Real-Time Fur over Arbitrary Surfaces

Jerome Lengyel
Microsoft Research
http://research.microsoft.com/~jedl

Emil Praun
Princeton University
http://www.cs.princeton.edu/~emilp

Adam Finkelstein
Princeton University
http://www.cs.princeton.edu/~af

Hugues Hoppe
Microsoft Research
http://research.microsoft.com/~hoppe

Abstract

We introduce a method for real-time rendering of fur on surfaces of arbitrary topology. As a pre-process, we simulate virtual hair with a particle system, and sample it into a volume texture. Next, we parameterize the texture over a surface of arbitrary topology using "lapped textures" — an approach for applying a sample texture to a surface by repeatedly pasting patches of the texture until the surface is covered. The use of lapped textures permits specifying a global direction field for the fur over the surface. At runtime, the patches of volume textures are rendered as a series of concentric shells of semi-transparent medium. To improve the visual quality of the fur near silhouettes, we place "fins" normal to the surface and render these using conventional 2D texture maps sampled from the volume texture in the direction of hair growth. The method generates convincing imagery of fur at interactive rates for models of moderate complexity. Furthermore, the scheme allows real-time modification of viewing and lighting conditions, as well as local control over hair color, length, and direction.

Additional Keywords: hair rendering, lapped textures, volume textures.

1. Introduction

A distinguishing characteristic of mammals is that they have hair. For many computer graphics applications, a sense of immersion requires the presence of realistic virtual creatures. Unfortunately, generating convincing imagery of people and animals remains a challenging problem, in part because the hair often looks artificial. This paper presents a method for rendering realistic fur over surfaces of arbitrary topology at interactive frame rates. In this paper, we use the terms *fur* and *hair* interchangeably.

Perhaps the most effective imagery of fur is due to Kajiya and Kay [4], who ray-traced a model with explicit geometric detail represented as volume textures. In the computer animation industry, fine geometric modeling of fur led to convincing artificial mammals, such as the lemurs in Disney's *Dinosaur* [1]. Another example is the dog fur in *101 Dalmatians* [2], which was represented using a stochastic (rather than geometric) model. However, rendering hair is computationally expensive, and the majority of the methods to date are too slow for interactive use (see Thalmann et al. [10] for a survey). In an interactive setting, Van Gelder and Wilhelms [11] showed that various parameters of fur can be manipulated in real time. However, their renderer is limited to drawing polylines, which limits the number of hairs that can be drawn per frame as well as the realism of the result.

Recent work by Meyer and Neyret [6][7] showed that rendering volume textures at interactive rates is possible by exploiting graphics card hardware. Lengyel [5] subsequently developed a similar method optimized for the specific case of fur. The method renders a furry surface as a series of concentric, semi-transparent, textured *shells* containing samples of the hair volume. By exploiting conventional texture mapping, the method allows interactive rendering of furry models represented by thousands of polygons.

Our approach is based on the shell method and thus achieves interactive frame rates. However, in this project, we address three limitations of previous work. First, the shell method requires a global parameterization of the surface. While this is easy to achieve for special cases (such as a torus or disc), it prevents application of this method to a surface of arbitrary topology, without first cutting the surface into pieces and rendering them separately. Second, the method requires a significant amount of texture memory, because each shell needs a separate texture covering the whole surface and these textures must be large enough to resolve individual hairs. Finally, the shell method provides an effective approximation to volume textures only when the viewing direction is approximately normal to the surface. Near silhouettes where shells are seen at grazing angles, the hair appears to be overly transparent, and gaps become evident between the shells.

In our work, we address the first two limitations of the shell method – parameterization and texture memory size – by using the *lapped textures* of Praun et al. [8]. Lapped textures cover a surface of arbitrary topology by repeatedly pasting small patches of example texture over the surface (Figure 1). Because the surface is covered by a collection of patches, we only need local parameterizations within the individual patches, as opposed to a global parameterization. Furthermore, the many patches instantiated over the surface can all share the same texture, thus greatly reducing texture memory overhead.

The third limitation – silhouettes – is not specific to the shell method. In interactive settings such as games, designers use low-polygon-count models to maintain high frame rates. Detailed textures help to embellish these coarse models, *except* at the silhouette where tangent discontinuities detract from the visual quality. Furthermore, in the specific case of fur, the visual cues at the silhouette play a critical role in perceiving the characteristics of the fur. Polygonal artifacts may be alleviated by using higher resolution models, or by clipping to a high-resolution 2D contour as described by Sander et al. [9]. To address the silhouette problem, we introduce a scheme for rendering textured *fins* normal to the surface near silhouettes. The fin textures are created using the same volumetric model for hair as the shell textures, but sampled in a different direction more appropriate for oblique viewing. Alternatively, an artist may create a fin texture directly.

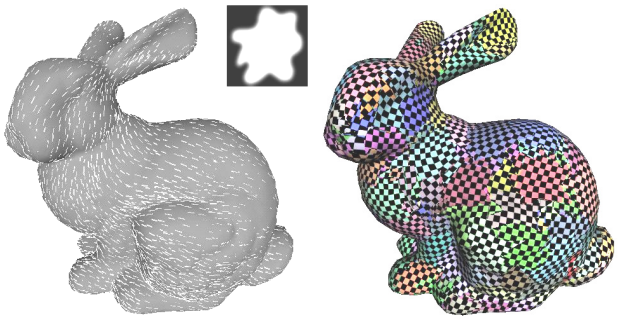
We offer interactive control over local and global properties of the fur, such as its direction, length, or color. For example, the user may globally adjust the angle of the hair with respect to the surface or may locally adjust the hair direction using a "combing tool".

The principal contributions of this work are: (1) integrating the shell method with lapped textures to allow for arbitrary topology and to reduce texture requirements; (2) improving the visual quality of silhouettes by rendering fins; and (3) demonstrating interactive local and global control over hair properties.

2. Approach

Here we present a brief overview of our approach, followed by a more detailed explanation in Sections 2.1 through 2.4.

Our system takes as input a triangle mesh representing the creature and a parametric model for the hair. Prior to runtime, we perform two operations: *geometry preprocessing,* in which we compute the lapped texture patches parameterizing the surface; and *texture preprocessing,* in which we grow a geometric model of a patch of hair and sample it into the shell and fin textures. At runtime we render a series of textured, concentric shells, offset from the original surface, as well as several fins perpendicular to the surface, to better suggest hair near the silhouettes.



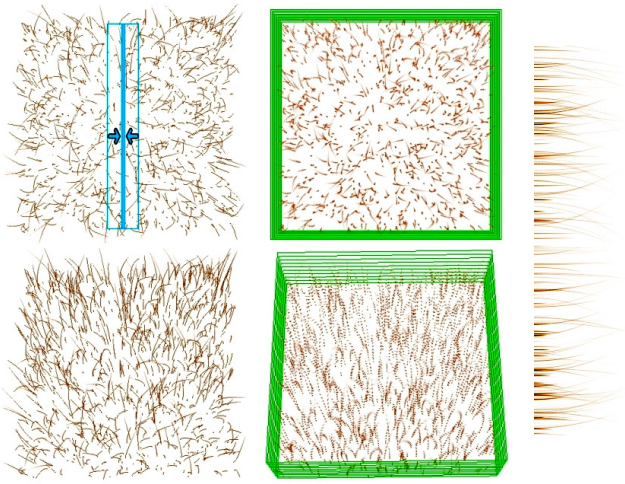
(a) Input: mesh and vector field; (b) Output: lapped patches

Figure 1. Geometry preprocessing: creation of lapped patches.

2.1 Geometry preprocessing

To build the patch parameterizations, we adapt the lapped textures scheme introduced by Praun et al. [8]. In brief, the method grows patches over random uncovered locations on the surface and locally parameterizes them onto the texture domain using a fast optimization process. The resulting collection of patches covers the surface, as shown in Figure 1.

In the original lapped textures construction, the seams between the patches are made less noticeable by letting the patches overlap, assigning the patches irregular boundaries, and alpha-blending these boundaries. For many types of hair textures (in particular, hair standing straight up), we find these measures to be unnecessary. Instead, we can simply let the patch boundaries correspond with edges in the mesh, and completely eliminate overlap between the patches. Because hair textures are stochastic, the resulting texture "discontinuities" along mesh edges are usually imperceptible. The main benefit of this *non-overlapping* texture parameterization is that each mesh triangle in a shell is rendered only once, rather than once for each patch overlapping it. Another benefit is that the texture patch need not have a boundary. Instead, the texture patch can be toroidal and therefore cover large surface patches through tiling. The results presented here are based on these non-overlapped parameterizations.



(a) geometric hair (b) shell textures (c) fin texture

Figure 2. Texture preprocessing. For illustration, the spacing between shell texture layers in (b) is exaggerated.

Even with a non-overlapped parameterization, we still benefit from the alignment of the patches to a global direction field over the surface. The consistent direction field helps hide the seams between patches when the hair has a preferred growth direction (e.g. the curly chimp hair in Figure 5). Also, we still benefit from the irregular "splotch" shape which gives each resulting patch an uneven mesh-edge boundary. For the special case of hair standing straight up, we can omit the global direction field and instead create an *isotropic* parameterization, where the local orientation of each patch does not align across adjacent patches [8].

2.2 Texture preprocessing

Using the method introduced by Lengyel [5], we create the shell texture based on the geometry of hair strands. The hair strands are generated through a particle system simulation. The simulation takes place within a small rectangular volume, with toroidal symmetry along the two skin axes so that the volume texture can be tiled. By varying the parameters, we can obtain hair that is straight, curly, sparse, dense, etc. (Figure 5).

Shell textures. To review, the shell method [5] creates a four-channel (RGBA) image for each layer of the shell model by overlaying a grid on the bounding box containing the hair strands. At each grid point we compute a color and opacity by intersecting a tall Gaussian filter with the geometric hair. The filter is tall because the inter-shell spacing is much greater than the grid spacing within each shell. For the innermost shell, all samples are assigned full opacity since this layer represents the skin surface.

To combine the shell method with lapped textures, we use the parameterization of each surface patch to access the layers of shell texture. Thus, for each layer, a texture patch is repeatedly instanced to produce a semi-transparent shell over the entire mesh. The process is repeated for each layer (Figure 4c). In our implementation, we typically sample 16 layers, and the texture patches have a resolution of 128×128.

Fin texture. We define a single instance of fin texture to be used on all the edges near silhouettes of the model. The texture is generated by compositing a slab of the geometric hair along an arbitrary surface tangent direction. The width of the slab (Figure 2, upper left, cyan box) determines the density of hair associated with the fin (Figure 2c). When rendering each fin, we randomly pick a texture coordinate interval that scales with edge length. The scaling factor is identical to that used to parameterize the mesh faces in the lapped texture construction. Ideally, every edge would have its own fin texture, sampled across the appropriate section of the geometric hair model, accounting for local mesh curvatures as well as the local density of fins. However, we found this to be unnecessary (since the visual quality of the models using a generic fin texture is good), and it would impose a drastic increase in texture storage. Since local density of fins depends on the model complexity, we manually select for each model the width of the texture slab composited to create the fin, choosing a density that seems visually appropriate for the given model and texture.

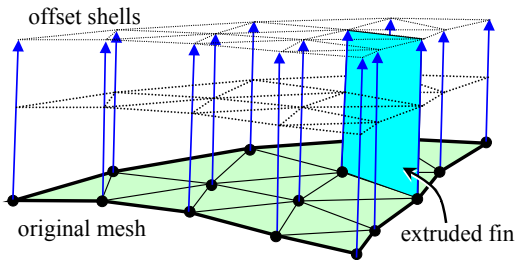


Figure 3. Runtime rendering: offset shells and extruded fin.

2.3 Runtime rendering

This section describes the three rendering steps (surface, fins, and shells) performed at runtime, in the order they are rendered.

Surface rendering (Figure 4a). For each frame, we first render an opaque version of the whole mesh, setting the Z-buffer. This includes both the innermost layer ("skin") of furry regions and the surface corresponding to the non-furry regions of the creature.

Fin rendering (Figure 4b). Next, we render textured fins. Each fin is a quadrilateral that is attached to a mesh edge and extends out along the surface normal direction (Figure 3). We have found that we obtain better results by rendering only the fins near the model silhouette. To maintain temporal coherence, we gradually fade-in the fins as they approach the silhouette, using the following scheme. For each mesh edge, we compute the dot product p between the fin normal and the infinite-viewer eye point. We then multiply the fin opacity by $\max(0, 2|p| - 1)$. We collect the fins with nonzero alpha and render them while testing, but not writing, the Z-buffer. (For comparison, Figure 4e shows the result of drawing all fins at full opacity.) Note that we render the fins corresponding to some edges that are invisible (i.e. just over the horizon), since the outer tips of such fins may actually be visible. The number of rendered fins is a small fraction of the total number of edges and is dominated by the faces in the shell layers. Thus, fin rendering has an relatively insignificant impact on the rendering performance of the system, yet it greatly improves the visual quality of the results.

Shell rendering (Figure 4c). Next, we render the offset shells from innermost to outermost (Figure 3). For each layer we composite all the patches of the (semi-transparent) lapped texture over what has already been rendered. During shell rendering, we both test and write the Z-buffer, and enable alpha-testing to avoid writing Z for pixels with zero coverage. We enable texture wrapping so that the shell texture can tile across large patches.

2.4 Rendering discussion

Hair lighting. We apply the shading method described in [5], which uses a modified Banks/Kajiya-Kay lighting model stored in a low resolution (32×32) texture-map lookup table:

$$\text{HairLighting}(u, v) = K_a + K_d(1 - u^2)^{P_d/2} + K_s(1 - v^2)^{P_s/2}$$
$$u = T \cdot L \quad // \ \cos(\text{angle}) = u; \ \sin(\text{angle}) = (1 - u^2)^{1/2}$$
$$v = T \cdot H \ .$$

T is the hair direction stored at each vertex. L is the light direction. H is the half-vector between the eye and the light. K_a, K_d, and K_s are the ambient, diffuse, and specular colors. P_d and P_s are the diffuse and specular powers. To compute u and v at runtime, T is stored as a set of texture coordinates, and the texture matrix is loaded with the current L and H vectors. We use the Banks self-shadowing approximation to darken the shells near the skin. In the current implementation, we use the same vector to store both the hair direction T and the shell-offset vector c_v (Section 3). This limits the effectiveness of the lighting to hair that is aligned mostly in the vertical direction. In the future, we plan to store both T and c_v at each vertex. Moreover, we hope to use a pixel shader with a hair-direction map that is filtered from the original hair geometry.

Visibility. Since the triangles associated with the shells and fins intersect each other in space, there does not exist a correct visibility order for rendering them. The shells offer the best visual approximation where the surface is oriented towards the viewer, whereas the fins look best near the surface silhouette. Our approach is to rely on the fin fading to locally control whether shells or fins appear more prominently.

Level-of-detail control. As the model recedes in the distance, we can reduce the number of rendered shells (Figure 4g and 4j). However, abruptly changing the number of shells can result in visual pops. We make these transitions visually smooth by blending between shell textures as follows. Given a set of shell textures, we composite together successive pairs of shells to form the *even* shells of the coarser level, while we leave the *odd* shells of the coarser level completely transparent. We repeat until only one shell remains. When the graphics hardware has finished transitioning to a coarser level over the entire model, we omit rendering of odd-numbered shells at the finer level. We determine the schedule of the blend based on the distance from the viewer to the model. The inter-level blending is performed on the volume texture patch before the full shell rendering.

Programmable vertex shaders. Within the coming year, commodity graphics hardware will have programmable vertex shaders. Such shaders will be ideal for accelerating shell rendering, since they can directly evaluate the geometric offsets between shells, in fact using only two additional shader instructions.

Overlapped parameterization. The original lapped textures covered a surface using overlapping patches, with patch boundaries defined at pixel granularity. Since patches were opaque, some faces were completely covered using a single patch (the topmost one covering the face). However, other faces had contribution from several patches. Since our hair textures are not opaque, we would need to render all faces with all patches that overlap them (not just the topmost). This would result in a slower frame rate and in hair density variations (Figure 4f). As mentioned in Section 2.1, we chose to extend the patches to the nearest edge boundaries, and to only render the topmost patch for each face. This basically amounts to producing a non-overlapping tiling of the surface. Since hair textures are stochastic in nature, visual masking tends to hide these polygonal discontinuities.

3. Interactive controls

In addition to traditional camera and lighting parameters, we also provide local interactive control over three aspects of the hair model — color, length, and direction. These attributes are specified as fields on the surface mesh. Currently, we associate to each mesh vertex a color and a vector c_v encoding length and direction.

Hair color. Using the graphics hardware, we linearly interpolate the vertex colors over each face and edge, in order to modulate the shell and fin textures. For example, see the dog in Figure 5. For finer control, one could instead specify color by painting into a texture atlas, as described by Hanrahan and Haeberli [3].

Hair length. In our prototype, we use by default 16 shell textures, spaced apart by one thousandth of the model diameter. We can adjust the length of the hair by varying the number and spacing of these layers. For local control, we offset the shell

layers at each vertex by a fraction of c_v. We have found that making the spacing between layers too large can lead to visible artifacts, for example straightening curly hair, or revealing the discrete nature of the layers. Thus, very long hair involves adding layers to the model, at the expense of rendering performance.

Hair direction. Hair can be combed by shearing the geometries of the shells and fins with respect to the base surface. Shearing is achieved by adding a tangential component to the vector c_v stored at each mesh vertex. A simple choice for this tangential direction is the direction field used when building the lapped texture (see Section 2.1). The user can then globally control the slant of the hair, adjusting from hair standing straight up to hair tilted at 45°. (At larger angles we lose the visual correlation of individual strands across layers.) As shown in Figure 4k and the accompanying video, local combing is done by dragging a circle to select an active region, and then dragging the selection in the plane of the current view to add a delta to the c_v of each selected vertex.

4. Results

Figure 5 and the accompanying videotape show a variety of furry models rendered in our prototype application. In these examples, the textures are sampled in 16 layers (except the dice which use 32 layers), where the image in each layer has a resolution of 128×128 pixels. The rendering rates shown in the table below are obtained on a AMD-K7 700MHz PC with an NVIDIA GeForce DDR 32MB card, using Microsoft DirectX 7. The dog is slower due to the extra bandwidth required by the per-vertex colors.

Model	bunny	cat	spider	dog	chimp	1die
# faces	5,000	5,000	5,000	5,000	4,910	1,450
# edges	7,547	7,500	7,500	7,500	7,365	2,400
# patches	306	252	338	192	228	246
½ layers (fps)	23	26	31	20	21	25
All layers (fps)	12	14	15	10	11	13

Table 1: Model complexities and rendering rates.

Note that the polygon counts for these models are low. We observe that the mottled nature of fur obscures tiny details in the models, thereby hiding objectionable tangent discontinuities along the low-resolution silhouettes.

The stored textures include the 16 layers of 128×128 shell texture and the single 512×64 fin texture. With full 32 bit/texel representation and mipmapping, these require a total texture memory footprint of 1.6 MB. Use of DXT5 texture compression reduces the memory requirement by a factor of 3, but makes the hair look thicker due to alpha compression artifacts. For the common case of hair standing straight up, all the shell textures can be encoded using a single image layer, for a total of only 200 KB (skin + shell + fin). In this case, an option is to taper the hair away from the skin by modulating the shell texture with an alpha ramp.

5. Summary and future work

We have shown that lapped textures and layered shells can be combined to effectively render hair. Together, they permit interactive high-resolution hair rendering over large, arbitrary surfaces using a moderate texture footprint. We introduced fin textures to improve the appearance of silhouettes, and showed that shells and fins integrate seamlessly. Finally, we presented techniques for level-of-detail transition and for user control of hair color, length, and direction.

This work suggests a number of areas for future research, such as:

Multiple hair models on the same creature. Real creatures have different kinds of hair. While it would be trivial to cut the model into separately rendered parts, a more interesting solution would allow a smooth transition from one kind of hair to another.

More interesting combing effects. According to the celebrated "hairy ball theorem", features such cowlicks, parts, and swirls are an unavoidable fact of life. Modeling such features may involve new representations, combing tools, and rendering techniques.

Shell multitexturing. It may be possible to render several shells in a single pass by "bumping" their texture coordinates according to vertex normals during a multitexturing combiner operation.

Programmable pixel shaders. As mentioned in Section 2.4, upcoming programmable vertex shaders will soon be able to evaluate both hair lighting and shell geometric offsets. Future programmable pixel shaders may be able to perform per-pixel lighting, which would be useful for wavy or curly hair patterns.

Hair dynamics. The hair could be moved according to physical simulation by shearing the layers. It may be possible to part the hair by allowing the shells to separate laterally.

References

[1] ESKURI, N. The art and technology of Disney's "Dinosaur". SIGGRAPH 2000 Course Notes.

[2] GOLDMAN, D. Fake fur rendering. Proceedings of SIGGRAPH 97, pp. 127-134.

[3] HANRAHAN, P. AND HAEBERLI, P. Direct WYSIWYG painting and texturing on 3D shapes. Proceedings of SIGGRAPH 90, pp. 215-223.

[4] KAJIYA, J. T., AND KAY, T. L. Rendering fur with three dimensional textures. Proceedings of SIGGRAPH 89, pp. 271-280.

[5] LENGYEL, J. Real-time fur. Eurographics Rendering Workshop 2000, pp. 243-256.

[6] MEYER, A., AND NEYRET, F. Interactive volume textures, Eurographics Rendering Workshop 1998, pp. 157-168.

[7] NEYRET, F. Modeling, animating, and rendering complex scenes using volumetric textures. IEEE Transactions on Visualization and Computer Graphics, 4(1), pp. 55-70, 1998.

[8] PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. Lapped textures. Proceedings of SIGGRAPH 2000, Computer Graphics, Annual Conference Series, pp. 465-470.

[9] SANDER, P., GU, X., GORTLER, S., HOPPE, H., AND SNYDER, J. Silhouette clipping. Proceedings of SIGGRAPH 2000, Computer Graphics, Annual Conference Series, pp. 327-334.

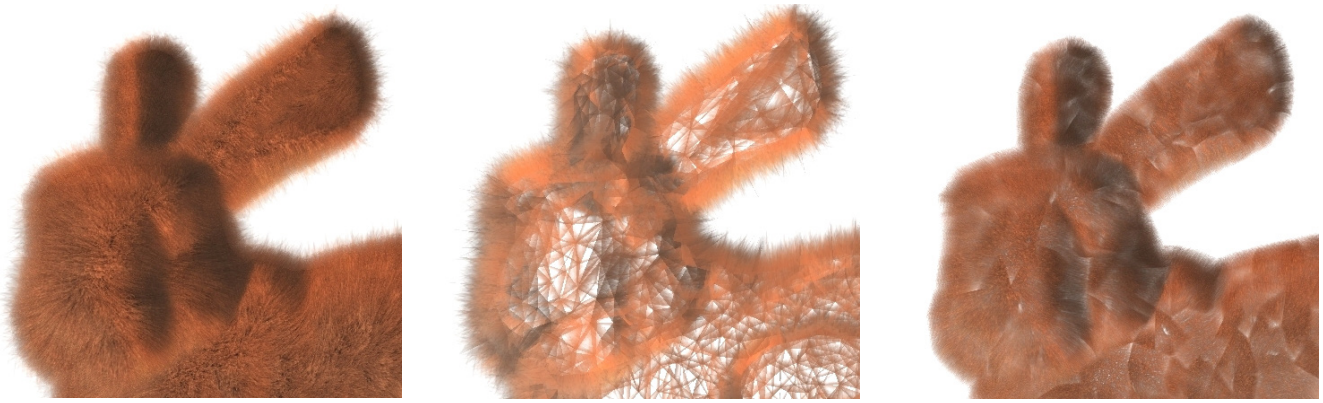
[10] THALMANN, N. M., CARION, S., COURCHESNE, M, VOLINO, P., AND WU, Y. Virtual clothes, hair and skin for beautiful top models. Computer Graphics International 1996.

[11] VAN GELDER, A., AND WILHELMS, J. An interactive fur modeling technique. Proceedings of Graphics Interface 1997.

Acknowledgements

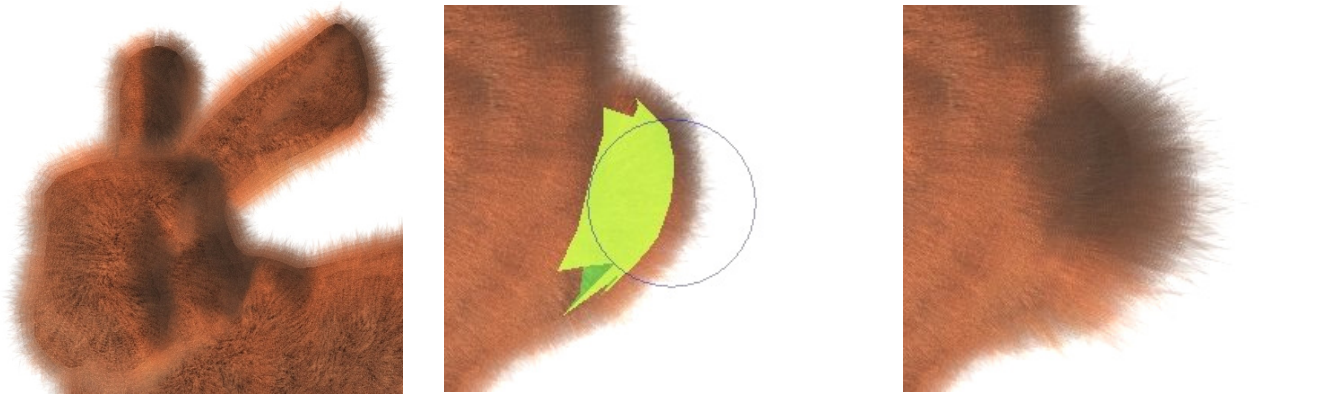
Scott Posch wrote the DX8 vertex shader for offsetting the shells on the fly. Tom Forsyth discussed potential tradeoffs between geometric layers and multistage texture layers. We appreciate the helpful suggestions given by Tom Funkhouser, Lee Markosian, and the anonymous reviewers. We thank Stanford University and Viewpoint Datalabs for the geometric models.

(a) surface (inner, opaque shell) (b) fins (alpha-blended) (c) shells (non-overlapped patches)



(d) final image (a+b+c) (e) fins (not alpha-blended) (f) shells (overlapped patches)

(g) 8-shell level of detail (h) globally lengthened hair (i) globally combed hair



(j) 4-shell level of detail (k) local selection (l) local combing

Figure 4. Overview of our fur rendering approach. (Images (e) and (f) are not direct results but are used to motivate choices in our approach.)

Figure 5. Furry objects. Flat-shaded meshes in the top row reveal the low triangle counts of the models. Renderings below use 16 shell layers (except dice which use 32). Close-ups appear in circles. The spider, Chia® cat, and chimp use fin textures sampled from the volume texture, whereas the other models use fins computed from hair that is longer and curlier than the hair used to create the shell textures.