Rendering Generalized Cylinders with Paintstrokes

Ivan Neulander* Michiel van de Panne†
Dynamic Graphics Project, University of Toronto

Abstract

We present an efficient technique for dynamically tessellating generalized cylinders. We make direct use of the generalized cylinder's screen-space projection in order to minimize the number of polygons required to construct its image. Used in conjunction with our A-Buffer polygon renderer, our technique strikes a good balance between speed and image quality when used at small to medium scales, generally surpassing other methods for rendering generalized cylinders.

Résumé

Nous présentons une technique efficace pour la tesselation dynamique des cylindres généralisés. Nous utilisons la projection sur l'écran pour minimiser le nombre de polygones nécessaires pour construire l'image. Utilisée avec un algorithme A-buffer, la technique est un bon compromis entre léfficacité et la qualité.

1 Introduction

A generalized cylinder is the surface produced by extruding a circle along a path through space, allowing the circle's radius to vary along the path. This paper presents an efficient method for rendering this surface in a polygon-based projective rendering system, using a primitive called the *paintstroke*.

Paintstrokes can serve as inexpensive building blocks for many types of complex geometry. Combined in large numbers, they can be used to efficiently render a variety of detailed natural phenomena such as fur, hair, branches, twigs, and pine needles. Simpler structures like wires, hoses, and pipes are equally suitable. Using their view-dependent tessellation, paintstrokes can also easily approximate volumetric opacity effects. This makes them useful in rendering objects such as water streams, icicles, and wisps of smoke, which has traditionally been difficult to accomplish with other projective-rendering methods, necessitating the expensive solution of ray-tracing.

Paintstrokes employ a multiscale dynamic tessellation of generalized cylinders. While this technique incurs some overhead as compared to a fixed tessellation scheme, it capitalizes on important symmetries and view-invariances of the generalized cylinder, whose screen projection can be accurately tiled with only a small number of relatively large polygons. The resulting savings in vertex transformations, rasterization overhead, and edge antialiasing more than compensate for the tessellation cost. Furthermore, by automatically adjusting the granularity of their tessellation, paintstrokes smoothly adapt their level of detail to the scale at which they are rendered.

2 Related Work

A variety of methods have been used in rendering generalized cylinders and similar tubular objects. Traditional polygonal models, using dynamic spline surface tessellation [SC88, Sil90, AES94] or polygonal simplification/refinement [Hop97] are effective, but still require relatively many polygons to ensure a smooth silhouette, a consistent projected tube thickness, and unwavering specular highlights. Moreover, the overhead of transforming and tessellating a spline surface at small scales can be high.

Jim Blinn [Bli89] describes a view-adaptive tessellation scheme for Gouraud-shaded cylinders that he calls *optimal tubes*, and an extension to handle constant-radius generalized cylinders. Blinn's approach is similar to ours, also applying view-dependent tessellation. However, the inability to handle specular reflection and radius variations limits the scope of applications for optimal tubes.

A variety of particle system approaches have also been applied, most notably the brush extrusion method proposed by Turner Whitted [Whi83], the cone-spheres of Nelson Max [Max90], and the polyline primitive that is often used in rendering hair [LTT91, RCI91]. The first two methods potentially require a large number of primitives to render a curved tube without discontinuities in the silhouette

*ivan@dgp.toronto.edu
†van@dgp.toronto.edu

or shading. Polylines, on the other hand, are highly effective—but only at very small scales, since they cannot display any breadthwise shading variation. Moreover, the pre-integrated shading model they typically employ, based on the cylindrical Phong integral introduced by Kajiya and Kay [KK89], is only intended for small-scale rendering.

At smaller scales, the use of textured impostors and volumetric textures to represent large numbers of tubular objects becomes viable. Because a textured impostor lacks the true geometry of the model it represents, it cannot faithfully capture the parallax, occlusion, and shading effects that one would expect to see at larger scales. Volumetric textures do capture these effects, but they require the considerable overhead of a ray-traced volumetric renderer. Despite the use of multiresolution models to speed up this rendering, as presented by Fabrice Neyret [Ney95], volumetric textures usually render much more slowly than projective rendering primitives.

3 Paintstroke Representation

The essential properties of a paintstroke can be described with a one-dimensional piecewise parametric function $\mathbf{ps}_m(t)$ with the real $t \epsilon [0, 1]$. $\mathbf{ps}_m(t)$ is defined using a set of $n \geq 2$ control points, $\{\mathbf{cp}_0, \mathbf{cp}_1, \ldots, \mathbf{cp}_{n-1}\}$, such that $\mathbf{ps}_m(0) = \mathbf{cp}_m$ and $\mathbf{ps}_m(1) = \mathbf{cp}_{m+1}$, where $0 \leq m \leq n$. As a notational shorthand, we omit the subscript m and simply write $\mathbf{ps}(t)$ when this does not introduce ambiguity. When discussing paintstrokes, we use the term *section* to denote the portion of a paintstroke between two adjacent control points. The term *segment* refers to a subset of a section.

The components of $\mathbf{ps}$(t) are visual attributes that vary along the length of the paintstroke: position (x, y, z), radius, colour (r, g, b), opacity, and reflectance. We respectively symbolize these as: $\mathbf{pos}$(t), rad(t), $\mathbf{colour}$(t), $\mathbf{op}$(t), $\mathbf{refl}$(t). The first two are interpolated using Catmull-Rom splines, while the rest are interpolated linearly. Another term we shall refer to is the *view vector*, defined as the unit vector from the viewer (at the origin) to a point on the paintstroke. Thus, $\mathbf{view}(t) = \mathbf{pos}(t)/\|\mathbf{pos}(t)\|$.

4 Paintstroke Tessellation

Traditional tessellation schemes subdivide a surface into a set of world-space or eye-space polygons, based on surface curvature criteria. The paintstroke in effect directly polygonizes the screen-projection of a generalized cylinder—not the full eye-space surface.

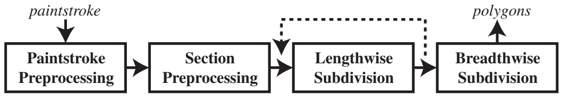


Figure 1: Stages of paintstroke tessellation.

This is what makes the name "paintstroke" appropriate to our primitive: an artist drawing a three-dimensional tube with a single stroke of the paintbrush capitalizes on the simplicity of this object's screen projection, as does our tessellation scheme. Figure 1 summarizes the basic steps of paintstroke tessellation that we are about to discuss. Some of the minor steps and algorithmic details, such as the construction of a paintstroke's endcaps, are omitted in this paper for brevity, but can be found in [Neu97].

4.1 Paintstroke and Section Preprocessing

The first step in the tessellation process transforms all the geometric data stored in the control points $\{\mathbf{cp}_0, \mathbf{cp}_1, \ldots, \mathbf{cp}_{n-1}\}$ from world-space to eye-space. This data consists of the $\mathbf{pos}$ and $\mathbf{N}_{gl}$ components. The latter is called the *global normal* vector and is used for global shading effects, discussed in §5.3.

Each section of the paintstroke, bounded by control points $\{\mathbf{cp}_0, \mathbf{cp}_1\}, \{\mathbf{cp}_1, \mathbf{cp}_2\}, \ldots, \{\mathbf{cp}_{n-2}, \mathbf{cp}_{n-1}\}$, is rendered individually. The polynomial coefficients of the interpolants $\mathbf{pos}(t)$ and $rad(t)$ are computed, as well as for their derivatives and antiderivatives.

4.2 Lengthwise Subdivision

Once the piecewise interpolants have been generated, the section between each pair of adjacent control points is recursively subdivided until the constraints discussed below have been satisfied. Whenever a segment is subdivided, the split occurs at the parametric midpoint, i.e. at $\mathbf{ps}(0.5)$. The two halves are then recursively subdivided in the same manner until no further subdivisions are required.

Whether paintstroke segment $\mathbf{ps}(t)$ for $t \,\epsilon\, [a, b]$, $a < b$ is subdivided depends on the behaviour of its $\mathbf{pos}(t)$ and $rad(t)$ components. If it is approximately linear in $\mathbf{pos}(t)$ and $rad(t)$, it is not subdivided, being subsequently drawn as a truncated cone. If there are inflection points in any component of $\mathbf{pos}(t)$ or in $rad(t)$, these need to be dealt with, as described in §4.2.3.

4.2.1 Position Constraint I

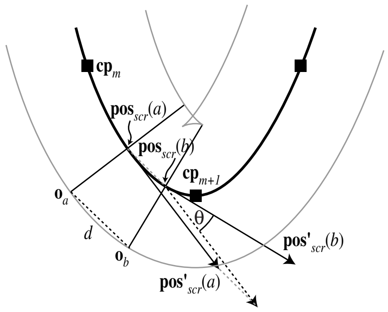


Figure 2: The elements of Position Constraint I.

The first position constraint is based on the angle θ between the two-dimensional tangent vectors $\mathbf{pos}'_{scr}(a)$ and $\mathbf{pos}'_{scr}(b)$. These are the (x, y) screen-space projections of $\mathbf{pos}'(t)$ at the segment's endpoints, $t = a$ and $t = b$. If θ exceeds a threshold value denoted by θ_{max}, the constraint forces a subdivision. $\theta_{max} \epsilon (0^0, 90^0)$ is a function of the segment's maximum length, as defined below, and a user-adjustable tolerance parameter $tol_\theta > 0$.

The vectors $\mathbf{pos}'_{scr}(a)$ and $\mathbf{pos}'_{scr}(b)$ are obtained by analytically differentiating the function $\mathbf{pos}_{scr}(t)$, the screen-projection of the paintstroke's path. The dot product of the normalized vectors yields $\cos\theta$. If this value is negative, we know that θ exceeds the maximum value of θ_{max}, so we immediately subdivide the segment. Otherwise, we need to determine θ_{max}. We begin by finding the segment's maximum length d, defined as the distance between the two *outside points* $\mathbf{o}_a$ and $\mathbf{o}_b$, which are the points lying on the outside boundary of the segment at $t = a$ and $t = b$, respectively (see Figure 2). The outside point $\mathbf{o}_a$ is computed by displacing the position point $\mathbf{pos}_{scr}(a)$ by one of the two vectors perpendicular to $\mathbf{pos}'_{scr}(a)$, namely

$$\begin{bmatrix} {pos_{scr}}'_y(a) \\ -{pos_{scr}}'_x(a) \end{bmatrix} \text{ or } \begin{bmatrix} -{pos_{scr}}'_y(a) \\ {pos_{scr}}'_x(a) \end{bmatrix} \tag{1}$$

which has been normalized and scaled to the screen-projected radius. The choice between the two is based on $\mathbf{pos}''_{scr}(a)$, since the second derivative vector always points toward the centre of curvature. The same algorithm is applied to obtain $\mathbf{o}_b$. Once the outside points have been determined, we use d^2, the square of the distance between them, and the parameter tol_θ, to compute $\cos^2\theta_{max} = \frac{d^2}{d^2+tol_\theta^2}$. This formula forces the lengthwise subdivision granularity to adapt to the screen-projected length of the paintstroke segment, in a manner that can be modified by tuning tol_θ.

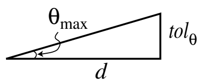
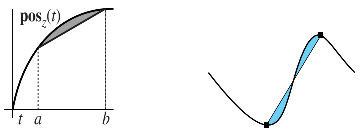


Figure 3: Geometric interpretation of θ_{max}.

4.2.2 Position Constraint II and Radius Constraint

The second position constraint maintains a desired degree of linearity in the z-component of $\mathbf{pos}(t)$. This is necessary to ensure that a curved segment is adequately subdivided even when viewed from an angle that makes its screen projection close to linear.

To implement this constraint, we compute over the interval $[a, b]$ the exact average values of $pos_z(t)$ and its linear interpolant, using the integrals shown in Figure 4. The absolute difference between the average values is a measure of $pos_z(t)$'s nonlinearity. The difference is then scaled by $\frac{d_{proj}}{pos_z(a)}$, a factor representing the foreshortening effect of the perspective transformation at $\mathbf{pos}(a)$, given the projection distance d_{proj}. Finally, this value is bounded by the user-specified tolerance tol_z, which yields the inequality $\frac{d_{proj}}{pos_z(a)}\left|\frac{\int_a^b pos_z(t)\,dt}{b-a} - \frac{pos_z(a)+pos_z(b)}{2}\right| < tol_z$.



(a) OK (b) Problem

Figure 4: Position Constraint II.

The radius constraint ensures a smooth lengthwise variation in the radius of a segment. It is precisely analogous to Position Constraint II, relying on the perspective-adjusted average difference between the $rad(t)$ function and its linear interpolant.

4.2.3 Inflection Point Constraints

Position Constraint II and the Radius Constraint both require that the spline interpolants to which they apply have monotonic derivatives over the entire segment $t \,\epsilon\, [a, b]$. Hence, must ensure that $pos_z(t)$

and $rad(t)$ have no inflection points in the open interval $t \in (a, b)$. Since the interpolants are cubics, finding an inflection point amounts to finding the zero of the interpolant's second derivative within the interval (a, b). If one is found, a subdivision is carried out at the value of t where it occurs, producing a pair of subsegments that do not contain the inflection point in their open intervals.

Inflection points may also occur within the screen-projected path of a segment. This can cause a nonlinear path segment to have equal tangents at the endpoints, thereby erroneously satisfying Position Constraint I. Locating this type of inflection point is expensive, but if we approximate the curve's perspective projection with a simple orthonormal one, the problem becomes more tractable. For a more detailed discussion of inflection points, refer to [Neu97].

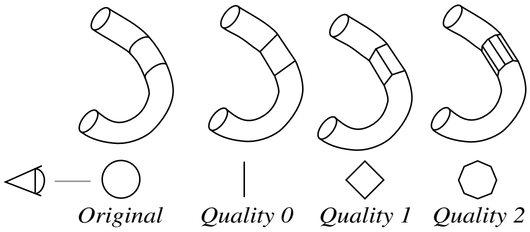
4.3 Breadthwise Subdivision

Breadthwise subdivision divides a segment into a ring of polygons which tile the truncated cone that the segment represents. The subdivisions occur relative to the centre and edges of the segment, which are view-dependent. The specifics of this tessellation depend on the paintstroke's *quality level*. Each paintstroke bears one of three possible quality levels, numbered 0, 1, and 2. This quantity is determined at an early preprocessing stage, indicated in Figure 1. As shown in Figure 5(a), the number of each quality level represents $\log_2 N$, where N is the number of polygons tiling the side of the segment that is closest to the viewer. Hence, a quality-zero segment is tessellated into a single polygon that always faces the viewer, a quality-one segment into two on each side (the viewer's side and the side opposite to it), and a quality-two segment into four on each side.

For quality-one and quality-two paintstrokes, the side opposite the viewer is often hidden, and can thus be safely ignored, saving considerable rendering time. This optimization takes into account the values of $\mathbf{pos}'(t)$, $rad'(t)$, and $\mathbf{view}(t)$, at a segment's endpoints. If the side opposite the viewer is hidden or nearly hidden, its tellessation is suppressed. More detail is provided in [Neu97].

4.3.1 Rendering Quality

The tessellation of quality-zero segments is the simplest: the entire segment becomes a single quadrilateral with vertices along the edges of the paintstroke, corresponding to the silhouette of the generalized cylinder. This scheme yields the smallest number of polygons, and the greatest savings over a general-purpose tessellation method. However, it also yields



(a) Tessellation Meshes

(b) Rendered Images

Figure 5: Paintstrokes generated at the three rendering quality levels.

the poorest rendering quality in several regards: (1) The shading is inaccurate, being based on the linear (x, y, z)-componentwise interpolation of high curvature over a single polygon. (2) A quality-zero segment disappears when viewed head-on, i.e. when the tangent of its path, $\mathbf{pos}'(\mathrm{t})$, is collinear with the view vector. (3) The self-occlusion effect accompanying high screen-space curvature—seen as a fold in the surface—is inaccurate. (4) Paintstrokes of this quality level do not support breadthwise opacity variation. Despite their limitations, quality-zero paintstrokes are still very useful at a small scale, where the above deficiencies are largely irrelevant.

In quality-one paintstrokes, each side of the segment is divided into two equal-sized quadrilaterals sharing a common edge along the middle. Interpolating normals across two polygons rather than one greatly improves the appearance of a shaded segment, and also improves the screen-space fold at high curvature. Furthermore, paintstrokes of this quality level no longer disappear when viewed head-on, although they may reveal their quadrilateral cross-section if their path is sufficiently straight.

Quality-two paintstrokes produce the highest quality images, both in their shading and in their appearance when viewed head-on. However, because they generate four polygons per side, their savings over a general tessellation scheme are less pronounced. They are best suited to rendering at larger scales, where high image quality is essential.

4.3.2 Determining the Polygon Vertices

To obtain a paintstroke polygon's vertices, we first determine their displacements from a point on the central path of the segment. These displacements are view-dependent vectors which all originate at $\mathbf{pos}(t)$, radiating outward as shown in Figure 6. We refer to them as the **out** vectors: $\mathbf{out}_{edge}(t)$ points to one of the segment's two lengthwise silhouette edges, while $\mathbf{out}_{centre}(t)$ reaches the breadthwise centre of the segment. The other two vectors, $\mathbf{out}_{mid_1}(t)$ and $\mathbf{out}_{mid_2}(t)$ are linear combinations of $\mathbf{out}_{edge}(t)$ and $\mathbf{out}_{centre}(t)$ that point to the angular midpoint between the centre and each edge.

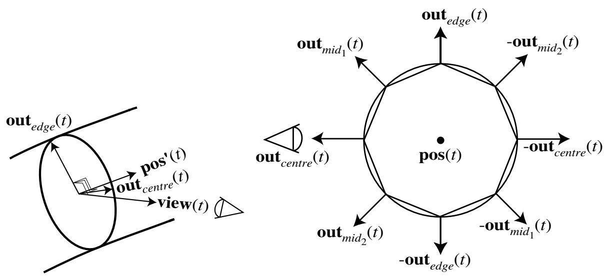


Figure 6: The view-dependent **out** vectors.

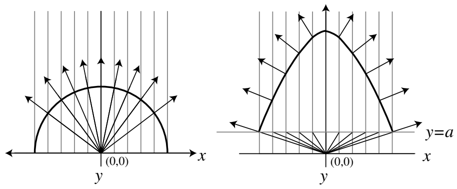
Vertices along the edges are computed as $\mathbf{pos}(t)+\mathbf{out}_{edge}(t)$ and $\mathbf{pos}(t)-\mathbf{out}_{edge}(t)$. For quality-zero paintstrokes, these are the only vertices used. For higher quality levels, the centre vertex on the viewer's side is given by $\mathbf{pos}(t)+\mathbf{out}_{centre}(t)$, and the one one on the opposite side by $\mathbf{pos}(t)-\mathbf{out}_{centre}(t)$. For quality-two paintstrokes, the remaining four vertices are determined in the same manner. The vertices are computed at both endpoints of the segment, yielding a ring (or semi-ring, if only the viewer's side is visible) of quadrilaterals.

4.4 Computing the Normals

Each normal vector is equal to its corresponding $\mathbf{out}(t)$ vector plus an adjustment vector $\mathbf{adj}(t)$ in the direction of $\mathbf{pos}'(t)$, whose norm is determined by the derivatives of the radius and position. [1] The $\mathbf{out}(t)$ vectors define the breadthwise normal variation of a paintstroke (which is equivalent to that of a plain cylinder), while the $\mathbf{adj}(t)$ vector represents the lengthwise normal variation, determined by the behaviour of the paintstroke's radius.

$$\mathbf{adj}(t) = -\frac{rad'(t)}{\|\mathbf{pos}'(t)\|^2}\,\mathbf{pos}'(t) \tag{2}$$

The normals within the interior of a polygon are derived by bilinearly interpolating the (x, y, z) components of the vertex normals across the polygon's screen-space projection. Consequently, the rate of change of an interpolated normal with respect to interpolation distance is smallest at the edges and greatest somewhere in the interior of a polygon. As Figure 7 illustrates, this is a very poor approximation of a generalized cylinder's breadthwise normal distribution. When a large amount of curvature is interpolated over a single polygon, the resulting image appears to have a ridge at the centre (see Figure 5) because the normals at that point are varying most rapidly. But when the curvature is expressed over multiple polygons (as with quality-one and quality-two paintstrokes), the approximation becomes much better.



(a) Cylinder (b) Polygon

Figure 7: Breadthwise normals of a true cylinder and a linearly interpolated polygonal representation.

For segments of quality zero, the normals along the lengthwise edges of a polygon are "nudged" toward the normals of the centre vertices. This is needed because the true edge normals are co-planar (given the orthonormal projection used in determining them), so the subsequent interpolation between the edges would never have the required perpendicular component in the central direction—at the middle of the polygon, the normal would simply vanish. The amount by which the edge normals are shifted toward the centre normal, specified by the nudge factor, determines both the range and distribution of the normals.

[1] Note that our calculation is based on an orthonormal projection of the paintstroke, a reasonable simplification for fairly small-scale rendering.

5 Special Rendering Effects

5.1 Lengthwise Opacity Variation

The lengthwise opacity of a paintstroke segment varies according to the values of $op_{min}(t)$ and $op_{max}(t)$. The former opacity is applied when the segment is viewed head-on, whereas the latter is used when it is viewed orthogonally to its path. For intermediate cases, an opacity value is interpolated between these extremes, based on the dot product of the normalized tangent vector and the view vector. The interpolated lengthwise opacity, op_{len}, is given by the following equation:

$$\begin{aligned} op_{len}(t) &= \alpha\, op_{max}(t) + (1-\alpha)\, op_{min}(t) \\ \alpha &= \left| \mathbf{view}(t) \cdot \frac{\mathbf{pos}'(t)}{\|\mathbf{pos}'(t)\|} \right| \end{aligned} \quad (3)$$

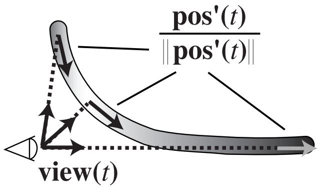


Figure 8: Lengthwise opacity model.

By exploiting this opacity interpolation, it is possible to simulate volume opacity, which varies according to the distance that penetrating light rays travel through a material. However, since we are basing the opacity on just a tangent vector, rather than the true distance, this effect is only an approximation. Nevertheless, as Figure 15(a) illustrates, the results are usually quite good.

5.2 Breadthwise Opacity Variation

The surface normals spanning the breadth of a paintstroke provide a simple and useful way of modulating the opacity across its breadth. This effect is achieved in each ring of polygons comprising a paintstroke segment by storing a dot product of the normal at each viewer-facing vertex with the view vector. All the dot products within the segment are then divided by the maximum dot product, which occurs at the centre vertex. The quotient is stored for each vertex $\mathbf{v}_i$ as the parameter s_i. Given the vertex normal $\mathbf{N}_i$, the equations for s_i and for the breadthwise opacity, o_i, are

$$s_i = \frac{\mathbf{N}_i \cdot \mathbf{view}}{\max_j(\mathbf{N}_j \cdot \mathbf{view})} \quad (4)$$

$$o_i = (1 - s_i) op_{edge} + s_i\, op_{centre} \quad (5)$$

This value of o_i is then multiplied by the lengthwise opacity value, op_{len}, to yield final opacity at each vertex.

Breadthwise opacity variation can be used to produce fuzzy paintstrokes (by using a high value for op_{centre} and a low value for op_{edge}) or to simulate the Fresnel effect for streams of water or icicles (by doing the reverse). Examples of both are shown in Figure 15(b).

5.3 Global Shading Algorithm

The surface normals derived in §4.4 enable us to apply accurate local shading to each individual paintstroke. However, this by itself fails to take into account the shadows that paintstrokes can cast onto themselves and other paintstrokes. While the problem could be rectified by explicitly computing shadows for all paintstrokes, as with shadow buffering [Wil78], this approach would significantly increase rendering time and memory requirements. Our solution, while not as general as true shadow calculation, produces good results for homogeneous layers of paintstrokes covering a roughly convex shape (see Figure 14). It is similar in spirit to the one proposed by Reeves and Blau [RB85].

Each control point of a paintstroke has associated with it a global normal $\mathbf{N}_{gl}$ and a global depth value d_{gl}. The former indicates the direction of the global surface to which the control point belongs, and the latter the relative depth from that surface, expressed as a value between zero (on the surface) and one (maximally distant from the surface). At any control point, the estimated amount of light penetration, $p\epsilon[0,1]$, relative to a light direction $\mathbf{L}$ is used to scale down the control point's reflectance using a negative exponential based on p.

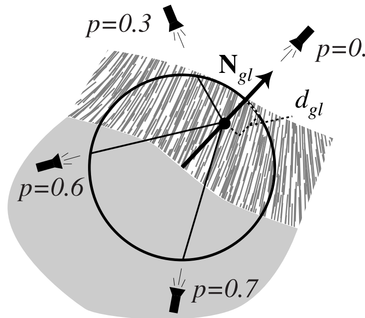


Figure 9: Penetration values at various light angles.

Figure 9 provides a geometric interpretation of p: We construct a unit sphere centred at the origin. The position of the control point in this model is defined to be $(1-d_{gl})\mathbf{N}_{gl}$, which always lies within the sphere. Now we extend a line segment in the direction of the light vector L, joining some point on the surface of the sphere to the control point within. Assuming that $\mathbf{N}_{gl}$ and $\mathbf{L}$ are normalized, the length of this line segment equals $2p$, where

$$\begin{aligned} 2p &= \sqrt{(\alpha\beta)^2 - \alpha^2 + 1} - \alpha\beta \qquad (6) \\ \alpha &= 1 - d_{gl} \\ \beta &= \mathbf{N}_{gl} \cdot \mathbf{L} \end{aligned}$$

6 Results

Profiling tests on a 200 MHz PowerPC 604e system indicate that paintstroke tessellation consumes between 5 and 10 percent of the total rendering time, depending on the amount of screen coverage. This statistic is, of course, based on the speed of our high-quality software-based polygon renderer, which is more than an order of magnitude slower than the hardware-based systems found in graphics workstations.

In order to provide a comparison between paintstrokes and efficient static models of generalized cylinders, we have implemented an algorithm that translates a paintstroke description into the following static model: an orthogonal extrusion along a given path of an n-sided regular polygon of varying size (see Figure 10). The algorithm is similar to one used by Jules Bloomenthal in [Blo85] for polygonizing tree branches, although ours is adaptive to the lengthwise curvature of the tube.

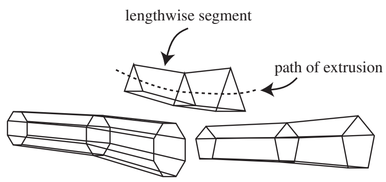
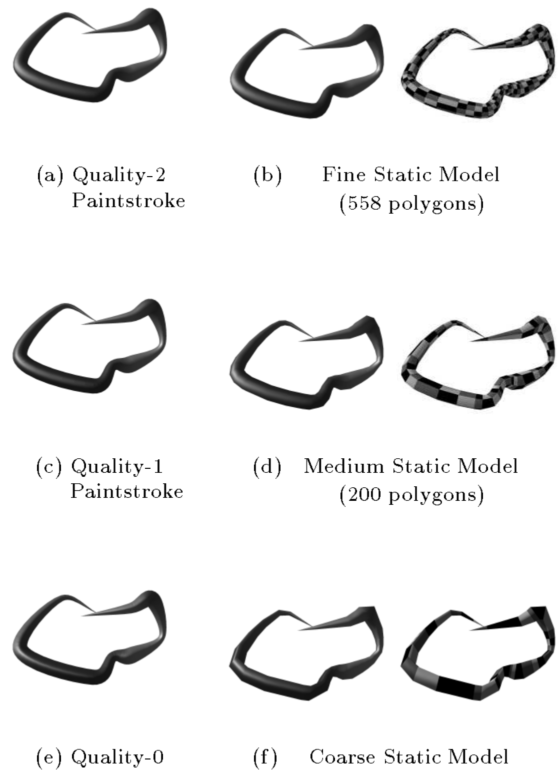


Figure 10: Extrusions of regular polygons.

We constructed a cluster of 27 of paintstrokes, and rendered a set of animations of it, comprising 50 frames in total. The animations were carried out at three different constant distances from the centre of the matrix, so as to simulate rendering at a large, medium, and small scale, as shown in Figure 12. At each scale, two animations were made, one using a higher ("conservative") quality level and the other using a lower ("aggressive") one.

We converted this paintstroke-based scene description into three static polygonal models, differing only in tessellation granularity, as depicted in Figure 11. Each of these was optimized for the large, medium, or small scale of the animation, respectively, by using the minimum number of polygons required to maintain reasonable image quality at its corresponding scale. Each polygonal model was then rendered in the same animations used with the paintstrokes, one at each scale.



(a) Quality-2 Paintstroke
(b) Fine Static Model (558 polygons)
(c) Quality-1 Paintstroke
(d) Medium Static Model (200 polygons)
(e) Quality-0 Paintstroke
(f) Coarse Static Model (96 polygons)

Figure 11: Paintstroke-based and static models of the benchmarked tube.

In calibrating the static tessellation algorithm, we sought the same level of image quality as was achieved in the aggressive paintstroke animations: no silhouette discontinuities or abrupt transitions in the shading. This was an empirical process that involved repeated trials in reducing the polygon count, while maintaining the quality of the entire animation sequence.

Scale	Avg. per Tube	Paintstroke		Static Polygonal Model		
		Conservative	Aggressive	Fine	Medium	Coarse
Large	Breadthwise Quality	2	1	**9-gon**	5-gon	triangle
	Total Polygons	268.4	135.0	**558**	200	96
	Polygons Rendered	238.3	118.2	**275.9**	98.6	46.5
	Pixel Area	1186.6	1188.0	**1190.6**	1182.0	1155.1
	Rendering Time (s/60)	11.01	7.48	**12.19**	6.78*	5.91*
Medium	Breadthwise Quality	1	0	9-gon	**5-gon**	triangle
	Total Polygons	100.6	46.5	558	**200**	96
	Polygons Rendered	87.6	43.1	277.5	**99.3**	47.4
	Pixel Area	292.7	292.6	295.1	**293.0**	286.2
	Rendering Time (s/60)	4.00	2.59	9.61	**4.78**	2.71*
Small	Breadthwise Quality	0	0	9-gon	5-gon	**triangle**
	Total Polygons	36.4	36.4	558	200	**96**
	Polygons Rendered	33.6	33.6	278.3	99.7	**47.8**
	Pixel Area	72.4	72.4	73.6	73.1	**71.4**
	Rendering Time (s/60)	1.48	1.48	8.52	3.45	**1.88**

Table 1: Comparison of paintstrokes with statically tessellated polygonal models.

(a) Large (b) Medium (c) Small

Figure 12: Models of the tube used in our comparison.

6.0.1 Benchmark Results

The results of these benchmarks appear in Table 1. Statistics were gathered for all 1,350 tubes rendered (27 tubes/frame $\times$ 50 frames) and then divided by 1,350 to provide a per-tube average. At each scale, the most interesting comparisons are between the paintstrokes and the polygonal model that is best suited to the scale. Figures describing the latter form a diagonal of boldfaced entries in Table 1.

The *Breadthwise Quality* in the figure refers to the quality level for paintstrokes, or the degree of the regular polygon that was extruded for the static models. The difference between *Total Polygons* and *Polygons Rendered* is due to clipping and backfaceculling. The *Rendering Time*, measured in sixtieths of a second, is on a 200 MHz PowerPC 604e system with 32 MB of RAM and 1 MB L2 cache. Asterisked entries indicate an unacceptably poor image quality, resulting from using an overly coarse static model relative to the scale.

6.1 Results Summary

Our comparison shows that paintstrokes can provide a faster and more efficient means of rendering generalized cylinders than statically tessellated models, even at the latter's optimal scale. This implies that even a dynamic polygonal model, which consistently maintains appropriate tessellation granularity, is unlikely to outperform paintstrokes in rendering generalized cylinders, unless it takes advantage of their symmetry and view-invariant properties as do the paintstrokes.

While these results are encouraging, they come with a few caveats. Hardware-based polygon renderers, with their pipelined architectures, tend to work faster (on a per-polygon basis) with static tessellations than with dynamic ones. Moreover, most hardware-based Phong shading systems cannot adequately cope with polygons containing a large amount of normal variation [BW86], which is generally the case with paintstroke polygons. Finally, although the image quality of paintstrokes is quite good, our present implementation lacks texture-mapping and true shadow generation, which would be useful at larger scales.

7 Conclusions

A wide variety of models used in computer graphics can be reasonably approximated by generalized cylinders. An efficient technique for rendering the latter is therefore of considerable utility. While a number of traditional rendering methods have been applied to the task, they generally fail to achieve a good balance of speed and image quality at small to medium scales. The purpose of this paper was to provide an efficient means of rendering generalized cylinders at precisely these scales. This was achieved through the paintstroke primitive and its supporting A-Buffer-based projective rendering architecture.

By applying a view-adaptive tessellation algorithm that exploits the simplicity and symmetry of the generalized cylinder's screen-space projection, paintstrokes are able to accurately approximate this surface using much fewer polygons than competing methods, producing savings in both rendering speed and memory consumption. In addition, paintstrokes provide view-dependent rendering effects that would be difficult to achieve with traditional polygonal models.

References

[AES94] S. S. Abi-Ezzi and S. Subramaniam. Fast dynamic tessellation of trimmed NURBS surfaces. In *Computer Graphics Forum*, volume 13, pages 107–126. Eurographics, Basil Blackwell Ltd, 1994.

[Bli89] James F. Blinn. Jim Blinn's corner: Optimal tubes. *IEEE Computer Graphics and Applications*, September 1989.

[Blo85] Jules Bloomenthal. Modeling the mighty maple. In *SIGGRAPH '85 Proceedings*, volume 19, pages 305–311. ACM SIGGRAPH, Addison Wesley, July 1985.

[BW86] Gary Bishop and David M. Weimer. Fast Phong shading. In *SIGGRAPH '86 Proceedings*, volume 20, pages 103–106. ACM SIGGRAPH, Addison Wesley, August 1986.

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH '97 Proceedings*, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997.

[KK89] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In *SIGGRAPH '89 Proceedings*, volume 23, pages 271–280. ACM SIGGRAPH, Addison Wesley, July 1989.

[LTT91] Andre M. LeBlanc, Russell Turner, and Daniel Thalmann. Rendering hair using pixel blending and shadow buffers. *The Journal of Visualization and Computer Animation*, 2:92–97, 1991.

[Max90] Nelson L. Max. Cone-spheres. In *SIGGRAPH '90 Proceedings*, volume 24, pages 59–62. ACM SIGGRAPH, Addison Wesley, August 1990.

[Neu97] Ivan Neulander. Rendering generalized cylinders using the A-Buffer. Master's thesis, University of Toronto, 1997. `http://www.dgp.toronto.edu/~ivan/research/thesis.pdf`.

[Ney95] Fabrice Neyret. A general multiscale model for volumetric textures. In *Proceedings of Graphics Interface '95*, pages 83–91, 1995.

[RB85] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH '85 Proceedings*, volume 19(3), pages 313–322. ACM SIGGRAPH, Addison Wesley, July 1985.

[RCI91] Robert E. Rosenblum, Wayne E. Carlson, and Edwin Tripp III. Simulating the structure and dynamics of human hair: Modelling, rendering and animation. *The Journal of Visualization and Computer Animation*, 2:141–148, 1991.

[SC88] Michael Shantz and Sheue-Ling Chang. Rendering trimmed NURBS with adaptive forward differencing. In *SIGGRAPH '88 Proceedings*, volume 22, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1988.

[Sil90] M. J. Silbermann. High-speed implementation of nonuniform rational B-splines. In *Curves and Surfaces in Computer Vision and Graphics*, pages 338–345. The International Society for Optical Engineering, August 1990.

[Whi83] T. Whitted. Anti-aliased line drawing using brush extrusion. *SIGGRAPH '83 Proceedings*, 17:151–156, July 1983.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78 Proceedings*, volume 12, pages 270–274. ACM SIGGRAPH, Addison Wesley, August 1978.

Figure 13: An example of high geometric detail that is efficiently captured with paintstrokes.

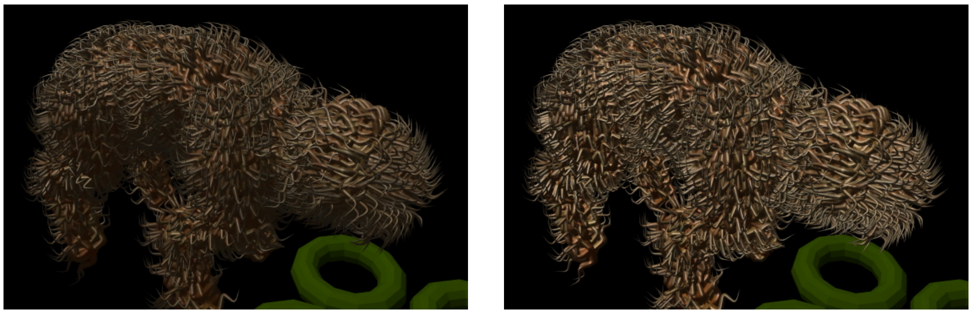
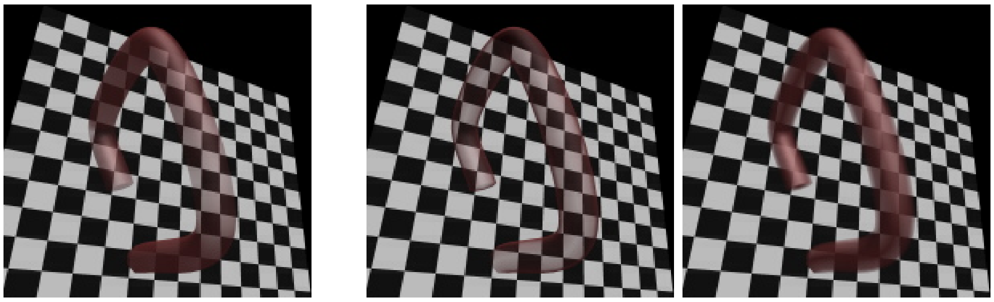


Figure 14: Image rendered with (left) and without (right) global shading.



(a) lengthwise

(b) breadthwise

Figure 15: Opacity variation.