HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information
Examensarbete i datavetenskap 20p (kursnamn+poäng)
C-nivå
Vårterminen 2005

Smooth silhouette rendering of low polygon models for computer games

Kristian Lindström

HS-IKI-EA-06-106

Smooth silhouette rendering of low polygon models for computer games

Submitted by Kristian Lindström to Högskolan i Skövde as a dissertation for the degree of BSc., in the Department of communication and information.

Datum

I certify that all material in this dissertation, which is not my own work has been identified and that no material is included for which a degree has previously been conferred on me.

Signed: _____

Smooth silhouette rendering of low polygon models for computer games

Kristian Lindström

Abstract

This dissertation presents a method capable of smoothing the silhouette of a 3D model using interpolation to find smooth edges. The method has as goal to be used with normal mapping to improve the performance and give a better result with a low polygonal count. To do this the lines located on the silhouette of a model is interpolated to find a curve that is used as clipping frame in the stencil buffer. This method is able to modify the silhouette for the better. The amount of interpolation is rather limited.

Keywords: Tangent spaces, normal map, interpolation, stencil buffer, rendering, Cubic Hermite Interpolation, silhouette edge detection.

Table of contents

1 Introduction

Silhouette smoothing is today an interesting as well as unexplored subject. The idea for this project was first though of when looking at the special case of normal mapped models. The silhouette of a normal mapped 3D-model is probably the technique where the low polygon silhouette becomes obvious; but the silhouette smoothing could well be used without normal maps. Normal Mapping has been an important inspiration to this work, though the 3D-models silhouette becomes too apparent, (based on seeing games like Doom 3, 2004; Far Cry, 2004). The mentioned games uses normal mapped models with a low number of polygons, which leads to a silhouette where the low number of polygons becomes obvious.

The problem addressed in this dissertation is to find a technique, capable of smoothing the silhouette of a 3D-model, preferably in real-time. Further the goal is to find a way of doing this that is good enough for actual use, (i.e. that the frame rate does not drop dramatically when used). The dissertation should not be confused with the work by Sander, Gu, Gortler, Hoppe and Snyder (2000), which apply a high resolution silhouette on an object that uses fewer polygons; whereas this work will concentrate on finding edges on the silhouette and apply a dynamically calculated smooth silhouette. This is done because the method used by Sander, et al. (2000) handles static models, while this method is able to smooth the silhouette dynamically for a 3D-model. Static models imply that its geometry can't be changed, while the geometric representation of a dynamic model does not change how the method works.

Normal mapping is a technique worth looking closely at as a game developer. For a brief description of normal maps see for example Hill (2004). Normal mapping is a great way to create a model, which will look as if the detail were much higher then it actually is (see Figure 1.1, for an illustration of this). In this case the usage of silhouette smoothing would result in a smoother silhouette while retaining a low polygon count. The polygon count of the model could be reduced several times and still have the smooth look of a model with a higher polygon count.

The goal of this dissertation is to find a method that is capable of smoothing the silhouette and do so in real-time. The method presented here is able to do so for dynamic models, (i.e. models where the geometric data may be changed in runtime).

The silhouette clipping algorithm goes through several stages. The implementation presented by this dissertation is using a normal mapped model. Some different silhouette edge detection algorithms are discussed. The data found by the silhouette edge detection algorithm is then used to calculate a smooth silhouette by using interpolation. There are several interesting methods for applying interpolation but in this work the interpolation algorithm was chosen based of the ease of implementation. The original rendered picture of the model is clipped using the stencil buffer with the interpolated mesh as the stencil.

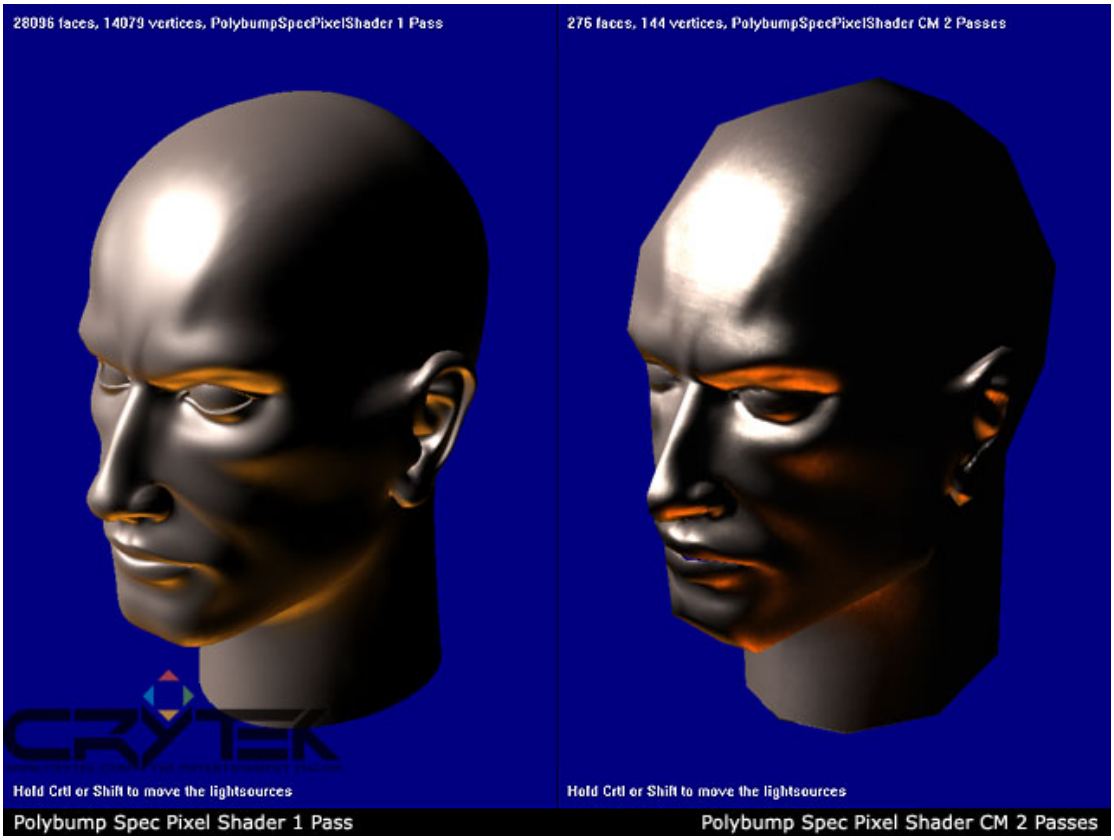


Figure 1.1, to the left is a model with 28 096 polygons, and to the right is a model with 276 polygons rendered using normal mapping techniques. Notice the high detail of the low polygon model at the right, and the hard-edged silhouette. *(Image courtesy of Crytech, Polybump).*

2 Background

This section discusses techniques that are vital in order to appreciate the following work. The section begins with a brief discussion of normal mapping with focus on what normal mapping is. There is some mathematics in the normal mapping process, which will be outlined in this section but not discussed further. In chapter 2.2 follows a discussion about silhouette edge detection in a straightforward manner. In this chapter an example of a silhouette edge detection algorithm is given. This kind of algorithms is not that hard to understand and the discussion is kept brief. Then in chapter 2.3 follows a discussion of interpolation in general and the cubic Hermite interpolation in particular. Last there is an introduction to the stencil buffer and how stencil testing can be used.

2.1 Normal mapping

Normal mapping is a well-known technique that uses a two dimensional texture to store information about the normal at each pixel. The textures red, green and blue pixel values (RGB), corresponds to the coordinates for the normal at each pixel.

Normal mapping has been used in recent games and the outlooks are good for the technique. The attribute that makes normal mapping so interesting for use in games is the ability to light a normal map so that it appear to have much more detail then it actually have. In turn a model with fewer polygons can be used to get a result that looks equally good as one with more polygons. The downside of the technique is that the silhouette of the model will not be improved by the technique and this can be very noticeable; see the silhouette of the image presented in Figure 1.1.

A recent game engine that uses normal mapping to improve rendering quality is Unreal engine 3 (2005) (Figure 2.1). Even though this figure is highly detailed the silhouette artefact can still be seen. For example the head of the model in Figure 2.1 shows this artefact and it can be seen in other parts as well.

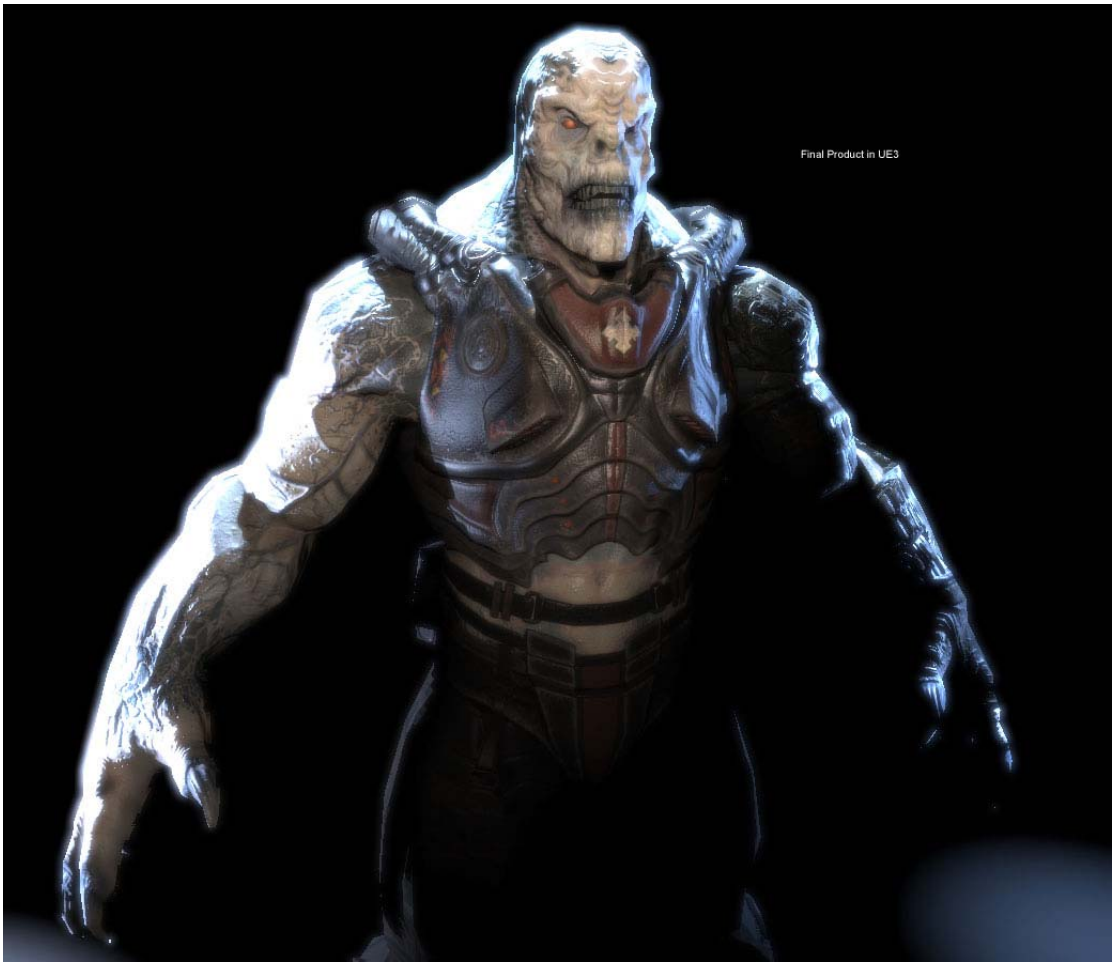


Figure 2.1, this figure shows recent game technology that makes extensive use of the normal mapping technique. This picture is rendered using the game engine Unreal Engine 3 (2005), (image courtesy of Epic Games Inc., this image is Copyright 2005, Epic Games Inc. All rights reserved. Used with permission.).

The basic technique is to light each pixel based on the corresponding value from the normal map, using the dot3 algorithm. The technique is commonly referred to as normal mapping, for more information on the technique see for example Akenine-Möller and Haines (2002).

To start with, the tangent space at each of the vertices must be calculated. Basically the issue of normal mapping is to transform the light position into tangent space, or local space at each vertex. The tangent space coordinate system uses the normalized normal, binormal and tangent at a specific vertex as unit vectors. The tangent space may be pre-computed for each of the vertices. The pre-computed data consists of the normal, binormal and tangent at a vertex (For reference se Figure 2.2). The tangent space does not change unless the geometry itself is changed and this is the reason why it can be pre-computed.

The light per vertex calculations can't be pre-computed and has to be recomputed at least when something has changed in the scene. The light value at a specific vertex can be found by multiplying the vector from the light position to the vertex by the 3x3 matrix that is built from the normal, binormal and tangent, so the matrix is the row major matrix shown in equation 2.1.

$$M = \begin{bmatrix} t_x & b_x & n_x & 0 \\ t_y & b_y & n_y & 0 \\ t_z & b_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The light vector is calculated as the light position minus the vertex position. Then to transform the position into tangent space the matrix M multiplies the light vector.

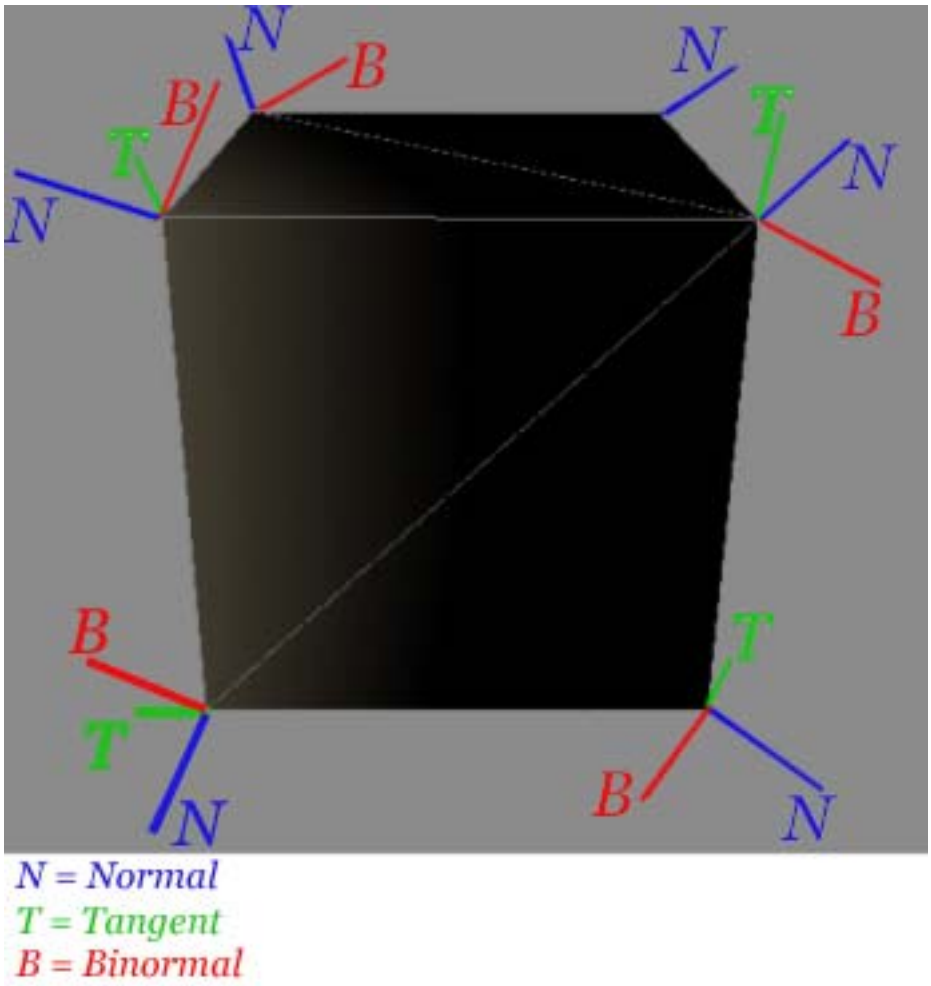


Figure 2.2, shows the tangent space for each of the visible vertices. This uses averaged normals, which gives a smoother model.

2.2 Silhouette edge detection

There are several methods to find a models silhouette; Akenine-Möller et al. (2002) gives some examples of such algorithms, which are mainly used for so-called non-photorealistic rendering. Many of these techniques can be used without knowing the exact details of the silhouette. For example (Raskar & Cohen, 1999) describes a method of rendering a thickened silhouette by first rendering the back faces of the object. The back faces are rendered in a single unlit colour, and the back face object is scaled to be bigger than the original model, or translated towards the viewer. And then the front faces are rendered with lighting and with normal scale. In this way the silhouette is obtained, but to do calculations on the silhouette this is not sufficient.

Buchanan and Sousa (2000) introduced the edge buffer for silhouette finding and this dissertation could be extended to use this technique. The edge buffer is simple in its representation, and built on a binary representation. One entity in the edge buffer is built from one original vertex, a structure containing an integer that represents which vertex ends the current line and a two bit binary. And there are as many structures as there are lines from the vertex. The two bit binary part of the structure is actually a representation of front and back faces. As can be seen here there is at least two key features to the edge buffer that makes it especially attractive to this work; namely that it will not take much of additional space if there are a representation of models that have an edge list represented. Just two extra bits of data per edge, if there are *n* bits of data that represent a models edge list, and there is *m* line segments. In this case there would only be needed $n+2m$ bits of data to represent the edge list as an edge buffer.

The edge buffer is a powerful tool for finding and more importantly to handle a silhouette. But in this work another and more straightforward and easy to implement method was chosen.

To find the silhouette edges of a model some data must be known. First of all there has to be a view position. This is where the viewer is positioned relative one of the vertices that is shared by both neighbouring faces on the edge. This can be calculated as the vector shown in Equation 2.1.

$$\vec{v} = \vec{p} - \vec{w} \tag{2.1}$$

$\vec{p}$ is one of the shared points, and $\vec{w}$ is the view position in world space. To determine if the edge is a silhouette edge or not the normals of the neighbouring faces must be found. The face normals can be found by choosing one vertex of the face and from that calculate the cross product of the vectors drawn from this vertex to the other two vertices, Equation 2.2 and Figure 2.3 shows how this is done.

$$\vec{n} = \left(\vec{v}_2 - \vec{v}_1\right) \times \left(\vec{v}_3 - \vec{v}_1\right) \tag{2.2}$$

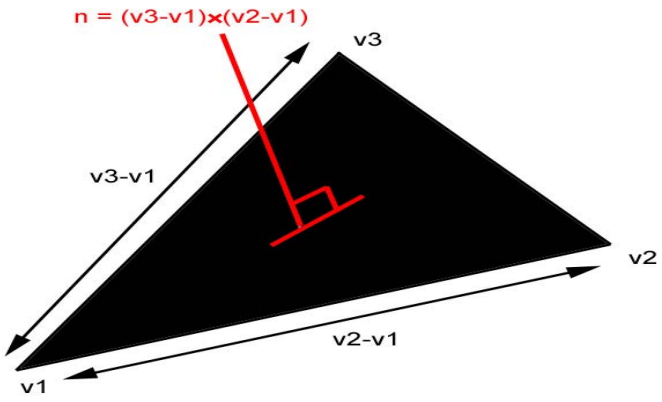


Figure 2.3, the face normal can be derived from the information provided by the vertex coordinates.

This knowledge is sufficient to determine if the edge is a silhouette edge or not. As suggested by Akenine-Möller and Haines (2002) the test shown in Equation 2.3 can be used to determine whether the edge that divides the two faces are a silhouette edge.

$$\left(\vec{n}_0 \cdot \vec{v} > 0\right) \neq \left(\vec{n}_1 \cdot \vec{v} > 0\right) \tag{2.3}$$

Where $\vec{n}_0$ and $\vec{n}_1$ are the neighbouring faces respective normal. This means that if one of the face normals is front facing and the other is back facing in relation to the viewer, then the edge is a silhouette edge.

2.3 Interpolation

One big part of this work is to find an effective way of smoothing the silhouette. In the previous chapter (2.2) a method for finding the line segments that are located on the silhouette was described. The line segments that are located on the silhouette should be smoothed which interpolating over the line segments can do. This chapter presents theory about interpolation and the information that have to be obtained in order to use cubic Hermite interpolation.

Cubic Hermite interpolation has some great advantages; the vertices and the tangent at those points are all that is needed for the interpolation. Further the technique works both for 2D and 3D, which can be of great importance. The implementation of this type of interpolation is compact and easy to handle and that is the reason why it was chosen for this project. Its two end points and the tangents at these points represent cubic Hermite interpolation, as shown in Figure 2.4.

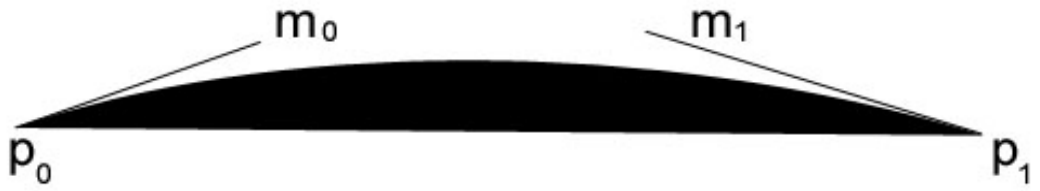


Figure 2.4, shows a cubic Hermite interpolation, where the vertex points are denoted by p_n and the tangents at the points are denoted by m_n

With the notations from Figure 2.4 the function for cubic Hermite interpolation can be derived and it is shown in Equation 2.4.

$$\vec{p}(t) = \left(2t^3 - 3t^2 + 1\right)\vec{p}_0 + \left(t^3 - 2t^2 + t\right)\vec{m}_0 + \left(t^3 - t^2\right)\vec{m}_1 + \left(-2t^3 + 3t^2\right)\vec{p}_1, \quad t \in [0,1] \tag{2.4}$$

For a more thorough presentation of cubic Hermite interpolation see Akenine-Möller and Haines (2002).

Because cubic Hermite interpolation does not care about anything besides the line currently interpolated it can appear as an indentation when used. This is the biggest problem of cubic Hermite interpolation and to get a better flow in the interpolation the use of a more sophisticated method may be required. As mentioned the silhouette of the model is going to be smoothed and the silhouette is the compound of every edge around the model. The cubic Hermite interpolation uses only one line at the same time. When

there is a complex compound of lines such as a models silhouette this might not be entirely sufficient.

Building curves with more than two points implies that there must be a way to handle the shared tangents at the points. A way to deal with the shared tangents is presented by kochanek-bartels curves. The obvious way to do this is to let the line itself contain the tangent information of its vertices, so that the same vertex can have different tangents. This would give a similar representation as the cubic Hermite interpolation, but at the same time more control over the interpolation is obtained. With Kochanek-Bartels curves the idea is to calculate a tangent based on the vertices, to read about the Kochanek-Bartels curves see for example, (Akenine-Möller and Haines, 2002).

2.4 Stencil buffer

The stencil buffer is an additional screen buffer that can be used to perform a lot of effects. The buffer has a binary representation per pixel, where either one or zero represents each pixel. This is often used to clip unwanted areas from being rendered and also to improve the control over rendering to certain areas. It is actually very similarly to how a cardboard stencil is used, hence the name. The stencil buffer is especially useful because of its properties of being able to restrict rendering to a certain area of the screen with pixel precision.

To enable the user of the stencil buffer to control it, the stencil test is used. For more information on the stencil buffer see Shreiner, Woo, Neider, and Davis (2004) or Figure 2.5.

In Figure 2.5 there is shown an example of a model that is rendered as a solid sphere when the stencil test passes where there is a zero. And the model is drawn as a wire frame if the stencil test passes when there is a one in the stencil buffer.

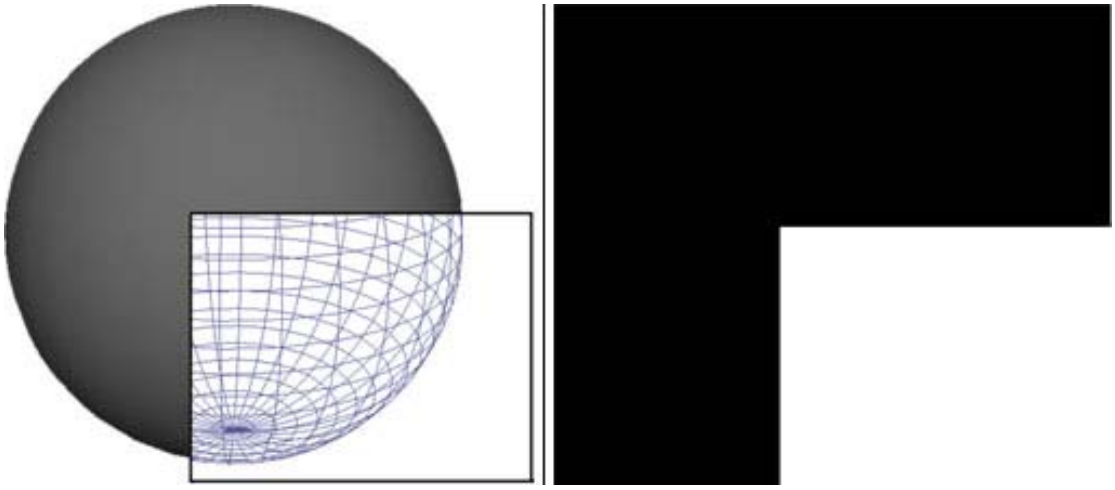


Figure 2.5, to the left the same model is rendered in two passes. If the stencil test should pass for zero (the black area of the right figure represents zeros in the stencil buffer) the solid model is rendered. If the stencil test should pass for one (the white area in the right figure represents ones in the stencil buffer) the model is rendered as a wire frame.

3 Problem

Normal mapping is a technique that has already become important in real-time rendering and game applications. If a method can be created to handle the angularity of the silhouette for normal mapped models, the technique would be able to produce a rendered image of high quality with much less polygons.

In this chapter the problem is described, in chapter 3.1 there is a discussion of what the problem is and why it is considered a problem. Some delimitation to the problem is discussed in chapter 3.2, and the expected results of the work are discussed in chapter 3.3.

3.1 Problem Definition

The problem addressed in this dissertation is to find an algorithm, capable of smoothing the silhouette of a 3D-model, preferably in real-time. Further the goal is to find a way of doing this that is good enough for actual use, (i.e. that the frame rate doesn't drop dramatically when used). This works aim will be to find edges on the silhouette and apply a dynamically calculated smooth silhouette. This is done because a previous method by Sander, at al. (2000) handles static models, while this method is aimed at smoothing the silhouette dynamically for a 3D-model. Static models imply that its geometry can't be changed, while the geometric representation of a dynamic model does not change how the method works. Smoothing a model dynamically would be of interest when for example using skeleton-driven deformation when animating, which is the common method used for character animation.

In the recent work by Loviscach (2004) there is one method that is capable of smoothing silhouettes presented. Loviscach's technique uses the silhouette edges to create a new quadratic geometry extruded from the lines on the silhouette. From the normals and vertex data an interpolation between the two vertices are calculated, and then masked or clipped with the help of a pixel shader. This is an interesting solution to the problem, but the problem with texturing will be introduced. Since the new quadratic geometry is added to the existing model new texture coordinates must be derived and mapped to the model.

In this project another approach to silhouette smoothing will be explored. The hypothesis consists of doing an interpolation in the same manner as Loviscach does. The interpolated data is drawn in the stencil buffer and used to clip the model. This will preserve the lighting of the model, which is important to this project because of the normal mapping delimitations. The downside of this approach is that it may be expensive in computational cost.

By using the stencil buffer to clip the rendered image both texture and the correct lighting will never need to be changed. The greatest profit of this method is that the texture coordinates will not be affected by the technique. This is a profit since everything can be done in a post-rendering phase. The model would be shown correctly while just the silhouette gets a smoother look. The idea of using the stencil buffer to clip the silhouette comes from the work of (Sander, Gu, Gortler, Hoppe and Snyder, 2000), on silhouette clipping.

Figure 3.1 was made to have a reference of how the technique should look. The picture to the left is rendered with the code for this project. The right is the same picture modified using Photoshop (2003). Since the right image is produced as a reference using Photoshop (2003) the application can not be expected to perform as well.

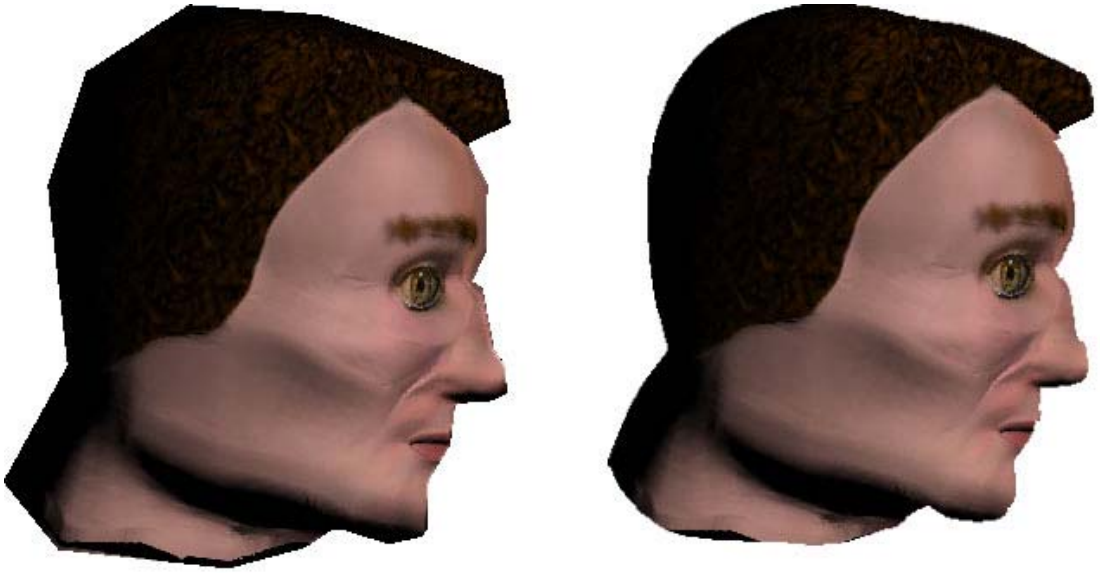


Figure 3.1, an approximation of how the model would look like when the silhouette smoothing technique is applied.

3.2 Delimitation

The work will only consider the problem when using normal mapping, though it could be interesting to apply to different methods, such as shadow volumes or level of detail. The method could be extended to fit any application of silhouette smoothing.

The work only considers silhouette smoothing for convex interpolation. For the other case, concave interpolation to be enabled there has to be a whole other method used. Further it can be hard to decide upon when to use one rather then the other.

The use of shaders were omitted, it could be interesting to extend this work, implementing the technique using shaders. This delimitation is chosen because of the straight forward implementation when using the stencil buffer. And further it is interesting to know how well it performs without the processing power gained from using shaders. If this technique would prove to be good enough without shaders it could easily be implemented using shader programs.

4 Method

The method consists of implementation and evaluation. Implementation is the bigger of the two and everything is produced during the project; such as the application code, 3D-model, textures, normal maps and exporter.

The first steps of the implementation were to create a 3D-model to use, write tools for exporting and to create textures and normal maps for the model. The model was created using Maya (2004). The reason to create the model was because it was easy to make the angularity of the silhouette as obvious as possible. The exporter was written in Maya Embedded Language (MEL) and it uses XML syntax to represent all of the models data. The reason to write a custom exporter script was because it can easily be extended to consider new data as the project evolves. The XML syntax makes it easy to write classes for importing into the application, and since XML is standardised others can easily understand it.

Since much of the work needed for implementing the silhouette smoothing algorithm is outside the scope for the implementation chapter. Such as modelling in Maya (2004), creating textures in Photoshop (2003), writing the exporter in MEL will not be discussed further.

It is hard to evaluate how good the method performs because there is not much easy to evaluate data produced. To create methods to evaluate the success rate is equally hard. So the two methods of evaluation that will be considered are the visual feedback and time consumption or frame rate.

The main evaluation method will be to look at the visual feedback presented by the method. By that is implied that if it looks good it can be accepted as good, or what you see is what you get. If the method can show an improvement in the smoothness of the models silhouette the result is a success.

The value on how much the silhouette has changed using this technique can be measured using data from the calculations. However those values do not give any quality data, just how big the difference is and the interesting part is how it looks.

The time consumption of the technique has a higher degree of measurability, which gives an approach that is easier to measure. This can be sampled both when the technique is used and when the technique is not used. The frame rates are then compared to see how much the frame rate drop when the technique is used.

This method is used to examine the effectiveness of the technique and is as valuable as the visual feedback. When numerical data is produced the testing is easier to evaluate.

The time consumption method will sample the frame rate at different times both with and without the technique and then the frame rates is compared. The comparison gives data that is easy to interpret.

5 Implementation

The implementation of normal mapping is somewhat technically complicated, since it is a big part of the implementation it is presented in chapter 5.1.

In the chapters 5.2 to 5.4 the implementation of the technique used to smooth the silhouette is presented. First there is a short discussion on implementing the silhouette edge detection algorithm. Interpolation is discussed in chapter 5.3 and how it relates back to the silhouette edge detection. There is also a discussion on finding the tangents used when interpolating. In chapter 5.4 the implementation of the stencil buffer is discussed.

5.1 Normal mapping

There are many different methods to calculate the local coordinate systems, but in this project the method is based on using cross products to transform the system. The normal is known, which makes it especially easy. To obtain the tangent the cross product shown in Equation 5.1 is used.

$$\vec{t} = \vec{n} \times \vec{e}_z \tag{5.1}$$

Where $\vec{e}_z$ is the global coordinate system's z-unit vector, and $\vec{n}$ is the averaged, or normalized vertex normal. To obtain the binormal the cross product shown in Equation 5.2 can be used.

$$\vec{b} = \vec{n} \times \vec{t} \tag{5.2}$$

This is almost sufficient, except for the special cases where the normal points down (i.e. the normal y-coordinate is smaller then the vertex y-coordinate). When this is the case the tangent has to be negated. The special case is showed in Figure 5.1.

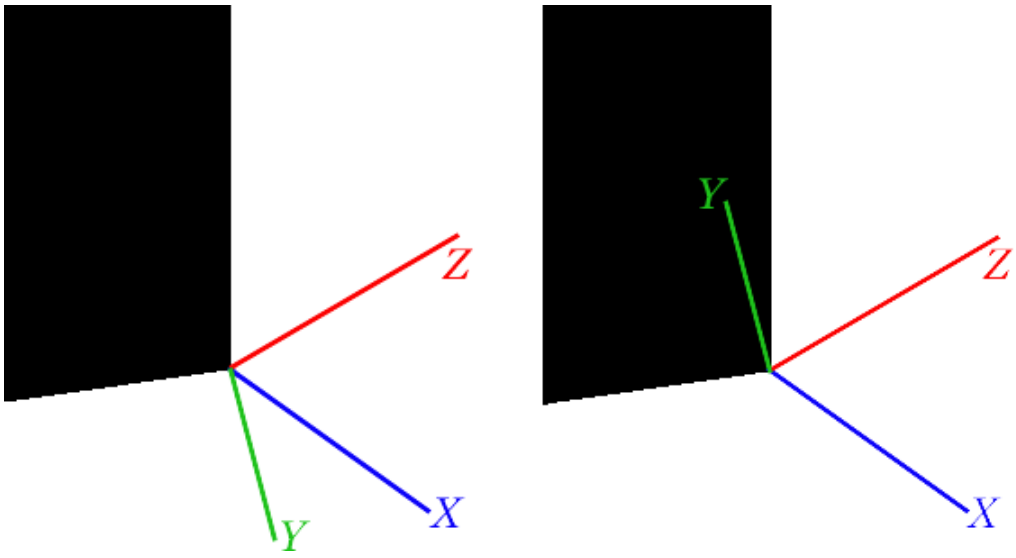


Figure 5.1, the left figure shows the error of the cross product when the vertex normal is pointing in a negative y-direction. The figure to the right shows the correct system that is found by negating the tangent.

The second stage of the implementation is to calculate the light per vertex. This can't be pre-calculated and has to be recomputed at least when something has changed in the scene. In this implementation the vertex light is recomputed every frame. The light value at a specific vertex can be found by multiplying the vector from the light position to the vertex by the 3x3 matrix that is built from the normal, binormal and tangent. As mentioned earlier the matrix is the row major matrix:

$$\vec{M} = \begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix} \tag{5.1}$$

The light vector is calculated as the light position minus the vertex position:

$$\vec{w} = \vec{l} - \vec{\mathrm{v}}_{\mathrm{n}} \tag{5.2}$$

Where $\vec{w}$ are the derived light vector, $\vec{l}$ the light sources position and $\vec{\mathrm{v}}_{\mathrm{n}}$ the vertex that are to be lit. The light value, referred to as $\vec{c}$, at a vertex can be calculated using a vector multiplied by the 3x3 matrix containing the tangent, normal and binormal data as shown in Equation 5.3.

$$\vec{c} = \vec{w} \times \mathrm{M} \tag{5.3}$$

See Figure 5.2 for a reference of how it turns out when rendered.

Figure 5.2, the model used is a compound of 234 faces and the normal map is created from a model with 29 292 faces, using the ATI NormalMapper. In the leftmost figure there is no lighting at all and the model is rendered with the wire frame visible. In the middle figure the model is again rendered without lighting but with a texture. Last in the rightmost figure, the model is rendered with the lighting equation for tangent space light. The model is shown with a normal map and the same texture as in the middle figure. This is rendered using the application for all cases.

5.2 Silhouette edge detection

As mentioned in the background chapter this is a straightforward task, and there are no problems when implementing it. In this implementation every line is tested which gives some overhead. There is one problem with this approach, and that is that there is an artefact when using the stencil buffer (as showed in Figure 5.4).

To solve this problem I chose to create the stencil buffer from the rendered model, so the entire model is rendered to the stencil buffer as shown in the right image of Figure 5.3.

By obtaining this silhouette the lines on the interior that represents a silhouette line will be ignored, (see Figure 5.3 for an example of this). This is done because otherwise we will end up with a graphical artefact where the stencil buffer is rendered on top of the model and this becomes really clear, as showed in Figure 5.4.

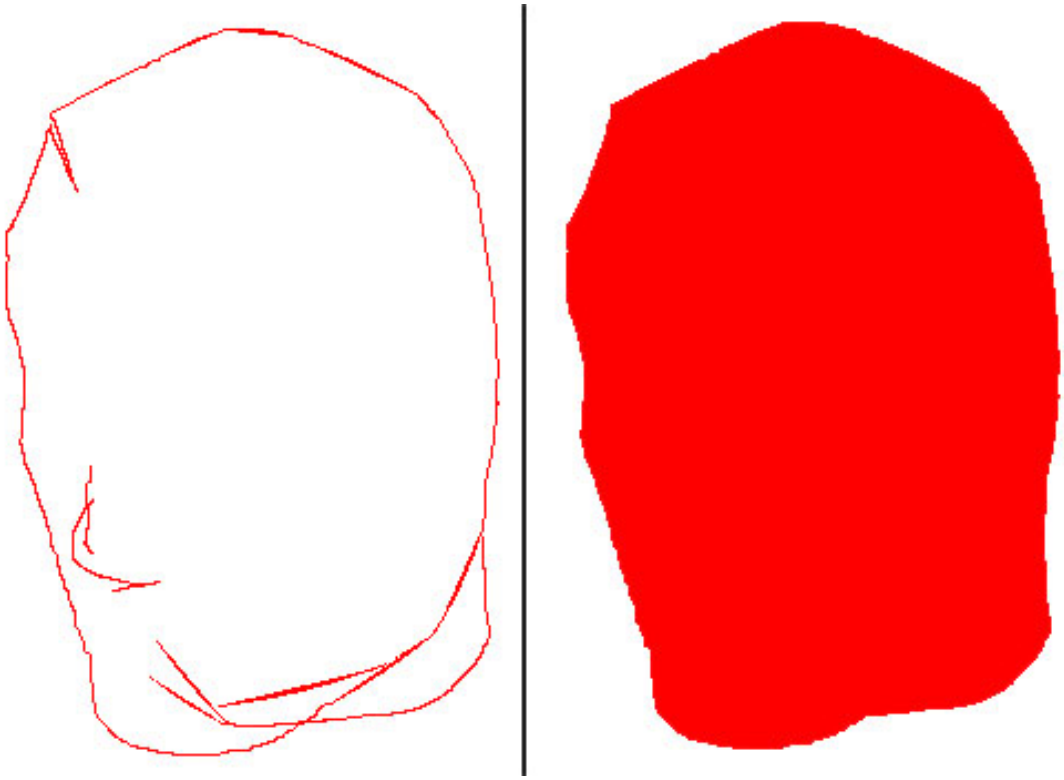


Figure 5.3, to the left is the silhouette found when every line is tested, and to the right is the desired silhouette, that is cleaned and only contains the outline of the model. Both cases is produced in the application.

Figure 5.4, the figure shows an error that the stencil buffer will introduce when the technique is used. There are two approaches to solving this problem, either the lines found on the models interior could be ignored or the whole silhouette could be rendered in the stencil buffer as one polygon.

5.3 Interpolation

The method of interpolation used in this work is called cubic Hermite interpolation or Hermite splines. The Hermite splines were chosen because they are simple and their representation only uses two vertices and their tangents.

For this implementation the vertices are known, (from chapter 5.2) if a line passes the test for silhouette lines, then that line should be interpolated. Knowing the two vertices the only problem is to find the tangents. This is a hard task and to find good values can be tricky.

First of all the tangents have to be adapted to the length of the line segment that is going to be interpolated. This is important because two lines that are not the same length can't

be equally interpolated. On a short line segment the interpolated curve must be smaller, (i.e. not as steep), then that of a long line segment as shown in Figure 5.5.

Figure 5.5, the figure on the left is interpolated with the same tangents as the figure in the middle, which gives an almost round shape. The rightmost figure shows how the left shape should be interpolated to avoid getting a shape that is too round.

However it is not entirely sufficient to take the length into account. There is one special case, which has to be addressed when viewing in three dimensions. When the line has a big length but is viewed from an angle where the points are near in the xy-plane, as shown in Figure 5.6. The same effect tends to occur as seen in Figure 5.5.

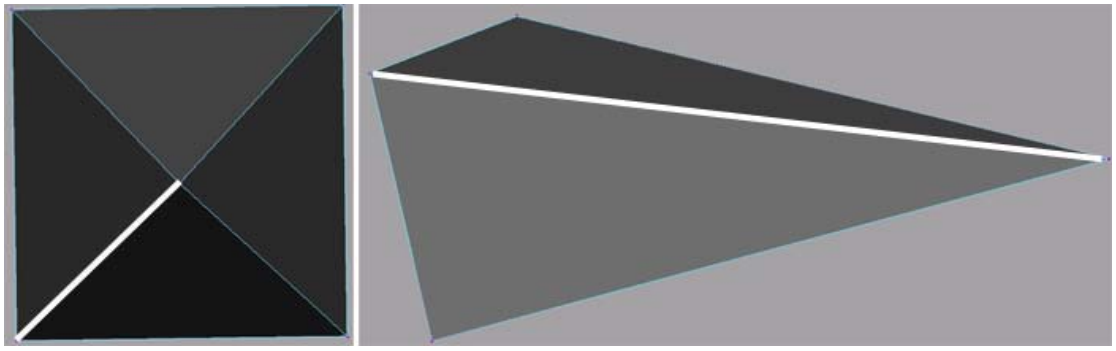


Figure 5.6, the white line is shown form two different view positions. The white line in the picture to the left will appear to be shorter then the line in the right picture.

To address this problem the length of the vector from the two points in the xy-plane viewed from the view position must be taken into account. This means that instead of measuring the actual length of the line segment, the length projected in the xy-plane is measured. With all parameters Equation 5.4 and 5.5 are found to calculate the tangents at each of the two points.

$$\vec{m}_0 = \left(\vec{p}_1 - \vec{p}_0\right) + \left(\vec{n} \cdot C_1\right) \cdot C_2 \left(\sqrt{\left(p_{1x} - p_{0x}\right)^2 + \left(p_{1y} - p_{0y}\right)^2}\right) \tag{5.4}$$

$$\vec{m}_1 = -\left(\vec{p}_1 - \vec{p}_0\right) + \left(\vec{n} \cdot C_1\right) \cdot C_2 \left(\sqrt{\left(p_{1x} - p_{0x}\right)^2 + \left(p_{1y} - p_{0y}\right)^2}\right) \tag{5.5}$$

$\vec{m}_n$ Is the tangent calculated; $\vec{p}_n$ correspond to the vertices on the silhouette line. The value n is the normal perpendicular to the line, and the two constants $C \in R$ are real value scaling constants. The scaling constants are used to control the amount of interpolation applied. The first scaling constant C_1 is used to influence how much of the normal will affect the interpolation. The second scaling constant C_2 is used to influence how much of the line segment length will affect the interpolation. This construction provides the user of the equation with more control since it can be calibrated to suit the model. Using more sophisticated interpolation algoritms could also solve this problem.

These are tailored equations for deciding suitable tangents for the interpolation. The amount of interpolation will be adapted to the line.

To calculate the equation of the interpolated curve Equation 2.4 in chapter 2.3.1 for ease of reading the equation is given again.

$$\vec{p}(t) = \left(2t^3 - 3t^2 + 1\right)\vec{p}_0 + \left(t^3 - 2t^2 + t\right)\vec{m}_0 + \left(t^3 - t^2\right)\vec{m}_1 + \left(-2t^3 + 3t^2\right)\vec{p}_1, \quad t \in [0,1] \tag{2.4}$$

For an explanation of the equation see for example chapter 2.3.1.

5.4 Stencil buffer

The stencil buffer views the model from the cameras view position, and then the interpolated lines are added to the stencil buffer. The interpolated lines are sampled ten times per segment; this should be considered as a high number of samples. The difference when the interpolation is sampled five times is small and would be sufficient. Sampling is when the equation is calculated with different values of the variable $t \in [0,1]$. The variable *t* will range from 0.0 to 1.0 with 0.1 as step length when sampled ten times.

These sampled points are drawn as one polygon per line with ten vertices in the stencil buffer representing the interpolated area around the model. It would be desirable to create a method that finds the silhouette in an ordered manner, see Figure 5.7 for an example of this. If an ordered silhouette could be found then all the drawing to the stencil buffer could be done with one polygon. The interpolation could be calculated from the first vertex and around the ordered silhouette. One way of finding such silhouettes would be to use Andrews Monotone Chain Algorithm, see Andrews, Bender and Zhang (1996).

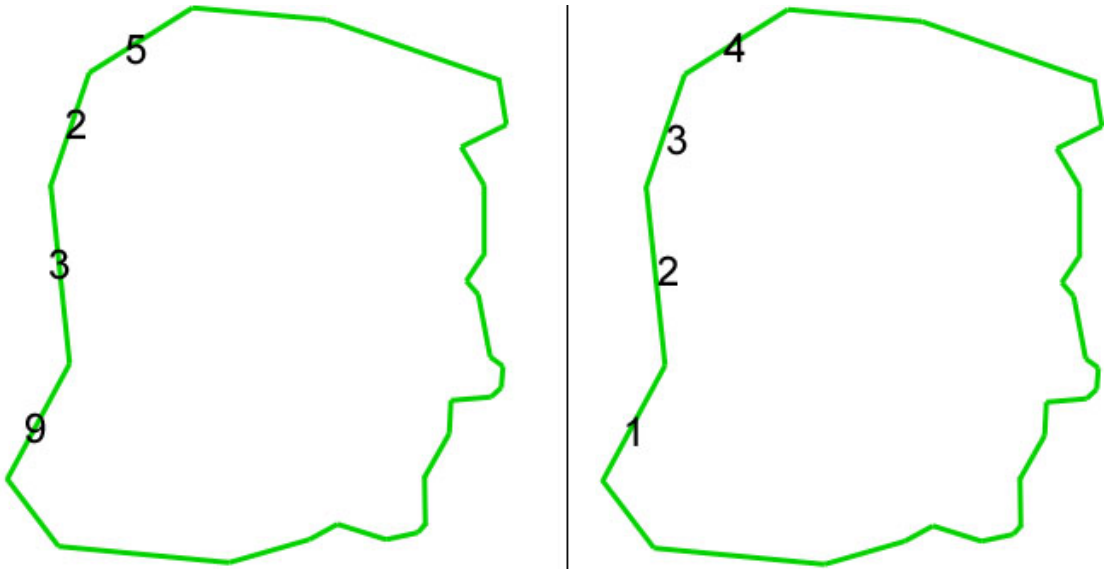
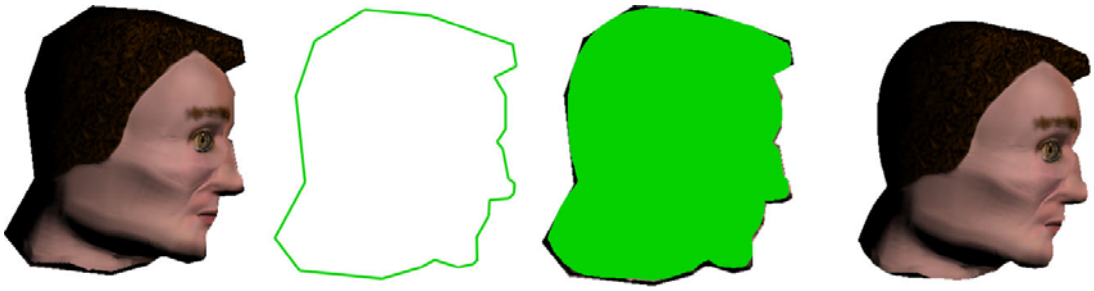


Figure 5.7, to the left is shown a silhouette that is not ordered and to the right is shown an ordered silhouette.

After rendering to the stencil buffer is done, the model is rendered inside the stencil buffer (i.e. where there are ones in the buffer). But since the model is smaller than the stencil buffer the model has to be scaled to be larger then the area of ones in the stencil buffer. Then the parts outside the stencil buffer are clipped leaving the model with the smooth silhouette, this is showed in Figure 5.8.



5.8a, the model

5.8b, the silhouette

5.8c, the model scaled to be bigger than the stencil buffer (green area)

5.8d, the model clipped in the stencil buffer

Figure 5.8, shows how the model is scaled and clipped in the stencil buffer. The model is rendered in the application, and then Photoshop (2003) was used to create the approximation.

6 Result and analysis

As mentioned the methods used to verify the outcome are somewhat hard to choose. In the chapter 6.1 and 6.2 the result will be presented, both the visual feedback, and the measured frame rates. Then the results is analysed in chapter 6.3

6.1 Visual feedback

The method discussed in this dissertation is only concerned with the visual feedback received. Because of this tight coupling to the visual appearance one of the main methods used to verify the success rate is to look at the model with and without the technique and see how well it performs. The goal of the technique is to smooth the silhouette of a model, and if the silhouette look smoother without artefacts the technique have succeeded for this test.

Figure 6.1 depicts a model from two angles both with and without the smoothing technique. Both images are rendered using the implementation. As can be seen the smoothness of the silhouette the technique does smooth the silhouette. And the small steps taken toward the smoother silhouette are looking good, which implies that the technique is working.

When the model is scaled for clipping in the stencil buffer the scaling can only be very small before the model begins to look strange. This is because the model is becoming bigger than the silhouette, so if the scaling is big, vital parts of the model will be clipped. This can't be accepted and this limits the changes in the smooth silhouette.

Another problem with the technique is that in some cases there can be an artefact from the smoothing, as showed in Figure 6.2. This artefact occurs when a line is seen from an angle where the points look like they are closer than they really are. Most of the artefact is resolved by the decision to look at the length of the line in the xy-plane but in some situations where the line segment is located in the xz-plane or yz-plane this artefact can be seen.

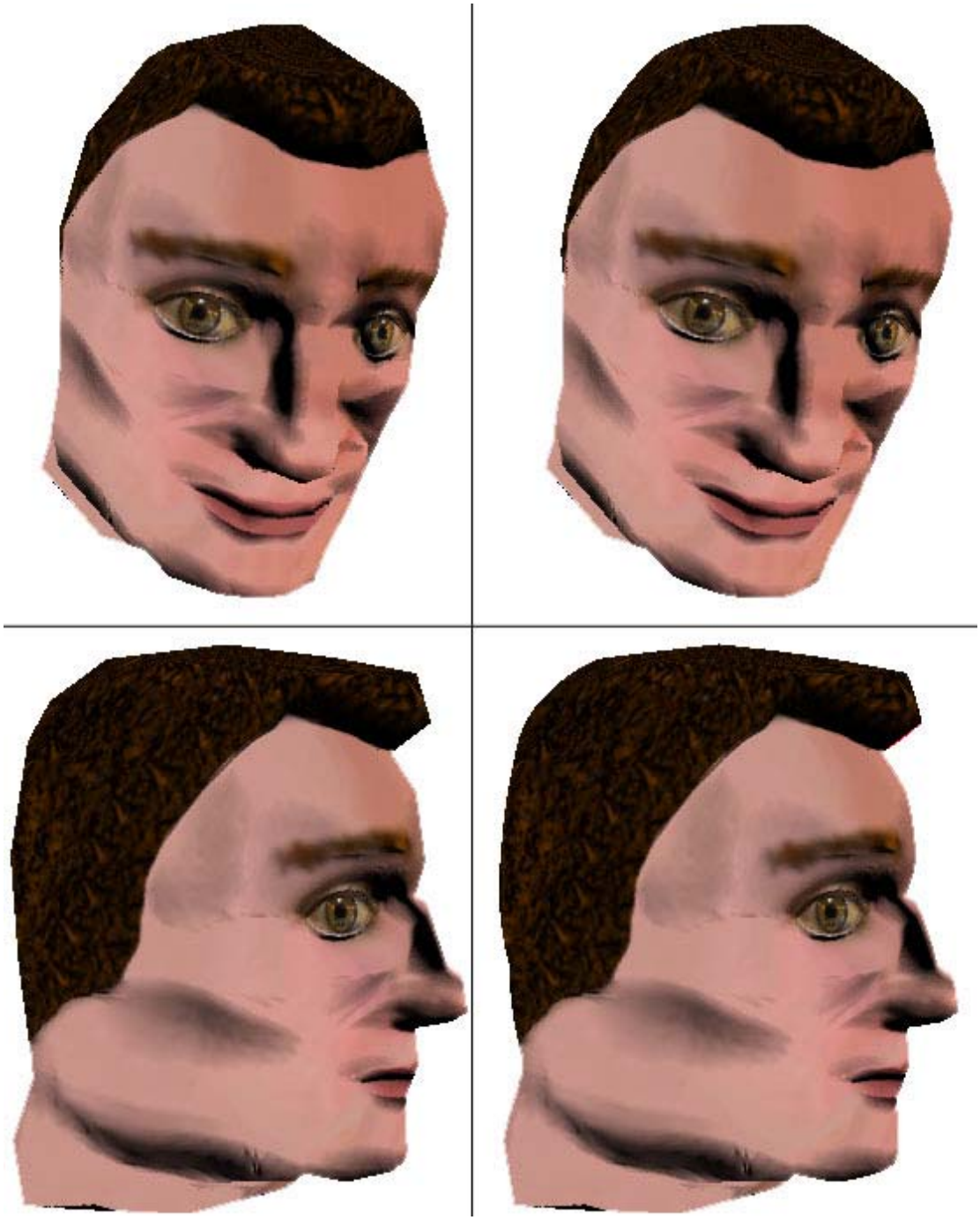


Figure 6.1, the figures to the left shows the model without the smoothing technique and the figures to the right shows the same figure with the technique. The models where the technique is used are clearly smoother.

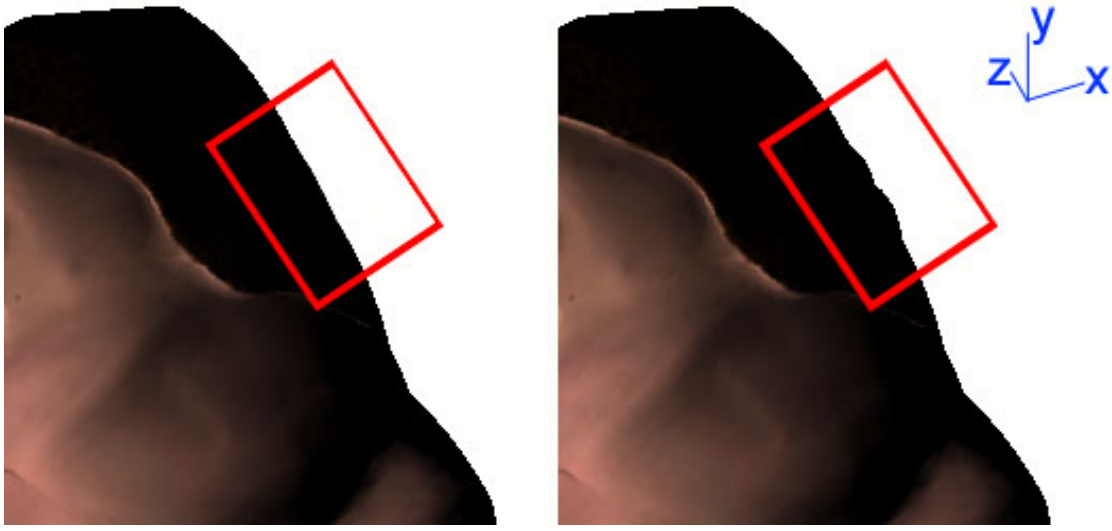


Figure 6.2, the left figure shows how it should look when the silhouette is interpolated. When the line on the silhouette is longer then it appears there can be an artefact. For this case the line has a great length in the z-axis, as shown in the right figure.

6.2 Frame rate

In chapter 6.2.1 a specification of the hardware used in the test is given. And in chapter 6.2.2 the data is presented.

6.2.1 Hardware

The computer that was used for the test is a notebook computer with a P4 2.0 GHz processor, 512 MB RAM, and a 40 GB hard drive. The graphics card is SiS650, with driver version 2.22. The operating system used is Windows XP with service pack 2.

6.2.2 Results

The test is done by first running the application without the technique and save the frame rate with twenty seconds intervals done twenty-five times. And after that the same test is done on the application when the technique is used. The intervals and number of test cases were chosen to give much data to handle. With twenty seconds interval the test is ongoing for eight minutes and potential differences in frames per second (FPS) from long time running can be found. The number of tests, twenty five times is used to get much data to look at, and to see if it changes over time.

The data obtained from the test is shown as numerical data in Table A.1 (in appendix A) and the same data is shown in a more compact form as a dot diagram in Diagram 6.1.

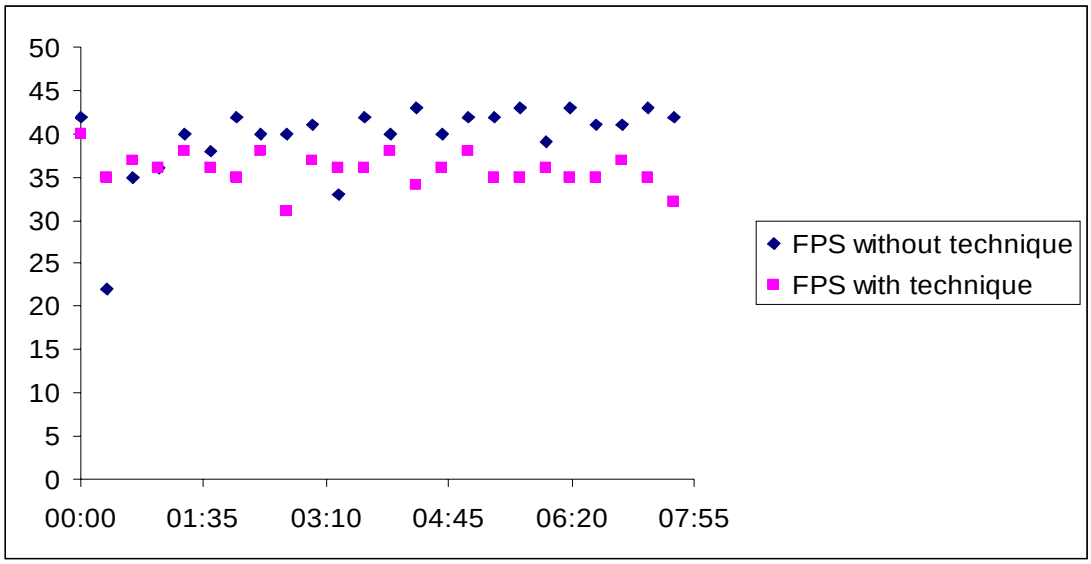


Diagram 6.1, the blue dots show the frame rate of the application when the technique is not used, and the black dots show the frame rate of the application when the technique is used. The samples are stored while running the application over eight minutes with twenty seconds interval.

As seen in Diagram 6.1 there is some loss from using the technique this should be expected and does not come as a surprise. The average FPS for the execution without the technique is 39.72 and the average when the technique is used is 35.96. This is a difference of 3.76 FPS, which is a significant difference. Still I believe that it is good enough for actual use and by optimizing I believe that the difference could be made small enough so that it do not have much importance.

The difference in the two cases where the technique is used and where it is not used is shown in Diagram 6.2.

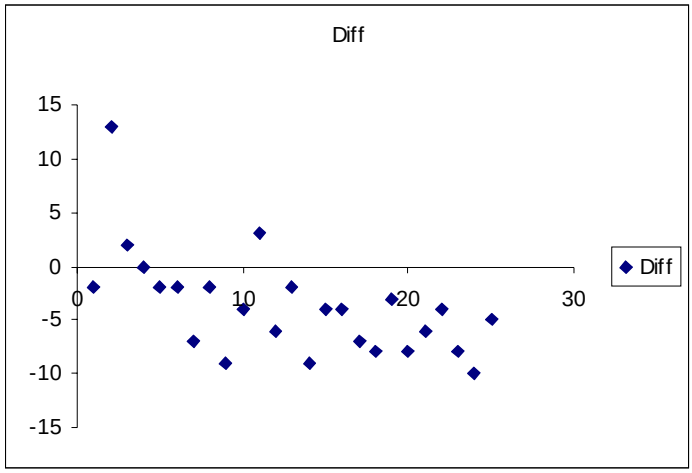


Diagram 6.2, shows the difference in frames per second (FPS) when the technique is used. This diagram shows that the difference is significant and except from some peeks there is a loss in FPS from using the technique.

Diagram 6.2 should be interpreted as the difference of the FPS at every timestamp, where each timestamp entity consists of a twenty seconds interval. The x-axis consists of this timestamps. The y-axis show how much the difference is in FPS and a negative value can be seen as the cost in FPS from using the technique.

6.3 Analysis

From looking at the model when the technique is used a clear improvement in the silhouette smoothness can be seen. This would mean that the test for visual feedback is a success. Though since there is some problems introduced the silhouette can just be smoothed to a certain degree and this is a problem. If the scaling of the model is too big the model will look really strange. Scaling the model always makes it bigger than the silhouette and this is ok to a certain degree. Figure 6.3 shows what this problem might look like.

Figure 6.3, the black frame indicates the actual silhouette of the scaled model. The white frame indicates the smoothed silhouette. In this figure the silhouette is much smaller then the model and the clipped model will look really strange, the effect is exaggerated to try to make the problem as clear as possible. This picture is produced with Photoshop (2003) by scaling the model inside the silhouette but this is how it would look in the application as well.

To understand why this is a problem, consider that with a smoother silhouette there is more interpolation. This means that the interpolated curves maxima will be bigger when more interpolation is added. This in turn means that to fill the bigger area the model must get more scaling in order to fill the entire silhouette. So the degree of smoothness for the silhouette is limited by this fact.

The other test is that of time consumption, or performance. The first diagram did show an average loss in frame rate by 3.76 FPS, this is about 9% of the original frame rate. It is hard to know the exact reason for the drastic frame rate drop and the reasons will be discussed further in the conclusion.

When looking on the difference diagram, there are some high values that would indicate that the technique actually gain rendering speed some times. These peeks should be considered to be noisy. The test was run separately one time when the technique was not used and one time where the technique was used. There could be different level of CPU work in the background or a different amount of hardware resources at the same timestamp for the different test runs. The reason for the noisy data is hard to conclude, but because the two tests are run independently there is reason to believe that the data is noisy. And because there are additional calculations when the technique is used it would be strange if the FPS were higher when more work is being done. The data in general show that there is some loss from using the technique and that should be expected, so the peeks are probably just due to coincidence.

7 Conclusions

This dissertation has discussed a technique that dynamically could smooth a silhouette of a model in 3D. The word dynamically in this context means that the models geometry may change and is of no real importance. The implementation is done by clipping the model in the stencil buffer. The smooth silhouette is calculated using an interpolation algorithm. The evaluation methods that were used were visual feedback and measuring of frame rate. Visual feedback was used since the whole method is used to make a model look better. Frame rate have a high measurability and that is the reason that that method was interesting to use.

The implementation is rather straight forward and most of the problems could be solved with known techniques. For interpolation cubic Hermite splines were used, since they are easy to work with and implement.

The technique performs well, but there are problems introduced by the techniques as well. One problem is that the amount of smoothing that can be applied is limited by the line segment length. This is a problem and it's hard to find a solution for it since the model must be scaled in order to be clipped. If this were to be solved the texture coordinate would have to be re-calculated. The other problem that was found when the visual feedback test was done was when a line segment is longer than it appears. This problem can be solved by projecting the lines in the xy-plane and calculate the length independent of the z-coordinate.

The frame rate suffers some decrease when the technique is used. The frame rate is stable and the decrease is small, though maybe not small enough to be neglected. It is hard to know why the frame rate drop is so big. One reason could be that there is rather much drawing operation in the implementation. By optimizing the problem can be made smaller.

There are some problems with the technique. The two big problems are that the amount of interpolation is limited and that there is some decrease in frame rate. The problem with the amount of interpolation can not be solved unless the project is extended by for example recalculation of the texture coordinates. Since the problem is that the model becomes bigger then its own silhouette the texture would have to be changed to compensate for that.

The frame rate problem might be solved by optimization. Every polygon is rendered twice, once to the stencil buffer and once to the screen. This is bound to give some overhead. The drawing to the stencil buffer only needs to draw everything inside the smooth silhouettes to the stencil buffer. It would save rendering time if the smooth silhouette could be drawn as one polygon to the stencil buffer. To do this the silhouette must be ordered, and this could be implemented and would certainly be an optimization.

The technique does however smooth a silhouette of a 3d-model and it does this dynamically and in realtime. The goal of the method is achieved and the model actually looks much better when smoothed. The problems introduced is not impossible to solve completely or partially.

The result of the technique is shown in Figure 7.1

Figure 7.1, the left picture shows how the model looks before applying the technique and the figure to the right show the model with the silhouette smoothing technique.

8 Future work

I would like to recommend another way of implementing the interpolation function. Instead of accepting every line that passes the test an additional test is done to see if the line is placed on the interior, if that's the case it should be ignored. This test would then be done for every line that passes the silhouette edge test. Instead of finding every line this would just find the outline of the model. In this way we obtain a silhouette that is connected which will prove more useful when implementing the stencil buffer (discussed in detail in chapter 5.5). This implementation would have to find the ordered silhouette as discussed in chapter 5, if this is done then the silhouette is found as a closed ordered silhouette around the model. With this information the clipping area in the stencil buffer could be drawn with one polygon, and this would improve the frame rate. There is just one polygon to render to the stencil buffer, which represents the entire model. In this work the whole model is rendered and this gives a large overhead of rendering just about all polygons of the model. As a reference the model used in this project consists of 234 polygons, which are rendered to the stencil buffer. With the approach where one polygon can be drawn we would save 233 polygon-drawing operations.

This implementation is done without any shaders, and it could be of great interest to implement big parts of the technique with shaders. The normal mapping is mainly done with shaders in modern applications. And there could be made great optimizations if shaders were used.

One more development that could be interesting to use is to try different interpolation algorithms. There are more sophisticated interpolation functions that could be interesting to use, for example it could be very rewarding to have a function that handle shared tangents as Kochanek-Bartels curves, (described by Akenine-Möller and Haines, 2002).

Stencil shadow could probably use this technique to improve the rendered shadows. Since stencil shadows uses some of the theory used in this work, the leap of using this technique would be small. It should be said that there are good existing methods to improve shadow techniques such as soft shadows that in most cases would be sufficient. For information on stencil shadows I would recommend reading the work by Kainulainen, 2002 which gives an up to date and thorough discussion of the subject.

Level of detail (LOD) algorithms could gain from the use of silhouette smoothing; this could be used to improve the silhouette of a model to make an object more legible when viewed at a great distance. The work by Sandlers et al. (2000) could be of great help when implementing this and it should not be a big step to implement. For more general information about LOD algorithms read the work by Hoppe (1996).

By implementing a technique for recalculating the texture coordinates the silhouette smoothing could be unlimited. The recalculation could be a really interesting extension of this work, since this is the big limitation of the work.

Evolving upon this technique and implement it with anti aliasing could give some interesting results. This could be a merge with anti aliasing helping both silhouette smoothing and anti aliasing.

9 References

Akenine-Möller, T. & Haines, E. (2002) *Real-time Rendering*, A K Peters Ltd.

Andrews M., Bender M.A. & Zhang L. (1996) *New algorithms for the disk scheduling problem*, 37th Annual Symposium on Foundations of Computer Science (FOCS '96)

Buchanan, J. & Sousa, M. (2000) *The Edge Buffer: A Data Structure for Easy Silhouette Rendering*, NPAR 2000: Symposium on Non-Photorealistic Animation and Rendering, pages 39–42, June 2000

Far Cry, (2004) [Computer program], CryTech

Hill, S. (2004) *Hardware Accelerating Art Production.* Available at Internet: http://www.gamasutra.com/features/20040318/hill_01.shtml [Accessed 06.05.16]

Hoppe, H. (1996) *Progressive Meshes*, Microsoft Research.

Kainulainen, J. (2002) *Stencil Shadow volumes.* Technical report. Telecommunications Software and Multimedia Laboratory. Helsinki University of Technology. Finland.

Loviscach, J. (2004) *ShaderX3: Advanced Rendering with DirectX and OpenGL*, Charles River Media.

Maya 6.0, (2004) [Computer program], Alias | Wavefront.

NormalMapper [Computer program], ATI Technologies Inc. Available on the Internet http://www.ati.com/developer/tools.html [Retrieved 06.05.16]

Photoshop 7, (2003) [Computer program], Adobe

Polybump [Computer program], CryTech. Available on the Internet http://www.crytek.com/downloads/index.php?sx=polybump [Accessed 06.05.16]

Sanders, V. S., Gu, X., Gortler, J. S., Hoppe, H. & Snyder, J. (2000) *Silhouette Clipping*, ACM SIGGRAPH 2000 Proceedings, pages 327-334

Shreiner, D., Woo, M., Neider, J. & Davis, T. (2004) *OpenGL Programming Guide fourth edition*, Addison-Wesley

Raskar, R. & Cohen, M. (1999) *Image Precision Silhouette Edges*, 1999 Symposium on Interactive 3D Graphics, pages 135-140

Unreal Engine 3, (2003) [Computer program], Epic Games Inc.

10 Appendix A

Timestamp	FPS without technique	FPS with technique
00.00	42	40
00.20	22	35
00.40	35	37
01.00	36	36
01.20	40	38
01.40	38	36
02.00	42	35
02.20	40	38
02.40	40	31
03.00	41	37
03.20	33	36
03.40	42	36
04.00	40	38
04.20	43	34
04.40	40	36
05.00	42	38
05.20	42	35
05.40	43	35
06.00	39	36
06.20	43	35
06.40	41	35
07.00	41	37
07.20	43	35
07.40	42	32
08.00	43	38

Table A.1, the table shows how the frame rate changes over eight minutes with an interval of twenty seconds.