Rendering Anti-Aliased Line Segments

Keen-Hon Wong, Xin Ouyang, Chi-Wan Lim†, Tiow-Seng Tan†
School of Computing, National University of Singapore

Jürg Nievergelt‡
Swiss Federal Institute of Technology, Zürich

Abstract

Bridging the modeling 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 modeling primitives: triangles, line segments and points. Our experiment shows that models comprising line segments are generally more efficient and effective for high quality rendering as compared to their corresponding pure point models.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation – Anti-aliasing, Viewing Algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – Curve, surface, solid, and object representations; I.3.6 [Computer Graphics]: Methodology and Techniques – Graphics data structures and data types

Keywords: rendering systems, point-based graphics, surface reconstruction, object scanning, 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 [6, 13, 15, 19] and image based rendering, that emphasize the efficiency 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 real-time rendering.

† e-mail: {limchiwa, tants}@comp.nus.edu.sg
‡ e-mail: nievergelt@inf.ethz.ch

The 2005 Computer Graphics International, 22-24 June, Stony Brook, New York, USA.

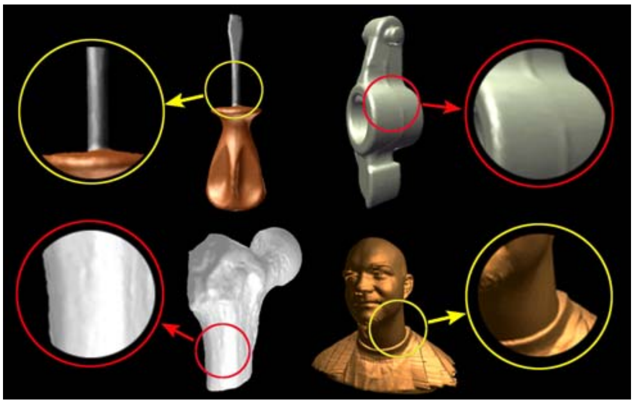

Figure 1: *Hybrid point and line segment models rendered by our method.*

The aliasing problem in computer graphics is caused by the disparity between image representation in the continuous real world and a 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, it is possible that high frequency portions of the continuous signals will masquerade as low frequencies, creating aliasing effects.

Various anti-aliasing techniques propose the use of a low pass filter to stop high frequencies before sampling, so as to render high quality anti-aliased texture mapped triangle models. Recently the Elliptical Weighted Average (EWA) filter [7] has been extended for rendering anti-aliased high quality point models [14, 19]. This paper is a contribution to the use of anti-aliasing techniques for high quality rendering of surface primitives, in particular line segments.

It is interesting to observe the fact that the more 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 [4, 10]. This is surprising in view of the fact that lines are prominent features in many types of scenes and applications, particularly of man-built objects; see the Screwdriver and Rocker arm models in Figure 1. Even objects of irregular shape usually contain some features that are best modeled 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. A line segment should be treated as a primitive, rather than as a degenerate special case of a triangle or polygon, for the same reason that a point is treated as a primitive rather than as a tiny triangle or polygon. A line segment is a simpler object than a long, thin triangle or rectangle. Three advantages come with using line segment as a primitive in surface modeling and rendering.

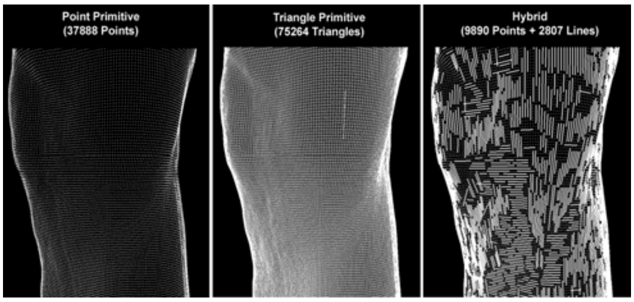

Figure 2: *A knee model in three different representations.*

First, line segments, together with points and triangles, provide an effective representation of object surfaces of any shape. 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 that are smooth 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, a few line segments can replace a large number of sequentially aligned points. Finally, line segments can be rendered much more efficiently than sequences of points. An experiment in Section 5 shows a realistic example where line segments achieve a speedup of 63% as compared to pure point models.

In this paper, we focus on studying the feasibility of using line segment as a rendering primitive, designing corresponding algorithms, reporting on an experiment, and describing research directions with potential to make line based rendering a standard technique in the growing arsenal of graphics tools.

Our main contribution is the extension of the anti-aliasing theory in texture mapping [7] 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 and Figure 9. The empirical validation of this approximation is based on an experiment consisting of two main components.

First, we implement a method to extract primitives from point sets. For a surface given by a set of points in 3D space, we use an adaptive method based on local surface features to extract all three primitives, points, line segments and triangles. This primitive extraction algorithm is developed for the purpose of assessing quality and performance of our proposed anti-aliased line segment rendering technique. Designing more efficient primitive extraction algorithm is left to future work.

Second, we implement a software graphics pipeline to render 3D models represented with any combinations of points, line segments, and triangles. Our experiment shows that the rendered quality of hybrid models is comparable to their corresponding high quality anti-aliased point models. In addition, hybrid models are rendered in significantly less time. This establishes the hybrid model as a competitive rendering alternative to pure point models.

Here is an overview of the paper. Section 2 summarizes related work. Section 3 presents the 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 our experiment to measure the rendering quality and efficiency of hybrid models as compared to pure point models. Section 6 concludes the paper with our ongoing work.

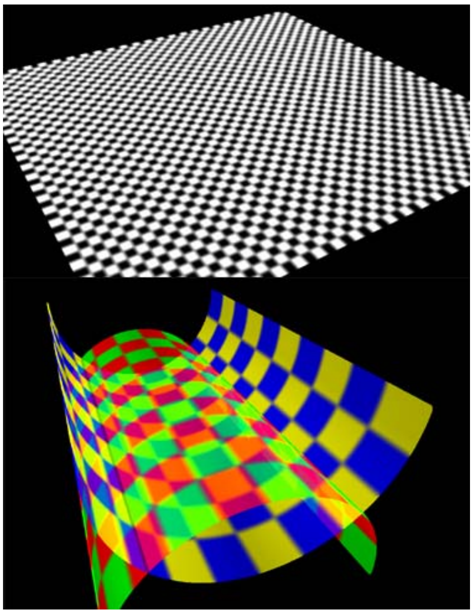

Figure 3: *Opaque* (*top*) *and transparent* (*bottom*) *anti-aliased checkerboard line segment models. Each line segment is painted with only one color and line segments of different colors are separately laid in checker boxes.*

2. RELATED WORK

Point Based Rendering. In 1984, Levoy and Whitted [9] pointed out that classic modeling primitives, i.e. triangles (or polygons), were less appealing for rendering objects with complex geometry. They suggested decoupling modeling geometry from the rendering process by introducing point as a universal primitive, where each point is associated with a small surface area and a normal. Using this idea, Rusinkiewicz and Levoy [15] proposed the QSplat system to render 3D point models. The QSplat system was designed during the course of the Digital Michelangelo Project to render models of a scale of hundreds of millions of triangles at interactive frame rates. In 1998, Grossman and Dally [6] developed the point sample rendering algorithm for high quality real-time rendering of complex objects. Sample points are rendered without any knowledge of surface topology.

In 2000, Pfister *et al.* [13] introduced the surfel, i.e. surface element, paradigm and developed the surfel point rendering pipeline. In 2001, Zwicker *et al.* [19] extended Heckbert's EWA filter [7], and derived 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, which can produce high quality texture filtered images from point samples. In 2002, Ren *et al.* [14] 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 [13, 14, 19]. We extend the anti-aliasing theory in texture mapping [7] to anti-aliased line segments in 3D models; see Figure 3.

Line Segment Rendering. The idea of using a line segment instead of a point 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 [17]. Deussen *et al.* [4] rendered 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-aliased line segments is a major contribution.

Anti-aliased Lines. Many papers covered the problem of rendering anti-aliased lines on screen. A good reference on the desirable characteristics for an anti-aliased line is described by Nelson [12]. In contrast, we render anti-aliased 3D lines where each line has an associated surface area and a normal.

Hybrid Rendering. Chen and Nguyen [3] 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 [15]. Dey and Hudson used a screen pixel as a point in another hybrid system PMR [5]. In our hybrid rendering solution, a point is an anti-aliased splat suggested by Zwicker *et al.* [19]; 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. Both hybrid systems have their emphasis on performance. Nevertheless, in our discussion, primitives are chosen based on local features of object surfaces. For example, parallel line segments are used to approximate a cylinder instead of using a bunch of thin triangles. It is beyond the scope of this paper to discuss acceleration due to LOD techniques as done in the above references.

3. Anti-Aliasing Line Rendering

In order to use the line segment (to be abbreviated as "line" in Sections 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 [7, 14], 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: $g(x) = g_c^{'}(x)\ i(x)$.

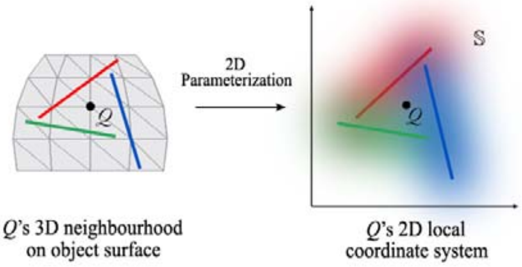

Figure 4: *2D parameterization of point Q's neighborhood to the local coordinate system $\mathbb{S}$. The color of Q is evaluated by accumulating contributions from the set L of lines of various colors in $\mathbb{S}$.*

We expand the above relations in reverse order to obtain:

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

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 $m^{-1}_{(k,t)}(\xi)$ 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:

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

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

Computing $J_{(k,t)}$. Similar to [14], 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 translation T, the corresponding point of $p^o(\mu)$ in the camera space is:

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

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

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 [14], 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 a 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 the footprint assembly problem [2, 11]. Instead, we propose a more precise and mathematically well formulated method which approximates the resampled line based on the two endpoints **S1I** and **S2I** 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 in object space 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 a combination of local affine approximation and affine mapping assumption, together called *local affine assumptions*, a resampled point is approximated by an elliptical Gaussian, which is a unit Gaussian that has both a scaling and rotation operation performed on it. Its intensity is inversely proportionally to the scaling of the unit Gaussian [14].

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 which bypasses 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 under 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 that contain the Gaussian endpoints, and the remaining middle part. 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 (line 3) 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 only 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). 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.

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 a 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.

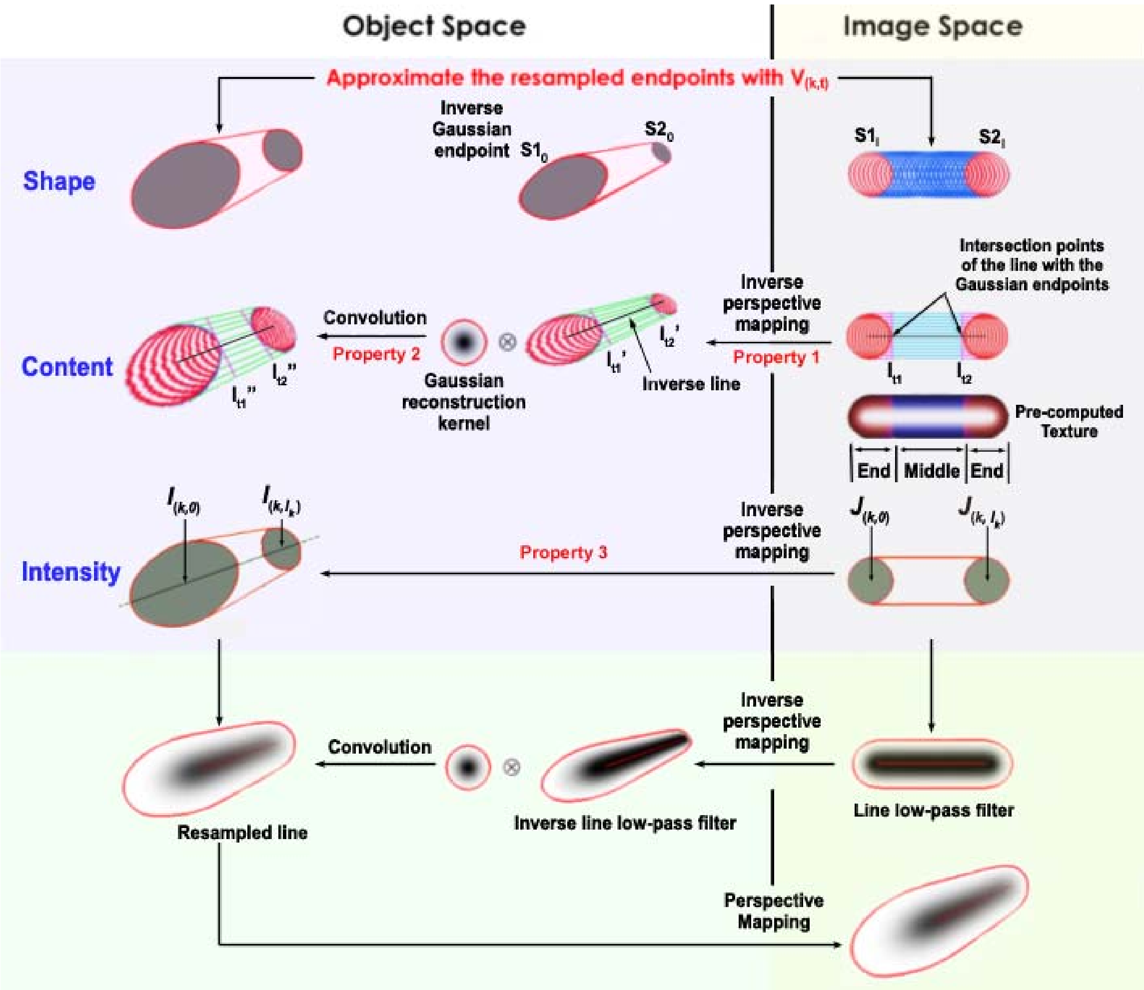

Figure 5: *Approximation of the resampled line.*

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 is largely dependent on the correctness and accuracy of the computations of shapes at the endpoint ellipses. Poor accuracy results 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 the line cannot be approximated resulting in its content being textured with a lot of mapping errors.

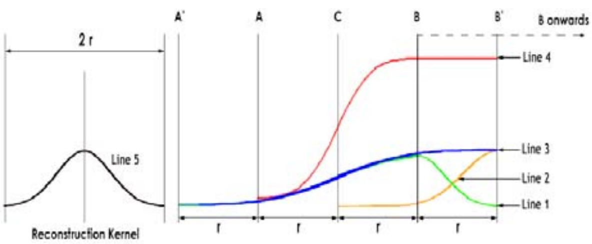

Figure 6: *Convolution of the inverse line low-pass filter* (*line 4 from* **A** *to* **B'**) *with the Gaussian reconstruction kernel* (*line 5 with cutoff r*) *shown in 2D. The result is a line Gaussian between* **A'** *and* **B'** (*line 3*) *which is the convolution from* **A** *to* **B** (*line 1*) *plus the convolution from* **B** *onwards* (*line 2*).

4. RENDERING

Figure 7 shows our rendering pipeline for hybrid models represented by points, line segments and triangles. This pipeline supports high quality anti-aliased points and line segments rendering, while triangles are rendered without anti-aliasing. Compared to the standard rendering pipeline, our rendering pipeline differs in two aspects. First, two extra stages, parameterization and surface transformation, are added after the view transformation stage. 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 modified Z^3 algorithm [8] 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

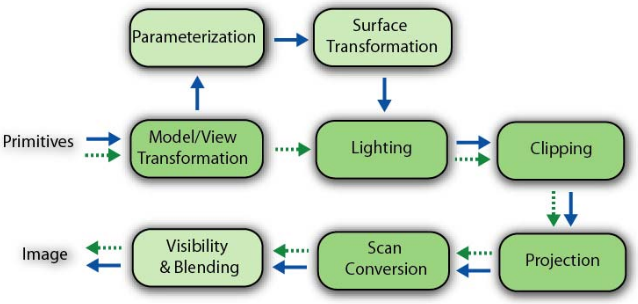

Figure 7: *The rendering pipeline for hybrid models. The rendering of triangles follows the dashed arrows while the rendering of line segments and points follows the bold arrows.*

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 compute $V_{(k,t)}$, 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 [16], 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 $\overline{3,4}$.

We have included a short discussion on how to compute the tangent points of the endpoint ellipses in the Appendix; see also [16]. We use both 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

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

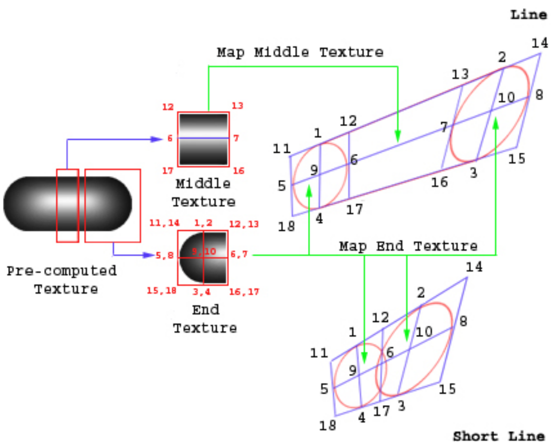

Figure 8: *Texturing line contents with influence textures.*

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

Here, x is defined as the coordinate along the line, while y is in the direction perpendicular to the line. The term *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 $-r < y < r$; 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 sections for texturing line 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 experiment, 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 the required influence textures in real-time. This is however unnecessary as our experiment shows that our simple approximation using only the end texture is generally sufficient to obtain good quality anti-aliased results.

5. EXPERIMENTAL RESULTS

We have conducted an experiment to assess our proposed anti-aliased line segment rendering method. In the experiment, an adaptive method based on local surface features is implemented to extract triangle and line segment primitives from pure point models. To form triangles, points expand through their neighboring points to first form circular discs that represent them within a small error. These discs are subsequently split into triangles. To form line segments, each point is tested with their neighboring points in turn to first form a short line segment and then be extended longer to represent more points that are within a small error away from the line segment. Remaining points of the model that are not represented by any triangle and line segment remain as points of the hybrid model. We term hybrid point, line segment and triangle models as *PLT hybrid* models, and hybrid point and line segment (without triangle) models as *PL hybrid* models.

We implemented the rendering pipeline described in Section 4 in C/C++, and performed the experiment on a P4 1.8 GHz with 512 MB RAM PC. Note that the rendering results of pure point models reported here are also from our own implementation of the theory given in [14, 19] rather than the probably more optimized version of [18].

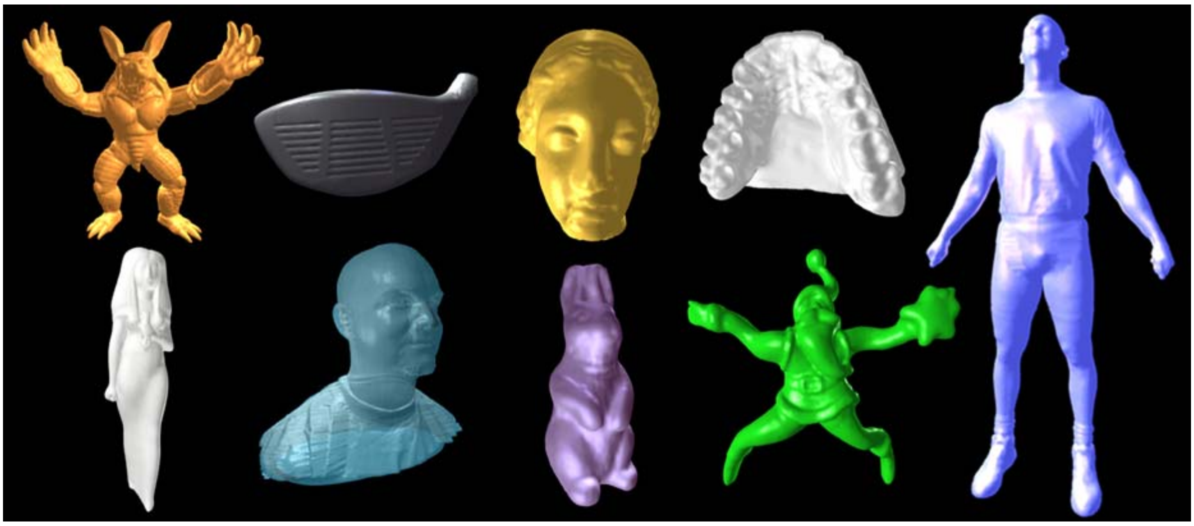

Figure 9: *Hybrid points and line segment models rendered by our method. The Upper body model is transparent, while the other models are opaque.*

In the experiment, we compare the quality and performance of rendering each model in Table 1 using (i) points only, (ii) a PL hybrid, and (iii) a PLT hybrid. Refer to our webpage http://www.comp.nus.edu.sg/~tants/line/ for video clips on the results. Figure 1 and Figure 9 show the rendering outcome of PL hybrid models in Table 1.

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	116604	62799	4104	56327	3952	5435

Table 1: *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	0.0013	0.9998	52.71	0.0012	0.9997	49.83

Table 2: *Rendering comparisons between pure point models and hybrid models.*

Table 2 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. The table shows the mean square error (*mse*, measuring the difference) and the normalized cross-correlation measure (*nccm*, measuring the similarity with 1 indicating identical image) [1] between the corresponding images of pure point and the two other hybrid models. These numerical results again suggest that the rendered hybrid models and their corresponding pure point models indeed have nearly the same quality.

Table 2 also shows significant speedups are achieved when rendering hybrid models as compared with their corresponding pure point models. For the twelve models used in this experiment, an average speedup of 32.11% is gained for PL hybrid models, and 31.63% for PLT hybrid models. Maximum speedups are obtained while rendering the Golf club model. The value is 63.32% for the Golf club's PL hybrid and 59.48% for the PLT hybrid.

6. CONCLUDING REMARKS

This paper demonstrates the feasibility of high quality rendering of 3D hybrid models of points, line segments and triangles. In particular, it illustrates a way to render anti-aliased line segments. Our experiment shows that the rendered hybrid models can achieve similar visual quality as their counterparts using points only. Compared to pure point models, hybrid models are significantly more efficient.

Hybrid models with line segments share many of the advantages and limitations of pure point models. Among the advantages that are important for interactive visualization is the fact that level of details can be adjusted locally. The absence of explicit connectivity information is a limitation that 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 may produce artifacts in the presence of highly nonlinear mappings.

Our current formulation of line segment as a rendering primitive have two limitations, due to the rigorous mathematical derivations we adopt. Firstly, each line segment can possess only one normal. Currently, this requirement rules out the possibility of using line segments to render arbitrary ruled surfaces. Secondly, each line segment can only be assigned a single color. This is because different color influences along the same line segment cannot be integrated. However, in practice, this restriction can be relaxed. Our experiment shows that rendering linearly texture mapped line segments can still produce images of reasonably good quality. The restriction on the use of only a single normal can also be removed when normal maps designated for line segments become feasible.

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 using 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 should 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, using line segment as a rendering primitive provides yet another option to support level of details rendering, to be exploited by novel data structures and algorithms that efficiently reflect the intricate interaction among the different surface primitives involved. In conclusion, we believe that the initial experiment reported in this paper establishes line segments as a surface primitive of potentially equal importance as triangles and points.

ACKNOWLEDGEMENTS

The pure point models used in our experiment were obtained from http://www.cyberware.com. This research is supported by the National University of Singapore under grant R-252-000-216-112.

REFERENCES

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

[2] 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.

[3] 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.

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

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

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

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

[8] 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.

[9] 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.

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

[11] 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.

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

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

[14] 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.

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

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

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

[18] 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.

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

APPENDIX

Following [14], 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 *C'* of *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.