EFFICIENT STIPPLE RENDERING

Jens Krüger and Rüdiger Westermann
tum.3D, Technische Universität München
Boltzmannstraße 3, 85748 Garching bei München
[jens.krueger,westermann]@in.tum.de

ABSTRACT

In many computer graphics applications, like stylized rendering, technical illustrations, digital printing, or volume rendering it is required that "continuous tone" models be represented by drawing dots that are either black or white at any given surface point. The arrangement of dots must be chosen so that the illusion of a continuous shade is preserved. In this paper, we propose a method to generate such effects on arbitrary surface and volume structures. The black and white patterns are carved out of a solid texture, in which the spatial arrangement of dots is encoded for arbitrary tones. These patterns are applied to the respective structures using hardware accelerated 3D textures. As a matter of fact, neither does our approach rely on any surface parameterization nor do point primitives have to be used to render the patterns. In addition, by introducing a non-linear gamma-function, charcoal or pen and ink like effects can be achieved. For stylized rendering, we further enhance the artistic impression of silhouettes by generating outlines.

KEYWORDS

Non-Photorealistic Rendering, Graphics hardware, Volume Graphics.

1. INTRODUCTION

In this paper, the emphasis is on imitating artistic drawing methods that are widely used in non-photorealistic rendering of static imagery – *Point Stippling* in combination with *Silhouette Rendering*. Both methods are assisted by graphics hardware, thus enabling interactive rendering of complex models and scenes. The basic rendering styles we are able to imitate by means of the proposed methods are illustrated in Figure 1. Many articles have extolled the virtues of non-photorealistic techniques for the rendering of polygonal and volumetric models. Besides its use in computer arts and entertainment, non-photorealistic rendering (NPR) techniques have been employed frequently in computer-generated publishing of medical or technical illustrations and sketches (Gooch and Gooch 2001). Many of these techniques exploit the fact that shading constitutes an important visual cue for deriving spatial information from a 2D image, as long as the relation between shadowed and illuminated areas is maintained. Based on this observation, fine artists and illustrators have developed a wide range of techniques for depicting shading in a stylized way. These techniques, among others, include stippling, hatching and painting with a limited palette of tones (see Figure 2). Apart from esthetic aspects, such techniques were often imposed by limitations of the available print media, computer assisted techniques predominantly tried to mimic traditional artistic methods and styles.

Stipple drawings are made up of points that combine to give the illusion of a "continuous tone" model. In classical halftoning, dots are either black or white, whereas more stylized impressions can be achieved by allowing points to accept shades of grey. The artist can control the size, the shape and the arrangement of points to generate additional visual cues that provide an effective means for enriching or even substituting traditional representations. The arrangement of dots must be chosen so that the illusion of a continuous shade is preserved (Hodges 1989).

a) b) c)

Figure 1: *a) Charcoal-like effects b) & c) Stipple Volume rendering with Silhouette Rendering*

The contribution of this work to the field of computer generated point stippling is twofold: Firstly, we present a simple, yet powerful and efficient method for generating stipple drawings of surface and volume structures. Therefore, stipple patterns are encoded in one single 3D solid texture by a stationary white noise distribution. At each object point, the pattern is carved out of this texture by sampling the respective value in a fragment shader program. Based on the comparison between the point's continuous tone and the grey value of the encoded texture sample, the fragment color is set to either black or white. Secondly, we enrich traditional stippling techniques to allow for the simulation of different drawing styles. We introduce a non-linear gamma function to be applied to the diffuse lighting term in the shader program. By means of this technique, artistic styles like charcoal-like effects can be simulated.

Our choice of technique was mainly driven by the following requirements. Our method should be able to render both surface and volumetric structures. For surfaces, it should be independent of any parameterization thus avoiding distortions in case of non-regularly spaced texture samples. The technique should not rely on point primitives thus reducing memory and bandwidth requirements. Finally, it should be applicable to large-scale geometric and volumetric models and scenes, yet suitable for real-time applications. We therefore decided to exploit the capabilities of modern graphics cards providing hardware supported vertex and fragment shader. Our approach relies on the functionality that is provided by any of today's commodity GPUs - the Pixel Shader 2.0 API.

a) b) c) d)

Figure 2: *Examples of artistic shading. a) Stippled Illustration of the Stanford bunny b) Baldung Grien,* The Fall of Man, *Detail, 1511, Woodcut c) Henry Matisse,* Figure décorative sur fond ornemental, *Detail, 1925-26, Oil on canvas, d) Cartoon-like toon shading*

2. PREVIOUS WORK

Imitating artistic stipple drawings by means of a computer can be performed in a variety of different ways. In particular, approaches differ in the way stippling patterns are generated, and in the primitives that are used to render these patterns. Deussen et al. (2000) and Secord (2002) employed relaxation techniques in screen space and in object space using Voronoi diagrams, respectively. Aiming at high-quality and even distributions of stipples, both techniques are not suited for interactive application in general. The benefits of

point primitives for the rendering of stipple dots have been demonstrated by Pastor and Strothotte (2002) and Lu et al. (2002). Whereas the former approach addresses the problem of how to generate coherent patterns that do not introduce flickering in animations, the latter one demonstrates the effectiveness of stippling techniques for enhancing or even substituting traditional volume renderings. 2D stippling textures have been introduced by Praun et al. (2001). Based on a consistent surface parameterization, 2D texture maps comprised of the stipple patterns to be used are mapped onto polygonal objects. Majumder and Gopi (2002) have developed a hardware accelerated charcoal rendering technique that can create a variety of different styles by applying contrast enhancement operators to 2D stipple textures. This is similar to the nonlinear lighting model that we use to gain more control over shading.

Techniques for extracting and rendering silhouettes can be classified into image-space, object-space or hybrid approaches. Saito and Takahashi (1990) introduced the G-Buffer, which stores depth maps of the rendered scene. On these maps, imaging operations are performed to detect silhouettes. Raskar and Cohen (1999) determined image-space silhouettes by clipping slightly transformed back-faces against the depth buffer. In this way, the extraction of geometric outlines can be avoided. In Object-space methods on the other hand silhouette edges are determined geometrically, either by testing every edge of the object or by applying more efficient randomized algorithms as proposed by Markosian (Markosian 2000). Next, visibility testing is performed to determine visible silhouettes. Object-space visibility tests calculate the intersections between silhouette edges analytically (Arthur Appel 1967, Aditi Majumder and M. Gopi 2002). Due to performance issues, the visibility determination might also be performed in image-space. This is accomplished by scan-converting silhouette edges in software and by comparing depth values to those values already in the depth buffer. Finally, as proposed by Isenberg et al. (2002), visible parts can be further processed to eliminate artifacts, and they are rendered using stylized primitives, e.g. lines (Lander 2000) or textured triangle strips (Northrup and Markosian 2000, Isenberg et al. 2002).

3. POINT STIPPLING

In the following, we will demonstrate the stippling of polygonal objects. As will be explained later, for the stippling of volumetric objects the algorithm only has to be slightly modified. We assume an illumination model using achromatic light and material be evaluated on a per-vertex or per fragment basis during rendering. For every fragment a grayscale shade is generated that can be accessed in the fragment shader program.

3.1 3D stipple textures

Figure 4: *The polygonal model is carved out of a 3D solid noise texture*

At the core of our approach we construct a solid noise texture that is comprised of uniformly distributed random values $\in$ (0,1). This texture is mapped onto the object to be stippled. Therefore, object coordinates are properly scaled to the range of (0,1), and at every mesh vertex its world space coordinate is issued as 3D texture coordinate. This coordinate is used to look-up the noise texture. In Figure 4 the basic algorithm is depicted. Note that by scaling the objects' coordinates and by employing the OpenGL extension GL_REPEAT, we can periodically replicate the solid texture to fill an arbitrarily extended region in 3D

space. This allows us to use 3D textures of reasonable resolution, and thus to keep memory requirements as low as possible. In general, periodic structures will only be visible if the resolution falls below a critical size. This will be demonstrated later on in this paper.

The described algorithm does not yet generate stipple drawings. Rather than that, the polygonal model is textured with random noise. In the present scenario, however, we aim at correlating the number of stipple points that are drawn in a certain area with the intensity of the shading in this area. In addition, in order to simulate traditional halftoning, stipples should be either black or white. This is achieved by letting each fragment compare its shade with the sampled noise value. Only if the noise value is larger than the illumination value is the stipple turned on. Otherwise it is turned off and the fragment color is set to white. Therefore, for every fragment the following simple stipple test is performed:

$$c_f = c_t \leq c_{illum} \; ? \; 1 \; : \; 0$$

Here, c_f and c_t denote the final fragment color and the noise value, respectively. c_{illum} is the shading value that has been generated for the fragment. In Figure 5 this process is illustrated.

Figure 5: *The solid texture is used to generate the stipple pattern according to surface illumination.*

The stipple test is performed in a fragment shader program. The program is evaluated for every rendered surface point. As input parameters it gets assigned the illumination value and the respective texture coordinate. Finally, the shader outputs either white or black depending on the reconstructed texture sample.

Due to the uniform distribution of random values in the solid texture, the number of fragments that are drawn black inversely corresponds to the brightness of the respective surface point. In dark regions statistically more texture samples are larger than the illumination values, which results in the appropriate stipple distribution.

3.2 Stipple distribution and shape

In particular the bright surface regions in Figure 5 show a very irregular distribution of stipple dots. In some areas only a small number of points can be found, while in other areas having the same brightness more points are placed. This is a direct implication of the limited resolution of the noise texture, which was only 16^3 in the current example. Independent of the resolution, any noise value is resampled with the same expectation value. However, by using lower resolutions, the variance of this random process is continuously increased. As a matter of fact, point stipples are spaced more and more regularly with increasing texture resolution. This is demonstrated in Figure 6, where noise textures of resolution 8^3, 16^3, and 64^3 were used. Particularly in the highlighted regions one can clearly observe much more regular stipple patterns with increasing resolution.

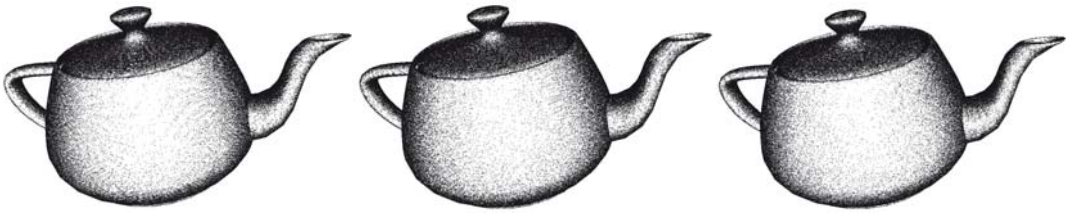

Figure 6: *The solid stippling texture is used at different resolutions.*

Apart from the stipple distribution, the 3D random texture also imposes restrictions on the shape of the stipple dots. In imitating the traditional halftoning process our technique renders dots as either black or white pixels, but visually more pleasant effects can often be achieved if anti-aliased dots are drawn. To generate such effects, anti-aliased point primitives or bi-linearly interpolated 2D stipple textures have been used in most previous examples.

To generate similar effects we might generate the final pixel color by averaging multiple samples from a supersampled image. The middle image of Figure 7 was generated by rendering the scene in full screen anti-aliasing mode, which is supported by most consumer class graphics cards. Due to the averaging process, many of the black dots become grey, thus reducing the contrast in the final image.

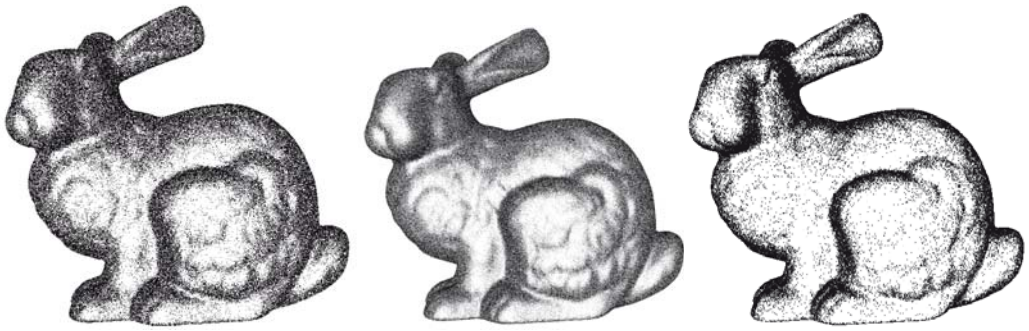

Figure 7: *Different alternatives to generate anti-aliased point stipples are shown. a) Original black-and-white stippling, b) Supersampling in screen space, c) surface based anti-aliasing*

In a second attempt, we can also perform anti-aliasing in object space. Therefore, in addition to the texture coordinate also each surface point normal $\vec{N}$ is specified in the fragment shader program. Then, a local tangent space is computed from the normal vector and an arbitrary vector $\vec{T}$ not coinciding with the normal. Usually, the tangent vector is parallel to the direction of increasing *s* or *t* on a surface parameterized over s and t. Because our approach does not rely on a parameterization, however, we choose $\vec{T}$ from two alternative vectors orthogonal to each other. In this way, we can always select a tangent non-collinear with the normal.

By performing two cross products in the shader program we end up with a local coordinate system that is attached to the surface point. Anti-aliasing is now performed in local tangent space by sampling the 3D noise texture at nearby points in the tangent plane around the current surface point. For each adjacent point we assume the same illumination as we have computed for the current point. This is in coincidence with the assumption that the surface is locally planar. For each neighboring sample the black-or-white stipple test is performed, and the results are averaged to obtain the final fragment color. This can be seen in the rightmost image of Figure 7.

Obviously, because we use a static tangent vector to derive the tangent space coordinate system, the positions of nearby points in the tangent plane vary with varying object orientation. In animations, the effect results in a slightly visible flickering of the generated grey-values.

3.3 Non-linear shading model

In the following, we will introduce a slight modification of the stippling approach as described. We propose a non-linear shading model, which can be used to create a variety of artistic styles ranging from charcoal to pen and ink drawings. In addition, it can be directly applied to other non-photorealistic techniques, for instance computer generated halftoning.

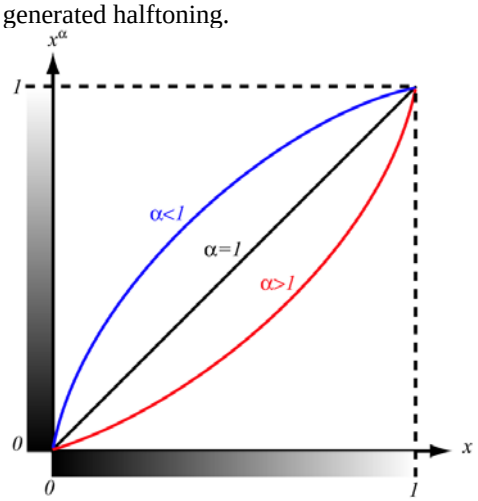

Figure 8: *Examples of gamma functions using different values for α*

Our proposed point stippling algorithm works by comparing at each surface point the intensity to the grey-value of the corresponding element in the 3D noise texture. Based on the result of this thresholding operation, the stipple is turned on or off. This approach, however, gives artists little control over the relative size of the differently patterned regions on the surface. In general, it is not possible to locally change the threshold used to determine the stipple pattern. We therefore modify the linear diffuse lighting equation

$$I = a_l a_m + \max(L \cdot N, 0) d_l d_m$$

where a_l and d_l are the ambient and diffuse light intensities, a_m and a_m represent the ambient and diffuse material intensities, L is the light direction at the surface point and N is the surface normal. We introduce a non-linear gamma function for calculating the diffuse component of I, resulting in

$$I = a_l a_m + \left(\frac{L \cdot N + 1}{2} \right)^\alpha d_l d_m$$

Applying a gamma function (see Figure 8) to the diffuse lighting term causes either the middle tones to be accentuated (for $\alpha < 1$) and the very light and dark tones to be compressed and vice versa for $\alpha > 1$. As a result, the boundary between differently textured regions is shifted. Moreover, the dynamic range compression can be used effectively to achieve charcoal-like effects as we demonstrate in Figure 9.

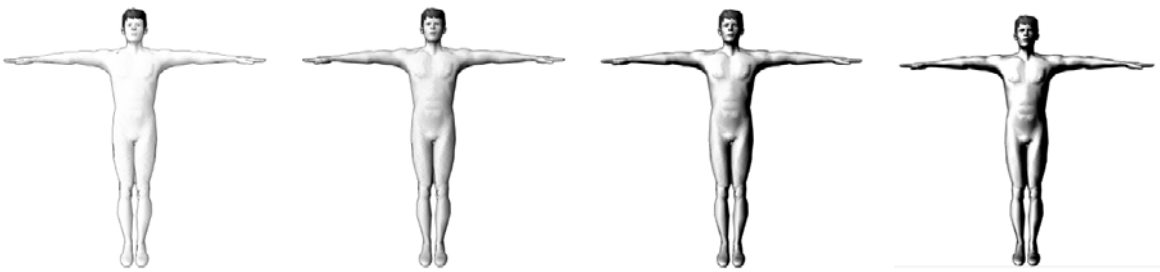

Figure 9: *Examples of drawing styles generated with α values of 1.1, 1.4, 2.0, and 2.7.*

One of the nice features of the described intensity modification is that it can be directly applied to any digital halftoning technique. Typically, an intensity value at each surface point is computed using standard Gouraud shading with achromatic light and materials. This intensity value is then used to determine which halftone pattern is applied to a surface point. Without ever modifying the halftone patterns, or the textures that are used to store these patterns, the illustrated effects can be achieved, by correcting the calculated intensity on a per fragment basis. Additional rendering styles that can be achieved by means of such modifications are depicted in Figure 10.

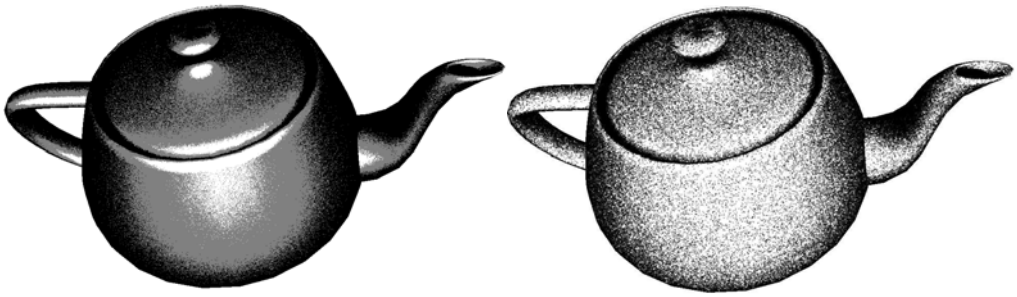

Figure 10: *Examples of different drawing styles. Left: Three tone stippling, Right: Anti-aliased stipples*

3.4 Volume Stippling

The proposed technique can directly be applied to volumetric structures. Because stippling is performed by means of a fragment shader program, it can be attached to any rendering technique as long as object coordinates are accessible in the shader. In 3D texture based volume rendering, for instance, texture coordinates specified at each vertex of the proximity geometry used to render the volume already provide this information. Furthermore, by considering the volume data itself in the stipple test, we can restrict the stippling process to particular scalar values, i.e. to enhance surface like structures.

4. SILHOUETTE RENDERING

For silhouette rendering we apply a method similar to the one proposed by Saito and Takahashi (1990). We generate the final image in two passes. In the first pass we fill three textures. Firstly, the image of the scene as it would look without the silhouettes. Secondly, the normal map of the image. This map can be generated by simply rendering into two render targets at once, and by appending a single line of code to the current fragment shader that writes the normal of every fragment into the second render target. Finally we also capture the depth buffer in a third texture.

In a second render pass we use the normal- and depth- buffer contents and compute first order derivatives on these buffers using central differences for every pixel on the screen. By thresholding these two derivates we can detect both internal and external edges and highlight them in the third texture by reducing the luminance of the respective fragments from the first texture generated in the first pass.

5. CONCLUSION

In this paper, we have emphasized novel approaches for computer generated stippling of surface and volume structures and for the rendering of silhouettes. For point stippling, the major contribution is the initiation of a 3D stipple texture - a new primitive that has several advantages compared to previous approaches. No longer does the stippling process depend on a surface parameterization, and many different rendering styles can be achieved by modulating texture samples on a per-fragment basis. Furthermore, our approach effectively takes advantage of hardware assisted 3D texture mapping, thus enabling interactive

stippling of large-scale models and scenes. Without any modifications, the technique can be used for point stippling volumetric objects. This is a direct implication of using a solid texture for the stippling process.

Our method is easy to implement and can be used as back-end in any rendering technique without that the core implementation needs to be modified. By specifying appropriate inputs to the fragment shader program, various stippling styles can be integrated into arbitrary applications. For instance, arbitrary smooth-step functions can be integrated into the anti-aliasing process by simply increasing the support of the applied low-pass filter. Although a slight overhead in terms of memory requirement is introduced by the 3D noise texture, due to the periodic replication of texture coordinates the initially specified texture can be kept at an acceptable resolution. On recent GPUs – such as the NVIDIA Geforce 8800 GTS – the overhead introduced by the stippling is around 1ms at a resolution of 1600x1200 and thus should only significantly impact applications running above 1000 fps without stippling.

The extraction of silhouettes was performed in screen space by employing a simple two pass rendering scheme, here in the worst case an overhead of about 5ms on current GPUs is introduced at a resolution of 1600x1200. Thus the algorithm will only impact applications running faster than 200 fps. Note that this is the *worst case* as many of today's systems already apply some deferred shading scheme that already extracts the normal- and a depth- map, in that case our silhouettes rendering approach becomes much faster.

We are currently trying to further improve our approaches with respect to the following issues. In animations, the described techniques introduce slight flickering or popping artifacts. In this respect, for point stippling we investigate ways to construct customized 3D mip-maps that allow for distance based anti-aliasing. For stylized silhouettes, on the other hand, popping as introduced might be desirable in artistic character animations. These effects can often be seen in cartoons and free-hand sketches. An elaborate investigation of the advantages of stylized rendering techniques in animation tools is a challenging future research direction.

REFERENCES

Arthur Appel, 1967, *The notion of quantitative invisibility and the machine rendering of solids.* Proc. ACM Natl. Mtg., page 387, Held in Washington, DC.

Oliver Deussen et al., 2000, *Floating Points: A Method for Computing Stipple Drawings*. In Markus Gross and F. R. A. Hopgood, editors, Proceedings of EuroGraphics 2000, volume 19, pages 40 – 51, Oxford, NCC Blackwell Ltd.

Elaine Hodges, 1989, *The Guide Handbook of Scientific Illustration.* Van Nostrand Reinhold, New York, 1989.

Bruce Gooch and Amy Gooch, 2001, *Non-Photorealistic Rendering*. A K Peters, Ltd., Natick

Tobias Isenberg et al., 2002. *Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Stylised Strokes.* Eurographics 2002, 21(3)

Jeff Lander, 2000, *Under the shade of the rendering tree.* Game Developer Magazine, February 2000

A. Lu et al., 2002, *Nonphotorealistic volume rendering using stippling techniques.* Proc. Visualization 2002, 81–89

Aditi Majumder and M. Gopi, 2002, *Hardware-Accelerated Real Time Charcoal Rendering.* In Proceedings of NPAR 2002, International Symposium on Non Photorealistic Animation and Rendering (Annecy, France, June 2002)

Lee Markosian, 2000, *Art-based Modeling and Rendering for Computer Graphics*. PhD thesis, Department of Computer Science at Brown University

J. D. Northrup and Lee Markosian, 2000, *Artistic Silhouettes: A Hybrid Approach.* In Proceedings of NPAR 2000, Symposium on Non-Photorealistic Animation and Rendering (Annecy, France, June 2000), pages 31–37, New York,

Oscar Meruvia Pastor and Thomas Strothotte, 2002, *Frame Coherent Stippling.* Eurographics 2002, 21(3)

Emil Praun et al., 2001, *Real-Time Hatching*. In Eugene Fiume, editor, Proceedings of SIGGRAPH'2001 (Los Angeles, August 2001), pages 581–586, New York

Ramesh Raskar and Michael Cohen, 1999, *Image Precision Silhouette Edges.* In Stephen N. Spencer, editor, Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics, pages 135–140, New York, April 1999

Takafumi Saito and Tokiichiro Takahashi, 1990, *Comprehensible Rendering of 3-D Shapes.* In Computer Graphics (SIGGRAPH '90 Proceedings), volume 24, pages 197–206, August 1990.

A. J. Secord, 2002, *Weighted voronoi stippling.* Proceedings NPAR 2002, pages 37–43