Art-Based Rendering of Fur, Grass, and Trees

Michael A. Kowalski[◇*] *Lee Markosian*[◇] *J.D. Northrup*[◇] *Lubomir Bourdev*[†]
Ronen Barzel[‡] *Loring S. Holden*[◇] *John F. Hughes*[◇]

[◇]Department of Computer Science
Brown University
{mak,lem,jdn,lsh,jfh}@cs.brown.edu

[†]Advanced Technology Group
Adobe Systems
lbourdev@adobe.com

[‡]Pixar
ronen@pixar.com

Abstract

Artists and illustrators can evoke the complexity of fur or vegetation with relatively few well-placed strokes. We present an algorithm that uses strokes to render 3D computer graphics scenes in a stylized manner suggesting the complexity of the scene without representing it explicitly. The basic algorithm is customizable to produce a range of effects including fur, grass and trees, as we demonstrate in this paper and accompanying video. The algorithm is implemented within a broader framework that supports procedural stroke-based textures on polyhedral models. It renders moderately complex scenes at multiple frames per second on current graphics workstations, and provides some interframe coherence.

CR Categories and Subject Descriptors: I.3.3: Computer Graphics: Picture/Image Generation; line and curve generation; bitmap and frambuffer operations; I.3.5 Computer Graphics: Computational Geometry and Object Modeling: curve, surface, solid, and object representations; I.3.7: Three-Dimensional Graphics and Realism: Color, shading, shadowing, and texture. **Additional Key Words:** Non-photorealistic rendering, graftals, procedural textures.

1 Introduction

Any art student can rapidly draw a teddy bear or a grassy field. But for computer graphics, fur and grass are complex and time-consuming. Even so, the artist's few-stroke rendering may have greater persuasive or evocative power than the usual computer-graphics rendering.

How does the artist effectively communicate the teddy bear or grass? By rapidly creating an impression of free-form shape – difficult to do with conventional 3D modeling systems – and then drawing a few well-chosen strokes. This paper describes some of our efforts to expand the expressive power of 3D graphics by adopting techniques for depicting complexity from the centuries-old disciplines of art and illustration.

Three goals in our work on art-based graphics are to give the designer of a scene control over the style of rendering; to ease the burden of modeling complex scenes by treating the rendering strategy as an aspect of modeling; and to provide interframe coherence for the kinds of stylized renderings we've developed. Other goals, less directly related to the work in this paper, include the development of systems for rapidly creating free-form shapes [9], and the control of scene composition.

Our approach to creating complex expressive renderings is to target the kinds of images made by artists and illustrators and reproduce the effects and techniques we observe in those images. Much 2D art and illustration is created by making strokes on a flat surface (paper, canvas), so we have based our work on what we call "stroke-based textures." We started from the drawings of Theodore Geisel ("Dr. Seuss") [4, 5], in part because they are such an extreme departure from the domain of conventional computer graphics.

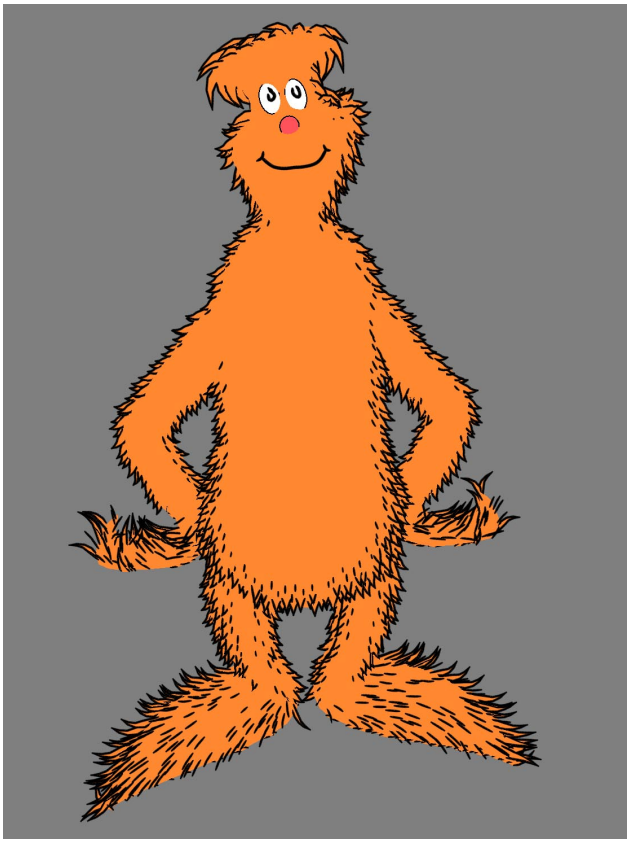


Figure 1 A furry creature, after Dr. Seuss. The fur is generated in a view-dependent way by a procedural stroke-based texture that places it near silhouettes, varying the style of the tufts according to how much the underlying surface faces the viewer.

[*]Currently at ATR Media Integration & Communications Research Laboratories, Kyoto, 619-0288, Japan.

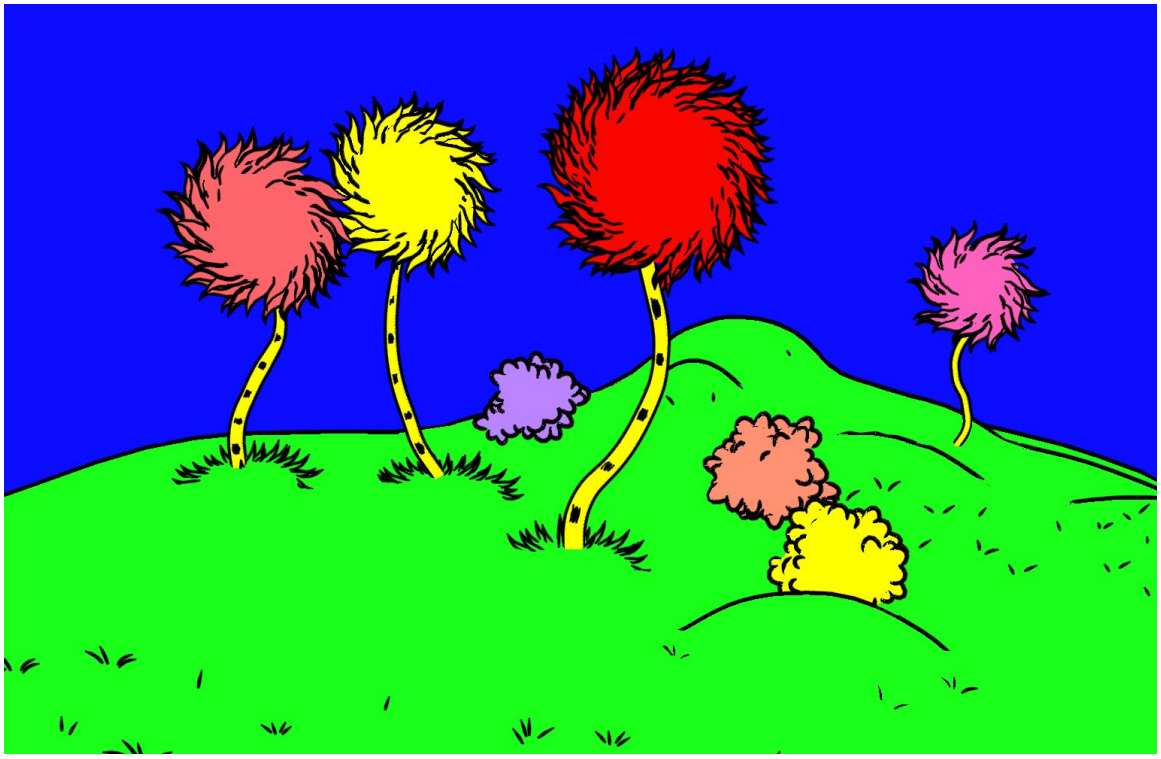


Figure 2 A more complex scene, again based on the style of Dr. Seuss. The grass, bushes, and truffula treetops are implemented with graftal textures that use the same basic algorithm to place graftals with a variety of shapes and drawing rules. The truffula tree trunks are drawn by stroke textures (not graftal textures) assigned to ribbon-like surfaces that always face the viewer. The treetops use the same type of graftal as seen in the previous figure, but with a different orientation rule: they always circulate clockwise around the treetop, no matter what the point of view. This cannot be modeled with any fixed geometry, of course.

There are two main research challenges: the development of algorithms and a software framework in which procedural stroke-based textures can be rendered, and the development of a user interface (within the context of a free-form modeling system) that allows a designer to assign and customize such procedural textures. In this paper, we describe our approach to the first of these challenges.

We have developed a system to generate and render stroke-based textures that mimic the styles of two artists, and a framework for generalizing this to other techniques. The system renders the images shown in this paper at several frames per second on a Sun Sparc Ultra 2 model 2/300 with Creator 3D graphics, and even faster on a high-end PC. Our main contributions are the system architecture, the (partial) temporal coherence of the texture elements, and the particular methods used to mimic Dr. Seuss's and Geoffrey Hayes's [7] styles.

2 Prior Work

Using art as a motivation for computer graphics techniques is not new, and our work builds on the efforts of many others. Fundamental to our ideas are the particle systems of Reeves [12, 13], which he used to create trees, fireworks, and other complex imagery from relatively simple geometry. Alvy Ray Smith's later use of particles, together with recursively defined L-systems that he called "graftals," extended this to more biologically accurate tree and plant models [15]. His "Cartoon Tree" is a direct precursor to the work in this paper. Graftals have since come to be described more generally: according to Badler and Glassner [1], "Fractals and graftals create surfaces via an implicit model that produces data when requested." We use the word "graftal" in this much more general sense.

We use a modified version of the "difference image" stroke-placing algorithm of Salisbury *et al.* [14] to place procedural texture elements at specific areas of the surface. Winkenbach and Salesin [17] described the use of "indication" (showing a texture on part of an object) in pen-and-ink rendering. And Strothotte *et al.* [16] extensively discuss the use of artistic styles to evoke particular effects or perceptions.

At a more mechanical level, Meier's work on particle-based brush strokes [11] was a major inspiration in two ways: first, her use of particles to govern strokes that suggest complexity in her Monet-like renderings showed that not all complexity need be geometric; second, the fixed spacing of the particles on the objects, which limited how closely one could zoom into the scene, inspired us to seek a similar but hybrid screen/object space technique.

The present work builds on our earlier efforts [10] to produce nonphotorealistic effects at interactive frame rates. One limitation of our earlier system was that it supported just one "style" at any given time – applying the style equally to every object in the scene. A more flexible system would allow the designer of a virtual scene to assign different nonphotorealistic "textures" to different surfaces within the scene. The framework we describe in the next section makes this possible. For instance, the fur texture on the creature in figure 1 is applied over most of the body but not the face, even in profile. Our other images show more examples of the selective use of distinct nonphotorealistic textures applied to objects in a scene according to what each represents.

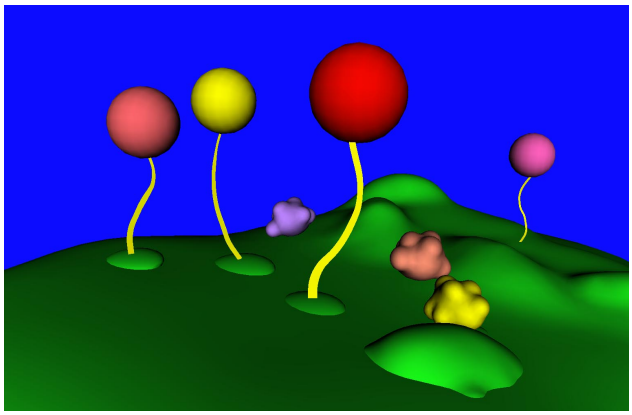


Figure 3 The same scene as in figure 2 rendered without graftal textures or the stroke-based textures on the truffula trunks.

3 Software Framework

Our procedural stroke-based textures are implemented within a general system for rendering polyhedral models using OpenGL [2]. In our system models are divided into one or more surface regions (called *patches*), to each of which the user can assign one or more procedural textures (called *textures*) – although just one is active at a time. The "procedure" that defines a texture needn't be complicated – many simply draw their patch in some conventional style (e.g., smooth-shaded or wireframe). One of our textures performs Floyd-Steinberg dithering [3]. Others perform a variety of hatching effects.

An important component of the system is the provision of *reference images*. These are off-screen renderings of the scene, subsequently read from frame-buffer memory to main memory and made available to the procedural textures. We currently use two kinds of reference images: a *color reference image* and an *ID reference image*.

To prepare the color reference image, the active texture of each patch is asked to render into it in some appropriate way, depending on how the texture will use the image. For example, the graftal textures described in the next section use the color reference image in a special way to decide where to draw tufts of fur, grass, or leaves. For the ID reference image, triangles (or edges) are each rendered with a color that uniquely identifies that triangle or edge. Lighting and blending are disabled so that the colors are preserved exactly.

After the ID reference image is prepared, all of its pixels are checked in one pass: when a pixel contains the ID of a triangle or edge, that pixel location is stored in a list on the patch that contains the triangle or edge. Later, the active texture of the patch can access the list of pixel locations in its main rendering loop. For example, the dithering texture simply runs the Floyd-Steinberg algorithm on the pixels of its patch.

The ID reference image can be used to determine the visibility of a point on a known triangle – the details of how to make this work robustly, even for triangles whose screen dimensions are less than a pixel, are beyond the scope of this paper (see [8]). If the triangle belongs to a patch of a convex surface, a simple test can be used: If the point (in screen space) is more than one pixel from the boundary of the patch, and also from any visible silhouette curve in the scene, then it is visible if and only if its triangle is front-facing and the value in the ID reference image at the point's screen position identifies a triangle of the same patch.

4 Graftal Textures

The textures described in this section place fur, leaves, grass or other geometric elements into the scene procedurally, usually to achieve a particular aesthetic effect (e.g., indicating fur at silhouettes but tending to omit it in interior surface regions). We'll call this class of textures *graftal textures*. They all share the same basic procedure for placing tufts, leaves, grass, etc., all of which we call *graftals*. The key requirements are that graftals be placed with controlled screen-space density in a manner matching the aesthetic requirements of the particular textures, but at the same time seem to "stick" to surfaces in the scene, providing interframe coherence and a sense of depth through parallax.

4.1 Placing graftals with the difference image algorithm

To meet these requirements, we have adapted the "difference image" algorithm (*DIA*) used by Salisbury *et al.* [14] to produce pen-and-ink-style drawing from grayscale images. Their algorithm controls the density of hatching strokes in order to match the gray tones of the target image. For each output stroke drawn, a blurred image of the stroke is subtracted from a "difference" image (initially the input image). The next output stroke is placed by searching in the difference image for that pixel most (proportionally) in need of darkening, and initiating a stroke there. The resulting image consists of marks whose density conveys the gray tones of the original.

The DIA meets our first requirement of placing marks (or in our case, graftals) with a controlled screen-space density. To control graftal placement according to a particular aesthetic requirement, each graftal texture simply draws its patch into the color reference image so that darker tones correspond to regions requiring a denser distribution of graftals. We call the result the *desire image*, and the value at a pixel in that image measures the *desire* that graftals be placed there. For example, to render the furry creature in figure 1, the reference image is drawn darker near silhouettes – easily done by placing a point light near the camera position. Also, some regions (e.g., the feet) can be explicitly darkened by the designer to promote a greater density of graftals there.[1]

To meet the requirement that graftals appear to stick to surfaces in the scene, we must convert the 2D screen position of a graftal (assigned to it by the DIA) to a 3D position on some surface. This is achieved in $O(1)$ time (per graftal) by using the ID reference image to find the triangle (and the exact point on the triangle with a ray-test) corresponding to a given screen position.

This now allows graftals to be distributed over surfaces in the scene to achieve a desired screen-space density – for a single frame. To create some interframe coherence, we modify the algorithm:

- In the first frame, graftals are placed according to the DIA.
- In each successive frame, the graftal texture first attempts to place the graftals from the preceding frame.
- Then, when all the "old" graftals have been considered for placement and accepted or rejected, the graftal texture executes the DIA to place new graftals into the scene as needed.

An existing graftal may fail to be placed in a frame for two reasons: (i) the graftal is not visible (it is occluded or off-screen); (ii) there is insufficient desire in the desire image at the graftal's screen position. This can happen if the original desire value at the graftal's screen position was small (e.g. the graftal is far from a silhouette).

[1] To further encourage drawing near silhouettes, we filter the desire image, replacing each desire value d with $2d - d^2$, where d ranges from 0 (no desire) to 1 (maximum desire).

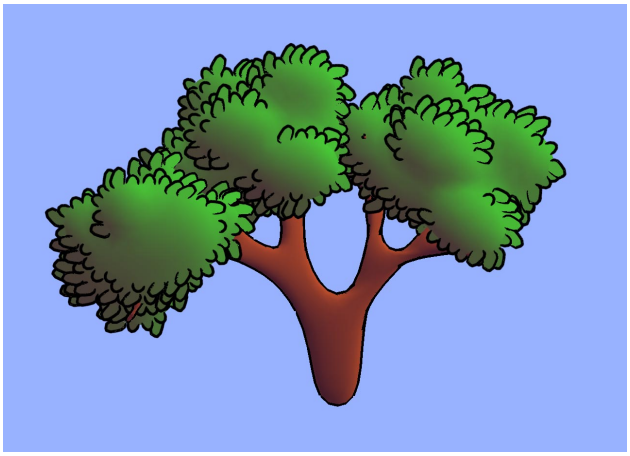


Figure 4 A tree rendered in the style of Geoffrey Hayes [7]. The leaves are drawn with graftals based on OpenGL triangle fans, rather than the triangle strip-based type of graftal shown in figure 5. The interior is shaded with the technical-illustrator shader of Gooch *et al.*[6].

It can also happen if the camera has zoomed out and the graftal's neighbors, now closer to it in screen space, have already subtracted the available "desire" in the vicinity. When a graftal fails to be placed in a frame, it is discarded. Otherwise it updates its attributes (as described below) and is drawn.

We make a final modification to the DIA: we use a bucket-sort data structure to find the pixel with the greatest desire. This key step can be completed in $O(1)$ time rather than the $O(\log(n))$ quad-tree method of Salisbury *et al.*, where n is the number of pixels.[2] This lets the algorithm run at interactive speeds on simple scenes.

4.2 Subtracting the blurred image

When a graftal is placed in the scene (either initially or in subsequent frames) it subtracts a blurred "image" of itself from the difference image. For this, graftals are treated as points with a given (variable) screen size, so the blurred image is just a Gaussian dot.

Pixels in the desire image are encoded with values ranging from zero (no desire) to one (maximum desire). Each graftal has an associated "volume" that determines how much total "desire" it subtracts from the desire image. This volume is proportional to the graftal's approximate screen space area. Intuitively, a visually large graftal subtracts a large volume, corresponding to a wide blurred dot: this eliminates desire in a wide region near the graftal, preventing others from being placed there.

Graftals can scale their geometry and volume so that they tend to maintain a desired screen-space size and relative density. For example, strictly adhering to the laws of perspective when zooming away from the model could result in graftals being drawn too small to be individually discernible. An artist might choose to draw them larger than they would realistically appear in this case. In any case, graftals that appear smaller in screen space should scale their volume accordingly in the DIA , or they will be placed too sparsely.

To perform such compensatory scaling, each graftal must keep track of its approximate screen space size. It does so by first converting its object-space length L to a screen-space measurement s in every frame (ignoring foreshortening). Then it chooses a scale factor r by which to multiply L as follows. As part of its definition, the graftal is given a desired screen space length d and corresponding volume v_0 (chosen by the user). At one extreme the graftal could take $r = d/s$, so that it always appears the same size on the screen regardless of distance. At the other extreme it could take $r = 1$, which would be strictly realistic. In our examples, we have taken a weighted average between the two extremes:

$$r = w(d/s) + (1 - w),$$

with weight $w = 0.25$. This approach moderates the degree to which the graftal scales with distance, providing a measure of resistance to change from its ideal size. Finally, the volume in each frame is calculated as $v = v_0(rs/d)^2$ to keep it proportional to the graftal's current screen size.

Let d_0 be the value in the desire image at the graftal's screen position, $\mathbf{x}_0$ (which has been verified as visible). Let $v > 0$ be the volume of the graftal. We seek a 2D Gaussian function g such that

$$g(\mathbf{0}) = d_0 \quad \text{and} \quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(\mathbf{x})\,dxdy = v.$$

This is given by $g(\mathbf{x}) = d_0 e^{-\pi d_0 |\mathbf{x}|^2 / v}$.

The function g has infinite support, but outside some radius its values are negligible. We set the "minimum usable desire" m to be the smallest value we can represent in the 8 bits we use to store desire. Then $g(\mathbf{x}) < m$ when $|\mathbf{x}| > (\log(d_0/m)v/(\pi g_0))^{1/2}$. This last value is the radius beyond which we need not subtract g from the desire image. We thus subtract $g(\mathbf{x} - \mathbf{x}_0)$ from pixels in the desire image whose distance from $\mathbf{x}_0$ is less than this value.

As the graftal subtracts its Gaussian from the desire image, it records the total desire subtracted. (It can't subtract more from a pixel than is stored there.) When all goes well, this quantity should equal the volume, v (ignoring discretization errors and the small portion of the Gaussian outside the maximum radius above). If the total is less than v, the graftal may draw itself with a reduced level of detail. If the total is too low (below 0.5 in all our examples), the graftal reports failure to its texture and is removed from the scene. To avoid "popping" when graftals appear and disappear, they may initially be drawn with reduced detail, quickly increasing to full detail over a short time, and reversing the process when they are removed. This has its limitations, though, as we discuss below.

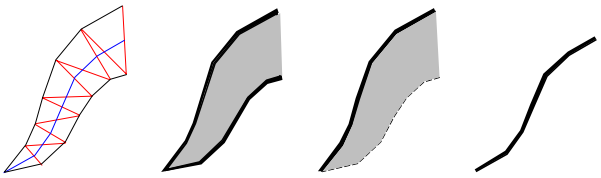


Figure 5 A fur graftal is based on a planar polyline and table of widths, used to construct a GL triangle strip (a). The graftal can render itself in three ways: It can draw a set of filled polygons with strokes along both borders (b) or just one (c); or it can draw just the spine (d).

4.3 Details of fur graftals

A fur graftal – the kind used for the furry creature in figure 1, and for the truffula tufts and grassy mounds at the base of the trees in figure 2 – is not particularly complex. It is based on a flat tapering shape by a gradually reducing width about a central spine (see figure 5). The central spine is a planar polyline, and the taper widths

[2] We thank Ken Lao for suggesting this idea.

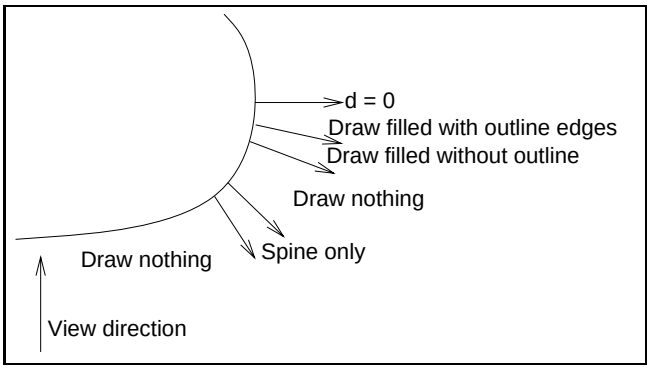


Figure 6 The dot product d of the view vector and surface normal determines a graftal's drawing style. In figure 1 the "Draw filled without outline" region is empty: we transition directly from "filled with outline" to "draw nothing."

are recorded in an array. For the model in figure 1, just a few taper widths were assigned; for the truffula tufts there were about seven. The shape of the central spine was drawn on graph paper and entered by hand.

After being placed with the DIA , each fur graftal determines how to orient and draw itself by computing the dot product d of the unit view vector $\vec{v}$ with the unit normal $\vec{n}$ to the underlying surface at the graftal's object-space position (see figure 6). For varying values of d, the tuft may be drawn filled, filled with one edge, filled with both edges, as a spine only, or not at all. For the fur in figure 1, we draw just the spine for $-0.75 < d < -0.6$. We draw the filled tuft with both edges for $-0.55 < d < 0$. Other schemes are possible, and we believe that adjusting the thresholds and drawing styles during "fade-in" and "fade-out" might help smooth these transitions.

Finally, the fur graftals are oriented to face the camera – that is, to lie in the plane containing the underlying surface normal and most nearly orthogonal to the view vector. They're placed so that in general they bend down. This behavior can be modified (as in the truffula tufts) so that they point clockwise, or so that they follow directions that have been "painted" onto the graftal texture's patch, as in the feet in figure 1.

5 Results and Future Work

Our system can produce scenes that evoke a remarkable sense of complexity, in a style that's new to 3D graphics, and at interactive rates. Figures 2 and 4 show the kinds of results that can be achieved with graftal textures. In each case the underlying geometry was simple to produce, yet the renderings have an expressiveness often lacking in computer graphics imagery. With our system, even the truffula scene can be rendered at several frames per second on a high-end PC.

The accompanying video[3] shows our system in action. It includes sequences captured in real-time and animation sequences rendered off-line and played back at significantly higher frame rates. The problem of poor frame-to-frame coherence stands out most noticeably in the latter case. Graftals that persist from frame to frame maintain geometric coherence (if we simply redistributed graftals at every frame, the flicker would be overwhelming); unfortunately, the DIA has no inherent interframe consistency, so it's easy for a graftal to be "crowded out" in one frame, replaced in the next, and so on, causing the flickering artifacts that are so noticeable in the video.

[3] See the Siggraph 99 Conference Proceedings Video Tape.

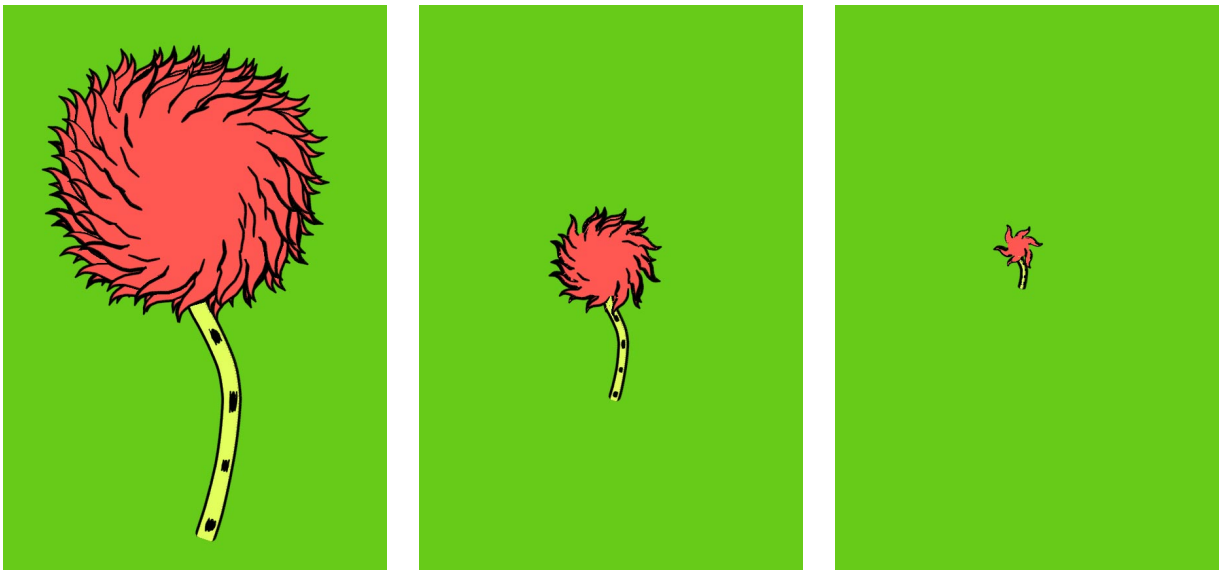
We have recently begun experimenting with some strategies to address this problem. One possibility is to make much greater use of fading and alpha blending when introducing graftals into the scene and taking them out. The degree to which this approach is usable depends quite a lot on the specific style being targeted. Fading in a large yellow truffula tuft outlined in black against a blue sky may be just as jarring as introducing it suddenly; but fading in semi-transparent blades of grass rendered with a watercolor-like effect over green terrain might seem perfectly acceptable.

One problem we have encountered in our early experiments with fading in tufts of fur like those in figure 1 occurs when all the tufts along a silhouette are newly introduced, and thus nearly transparent. The model then appears (briefly) to be missing its fur along that silhouette. A possible solution that we have not yet implemented is to maintain a separate population of tufts drawn on back-facing surfaces. This requires an auxiliary ID reference image prepared with front-facing triangles culled. It also requires two separate calls to our modified DIA each frame – one to place front-facing tufts, another to place back-facing ones. The point is that tufts emerging into view from behind a silhouette (as the object turns) would already be drawn and thus would not "pop in." Each frame might take twice as long to render – possibly a worthwhile trade-off if the resulting animations are significantly more watchable.

Another strategy we have experimented with is to use *static* graftals (see figure 7). With this approach, graftals are assigned fixed positions on the surface, rather than being generated each frame as needed. They still draw in a view-dependent way – those far from a silhouette, say, may not draw at all. This works quite well as long as the camera does not zoom out too far: in that case the graftals are drawn too densely in screen space. We can overcome this by assigning graftals several levels of priority – say numbered 0 through 2. Each level is distributed evenly over the surface, with those in a given level outnumbering those in the next lower level by a factor of about four. In a given frame, every graftal at level 0 "draws" itself view-dependently (possibly not at all if far from a silhouette). Each also subtracts its blurred image from the desire image, as in the DIA , and measures its success rate. If, collectively, this rate is high enough, the next level is given the chance to draw, and goes through the same procedure to decide whether the last level should also have the chance to draw. This strategy is suitable for localized objects -- we have tested it on versions of the truffula treetops – but not for landscapes where the choice of what level graftal to draw must vary over the surface according to distance from the camera. Our preliminary results indicate the effectiveness of this strategy. We demonstrate this in the accompanying videotape. Figure 7 shows a truffula tree top with static graftals drawn at three levels of detail.

6 Acknowledgments

We thank Michael LeGrand for creating the "furry creature" model. Thanks also to Andy van Dam and the Graphics Group, and to our sponsors: the NSF Graphics and Visualization Center, Advanced Network and Services, Alias/Wavefront, Autodesk, IBM, Intel, Microsoft, National Tele-Immersion Initiative, Sun Microsystems, and TACO. Lee Markosian received support for his graduate education from Intel through the Intel Foundation Ph.D. Fellowship program.



(a) (b) (c)

Figure 7 A truffula treetop with static graftals organized in a three-level hierarchy. When the camera is close, all three levels are drawn (a). As the camera zooms out, only two levels are drawn (b), and finally just the base level is drawn (c).

References

[1] Norman I. Badler and Andrew S. Glassner. 3D object modeling. In *SIGGRAPH 97 Introduction to Computer Graphics Course Notes*. ACM SIGGRAPH, August 1997.

[2] OpenGL Architecture Review Board. *OpenGL Reference Manual, 2nd Edition*. Addison-Wesley Developers Press, 1996.

[3] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 2nd edition, 1992.

[4] Dr. Seuss (Theodor Geisel). *The Lorax*. Random House, New York, 1971.

[5] Dr. Seuss (Theodor Geisel). *The Foot Book*. Random House, New York, 1988.

[6] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH 98 Conference Proceedings*, pp. 447–452. ACM SIGGRAPH, July 1998.

[7] Geoffrey Hayes. *Patrick and Ted*. Scholastic, Inc., New York, 1984.

[8] Lee Markosian. *Art-based Modeling and Rendering for Computer Graphics*. PhD thesis, Brown University, November 1999 (expected completion).

[9] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John Hughes. Skin: A constructive approach to modeling free-form shapes. In *SIGGRAPH 99 Conference Proceedings*. ACM SIGGRAPH, August 1999.

[10] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-time nonphotorealistic rendering. In *SIGGRAPH 97 Conference Proceedings*, pp. 415–420. ACM SIGGRAPH, August 1997.

[11] Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH 96 Conference Proceedings*, pp. 477–484. ACM SIGGRAPH, August 1996.

[12] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[13] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH 85 Conference Proceedings*, pp. 313–322. ACM SIGGRAPH, July 1985.

[14] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97 Conference Proceedings*, pp. 401–406. ACM SIGGRAPH, August 1997.

[15] Alvy Ray Smith. Plants, fractals and formal languages. In *SIGGRAPH 84 Conference Proceedings*, pp. 1–10. ACM SIGGRAPH, July 1984.

[16] T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey. How to render frames and influence people. In *Computer Graphics Forum*, volume 13, pp. 455–466. Eurographics, Basil Blackwell Ltd, 1994. Eurographics '94 Conference issue.

[17] Georges Winkenbach and David H. Salesin. Computer–generated pen–and–ink illustration. In *SIGGRAPH 94 Conference Proceedings*, pp. 91–100. ACM SIGGRAPH, July 1994.