Edge-Enhancement – An Algorithm for Real-Time Non-Photorealistic Rendering

Marc Nienhaus
Hasso-Plattner-Institute for Software Engineering at the University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3
D-14482 Potsdam, Germany
nienhaus@hpi.uni-potsdam.de

Juergen Doellner
Hasso-Plattner-Institute for Software Engineering at the University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3
D-14482 Potsdam, Germany
doellner@hpi.uni-potsdam.de

ABSTRACT

In this paper, we propose an algorithm for enhancing edges of real-time non-photorealistic renderings. It is based on the edge map, a 2D texture that encodes visually important edges of 3D scene objects, and includes silhouette edges, border edges, and crease edges. The edge map allows us to derive and augment a wide range of non-photorealistic rendering styles. Furthermore, the algorithm is designed to be orthogonal to complementary real-time rendering techniques. The implementation is based on multipass rendering: First, we extract geometrical properties of 3D scene objects generating image-space data similar to G-buffers. Next, we extract discontinuities in the image-space data using common graphics hardware to emulate image-processing operations. In subsequent rendering passes, the algorithm applies texture mapping to combine the edge map with 3D scene objects.

Keywords

Edge-enhancement, real-time rendering, G-buffer, non-photorealistic rendering, shading

1. INTRODUCTION

Non-photorealistic rendering (NPR) has become a popular research topic in computer graphics for the last decade. Its application areas are manifold and include illustrations, painterly renderings, and cartoon production. NPR algorithms range from object-based, image-based, or hybrid approaches implementing different rendering styles, e.g., edge enhancement [Ras99a][Nor00a], hatching [Pra01a], or cartoon-style rendering [Lak00a].

Image-based algorithms represent one of the most important categories in NPR. As one of the most important approaches, geometric buffers, known as G-buffers, preserve geometric properties of scene objects in image space such as normals, depth, or object identifiers [Sai90a]. They are evaluated and combined in a post-processing step using image processing. Finally, derived image-space information such as contour lines, is combined with the results of scene rendering to enhance scene elements that are important for visual perception. The derived information is not directly available in object space. Furthermore, G-buffers are general with respect to shape type, and can be implemented in a robust way. Due to its versatility and generality, a challenging task, therefore, is to find appropriate implementations of G-buffer techniques in real-time rendering.

Recent developments in computer graphics hardware, e.g., multitexturing and programmable rendering pipelines, lead to new rendering algorithms. To achieve interactive frame rates, NPR algorithms must be completely accelerated by hardware. The presented work presents such an algorithm for a specific application of G-buffers, namely edge enhancement.

Our edge-enhancement algorithm is based on G-buffers. To achieve real-time performance, G-buffer creation and deriving information are both implemented based on texturing capabilities of graphics hardware. Texturing is a fundamental rendering operation, which is deployed to extract geometric properties, to store intermediate images, to emulate image-processing, and to combine image-space data given as 2D texture with 3D scene objects.

Journal of WSCG, Vol.11, No.1., ISSN 1213-6972
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.

Our algorithm encodes silhouette edges, border edges, and crease edges of 3D scene objects in a 2D texture, called *edge map*. It is created based on image-space information derived from G-buffers, and applied to 3D scene objects as projective 2D texture [Seg92a] in object space.

The implementation is based on a multipass rendering algorithm: 1) We extract geometrical properties of 3D scene objects generating image-space data similar to G-buffers. As a prerequisite, the shapes must provide per-vertex normals. 2) We extract discontinuities in the image-space data, stored as 2D textures, using texturing to emulate image-processing operations. 3) In subsequent rendering passes, the algorithm uses projective texturing to combine the edge map with 3D scene objects. Since fragment shading can be used to specify in detail how edge map contents are blended with fragments of scene objects, various NPR styles can be supported. The edge map distinguishes between two categories of edges, profile edges and inner edges – fragment shading can also handle them differently.

The implementation of the algorithm does not block framebuffer resources because edge maps are created by rendering directly into the texture. Hence, it is orthogonal to other real-time rendering algorithms. For example, it cooperates with real-time mirroring.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 gives a classification of edges. Section 4 describes the implementation of the algorithm. Section 5 presents applications. Section 6 draws conclusions.

2. RELATED WORK

Saito and Takahashi [Sai90a] introduced the G-buffer as a two-dimensional data structure that stores geometric properties of 3D shapes. Important G-buffers are the normal-buffer, z-buffer, and Id-buffer. Furthermore, they describe image-processing operations used to analyze G-buffer contents to produce "comprehensible images", for instance, edge-enhanced or hatched renderings of 3D scene objects. Common image-processing operations applied to 2D images, however, are time consuming and currently not appropriate for real-time rendering. We provide a solution for generating and analyzing G-buffer contents in real-time based on texturing.

Recent research in the field of real-time NPR exploits current graphics hardware. Praun et al. [Pra01a] implement a technique for real-time hatching of 3D shapes. Freudenberg et al. [Fre02a] develop a similar technique for real-time halftoning. They make extensive use of the programmable rendering pipeline and texture blending capabilities of current graphics

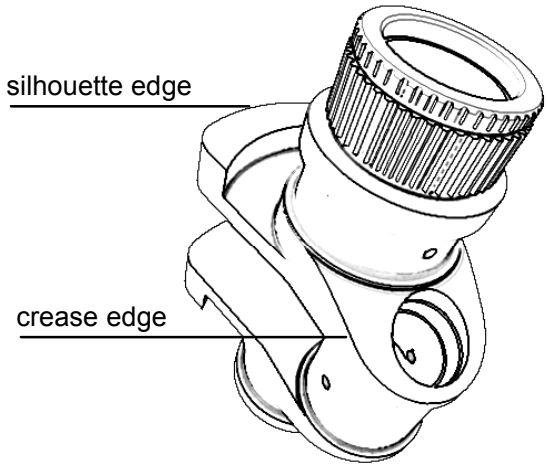

Figure 1: The algorithm treats differently silhouette edges and crease edges. (Model used with permission)

hardware. Our work is complementary and can be combined with these techniques.

NVidia presents an image-space technique to render edges of 3D shapes onto a screen-aligned quad [Dom02a]. It samples adjacent texture values to process encoded normals and detect discontinuities in the normal-buffer. ATI extends this approach by detecting discontinuities in the normal-buffer, z-buffer, and Id-buffer [Mit02a]. This way, edges of 3D shapes, regions in shadow, and texture boundaries can be outlined. Our algorithm works in a similar way but distinguishes different types of edges and uses projective texturing to be orthogonal to other real-time rendering techniques and to support different NPR styles.

Peercy et al. implement an interactive multipass programmable shading system [Pee00a]. It uses projective texturing to combine intermediate shading results with 3D scene objects but does not detect edges. In our approach, projective texturing is used to combine edge maps with 3D scene objects.

Gooch et al. [Goo99a] explore technical illustrations in-depth. We combine their illumination model for technical illustrations [Goo98a] with our edge-enhancement algorithm.

Decaudin introduces cartoon-style rendering of 3D scenes [Dec96a][Dec96b]. His technique uses normal-buffer and z-buffer to detect discontinuities in image-space. The results are edge-enhanced cartoon-style renderings. It is not intended, however, as a real-time rendering technique. Lake et al. extends cartoon style rendering to real-time using texture mapping hardware [Lak00a]. We adapt their approach and combine it with our edge-enhancement

algorithm. Claes et al. discusses artifacts that occur between two consecutive color patches in a cartoon-style rendering [Cla01a]. A variant of our algorithm can be applied to create real-time cartoon-style renderings without these rendering artifacts.

3. EDGE CLASSIFICATION

The edge map is based on the following classification of edges; we assume that 3D scene objects are represented by polygonal meshes.

- A **silhouette edge** is an edge adjacent to a polygon facing towards the camera (front-facing) and one polygon facing in the opposite direction (back-facing).
- A **border edge** is an edge to exactly one polygon.
- A **crease edge** is an edge between two front-facing (or back-facing, respectively) polygons whose dihedral angle is above some threshold. The dihedral angle defines the intensity of a crease edge.

For a given 3D scene object, both silhouette edges and border edges outline the *profile* of that object, while crease edges outline *inner forms* of the object. An example illustrates different types of edges in Figure 1.

A	B	C
D	X	E
F	G	H

Figure 2: A-H are neighboring pixel of X.

In a typical 3D scene, abrupt changes in the z-buffer occur at silhouette edges and border edges, i.e., 0^{th} order discontinuities of the z-buffer indicate profiles. A 1^{st} order differential operator in image space will detect them.

Abrupt changes in the normal-buffer occur typically at crease edges. A normal-buffer is a three-channel image containing encoding normalized x, y, and z coordinates of normals derived from fragments resulting from 3D scene objects. 0^{th} order discontinuities of the normal-buffer indicate crease edges and, furthermore, profile edges if the normal of the adjacent front-facing polygon varies from the normal of an underlying polygon. So, profile edges producing small discontinuities in depth that can hardly be detected can be detected in the normal-buffer.

4. EDGE MAP ALGORITHM

Conceptual Structure of the Algorithm

The algorithm is conceptually divided into two parts. In the first part, it constructs the edge map. For this, it generates image-space data similar to G-buffers to geometrical properties of 3D scene objects. Then, the algorithm applies image-space operations to extract discontinuities in G-buffer contents. The discontinuities form edges of 3D shapes.

In the second part, the algorithm combines edges with any NPR-rendering algorithm using fragment shading. It uses the intenseness of discontinuities to

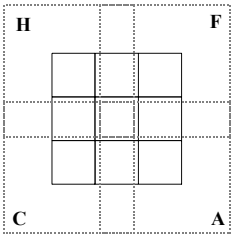

Figure 3: Accessing neighboring texels by shifting texture coordinates diagonally.

blend between edge color and surface shade of 3D shape. Furthermore, it distinguishes between profile and inner edges and applies different edge colors.

Detection of Edge Intensities

4.1.1 Generation and Storage of G-Buffers

In the first rendering pass, all 3D scene objects that are declared in the scene graph to be enhanced are rendered into a 2D texture with fragment normals and fragment z-value.

To derive that information, two textures are used for scene rendering. 1) A normalization cube map textures 3D shapes and produces eye-space normalized fragment normals, which are encoded in the RGB color channels. 2) A 1D alpha-texture, projected from the camera position along the viewing direction [Hei99a], textures 3D shapes and produces fragment z-values, which are stored in the alpha channel. Note that the alpha-texture produces z-values that correspond to the linear distance from the camera position, in contrast to the values stored in the z-buffer on graphics hardware. So, the accuracy of discontinuities in z-values is independent from the position of the shape. Furthermore, no additional rendering pass is needed to construct z-buffer contents on graphics hardware to produce fragment z-values. Unfortunately, the alpha-texture has lower precision than the z-buffer. To cope with the low precision, we minimize the viewing frustum in z-dimension. We adjust the near and far clipping-planes to exactly match the bounding volumes of shapes to

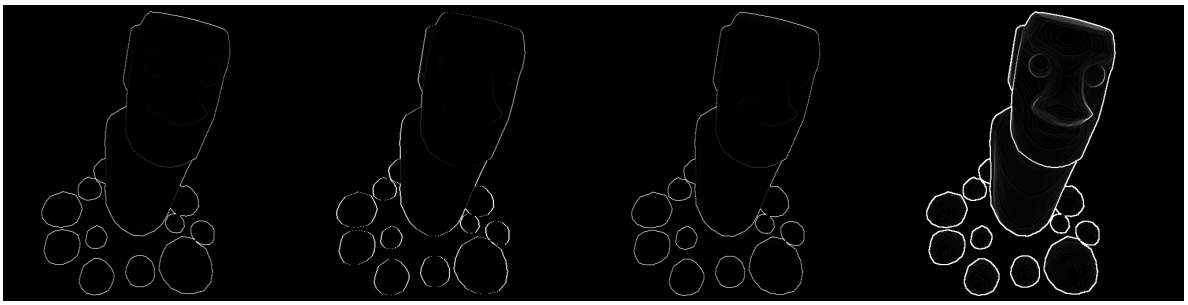

Figure 4: Blending all three intermediate renderings (left) results in the final edge map (right).

be enhanced. In the future, graphics hardware will provide textures of higher precision [Kil02a].

RGB and alpha values encoding normal and depth information are combined using fragment processing and rendered into one 2D texture T_G. Thus, the texture stores both G-buffer data in a single texture. T_G and the framebuffer are of equal size and, therefore, provide a one-to-one relationship between pixels and texels. The implementation uses the render-to-texture extension or, alternatively, a P-buffer canvas [Kil02b] for this rendering pass. They are one of the reasons for the real-time capability of our algorithm.

4.1.2 Edge-Detection in Image-Space

Today's 3D rendering systems (still) do only provide a limited image processing functionality and is typically not accelerated by graphics hardware. To detect discontinuities in image data, linear filtering can be used, for instance, based on the Sobel filter. For each pixel to be filtered, linear filtering takes into account its neighboring pixels (Fig. 2).

Our algorithm implements linear filtering based on intermediate rendering passes, which render a textured screen-aligned quad using multi-texturing. Let us assume to have four texture units available on graphics hardware, each of which uses T_G as texture. The quad fits completely into the viewport of the P-buffer. We present two implementation variants.

The first variant considers four neighboring texture samples, processing discontinuities between values of the two diagonally opposite sample pairs (A, H) and (C, F). For this, we shift diagonally texture coordinates for each texture unit in *s* and *t* direction. Using the reciprocal of width and height of the P-buffer permits us to access the adjacent texture samples A, C, F, and H (Fig. 3). The technique assembles texture color and alpha values and then filters out discontinuities using fragment processing. We evaluate the RGB values by:

$$I_N = \frac{1}{2} \cdot ((\text{expand(A) dot expand(H)}) + (\text{expand(C) dot expand(F)}))$$

The dot products correspond to the cosines of the angles between opposite normal vectors. The averaged result I_N denotes the intensity of discontinuities in the normal buffer[1]. The resulting intensity is stored in the RGB channels of the P-buffer.

We evaluate the alpha values by:

$$I_Z = (1 - \frac{1}{2}|A - H|)^2 \cdot (1 - \frac{1}{2}|C - F|)^2$$

I_Z denotes the intensity of discontinuities of z-values; it is stored in the alpha channel of the P-buffer.

After the first intermediate rendering pass the P-buffer contains intensity values in the RGB channels that represent discontinuities in the normal-buffer and intensity values in the alpha channel that represent discontinuities in the z-buffer.

The second variant implements a common C_1-discontinuity operator that considers eight neighboring samples (A-H) and the center texture sample X. It is only applied to the z-buffer. Saito and Takahashi as well as Decaudin propose the following C_1-discontinuity operator to detect discontinuities in the z-buffer:

$$I_Z = \frac{1}{8}(|A - X| + 2|B - X| + |C - X| + 2|D - X| + 2|E - X| + |F - X| + 2|G - X| + |H - X|)$$

We calculate texture coordinates to sample neighboring values A-H. No shift in texture coordinates is needed for X. The technique requires three intermediate passes, each one calculating a portion of the operator. In the first intermediate pass, we access A, B, C, and X. Then,

$$\frac{1}{8}|A - X| + \frac{1}{4}|B - X| + \frac{1}{8}|C - X|$$

[1] A, C, H, and F correspond to the RGB values or the alpha value, respectively, of each texture sample. expand(c) evaluates $f(x) = 2 \cdot x - 1$ for each component of color c. Thus expand(A) yields the normal value encoded in the RGB color channel of texel A.

is processed for each fragment based on the alpha components. The result is stored in the alpha channel of the P-buffer. In the second intermediate pass, we access D, E, and X using equation

$$\frac{1}{4}|D-X|+\frac{1}{4}|E-X|$$

The result is added to the P-buffer using blending. Finally, the third intermediate rendering pass evaluates

$$\frac{1}{8}|F-X|+\frac{1}{4}|G-X|+\frac{1}{8}|H-X|$$

and, again, adds it to the P-buffer. Figure 4 demonstrates the described accumulation of intermediate results.

We observed that the edge intensities calculated by the first variant are visually superior compared to those calculated by the second variant (Fig. 5). In particular, the first variant produces more anti-aliased edges.

In the final step, we copy the P-buffer RGBA contents into a texture – the edge map – that can be used by NPR techniques.

Using the Edge Map for NPR

The edge map is available for subsequent scene graph rendering passes. As one application, we can project back the edge map onto 3D scene objects using projective texturing. In this case, we must use the same camera settings for both, edge map construction and scene graph rendering.

Projective texturing permits us to introduce edge information in the final image by allocating just one texture unit of the graphics hardware. In addition, rendering algorithms set up fragment shading

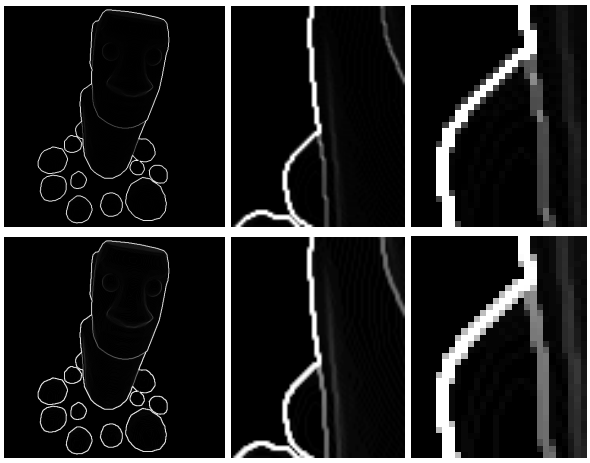

Figure 5: The C_1-operator (upper images) produces artifacts. The diagonally filtering detects finer discontinuities and produces anti-aliased edge intensities.

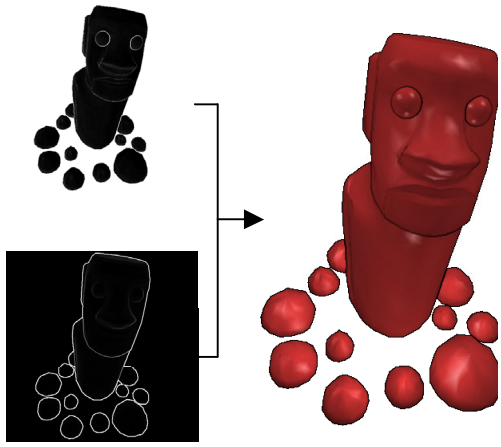

Figure 6: Normal (upper left) and depth (lower left) discontinuities form edges to enhance a simply shaded statue.

calculations that consider intensity values stored in the edge map. Figure 6 gives an example of an edge-enhanced simply shaded statue.

Rendering Passes

The rendering passes that occur using our technique are summarized below:

1. Scene-graph rendering pass of edge-enhanced scene objects to generate normal-buffer and z-buffer.
2. Intermediate rendering pass to generate edge intensities (1 pass for diagonal filtering, 3 passes for C_1-discontinuity operator).
3. Scene-graph rendering passes of the NPR techniques using the edge map, e.g., as projective texture.

Intermediate rendering passes are not time consuming since they just render a single screen-aligned quad. Since the edge map is stored in a 2D texture, it is orthogonal to subsequent rendering passes. Management of framebuffer resources, for instance the stencil buffer of OpenGL, is not necessary. The following section discusses edge-enhancement in combination with extended real-time rendering techniques.

5. APPLICATIONS OF EDGE MAPS

Technical Illustrations

Gooch et al. study technical illustrations that communicate shape, structural and material composition of objects more comprehensibly than traditional lighting and shading [Goo99a]. Enhancing differently profile edges and inner edges assists the illustrative representation of objects.

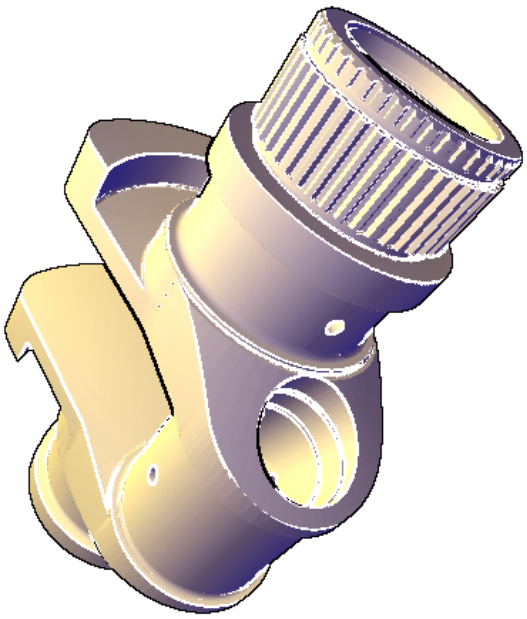

Figure 7: Gooch Illumination model for technical illustrations enhanced with different edge colors using the edge map. (Model used with permission)

We implement their illumination model for technical illustrations using Phong shading based on vertex and fragment processing. Fragment processing distinguishes between profile edges and inner edges: First, for inner edges, we linearly interpolate between the color given by the lighting model and white as edge color using intensity values in the RGB channels of the edge map as weights. Then, for profile edges, we linearly interpolate between the resulting color of the previous weighting and black as edge color using the intensity values in the alpha channel of the edge map as weights. Altogether, one frame requires two traversals of the scene graph and one intermediate rendering pass to implement edge-enhanced technical illustrations (Fig. 7).

Edge-Enhanced Real-Time Hatching

Halftoning produces a variety of non-photorealistic renderings, for instance, pen-and-ink style drawings. Freundenberg et al. introduced a real-time algorithm for halftoning [Fre02a] that uses a smooth threshold function to achieve anti-aliased halftone results. We implement this technique using a prioritized stroke texture to generate real-time edge-enhanced hatching. Since it operates on a per-fragment basis, it can be combined with edge maps in a straightforward way (Fig. 8). In general, a wide range of edge-enhanced NPR styles can be derived.

Mirroring & Edge-Enhanced Renderings

Our edge-enhancement algorithm can be combined with many advanced, multipass real-time rendering algorithms [Bly99a] such as mirroring, shadow casting, and bump-mapping.

Figure 8 shows a mirrored, edge-enhanced scene. The implementation uses the stencil buffer to mask the mirror surface in screen-space, and requires two rendering passes, one for the scene without mirror and one for the mirror. We have to apply the edge-enhancement algorithm twice, for the original scene and for the mirrored scene. The scene is rendered from different viewpoints and, hence, the edges are detected correctly (Fig. 8). No extension to the existing scene graph implementation of the mirror rendering technique needs to be implemented.

Edge-Enhanced Cartoon-Style Rendering

Lake et al. define a 1D texture that stores cartoon-like shades depending on material and light properties [Lak00a]. The dot product between the surface normal and the light direction vector on the surface is used to index that texture. Nearest filtering produces edged though jagged transition between colors.

We implement the cartoon-style rendering technique by Lake et al. and combine it with our edge map to enhance edges (Fig. 9).

Furthermore, we can reduce artifacts between consecutive color patches that occur when using nearest texture filtering. For this, we use a variation

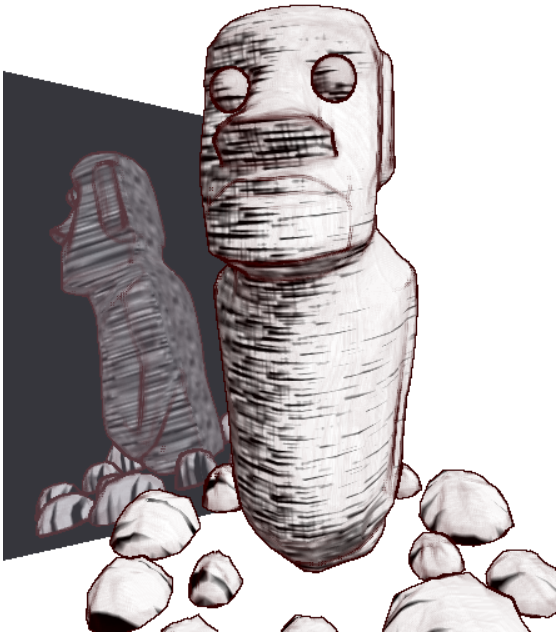

Figure 8: Real-time hatching in combination with our edge-enhancement algorithm applied to a statue. The mirrored statue shows enhanced edges, too, which are perspectively correct.

Figure 9: The edge-enhanced, cartoon-shaded "Olaf". Nearest texture-filtering produces artifacts between consecutive color patches (left). Sampling neighboring color values smoothes color transition (right).

of our algorithm.

In the first step, we render the scene in a cartoon-style using vertex programming and store the RGB channels of the P-buffer as 2D texture. In the next step, we sample that texture to anti-alias its contents. The result is stored in a 2D texture, called cartoon map. Then, the edge map is created as described before. In the final step, we render the scene using both, edge map and cartoon map to generate anti-aliased cartoon-style renderings in real-time (Fig. 9). The example demonstrates the versatility of our approach with respect to sampling and processing texture values in intermediate rendering passes.

6. CONCLUSION

The presented algorithm handles silhouette, border, and crease edges. Edge intensities are available as edge map, a 2D texture that contains the screen-space information. The edge map is well suited to cooperate with many real-time rendering techniques, in particular NPR techniques. Its implementation is completely accelerated by today's graphics hardware.

The approach is limited with respect to edge-enhancement, which is restricted to the geometrical boundaries of 3D scene objects due to projective texture mapping and the demanding shading calculations to enhance edges. Furthermore, edges in the edge map are determined from 3D shape and are just a few texel wide. So, artistic silhouettes (e.g., [Nor00a]) in the periphery of the surface are not possible.

Regarding the future development of OpenGL, real-time rendering techniques based on high-level shading languages can benefit from the presented algorithm due to its orthogonality. In particular, non-photorealistic shaders (e.g., [Fre02b]) could directly take advantage of edge maps to increase their expressiveness.

7. ACKNOWLEDGMENTS

Thanks to Florian Kirsch and the other members of the VRS group at the Hasso-Plattner-Institute for their cooperation.

8. REFERENCES

[Bly99a] Blythe, D., Grantham, B., Kilgard, M. J., McReynolds, T., and Nelson, S. R. Advanced Graphics Programming Techniques Using OpenGL. In SIGGRAPH'99 Course Notes, August, 1999

[Cla01a] Claes, J., Di Fiore, F., Vansichem, G., and Van Reeth, F. Fast 3D Cartoon Rendering with Improved Quality by Exploiting Graphics Hardware. Proceedings of Image and Vision Computing New Zealand (IVCNZ), pp.13-18, November 2001.

[Dec96a] Decaudin, P. Rendu de scénes 3D imitant le style «dessin animé». Rapport de Recherche 2919. Université de Technologie de Compiègne, France, 1996.

[Dec96b] Decaudin, P. Modélisation par Fusion de Formes 3D pour la Synthèse d'Image – Rendu de Scènes 3D imitant le Style"Dessin Animé". Ph.D. Thesis, Université de Technologie de Compiègne, France, December 1996.

[Dom02a] Dominé, S., Rege, A., and Cebenoyan, C. Real-Time Hatching (Tribulations in). Game Developers Conference 2002, San Jose, CA, March 2002. http://developer.nvidia.com/docs/IO/2648/ATT/GDC2002_RealTimeHatching.pdf (PDF format)

[Fre02a] Freudenberg, B., Masuch, M., and Strothotte, T. Real-Time Halftoning: A Primitive For Non-Photorealistic Shading. 13th Eurographics Workshop on Rendering. Pisa, Italy, pp.1-4, June 2002.

[Fre02b] Freudenberg, B. A Non-Photorealistic Fragment Shader in OpenGL 2.0. SIGGRAPH 2002 Talk. San Antonio, July 2002.

[Goo98a] Gooch, A., Gooch, B., Shirly, P., and Cohen, E. A Non-Photorealistic Lighting Model for Automatic Technical Illustration. In Computer Graphics (Proceedings of SIGGRAPH'98), Orlando, FL, pp.447-452, July 1998.

[Goo99a] Gooch, B., Sloan, P. S., Gooch, A., Shirley, P., and Riesenfeld, R. Interactive Technical Illustration. ACM Symposium on Interactive 3D Graphics 1999, Atlanta, GA, pp.31-38, April 1999.

[Hei99a] Heidrich, W. High-quality shading and lighting for hardware-accelerated rendering. Ph.D. Thesis, Universität Erlangen, February 1999.

[Kil02a] Kilgart, M. (Ed.). NVidia OpenGL Extension Specification for the CineFX Architecture (NV30). NVidia Corporation, August 2002. http://developer.nvidia.com/docs/IO/3260/ATT/nv30specs.pdf (PDF format)

[Kil02b] Kilgart, M. (Ed.). NVidia OpenGL Extension Specifications. NVidia Corporation, October 2002. http://developer.nvidia.com/docs/IO/1174/ATT/nvOpenGLspecs.pdf (PDF format)

[Lak00a] Lake, A., Marshall, C., Harris M., and Blackstein, M. Stylized Rendering Techniques for Scalable Real-Time 3D Animation. In Proceedings of NPAR 2000: First international symposium on Non-photorealistic animation and rendering, Annecy, France, pp. 13-20, June 2000.

[Mit02a] Mitchell, J. L., Brennan, C., and Card, D. Real-Time Image-Space Outlining for Non-Photorealistic Rendering. SIGGRAPH 2002 Sketch, San Antonio, July 2002.

[Nor00a] Northrup, J.D., Markosian. L. Artistic Silhouettes: A Hybrid Approach, In Proceedings of NPAR 2000: First International Symposium on Non-Photorealistic Animation and Rendering, Annecy, France, pp.31-37, June 2000.

[Pee00a] Peercy, M. S., Olano, M., Airey, J., and Ungar, J. P. Interactive Multi-Pass Programmable Shading. In Computer Graphics (Proceedings of SIGGRAPH'00), New Orleans, LA, pp.425-432, July 2000.

[Pra01a] Praun, E., Hoppe, H., Webb, M., and Finkelstein, A. Real-Time Hatching. In Computer Graphics (Proceedings of SIGGRAPH'01), pp.579-584, 2001.

[Ras99a] Raskar, R., and Cohen, M. Image Precision Silhouette Edges. ACM Symposium on Interactive 3D Graphics 1999, Atlanta, pp.135-140. 1999.

[Sai90a] Saito, T. and Takahashi, T. Comprehensible Rendering of 3-D Shapes. In Computer Graphics (Proceedings of SIGGRAPH'90), 24(4), pp.197-206, August 1990.

[Seg92a] Segal, M., Korobkin, C., van Widenfalt, R., Foran, J., and Haerberli, P. Fast Shadows and Lighting Effects using Texture Mapping. In Computer Graphics (SIGGRAPH'92 Proceedings), 26(2), pp.249-252, July 1992.