Stochastic Simplification of Aggregate Detail

Robert L. Cook John Halstead Maxwell Planck David Ryu
Pixar Technical Memo #06-05a
Pixar Animation Studios

Abstract

Many renderers perform poorly on scenes that contain a lot of detailed geometry. The load on the renderer can be alleviated by simplification techniques, which create less expensive representations of geometry that is small on the screen. Current simplification techniques for high-quality surface-based rendering tend to work best with element detail (i.e., detail due to the complexity of individual elements) but not as well with aggregate detail (i.e., detail due to the large number of elements). To address this latter type of detail, we introduce a stochastic technique related to some approaches used for point-based renderers. Scenes are rendered by randomly selecting a subset of the geometric elements and altering those elements statistically to preserve the overall appearance of the scene. The amount of simplification can depend on a number of factors, including screen size, motion blur, and depth of field.

I.3.7 [Three-Dimensional Graphics and Realism]: Animation—Simplification

Keywords: Level of detail, stochastic sampling, simplification.

1 Introduction

Geometrically detailed scenes can overwhelm a renderer, increasing its memory requirements and severely degrading its performance. Usually, though, much of the detail is too small to be represented at the resolution of the rendered image, so that a version of the scene without this unrenderable detail can be substituted without perceptibly changing the image. Reducing detail manually [Crow 1982] can be prohibitively labor-intensive, so automatic simplification techniques are crucial for the efficient rendering of complex scenes.

Scenes can be complex because of element detail or aggregate detail. *Element detail* comes from having geometric elements that individually contain a lot of detail; *aggregate detail* comes from having a lot of elements, even if the individual elements are very simple. Existing simplification techniques have been mostly designed for element detail, but aggregate detail is becoming more prevalent as more procedurally generated models are used in animation and special effects. Consider for example the landscape in Figure 3, which is filled with plants like the one in Figure 1 stretching from the near distance to the far horizon. This scene contains over a hundred million leaves, and the unsimplified version is unrenderable with the RenderMan program.

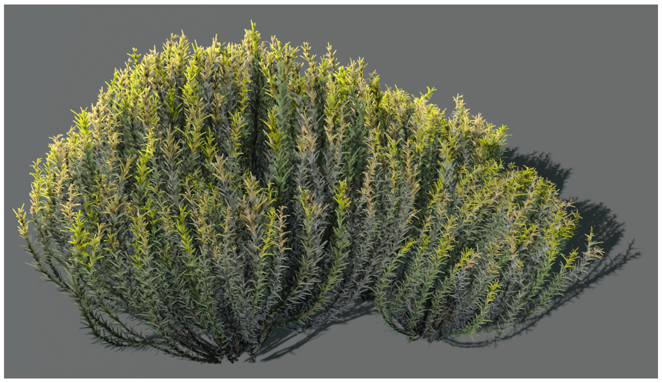
Figure 1: *A plant with 320,000 leaves.*

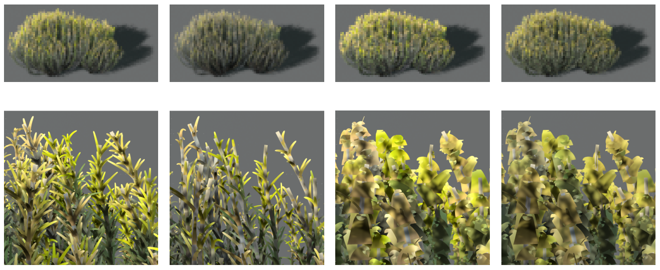
(a) (b) (c) (d)

Figure 2: *Distant views of the plant from Figure 1 with close-ups below: (a) unsimplified, (b) with 90% of its leaves excluded, (c) with area correction, (d) with area and contrast correction.*

The types of simplification techniques most relevant for film production are *surface-filtering methods* (such as mesh simplification), which create low-resolution geometry, and *texture-based methods* (such as impostors and volume textures), which create pixel arrays. For scenes like the one in Figure 3, we found that mesh simplification didn't help since the individual elements were already extremely simple. Impostors didn't help because the camera's point of view was changing and the models were animated. Volume texture approaches didn't help because the amount of detail required in the foreground made the storage requirements prohibitive.

It was scenes like this that led us to develop a new approach to simplification that can handle large numbers of small geometric elements. The new technique renders a scene using a randomly selected subset of the elements, altering them statistically to preserve the overall appearance of the scene. This is neither a surface-filtering method nor a texture-based method; it is a *stochastic method*. Stochastic methods have previously been used mainly for point-based rendering.

2 Previous Work

There is a large body of level-of-detail research with many techniques for creating simplified versions of a scene (see [Garland 1999] and [Luebke et al. 2002] for good surveys). This section briefly reviews a few representative techniques with an emphasis

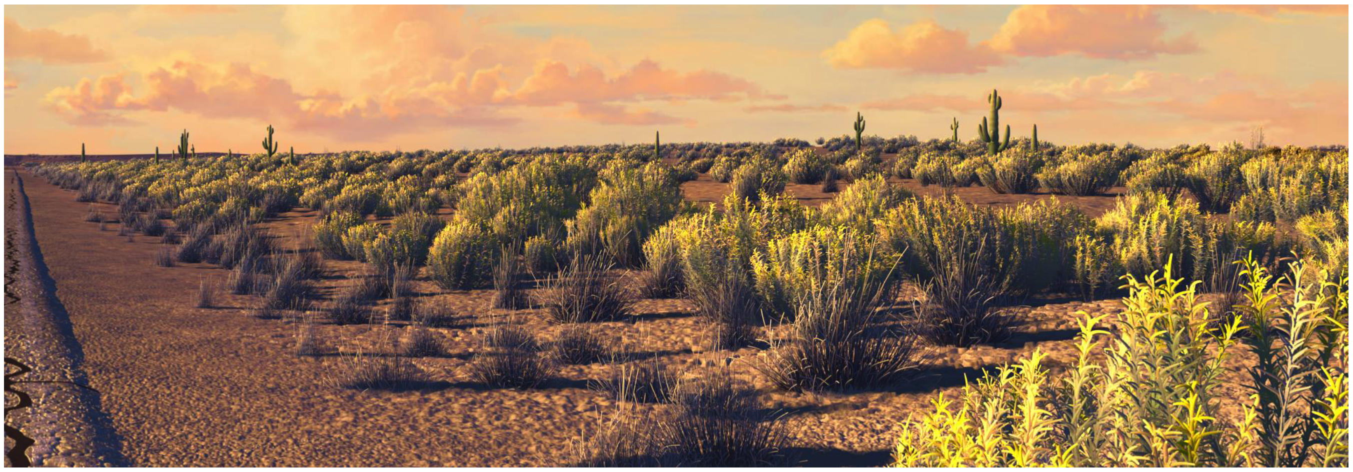
Figure 3: *A scene renderered with stochastic simplification that was unrenderable in RenderMan without simplification.*

on how they perform with aggregate detail. The types of techniques include adaptive generation, surface filtering, textures, and stochastic simplification.

Adaptive-generation methods are integrated into the model creation process and adjust the number of elements generated based on screen size. Simple models can be generated at render time [Reeves 1983], but complex models are more expensive to generate so they are usually pre-computed [Prusinkiewicz et al. 1994]; simplification involves pre-computing the model at a few different detail levels. Some very impressive images have been made using approximate instancing and combining rendered subscenes [Deussen et al. 1998]. These methods do well with aggregate detail, but their applicability is limited because they are integrated into specific model generation algorithms, whereas other methods operate on the resulting generated elements and are thus independent of the modeling algorithm. Moreover, many model generation algorithms are complex, and integrated level-of-detail calculations can make them even more complicated.

Surface-filtering methods create lower-resolution geometric versions of pre-generated objects. Most of the research in this area has focused on creating simplified surface representations; examples include progressive meshes ([Hoppe 1996]), multiresolution methods ([Lounsbery et al. 1997]), and simplification envelopes ([Cohen et al. 1996]). Some methods (such as [Luebke and Erikson 1997] and [Garland and Heckbert 1997]) are able to merge disjoint surfaces. [Erikson and Manocha 1999] uses error metrics stored in point clouds to preserve area and surface attributes as vertices are removed. Recent work by [Yoon et al. 2006] uses PCA analysis to compute a plane that approximates the triangles in each region of space and stores these planes and the average local surface attributes in a k-d tree for use in ray tracing.

Texture-based methods represent the simplified geometry using 2D or 3D textures. In general, they are able to handle aggregate detail better than surface-filtering methods. 2D methods such as impostors [Schaufler and Stürzlinger 1996], textured depth meshes [Wilson and Manocha 2003], and billboard clouds [Décoret et al. 2003] [Lacewell et al. 2006] work well in many situations, but their 2D nature can become evident if the camera is moving. This can be ameliorated by blending between multiple textures with different resolutions and different view angles, but this increases memory requirements and can still have artifacts. Moreover, our plants are procedurally animated so that every leaf is moving differently in every frame; this would require having a different texture per frame, which would be prohibitively expensive.

We found volume texture methods such as [Neyret 1998] to be more promising because they are not view dependent. Unfortunately, our detail and resolution requirements make the storage requirements prohibitive. To illustrate the magnitude of the problem, figure 17 in [Neyret 1998] uses an 18 MB low-resolution reference volume (256^3 voxels) to render a low-resolution image (768x512 pixels, 1 sample per pixel). A high-quality, high-resolution rendering that included objects close up on the screen would require several gigabytes. We got the best results with brick maps [Christensen and Batali 2004], but even with this sparse octree representation of volume textures the image quality was nowhere near acceptable even with the largest practical textures. Furthermore, although limited animation can be done by deforming a single reference volume ([Neyret 1995]), complex procedural animation requires multiple reference volumes, making the storage requirements even greater.

Stochastic simplification methods have mainly been used for point-based renderers. These renderers use a cloud of surface sample points which are rendered as simple primitives such as splats [Rusinkiewicz and Levoy 2000]. The density and splat size of the sample points depends on the amount of detail required; the sample points can be pre-computed and stored in a multi-resolution tree structure. [Dachsbacher et al. 2003] stored the sample point hierarchy in a sequential list. [Stamminger and Drettakis 2001] used an object-space hierarchy and generated the sample points adaptively based on screen size. [Wand et al. 2001] placed the sample points at randomly selected locations on the surface, and [Wand and Straßer 2002] used stratified sampling to build a point hierarchy from animation keyframes. [Deussen et al. 2002] used stratified sampling to determine the point sample locations. [Gobbetti and Marton 2004] created a multi-resolution hierarchy for unstructured point samples by randomly selecting uniformly distributed samples for each level of the hierarchy. [Kalaiah and Varshney 2003] used PCA analysis to extract a statistical model from a dense set of input points and used that model to stochastically generate the sample points. These methods can preserve area by adjusting the size of the splats used to render the points. [Pfister et al. 2000] [Wand and Straßer 2002] were able to preserve contrast by pre-filtering.

[Klein et al. 2002] used a stochastic approach to simplifying polygonal scenes by rendering a random subset of the scene elements. Their method is designed for interactive walkthroughs, however, and is inappropriate for high-quality rendering because it does not alter the elements in the subset, and as a result it does not preserve area or contrast. [Deussen et al. 2002] simplified line

a	element surface area
B	bounding box size of the object in pixels
b	B scaled so simplification begins at $b = 1$
c	color of an element
D	depth complexity of the object
h	value of b at which $\lambda = 1/2$
k	number of elements sampled per pixel
N	number of elements in the object
r	ratio of a to V
s	area scaling correction factor
t	size of transition region for fading out excluded elements
V	visible area of the object
x	position of an element in the rendering priority order
α	scaling factor used in variance reduction
λ	level of detail, fraction of elements to be included
σ^2	variance

Figure 4: *Some symbols used in this paper*

renderings by randomly reordering the lines, storing them in a list, reading just the first part of the list, and correcting the width of the rendered lines to preserve the overall projected area. This approach is similar to ours.

3 The Algorithm

Previous high-quality level-of-detail techniques for surface renderers determine the properties of the simplified representation by averaging the properties of neighboring elements in the original model. Surface-filtering methods use these averages to create less detailed geometric representations; texture-based methods use them to create arrays of pixels or voxels.

Our approach does not rely on averaging neighboring elements; instead, the simplified model is a randomly selected subset of the original elements. The properties of the selected elements are modified so that the statistical properties of the scene are preserved. Each selected element is modified independently of its neighbors.

The disadvantage of this independence is that the simplification is not as locally accurate because it ignores local variations that other algorithms can consider, including geometric properties like the local error of the simplified surface [Hoppe 1998] or local perceptual factors like whether the element is on a silhouette [Luebke and Hallen 2001] [Williams et al. 2003] (though there might be ways to partially account for local factors, as discussed in Section 5.) The advantage is that the performance scales well and the memory requirements are modest.

There are 5 aspects to making this approach work; each is discussed in detail below.

1. **Detail level.** Determining how many elements to exclude.
2. **Rendering priority.** Determining the order in which elements are excluded.
3. **Area preservation.** Altering the size of the included elements so the area of the object does not change.
4. **Contrast preservation.** Altering the shading of the included elements so the contrast of the image does not change.
5. **Smooth animation.** Making the excluded elements fade out instead of disappear abruptly.

3.1 Detail Level

For each rendered object, we need to determine the fraction of the elements to include during rendering. We call this the level-of-detail parameter λ. Many factors can be considered when determining λ; in this section, we only consider size and blur and ignore other possible factors such as contrast. We assume that λ is the product of these factors:

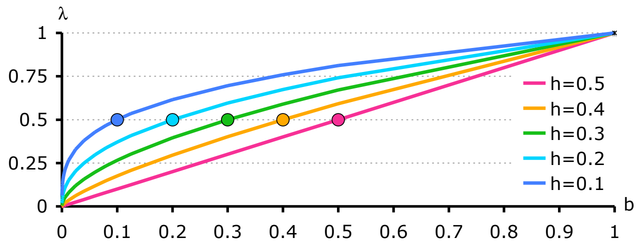

Figure 5: *λ as a function of b for a few different values of h. Half of the elements are used (i.e., $\lambda = 1/2$) at $b = h$.*

$$\boxed{\lambda = \lambda_{size}\lambda_{blur}} \tag{1}$$

Because more elements can be excluded as the object covers fewer pixels, λ_{size} depends on B, the size of the bounding box of the object in pixels. We control λ_{size} with two parameters: a bounding box size B_0 at which simplification begins and a bounding box size hB_0 at which half of the elements should be excluded (i.e., at which $\lambda_{size} = 1/2$), which provides control over how aggressively to simplify as the object gets smaller. B_0 should be where the shapes of individual elements are no longer discernible, usually when they are about the size of a pixel, but it also depends on the desired image quality. If we let $b = B/B_0$, then for $b < 1$ we set

$$\boxed{\lambda_{size} = b^{\log_h(1/2)}} \tag{2}$$

Note that for $h = 1/2$, this becomes $\lambda_{size} = b$ and the average number of included elements per pixel is constant. h should never be greater than $1/2$ because λ_{size} would decrease faster than b; this would mean using fewer elements per pixel as the object got smaller, and in order to preserve the overall object area those elements would have to get larger on the screen as the object got smaller.

We can also simplify more in regions that are blurred; this type of simplification is important since motion-blurred and out-of-focus elements are very expensive to render. The amount of simplification should depend on the amount of blur relative to the element size. We do not have a precise model for how λ_{blur} should change with the amount of blur, but we can use a formula similar to Equation 2, using blur amounts in place of b and h. Fortunately the exact formula is not critical.

3.2 Rendering Priority

To avoid artifacts during animation, the elements must be excluded in a consistent order. This *rendering priority order* should not be correlated with geometric position, size, surface normal, color, or other characteristics (e.g., excluding elements from top to bottom would be objectionable). Some objects are constructed in such a way that the rendering priority order can be determined procedurally, but we have found it more general and useful to determine the order stochastically. A simple technique is to assign a random number to each element, then sort the elements by their random numbers. This is usually sufficient in practice. Poisson disk sampling or stratified sampling can be used to ensure that similar elements (geometrically close, with similar normals, etc.) are not close to each other in the rendering priority order; this spreads out the visual effects of simplification during animation and allows somewhat more aggressive simplification.

When N, the number of elements in the object, is large, the time spent doing even trivial rejects can be significant, so it is important

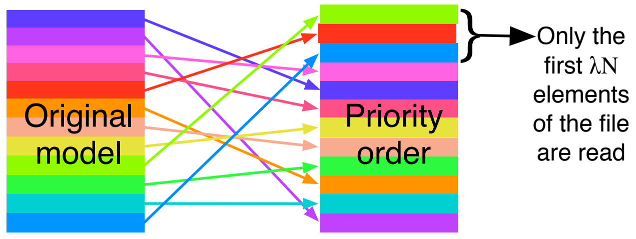

Figure 6: *Elements can be reordered in a preprocessing step.*

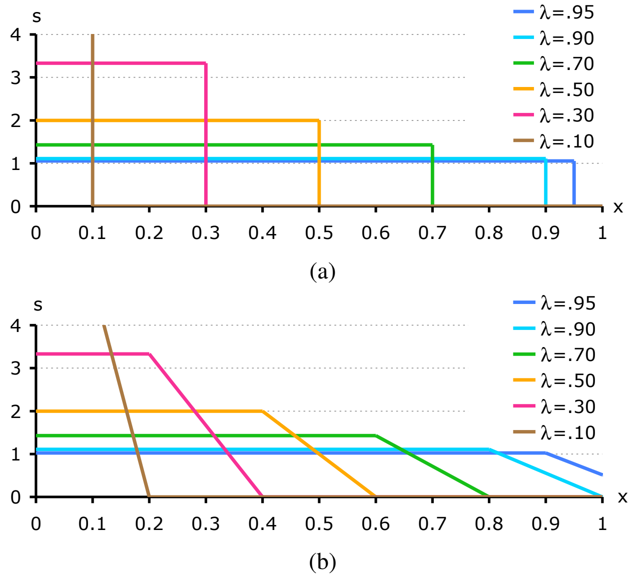

Figure 7: *For smaller values of λ, more elements are excluded, and the included elements are enlarged more. In (a), elements are excluded abruptly; in (b) the exclusion is gradual.*

that the excluded elements are never even referenced. This can be done by storing the elements in a file in priority order, so that only the first λN elements in the file need to be read at rendering time (Figure 6). This file can be created as a post-process to any modeling process. This works especially well in a film production environment, where it is common to use a variety of specialized modelers and scenes are often built up using random geometric variations of a small number of pre-built topologies.

3.3 Area preservation

We preserve area in a manner related to [Deussen et al. 2002]. The total area of the object is Na, where a is the average surface area of the individual elements. Excluding elements decreases the total area to λNa; to compensate for this, the area of the included elements should be scaled by an amount s so that $(\lambda N)(as) = Na$, which we solve to get:

$$\boxed{s = 1/\lambda} \tag{3}$$

Figure 7a shows s as a function of x, the position of the element in the priority order. s is $1/\lambda$ for the included elements ($x \leq \lambda$) and 0 for the excluded elements ($x > \lambda$).

For example, the unsimplified plant in Figure 2a is noticeably less dense when 90% of its leaves have been excluded, as shown in Figure 2b. In Figure 2c, we correct for this by making the remaining leaves 10 times larger so that the total area of the plant remains the same. Depending on the type of element, this can be done by scaling in one or two dimensions; in this example, the leaf widths

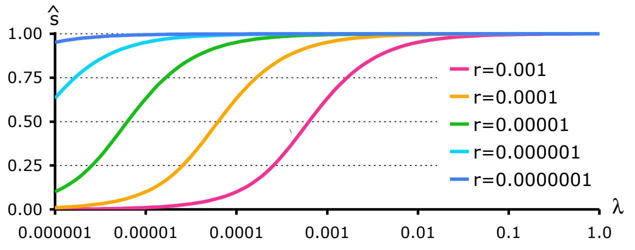

Figure 8: *$\hat{s}$ from Equation 7 for different values of r. For example, for an object with 100,000 elements and a depth complexity of 10, $r = .0001$ and the approximation $s = 1/\lambda$ is off by about 5% when the object is simplified to 100 elements ($\lambda = 0.001$).*

are scaled, as shown in close-up view in Figure 2c. The widening visible in this magnified view is not noticeable in practice because the elements are so small that their shapes are not discernible.

Equation 3 preserves the surface area, but what we really need to preserve is the visible area. The ratio between these two areas changes because reducing the number of elements changes the depth complexity, though the change is small unless λ is very small. For example, the total element surface area of an object with a depth complexity of 2 is twice as great as the visible object area because half of the element area is hidden. If the object were reduced to a single element and we set $s = 1/\lambda$, then the visible area of the scaled element would be twice the visible area of the object because the depth complexity would have changed to 1.

To calculate s taking this into account, we first calculate the ratio $r = a/V$ of the average element area a to the total visible area of the object V. The depth complexity D is total element area over V, so $D = Na/V$ and thus $r = D/N$. (D can be calculated from the alpha values if the object is rendered with the elements partially transparent.) This ratio r can be pre-calculated because it does not change with scale. (It can change with orientation, but this effect is usually small enough to be ignored.)

Next we calculate the expected visible area of n random elements. The first element is completely visible, so it covers r of the visible object area and leaves $1 - r$ uncovered. The second element covers r of that uncovered area, leaving $1 - r$ of it uncovered, so the uncovered area is now $(1 - r)^2$ and the visible part is $1 - (1 - r)^2$. After n elements, the visible part of the area is:

$$1 - (1 - r)^n \tag{4}$$

When the object is simplified, the average element area is as, so it covers rs of the visible object area. After λn elements, the visible part of the area is:

$$1 - (1 - rs)^{\lambda n} \tag{5}$$

Setting expressions (4) and (5) equal and solving for s, we get

$$\boxed{s = \frac{1-(1-r)^{1/\lambda}}{r}} \tag{6}$$

This formula for s is only needed when the simplification is extremely aggressive. This is evident if we use the binomial theorem to get the following power series for s:

$$s = \frac{1}{\lambda}\hat{s} = \frac{1}{\lambda}\left(1 - \frac{(1-\lambda)}{2!}\left(\frac{r}{\lambda}\right) + \frac{(1-\lambda)(1-2\lambda)}{3!}\left(\frac{r}{\lambda}\right)^2 - \cdots\right) \tag{7}$$

so that $\hat{s} \approx 1$ and $s \approx 1/\lambda$ when λ is large compared to r. For example, Figure 8 shows $\hat{s}$ as a function of λ for various values of r. The difference is small except for small values of λ, when the object is so small on the screen that the difference is usually not noticeable anyway.

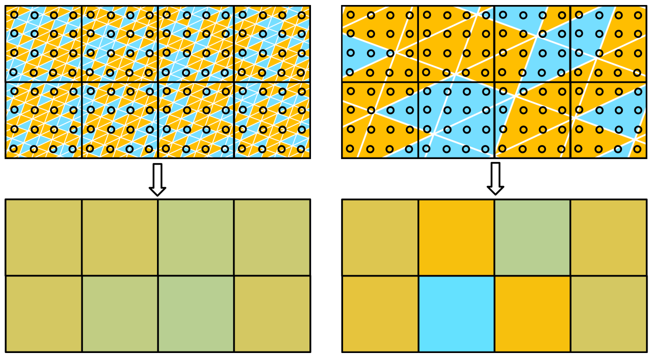
Figure 9: *Example of contrast change. Detailed geometry is shown on the upper left and a simplified version on the upper right. Triangles were set to one of two colors using a random number generator. The squares indicate pixels, and the dots indicate samples. The lower images show the resulting pixel colors. The simplified version has more contrast because fewer triangles are averaged.*

3.4 Contrast preservation

From the central limit theorem, we know that sampling more elements per pixel decreases the pixel variance (as illustrated in Figure 9). As a result, excluding more elements from an object increases its variance (i.e., contrast). Notice how the simplified plant in Figure 2c has a higher contrast than the unsimplified plant in Figure 2a.

Other methods compensate for this automatically because contrast reduction is a side effect of averaging the properties of neighboring elements. Our method compensates instead by averaging the properties of each included element with the properties of the overall population.

To see how this works, we start with the variance σ^2_{elem} of the color of the elements, which is

$$\sigma^2_{elem} = \sum_{i=1}^{n} (c_i - \bar{c})^2 \tag{8}$$

where c_i is the color of the i^{th} element and $\bar{c}$ is the mean color. When k elements are sampled per pixel, the expected variance of the pixels is related to the variance of the elements by:

$$\sigma^2_{pixel} = \sum_{i=1}^{k} (w_i)^2 \sigma^2_{elem} \tag{9}$$

where the weight w_i is the amount the i^{th} element contributes to the pixel. For this analysis, we assume that each element contributes equally to the pixel with weight $1/k$:

$$\sigma^2_{pixel} = \sum_{i=1}^{k} (\frac{1}{k})^2 \sigma^2_{elem} = k(\frac{1}{k})^2 \sigma^2_{elem} = \sigma^2_{elem}/k \tag{10}$$

The pixel variance when the unsimplified object is rendered is:

$$\sigma^2_{unsimplified} = \sigma^2_{elem}/k_{unsimplified} \tag{11}$$

and the pixel variance when the simplified object is rendered is:

$$\sigma^2_{simplified} = \sigma^2_{elem}/k_{simplified} \tag{12}$$

We can make these variances the same by altering the colors of the included elements to bring them closer to the mean:

$$c'_i = \bar{c} + \alpha(c_i - \bar{c}) \tag{13}$$

which equals the original color when $\alpha = 1$ and the mean color when $\alpha = 0$. The variance of the elements is reduced to:

$$\begin{aligned} \sigma'^2_{elem} &= \sum_{i=1}^{n} (c'_i - \bar{c})^2 && (14)\\ &= \sum_{i=1}^{n} (\bar{c} + \alpha(c_i - \bar{c}) - \bar{c})^2 && (15)\\ &= \alpha^2 \sum_{i=1}^{n} (c_i - \bar{c})^2 && (16)\\ &= \alpha^2 \sigma^2_{elem} && (17) \end{aligned}$$

which in turn reduces the variance of the pixels to:

$$\begin{aligned} \sigma'^2_{simplified} &= \sigma'^2_{elem}/k_{simplified} && (18)\\ &= \sigma^2_{elem}\alpha^2/k_{simplified} && (19)\\ &= \sigma^2_{unsimplified}\alpha^2 k_{unsimplified}/k_{simplified} && (20) \end{aligned}$$

We want $\sigma'^2_{simplified} = \sigma^2_{unsimplified}$, which is true when

$$\alpha^2 = k_{simplified}/k_{unsimplified} \tag{21}$$

An image of a simplified object with these altered element colors will have the same variance as an image of the unsimplified object with the original element colors.

Because the number of elements per pixel is inversely proportional to the object size, $k_{unsimplified} = k_1/b$, where k_1 is the value of k at $b = 1$ (which we can estimate by dividing N by the number of pixels the object covers when $b = 1$). Similarly, $k_{simplified} = \lambda k_1/b$. Using these expressions in Equation 21 we get

$$\boxed{\alpha = \sqrt{\lambda}} \tag{22}$$

Many renderers, however, have a maximum number of visible objects per pixel k_{max} (e.g., 121 if the renderer point-samples 11x11 locations per pixel). When the elements are small, this limit causes the contrast of the non-simplified object to increase. We can alter the contrast of the simplified object to match by using the following formula for α:

$$\boxed{\alpha = \sqrt{\frac{min(\lambda k_1/b, k_{max})}{min(k_1/b, k_{max})}}} \tag{23}$$

Figure 10 shows α as a function of b for different values of h. Notice that for smaller values of h, the Equation 23 contrast only changes in the middle distance. When the object is close, the contrast is unchanged because there is no simplification. When the object is far away, the contrast is unchanged because there are more than k_{max} included elements per pixel, so that k_{max} elements contribute to the pixel in both the simplified and unsimplified cases. The maximum contrast difference occurs at $b = k_1/k_{max}$. The smaller λ is at this distance, the greater this contrast difference will be; contrast correction is thus more important with more aggressive simplification (larger values of h in Figure 10). Figure 2d shows the plant in Figure 2c with contrast correction.

If there are different types of elements in a scene (e.g. leaves and grass), each type needs its own independent contrast correction. Otherwise, blending across types could give colors not in either type. This is also important for preserving the object's appearance when the colors are correlated with geometric position (e.g., if new leaves near the tips of the branches are greener than older leaves in the interior).

The value of $\bar{c}$ should be based on the final shaded colors of the elements; this can be difficult to compute in practice but it can often be approximated by reducing the variance of the shader inputs. This does not work well for surfaces with narrow, bright highlights; at this point, our only remedy is to make the simplification less aggressive - i.e., use a smaller h.

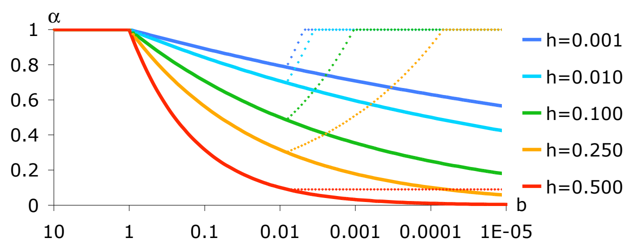

Figure 10: *Contrast correction is more important for aggressive simplification (larger values of h). Solid lines show Equation 22; dotted lines show Equation 23. Parameters: $k_1 = 1$, $k_{max} = 121$.*

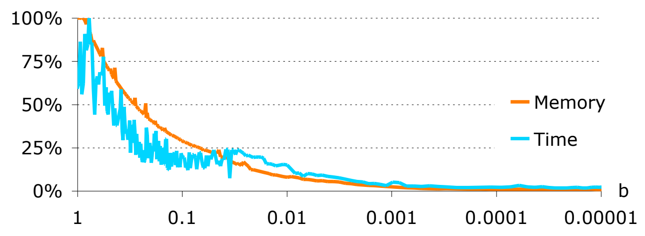

Figure 11: *Rendering time and memory as a function of b for the plant in Figure 1, shown as a % of values for the unsimplified scene.*

3.5 Smooth Animation

The detail level of an object can change significantly during an animation, so changes in simplification level need to be smooth and continuous. As elements are excluded, they should gradually fade out instead of just abruptly disappearing. This can be done by gradually making the elements either more transparent or smaller as they are excluded. The latter is shown in Figure 7b, where the size t of the transition region is 0.1. The red line shows that for a desired level of detail that excludes 70% of the elements ($\lambda = .3$), the first 20% of the elements in the rendering priority order ($x <= \lambda - t = .2$) are enlarged by $1/\lambda = 10/3$ and the last 60% ($x > \lambda + t = .4$) are completely excluded. From x=.2 to x=.4, the areas gradually decrease to 0. As we zoom in and λ increases, the elements at $x = .4$ are gradually enlarged, reaching their fully-enlarged size when $\lambda = .5$ (the yellow line). The area under each line is the total surface area of the included elements and is constant.

4 Results

All of the sagebrushes in Figure 3 are variations of the single plant in Figure 1. This plant was created with custom modeling code, and its leaves were randomly reordered and written into a file. For each plant in the scene, the number of leaves rendered was determined by λ from Equation 2 using the screen size of the plant's bounding box. That number of leaves was read from the head of the file, accessed through an RiProcedural call in the RenderMan Shading Language [Apodaca and Gritz 1999]. Each plant was randomly scaled and rotated, and its leaves were procedurally animated and rendered as curved line segments called RiCurves [Apodaca and Gritz 1999], which were thickened using Equation 6. Separate contrast correction was done for the green leaves and the yellow flower tips using Equation 23.

Figure 11 shows rendering statistics for an animation of the plant in Figure 1 receding into the distance (available at [Cook et al. 2007]); the memory usage decreased from 409.3MB to 3.4MB and the rendering time from 239.3 sec. to 1.7 sec. The same shadow map was used for all frames; this does not produce artifacts when the number of elements changes because preserving area also preserves the probability of being in shadow. The scene in Figure 3 contains over one hundred million leaves and requires so much memory that without simplification it is unrenderable with RenderMan. Figure 12 shows a variety of plants rendered with this technique.

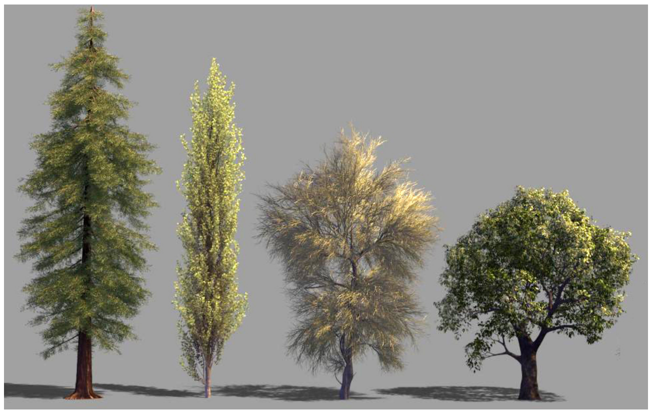

Figure 12: *Some plants rendered with stochastic simplification.*

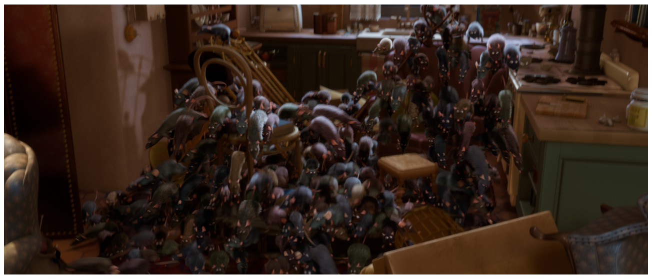

Figure 13: *A swarm of rats.*

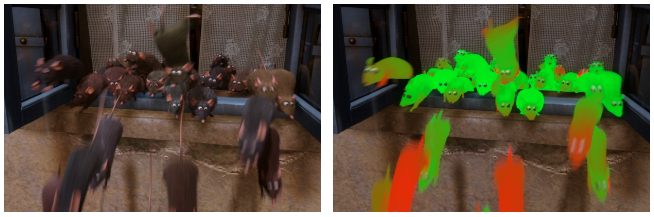

Figure 14: *In the left image, simplification depends on motion blur. In the right image, color indicates λ_{blur}; green areas are more detailed, red areas are more simplified.*

The scene in Figure 13 contains 240 million hairs and was unrenderable without simplification, which reduced the hair count by 94%. The hairs are computed at rendering time rather than being precomputed and stored in a file. They are generated using stratified sampling; this is done by dividing the body into patches and generating the hairs for each patch independently. The number of hairs in each patch depends on its screen size and the amount of blur due to motion and depth of field. Figure 14 shows how the amount of simplification varies with motion blur. Sometimes one part of a rat moves quickly while other parts are still, so it is important that λ be allowed to vary within each rat, unlike the sagebrush model in which the level of detail did not vary within an object. In rendering the rats in this image, stochastic simplification reduced the number of hairs by 88%, the memory usage by 84%, and the rendering time by 75%. Figures 15 and 16 are from a shot with a rack focus, and λ depended on the depth-of-field blur. In Figure 15, 97% of out-of-focus background rat's hairs were excluded ($\lambda_{size} = 0.14$, $\lambda_{blur} = 0.17$). In Figure 16, 70% of the out-of-focus foreground rat's hairs were excluded ($\lambda_{size} = 0.39$, $\lambda_{blur} = 0.78$).

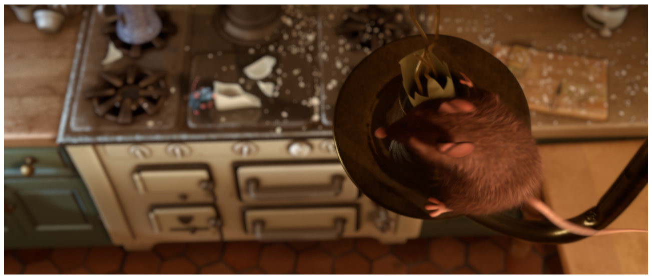
Figure 15: *Depth of field with focus on the closer rat.*

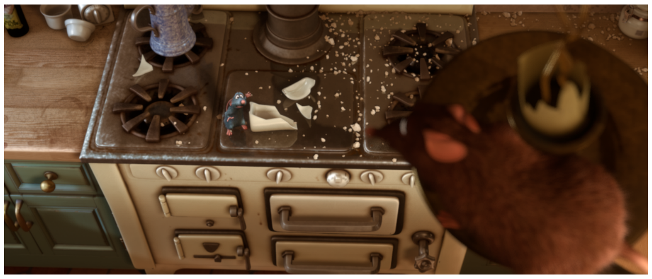
Figure 16: *Depth of field with focus on the far rat.*

5 Discussion and Conclusions

We have found this stochastic approach to simplification effective in reducing the complexity of scenes with large numbers of simple, disconnected elements. It is easy to implement: just randomly shuffle the elements into a file and use code like that in the Appendix to read just the included elements. It also fits well into a production pipeline. We have successfully used it on a variety of models in highly complex scenes that we found otherwise impossible to render at high quality.

This approach is designed for aggregate detail (e.g., snow, rain, water spray, dust, sand, feathers, swarms of insects, schools of fish). The method is not suitable for element detail because removing surface elements would create holes in the surface. For scenes that have both element and aggregate detail, this approach could be used in conjunction with a surface filtering method.

The elements must be small enough that individual element shapes are not more important than the shape of the whole, and there have to be sets of elements that are roughly similar in appearance. If the appearance factors such as size, normals, and color are more widely varied, the simplification cannot be as aggressive.

This paper lays out the principles of contrast correction, but the implementation is still ad-hoc; we would like a method that doesn't have to be tailored to each model. For example, we would like to have a general method for dividing elements into groups of similar characteristics, such as size or color. In addition, we currently cannot simplify shiny objects as aggressively as matte objects because we need a better way to preserve appearance when brightness varies abruptly with normal direction. We believe local characteristics such as neighboring colors, normal distribution, local contrast, and proximity to the silhouette edge could be stored in a volume texture and used to adapt the simplification to local variations; the texture filter size would depend on λ so that a larger texture region would be sampled as more elements were excluded. Instead of determining the priority order randomly, we could use an error metric, so that we start with the element closest to the mean and successively select as the next element the one that gives the best approximation to the statistics of the population.

We have presented a stochastic approach to the simplification of aggregate detail for high-quality surface rendering. Because it is statistically based and does not filter neighboring geometry, it can be less locally accurate than other approaches; but this very lack of local information makes the element calculations independent, which is a significant advantage when handling very large amounts of aggregate detail. We believe statistical approaches like this will become increasingly important.

Acknowledgments

We would like to thank Eben Ostby, Tony Apodaca, and Brian Smits for early feedback on this idea, Martin Reddy for reviewing our drafts, Marco DaSilva for help with the implementation, and Paul Kanyuk for testing. All images ©2007 Disney/Pixar.

Appendix. Sample code

```
// This code was written as a compact example, not for speed or generality.
// For example, these are in-memory routines, but in practice the elements
// would be streamed in from a file. The "Simplify" routine takes an array
// "eIn" of elements in rendering priority order and returns an array of
// elements "eOut" to be rendered. The variable "L" is lambda. These
// diagrams illustrate how s and t change near L=0 and L=1:
//
//    (when L-trans<0)                               (when L+trans>1)
//   sMax-> s            ...sss <- sMax               ...sss <- sMax
//          |s                  s                          . s
//          | s                   s                        .   s
//          |  s<t>      or         s<- t ->     or        .<-t->s <- sMin
//          |   s                   . s                    .     |
//          |    s                  .    s                 .     |
// sMin=0-> +-----s- x           ---+------sss...     -----+-----+ x
//          0     L                 L                      L     1
Simplify( element *eIn, element *eOut, double b, double k1, int nIn, int *nOut ) {
    double h=0.4, kmax=121, k=k1/b;
    double L = (b>=1) ? 1 : pow(b,log(0.5,h));
    double trans = 0.1;                                       // halfsize of transition region
    double t = (L-trans<0) ? L : (L+trans>1) 1-L : trans;     // t is trans adjusted for ends
    double sMax = (L+trans<1) ? 1/L : 1/(1-t*t/trans);        // max area scaling for this L
    double sMin = ( L + trans < 1 ) ? 0 : sMax*(1.0-t/trans); // min area scaling for this L
    *nOut = (u+t) * nIn;                                      // # used, included transition
    for (i=0; i<*nOut; i++) {
         double x = (i+0.5)/nIn;                              // position in priority order
         double sLerp = (x<u-t) ? 1 : (x<L+t) ? (L+t-x)/(2*t) : 0;
         double s = sMin + (sMax-sMin) * sLerp;               // area scaling for this element
         double alpha = sqrt(min(kmax,k*L)/min(kmax,k));      // contrast correction
         eOut[i] = eIn[i];
         eOut[i]->scaleAreaBy(s);                             // scales area of element
         eOut[i]->scaleContrastBy(alpha);                     // scales contrast of element
    }
}
```

References

Apodaca, A., and Gritz, L. 1999. *Advanced RenderMan: Creating CGI for Motion Pictures*. Morgan Kaufmann Publishers Inc.

Christensen, P., and Batali, D. 2004. An irradiance atlas for global illumination in complex production scenes. In *Eurographics Symposium on Rendering*, 133–141.

Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., and Wright, W. 1996. Simplification envelopes. In *Proceedings of ACM SIGGRAPH 1996*, ACM, 119–128.

Cook, R. L., Halstead, J., Planck, M., and Ryu, D. 2007. Stochastic simplification of aggregate detail. Tech. Rep. 06-05a, Pixar Animation Studios. `http://graphics.pixar.com/StochasticSimplification/`.

Crow, F. C. 1982. A more flexible image generation environment. In *Proceedings of ACM SIGGRAPH 1982, Computer Graphics*, ACM, 9–18.

Dachsbacher, C., Vogelgsang, C., and Stamminger, M. 2003. Sequential point trees. In *Proceedings of ACM SIGGRAPH 2003, ACM Transactions on Graphics*, ACM, 657–662.

DÉCORET, X., DURAND, F., SILLION, F. X., AND DORSEY, J. 2003. Billboard clouds for extreme model simplification. In *Proceedings of ACM SIGGRAPH 2003*, ACM, 689–696.

DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., MĚCH, R., PHARR, M., AND PRUSINKIEWICZ, P. 1998. Realistic modeling and rendering of plant ecosystems. In *Proceedings of ACM SIGGRAPH 1998*, ACM, 275–286.

DEUSSEN, O., COLDITZ, C., STAMMINGER, M., AND DRETTAKIS, G. 2002. Interactive visualization of complex plant ecosystems. In *Proceedings of the Conference on Visualization 2002*, 219–226.

ERIKSON, C., AND MANOCHA, D. 1999. Gaps: General and automatic polygonal simplification. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, 79–88.

GARLAND, M., AND HECKBERT, P. 1997. Surface simplification using quadric error meshes. In *Proceedings of ACM SIGGRAPH 1997*, ACM, 209–216.

GARLAND, M. 1999. Multiresolution modeling: Survey & future opportunities. In *Eurographics '99 State of the Art Report*.

GOBBETTI, E., AND MARTON, F. 2004. Layered point clouds. In *Symposium on Point-Based Graphics*, 113–120.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of ACM SIGGRAPH 1996*, ACM, 99–108.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization*, 35–42.

KALAIAH, A., AND VARSHNEY, A. 2003. Statistical point geometry. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 107–115.

KLEIN, J., KROKOWSKI, J., FISCHER, M., WAND, M., WANKA, R., AND AUF DER HEIDE, F. M. 2002. The randomized sample tree: a data structure for interactive walkthroughs in externally stored virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, 137–146.

LACEWELL, J. D., EDWARDS, D., SHIRLEY, P., AND THOMPSON, W. B. 2006. Stochastic billboard clouds for interactive foliage rendering. *Journal of Graphics Tools 11*, 1, 1–12.

LOUNSBERY, M., DEROSE, T., AND WARREN, J. 1997. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics 16*, 1 (January), 34–73.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of ACM SIGGRAPH 1997*, ACM, 199–208.

LUEBKE, D., AND HALLEN, B. 2001. Perceptually driven simplification for interactive rendering. In *Proceedings of the Eurographics Workshop on Rendering*, 223–234.

LUEBKE, D., REDDY, M., COHEN, J., VARSHNEY, A., WATSON, B., AND HUEBNER, R. 2002. *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers Inc.

NEYRET, F. 1995. Animated texels. In *Computer Animation and Simulation '95*, Eurographics, 97–103.

NEYRET, F. 1998. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (January), 55–70.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000*, ACM, 335–342.

PRUSINKIEWICZ, P., JAMES, M., AND MECH, R. 1994. Synthetic topiary. In *Proceedings of ACM SIGGRAPH 1994*, ACM, 351–358.

REEVES, B. 1983. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics 2*, 2 (April), 91–108.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, ACM, 343–352.

SCHAUFLER, G., AND STÜRZLINGER, W. 1996. A three dimensional image cache for virtual reality. In *Eurographics '96 Proceedings*, 227–235.

STAMMINGER, M., AND DRETTAKIS, G. 2001. Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, Springer-Verlag, London, UK, 151–162.

WAND, M., AND STRASSER, W. 2002. Multi-resolution rendering of complex animated scenes. *Computer Graphics Forum 21*, 3, 483–483.

WAND, M., FISCHER, M., PETER, I., AUF DER HEIDE, F. M., AND STRASSER, W. 2001. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proceedings of ACM SIGGRAPH 2001*, ACM, 361–370.

WILLIAMS, N., LUEBKE, D., COHEN, J., KELLEY, M., AND SCHUBERT, B. 2003. Perceptually guided simplification of lit, textured meshes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics*, 113–121.

WILSON, A., AND MANOCHA, D. 2003. Simplifying complex environments using incremental textured depth meshes. In *Proceedings of ACM SIGGRAPH 2003*, ACM, 678–688.

YOON, S.-E., LAUTERBACH, C., AND MANOCHA, D. 2006. R-lods: fast lod-based ray tracing of massive models. *The Visual Computer 22*, 9, 772–784.