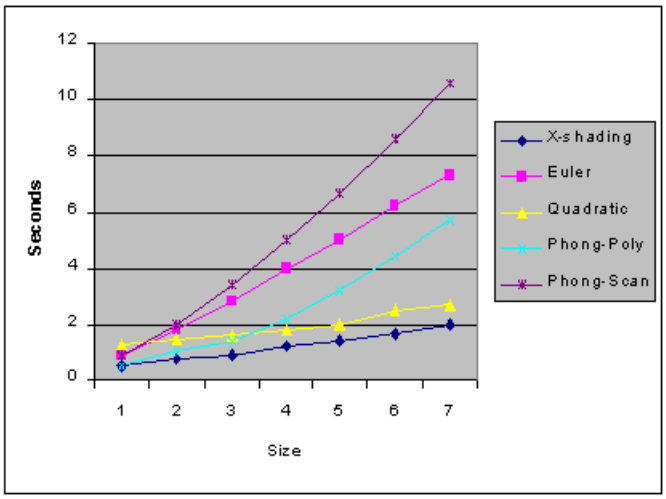

Figure 4.1: Timing of different shading algorithms and different types of setup.

It can also be noted that the polygon setup for quadratic shading is much faster than the scanline setup version. This is of course mainly depending on the fact that the normals at the edges has to be interpolated and normalized. This approach could be made faster by using reflection interpolation or Chebyshev interpolation on the edges. However, it can hardly compete with the X-shading method which is faster than quadratic shading.

4.2 High quality shading

An object that has not been tesselated enough will not appear smooth on the contour. Several of the toruses rendered in this thesis have had very few polygons. Still, the shading produced by for example X-shading produced very similar results as the Phong method. It is clear that the approximation used for the mid-edge vectors are good enough for most cases. The X-shading technique should therefore be suitable for high quality software renderings. It is especially important that the highlight is computed correctly. The diffuse light looks good even when Gouraud shading is used, even though Mach-bands are more salient. The Hybrid shading approach presented in paper I shows this. The X-shading approach could also utilize this fact and compute the specular light with bi-quadratic interpolation and compute the diffuse light with bilinear interpolation. This will of course make X-shading even faster without sacrificing to much of the quality.

4.3 Choice of method

When figure 4.1 is studied it can certainly be questioned why so much work were focused on scanline setup algorithms in the first place. Especially since the polygon setup approach generally is faster. The reason for this is mainly that was not clear from the start that this would be the result. There did not even exist a clear distinction between scanline and polygon setup approaches. Most text books use the scanline setup approach as an example when describing shading. Hopefully our work has contributed to make this distinction clear. Moreover, one of the first polygon setup apporoaches by Bishop and Wimer [11] had a rather high break even point and the setup was rather messy. Therefore, we concentrated our initial research on the inner shading loop and on how the normalization could be avoided. In the beginning we were more interested in counting operations in the inner loop than optimizing the setup.

Was this method wrong? Had it been better to go straight on to polygon setup approaches? The answer is no. The reason for this is that we then would have missed several important results. Most importantly, we would never have considered Chebyshev's recurrence relation for doing slerp, since it is a typical scanline setup approach.

Even though X-shading often will be preferred over this type of shading, the method of using Chebyshev's recurrence relation for doing slerp can be used in all other areas where slerp is used. Hence, a valuable result came from approaching shading by first considering scanline setup approaches. Furthermore, it would not have been discovered that it is possible to do shading using complex numbers as explained in paper III, even though this is more theoretically interesting than useful. Another insight that came from choosing scanline setup approaches first, was that the normalization curve is symmetrical and therefore can be reused for the other half of the scanline. Hence, only half the number of reciprocal square roots are necessary. It should be noted that this fact was not used in the comparison in figure 4.1. The scanline approach for Phong could therefore be made faster. However, the rasterization process must be changed so that the reciprocal square root is reused for the second half which will make the method a bit messier. On the other hand, the method proposed by Ouyang and Maynard is even messier since the scanline is temporarily extended to reduce the number of square roots.

4.4 Future work

The proposed method of using the Chebyshev's recurrence relation for doing slerp is a typical scanline setup approach. Hopefully, it will turn out to be possible to solve how this can be modified to a polygon setup approach in the future.

It could be investigated how other mid-edge vector approximations could be used for X-shading. For instance, with some modification, can the reflection vector technique proposed in paper I be used for this. However, the approximation will include a division and will thus be slower.

Several modifications of the Phong-Blinn illumination model have been proposed in this thesis, which have improved both speed and quality. We believe that there could be similar modifications that could be done for micro facet approaches. For instance, it should be investigated how the fast specular highlight function proposed in paper XI could be used as a micro facet distribution.

Acknowledgments

This thesis would not have been completed without the help and support from many people. In particular, I would like to thank the following people:
⋆ Ewert Bengtsson, my supervisor, for his supervision and that he have let me work on all the crazy ideas I have had, but always leading me on the right path when going astray. I am also grateful for the opportunity he gave me by letting me work together with Tony Barrera.
⋆ Tony Barrera an autodidact mathematician and computer graphics algorithms wizard, but also friend and collegue. K=S!
⋆ KK-foundation for giving me the funding
⋆ All the people at the Centre for Image Analysis. I have not had time to work closely with you, but I always enjoyed my short visits to the centre.
⋆ Creative Media Lab and Univeristy of Gävle for giving me a place to work and also for the support I have got from you.
⋆ My wife, my kids, my parents.... and many more.

Gävle, Mars 2004

References

[1] J. Abouaf, *Inside 3D Studio Max 3*, Chapter 17, Edited by P. Miller, New Riders, pp. 752, 1999.

[2] A. M. Abbas, L. Szirmay-Kalos, T. Horvath, *Hardware Implementation of Phong Shading using Spherical Interpolation*, Periodica Polytechnica, Vol. 44, Nos 3-4, 2000.

[3] A. M. Abbas, L. Szirmay-Kalos, G. Szijarto, T. Horvath, T. Foris *Quadratic Interpolation in Hardware Rendering* Spring Conference of Computer Graphics, 2001.

[4] A. M. Abbas, L. Szirmay-Kalos, T. Horvath, T. Foris *Quadratic Shading and its Hardware Implementation*, Machine Graphics and Vision, Vol. 9, No. 4, pp. 825-804, 2001.

[5] R. A. Adams, *Calculus, a complete course*, Addison-Wesley, pp. 628, 1995.

[6] S. Albin & B. Peroche, *Directionally dependent light sources*, WSCG 2003, Plzen (Czech Republic), February 2003.

[7] E. Angel ,*Interactive Computer Graphics, A Top-Down Approach with OpenGL*, Third Ed. pp. 290, 2003 .

[8] R. Barzel, *Lighting Controls for Computer Cinematography*, Journal of Graphics Tools 2(1), pp.1-20, 1997

[9] B. G. Becker, N. L. Max, *Smooth Transitions between Bump Rendering Algorithms* In Proceedings SIGGRAPH, pp. 183-190. 1993.

[10] U. Behrens. *Graphics Gems IV*, pp. 404-409 (1994, Boston). Academic Press. Edited by Paul Heckbert.

[11] G. Bishop, D. M. Weimer, *Fast Phong Shading* Computer Graphics, vol. 20, No 4, pp. 103-106, 1986.

[12] J. F. Blinn, *Simulation of Wrinkled Surfaces*, In Proceedings SIGGRAPH 78, pp. 286-292, 1978

[13] J. F. Blinn, *Models of Light Reflection for Computer Synthesized Pictures* In Proceedings SIGGRAPH, pp. 192-198. 1977.

[14] J. W. Brown, R. V Churchill, *Fourier Series and Boundary Value Problems* McGraw-Hill, pp. 39-104. 1993.

[15] R. L. Burden, J. D. Faires, *Numerical Analysis*, Brooks/Cole, Thomson Learning, pp. 507-516 , 2001.

[16] E. Catmull, R. Rom, *A Class of Local Interpolating Splines* Computer Aided Geometric design, pp. 317-326. 1974.

[17] Z. J. Cendes, S. H. Wong, C^1 *Quadratic Interpolation over Arbitrary point sets* Computer Graphics & Applications, Vol. 7, No. 11, pp. 8-16, 1987.

[18] M. Doggett, A. Kugler, W. Strasser, *Displacement Mapping using Scan Conversion Hardware Architectures* Computer Graphics Forum, Vol. 20 No 1. pp 13-26. 2000.

[19] Julie O'B. Dorsey, F. X. Sillion, D. P. Greenberg, Design and Simulation of Opera Lighting and Projection Effects, Computer Graphics, Volume 25, Number 4, pp. 41-50, July 1991.

[20] T. Duff, *Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays* ACM, Computer Graphics, Vol. 13, pp. 270-275, 1979.

[21] I. Enrst, H. Rüssler, H. Schultz, O. Wittig *Gouraud Bump mapping* Workshop on Graphics Hardware, pp. 47-53. 1998.

[22] C. Everitt, *Mathematics of Per-Pixel Lighting* http://developer.nvidia.com.

[23] M. Feda, *Improving Intermediate Radiosity Images Using Directional Light*, Graphics Gems V, edited by Alan W. Paeth, Academic Press. pp. 290-293, 1995.

[24] R. Fernando, M. Kilgard, *The Cg Tutorial The Definite Guide to Programmable Real-Time Graphics*, Addison-Wesley. Pp 134-139, 2003.

[25] F. Fisher, A. Woo, *R.E versus N.H specular highlights* Graphics Gems IV, Boston Academic Press, Editor P. Heckbert, pp. 388-400, 1994.

[26] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics - Principles and Practice* Addison-Wesley, 1997.

[27] N. Fraser, *Stage Lighting Design a practical guide*, The Crowood Press Ltd., 1999.

[28] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, L. Israel, *A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories* Computer Graphics (Proc. of SIGGRAPH '89), Vol. 23, No. 3, pp 79-88.

[29] A. Glassner, *Situation Normal*, Andrew Glassner's Notebook- Recreational Computer Graphics, Morgan Kaufmann Publishers, pp. 87-97, 1999.

[30] M. Goesele, X. Granier, W. Heidrich, H.-P. Seidel *Accurate Light Source Acquisition and Rendering*, Proc. of SIGGRAPH '03 (Special issue of ACM Transactions on Graphics), 2003, pp 621-630.

[31] R. C. Gonzales, R. E Woods, *Digital Image Processing* Addison-Wesley, pp. 300-302. 1993.

[32] H. Gouraud, *Continuos Shading of Curved Surfaces*, IEEE transactions on computers vol. c-20, No 6, June 1971.

[33] A. Hast Licentiate Thesis: Improved Fundamental Algorithms for fast Computer Graphics, 2002.

[34] P. S. Heckbert, *Survey of Texture Mapping* IEEE Computer Graphics and Applications, Nov. pp. 56-67. 1986.

[35] C. Hecker, *Perspective Texture Mapping Part I: Foundations.* Game Developers Magazine, pp. 16-25, April/May 1995.

[36] D. Hearn M. P. Baker, *Computer Graphics - C version* Prentice Hall, pp. 345. 1997.

[37] F.S. Hill JR, *Computer Graphics - using OpenGL* Prentice Hall, pp. 659. 1997.

[38] S. Horbelt, A. Munos, T. Blu, M. Unser, *Spline Kernels for Continuous-Space Image Processing* IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings, Vol. 4, pp. 2191-2194. 2000.

[39] T. Ikedo, E. Ohbuchi, *A Realtime Rough Surface Renderer*, Proc. of Computer Graphics International 2001 (CGI2001), IEEE Computer Socity Press, July 2001.

[40] M. Kameyama, Y Kato, H. Fujimoto, H. Negishi, Y. Kodama, Y. Inoue, H. Kawai *3D Graphics LSI Core for Mobile Phone Z3D* Eurographics/SIGGRAPH Graphics Hardware 2003.

[41] M. R. Kappel, *Shading: Fitting a Smooth Intensity Surface* Computer-Aided Design, Vol. 27, No. 8, pp. 595-603, 1995.

[42] M. J. Kilgard *A Practical and Robust Bump-mapping Technique for Today s GPUs* Game Developers Conference, Advanced OpenGL Game Development. 2000.

[43] D. Kirk, O. Lathrop, D. Voorhies, *Quadratic Interpolation for Shaded Image Generation* Patent Nr: US5109481, 1992.

[44] D. Kirk, D. Voorhies, *The Rendering Architecture of the DN10000VS* Computer Graphics vol. 24, pp. 299-307, August 1990.

[45] A. Kugler *IMEM: An Intelligent Memory for Bump- and Reflection-Mapping* Workshop on Graphics Hardware, pp. 113-122. 1998.

[46] A. A. M. Kuijk, E. H. Blake, *Faster Phong Shading via Angular Interpolation* Computer Graphics Forum, vol 8, No 4, pp. 315-324 1989.

[47] Y. C. Lee, C. W. Jen, *Improved Quadratic Normal Vector Interpolation for Realistic Shading* The Visual Computer, 17, pp. 337-352, 2001.

[48] B. Lee, J. Kim, C. Lee, *High Quality Image Interpolation for Color Filter Arrays* IEEE International Conference on Systems, Man, and Cybernetics, Vol. 2, pp. 1547-1550. 2000.

[49] J. E. Marsden, M. J. Hoffman, *Basic Complex Analysis*, W. H. Freeman and Company, pp. 17, 1996.

[50] G. Miller, M. Halstead, M. Clifton *On-the-Fly Texture Computation for Real-Time Surface Shading*, IEEE Computer Graphics and Applications, Vol. 18, No. 2, March-April 1998

[51] D. P. Mitchell, A. N. Netravali, *Reconstruction filters in Computer Graphics* Computer Graphics, Vol. 22, No. 4, pp. 221-228. 1988.

[52] C. Narayanaswami, *Efficient Parallel Gouraud Shading and Linear Interpolation over Triangles*, Computer Graphics forum, Vol. 14, No. 1, pp. 17-24, 1995.

[53] W. K. Nicholson, *Linear Algebra with Applications*, PWS Publishing Company, pp. 275,276, 1995.

[54] T. Nishita, I. Okamura, E. Nakamae, , *Shading Models for Point and Linear Sources*, ACM Transactions on Graphics Volume 4, pp. 124-146, 1985.

[55] S. Ouyang, D. E. Maynard, *Phong Shading by Binary Interpolation* Comput. & Graphics vol. 20, No 6, 1996, pp.839-848.

[56] M. Peercy, A. Airey, B. Cabral, *Efficient Bump Mapping Hardware*, In proceedings of SIGGRAPH 97, August 3-8, pp 303-306. 1997.

[57] B. T. Phong, *Illumination for Computer Generated Pictures* Communications of the ACM, Vol. 18, No 6, June 1975

[58] M. J. D. Powell, M. A. Sabin, *Piecewise Quadratic Approximations on Triangles* ACM Trans. Math. SoftwareVol. 3, pp 316-325, Dec. 1977.

[59] E. Saxe, A. A. Lastra, M. Hughes, *Higher-Order color Interpolation for Real-Time Radiosity Display* UNC-CH Department of Computer Science Technical Report TR96-023, 1996.

[60] A. Schilling, *Towards Real-Time Photorealistic Rendering: Challenges and Solutions* In Proceedings SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware, pp. 7-15, 1997.

[61] C. Schlick, *A Fast Alternative to Phong's Specular Model* Graphics Gems, volume = 4 pp. 385-387, Academic Press, 1994.

[62] L. Seiler, *Quadratic Interpolation for Near-Phong Quality Shading* Proceedings of the conference on SIGGRAPH 98: conference abstracts and applications, Page 268, 1998.

[63] C. E. Shannon, *A mathematical theory of communication* Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948.

[64] K. Shoemake, *Animating rotation with quaternion curves*, ACM SIGGRAPH Computer Graphics , Proceedings of the 12th annual conference on Computer graphics and interactive techniques, Volume 19 Issue 3, July 1985.

[65] M. Sonka, V. Hlavac, R. Boyle, *Image Processing, Analysis and Machine Vision* International Thompson Publishing, pp. 18-22, 1999.

[66] Sudeep Rangaswamy, Visual Storytelling through Lighting, Visual Arts Proceedings, Game Developers Conference, 2000.

[67] J. Sung Kim, J. Hyun Lee, K. Ho Park *A Fast and Efficient Bump Mapping Algorithm by Angular Perturbation* Computers and Graphics No. 25, pp. 401-407, 2001.

[68] K. Turkowski *The use of Coordinate Frames in Computer Graphics*, Graphics Gems Edited by Andrew Glassner, pp 522-532.

[69] K. Turkowski *Properties of Surface-Normal Transformations*, Graphics Gems Edited by Andrew Glassner, pp 539-547.

[70] M. Unser, *Splines - a Perfect Fit for Signal and Image Processing* IEEE Signal Processing Magazine , Vol. 16 No. 6, pp. 22-38, Nov. 1999

[71] J. R. Vacca, *VRML, Bringing Virtual Reality to the Internet*, Academic Press, pp. 243, 1996.

[72] C. P. Verbeck & D. P. Greenberg, *A Comprehensive Light-Source Description for Computer Graphics*, IEEE Computer Graphics and Applications, pp 66-75, 1984.

[73] D. R. Warn, Lighting Controls for synthetic images, Computer Graphics, vol 17, No 3, pp 13-21, 1983.

[74] A. Watt, M. Watt, *Advanced Animation and Rendering Techniques, Theory and Practice*, Addison-Wesley, pp. 52-62, 1992.

[75] L. Williams, *Pyramidal Parametrics*, In proceedings SIGGRAPH' 83, vol. 17, No. 3, pp. 1-11, July 1983.

[76] M. Woo, J Neider, T. Davis,*OpenGL Programming Guide* second edition. Addison-Wesley, pp. 184-186. 1997.

[77] C. Wynn, *Implementing Bump-Mapping using Register Combiners*, Newton-Raphson fast combiner normalization technique. http://developer.nvidia.com.

[78] Q. Xu and J. Sun, *An Implementation to lighting design system*, Proceedings of the International Conference on Imageand Graphics (ICIG' 2000), Tianjin, China, August 2000.

Acta Universitatis Upsaliensis

Comprehensive Summaries of Uppsala Dissertations
from the Faculty of Science and Technology
Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series *Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology.* (Prior to October, 1993, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science".)

Distributor:
Uppsala University Library,
Box 510, SE-751 20 Uppsala, Sweden
www.uu.se,acta@ub.uu.se

ISSN 1104-232X
ISBN 91-554-5916-1