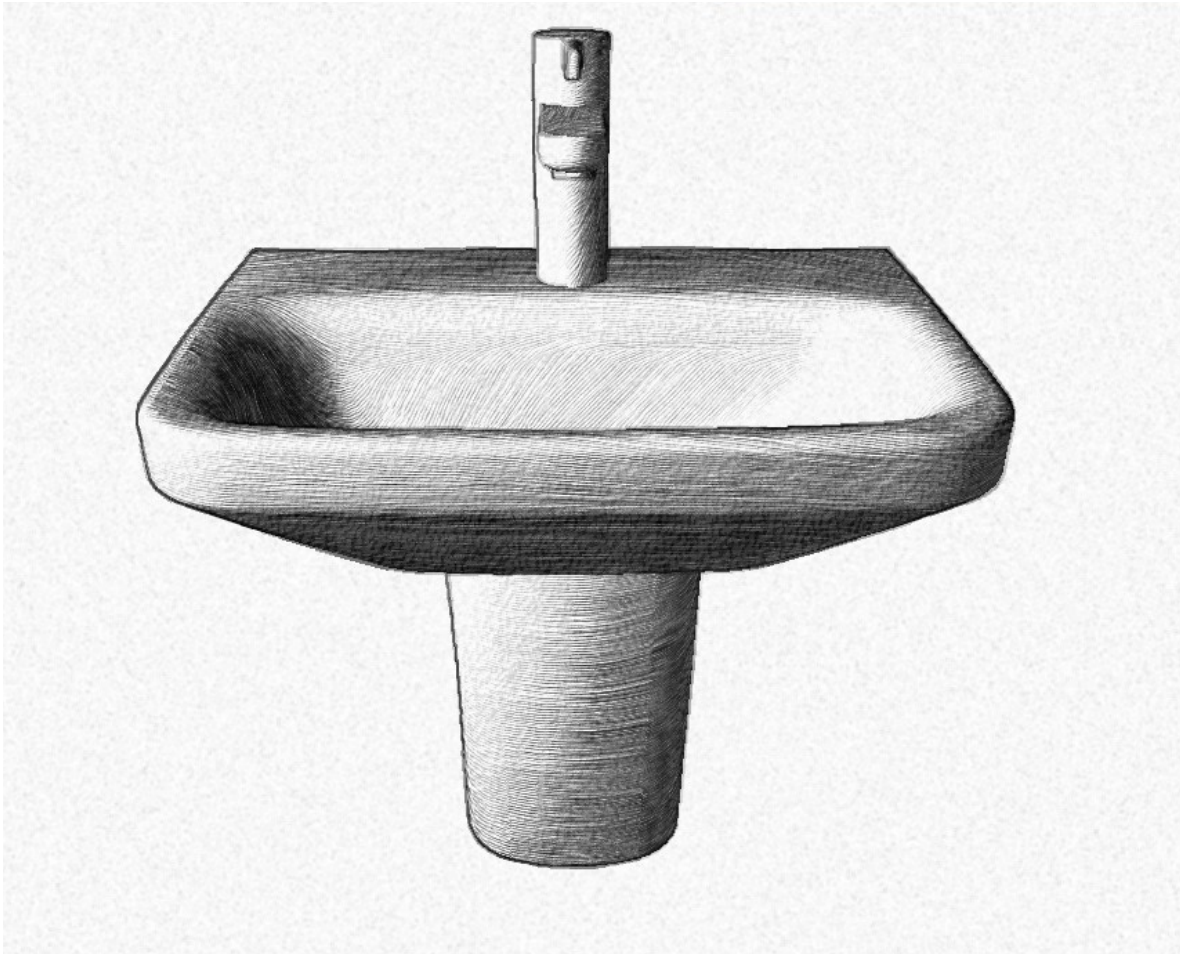

Real-time sketch rendering

A system for non photo realistic rendering, mimicking graphite pencil drawings.

Master's thesis in Algorithms Language and Logic

GUSTAV ÖRNVALL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

Real-time sketch rendering A system for none photo realistic rendering, mimicking graphite pencil drawings by artists.

Gustav Örnvall

Examiner: Ulf Assarsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: A sink model that has been rendered by the system described in this thesis.

Department of Computer Science and Engineering
Göteborg, Sweden May 2014

Abstract

This paper presents a real-time technique for rendering 3D surfaces in a pencil drawing style. It is fundamentally based on a paper which introduces a real-time pencil rendering system (Lee et al., 2006). The authors uses strokes drawn along the principal curvature directions to shade the interior of a 3D surface. The main focus of this thesis has been to implement a sketching system with a good support of arbitrary surfaces which are not uniformly tessellated and to provide results on par with the paper (Lee et al., 2006). To handle arbitrary surfaces which are not uniformly tessellated, a new way of computing a curvature direction field has been implemented. This paper presents an implementation which visualize the outline of a surface and manages to sketch the geometry of complex surfaces in good quality.

Acknowledgements

I would like to thank my supervisor Erik Sintorn from Chalmers University of Technology for regularly feedback and comments on my work. I would also like to thank those selected friends who helped me proofread the report.

Gustav Örnvall, 29/5/14

Contents

1

Introduction

This thesis describes the implementation of a real-time sketch rendering algorithm which takes into consideration some of the techniques used by artists when producing a graphite sketch of an object. These sketches often include overlapping strokes for shading and contour drawings for outlining an object. It was found that the direction of a stroke is of great importance to visualize geometry. To find the stroke direction an existing algorithm to compute this has been investigated by (Alliez et al., 2003) and it was found that the quality was very dependent on the tessellation of a mesh. In a real world application one might not have the meshes of such a quality, which led to that this thesis introduces an alternative heuristic to compute these direction fields and it also presents a complete system to render an input mesh as a graphite sketch. The problems to solve are,

- Draw the outline of a surface
- Draw the interior considering the geometry
- Make it visually appealing

1.1 Background

For the past decades the research area of computer graphics has been driven towards photorealism. An area where images created by a computer is judged by how closely they resemble a photograph. These images are rendered by a physics-simulation, by computing the behaviour of light inside a scene. On the other hand in a non photo realistic rendering, the images are judged by how expressive they are or by how closely it resembles the work of a human artist (Gooch and Gooch, 2001). When presenting a scene in non photo realistic rendering, a reality simulation is not as important as creating an illusion of reality and the stylization is driven by human perception. By highlighting key features and abstracting away details non photo realistic rendering manages to produce powerful and expressive images, which are very well suited for blueprints and sketches.

1.2 Delimitations

The thesis has been proposed by a company residing in Gothenburg with a wish to enhance one of their products by being able to present a blueprint in a hand sketched fashion. The real-time behaviour is still desired after being enhanced by a hand sketching effect and thus, sets a performance limitation on the implementation.

1.3 Related work

In the field of non-photo-realistic rendering, a variety of techniques have been developed to provide rendering of physical materials in a pen style.
(Hertzmann and Zorin, 2000) introduce a rendering technique for automatic generation of line-art illustrations of piecewise-smooth free-form surfaces. They address two general problems, computing silhouette curves of smooth surfaces and generating smooth direction fields on surfaces used as a foundation to draw lines to visualize geometry.

(Majumder and Gopi, 2002) introduced a real-time rendering of 3D objects in a charcoal fashion, by focusing on contrast enhancement techniques accelerated using graphics hardware. Their algorithm applies a contrast operator on a noise texture to generate a charcoal grain texture, and the same operator to modify a grey scale Phong shading model. The contrast enhanced grey value of a vertex is used to index their grain texture for mapping the texture onto the original 3D model.

(Webb et al., 2002) discussed two real-time hatching schemes, which leverage for that time, advances in texture mapping hardware. Both their schemas provide enhanced control of tone, and with that avoid aliasing effects compared to previous hatching schemas. Their work is an extension of their own previous work on tonal art maps (Praun et al., 2001) Where a tonal art map can be seen as a levelled texture structure that capture hatching at various tones and scales. They use multi-texturing to blend several TAM images of a triangle to gain visual continuity.

(Lee et al., 2006) produced a system for real time pencil rendering of 3D objects, in a simpler approach than Majumder and Gopi still producing an impressive result. By further investigating the inherent characteristics of pencil drawing compared to Saito and Takahashi who introduced line drawing on an object surface by computing a direction field on a 2D image to use in mapping orient able textures. They instead of using a 2D image, compute the principal curvature direction on a mesh based on a method presented in the paper (Alliez et al., 2003). They also present a more efficient technique for multiple contour drawing than Loviscah who demonstrated the possibility to draw multiple contours with graphics hardware. By distorting the underlying plane of the contours to produce trajectories.

2

Method

This system has been constructed after a pre-study in the field of non-photo-realistic rendering. Mainly focused on papers presenting methods of replicating hand drawn images and the papers they were combined from. The pre-study resulted in identified effects and techniques that could be used to create a realistic hand sketch. They have been selected by comparing previous work, discussion with both my supervisor at Chalmers University of Technology and my constituent at the company who proposed the thesis, as well as personal experience in graphite drawing.

These identified effects and techniques will be presented in detail in the following sections. The primary needed techniques found to give a visually pleasing result were; being able to render contours, an interior shading mimicking graphite sketching which considers the material its drawn upon and computing curvatures for the surface to be used when drawing graphite strokes.

2.1 Contour drawing

To visualize a 3D model onto a 2D plane as a paper, some kind of volume reference is needed. For a hand drawing, a common way of doing this is to outline the contours of the drawn object.

2.1.1 Procedural Geometry Silhouetting

By modifying and selecting specific parts of geometry, one is able to draw silhouettes. There are several ways of doing this and several of them are based on a technique presented by (Rossignac and van Emmerik, 1992). Their basic idea is to render the front-faces as normal and then render the back-faces in such a way that their silhouette edges are visible.

The various ways of rendering these back-faces, each has its strength and weaknesses. For example, one could only render the edges of the back-faces, using biasing or other

techniques to ensure these lines are drawn in front of the front-faces and by doing so only the lines of the silhouette edges will be visible.

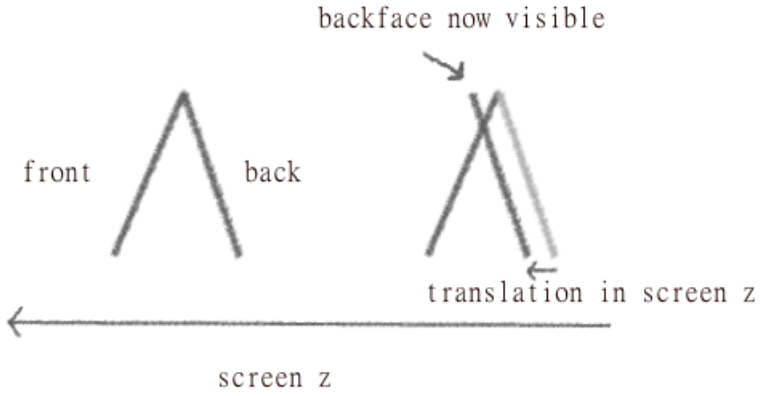

Figure 2.1: Translating the back face forward in screen z

To make wider lines, one could render the back-faces themselves in black, then translating them forward in screen Z. Depending on the translation one could then decide how wide the line will be by choosing how much of the back-facing triangle to make visible. The translation method could, for example be by a fixed amount or by an amount which considers the non-linear nature o z-depths. The greatest weakness is, that the backfaces will not create lines with uniform width because of that the translation not only depend on the back-face but on the neighbouring front-face(s) as well.

This weakness was solved by (Raskar and Cohen, 1999). Instead of translating the back-faces forward in screen Z they extend the back-face out along its edges by the amount needed to see a consistent line, this method is called triangle fattening and the fattening is determined by the slope of the triangle and the distance from the viewer. However, for thin triangles it will not be able to create lines with uniform width as one corner becomes longer but it could be solved by joining the edges to form mitered corners. Both the fattening and translation techniques could be processed independently for each polygon which makes them good for hardware implementation.

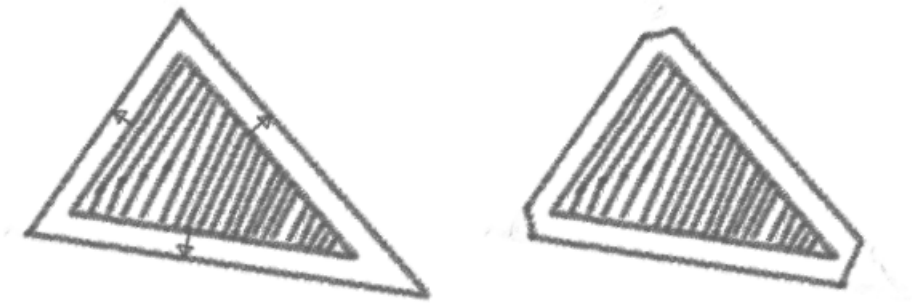

Figure 2.2: To the left, a back face is extended along its plane. To the right the edges has been joined to form mitered corners

However both the fattening and bias translation techniques gives little control over the silhouette appearance (Akenine-Möller et al., 2008).

2.1.2 Image space approach

A basic approach in image space is to find and draw contours from a 3D scene to an image and apply an edge detection algorithm to it. These edges however, might not be fitting for outlining a 3D model. As it will detect a lot of edges irrelevant to represent the shape of the model, for example a textured area will generate a lot of edges and two overlapping objects of the same color will not be detected (Isenberg et al., 2003). A better method for outlining would be to apply the edge detection on extracted data from a depth and normal buffer that is written to a frame-buffer. In a depth buffer the variation of depth between adjacent pixels is usually small over a model but large between models. Applying an edge detection algorithm on the depth map will most likely find outlines separating objects from each other and interior depth differences.

In a normal buffer one could detect outlines between models at the same depth and also creases in the surface. The outlines found from an edge detector on the normal buffer will be changes in surface orientation of a model thus, finding creases and for overlapping models one will have opposing normals resulting in an outline (Isenberg et al., 2003).

By taking advantage of both the depth and normal buffer and add the outlines found from them respectively one will get an outline for a 3D model and as the contours could be stored in a framebuffer, one is easily able to control their appearance by post processing effects.

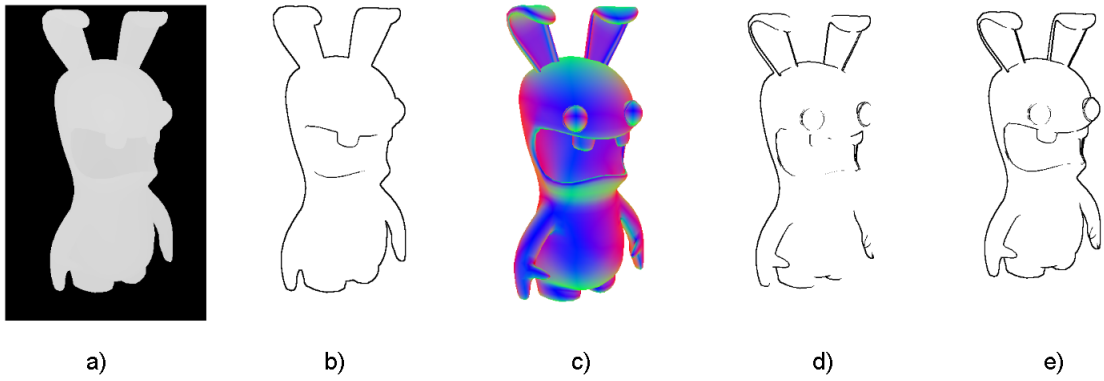

Figure 2.3: a: The depth buffer, b: Edges found from depth buffer, c: The world space normals, d: Edges found from the world space normals, e: The edges from both buffers combined (Model courtesy of Turbosquid)

This method was selected to gain the ability of controlling the appearance but also because the normal and depth information is used in other steps of the algorithm (screen-space ambient occlusion). The loss of control by not using a procedural geometry method was considered negligible.

2.2 Interior Shading

To shade a model the system uses a combination of texture mapping, rotations and standard lightning computations to visualize geometry in a sketched fashion. The system uses a set of fixed texture which is drawn images of graphite strokes by different hardness and brightness. The lightning computation determines the tone to be used for each region in the model and the textures are mapped and rotated in screen-space onto the model. This chapter will describe the specification needed on a texture, how its is built and mapped into screen space.

2.2.1 Pencil Texture

The texture that will represent a pencil needs to express brightness, hardness and thickness of a graphite pencil. It needs a specific direction so it can be used as a reference when rotating the texture along a computed line direction. Finally it also needs to be seamless for continuity when mapping the texture into screen-space.

To express brightness variations a suitable approach is the Tonal Art Map(TAM) structure proposed by (Praun et al., 2001) which simply put is a texture array, where the index decides the brightness by increasing amounts of strokes. Each level should contain the strokes of a lower level to keep the continuity when drawing adjacent textures as well as to avoid aliasing artefacts in figure 2.4 this is visualized.

Figure 2.4: The tam structure

2.2.2 Texture generation

The desired texture array could be created by using a texture generation technique, by deciding upon a stroke that resemble a graphite pencil and uniformly distribute them across a region. To create the different layers with similar orientations, strokes are drawn along a specific direction with some angle perturbation. The brightness levels are created with perturbation on top of a lower level, by drawing more uniformly distributed oriented strokes on top of a lower level to create a new layer, pixels will be darkened by overlapped strokes.

However, this does not capture the dynamics of overlapping graphite strokes in the real world as the increase of darkness is lowered when several strokes are overlapped. A graphite pencil does not tend to be black, but stays as dark grey when repeating strokes on top of each other. In the paper (Lee et al., 2006) a texture generation technique as the previously described is used which counter this just mentioned problem by setting a maximum amount of darkness that can be increased by a stroke. The color in their texture generation technique is determined by,

$$c_t' = c_t - \mu_b c_a, \;\; c_a = c_t(1.0 - c_s), \tag{2.1}$$

where c_t and c_t' is the current and updated texture colors. c_s is the stroke color, c_a is the amount of darkness a stroke is allowed to increase and μ_b is a parameter to determine how much darkness to add to the current texture. Another observation they found when looking at graphite strokes is that even though strokes are added on a paper many times over regions not drawn with black lead at the first stroking tend to remain unstained. To achieve this property they also use a limit on 2.1,

$$\text{If} c_t \text{ is white enough}, c_a = \mu_w c_a \tag{2.2}$$

which will preserve white pixels by selecting the parameter μ_w.

2.2.3 Texture mapping

Instead of setting up a texture generation, a hand-drawn texture of strait lines in a TAM structure was created and used.

To express geometry, one could draw strokes which follow the principal curvature direction which is done in the papers (Hertzmann and Zorin, 2000) and (Lee et al., 2006). The principal curvature direction is a measure of how the surface bends with different magnitudes in different directions and can roughly be described as two orthogonal vectors, in the tangent plane of a point in a surface. Where one represent the maximum

curvature direction and the other the minimum curvature direction. The computation of the principal curvature direction will be described in the next two chapters.

With the principal curvatures computed we want to use them to get screen aligned lines which follow them. If the principal curvature direction is the same over a surface we want the whole surface to have lines in that direction. We also want continuity between triangles, which means that we want the same direction in a vertex, whichever triangle the vertex is in and we want the direction to be the same over edges while inside of the triangle we want the directions interpolated.

This is achieved by using the hand-drawn texture with three different rotations, one per vertex. For each vertex v of a triangle, we calculate a rotation that matches the 2D projection of the principal curvature direction of v. This gives us three texture rotations per triangle, for each fragment the interpolated barycentric coord has been used as a weight to select which of these textures to use.

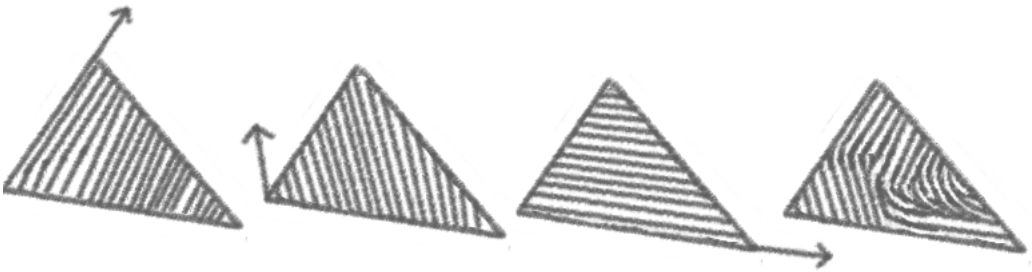

Figure 2.5: The textures interpolated by the barycentric coords.

The interpolated fragment value will be the same over an edge and for each vertex the fragment will come from a texture rotated by the rotation from the principal curvature direction belonging to that vertex. Even though the texture rotations are performed independently per face, there will still be continuity over adjacent faces because they share an edge, by that they will have two directions in common (leads to the same rotation for two of three vertices).

2.2.4 Implementation

The texture mapping has been implemented using graphics hardware in a vertex, geometry and fragment shader.

In the vertex shader we create a 2D projection of the principal curvature direction which will be passed to the geometry shader.

The three passed 2D projected principal curvatures are used to create three rotated texture coordinates for each triangle where the rotation is the angle between the direction of strokes in the texture and the projected principal curvatures.

With the texture coordinates we are now able to interpolate a fragment value in-between three texture lookups(one for each texture coord) by using the barycentric coord of each fragment.

2.3 Curvature

As mentioned in section 2.2.3 it is desired to rotate texture coordinates along lines of curvature. To do this it is first needed to extract principal curvatures for each vertex of a mesh. This section will describe the algorithm from the paper (Alliez et al., 2003) on how to extract a curvature tensor field from an input surface, and then my improvement.

2.3.1 Principal curvature direction

In order to have a continuous tensor field over the whole surface, a piecewise linear curvature tensor field is built by estimating the curvature tensor for each vertex in the mesh and interpolating the values linearly across faces. This calculation is not very natural for a vertex, however. For every edge e of the mesh there is an obvious minimum and maximum curvature(along and across the edge). Considering this a more natural curvature tensor can be defined at each point along an edge, which is described in (Cohen-Steiner and Morvan, 2003) who introduces a a measure of a discrete mean curvature of a surface.

To find an averaged line density of the tensors a sphere is placed centred at each vertex, the sphere is used to find intersecting edges which will be added as a contribution when all intersecting edges has been gathered they will be integrated under the sphere. different edge contributions and to integrate them under the sphere,

$$\tau\left(v\right) = \frac{1}{|B\mid} \sum_{edges\, e} \beta\left(e\right)\ |e \cap B\ |\overline{e}\ \overline{e}\ ^{t}, \tag{2.3}$$

where v is an vertex on the surface, $|B\mid$ is the surface area of the sphere, $\beta\left(e\right)$ is the signed angle between the normals to the two oriented triangles incident to edge e (positive if convex, negative if concave). $|e \cap B\mid$ is the length of the intersection of an edge e and the sphere. $\overline{e}\ \overline{e}\ ^{t}$ is a matrix of the unit vector in the same direction as e multiplied with the transposed direction of e

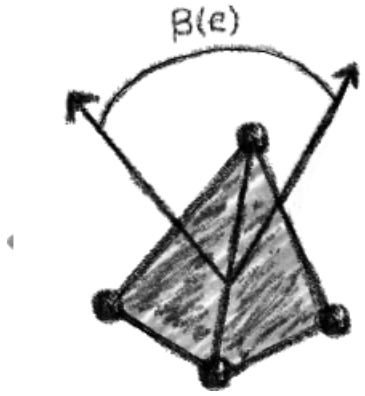

Figure 2.6: The signed angle $\beta(e)$ which tells if the edge is positive or concave

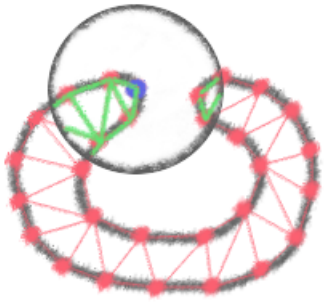

Figure 2.7: Marked edges using a sphere

As seen in 2.7 this method of only selecting edges intersecting the sphere could, depending on the sphere size, receive input from edges which describes curvature of a different surface then the one we want to compute. This could be solved by further weighting the edge contributions depending on which feature of the surface an edge belong to (Alliez et al., 2003). The fixed size of a sphere will also lead to variations in the number of edge contributions on a mesh which do not have uniformly distributed edges over the whole mesh.

2.3.2 Edge traversal

To counter the previously mentioned problems in 2.3.1 another approach to compute the principal curvatures was developed. The new method takes the same approach of computing the various edge direction contributions to each vertex to remain intact with the mathematical foundation. With a new heuristic on how to collect the edges, instead of using intersecting edges inside a sphere, the edges are found by stepping through edges to neighbouring vertices,

$$\tau\left(v\right) = \frac{1}{\#edges} \sum_{edges\, e \epsilon S} \beta\left(e\right) \left|e\right| \overline{e}\ \overline{e}^{\,t}, \tag{2.4}$$

where S is the set of edges which are connected to a vertex by a selected amount of steps.

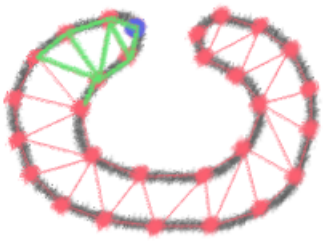

Figure 2.8: Marked edges using the stepping technique

The goal of this heuristic was to handle non-uniform meshes better than the previous approach with a fixed size sphere. When using a fixed size of steps instead every vertex will only collect edges connected to the vertex we want to compute the curvature direction for and we will also get the same amount of edges as it has neighbours leading to that complex and non complex areas would roughly get the same amount of information from their connected edges.

2.3.3 Smoothing undefined regions

A problem with using principal curvature directions, which both section 2.3.1 and 2.3.2 suffers from is that for umbilical points (points with coinciding principal curvatures) and flat areas the direction field is not well defined. They are not well defined due to that all umbilical points are singularities which means that the they are not defined anywhere on a sphere and because of that will receive arbitrarily principal curvatures and on flat areas both principal curvatures will be very small and will likely result in a very complex direction field which would not be well suited to draw lines after, though to that an artist most likely would draw a flat region with a simple pattern (Hertzmann and Zorin, 2000).

One way to counter this problem is to smooth the result between well defined curvatures and not well defined ones. To decide if a vertex is well defined or not, one could look on the principal curvatures k1 and k2 and their relation to each other due to that for each point on a surface the point will belong to one of the following categories.

Finding out which category a vertex belonged was however, found to be very hard since the equation 2.3 and 2.4 are both approximations of k1 and k2. Another method was then devised to give a measurement on how well defined the principal curvature is for a given vertex, this method inspects $\beta(e)$ (the signed angle between the normals to the oriented triangles incident to edge e). $\beta(e)$ could be used to find if a vertex resides in a plane due to that when the angle $\beta(e)$ is zero, the edge lies in a plane. By finding the average of the angle over all edges contributed we will have an approximated measurement on how well defined the curvature is. Due to that for a small averaged value the vertex is most likely in a plane and for a large value the vertex is most probably not. This value is then used to scale the smoothing in-between two vertices.

2.3.4 Implementation

There are two implementations on the computation of principal curvatures based on the two methods 2.3.1 and 2.3.2 to be able to compare them against each other. They follow the equations given to produce an array containing the principal curvature directions. Which are passed as an vertex array to the graphics hardware for further processing, the processing is described in 2.2.3.

2.4 Image improvements

The outlines and interior shading alone creates a graphite sketch however by using post processing the visual result could be enhanced even further. This section presents a few possible improvements that has been used in the system.

2.4.1 Brightness adjustment

To achieve perception of brightness and shadows as in the real world, an artist often use contrast enhancement techniques this to overcome the limited medium of a graphite pencil. For example specular highlights could be left as sharp white regions in contrast to darker sides, to give the same perception as a specular lit region would do. In NPR the same behaviour could be achieved by using a contrast enhancement operator, which often works on a gray scale texture or gray shaded model. The operator is a function that maps a gray value x to another gray value y (Majumder and Gopi, 2002). For example if a square root function is used, bright regions get enforced intensity variation. In the paper (Lee et al., 2006) such an operator is used.

In this paper the square root is used on the color received from the 2.2.3 to get a brightness adjusted value.

2.4.2 Paper effect

Drawing a pencil sketch is most often performed on a sheet of paper; the paper comes with various material properties, such as textures and roughness. When drawing on a paper, these properties affect the outcome in such a way that graphite has various falloffs depending on where a line is drawn. To mimic this behaviour the system consists of a paper implementation proposed by and it has also been used by (Lee et al., 2006). The method uses a texture that resembles a paper and a matching height map for the texture. The effect is considering the interaction among the pencil and a paper in a single stroke. If the stroke direction is opposite of the paper normal, the stroke is darkened. This will give the impression that the drawing has been made on a rough, textured surface such as a paper is.

Figure 2.9: A texture of a rough paper

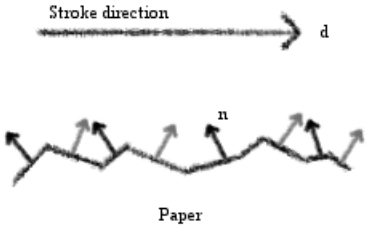

Figure 2.10: A stroke hitting the paper

The 2.9 is the texture used in this paper which is used as a background for the final sketch and the strokes has been adjusted by using a normal map created from the paper texture.

2.4.3 Screen space ambient occlusion

Ambient occlusion is an approximated amount by which a point is occluded by surrounding geometry. Techniques for ambient occlusion allow a simulation of proximity shadows and produce a subtle effect in computer generated scene, but with a dramatic improvement of visual realism. The effect is not limited to only produce visual realism and in NPR it could be used to find and control occluded areas to further express them in a desired fashion. The ambient occlusion is often expressed as an occlusion factor for each point where more occlusion means less light and less occlusion means more light. A good approximation of an occlusion factor is often time consuming and performed with casting a large number of rays, which is not practical for real-time rendering. In 2007 Crytek implemented a real-time solution for their game Crysis, which is a screen space approach and the whole process can be done on the GPU and is fully dynamic and independent of scene complexity. (John, 2013)

The method Crytek uses is to sample the depth buffer at points found from samples in a sphere; they project samples into screen space to get coordinates into the depth buffer. Then they sample the depth buffer and check if the sample position is behind the sample depth (i.e inside geometry), which will then be added as a contribution to the occlusion factor. This leads to a quality proportional to the number of samples and as the sample kernel is a sphere, flat regions end up grey as roughly half of the samples will be inside the wall and convex corners appear lighter as fewer samples fall inside geometry. These artefacts are visually acceptable and the technique manages to produce an effect similar to photorealism. (John, 2013)

Another, yet similar approach and the one used is to use a normal-oriented hemisphere instead of a sphere, which means a hemisphere oriented along the surface normal at a pixel is used for the sampling of the depth buffer. This somewhat avoids the problems

of convex corners and flat surfaces but requires per-fragment normal data. (Bavoil Louis, 2008)

Figure 2.11: Sampling using a sphere

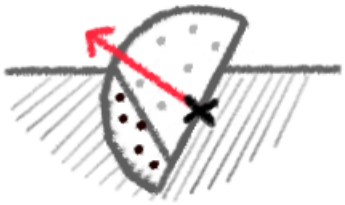

Figure 2.12: Sampling with a normal-oriented hemisphere

2.5 System overview

The rendering system created by using these methods can be described as two separate processes, one for the contour drawing and one for the interior shading. These two processes are both rendered to separate images, the images is then merged together in a post-processing pass. The post-processing pass, besides merging the images also performs the image adjustments described.

The contour drawing process, gathers normal and depth data of a input mesh and store the information in a texture. This texture is then used to detect and extract contours in image space. These contours are then enhanced by drawing them on top of each other slightly distorted, with different tones.

The interior shading process starts by determining the brightness of pixels which is used to select which pencil stroke texture to use. To express geometry these textures are rotated and mapped in the principal curvature directions. The principal curvature directions has been computed in a pre-processing pass which has implemented the method 2.3.2. The pencil strokes are then drawn on a paper with the paper effect 2.4.2.

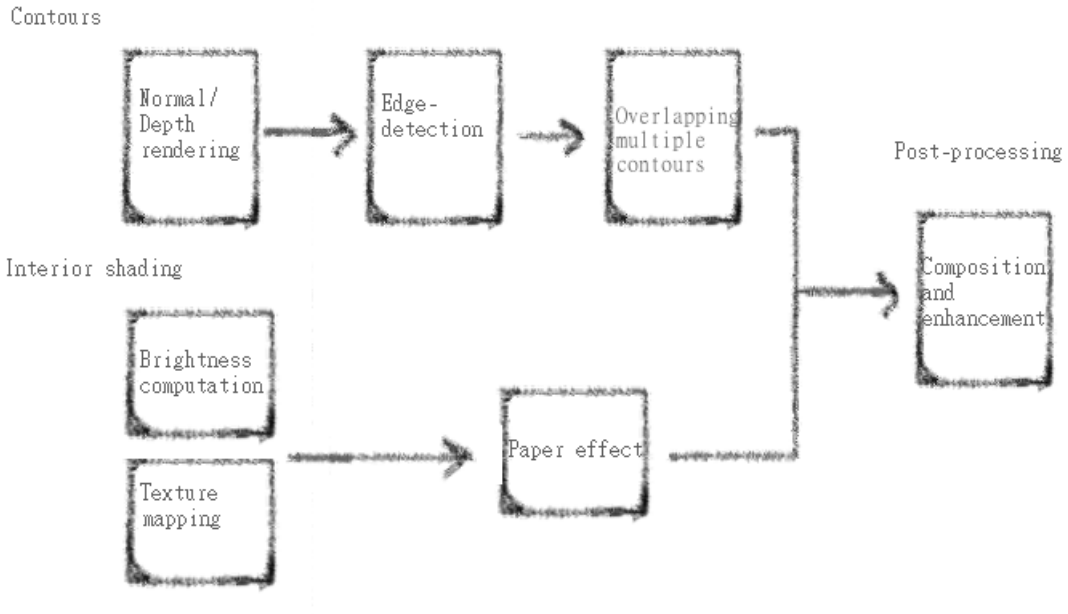

Figure 2.13: A visual representation of the system

3

Results

The system is implemented in c++, OpenGL and the OpenGL Shading language, the results has been obtained running the system on a Windows PC, with a Intel(R) Core(^{TM})2 CPU 2.6GHz , 4GB memory and a GeForce 8800 GTX graphics card with 2GB video memory. Rendering of the models has been in 60FPS the time consuming part of the system is the preprocessing step where all vertices are processed to compute its principal curvature direction. Which is done in about 30 seconds for 6000 vertices.

3.1 Performance

Figure	Model	nr of vertices	Avg. FPS
3.5	Smooth mesh	2178	60
3.12	Sink	5287	60
3.13	Rabbit	5699	60

3.2 Contours

The depth buffer and normal data is rendered to a framebuffer as mentioned in 2.1.2 of the same resolution as the rest of the system to be able to merge the contours with the other images created by the system.

On the data a Sobel kernel is used for edge detection which finds differences in intensity of the normal and depth data. Detected fragments considered a silhouette are discarded, leaving a black pixel. This results in an outline which looks like it has been drawn with india ink which can be seen in 3.1.

To get a line resembling being drawn with graphite the detected fragments are no longer just discarded but instead drawn multiple times by blending together several textures containing the contours with different grayscale values to get a blended look similar to graphite. When an artist draw lines he/she often draw interleaving shorter lines to produce a long line, to achieve the same effect the texture coordinates used when merging the contours are transformed by different sinus waves. This can be seen in 3.2.

Figure 3.1: Discarded fragments marked by the edge detection

Figure 3.2: Contours rendered by blending different grayscale contours with transformed texture coordinates

3.3 Interior shading

Shading the interior is done by mapping a texture following the TAM structure visualized in 2.4. The desired tone is found by computing a diffuse component following,

$$I_{diff} = (\mathbf{n} \cdot \mathbf{l}) \tag{3.1}$$

This will give us a value between 0 and 1 which says describes no light or fully lit. By multiplying this value with the amount of textures stored in the TAM structure we will get an index on which texture to use. This means that fully lit areas will use the white texture and an area with no light will be filled with strokes. The result of simply mapping these textures in screen space can be seen in 3.3.

If we add the image adjustments described in 2.4. The fragment color is modified both with a contrast operator and the paper roughness. We will get a better overall tone and we can see strokes hitting the paper with different falloffs, which can be seen on the side of the sink in 3.4.

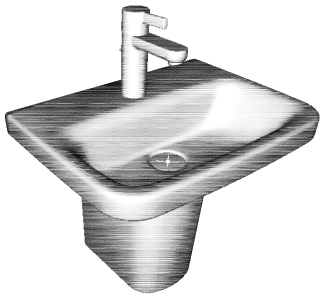

Figure 3.3: A sink shaded without image adjustments contours or curvature

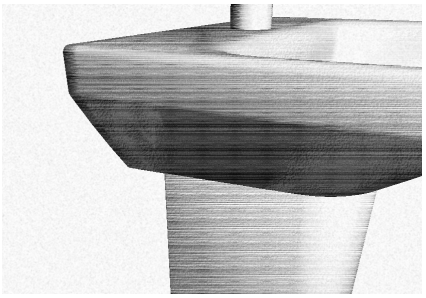

Figure 3.4: A sink shaded with image adjustments

3.4 Curvature

By computing the principal curvature using the equation 2.3 on a smooth mesh with uniformly distributed edges we get a good result which draw the mesh in a way which could be expected from an artist which can be seen in 3.5. However, if we apply the same equation on a more complex mesh we will get a bad result, see 3.6. We can see that inside the sink, the result is good but overall the result is poor. The difference is that inside the sink, this models edges are in that region uniformly distributed but for the larger areas as its foot and the top the model has very few edges, the edge distributions can be see in 3.10.

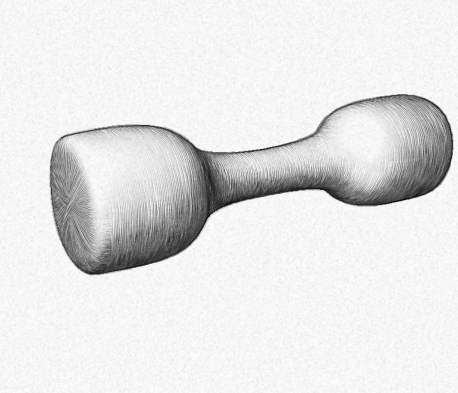

Figure 3.5: A smooth mesh with edges uniformly distributed

Figure 3.6: Using the sphere technique

As described in 2.3.2 a new method was devised to counter this problem, the result from the new algorithm using one and two steps can be seen in 3.7 and 3.8.

Figure 3.7: Using the edge traversal algorithm with one step

Figure 3.8: Using the edge traversal algorithm with two steps

Comparing the two different algorithms 3.6 and 3.8 their result is really similar at

at first glance, they both have bad continuity over flat surfaces. However, if one look at the front of the sink, following the sink edge a much better curvature is found in 3.8.

To solve the bad continuity on flat surfaces the smoothing described in 2.3.3 is performed.

In 3.9 we can see that the first tested heuristic gives good curvature for the inside of the sink which can be expected as that area contains a lot of edges which are close to uniformly distributed, see the distribution in 3.10. But if we look at the foot and top there are larger and more varied edges which results in an unexpected curvature.

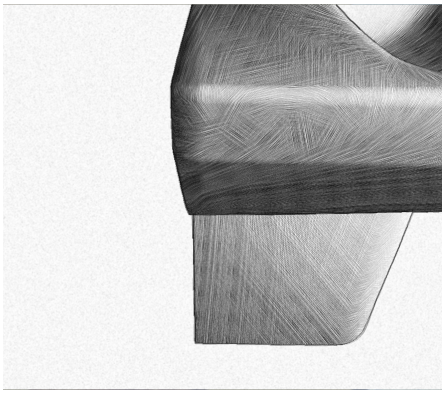

Figure 3.9: The sphere technique smoothed

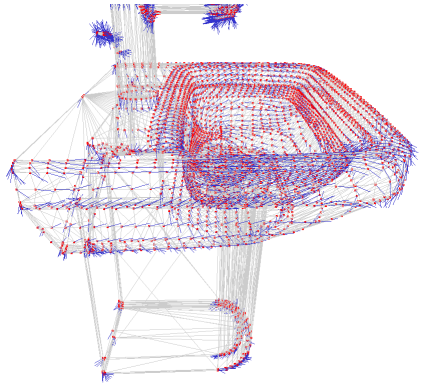

Figure 3.10: The edges are in grey, the vertices a red dot and the computed curvature direction is the blue line

In 3.11 we can see that the new proposed heuristic is better for large polygons (the sink foot consists of several big quads) and an overall good curvature is visualized.

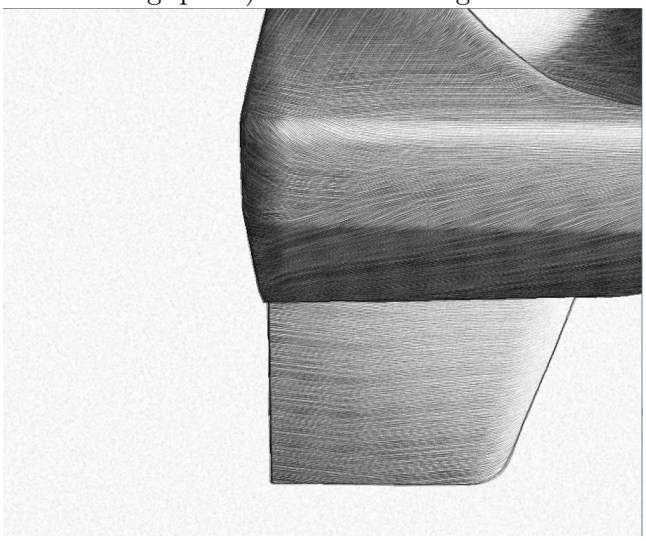

Figure 3.11: The edge traversal algorithm with two steps and smoothed

3.5 Final results

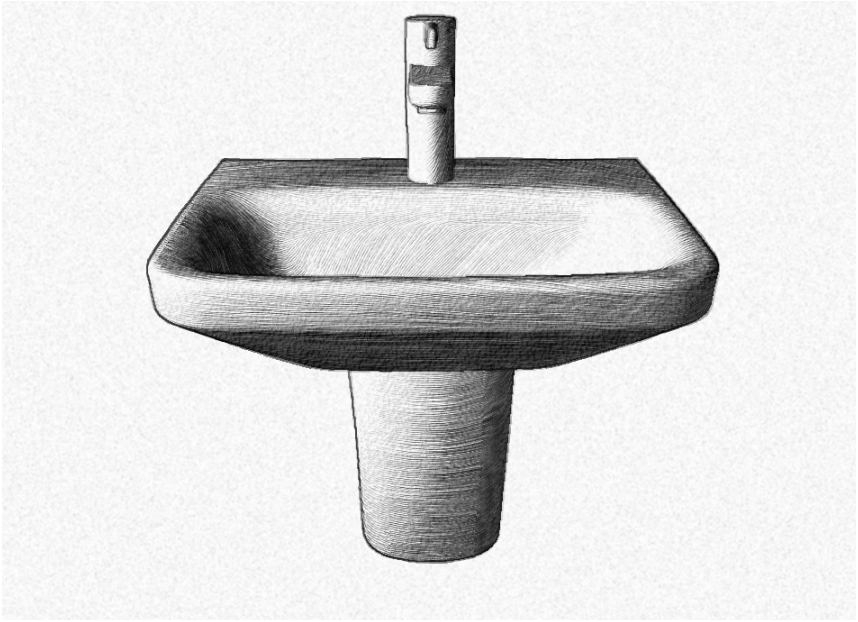

Figure 3.12: A sink shaded using the whole system

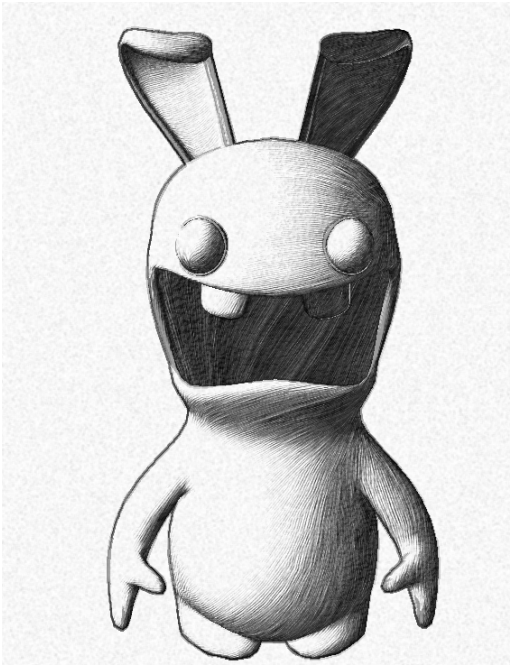

Figure 3.13: A rabbit shaded using the whole system (Courtesy of TurbuSquid)

4

Limitations and Future Work

When moving the camera, or by using an animated surface in the scene, a viewer may witness a distortion of the textures and a lack of temporal coherence which is due to the rotation and mapping of a texture in screen space. For example, when looking at strokes going strait from left to right and the camera is tilted the textures will still go from left to right.

The algorithm for computing the direction fields tends to create a vortex if two regions with opposing directions are smoothed over, the parametrization and number of iterations heavily affects the result and there is no general setting which produces a good result for all kinds of meshes.

The image space approach of finding contours works best on certain depths and if the camera is to close or far away it could fail to identify all of the expected and desired contours.

An important area for future work would be to improve the look of the contour drawing by using the same stroke texture as the interior drawing uses but with forced darkness. The result could perhaps also be improved by finding more contours both outlining the geometry but also interior regions.

There are more characteristics of pencil drawing the algorithm could benefit from, as velocity and pressure tampering (an artist rarely draws a stroke in constant velocity or pressure) Some more straightforward improvements could be to add color pencils with the material property of a mesh and eraser effects.

5

Discussion

In the introduction three problems were stated to be solved to be able to produce a graphite sketch, draw an outline, draw the interior and produce a good visual result.

The outline produced by the system is a subtle effect but it gives an overall better look on the results. One problem with the outline using the image space method is that it is hard to give precise control on where you want an outline. The Sobel edge detection finds the outlines, but depending on the distance from the the object to the viewer more or less outlines could be detected. However if we used a fixed camera position, one is able to control the parameters on what the edge detection should consider as an edge and with that produce good outlines.

The interior shading, thanks to the principle curvature direction manages to visualize the geometry as desired. However, the texture used is not of the best quality and one can see some repeating patterns which is due to that the lines in the texture is not perfectly distributed, with a better texture the overall look would be improved. Another problem is that for the complex models with many ”undefined” regions the system will use a lot of smoothing which tends to result in lines creating a vortex.

The image improvements performed, especially using a paper are all subtle effects but they do improve the overall result.

Overall during this project, most of the focus has been on comparing the two equations for computing a curvature direction field 2.3 and 2.4. The new equation was found to be better suited for certain models, especially non uniformed tessellated models but compared to the original method it performed poorly on uniformed tessellated models. I believe however, that with a better parameter setting and tedious testing the algorithm should be on par with the first inspected algorithm.

6

Conclusions

The system manages to produce graphite sketches of good quality, the interior shading catches the geometry of the model due to the principal curvature directions. The silhouettes found by the image space approach and the screen space mapping manages to create a sketch which looks to be drawn on a paper.

The edge traversal in Section 2.3.2 to compute the principal curvature direction, creates better continuity over large polygons compared the sphere technique according to my results. However, the sphere technique tends to work better on meshes with uniformly distributed edges.

Bibliography

T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-time rendering [Elektronisk resurs]*. A.K. Peters, Wellesley, Mass., 3rd ed. edition, 2008.

P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 485–493, New York, NY, USA, 2003. ACM. ISBN 1-58113-709-5. doi: 10.1145/1201775.882296. URL http://doi.acm.org/10.1145/1201775.882296.

S. M. Bavoil Louis. Screen space ambient occlusion@ONLINE, Sept. 2008. URL http://developer.download.nvidia.com/SDK/10.5/direct3d/Source/ScreenSpaceAO/doc/ScreenSpaceAO.pdf.

D. Cohen-Steiner and J.-M. Morvan. Restricted delaunay triangulations and normal cycle. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 312–321, New York, NY, USA, 2003. ACM. ISBN 1-58113-663-3. doi: 10.1145/777792.777839. URL http://doi.acm.org/10.1145/777792.777839.

B. Gooch and A. Gooch. *Non-photorealistic rendering*. A K Peters, Natick, Mass, 2001. ISBN 1568811330; 9781568811338. URL www.summon.com.

A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 517–526, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi: 10.1145/344779.345074. URL http://dx.doi.org/10.1145/344779.345074.

T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Comput. Graph. Appl.*, 23(4):28–37, July 2003. ISSN 0272-1716. doi: 10.1109/MCG.2003.1210862. URL http://dx.doi.org/10.1109/MCG.2003.1210862.

C. John. Ssao tutorial@ONLINE, Jan. 2013. URL http://john-chapman-graphics.blogspot.co.uk/2013/01/ssao-tutorial.html.

H. Lee, S. Kwon, and S. Lee. Real-time pencil rendering. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pages 37–45, New York, NY, USA, 2006. ACM. ISBN 1-59593-357-3. doi: 10.1145/1124728.1124735. URL `http://doi.acm.org/10.1145/1124728.1124735`.

A. Majumder and M. Gopi. Hardware accelerated real time charcoal rendering. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pages 59–66, New York, NY, USA, 2002. ACM. ISBN 1-58113-494-0. doi: 10.1145/508530.508541. URL `http://doi.acm.org/10.1145/508530.508541`.

E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 581–, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383328. URL `http://doi.acm.org/10.1145/383259.383328`.

R. Raskar and M. Cohen. Image precision silhouette edges. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, I3D '99, pages 135–140, New York, NY, USA, 1999. ACM. ISBN 1-58113-082-1. doi: 10.1145/300523.300539. URL `http://doi.acm.org/10.1145/300523.300539`.

J. R. Rossignac and M. van Emmerik. Hidden contours on a frame-buffer. In *Proceedings of the Seventh Eurographics Conference on Graphics Hardware*, EGGH'92, pages 188–203, Aire-la-Ville, Switzerland, Switzerland, 1992. Eurographics Association. doi: 10.2312/EGGH/EGGH92/188-203. URL `http://dx.doi.org/10.2312/EGGH/EGGH92/188-203`.

M. Webb, E. Praun, A. Finkelstein, and H. Hoppe. Fine tone control in hardware hatching. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pages 53–ff, New York, NY, USA, 2002. ACM. ISBN 1-58113-494-0. doi: 10.1145/508530.508540. URL `http://doi.acm.org/10.1145/508530.508540`.