Ray Tracing NPR-Style Feature Lines

A.N.M. Imroz Choudhury*
Scientific Computing and Imaging Institute

Steven G. Parker†
NVIDIA Corporation
Scientific Computing and Imaging Institute

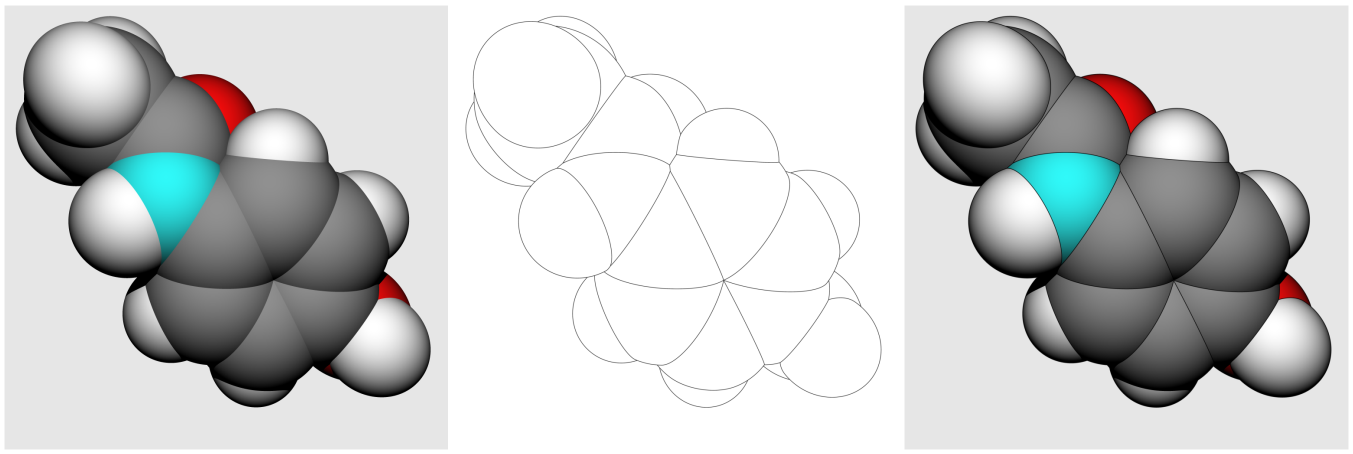


Figure 1: *Acetaminophen molecule rendered with lambertian shading (left), using our NPR line renderer (middle), and a composite of both renderings (right). Between any two bonded atoms, a line marks the intersection; around the outer edge of the molecule, silhouette edges are visible.*

Abstract

We present an algorithm for rendering constant-width line-like primitives within a ray tracing framework, which can render simple NPR-style feature lines, including silhouette edges, crease lines, and primitive intersection lines. The algorithm is based on ray differentials, which measure distances in screen space and are used to detect and render constant-width feature lines. This technique allows for a host of NPR-style raster algorithms to be ported to ray tracers.

CR Categories: K.6.1 [Management of Computing and Information Systems]: Project and People Management—Life Cycle; K.7.m [The Computing Profession]: Miscellaneous—Ethics

Keywords: ray tracing, lines, creases, silhouettes, intersections

1 Introduction and Background

Though the goal of ray tracing and other physically based rendering techniques is ultimately to produce photorealistic images, it is often helpful to use non-photorealistic rendering techniques to illustrate or highlight certain features [**?**; **?**], such as the implied molecular bonds in Figure 1 or the overall shape of the rounded cube in Figure 7. In raster graphics, many NPR techniques use *line primitives* for such goals as conveying the shape of complex models with minimal on-screen clutter [**?**; Judd et al. 2007; Ohtake et al. 2004], illustrating technical models by their outlines and hidden contours [Dooley and Cohen 1990], and conveying confidence in architectural models for inviting further discussion and design [Potter et al. 2009]. Lines are useful in such applications because they are expressive of underlying shape without being overly intrusive [Saito and Takahashi 1990].

Drawing lines in a raster graphics setting is a primitive operation that uses classic line rasterization algorithms (e.g., the Bresenham algorithm [Bresenham 1965]). The lines thus produced can be used to highlight features such as sharp corners and silhouette edges. Because three-dimensional geometry is also rasterized to the screen, 3D primitives can be freely mixed with line primitives to produce a variety of illustrative effects.

However, in ray tracing there is no obvious way to "rasterize" a line; instead, all primitives are detected by intersecting camera rays with scene geometry. Because lines are infinitely thin, they are troublesome for the ray tracing algorithm, which operates on "physical" primitives that have at least two dimensions. It is possible to represent lines with, e.g., long, thin pipe-like primitives; however, such primitives have world-space thickness, and therefore change their appearance on-screen as the camera position or zoom level changes. In raster graphics, however, line-drawing algorithms render lines with a constant thickness on-screen, so their appearance is not a function of the scale of the scene or the viewing projection.

This paper presents a method for ray tracing feature lines within regularly rendered scenes with a scale-independent on-screen width that is controlled by the user. Our method demonstrates how a variant of line rasterization can be included in a ray tracer, thus allowing for the inclusion of NPR-style enhancements. It is based on *ray differentials* [Igehy 1999], which we use a variant of to detect edges in the view of the scene geometry projected to the image plane; these edges are in turn used as the basis for line drawing. Including these

*email: roni@cs.utah.edu
†email: sparker@nvidia.com

lines gives the viewer additional cues to relative positions of objects within the scene, and also enhances particular features within objects, such as sharp corners.

1.1 Feature Lines

Feature lines are linear manifolds that denote geometrically interesting features of objects. The ones this paper focuses on are

- *silhouette edges* [Gooch et al. 1998], marking the boundary of an object in screen space against the background;
- *intersection lines*, marking the curves along which two primitives intersect;
- and *crease edges* [Saito and Takahashi 1990], indicating curves along which there is a discontinuity in a primitive's normal field (e.g. the sharp corners of a box).

Crease edges can highlight the bounding surface of an object by indicating salient features such as sharp folds, while silhouette edges and intersection lines help set apart different objects from their neighbors. Such visual cues are important when understanding the relative positions of objects in a scene is the primary goal, such as in technical illustrations [Dooley and Cohen 1990], molecular graphics [Tarini et al. 2006], or particle data visualization [Bigler et al. 2006; Bigler 2004].

1.2 Ray Differentials and Screen Space

Igehy describes ray differentials [Igehy 1999], which provide a measure of how a given ray's screen space "neighbors" interact with the scene geometry. By parameterizing a ray both in terms of where it intersects the image plane, and in terms of its propagation distance (i.e. its t value), it is possible to derive derivatives of the ray with respect to screen dimensions. The ray differential describes how a slightly offset ray would behave; generalizing, it is possible to describe how the ray's *screen footprint* behaves.

In Igehy's paper, the primary application of ray differentials is in surface-local texture filtering for higher quality images. He also mentions geometric level-of-detail, ray tracing caustics, and dull reflections; these applications all depend in some form on understanding what happens in the screen-space vicinity (i.e. the footprint), of a traced ray. Our method uses ray differentials to measure *distances in screen space*, allowing for the detection of features in world space, and drawing corresponding feature lines with constant width on the screen.

2 Related Work

The computer graphics literature shows a wealth of line rendering techniques, mostly within the domain of non-photorealistic rendering. Here we review several examples in order to give a flavor of the types of techniques, and to motivate their usefulness in general.

2.1 Wireframe Rendering

Perhaps the simplest way to include NPR-style lines in a rendering is to use a simple two-pass rendering algorithm: first solid geometry is rendered, then the geometry wireframes are overlaid. For certain geometries, this process yields both crease and silhouette edges (as for cubes or hexahedra), and the approach has been hybridized with a ray tracer for mesh visualization [Ernst and Greiner 10-12 Sept. 2007]. Refinements of this basic technique exist (Bærentzen et al. [Bærentzen et al. 2006], for example, developed a single-pass, fragment-shader based approach that produces high-quality images with little to no loss in performance) but such techniques cannot capture other interesting features, such as intersection lines.

2.2 Applications of Line Drawing

There are several rendering algorithms that include NPR-style lines, most of them applying to raster graphics. To give an idea of the range of applications, several such techniques are reviewed here.

Expressing Shape. Dooley et al. [Dooley and Cohen 1990] describe an algorithm for creating expressive line renderings of 3D objects using solid, dashed, and dotted lines of varying thickness to indicate places where lines meet or run behind visible surfaces. Their system focuses on silhouettes and creases to convey overall structure; with just a few lines, whole objects can be understood, including ordinarily invisible parts that are occluded by the visible surfaces.

In the approaches of Dooley and Strothotte, the entire rendering is made from line primitives. There are also applications in which lines are drawn on top of 3D renderings in order to draw attention to or otherwise emphasize particular features. Saito et al. [Saito and Takahashi 1990] explore techniques for extracting edges, creases, and silhouettes by collecting depth or other values in extended framebuffers and then applying standard image processing techniques. Raskar et al. [Raskar and Cohen 1999] describe a method for extracting and display silhouette edges with image precision, optionally rendering such lines with varying thickness to emulate charcoal or watercolor drawings. Gooch et al. [Gooch et al. 1998] describe an NPR lighting model that includes silhouette edges. Suggestive contours [DeCarlo et al. 2003], ridge-valley lines [Ohtake et al. 2004], and apparent ridges [Judd et al. 2007] are other approaches that use surface curvature to define classes of lines that can convey shape.

Emulating Artistic Strokes. Strothotte et al. [Strothotte et al. 1994] present a "sketch renderer" that emulates human-made pencil sketches in drawing scenes. The system can redraw portions of scenes at runtime with more or less detail than before, drawing the viewer's eye to new places within the image and allowing the modeller to influence how viewers see the scene. In a similar application, Potter et al. [**?**] use perturbed lines to express various levels of sketchiness in architectural renderings, allowing viewers to see which parts are rendered with higher confidence, and which parts require further discussion and design.

Scientific Visualization. Bigler et al. [Bigler et al. 2006; Bigler 2004] discuss the use of silhouette edges for enhancing geometric features in glyph-based scientific visualization of particle data sets. Their approach adds silhouettes to a ray traced image by first creating a depth buffer and then convolving the depth values with a Laplacian kernel to detect "edges" in the depth image (similarly to the approach of Saito et al. [Saito and Takahashi 1990]); finally, these values are compared with a threshold to decide where to place the edges on top of the rendered image. By varying the threshold, the silhouettes will capture either groups of objects that are close together, or each object by itself.

This approach is notable as one of the few techniques that combines a raster-style algorithm with a ray traced image. The method presented here aims to demonstrate that, in general, such algorithms can be adapted to work fully within a ray tracing framework. The wealth of NPR techniques for raster graphics shows their usefulness in many situations: we wish to incorporate the general machinery underlying these techniques into the framework of ray tracers.

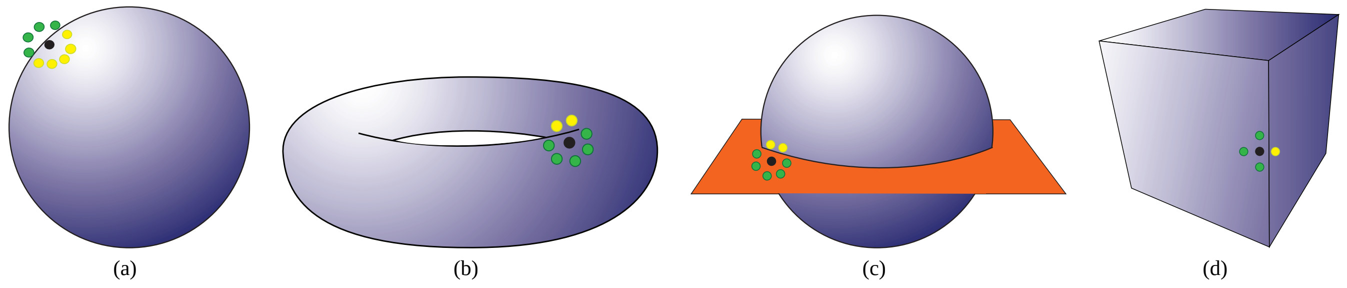


Figure 2: *The different types of feature lines our algorithm captures, illustrated schematically. (a) Silhouette edges, where the edge of an object lies against the background. (b) Self-occluding silhouettes, where an edge of an object lies against a farther portion of the same object. (c) Intersection lines, where two objects intersect. (d) Crease edges, where an object's normal changes discontinuously.*

3 Computing and Drawing Feature Lines

Generally, a ray tracer shoots several *sample rays* from a camera through an image plane into the scene being rendered. In the simplest case, one sample ray is shot through the center of each pixel of the target image. Each sample ray computes a color by interacting with scene geometry (possibly shooting shadow or other secondary rays); the sample colors are combined to yield colors for each pixel in the final rendered image.

3.1 Tracing Ray Stencils

Detecting whether a sample ray strikes the scene near a feature line requires knowing what happens in some neighborhood of the sample ray, i.e., how the ray differential interacts with the scene. To approximate the differential for a sample ray s, a disc-shaped *ray stencil* $D_h^N(s)$ is constructed about s with radius h in screen space, and using a quality parameter N. The parameter N determines the number of rays M in the stencil (note that M does not count the sample ray itself). Figure 3a describes the construction of $D_h^N(s)$.

Within the disc sampled by the ray stencil, feature lines can be found as follows. s strikes some primitive in the scene, or else it strikes the background: in either case, it is associated with a *geometry ID* g_s describing what it has struck,[1] and a parametric distance t_s to the intersection point. Each stencil ray r is traced and associated with its own geometry ID g_r and parametric distance t_r.

Let m be the number of rays r in $D_h^N(s)$ for which $g_r \neq g_s$ (i.e. the number of rays striking *different* geometry from s). Depending on the value of m, one of the following situations results:

1. *The stencil straddles different geometry IDs ($0 < m \leq M$).* Some of the stencil rays strike different objects than the sample ray does so the sample ray is near either a silhouette edge or an intersection line (Figure 1 demonstrates both types).

2. *All the stencil rays strike the same geometry as the sample ray ($m = 0$).* When the entire ray stencil strikes the same object, the possible feature lines are self-occluding silhouettes and crease edges. A crease edge occurs when the surface normal changes discontinuously (Figure **??**). Numerically, we define a crease edge occuring on any sample whose ray stencil indicates a very large magnitude normal gradient. The normal gradient is computed using the finite difference stencil contained withing the ray stencil (Figure 3a). If the gradient is larger than some threshold, then the sample lies near a crease edge.

   If a crease edge is not found, then the sample must be checked for a self-occluding silhouette (Figure 4). These occur when some of the stencil rays strike the object at a significantly farther parametric distance than the sample ray does. Define d to be the number of stencil rays r for which $|t_r - t_s| > T$ (where T is a threshold that depends on the particular scene and primitive, but can usually be set to some fraction of t_s, allowing for some view dependence on how the silhouette is drawn). If $d > 0$ then the sample lies near a self-occluding silhouette.

[1] In the simplest setup, the geometry ID is identical with the "primitive ID" of the primitive the ray has struck. However in some cases it may be more meaningful to associate a more complex object made up on simpler primitives (e.g. a ray striking a meshed object will actually strike a triangle primitive, but the geometry ID can instead reference the mesh itself), hence the use of the term *geometry ID*. Both primitive IDs and geometry IDs can usually be identified with the unique pointer value specifying the appropriate object in memory.

When the feature type is determined, an *edge strength metric* is used to compute how dark the associated feature line should be drawn. The stronger the edge, the darker it will be drawn. An edge strength metric is a function e_M mapping a natural number less than or equal to M to a real number in the range $[0, 1]$. The edge strength e_s for the sample ray s is $e_M(m)$ if the feature is a silhouette or intersection, $e_M(d)$ if it is a self-occluding silhouette, and 1 for a crease edge.

A simple example of an edge strength metric, which is used for the examples in this paper, is

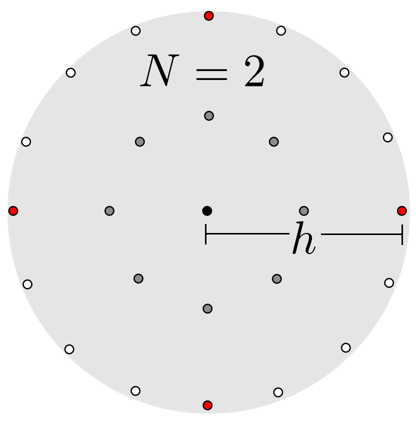
$$e_M(i) = 1 - \frac{|i - \frac{1}{2}M|}{\frac{1}{2}M}. \tag{1}$$

This function rises linearly with i from zero to one for $i \in [0, \frac{1}{2}M)$, and then decreases linearly back to zero for $i \in (\frac{1}{2}M, M]$. It reflects the fact that when a ray stencil is situated with half its area in one geometry region and half in another, it is measuring the strongest possible edge between two regions (Figure 3c). In this case, $m = \frac{1}{2}M$, and $e_M(m) = 1$. The value of this function is used to blend the sample color c_s with black, thus rendering feature lines.

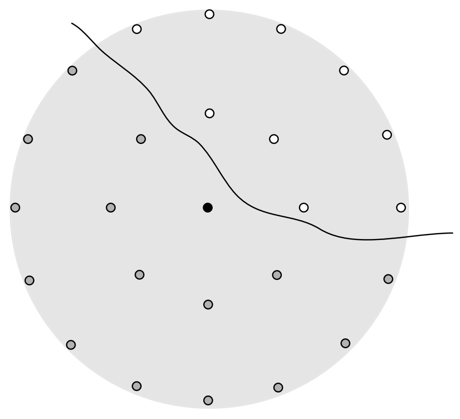
The result of tracing and shading a sample ray, and then tracing the associated ray stencil, is therefore a sample color c_s and an edge strength e_s.

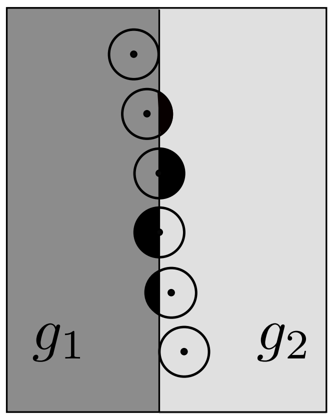
3.2 Feature Line Rendering Algorithm

To find and draw feature lines, we use a modified version of the ordinary ray tracing algorithm. For a particular sample ray, the al-



(a)


(b)



(c)

Figure 3: *The details of how ray stencils work. (a) The ray stencil $D_h^N(s)$ samples a disc of radius h around the sample ray position s (black). The samples lie in N concentric circles (dark gray, light gray/red) about s, where N is the* quality parameter *(in this example, $N = 2$ for two rings of samples). The largest circle has radius h. The red samples indicate a finite difference stencil that can be used to measure gradients in searching for crease edges. (b) A ray stencil being used to measure foreign primitive area. s strikes some primitive below the black line, and a different primitive lies above the line. The stencil (which excludes s) contains twenty four rays ($M = 24$), and nine of them strike a foreign primitive ($m = 9$). Using the linear edge strength metric (Equation 1), which measures how close to* half *of the samples strike a different primitive, we have $e_s = e_M(9) = 62.5\%$. (c) In the limit as $N \to \infty$, a ray stencil becomes a circular disc, minus its center point. The disc moves across a primitive boundary, from geometry ID g_1 to g_2, acting as an* area indicator *for the portion of the filter that lies on a foreign primitive (shaded black, the foreign primitive is g_2 for the top three discs, and g_1 for the bottom three). For the six "snapshots" shown in this example, the disc moves a distance in screen space of $2h$, where h is the disc radius. The foreign area increases from zero to one-half, "flips" to the left side as the center of the filter crosses the boundary, and then decreases back to zero. The foreign primitive area is used to define an edge strength, which in turn is used to render feature lines.*

gorithm runs as follows: the sample ray s is traced and shaded as normal with color c_s. A ray stencil $D_h^N(s)$ is constructed about s and the stencil rays are traced until they strike the scene geometry. Depending on m, one of the two cases in Section 3.1 is triggered, and an edge strength e_s is computed. The sample color c_s is blended with black, using the edge strength as an interpolation factor, yielding a darkened color $c_s(1 - e_s)$, and this color replaces the original sample color. For full edge strength, the sample will be black and for zero edge strength it will be shaded as normal. Between these extremes lies a spectrum of darkened colors, usually serving as a "halo" for the darkest part of the line. By changing the radius of the stencil h, the thickness of the line can be varied. Because the lines are computed in screen space, this value h always yields lines of the same width, regardless of the scale of the scene, camera zoom parameters, etc. Furthermore, because edge strength determines darkness, the lines are inherently anti-aliased via prefiltering (in much the same way as the wireframe techniques of Bærentzen et al. [Bærentzen et al. 2006]).

In essence, this procedure searches for zeroth and first order discontinuities in the geometry ID image and depth image in order to find the target feature lines. The method is therefore similar to the approach of Saito et al. [Saito and Takahashi 1990], but fully adapted to work within a general ray tracing framework.

4 Discussion

4.1 Extension to Multiple Bounces

The key insight in the algorithm presented here is the computation of a ray stencil in the screen space. The stencil approximates the ray differential, which in turn represents the changes taking place in the image as one moves some distance in screen space. The algorithm draws feature lines on the image itself, depending on how the geometry behaves when projected to the image plane. It does not draw feature lines on reflected images, such as those produced by specular reflection. The method can be extended to do so by continuing to trace stencil rays that would normally be reflected due to effects such as specular reflection. This can be done in the case where all the stencil rays strike the same object, and that object supports specular reflection. In essence, by tracing such rays to see what primitives they end up striking in the second bounce, reflected images can have their feature lines computed exactly as the primary image does. This process can be repeated so long as all of the stencil rays strike the same object in each bounce.

4.2 Ray Stencils and Image Filtering

The idea of ray differentials and their instantiation with ray stencils is the central concept of our method. By arranging the ray stencil in screen space with a fixed radius, and allowing the rays the compute scene information, a projection of the scene onto the disc sampled by the stencil is accomplished. Through the stencil ray t values and geometry IDs, the projection actually produces two images: a geometry identification image and a depth image.

In computing the silhouette and intersection lines, the counting process described in Section 3.2 is essentially an *area estimation*, taking the area of the stencil covering "foreign" geometry IDs to be a measure of how strong of an edge lies within the disc. Formulating the process this way recalls the SUSAN edge detector from image processing [Smith and Brady 1997] which also measures areas of dissimilar brightness.

As constructed, the ray stencils contain a first-order finite difference stencil. This is used to compute the gradient of the normal image, discontinuities of which indicate sharp corners. First-order discontinuities in the normal are related to second-order discontinuities in

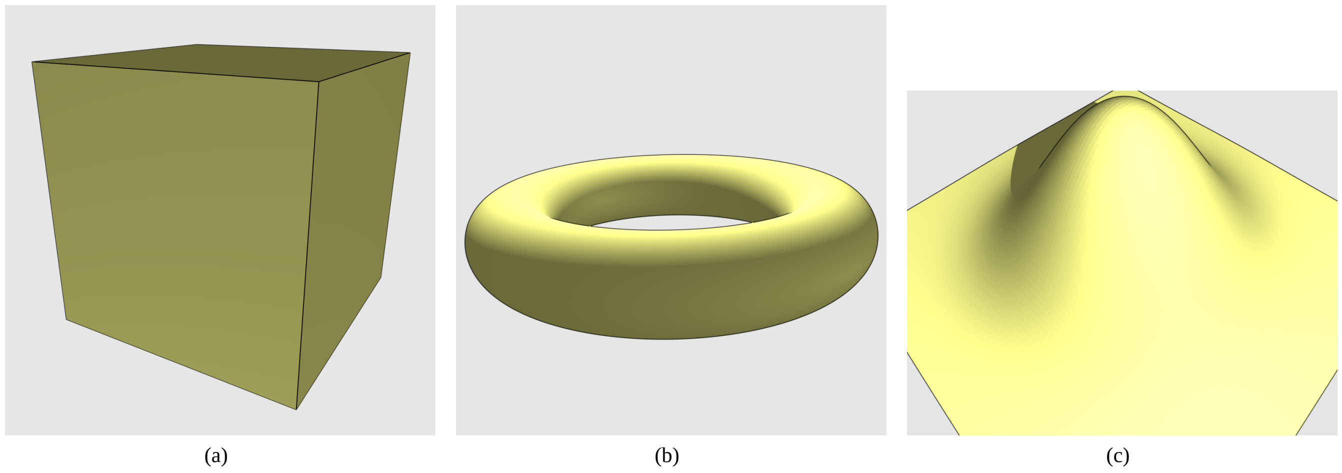


Figure 4: *Examples of crease lines and self-occluding silhouette edges. (a) The shading on the cube faces hints at a discontinuous normal field; the crease lines highlight the location of the discontinuity. (b) The shape of a torus is more strongly expressed when the self-occluding silhouettes extending from the central hollow are shown. (c) A gaussian function is rendered with a self-occluding silhouette, emphasizing the bump.*

the depth image (i.e. sudden changes in the normal direction are accompanied by sudden changes in the rate of change of depth values with respect to some viewing direction).

For each type of feature line our algorithm detects, the action of the ray stencil can be explained in terms of searching for zeroth, first, or second order discontinuities in one function or another, along the same lines as discussed in, e.g., Saito et al. [Saito and Takahashi 1990]. Our method demonstrates that general image filtering techniques are possible, fully within the ray tracing framework.

4.3 Anti-Aliasing

One particular advantage of our method is that the lines drawn are naturally anti-aliased. This arises from the nature of the area estimation performed by the ray stencils. Ideally, as a ray stencil moves across a line-like feature in image space, the foreign geometry ID area increases from zero to fifty percent, and then falls back to zero again. The edge strength metric derived from these values, when used to determine darkness, produces a line that is dark in the middle and smoothly lightens toward white at distances equal to the radius of the stencil. By increasing the quality parameter N in the stencil construction, the area estimation becomes finer and more levels of smoothness can be used at the cost of tracing more stencil rays. Bærentzen et al. [Bærentzen et al. 2006] achieve a similar effect by using smoothly varying “edge” functions in their fragment shader.

If lines with different qualities are needed, different edge strength metrics can create different kinds of lines. For instance, exponentiating the second term of Equation 1 will produce an edge strength metric with a faster transition from dark to light, giving sharper edged lines.

The stencil rays can also be used for scene anti-aliasing. Because the stencil rays are used for essentially an image processing task that depends on visibility and depth, they are intersected with scene geometry but not shaded. However, traversing and intersecting rays with the scene is the dominant cost in tracing them; for a small extra cost the stencil rays can be shaded and used for multisampling, in addition to computing feature lines.

5 Results

The technique presented in this paper opens ray tracing to new styles of rendering. In this section, we review several examples of the technique.

5.1 Primitive joints

Figure 1 shows a space-filling model [Corey and Pauling 1953] of an acetaminophen molecule, with NPR lines shown by themselves in the middle panel. In the space-filling model, spheres representing atoms always intersect to show atomic bonds; marking the lines along which the atoms intersect can make the structure subtly more apparent, aiding in the understanding of such images. Techniques for approximating global illumination, such as ambient occlusion, have been shown to be useful for certain visualization applications, including particle [Gribble and Parker 2006] and molecular [Tarini et al. 2006] visualization. These methods are especially helpful in understanding subtle three-dimensional placement. One of the effects of using ambient occlusion for a molecular model is to *darken* the atomic joints; in this case, drawing the intersection lines serves as a non-photorealistic way to indicate the darkened regions, evoking some of the core effect of the ambient occlusion renderings. Rendering intersection lines directly generalizes the darkening effect by drawing lines even for primitives that intersect at shallow angles, in cases where ambient occlusion would *not* significantly darken the joint between them.

5.2 Mesh visualization

Triangle meshes are a standard and widespread way to represent three-dimensional geometry; consequently, much work in the ray tracing community aims toward improving rendering performance for such meshes [Ize et al. 10-12 Sept. 2007; Wald et al. 2006; Wald et al. 2008]. In a raster graphics setting, wireframe techniques can reveal the mesh structure by showing the triangle joints. This can be useful for many purposes, such as debugging mesh model geometry or evaluating their quality during design and construction.

When ray tracing a mesh model (Figure 5), our technique offers two choices for the stencil ray geometry IDs: either the ray can as-

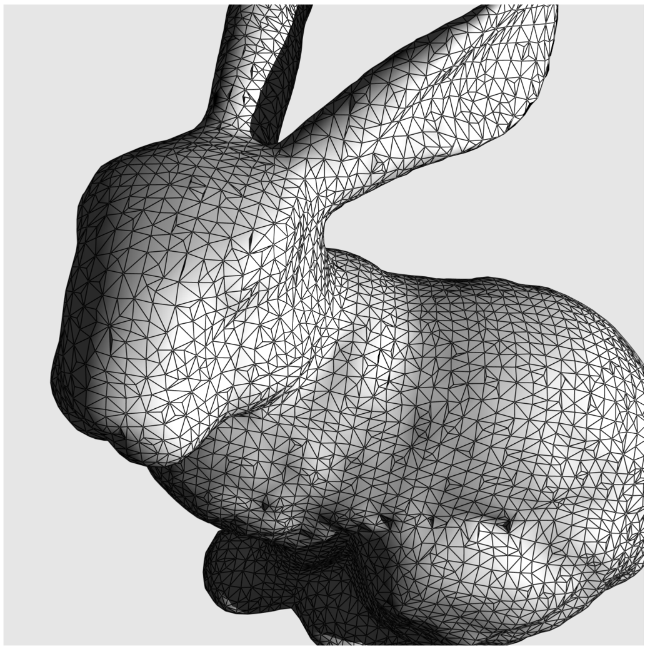


Figure 5: *Our method reveals intersection lines between the triangles in the bunny model, allowing for the visualization of meshes.*

sociate with the object representing the whole object, or with the individual triangle primitive within the model. In the former case, the technique treats the mesh as a single, whole object, and it will show the silhouette of the meshed object.[2] In the latter case, however, the joints between the triangles will also be drawn, revealing the mesh structure and connectivity.

5.3 Particle data sets

Bigler et al. [Bigler et al. 2006; Bigler 2004] have demonstrated the usefulness of advanced shading models and NPR techniques in examining particle data sets produced by the Material Point Method (MPM) [Bardenhagen and Kober 2004; Sulsky et al. 1995]. The data sets are usually visualized using a glyph to represent each particle; important insights come from understanding how the particles are arranged and how their arrangement changes over simulated time.

As discussed by Bigler, silhouette edges can act as a cue to structure. The authors of this paper have explored the use of different glyph geometries for visualizing MPM data [Choudhury et al.]. By using the technique described in this paper, we highlight not only silhouette edges but also places where particles intersect (Figure 6, left), while crease edges enhance the individual hexahedral glyph shapes (Figure 6, right). Seeing where particles overlap each other is especially important for MPM data, as it usually indicates error or instability in the simulation. As with the molecule rendering (Figure 1), the NPR lines elaborate the primitives' physical relationship to each other, which is very important to understand in a setting like scientific visualization.

5.4 Other NPR Techniques

To demonstrate that our framework of ray stencils is useful for other NPR algorithms as well, we have implemented *apparent ridges* according to the paper by Judd et al. [Judd et al. 2007]. Figure 7 shows a "rounded cube" (actually a superellipsoid) on which apparent ridges, which are the loci of points at which the "view dependent curvature" is locally maximal, can be seen.

[2]If the normals are interpolated across the triangle faces for shading purposes, then the technique will not pick up sharp corners at the triangle edges.

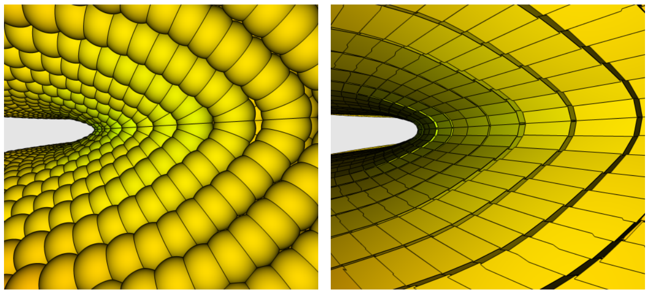


Figure 6: *A particle data set rendered using spheres (left) and hexahedral elements (right). In both images, our NPR technique is used to render intersection lines, silhouette edges, and crease edges. The intersection lines make very clear the relative positions of the glyphs. In particular, the spheres are seen to overlap very often in the left image, while slight intersection between the hexahedral glyphs can be seen where the boundary lines appear kinked.*

The view dependent curvature is described by a rectangular matrix Q that depends both on an object's surface curvature, and the viewing projection. It transforms vectors in screen space to normal perturbation vectors in world space: in terms of ray stencils, this means measuring the normal gradient in screen space is sufficient to construct Q. The singular value decomposition of Q yields the view dependent curvature for each sample ray, which in turn can be used to render the apparent ridges.

This example demonstrates that the ray tracing framework, together with ray stencils, is flexible enough to allow for "porting" NPR techniques from a raster graphics setting to a ray tracer. Given the large number of techniques discussed in Section 2, the apparent ridges example shows that it is possible to use NPR methods within ray tracers.

6 Conclusions and Future Work

We have presented a method for computing and rendering feature lines, using only the machinery of a ray tracing engine. The method searches for indications of such feature lines in the screen space vicinity of a sample ray, allowing for the ray tracer to render constant width lines, emulating methods that already exist in the raster graphics literature. The technique is useful for drawing attention to specific geometric features, such as the relative placement and grouping of primitives, which can promote better comprehension of certain images.

One major area of future work for our method lies in searching for optimizations. Currently, we trace stencil rays independently; however, because stencil rays are, by definition, more likely to be coherent, there should be ways to leverage that coherence to compute their scene intersections more quickly. In addition, because of the stencil rays' geometric regularity, there may be opportunities for re-using results from previous instantiations of ray stencils as well.

Our method is particularly useful for scientific visualization, in which ray tracing has been increasingly used to handle the sheer number of primitives that can arise from modern scientific simulations. NPR-style effects in scientific visualizations are useful because of the way they draw the eye to certain features that can pro-

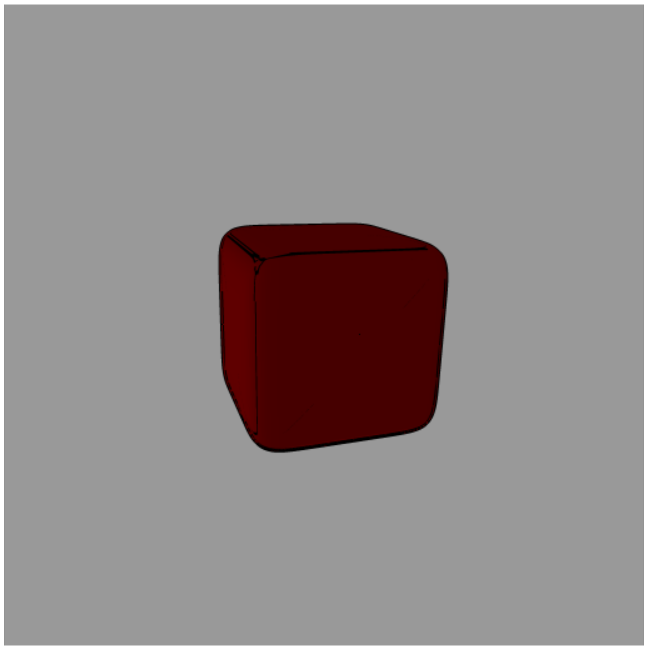


Figure 7: *A rounded cube overlaid with apparent ridges, as computed by a ray tracer using ray stencils.*

vide insight into the data. More generally, in any situation where ray tracing is useful for one reason or another, but NPR effects are also useful, our technique brings the two technologies together.

References

BÆRENTZEN, A., NIELSEN, S. L., GJØL, M., LARSEN, B. D., AND CHRISTENSEN, N. J. 2006. Single-pass wireframe rendering. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, ACM, New York, NY, USA, 149.

BARDENHAGEN, S. G., AND KOBER, E. M. 2004. The generalized interpolation material point method. *Computer Modeling in Engineering and Sciences 5*, 6, 477–496.

BIGLER, J., GUILKEY, J., GRIBBLE, C. P., HANSEN, C. D., AND PARKER, S. G. 2006. A case study: Visualizing material point method data. In *Proceedings of Euro Vis 2006*, 299–306, 377.

BIGLER, J. L. 2004. *Use of Silhouette Edges and Ambient Occlusion in Particle Visualization*. Master's thesis, University of Utah.

BRESENHAM, J. E. 1965. Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1, 25–30.

CHOUDHURY, A. I., GUILKEY, J. E., AND PARKER, S. G. Physically based deformation visualization for particle datasets. Unpublished.

COREY, R. B., AND PAULING, L. 1953. Molecular models of amino acids, peptides, and proteins. *The Review of Scientific Instruments 24*, 8 (August), 621–627.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3, 848–855.

DOOLEY, D., AND COHEN, M. F. 1990. Automatic illustration of 3d geometric models: lines. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 77–82.

ERNST, M., AND GREINER, G. 10-12 Sept. 2007. Early split clipping for bounding volume hierarchies. *Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on*, 73–78.

GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 447–452.

GRIBBLE, C. P., AND PARKER, S. G. 2006. Enhancing interactive particle visualization with advanced shading models. In *APGV '06: Proceedings of the 3rd symposium on Applied perception in gr aphics and visualization*, ACM Press, New York, NY, USA, 111–118.

IGEHY, H. 1999. Tracing ray differentials. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 179–186.

IZE, T., SHIRLEY, P., AND PARKER, S. 10-12 Sept. 2007. Grid creation strategies for efficient ray tracing. *Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on*, 27–32.

IZE, T., WALD, I., ROBERTSON, C., AND PARKER, S. Sept. 2006. An evaluation of parallel grid construction for ray tracing dynamic scenes. *Interactive Ray Tracing 2006, IEEE Symposium on*, 47–55.

JUDD, T., DURAND, F., AND ADELSON, E. H. 2007. Apparent ridges for line drawing. *ACM Trans. Graph. 26*, 3, 19.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, 609–612.

POTTER, K., GOOCH, A., GOOCH, B., WILLEMSEN, P., KNISS, J., RIESENFELD, R., AND SHIRLEY, P. 2009. Resolution independent npr-style 3d line textures. *Computer Graphics Forum 28*, 1, 52–62.

RASKAR, R., AND COHEN, M. 1999. Image precision silhouette edges. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, 135–140.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph. 24*, 4, 197–206.

SMITH, S. M., AND BRADY, J. M. 1997. Susan—a new approach to low level image processing. *International Journal of Computer Vision 23*, 1 (May), 45–78.

STROTHOTTE, T., PREIM, B., RAAB, A., SCHUMANN, J., AND FORSEY, D. R. 1994. How to render frames and influence people. *Computer Graphics Forum 13*, 3, 455–466.

SULSKY, D., ZHOU, S.-J., AND SCHREYER, H. L. 1995. Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications 87*, 1–2 (May), 236–252.

TARINI, M., CIGNONI, P., AND MONTANI, C. 2006. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5, 1237–1244.

WALD, I., IZE, T., KENSLER, A., KNOLL, A., AND PARKER, S. G. 2006. Ray tracing animated scenes using coherent grid traversal. *ACM Trans. Graph. 25*, 3, 485–493.

WALD, I., IZE, T., AND PARKER, S. G. 2008. Fast, parallel, and asynchronous construction of bvhs for ray tracing animated scenes. *Computers and Graphics 32*, 1 (February), 3–13.