Rendering Anti-Aliased Line Segments

Keen-Hon Wong[‡], Xin Ouyang[†], Tiow-Seng Tan[†], Jürg Nievergelt[§], Chi-Wan Lim[†]

School of Computing, National University of Singapore

Abstract

Bridging the modelling and rendering gap between the existing triangle and point primitives, we explore the use of line segments as a new primitive to represent and render 3D models. Our main contribution extends the anti-aliasing theory in texture mapping to anti-aliased line segment rendering, and presents an approximation algorithm to render high quality anti-aliased opaque and transparent line segments in 3D models. This anti-aliasing technique is empirically validated by building a software pipeline to render models of any combination of the three types of surface primitives: triangles, line segments and points. Experiments show that models comprising line segments are potentially efficient and effective for high quality rendering as compared to their corresponding pure point models.

Additional Keywords: Rendering Systems, Point-based Graphics, Surface Reconstruction, Object Scanning/Acquisition, Level of Details.

1. INTRODUCTION

Rendering, the final link in the chain that creates an image of a real or conceived object, is based on a model of the object to be depicted. Objects may be represented by different models for different purposes. For the purpose of rendering, a model needs no more details than necessary to generate an image of desired resolution and quality. Often, a compact representation that permits efficient processing is equally, if not more, important than details. It is therefore interesting to study different models that adapt to different requirements.

Historically, surface triangulation was the predominant model permeating all or most of the stages of the graphics pipeline. Recently, other models have become prominent, such as point based methods [GrDa98, PZVG00, RuLe00, ZPVG01] and image based rendering, that emphasize ease of rendering at the cost of neglecting surface geometry and topology. This paper aims to contribute in the search for line segment models that are tuned to the needs of rapid rendering.

Aliasing problem in computer graphics is caused by the disparity of image representation in continuous real world and discrete computer world. Images are represented as continuous signals in the world space, and discrete signals in the screen space. The screen images are created by sampling and quantizing their corresponding world space images. If the sampling frequency is less than the Nyquist frequency, high frequency portions of the continuous signals will masquerade as low frequencies, creating aliasing effects.

Various anti-aliasing techniques have since been proposed to use low pass filter to stop high frequencies before sampling, so as to render anti-aliased high quality texture mapped triangle models. Recently the Elliptical Weighted Average (EWA) filter [Heck89] has been extended for rendering anti-aliased high quality point models [ZPVG01, RPZ02]. This paper is a contribution to the use

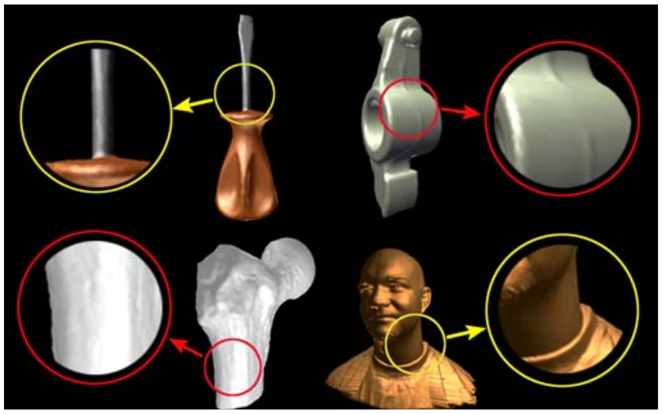


Figure 1: *Hybrid point and line segment models rendered by our method. Pictures in the leftmost and rightmost columns are the zoomed in views of circled regions on pictures in the middle columns.*

of anti-aliasing techniques for high quality rendering of surface primitives.

It is interesting to reflect on the fact that the most prominent rendering primitives, the point and the triangle, are simplexes in 0 and in 2 dimensions, whereas the 1-simplex, i.e. the line segment, has been used as a primitive only in special contexts, such as [DCSD02, LoTa97]. This is surprising in view of the fact that lines are prominent features in many types of scenes and applications, in particular of man-built objects; see the Screwdriver and Rocker arm models in Figure 1. But even objects of irregular shape contain features that are best modelled by lines, such as ridges or boundaries of various kinds, see the Ball joint and Upper body models in Figure 1. Figure 2 illustrates a knee model represented by triangles, points as well as hybrid points and line segments.

It appears wasteful to ignore lines, a feature that appears naturally in many scenes and readily translates into a simple primitive, the line segment. Line segments should be treated as a primitive, rather than as a degenerate special case of triangles or polygons, for the same reason that points are treated as a primitive rather than as tiny triangle or polygons. A line segment is a simpler object than a long, thin triangle or rectangle. Three advantages come with using line segments as a primitive in surface modeling and rendering.

First, line segments together with points and triangles provide an effective representation of object surfaces. Triangles are suitable for uses in surface areas of flat regions, line segments for cylindrical and conical regions, and points for jagged regions. Second, the use of line segments facilitates compact representations of models with regularity along one dimension. In comparison with triangle meshes, line segments save storage space by keeping less connectivity information among vertices. In comparison with point clouds, few line segments can replace a large amount of sequentially aligned points. Finally, line segments could be rendered much more efficiently than sequences of points. Experiments in section 5 show that a maximum 63.32% speedup can be achieved as compared to rendering pure point models.

‡Email: keenhon@crimsonlogic.com
†Emails: {ouyangxin, tants, limchiwa}@comp.nus.edu.sg
§Email: nivergelt@inf.ethz.ch

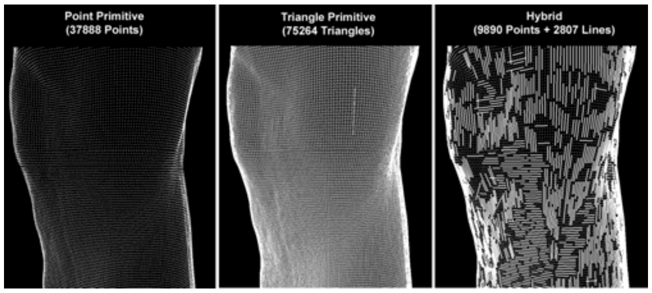


Figure 2: *A knee in three different representations. The hybrid point and line segment model is obtained using our primitive extraction algorithm for point sets.*

In this paper, we focus on studying the consequences of using line segments as a rendering primitive, design corresponding algorithms, report on experiments that show the validity of this approach, and describe research directions that promise to make line based rendering a standard technique in the growing arsenal of graphics tools.

Our main contribution extends the anti-aliasing theory in texture mapping [Heck89] to render anti-aliased line segments in 3D models. Though the extension does not result in a closed form solution, we present an approximation to render high quality anti-aliased opaque and transparent line segments representing 3D models as shown in Figure 1, Figure 9 and Figure 10. The empirical validation of this approximation involves building an experimental set-up consisting of two main components.

First, we implement two methods to extract primitives from triangle meshes and point sets respectively. For a surface given by a triangle mesh, we use contour planes to intersect with triangles to extract line segment primitives. For a surface given by a set of points in 3D space, we use a surface local feature based adaptive method to extract all the three point, line segment and triangle primitives. Both of the two primitive extraction algorithms are developed to mainly serve for the purpose of assessing quality and performance of our proposed anti-aliased line segment rendering method. Designing more efficient primitive extraction algorithm is left as part of our future work.

Second, we implement a software graphics pipeline to render 3D models represented with any combinations of points, line segments, and triangles. Our experiments show that the rendered quality of hybrid models is comparable to their corresponding high quality anti-aliased point models. Additionally, there is a significant speed up in the rendering time using hybrid models. This establishes hybrid of line segments and points as a competitive representation and rendering alternative to pure point models.

Here is an overview of the paper. Section 2 summarizes related work. Section 3, the main contribution in this paper, presents an anti-aliasing theory for rendering line segments and an approximation used in our implementation. Section 4 describes a pipeline to render point, line segment and triangle primitives. Section 5 reports on two sets of experiments to measure the rendering quality, efficiency and effectiveness of hybrid models as compared to point models. Section 6 concludes the paper with our ongoing work.

2. RELATED WORK

Point Based Rendering. In 1984, Levoy and Whitted [LeWh85] pointed out that classic modelling primitives, i.e. triangles (or polygons), were less appealing for rendering objects with extremely complex geometry. They suggested decoupling modelling geometry from the rendering process by introducing point as universal primitive, where each point is associated with a small surface area and a normal. Using this idea, Rusinkiewicz and Levoy [RuLe00] proposed the QSplat system to render 3D point models. The QSplat system was designed during the course of the Digital Michelangelo Project to render hundred-million-triangle models at interactive frame rates. In 1998, Grossman and Dally [GrDa98] developed the point sample rendering algorithm for real-time, high quality rendering of complex objects. Sample points are rendered without any knowledge of surface topology.

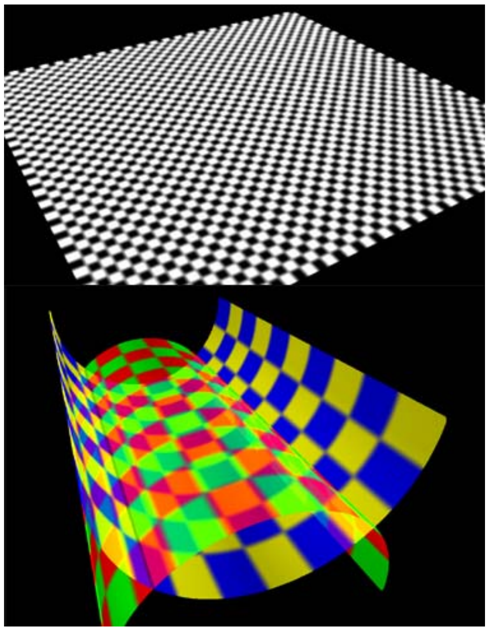


Figure 3: *Opaque* (*top*) *and transparent* (*bottom*) *anti-aliased checkerboard line segment models. Each line segment is painted with only one color and line segments of different colors are separately laid in checker boxes.*

In 2000, Pfister *et al.* [PZVG00] introduced the surfel, i.e. surface element, paradigm and developed the surfel point rendering pipeline. In 2001, Zwicker *et al.* [ZPVG01] extended Heckbert's EWA filter [Heck89], deriving a rigorous mathematical formulation of screen space EWA texture filtering for irregular point data. Based on the newly derived resampling filter, they developed another point rendering technique called surface splatting. Surface splatting is able to produce high quality texture filtered images from point samples. In 2002, Ren *et al.* [RPZ02] proposed a hardware implementation of surface splatting by building a visibility map to speed up point visibility testing; however, it does not handle transparency. Our work on rendering line segments follows in the direction of [PZVG00, ZPVG01, RPZ02]. We extend the anti-aliasing theory in texture mapping [Heck89] to anti-alias line segments contained in 3D models.

Line Segment Rendering. The idea of using line segments instead of points as a rendering primitive appears to have originated for the purpose of rendering features of plants, which naturally exhibit long, thin line features, such as twigs [WePe95]. Deussen *et al.* [DCSD02] render plant images using a mix of triangles, line segments and points, preferring triangles when high quality is required, line segments or points when details can be neglected. In contrast to their technique of using line segments as auxiliary primitives to speed up rendering of low level-of-detail sub-images, the approach presented in this paper treats line segments as first class primitives to be used for high quality rendering. Thus, the development of a technique for anti-aliasing line segments is a major contribution of this paper.

Anti-aliased Lines. Many papers treat the 2D problem of anti-aliasing lines on screen. A good reference on the desirable characteristics for an anti-aliased line is described by Nelson [Nels96]. In contrast, we anti-alias 3D lines where each line has an associated surface area and a normal.

Hybrid Rendering. Chen and Nguyen [ChNg01] introduced a hybrid point and polygon rendering system called POP, in which a point is represented by a bounding sphere as in the QSplat system [RuLe00]. Dey and Hudson used a screen pixel as a point in another hybrid system PMR [DeHu02]. In our hybrid rendering solution, a point is an anti-aliased splat suggested by Zwicker *et al.* [ZPVG01]; points and line segments can be seamlessly hybridized together, producing high quality images. Both POP and PMR build hierarchical LOD structures for models. During rendering, the LOD control decides in what circumstances triangles should be substituted by points to maximize the frame rate. The hybridity in both systems have their emphasis on performance. Nevertheless, in our discussion, primitives are chosen based on local features of object surfaces. For examples, parallel line segments are used to approximate a cylinder instead of a bunch of thin triangles. It is beyond the scope of this report to discuss LOD acceleration techniques as done in the above references.

3. ANTI-ALIASED LINE RENDERING

In order to use the line segment (to be abbreviated as “line” in section 3 and 4) as a first-class rendering primitive, we develop a mathematically rigorous anti-aliasing theory and an approximation that admits efficient algorithms. These have been tested in the implementation described in Sections 4 and 5.

3.1 Anti-Aliasing Line

To reconstruct a surface from a set of lines, we define $P_k(t)$ to be a point on a line k with length l_k where $t \in [0, l_k]$. Let Q be a point on the object surface. We define a 2D local coordinate system $\mathbb{S}$ within a small neighborhood around Q (Figure 4). A set L of lines in the neighborhood of Q is parameterized to $\mathbb{S}$. Each point $P_k(t)$ has a mapping to a point $u_k(t) \in \mathbb{S}$. Let r be the reconstruction filter and w_k be the color coefficient of line k. Hence, the color of any point $u \in \mathbb{S}$, as a continuous function $f_c(u)$, can be evaluated by accumulating contributions from lines using r :

$$f_c(u) = \sum_{k \in L} w_k \int_0^{l_k} r(u - u_k(t))\ dt.$$

We define an invertible $\Re^2 \to \Re^2$ mapping $x = m(u)$ from $\mathbb{S}$ to screen space; $h(x)$ as the low-pass filter; and $i(x)$ as the sampling function. Following [Heck89, RPZ02], we have the corresponding equations for the 3 steps of proper anti-aliasing:

1. Warping: The color of the point x in the screen space is
$$g_c(x) = (f_c \circ m^{-1})(x) = f_c(m^{-1}(x)) .$$
2. Band-limit the frequency:
$$g_c^{'}(x) = g_c(x) \otimes h(x) = \int_{\Re^2} g_c(\xi) \cdot h(x - \xi)\ d\xi .$$
3. Sampling: $g(x) = g_c^{'}(x)\ i(x)$.

We expand the above relations in reverse order to obtain:

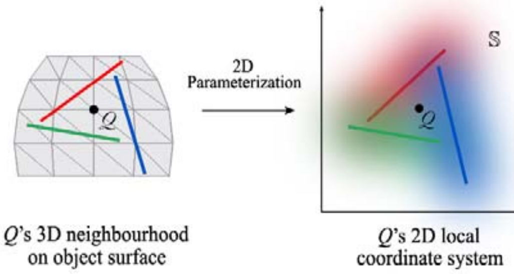

2D
Parameterization
Q
Q
𝕊
Q's 3D neighbourhood on object surface
Q's 2D local coordinate system

Figure 4: *2D parameterization of point Q's neighborhood to the local coordinate system $\mathbb{S}$. The color of Q is evaluated by accumulating contributions from the set L of lines of various colors in $\mathbb{S}$.*

$$g_c^{'}(x) = \sum_{k \in L} w_k \rho_k(x)$$

where $\rho_k(x) = \int_0^{l_k} \int_{\Re^2} r(m^{-1}(\xi) - u_k(t)) \cdot h(x - \xi)\ d\xi\ dt$

is the resampling filter for line k. Substituting the mapping $\xi = m(u)$ into $\rho_k(x)$ yields:

$$\rho_k(x) = \int_0^{l_k} \int_{\Re^2} r(u - u_k(t)) \cdot h(x - m(u)) \cdot \left| \frac{\partial m}{\partial u} \right| du\ dt .$$

Replacing the general mapping $m^{-1}(\xi)$ with its local affine approximation $m_{(k,t)}^{-1}(\xi)$ yields:

$$\rho_k(x) = \int_0^{l_k} \int_{\Re^2} r(u - u_k(t)) \cdot h(x - m_{(k,t)}(u)) \cdot |\ J_{(k,t)}\ |\ du\ dt$$

where $J_{(k,t)} = \dfrac{\partial m_{(k,t)}}{\partial u}$.

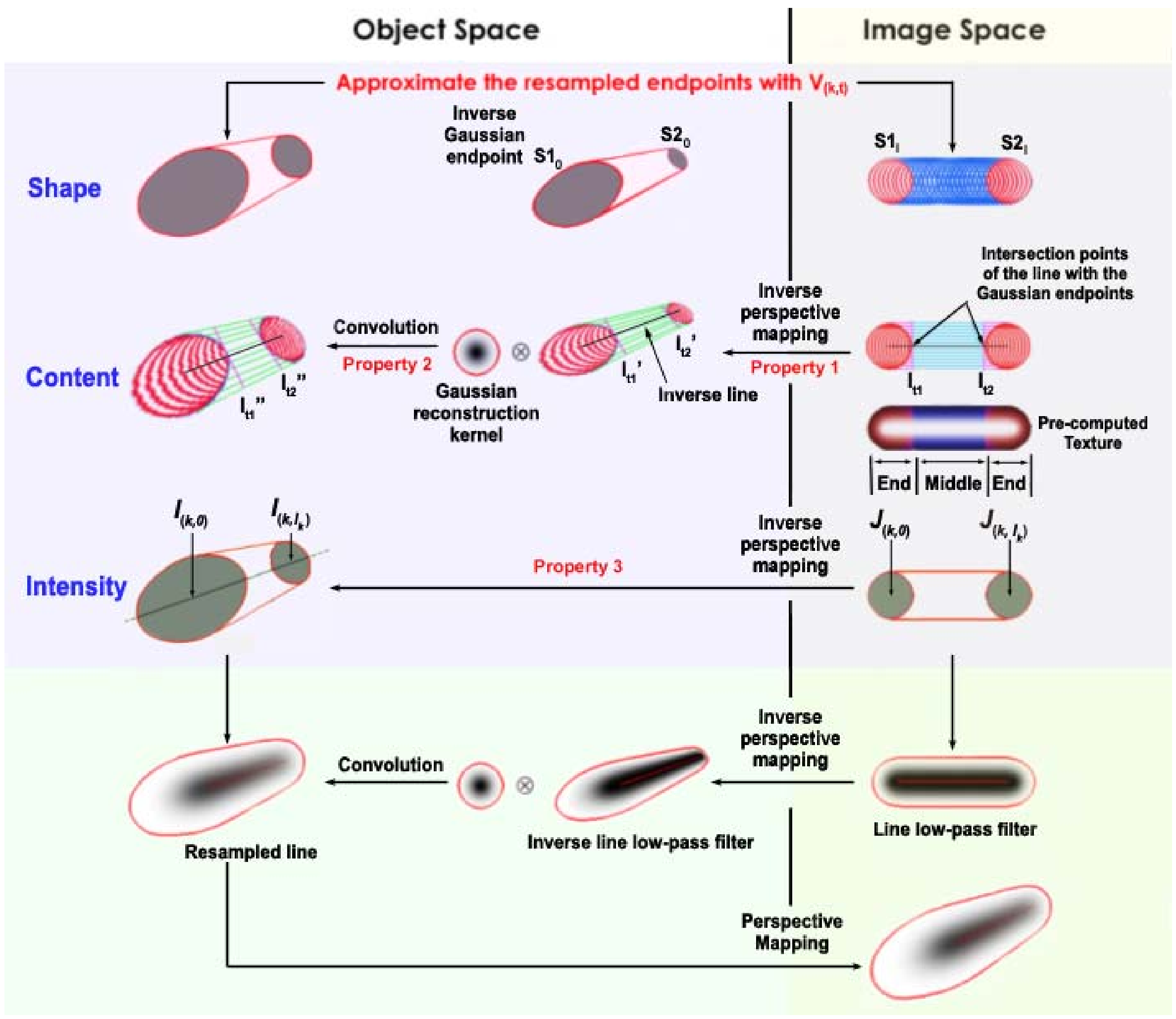
Note that $J_{(k,t)}$ is the Jacobian for the affine mapping at $u_k(t)$. We then use the Gaussian kernels $r(u) = g_{V_r}(u)$ as the reconstruction filter and $h(x) = g_{V_h}(x)$ as the low-pass filter with variances V_r and V_h respectively to get:

$$\begin{aligned}
\rho_k(x) &= \int_0^{l_k} \int_{\Re^2} g_{V_r}(u - u_k(t)) \cdot g_{V_h}(x - m_{(k,t)}(u)) \cdot |\ J_{(k,t)}\ |\ du\ dt \\
&= \int_0^{l_k} \int_{\Re^2} g_{V_r}(u - u_k(t)) \cdot g_{V_h}((m_{(k,t)}^{-1}(x) - u) \cdot J_{(k,t)})\ |\ J_{(k,t)}\ |\ du\ dt \\
&= \int_0^{l_k} \int_{\Re^2} g_{V_r}(u - u_k(t)) \cdot g_{J_{(k,t)}^{-1} V_h J_{(k,t)}^{-1^T}}(m_{(k,t)}^{-1}(x) - u)\ du\ dt \\
&= \int_0^{l_k} g_{V_{(k,t)}}(m_{(k,t)}^{-1}(x) - u_k(t))\ dt \text{ where } V_{(k,t)} = J_{(k,t)}^{-1} V_h J_{(k,t)}^{-1^T} + V_r .
\end{aligned}$$

Note that $V_{(k,t)}$ is the variance of an elliptical Gaussian.

Computing $J_{(k,t)}$. Similar to [RPZ02], we construct a local surface coordinate system for every point $P_k(t)$ on line k by approximating the surface with its tangent plane given by the line normal $\vec{n}_k$. This coordinate system is defined by two orthogonal basis vectors $\vec{u}$ and $\vec{v}$, where $\vec{v}$ is in the direction from the starting point to the ending point of the line, and $\vec{u} = \vec{v} \times \vec{n}_k$.

We note that the object space coordinate of $P_k(t)$ is $\vec{o} + t\ \vec{v}$ where $\vec{o}$ is the position of the starting point of the line in object space. Hence a point $\mu = (s_u, s_v)$ in the local surface coordinate system of $P_k(t)$ corresponds to a point $p^o(\mu) = (\vec{o} + t\ \vec{v}) + s_u\vec{u} + s_v\vec{v}$ in object space. If we assume the mapping from object space to camera space only involves uniform scaling S, rotation R, and



Object Space
Image Space
Approximate the resampled endpoints with $V_{(k,t)}$
Inverse Gaussian endpoint $S1_0$
$S2_0$
$S1_I$
$S2_I$
Shape
Content
Convolution
Property 2
Gaussian reconstruction kernel
Inverse line
Inverse perspective mapping
Property 1
Intersection points of the line with the Gaussian endpoints
Pre-computed Texture
End
Middle
End
Intensity
Property 3
Inverse perspective mapping
Convolution
Inverse perspective mapping
Resampled line
Inverse line low-pass filter
Line low-pass filter
Perspective Mapping

Figure 5: *Approximation of the resampled line.*

translation T, the corresponding point of $p^o(\mu)$ in the camera space is:

$$\begin{aligned} p^c(\mu) &= RS\,p^o(\mu)+T \\ &= RS\,\bar{o}+T+t\,RS\,\bar{v}+s_u RS\,\bar{u}+s_v RS\,\bar{v} \\ &= \tilde{o}+s_u\tilde{u}+(t+s_v)\tilde{v} \end{aligned}$$

where $\tilde{o}=(RS\,\bar{o}+T)$, $\tilde{u}=RS\,\bar{u}$, and $\tilde{v}=RS\,\bar{v}$.

Let $\tilde{o}=(o_x,o_y,o_z)$ be the center, $\tilde{u}=(u_x,u_y,u_z)$ and $\tilde{v}=(v_x,v_y,v_z)$ be the basis vectors defining a local surface coordinate system in camera space. Then the Jacobian $J_{(k,t)}$ at point $u_k(t)$ is:

$$J_{(k,t)}=\frac{\eta}{(o_z+t\,v_z)^2}\begin{bmatrix} u_x o_z-u_z o_x+t(u_x v_z-u_z v_x) & v_x o_z-v_z o_x \\ u_z o_y-u_y o_z+t(u_z v_y-u_y v_z) & v_z o_y-v_y o_z \end{bmatrix}$$

where $\eta=\dfrac{v_h}{2\tan(\frac{fov}{2})}$.

As in [RPZ02], η is a scaling factor determined by the view frustum where v_h and *fov* denote the view port height and field of view of the view frustum respectively.

No Closed Form Solution. To simplify $\rho_k(x)$, we attempt to find a closed form solution since there exists closed form approximation for the integration of Gaussian points along a line, as a *Gaussian line*, using the error function $erf(x)$. However, we arrive at the expression:

$$\rho_k(x)=\int_0^{l_k} c_{(k,t)}\exp^{d_{(k,t)}}dt \quad \text{where } d_{(k,t)}=\frac{q_1(t)}{q_2(t)}$$

in which $q_1(t)$ and $q_2(t)$ are polynomials of degree 6, and $c_{(k,t)}$ is a polynomial in t. In general, an exponential whose exponent is a rational function cannot be integrated in closed form. Nevertheless, we can still implement the line primitive by approximating $\rho_k(x)$ as explained in the next subsection.

3.2 Resampled Line Approximation

Following the theory of anti-aliasing in object space, we would need to first, map the low-pass filter of every point along the line to object space using inverse perspective mapping; second, convolute each of these with the Gaussian reconstruction kernel to obtain a *resampled point*; and finally to sum all to form the *resampled line*. One way to render the resampled line is to use a set of discrete point samples along the line to do an approximation. However the immediate questions are how many points should be used, and how the points should be placed along the line. This is known as footprint assembly problem [CDK04, MPFJ99]. Instead, we propose a more precise as well as mathematically well formulated method which approximates the resampled line based on the two endpoints **S1ₗ** and **S2ₗ** of the line; refer to Figure 5 for the following discussion.

To do that, we first note that the integration of the convolution of each Gaussian point on a line with a Gaussian kernel is equal to the convolution of the corresponding Gaussian line with the Gaussian kernel. Hence we first integrate the low-pass filter for every point along the line to get the *line low-pass filter*. Then we inverse perspective map this filter to object space to get the *inverse line low-pass filter*. The latter is then convoluted with the Gaussian reconstruction kernel to obtain the resampled line, which can then be perspective projected to image space as the resampling filter ρ_k of line *k*.

For practical purposes, we use Gaussian kernels with cutoff radii *r* in the above computation. In addition, we rely on the following properties of perspective mapping and Gaussian convolution:

[Property 1] A convex shape remains convex after the mapping. This also means that a straight line from object space is mapped to a straight line in screen space and vice versa, and a continuous line is mapped to a continuous line in screen space and vice versa.

[Property 2] The convolution of a function having finite influence with the Gaussian kernel having a cutoff radius results in a convoluted function having finite influence. The convoluted function is a scaled and blurred version of the original function.

[Property 3] Using local affine approximation and affine mapping assumptions (together called *local affine assumptions*), a resampled point is approximated by an elliptical Gaussian, which is a scaling and rotation of a unit Gaussian. Its intensity is inversely proportionally to the scaling of the unit Gaussian [RPZ02].

For clarity, we discuss the above approximation of the resampled line in three parts: first, its shape (the outline of the resampled line), then its content (the outcome of the convolution before modulation by the intensity), and finally its intensity (used to modulate the content).

Shape Determination. We adopt the abbreviated process bypassing the calculation of (**S1o, S2o**) to approximate the shapes of the resampled endpoints of the line using $V_{(k,t)}$: the shape of the resampled line is a scaled version of the shape of the inverse line low-pass filter [Property 2]. This means that the shapes of the *resampled endpoints*, i.e. resampled points at endpoints of the line, are scaled versions of (**S1o, S2o**), which are ellipses by the local affine assumptions. With this, we can determine the shape of the resampled line by connecting the shapes of the resampled endpoints with tangent lines [Property 1].

See Figure 5. Let **lt1** and **lt2** be the tangent lines at the intersection points of the line with the Gaussian endpoints. These tangent lines partition the line low-pass filter into three parts: the two end parts (each colored in both red and pink) that contain the Gaussian endpoints, and the remaining middle part (colored in light blue). Let **lt1** and **lt2** map to **lt1'** and **lt2'** respectively due to the inverse perspective mapping, and **lt1'** and **lt2'** correspond to **lt1"** and **lt2"** respectively due to the scaling of the inverse end parts. Then by [Property 1], **lt1'** and **lt2'** are tangent lines at the intersection points of the inverse line with the inverse Gaussian endpoints. Therefore we approximate **lt1"** and **lt2"** to be tangent lines at the intersection points of the inverse line with the resampled endpoints.

Content Determination. Refer to Figure 6. It shows for a line starting at **A** and extending beyond **B',** its resampled line (in blue) between **A'** and **B'** is a Gaussian line that is a scaled and blurred version of the inverse line low-pass filter between **A** and **B** [Property 2]. Note that we use one local affine approximation for the calculation of the content between **A** and **B'** instead of different local affine approximation for every point on this part of the line. With this resampled line, the content of the resampled end part is a scaled version of the content of the inverse perspective mapping of the end part (with the local affine assumptions). And the content of the resampled middle part between **lt1"** and **lt2"** is approximated by a scaled version of the content of the inverse perspective mapping of the middle part, which connects both resampled end parts to create a continuous content. We can texture the content of the resampled line using the influence texture of a Gaussian line with unit variance.

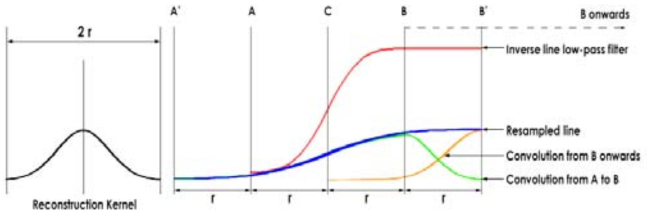


Figure 6: *Convolution of the inverse line low-pass filter* (*red curve from* **A** *to* **B'**) *with the Gaussian reconstruction kernel* (*black curve with cutoff r*) *shown in 2D. The result is a line Gaussian between* **A'** *and* **B'** (*blue curve*)*, which is the sum of the convolution from* **A** *to* **B** (*green curve*) *plus the convolution from* **B** *onwards* (*orange curve*).

Intensity Determination. By determining the Jacobians $J_{(k,0)}$ and $J_{(k,l_k)}$ at both endpoints of line *k*, we can calculate the intensities $I_{(k,0)}$ and $I_{(k,l_k)}$ at both resampled endpoints [Property 3]. Every point on the line has different intensity; we approximate the intensity of these points by linearly interpolating between $I_{(k,0)}$ and $I_{(k,l_k)}$. The content is modulated by the intensity. This approximation approach is in line with the calculation of the content of the resampled line.

We note that the accuracy of the above approximation depends greatly on the correct and accurate computations of shapes of the endpoint ellipses. A bad case is thus when the local affine assumptions for the two endpoints do not hold well. This happens when $\vec{v}$ (the direction from the starting point to the ending point) of the line is nearly parallel to the eye viewing direction. In such a case, the error caused by the deviation of the affine transformation from the perspective projection increases dramatically, and the shape of line cannot be approximated well and consequently its content is textured with a lot of mapping errors.

4. RENDERING

Figure 7 shows our rendering pipeline for models represented by points, line segments and triangles. This pipeline supports high quality anti-aliased point and line segment rendering, while triangles are rendered without anti-aliasing. Comparing with standard rendering pipeline, our rendering pipeline differs in two aspects. First, two extra stages, parameterization and surface transformation, are added after the view transformation. The parameterization stage computes the Jacobians and ellipse equations. These values are then passed to the surface transformation stage to compute the vertices for mapping influence textures. Second, an extra visibility and blending stage is added after the scan conversion stage. We choose to use the

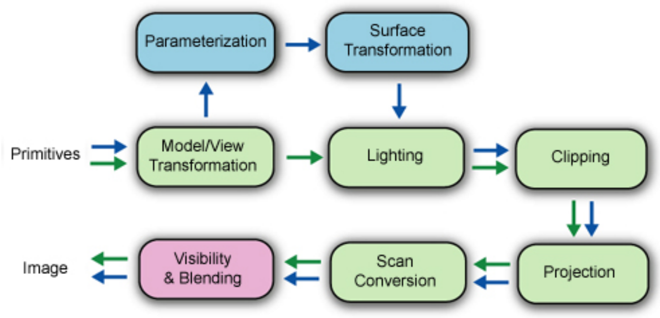


Figure 7: *The rendering pipeline for hybrid models. The Rendering of triangles follows the green arrows while the rendering of line segments and points follows the blue arrows.*

modified Z^3 algorithm [JoCh99] as we need to blend resampled lines together and to handle transparency. Since existing graphics hardware does not support this stage, we implemented the whole rendering pipeline in software. We note that the performance of the pipeline can be improved with parallel processing support, as well as line stripping and fanning support similar to the case of triangle stripping and fanning.

Line Rendering. In Section 3.1, we derive a way to compute the Jacobian $J_{(k,t)}$ at any point on a line. Using $J_{(k,t)}$, we calculate $V_{(k,t)}$, which determines an elliptical Gaussian. By substituting t with 0 and l_k, we are able to approximate the shape of the resampled endpoints of the line with ellipses. Using standard algebra [Stot98], we derive 18 vertices (in line with Section 3.2 on the shape determination) in two phases to texture the content of a line primitive (Figure 8):

[*Phase 1*: *Deriving vertices* 1 *to* 8]. Vertices 1 to 4 are vertices on the tangent lines touching the border of the ellipses. Vertices 5 to 8 are the intersection points of the ellipses with the line (extended to intersect each ellipse at two vertices).

[*Phase 2*: *Deriving vertices* 9 *to* 18]. Vertices 9 and 10 are intersection points of line $\overline{5,8}$ with the lines $\overline{1,4}$ and $\overline{2,3}$ respectively. The last 8 vertices are intersection points of the tangent lines at vertices 5 to 8 with the two lines passing through $\overline{1,2}$ and $\overline{3,4}$.

See [Stot98] on the details how to compute the tangent points of the endpoint ellipses. A short discussion is also included in the Appendix. We use an end and a middle texture sliced from a pre-computed influence texture obtained by integrating unit Gaussians, of cutoff radius r, from 0 to l for some $l > 2r$, to texture lines' contents (as mentioned in Section 3.2 on the content determination). Let

$$T_1(x,y) = \tfrac{1}{2\pi}\sqrt{\tfrac{\pi}{2}}e^{-\frac{1}{2}y^2}\left(erf\left(\sqrt{\tfrac{1}{2}(r^2-y^2)}\right) + erf\left(\tfrac{1}{\sqrt{2}}x\right)\right) \text{, and}$$

$$T_2(x,y) = \tfrac{1}{\pi}\sqrt{\tfrac{\pi}{2}}e^{-\frac{1}{2}y^2}\left(erf\left(\sqrt{\tfrac{1}{2}(r^2-y^2)}\right)\right).$$

Here, x is defined as the coordinate along the line, while y is in the direction perpendicular to the line. *erf* denotes the error function. The result of the integration of unit Gaussians from $[0,r]$ is the end texture, given by T_1 when $x^2 + y^2 \le r^2$, and by T_2 when $x^2 + y^2 > r^2$, $0 < x < r$ and $-r < y < r$; while the result of the integration of unit Gaussians from $(r, l - r)$ is the middle texture, given by T_2 when $r < x < l - r$ and $-r < y < r$.

Note that each end part of a line is partitioned with center at vertex 9 (or 10) into four for texturing content. This aligns the center of the end texture with vertex 9 (or 10) to achieve a better texturing for the endpoint ellipse. The less expensive alternative of texturing the whole end texture in one step is sufficient as we learn from our experiments, though undesirable skewing of a line may appear occasionally when the line is very close to the eye.

The content of the line is then linearly modulated by the intensity, as mentioned in Section 3.2 on the intensity determination. Finally the resampled line is multiplied with the color coefficient w_k and blended into the rendering buffers, including color buffers, z buffers and accumulation buffers.

Handling of Short Lines. Short lines are lines whose resampled endpoints overlap each other. A line may not be short, but may become a short line in the image due to projection. Similar to the usual line case, we derive 15 points for texturing the content of a short line (Figure 8). Points 6, 12, and 17 are derived by intersecting the line connecting the intersection points of the ellipses with lines $\overline{5,8}$, $\overline{1,2}$ and $\overline{3,4}$ respectively. Each short line is textured using only the end texture in our approximation. We note that a better approximation would require the computation of an influence texture that depends on the length of the short line. This means either storing more pre-computed influence textures (besides the end and the middle texture computed for the usual line) or computing needed influence textures on-the-fly. This is however unnecessary as our experiments show that our simple approximation using only the end texture is generally sufficient to obtain good quality anti-aliased rendering results.

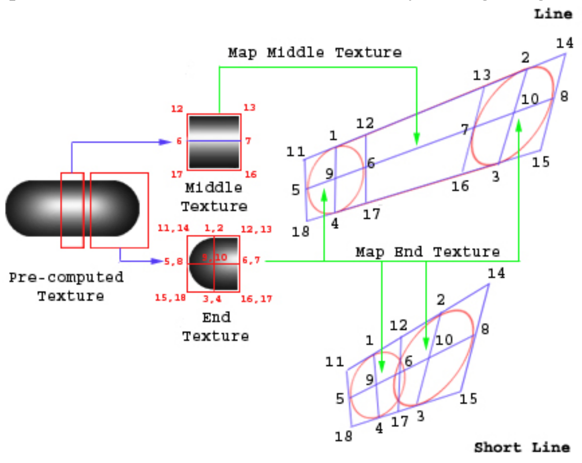


Figure 8: *Texturing line contents with influence textures.*

5. EXPERIMENTS AND RESULTS

In the course of this project, we have done two experiments to assess our proposed anti-aliased line segment rendering method. In the first experiment, contour planes are used to extract the line segment primitive from triangle meshes. In the second experiment, a surface local feature based adaptive method is implemented to extract all the three point, line segment and triangle primitives from point sets. These two primitive extraction algorithms are implemented only as a quick way to experiment with our rendering pipeline. For simplicity, we term hybrid point and line segment models as *PL hybrids*, and hybrid point, line segment and triangle models as *PLT hybrids*.

We implemented the rendering pipeline described in Section 4 in C/C++, and performed the experiments on a P4 1.8 GHz with 512 MB RAM PC. Note that the rendering results of point models are also from our own implementation of the theory given in [RPZ02, ZPVG01] rather than the probably more optimized version of [ZPKG02]. Accompanying videos showing rendering results of

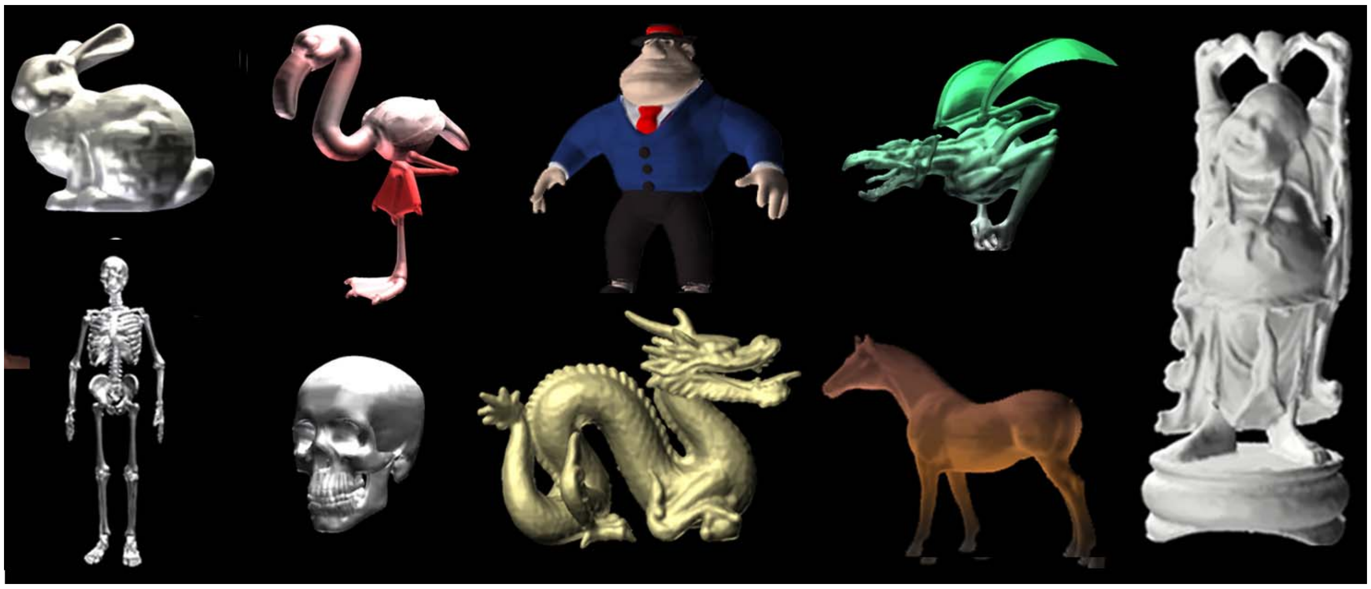


Figure 9: *Line segment models used in experiment one rendered by our method.*

models used in this paper are available from the project webpage *http://www.comp.nus.edu.sg/~tants/line.html*.

Experiment One. In this experiment, we render each model in Table 1 at 20 different viewpoints chosen around the model without any priori knowledge. Altogether 180 images of point models and 180 images of line segment models are used. These images are of size 512x512 pixels.

Models	Number of Primitives		Image Differences	
	Lines	Points	Mean	Standard Deviation
Bunny	26834	128642	0.13	2.91
Flamingo	27876	126938	0.12	2.70
Al	46595	163205	0.09	2.17
Gargoyle	46106	146704	0.09	1.88
Dragon	78305	241893	0.19	3.62
Buddha	115883	338709	0.31	4.45
Horse	26518	76601	0.02	1.56
Skull	57247	156434	0.04	1.83
Skeleton	87838	221856	0.17	3.89
		Average	0.13	2.78
		Maximum	0.31	4.45

Table 1: *Number of primitives and image differences.*

The rendering quality is evaluated both visually and numerically. Figure 9 shows rendering outcome of line segment models in Table 1. We compute the mean of the absolute (pixel) differences between two images rendered using line segments and points. These differences are calculated from a composite of red, green and blue channels (each channel has 256 values). Table 1 shows the average of the means from the 20 different viewpoints per model and the average of the standard deviations. We find that the means of the absolute differences are nearly zero with an average of 0.13 and the average standard deviations of 2.78 (second last row in the table). The maximum of such mean is 0.31 and the maximum standard deviation is 4.45 (last row in the table). The numerical results suggest that the rendered line segment and point models have nearly the same quality.

Experiment Two. In this experiment we compare the quality and performance of rendering each model in Table 2 using (i) purely points, (ii) a PL hybrid, and (iii) a PLT hybrid.

As in experiment one, the image quality of the models used is also evaluated both visually and numerically. Figure 1 and Figure 10 show the rendering outcome of PL hybrids from Table 2. Table 3 is obtained by rendering color images (each channel has 256 values) of size 512x512 pixels for 50 different viewpoints chosen around each of the model without any priori knowledge. Between the corresponding images of pure point and the two different hybrids, the table shows the mean square error (*mse*, measuring the difference) and the normalized cross-correlation measure (*nccm*, measuring the similarity with 1 means identical image) [ASS02]. These numerical results again suggest that the rendered hybrid models and their corresponding point models indeed have nearly the same quality.

Models	Points	PL Hybrid		PLT Hybrid		
	# points	# points	# lines	# points	# lines	# triangles
Armadillo	172974	155264	2336	155264	2336	0
Ball joint	137062	89915	5563	90112	5551	135
Golf club	209779	39138	12903	53053	11519	7190
Igea artifact	134345	75517	6450	75536	6444	41
Isis	187644	89650	10229	86876	10207	180
Male	303380	115329	17651	116304	17572	513
Upper body	148138	88709	6414	88591	6144	57
Rabbit	67038	46045	2533	46045	2533	0
Rocker arm	40177	26188	1454	26830	1336	273
Santa	75781	62601	1696	62601	1696	0
Screwdriver	27152	17458	1002	17604	972	86
Teeth casting	116604	62799	4104	67785	3779	2419

Table 2: *Hybrid models obtained through pure point models.*

Models	PL Hybrid			PLT Hybrid		
	Quality		Speed Up %	Quality		Speed Up %
	mse	*nccm*		*mse*	*nccm*	
Armadillo	0.0017	0.9999	6.07	0.0017	0.9999	6.07
Ball joint	0.0156	0.9985	25.31	0.0034	0.9999	25.13
Golf club	0.0048	0.9998	63.32	0.0045	0.9997	59.48
Igea artifact	0.0034	0.9998	27.88	0.0034	0.9997	27.61
Isis	0.0132	0.9982	41.55	0.0132	0.9982	43.03
Male	0.0051	1.0001	51.16	0.0051	1.0003	50.97
Upper body	0.0076	0.9999	29.90	0.0076	0.9999	30.09
Rabbit	0.0036	0.9999	23.62	0.0036	0.9999	23.62
Rocker arm	0.0016	1.0001	24.03	0.0016	1.0002	24.25
Santa	0.0021	0.9999	13.19	0.0021	0.9999	13.19
Screwdriver	0.0225	0.9996	26.55	0.0230	0.9997	26.30
Teeth casting	0.0013	0.9998	52.71	0.0012	0.9997	49.83

Table 3: *Pure point models compared with hybrid models.*

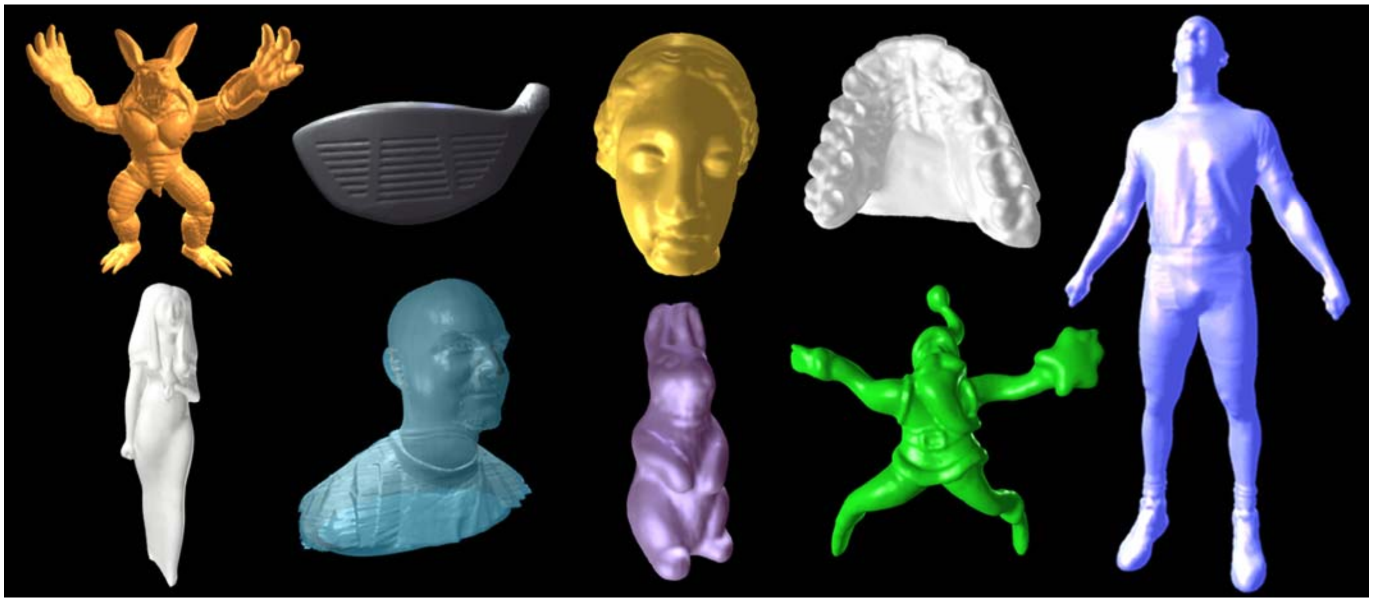


Figure 10: *Hybrid points and line segment models used in experiment two rendered by our method. The Upper body model is transparent, while other models are opaque.*

Figures in Table 3 show significant speedups are achieved to render hybrid models as compared with their corresponding pure point models. For the eleven models used in this experiment, an average speedup of 32.11% is gained for PL hybrids, and 31.63% for PLT hybrids. Maximum speedups are obtained while rendering the Gulf Club model. The value is 63.32% for the Gulf Club's PL hybrid and 59.48% for the PLT hybrid. We expect much higher speedups if we can implement the pipeline in hardware. We are currently investigating this possibility.

6. CONCLUSION AND FUTURE WORK

This paper demonstrates the feasibility of high quality rendering of 3D hybrid models of points, line segments and triangles. In particular, it shows a way to render anti-aliased line segments. Experiments show that the rendered hybrid models have the same high quality as their counterparts using points only. As compared to pure point models, hybrid models enjoy significant rendering efficiency too.

Hybrid models with line segments share many of the advantages and limitations of pure point models. Among the advantages important for interactive visualization is the fact that level of details would be adjusted locally. Among the limitations is the basic fact that the absence of explicit connectivity information makes deformation more difficult than it is in surface triangulations. Also, the use of local affine assumptions in the neighborhood of a point or line segment produces artifacts in the presence of highly nonlinear mappings.

Our current formulation of line segments as a rendering primitive have two limitations, due to the rigorous mathematical derivations we adopt. First, each line segment is required to possess only one normal. Currently, this requirement rules out the possibility of using line segments to render ruled surfaces. Second, each line segment can only be assigned a single color. This is because multi-color line segments or textured line segments do not permit color influences along the line segment to be integrated. However in practice, this restriction can be relaxed. Our experiments show that rendering linearly texture mapped line segments can still produce reasonably good quality images. The restriction on the use of only a single normal could also be removed when normal maps designated for line segments become practical.

Many research issues remain in order to place the neglected rendering primitive "line segment" on an equal footing with its well-known counterparts "triangle" and "point". First, the problem of efficiently acquiring, storing and editing models containing line segments goes far beyond our discussions in this paper. The algorithm we have implemented to extract surface primitives from point sets is a computationally expensive pre-processing phase. An important topic is to investigate better ways to extract line segments from the highly structured point clouds output by 3D scanners. Second, our implementation of the rendering pipeline, catering to all three types of surface primitives, is software based and hence not efficient for interactive applications. The rendering pipeline must be mapped to modern programmable GPUs to achieve better frame rates. One technical problem involves the efficient computation of tangents common to two ellipses. Third, line segments as a rendering primitive provide yet another option to support level-of-detail rendering, to be exploited by novel data structures and algorithms that efficiently reflect the intricate interaction among the different surface elements involved. In conclusion, we believe that the initial experiments reported in this paper establish line segments as a surface primitive of potentially equal importance as triangles and points.

ACKNOWLEDGEMENT

We obtained models used in the experiments from Stanford graphics laboratory website, large geometric models archive of Georgia Institute of Technology as well as the Cyberware.com.

References

[ASS02] I. Avcibas, B. Sankur and K. Sayood. Statistical Evaluation of Image Quality Measures. *Journal of Electronic Imaging*, vol. 11:2, 2002, pp. 206–223.

[CDK04] B. Chen, F. Dachille, and A. E. Kaufman. Footprint Area Sampled Texturing. In *IEEE Trans on Visualization and Computer Graphics,* vol. 10;2, 2004, pp. 230–240.

[ChNg01] B. Chen and M.X. Nguyen. POP: A Hybrid Point and Polygon Rendering System for Large Data. In *Proc. of IEEE Visualization 2001*, pp. 45–52.

[DCSD02] O. Deussen, C. Colditz, M. Stamminger and G. Drettakis. Interactive Visualization of Complex Plant Ecosystems. In *Proc. of IEEE Visualization 2002*, pp. 219–226.

[DeHu02] T.K. Dey and J. Hudson. PMR: Point to Mesh Rendering, A Feature-Based Approach. In *Proc. of IEEE Visualization 2002*, pp. 155–162.

[GrDa98] J.P. Grossman and W.J. Dally. Point Sample Rendering. In *Proc. of* 9th *Eurographics Workshop on Rendering 1998*, pp. 181–192.

[Heck89] P. Heckbert. Fundamentals of Texture Mapping and Image Warping. *Master's Thesis,* University of California, Berkeley, 1989.

[JoCh99] N. Jouppi and C. Chang. Z^3: An Economical Hardware Technique for High-Quality Antialiasing and Transparency. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware 1999*, pp. 85–93.

[LeWh85] M. Levoy and T. Whitted. The Use of Points as a Display Primitive. *Technical Report TR 85-022*, University of North Carolina at Chapel Hill, 1985.

[LoTa97] K.L. Low and T.S. Tan. Model Simplification using Vertex Clustering. In *Proc. of Symposium on Interactive 3D Graphics 1997,* pp. 75–81.

[MPFJ99] J. McCormack, R. Perry, K. I. Farkas, and N. P. Jouppi. Feline: Fast Elliptical Lines for Anisotropic Texture Mapping. In *Proc of SIGGRAPH 1999,* pp. 243–250.

[Nels96] S.R. Nelson. Twelve Characteristics of Correct Antialiased Lines. *Journal of Graphics Tools*, vol.1:4, 1996, pp. 1–20.

[PZVG00] H. Pfister, M. Zwicker, J. van Baar and M. Gross. Surfels: Surface Elements as Rendering Primitives. In *Proc. of SIGGRAPH 2000*, pp. 335–342.

[RPZ02] L. Ren, H. Pfister and M. Zwicker. Object Space EWA Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering. In *Proc. of Eurographics 2002,* pp. 461–470.

[RuLe00] S. Rusinkiewicz and M. Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proc. of SIGGRAPH 2000*, pp. 343–352.

[Stot98] W. Stothers. Cabri Pages – Conics, *www.maths.gla.ac.uk/~wws/cabripages/algebra.html*, 1998.

[WePe95] J. Weber and J. Penn. Creation and Rendering of Realistic Trees. In *Proc. of SIGGRAPH 1995*, pp. 119–128.

[ZPKG02] M. Zwicker, M. Pauly, O. Knoll and M. Gross. Pointshop 3D: An Interactive System for Point-based Surface Editing. In *Proc. of SIGGRAPH 2002*, pp. 322–329.

[ZPVG01] M. Zwicker, H. Pfister, J. van Baar and M. H. Gross. Surface Splatting. In *Proc. of SIGGRAPH 2001*, pp. 371–378.

Appendix

As in [RPZ02], the ellipse of a point can be determined by decomposing $V_{(k,t)}$ into a scaling matrix $\Lambda = \begin{bmatrix} r_0 & 0 \\ 0 & r_1 \end{bmatrix}$ and a rotation matrix $Rot(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$. Together with translation (t_x, t_y) (depends on the location of the point), the equation of an ellipse is:

$$\frac{((x-t_x)\cos(\theta)+(y-t_y)\sin(\theta))^2}{r_0^2}+\frac{((x-t_x)\sin(\theta)+(y-t_y)\cos(\theta))^2}{r_1^2}=1.$$

An ellipse is a specific type of plane conic. A plane conic has an equation of the form:

$$a x^2 + b x y + c y^2 + f x + g y + h = 0.$$

In terms of homogeneous coordinates, this becomes

$$a x^2 + b x y + c y^2 + f x z + g y z + h z^2 = 0.$$

Given two ellipses *C* and *D*, we find the dual of *C*, *C'*, with respect to *D*. The intersection of *C'* and *D* produces the common tangent points. The plane conic can be written as:

$$\tilde{x}^T M \tilde{x} = 0$$

$$\text{where } \tilde{x} = (x, y, z) \text{ and } M = \begin{bmatrix} a & b/2 & f/2 \\ b/2 & c & g/2 \\ f/2 & g/2 & h \end{bmatrix}.$$

Suppose that *C:* $\tilde{x}^T M_C \tilde{x} = 0$ and *D:* $\tilde{x}^T M_D \tilde{x} = 0$ are the two ellipses. Then the dual of *C* with respect to *D* is the conic with equation $\tilde{x}^T M_D M_C^{-1} M_D \tilde{x} = 0$.

After computing the duals, we can combine the conic equations of the ellipse *C* and the dual *D'* to form a quartic equation and also for *D* and *C'*. Solving the quartic equation will produce tangent points at both ellipses.