# Alpha Distribution for Alpha Testing

Cem Yuksel
University of Utah
cem@cemyuksel.com

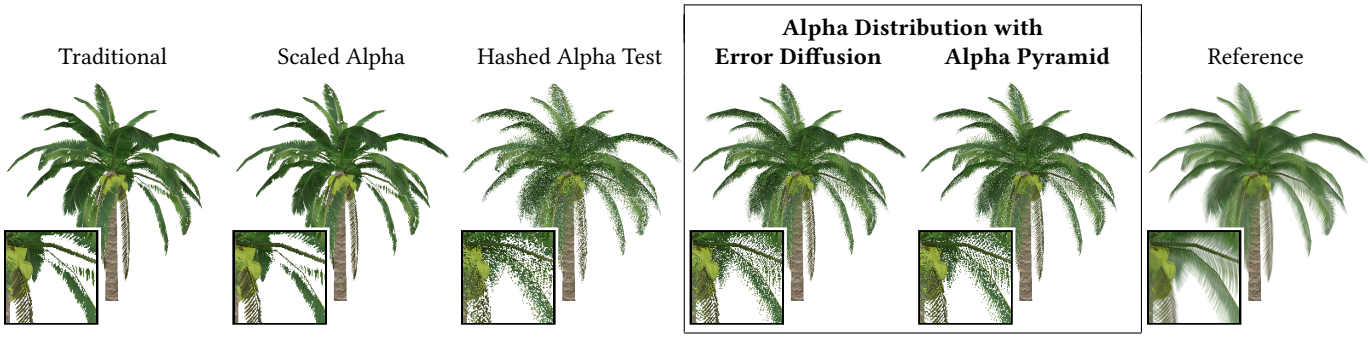


**Figure 1:** *Comparison of alpha testing using different methods. Insets highlight one region of the images where traditional alpha testing leads to geometry loss. Scaling the alpha values [Castaño 2010] cannot fix this problem. Hashed alpha testing [Wyman and McGuire 2017a,b] solves geometry disappearance, but introduces substantial amount of noise as well as shader complexity. Our alpha distribution approach with either error diffusion or alpha pyramid algorithms provides a close match to the reference image with low noise.*

## ABSTRACT

Alpha testing is widely used for rendering surfaces with transparency. While it works well when transparency is a binary function, semi-transparent regions cause problems, as they are classified as either fully transparent or opaque. Unfortunately, semi-transparent texture regions often appear in coarser mipmap levels, causing surfaces to disappear with distance. We introduce the *alpha distribution* approach for pre-processing the alpha values of a texture such that alpha testing produces expected results with semi-transparency without any modification to render-time operations. We describe two separate algorithms for alpha distribution with similar qualitative behavior. Our results show that alpha distribution can produce high-quality results with low noise. We also explain how alpha distribution can be extended for high-quality rendering with alpha-to-coverage.

## CCS CONCEPTS

• **Computing methodologies** → **Visibility**;

## KEYWORDS

alpha test, alpha map, error diffusion

**ACM Reference Format:**
Cem Yuksel. 2018. Alpha Distribution for Alpha Testing. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D'18).* ACM, New York, NY, USA, Article 1, 7 pages. https://doi.org/10.1145/3203185


*I3D'18, May 2018, Montreal, Quebec, Canada*

ACM ISBN 2577-6193/2018/5-ART1…$15.00
https://doi.org/10.1145/3203185

## 1 INTRODUCTION

Alpha testing is a simple method for handling a binary transparency function. During rendering, fragments that have alpha values below a user-specified threshold $\alpha_\tau$ are discarded (i.e. considered fully transparent) and other fragments are taken as fully opaque. This provides a simple and efficient mechanism for handling binary visibility. Therefore, alpha testing is widely used for rendering objects like fences, hair and grass blades, leaves, and various forms of vegetation with low-poly representations. Thus, it allows avoiding the geometric complexity needed for modeling the detailed shapes of such objects.

On the other hand, alpha testing is not designed for handling semi-transparent surfaces. Yet, when combined with mipmapping, even if the original texture has a binary transparency function, the filtered alpha values at coarse levels often form semi-transparent regions. It is not uncommon that the alpha values at coarser mipmap levels collectively fall on one side of the threshold $\alpha_\tau$, causing the well-known problem of geometry disappearing beyond certain distances. Unfortunately, the common simple fix of adjusting $\alpha_\tau$ per mipmap level cannot always prevent geometry disappearance with distance [Castaño 2010].

Recently, Wyman and McGuire [2017a; 2017b] introduced *hashed alpha testing* that cleverly solves the problems of alpha testing by using a stochastic alpha threshold with a stable hash function. While certainly effective, this approach leads to a substantial amount of noise in the final image. Furthermore, though the hash value per fragment can be computed efficiently, it introduces fair amount of additional complexity to fragment shaders.

In this paper we introduce *alpha distribution*, an extremely simple approach for solving the problems of traditional alpha testing and extending it to handle semi-transparent surfaces. Alpha distribution works as a pre-process that alters the alpha values of a given texture, including all mipmap levels, such that a desirable subset of texels would pass the alpha test. The render-time process

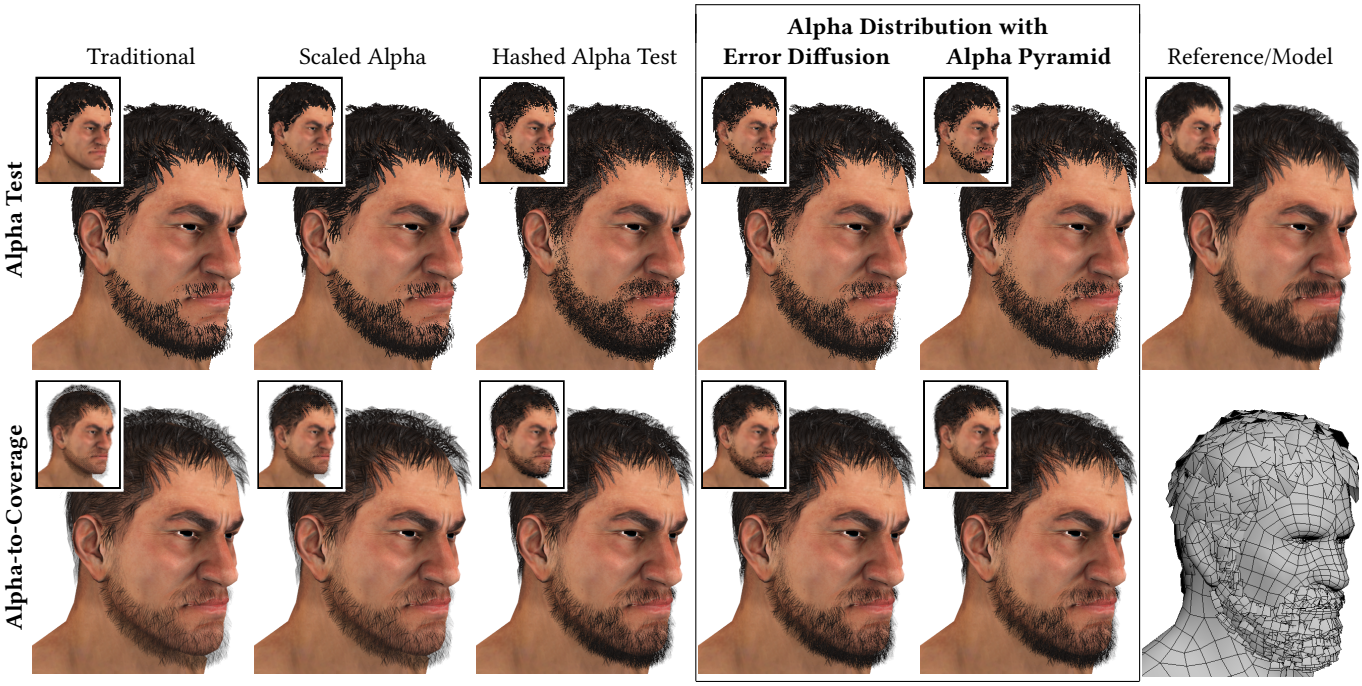


**Figure 2:** *Comparison of different methods for (top row) alpha testing and (bottom row) alpha-to-coverage with semi-transparent hair and beard textures. Insets show the same model rendered from a far view. Notice that traditional alpha testing entirely misses the beard in far view and produces incorrect visibility for both hair and beard with alpha-to-coverage even in close-up view. Scaling the alpha values [Castaño 2010] cannot fix these problems. Our alpha distribution approach and hashed alpha testing [Wyman and McGuire 2017a,b] produce the expected results, as compared to the reference. Alpha distribution results with alpha-to-coverage are generated using precomputed sample mask textures.*

is identical to traditional alpha testing, so no modification is needed to the fragment shader code. We describe two different algorithms for alpha distribution, both of which can handle semi-transparent surfaces with low noise, while attempting to preserve the details of the original texture. Our results show that alpha distribution leads to improved rendering quality, as compared to prior alpha testing methods (Figure 1). We also describe how alpha distribution can be extended to handle alpha-to-coverage that converts the alpha values to sub-pixel MSAA coverage samples for high-quality rendering with anti-aliasing and semi-transparency.

## 2 RELATED WORK

Alpha testing is widely used and the problem of geometry disappearance with distance is well-known, but it received limited attention in the computer graphics research community. Unfortunately, simple solutions to the disappearance problem are undesirable for different reasons. Castaño [Castaño 2010] suggests scaling the alpha values by first finding a desirable alpha threshold per mipmap level, such that the same percentage of texels would pass the alpha test as the highest resolution level. This simple fix can help in some cases, but it does not always improve the results. Another solution is disabling mipmap filtering, which would mean either no mipmap filtering for the color channels or a secondary texture fetch for the alpha values. Also, neither one of these solutions can handle semi-transparency. Screen-door transparency [Mulder et al. 1998], however, can handle semi-transparent objects by masking the visibility for a subset of pixels. Our alpha testing solution with alpha distribution is closely related to screen-door transparency, but we do not consider a screen-space sampling pattern.

Hashed alpha testing [Wyman and McGuire 2017a,b] is an elegant solution that was recently introduced. It eliminates the disappearance problem by picking a stochastic threshold value $\alpha_\tau \in (0, 1]$ per fragment. This provides a form of stochastic transparency [Enderton et al. 2010]. On the other hand, it introduces substantial amount of noise to the final image and a fair amount of additional complexity to fragment shaders.

Handling transparency accurately requires ordered alpha compositing [Porter and Duff 1984]. This can be accomplished using depth peeling [Everitt 2001] or more recent methods like order-independent transparency [Wyman 2016]. However, the computation cost of these techniques make them undesirable in practice.

## 3 ALPHA DISTRIBUTION

Alpha distribution works on the alpha channel of a texture and its mipmap levels. The goal is to adjust the alpha values such that a desirable subset of texels would pass the alpha test at render time.

Since the render time computation is identical to traditional alpha testing, alpha distribution does not impact performance or shader complexity. The quality of alpha distribution, however, depends on how the alpha values are precomputed. Below we describe two simple algorithms for alpha distribution. It is certainly possible to imagine other alternative methods for alpha distribution than the ones described in this section.

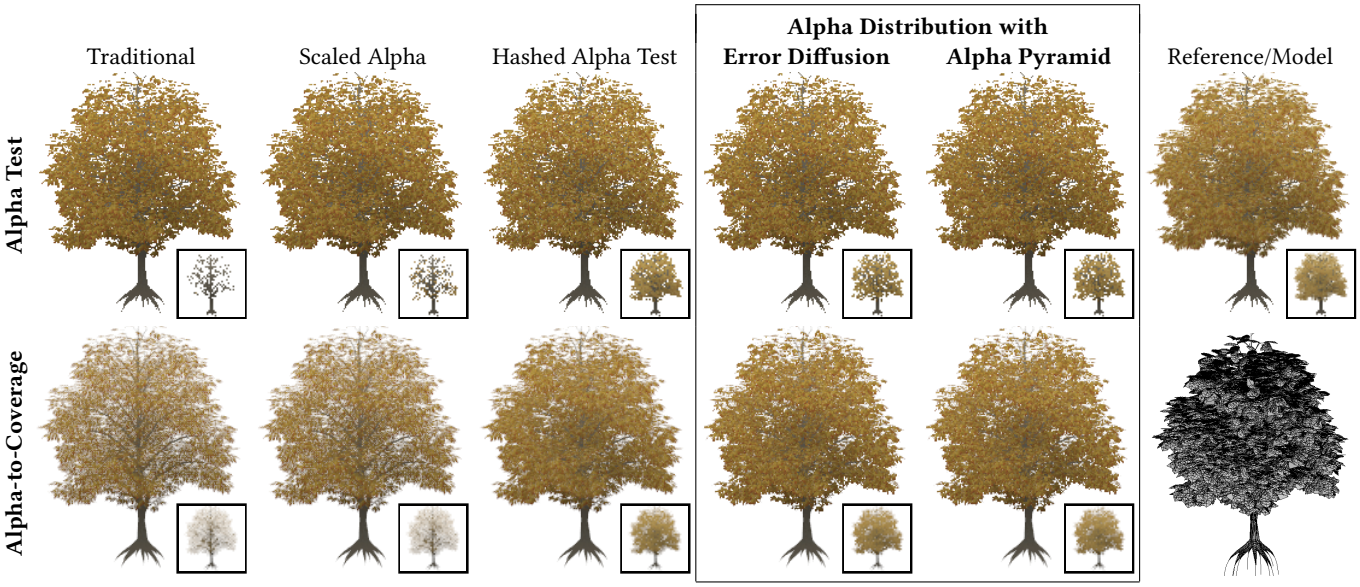


**Figure 3:** *Comparison of different methods for (top row) alpha testing and (bottom row) alpha-to-coverage using a leaf texture with binary transparency. Insets show the same model rendered from a far view. Notice that geometry disappears with traditional alpha testing in far view. When using alpha-to-coverage with traditional alpha testing, visibility loss starts even at relatively close distances and worsens in far view. Scaling the alpha values [Castaño 2010] produces similar results. Our alpha distribution approach and hashed alpha testing [Wyman and McGuire 2017a,b] closely match the reference, including the far views. The complexity of this model hides the noise introduced by hashed alpha testing. Alpha distribution results with alpha-to-coverage are generated using precomputed sample mask textures. Tree model created by Xfrog.*

## 3.1 Error Diffusion

Our realization is that the problem of generating a binary visibility function from alpha values is very similar to halftoning. Therefore, one simple way of modifying the alpha values would be following an error diffusion approach that is designed for dithering. In our tests we use the traditional error diffusion technique of Floyd and Steinberg [1976], though more recent alternatives [Ostromoukhov 2001; Zhou and Fang 2003] can also be used for this task.

Just like halftoning, error diffusion for alpha distribution sequentially visits each texel $i$ at position $(x, y)$ in scanline order and computes a quantized alpha value $\hat{\alpha}_i \in \{0, 1\}$ by comparing its alpha value $\alpha_i$ to the threshold $\alpha_\tau$. The quantization error $\epsilon_i = \alpha_i - \hat{\alpha}_i$ is distributed to the neighboring texels that are not already visited. Using the Floyd and Steinberg [1976] pattern, the portions 7/16, 3/16, 5/16, and 1/16 of the quantization error $\epsilon_i$ are added to the alpha values of the neighboring texels at $(x + 1, y)$, $(x - 1, y + 1)$, $(x, y + 1)$, and $(x + 1, y + 1)$, respectively. First, all mipmap levels are generated; then, each mipmap level of the original texture is independently modified using error diffusion.

Error diffusion prevents large portions of the texture from collectively falling one side of $\alpha_\tau$ by distributing the quantization error. The modified alpha values produce a dithering pattern that approximates the appearance of the original alpha values with binary visibility. As a result, it solves the disappearance problem and allows representing semi-transparent regions using dithering.

## 3.2 Alpha Pyramid

Error diffusion works well, but the dithering pattern it produces might be undesirable. Therefore, we introduce an alternative method for alpha distribution that we call *alpha pyramid*.

The goal of the alpha pyramid method is to determine the number of *visible texels* that should pass the alpha test for each mipmap level and then modify the alpha values to achieve this. Like error diffusion, it processes each mipmap level completely independently. Nonetheless, due to the nature of the alpha pyramid algorithm, the resulting mipmap levels preserve some correlation, especially when the texture resolution is a power of 2.

Starting with a given input image (i.e. the original texture or a mipmap level), the alpha pyramid method begins by computing the average alpha value $\bar{\alpha}$. The number of visible texels $n$ to pass the alpha test is determined by

$$n = \left\lceil \frac{\bar{\alpha} N}{2\alpha_\tau} \right\rceil, \tag{1}$$

where $N$ is the total number of texels. Note that when $\alpha_\tau = 1/2$, the average opacity of the surface resulted by having $n$ texels pass the alpha test corresponds to $\bar{\alpha}$. Thus, the alpha pyramid method is designed for $\alpha_\tau = 1/2$. When using a different threshold value, it might be favorable to simply scale the alpha values as a preprocess and keep $\alpha_\tau = 1/2$.

To determine which specific texels should pass the alpha test, we first build a pyramid of successively lower resolutions of the input image, similar to mipmap levels, as shown in Figure 5. If the resolution of the input image is a power of 2, the construction of the pyramid is identical to mipmaps. If the resolution of a pyramid level is not divisible by 2, however, texels near the boundary are grouped with the nearest $2 \times 2$ texel regions. Thus, unlike mipmaps, each texel of a pyramid level corresponds to 4, 6, or 9 unique texels of the level below it. Each texel at each pyramid level corresponds to a distinct region of the input image. At each texel $i$ of a pyramid level $\ell$ we store $\alpha_i^\ell$, the sum of the alpha values for the input image

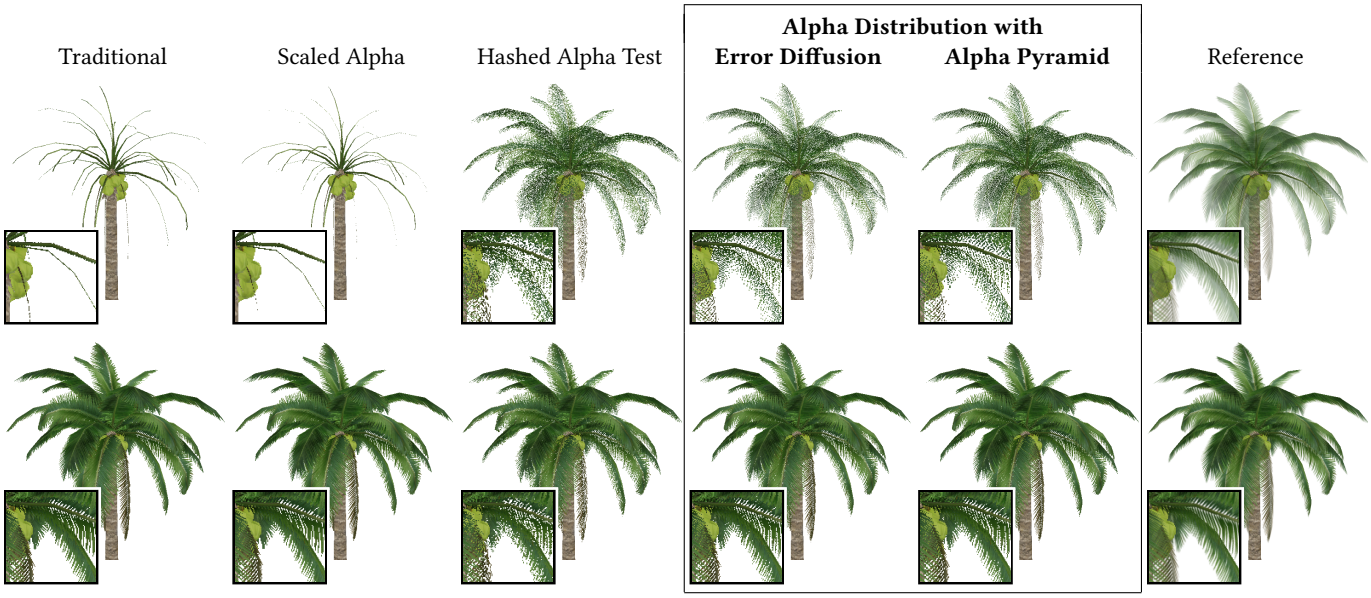


**Figure 4:** *The same model as in Figure 1, using leaf textures with two different nonzero alpha values: (top) slightly below $\sigma_\tau$ and (bottom) 1. Notice that semi-transparent regions are invisible with traditional alpha testing. Hashed alpha testing [Wyman and McGuire 2017a,b] works well in both cases, but leads to apparent noise. Our alpha distribution methods produce good results with diminished noise in both cases.*

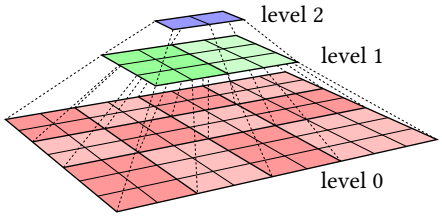


**Figure 5:** *Levels of the alpha pyramid for a $7 \times 9$ texture. Each texel of a level corresponds to 4, 6, or 9 unique texels of the lower level.*

texels it represents. In this notation $\alpha_i^0 = \alpha_i$.

Then, starting from the top of the pyramid, we assign an integer *visibility value* to each texel of each level $\ell \geq 1$. The visibility value of a texel at pyramid level $\ell$ determines how many of its corresponding input image texels should be visible. Thus, $n$ is the overall visibility value for the entire texture. We begin with distributing $n$ to the texels of the highest pyramid level $L$, based on their alpha values. Each texel $i$ is first assigned $\tilde{n}_i^L = \lfloor \alpha_i^L / 2\alpha_\tau \rfloor$ as its initial visibility value. The remaining visibility value (i.e. $n - \sum_i \tilde{n}_i^L$) is distributed to texels with the highest remaining alpha values (i.e. $\alpha_i^L - 2\alpha_\tau \tilde{n}_i^L$). Any contention is resolved by randomly picking one of the competing texels. Note that it is important to employ a random selection process for handling contentions to avoid generating regular patterns. The same process is repeated for all pyramid levels down to level 1. For each texel $i$ at level $\ell > 1$, its visibility value $n_i^\ell$ is distributed to the corresponding texels at level $\ell - 1$ below it. Note that when $\alpha_\tau \geq 1/2$, it is guaranteed that the resulting $n_i^\ell$ is smaller than or equal to the number of input image texels represented by texel $i$ of pyramid level $\ell \geq 1$.

To compute the final alpha values of the input image (i.e. pyramid level 0), texels of the input image that correspond to texel $i$ of pyramid level 1 are sorted based on their alpha values. The first $n_i^1$ texels with highest alpha values are marked as visible texels. Once again, random selection is used for resolving contentions when ordering. Finally, binary alpha values are assigned based on whether a texel is marked as visible.

The alpha pyramid method produces similar results to error diffusion, but it leads to marginally less noise and it better preserves the high-frequency details of the input texture.

## 4 ALPHA-TO-COVERAGE

The alpha-to-coverage feature of GPUs automatically converts the alpha values of fragments to a fixed set of sub-pixel sample coverage patterns. This provides an easy mechanism for rendering semi-transparent objects. The problem is that fragments with similar alpha values correspond to the same coverage pattern and two fragments on a pixel with the same coverage pattern perfectly occlude each other, thereby effectively discarding one of the two fragments. Unfortunately, this case is typical for various models used with alpha testing and mipmapping, resulting in substantial visibility loss with alpha-to-coverage.

Hashed alpha testing [Wyman and McGuire 2017a,b] resolves this issue by extending its stochastic alpha threshold to sub-pixel coverage samples. Using multiple coverage samples per pixel also reduces the visible noise produced by hashed alpha testing, but it introduces further shader complexity.

When using alpha-to-coverage with alpha distribution, we must first adjust our quantization levels. When modifying the alpha values, instead of using a single threshold $\alpha_\tau$, we simply use as many quantization levels as the number of sub-pixel coverage samples provide. In this case, alpha distribution ensures that large portions of the texture are not slightly below (or above) the threshold between different quantization levels.

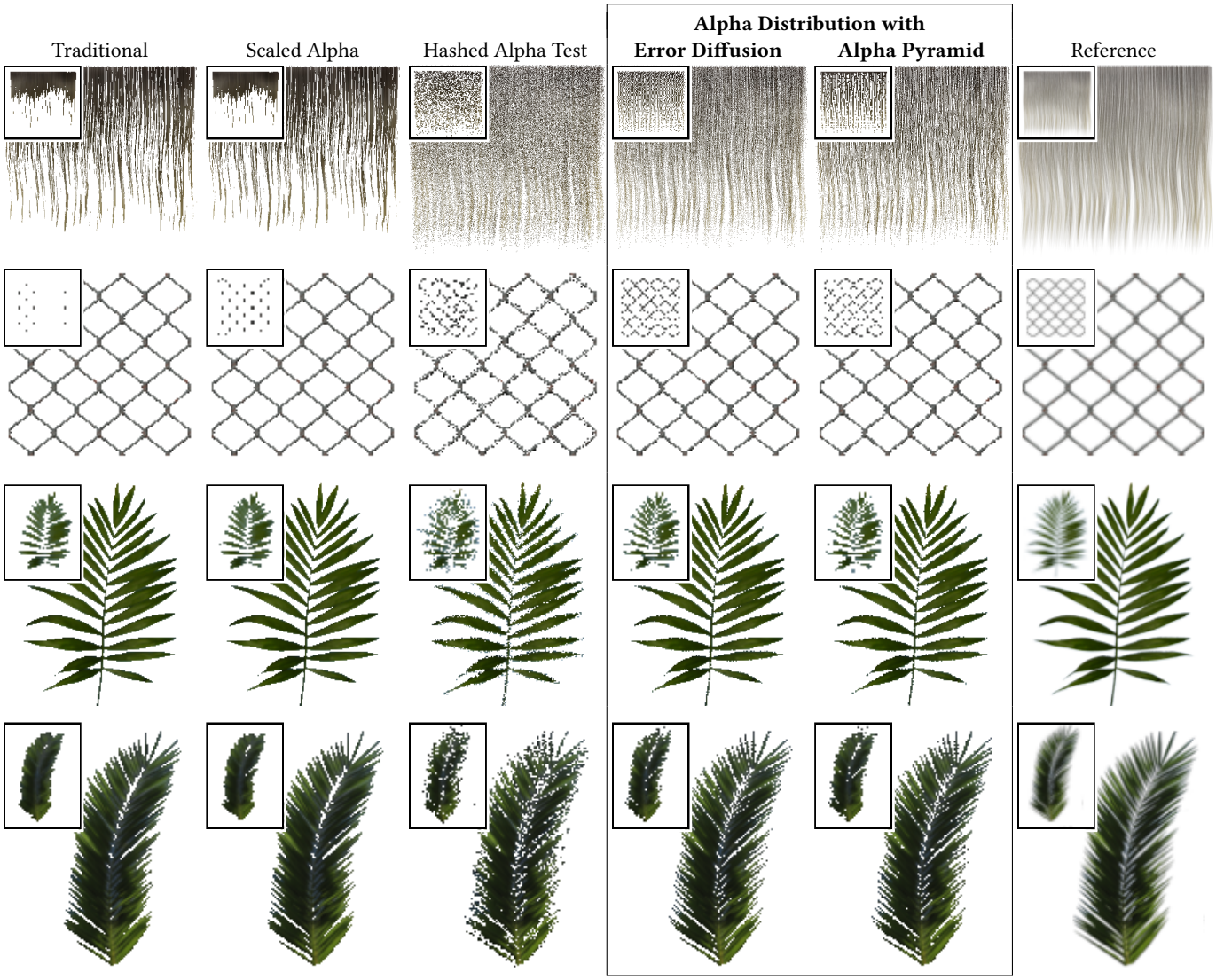


**Figure 6:** *Alpha testing mipmap levels of different textures on a plane using different methods. Insets showing the same textures rendered from a far view using coarser mipmap levels.*

Below we provide two separate solutions for handling alpha-to-coverage. They produce very similar results, but they have different trade-offs in terms of performance and shader complexity.

## 4.1 Sample Mask Texture

Our first solution involves introducing a secondary texture that stores a sample mask. The bits of this sample mask indicate which sub-pixel samples should be considered covered. We use the same resolution sample mask texture as the original texture.

For each texel, we determine the number of nonzero sample mask bits based on its alpha value. Then, we randomly select the bits that are set as nonzero. Therefore, the same alpha quantization with the same number of nonzero bits produces different sample masks for different texels. As a result, unless the textures of two fragments are perfectly aligned, the correlation of their coverage sample patterns is avoided and semi-transparent surfaces can be rendered without apparent visibility loss.

Obviously, introducing a sample mask texture suffers the additional cost of storing and sampling a secondary texture during rendering. The alpha channel of the original texture is no longer used, since the coverage is entirely determined by the sample mask texture. Yet, the sample mask data cannot be stored in the alpha channel of the original texture, since we cannot use bilinear filtering with the sample mask (we use mipmaps with a nearest filter).

## 4.2 Hashed Sample Mask

Instead of precomputing a sample mask texture, it is also possible to compute a sample mask within the fragment shader using the filtered alpha value. The alpha value identifies the number of bits $s$ of the sample mask must be set. We simply set the first $s$ bits and randomly shift the bits with wrap around. This operation is similar to the alpha-to-coverage strategy used with hashed alpha

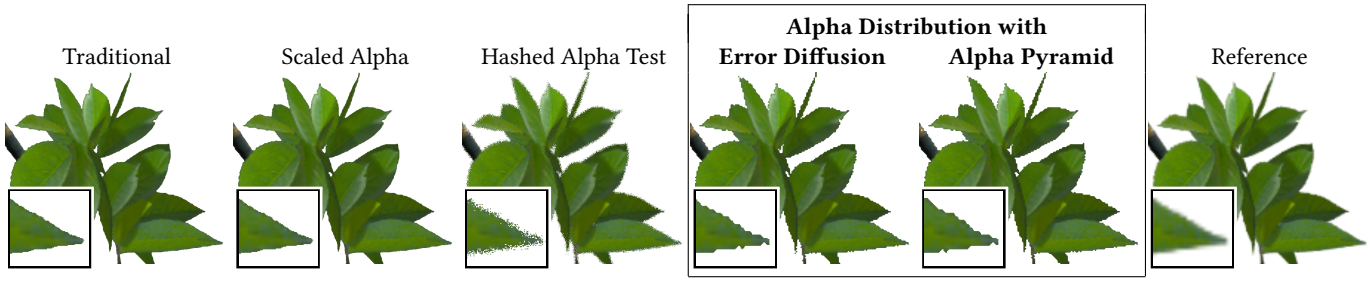


**Figure 7:** *Comparison of different alpha testing method in extreme close-up where bilinear magnification filtering is used. Notice that traditional alpha testing works well in this case. Hashed alpha testing [Wyman and McGuire 2017a,b] produces substantial amount of noise. Our alpha distribution leads to texel-size staircase artifacts around edges due to quantization of the alpha values.*

testing [Wyman and McGuire 2017a,b]. In fact, we can use the same hash function for generating a stable random number. Unlike hashed alpha testing, however, no stochastic decision is used for determining $s$. Thus, we only need a single random value.

The result of computing the sample mask within the fragment shader is qualitatively similar to using a precomputed sample mask texture. The main trade-off is replacing the additional texture lookup with additional shader complexity.

## 5 RESULTS

We provide various comparisons of our alpha distribution methods to traditional alpha testing with unmodified textures, scaled alpha values [Castaño 2010], and the recent hashed alpha testing method [Wyman and McGuire 2017a,b]. We use $\alpha_\tau = 1/2$ for all alpha test examples in this paper. We use the recommended anisotropic hash function with hashed alpha testing for all 3D models. The reference images are generated with depth peeling [Everitt 2001], using as many passes as needed to include *all* visible layers. Obviously, regardless of the method used, neither alpha testing nor alpha-to-coverage can be expected to perfectly reproduce the reference images, but they are provided as ground truth references for the desired appearance.

Figure 1 shows a palm tree with semi-transparent leaves. As expected, with traditional alpha testing parts of the leaves are fully opaque and the other parts are invisible. This behavior is observed in close-up views as well. Scaling the alpha values [Castaño 2010] does not provide a visible difference. Hashed alpha testing [Wyman and McGuire 2017a,b] perfectly reproduces the overall visibility of the reference, but it leads to substantial amount of noise, such that the details of the leaves are difficult to see. Alpha distribution, on the other hand, provides a close match to the reference with low noise. Insets show that the details of the leaves are faithfully reproduced with both alpha distribution methods, while arguably the alpha pyramid method leads to slightly lower apparent noise and better preserved details than error diffusion.

Another example with semi-transparent textures is shown in Figure 2 (top row). The hair and beard strands are modeled as collections of textured faces. In this case traditional alpha testing produces reasonable results for close-up views, but the beard entirely disappears in far view, where scaled alpha values [Castaño 2010] only barely improve the result. Both hashed alpha testing [Wyman and McGuire 2017a,b] and our alpha distribution methods produce the desired result. While hashed alpha testing leads to noisy results for this example as well, it is not as noticeable due to the nature of the hair and beard textures.

The bottom row of Figure 2 includes comparisons using alpha-to-coverage. When using hardware-supported alpha-to-coverage, due to the fixed coverage pattern determined by the alpha value, overlapping semi-transparent faces perfectly occlude each other, resulting in transparent beard and hair with traditional alpha-to-coverage and scaled alpha values. Hashed alpha testing works well with alpha-to-coverage and produces results with diminished noise, as compared to its alpha-test version. Our alpha distribution methods produce results with similar quality.

Figure 3 shows comparisons using a complex model with a texture map containing binary transparency. Yet, the geometry disappearance is evident in far view with traditional alpha testing. Alpha-to-coverage, on the other hand, does not work well even with relatively close views. Scaling the alpha values [Castaño 2010] does not help in this case either. Hashed alpha testing [Wyman and McGuire 2017a,b] and our alpha distribution methods provide similar visibility as compared to the reference with both alpha testing and alpha-to-coverage, including far views.

The quality of alpha testing depends on the transparency function. Figure 4 shows the same model in Figure 1, but using textures with different alpha values. When the majority of alpha values are (slightly) below the threshold, traditional alpha testing simply does not work, but it performs well when the majority of alpha values are close to 1. Hashed alpha testing [Wyman and McGuire 2017a,b] produces noisy results in either case. Our alpha distribution methods work well in both cases.

Figure 6 shows different textures on a camera-facing quad, rendered using different methods. In all examples, when using coarser mipmap levels in far view, traditional alpha testing either loses visibility or leads to too much opacity. Scaling the alpha values [Castaño 2010] makes little to no difference in these examples. The noise with hashed alpha testing [Wyman and McGuire 2017a,b] is quite apparent. Our alpha distribution methods introduce limited noise and produce the expected visibility. Notice that the alpha pyramid method arguably preserves the high-frequency details marginally better than error diffusion.

All of these results show that alpha distribution indeed solves the problems of traditional alpha testing. As compared to hashed alpha testing [Wyman and McGuire 2017a,b], alpha distribution produces substantially less noise with alpha testing, but it provides

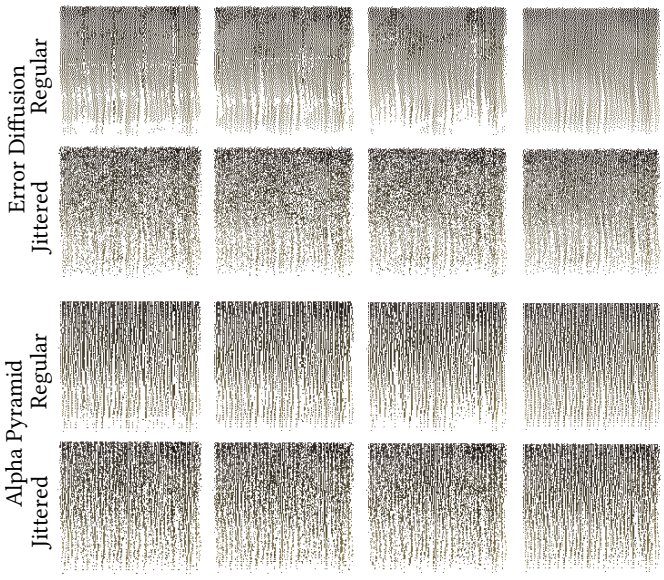


**Figure 8:** *Alpha distribution with error diffusion and alpha pyramid on a camera-facing quad with each column showing a slightly different camera distance. Visible pattern is formed with regular sampling and eliminated using jittering texture lookups.*

no apparent qualitative improvement in alpha-to-coverage. Yet, in both alpha testing and alpha-to-coverage, alpha distribution leads to simpler fragment shaders.

## 6 LIMITATIONS

One obvious limitation of alpha distribution is that when the threshold $\alpha_\tau$ is changed, the alpha values need to be recomputed.

Also, since we precompute the alpha values, when a texture is tiled, the same visibility pattern is repeated on each tile. This is not the case for hashed alpha testing [Wyman and McGuire 2017a,b], as the hash value is computed per fragment.

One important drawback of alpha distribution is that it does not work well in extreme close-ups, where magnification filtering is used, as shown in Figure 7. Because of the quantization of the alpha values, texel-size staircase artifacts appear near all edges. On the other hand, traditional alpha testing using the unmodified alpha values can produce relatively smooth edges using bilinear interpolation of the original alpha values. Therefore, if the original texture does not have semi-transparent regions, this issue can be avoided by simply keeping the original alpha values for level 0. In this case hashed alpha testing [Wyman and McGuire 2017a,b] leads to a different kind of undesirable result in the form of excessive noise, so it must be faded out in close-ups.

Another drawback of alpha distribution is that in some special cases the precomputed alpha pattern can align with regular pixel samples, leading to some form of Moiré pattern, as shown in Figure 8. This pattern is more apparent with error diffusion, but it is visible with alpha pyramid as well. It is possible to ignore this artifact, since it becomes apparent only at certain distances and view angles. To eliminate this artifact, we must break the regular sampling pattern. We can do so by jittering the texture coordinate slightly (using a hash function). This solution introduces some minor noise, but eliminates the visible pattern. We have not used this solution in any example in this paper, except for Figure 8.

## 7 CONCLUSION

We have introduced the alpha distribution approach as simple mechanism for solving the visibility problems with alpha testing and extending it to semi-transparent surfaces. One contribution of this paper is the recognition of the similarity between halftoning and alpha testing. Thus, we have shown that error diffusion [Floyd and Steinberg 1976], a method that is commonly used for halftoning, can be applied to alpha distribution as well. We have also described the alpha pyramid method as an alternative algorithm for alpha distribution. Finally, we have discussed how alpha distribution can be applied to alpha-to-coverage.

One interesting future work would be experimenting with different error-diffusion techniques and comparing their results in alpha testing and alpha-to-coverage.

One important advantage of alpha distribution is that it requires no modification to fragment shader code for alpha testing. Therefore, we have not included any performance tests in this paper. On the other hand, supporting alpha-to-coverage with alpha distribution requires some relatively minor changes to fragment shaders. We expect that the performance implications of these changes would highly depend on which one of the two alternative alpha-to-coverage methods we have described is used, as well as the utilization of the GPU resources for handling other rendering tasks.

## ACKNOWLEDGMENTS

Special thanks to Chris Wyman for insightful discussions and comments on our preliminary results. We also thank Pete Shirley and Konstantin Shkurko for their comments and Morgan McGuire for his graphics repository that includes the tree model in Figure 3. This work was supported in part by NSF grant #1409129.

## REFERENCES

Ignacio Castaño. 2010. Computing Alpha Mipmaps. http://the-witness.net/news/2010/09/computing-alpha-mipmaps/.

Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. 2010. Stochastic Transparency. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 157–164.

Cass Everitt. 2001. *Interactive Order-Independent Transparency.* while paper. NVIDIA.

Robert W. Floyd and Louis Steinberg. 1976. An Adaptive Algorithm for Spatial Greyscale. *Proceedings of the Society for Information Display* 17, 2 (1976), 75–77.

Jurriaan D. Mulder, Frans C. A. Groen, and Jarke J. van Wijk. 1998. Pixel Masks for Screen-door Transparency. In *Proceedings of the Conference on Visualization '98 (VIS '98)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 351–358.

Victor Ostromoukhov. 2001. A Simple and Efficient Error-diffusion Algorithm. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 567–572.

Thomas Porter and Tom Duff. 1984. Compositing Digital Images. *SIGGRAPH Comput. Graph.* 18, 3 (Jan. 1984), 253–259.

Chris Wyman. 2016. Exploring and Expanding the Continuum of OIT Algorithms. In *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*, Ulf Assarsson and Warren Hunt (Eds.).

Chris Wyman and Morgan McGuire. 2017a. Hashed Alpha Testing. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '17)*. ACM, New York, NY, USA, Article 7, 9 pages.

Chris Wyman and Morgan McGuire. 2017b. Improved Alpha Testing Using Hashed Sampling. *IEEE Trans. on Visualization and Computer Graphics* PP, 99 (2017), 1–12.

Bingfeng Zhou and Xifeng Fang. 2003. Improving Mid-tone Quality of Variable-coefficient Error Diffusion Using Threshold Modulation. *ACM Trans. Graph. (Proceedings of SIGGRAPH'03)* 22, 3 (July 2003), 437–444.