Weighted Blended Order-Independent Transparency

Morgan McGuire Louis Bavoil
NVIDIA

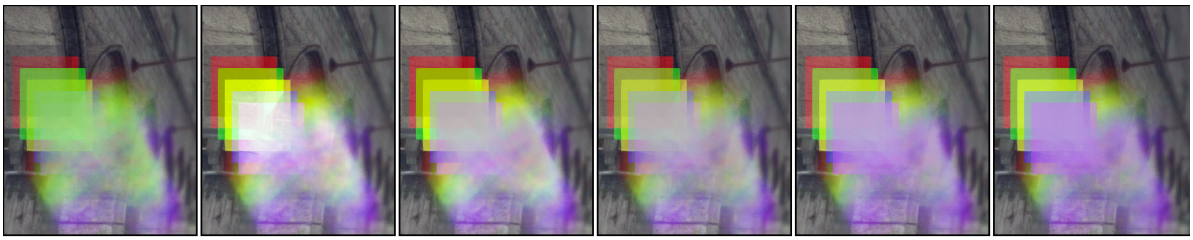


Figure 1. Six colored squares and particle billboards with $\alpha = 0.35$ at different depths in Sponza. From left to right: The unsorted OVER worst case, blended order-independent transparency approximations of increasing quality, and the common sorted OVER compositing.

Abstract

Many rendering phenomena can be modeled with partial coverage. These include flames, smoke, hair, clouds, properly-filtered silhouettes, non-refractive glass, and special effects such as forcefields and magic. A challenge in rendering these is that the value of pixel partly covered by multiple surfaces depends on the depth order of the surfaces. One approach to avoid the cost of storing and sorting primitives or fragments is to alter the compositing operator so that it is order independent, thus allowing a pure streaming approach.

We describe two previous methods for implementing blended order-independent transparency, and then introduce two new methods derived from them. Both new methods guarantee correct coverage of background and strictly improve color representation over the previous methods. Because these require only classic OpenGL-style blending and bounded memory, they may be preferred to A-buffer like methods for mobile devices, consoles, and other constrained rendering environments. They are attractive for all platforms for models such as particle systems and hair, where discrete changes in surface ordering that will be perceived as popping are undesirable and a soft transition between surfaces is preferred.

 ISSN 2331-7418

1. Partial Coverage

A surface is said to exhibit **partial coverage** of its bounds when it transmits a fraction of light without refraction and independent of the frequency of the light. Such a surface may also emit or reflect light. Partial coverage is useful for modeling surfaces with cutouts, such as a plant leaf modeled as a square with an opacity map; MIP-maps of and magnification filters (such as bilinear interpolation) applied to such surfaces; window screens and other barely-perceptible detail; layered images for compositing; antialiased boundaries of surfaces and stroked lines; and volumes comprising both fine opaque structure and empty regions, such as hair and smoke. Partial coverage alone cannot accurately model colored or refractive glass, although it is often used in practice to approximate glass. For in-depth discussion, see Porter and Duff's seminal work formalizing partial coverage [1984], Smith and Blinn's [1996] methods for computing it from photographs, and McGuire and Enderton's [2011] appendix surveying blending methods for many phenomena.

We model a surface exhibiting partial coverage with scalar coverage α and a color C (which represents radiance towards the eye in most applications) that has been premultiplied by its coverage. The value $\alpha = 1$ indicates that the surface completely covers its bounds, $\alpha = 0.5$ means that the surface covers half the area, and $\alpha = 0$ means that the surface covers none of the area. For a non-emissive surface, $C[\lambda] \leq \alpha$ for each color channel λ. For an emissive surface, the color may exceed the coverage.

An alternative is to model an unmultiplied color U and coverage α, where $C = U\alpha$. We recommend using premultiplied color for two reasons. It allows surfaces that have zero coverage to still contribute intensity to the scene by emission, which is useful for light ("god") rays and visible light cones as well as special effects. It also gives desirable coverage-weighted filtering for bilinear and trilinear filtering and MIP-map generation, thus avoiding color bleeding from $\alpha = 0$ regions into covered areas.

The final net result C_f of a background surface 0 partially covered by a foreground surface 1 is given by Porter and Duff's **OVER** operator for premultiplied coverage[1],

$$C_f = C_1 + (1 - \alpha_1)C_0 \tag{1}$$

which is pervasive in graphics and is often simply called "back-to-front alpha blending" or "compositing". This method can be applied iteratively in back-to-front order to composite multiple surfaces:

$$C_f = \Big[C_n + (1 - \alpha_n) \cdots \Big[C_2 + (1 - \alpha_2)\big[C_1 + (1 - \alpha_1)C_0\big]\Big] \cdots \Big] \tag{2}$$

Porter and Duff were careful to note that the OVER compositing operator assumes that partially-covered locations are distributed in a statistically independent manner

[1] For unmultiplied color values, $C_f = \alpha_1 U_1 + (1 - \alpha_1)C_0 = C_0 + \alpha_1(U_1 - C_0)$.

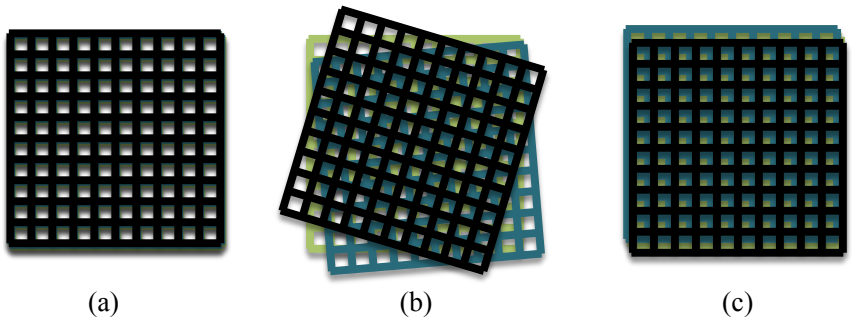


Figure 2. Three identical window screens, each with 50% coverage. a) Perfect alignment by stacking produces 50% net coverage. b) Arbitrary misalignment produces net coverage between 50% and 100%. c) Perfect alignment with offsets produces 100% coverage.

between surfaces. For example, in figure 2a, the series of n exactly-aligned window screens, each with coverage α, violates this assumption. In that diagram, the net coverage of the background is α, not $(1-(1-\alpha)^n)$ that OVER predicts. In contrast, the arbitrarily misaligned screens in figure 2b satisfy the assumption of OVER. Note that it is also possible to produce 100% coverage of the background with only a finite number of layers whose opaque portions are aligned as shown in figure 2c. The same alignment arguments hold for volumes of glass and smoke, although there one has to consider the interactions of individual photons and coverage by molecules instead of macroscopic "opacity." Because the sorted OVER operator relies on the assumption of arbitrary alignment, OVER compositing is not a ground truth solution for partial coverage. Therefore, differences from the OVER result (e.g., in figure 1) are not necessarily errors. We still compare against sorted OVER compositing because it an industry-accepted approximation and any new solution will be evaluated for deviations from that as well as from ground truth.

Because the OVER operator is not commutative, to evaluate it one must either ray trace a pixel, successively "peel" depth layers [Everitt 2001; Bavoil and Myers 2008] by rasterization, split and then traverse primitives in back-to-front order for rasterization [Fuchs et al. 1980], or accumulate a list of fragments [Carpenter 1984; Mark and Proudfoot 2001] for subsequent sorting and compositing. Each of these carries a time and memory cost, and in some cases requires special hardware [Salvi et al. 2011] for real-time rendering.

2. Order-Independent Transparency

A number of approaches have been developed to approximate compositing without explicit ordering. These are commonly called **order-independent transparency** (OIT) methods. Some key ideas exploited by different families of these methods are:

- Convert α to binary coverage and store many samples per pixel [Young 2006; Enderton et al. 2010].
- Store a subset of the closest surfaces [Jouppi and Chang 1999; Bavoil et al. 2007; Myers and Bavoil 2007; Maule et al. 2013].
- Form a continuous model of density over depth [Lokovic and Veach 2000; Sintorn and Assarsson 2009; Jansen and Bavoil 2010; Salvi et al. 2011]
- Redefine the compositing operator so that its arguments commute, which we call **blended** order independent transparency.

The first three categories have been heavily explored, and we cite only a few representative techniques. They focus on image quality when working with relatively advanced GPUs. The last category instead seeks simplicity. Compositing operators that commute necessarily create different images than ordered OVER compositing, but in exchange they gain features required for some applications and target platforms:

1. No specialized hardware: only requires render-to-texture and conventional blending (which are supported on WebGL, mobile GLES, and all current consoles).
2. Low memory requirements: uses bounded, small additional space (3 to 5 additional scalars in one or two textures).
3. No popping when surfaces pass through one another.

We now give a simple example of the last feature, which Sintorn and Assarsson [2009] noted is important even on the most advanced GPUs. Consider smoke modeled as a particle system and rendered with an A-buffer. The A-buffer [Carpenter 1984] gives *correct* results for partial coverage. When a high-opacity dark smoke particle moves away from the viewer and passes through a lighter particle, the color of the pixels it covered instantly switch from dark to light after resolving the A-buffer. This is an artifact of modeling the smoke with too few particles–ideally the smoke model would be comprised of one particle per smoke molecule. However, any practical model will have many fewer and much larger particles than real smoke. Under that modeling limitation, an accurate rendering of the poor model is undesirable. A temporally coherent result, such as the one produced by any of the blended OIT methods, is preferable. The smooth transitions better approximate the intended scene rather than faithfully exposing limitations of the scene model. Hair or fur modeled as a set of thin tubes has the same property, since the model usually has far fewer individual hairs than found on a real head or pelt.

3. Blended OIT

3.1. Meshkin's Method

Meshkin [2007] was the first to introduce blended OIT by formulating a commuting compositing operator. The higher-quality and simpler of his two operators is a "weighted sum":

$$C_f = \left(\sum_{i=1}^{n} C_i\right) + C_0\left(1 - \sum_{i=1}^{n} \alpha_i\right). \tag{3}$$

This can be implemented on a graphics API like OpenGL for a floating point render target by:

```
drawOpaqueSurfaces();
copyColorBufferToTexture(C0Texture);

glDepthMask(GL_FALSE);
glEnable(GL_BLEND);
glBlendFunc(GL_ONE, GL_ONE);
bindFragmentShader("
  ...
  uniform sampler2D C0Texture;
  void main() {
     ...
     vec3 C0 = texelFetch(C0Texture, ivec2(gl_FragCoord.xy), 0).rgb;
     gl_FragColor = vec4(Ci - ai * C0, 1.0);
  }", C0Texture);
drawTransparentSurfaces();
```

Listing 1. Meshkin's Method

For render targets that do not support blending of negative values, the positive and negative terms can be processed in separate passes or the α values accumulated in a separate pass.

The results are close to back-to-front sorted OVER compositing when α is small and all colors are similar. When α is large, the intensity and net coverage of the background may deviate significantly from the OVER operator. As shown in figure 1, they can even lie outside of the gamut of the original surface and background colors.

3.2. Bavoil's and Myer's Method

Bavoil and Myer [2008] improved Meshkin's operator with a better approximation of both coverage and color[2]. Their "weighted average" operator,

$$C_f = \frac{\sum_{i=1}^{n} C_i}{\sum_{i=1}^{n} \alpha_i} \cdot \left(1 - \left[1 - \frac{1}{n}\sum_{i=1}^{n} \alpha_i\right]^n\right) + C_0 \left[1 - \frac{1}{n}\sum_{i=1}^{n} \alpha_i\right]^n \tag{4}$$

can be implemented as:

```
drawOpaqueSurfaces();

bindFramebuffer(accumTexture, countTexture);
 glDepthMask(GL_FALSE);
glEnable(GL_BLEND);
glBlendFunc(GL_ONE, GL_ONE);
bindFragmentShader("...
  gl_FragData[0] = vec4(Ci, ai);
  gl_FragData[1] = vec4(1);
  ...}");
drawTransparentSurfaces();
unbindFramebuffer();

glBlendFunc(GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA);
bindFragmentShader("...
  vec4 accum = texelFetch(accumTexture, ivec2(gl_FragCoord.xy), 0);
  float n = max(1.0, texelFetch(countTexture, ivec2(gl_FragCoord.xy), 0).r);
  gl_FragColor = vec4(accum.rgb / max(accum.a, 0.0001),
      pow(max(0.0, 1.0 - accum.a / n), n));
  ...}", accumTexture, countTexture);
```

Listing 2. Bavoil's and Myer's Method

3.3. A New Blended OIT Method

The operator from equation 4 averages the coverage values of all composited surfaces. This means that introducing surfaces with $C = 0, \alpha = 0$, which should be completely invisible, will change the resulting image by overriding the color of surfaces with much higher coverage. We extend the previous operator to compute *exact* coverage of the background, which also improves the color term. Our new operator is

$$C_f = \frac{\sum_{i=1}^{n} C_i}{\sum_{i=1}^{n} \alpha_i} \left(1 - \prod_{i=1}^{n}(1-\alpha_i)\right) + C_0 \prod_{i=1}^{n}(1-\alpha_i) \tag{5}$$

We describe an implementation in the following section. The net coverage of the background is that of Porter and Duff (but obtained without requiring explicit order-

[2]The racing game *Pure* [Moore and Jefferies 2009] later used a more ad hoc order-independent blending method for low-contrast foliage inspired by this.

ing) and the same as Sintorn et al. and Enderton et al. (but obtained in a single pass over the geometry and without requiring multisampling.)

3.4. Depth Weights Improve Occlusion

The color of the combined partial coverage surfaces in equation 5 is always dominated by the surfaces with the highest coverage, regardless of where they appear in the depth ordering. Also, multiple surfaces with different colors and similar coverage will yield their average color regardless of ordering. Consider the case of a white cloud passing in front of a dark cloud. We desire a result that is closer to the white cloud in that case, and the opposite were their depths exchanged.

To achieve this goal, we extend our new blending method with weights for the colors that decrease with distance from the camera:

$$C_f = \frac{\sum_{i=1}^{n} C_i \cdot w(z_i, \alpha_i)}{\sum_{i=1}^{n} \alpha_i \cdot w(z_i, \alpha_i)} \left(1 - \prod_{i=1}^{n} (1 - \alpha_i)\right) + C_0 \prod_{i=1}^{n} (1 - \alpha_i) \tag{6}$$

Assume that z is a camera-space value that is zero at the center of projection and decreases away from the camera towards $-\infty$. Any monotonically decreasing, non-zero function of $|z|$ on the expected depth range, such as $w(z,\alpha) = |z|^{-k}$ or $w(z,\alpha) = z - z_{\text{far}} + \varepsilon$, grants nearer surfaces higher weights. It thus creates an occlusion cue between transparent surfaces. One can also choose a function of the homogenous clip-space z value instead of a linear camera space z. Independent of the weight function, the net coverage of the background remains exactly correct and the color will always be an interpolation of colors from the transparent surfaces–it never extrapolates beyond them, so the result is always plausible. Note that choosing $w(z,\alpha) = 1$ simply reduces equation 6 back to equation 5.

The weight function acts like an estimator of the occlusion of each surface, allowing a surface to moderate its own contribution by assuming a uniform distribution of other surfaces between it and the viewer, without explicit (order-dependent) information about those surfaces. When that estimate is incorrect (e.g., because there are no other surfaces), the normalization term corrects the net contribution. In this way, it is a stateless heuristic approximating the true visibility function, comparable to Enderton et al.'s [2010] stochastic visibility estimator. We even considered calling it $v()$ to match their notation, but chose to call this factor a weight instead of a true visibility because it is on an arbitrary scale that requires normalization during compositing. Under the assumption of uniform density of the medium, the correct visibility heuristic would be exponential falloff according to the Beer-Lambert law (as with atmospheric perspective). Scenes with small numbers of discrete surfaces or clumps of smoke as in our figures do not have uniform density. Furthermore, a true exponential function's values would overflow available precision in a 16-bit floating point buffer near the camera, and underflow precision far from the camera. So, we recommend rational

polynomial weights that exhibit slower asymptotic growth.

If the weight function depends solely on z, a surface very near the camera with so low coverage as to be imperceptible during ordered compositing can then undesirably color distant and more opaque surfaces. Thus we recommend making the weight function also decrease with coverage.

For individual result figures in this paper we give weight functions that we tuned for those scenes. We also recommend three generic weight functions appropriate for arbitrary scenes with large depth ranges. We tuned these to work well for 16-bit floating point accumulation buffers with $0.1 \leq |z| \leq 500$:

$$w(z,\alpha) = (\alpha+0.01)^4 + \max\left[10^{-2}, \min\left[3\times10^3, \frac{100}{10^{-5}+(|z|/5)^2+(|z|/200)^6}\right]\right] \quad (7)$$

$$w(z,\alpha) = (\alpha+0.01)^4 + \max\left[10^{-2}, \min\left[3\times10^3, \frac{100}{10^{-5}+(|z|/10)^3+(|z|/200)^6}\right]\right] \quad (8)$$

$$w(z,\alpha) = (\alpha+0.01)^4 + \max\left[10^{-2}, \min\left[3\times10^3, \frac{0.3}{10^{-5}+(|z|/200)^4}\right]\right] \quad (9)$$

$$w(z,\alpha) = (\alpha+0.01)^4 + \max\left[10^{-2}, \min\left[3\times10^3, 10^{10}(1-d(z))^3\right]\right] \quad (10)$$

The exponents can all be evaluated with repeated products–true exponentiation is not required. In equation 10, $d(z)$ is the value in `gl_FragCoord.z` during OpenGL fragment shader execution,

$$d(z) = \frac{(z_{\text{near}}z_{\text{far}})/z - z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \quad (11)$$

where all z values are negative in camera space. Note that for the common case of billboards parallel to the camera's view axis, the more expensive portion of these weight values can be computed once per billboard at the vertex stage instead of once per pixel, since they are uniform with respect to z.

Without the range clamping in these functions, underflow will result in overly-dark areas where α is small and $|z|$ is large. Overflow will result in infinity (which renders as black on most GPUs) where $|z|$ is small. The maximum finite 16-bit floating point number is 6.55×10^4, so even with the clamping, these will saturate after about 20 layers close to the camera and during compositing the accumulated weight result should be clamped as shown in listing 3.

Listing 3 gives an implementation of both of our new blending methods, where the "revealage" (as opposed to "coverage") texture tracks the amount of background revealed, instead of the amount covered as in a coverage mask. All DX11-class GPUs support the per-render target blending functions required for the implementation. For older devices (e.g., some DX9-generation consoles) that do not have independent blending, listing 4 refactors the texture storage so that a single blending function with separate blending of the alpha channel can be specified.

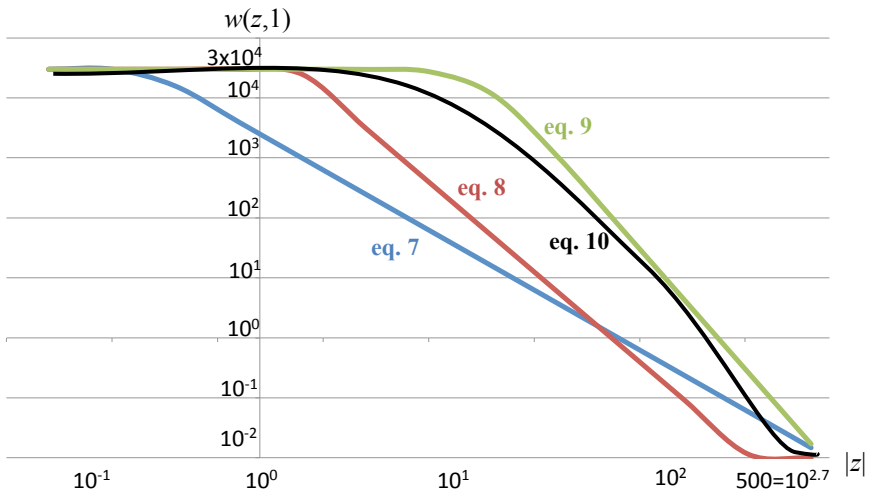


Figure 3. Log-log plots of sample weight functions. Pushing the knee of the curve towards the upper right helps avoid silhouetting of objects in the mid-range at the expense of discrimination between different transparent surfaces that are either very close or very far from the camera.

```
drawOpaqueSurfaces();

clear accumTexture to vec4(0), revealageTexture to float(1)
bindFramebuffer(accumTexture, revealageTexture);
glDepthMask(GL_FALSE);
glEnable(GL_BLEND);
glBlendFunci(0, GL_ONE, GL_ONE);
glBlendFunci(1, GL_ZERO, GL_ONE_MINUS_SRC_ALPHA);
bindFragmentShader("...
  gl_FragData[0] = vec4(Ci, ai) * w(zi);
  gl_FragData[1] = vec4(ai);
  ...}");
drawTransparentSurfaces();
unbindFramebuffer();

glBlendFunc(GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA);
bindFragmentShader("...
  vec4 accum = texelFetch(accumTexture, ivec2(gl_FragCoord.xy), 0);
  float r = texelFetch(revealageTexture, ivec2(gl_FragCoord.xy), 0).r;
  gl_FragColor = vec4(accum.rgb / clamp(accum.a, 1e-4, 5e4), r);
  ...}", accumTexture, revealageTexture);
```

Listing 3. Our New Method

```
drawOpaqueSurfaces();

bindFramebuffer(A, B);
glColorClearValue(0,0,0,1);
glClear();
glDepthMask(GL_FALSE);
glEnable(GL_BLEND);
glBlendFuncSeparate(GL_ONE, GL_ONE, GL_ZERO, GL_ONE_MINUS_SRC_ALPHA);
bindFragmentShader("...
  gl_FragData[0] = vec4(Ci * w(zi), ai);
  gl_FragData[1].r = ai * w(zi);
  ...}");
drawTransparentSurfaces();
unbindFramebuffer();

glBlendFunc(GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA);
bindFragmentShader("...
  vec4 accum = texelFetch(ATexture, ivec2(gl_FragCoord.xy), 0);
  float r = accum.a;
  accum.a = texelFetch(BTexture, ivec2(gl_FragCoord.xy), 0).r;
  gl_FragColor = vec4(accum.rgb / clamp(accum.a, 1e-4, 5e4), r);
  ...}", A, B);
```

Listing 4. Our New Method for Platforms without Per-Render Target Blending

For *monochrome* transparent surfaces (e.g., neutrally-light window glass, fog, and smoke), all values can be packed into a single render target as `vec4(vec2(Ci.r, ai) * w(zi, ai), 0, ai)` under listing 4 to reduce bandwidth or support platforms that do not allow multiple render targets.

We've assumed partial coverage transparency throughout, where the background intensity is attenuated during compositing but its hue is unaltered. When bandwidth is not the bottleneck, non-refractive *colored* transmission can be implemented as a simple extension by processing a separate coverage value per color channel. This is similar to the way that colored stochastic shadow maps are implemented [McGuire and Enderton 2011].

If using soft particle [Lorach 2007] blending against opaque surfaces, then scale both `Ci` and `ai` by the softness weight at the top of the first fragment shader.

As shown in figure 1, in some scenes this new depth-weighted blended OIT is extremely close to the OVER operator applied with sorting. In addition to not requiring sorting, it has the further advantages of operating per-pixel (since primitive sorting without splitting is itself not correct in all cases) and of smoothly transitioning when transparent surfaces pass through each other.

For the specific cases of particle systems and hair, we note that many video games use multiple particle systems and hair systems per scene and submit each in its own

draw call. In this case, one can coarsely sort entire systems on the GPU and then make one transparent rendering and compositing pass per system. Within that pass, depth-weighted blended OIT precludes the need for sorting individual particles or hair. Because the depth extent of the individual system is small and known, in this case one can tune a very effective depth weighting function.

3.4.1. Limitations

The quality of the approximation for the new method with depth weights is worst in scenes with large depth ranges that contain tight clusters of differently-colored transparent surfaces submitted in a single rendering pass. In this case, it is hard to choose a depth weight function that will discriminate within a cluster as well as between clusters. The results remain robust and strictly more accurate than the previous blended OIT methods in this case, however.

Unlike the results of our blended OIT without depth weights, depth-weighted results are surface order-independent but not translation-invariant along the depth axis. For example, moving the entire scene to a different depth range can change its color. This is acceptable for some applications because the color transition will occur slowly as the objects move in z. Furthermore, the results will approach our blended method's results in the distance and give the best occlusion discrimination in the foreground when rendered with the weight functions that we describe.

To give some intuition for when our method may produce less satisfying output, we frame its implicit assumptions:

- Partially-covered locations are not correlated between layers (the Porter and Duff assumption)
- Surfaces either have similar colors or are distributed relatively uniformly in depth
- Color precision is most important for nearby surfaces, but accuracy is important throughout the depth range

We call these "implicit" because we did not use them in the derivation.

3.4.2. Upsampling

For transparent surfaces modeling volumes, such as particle systems and light rays, it is a common practice to compute a low resolution image with its own coverage from these surfaces and then upsample and composite it over the framebuffer containing the image of the opaque surfaces [Nguyen 2007]. Bilinear biliateral and gaussian bilateral schemes using the depth buffer as the biliateral key are common choices. Tatarchuk et al. [2013] recently introduced a more sophisticated depth variance metric.

When using blended OIT, it is important to perform the coverage normalization process before applying upsampling weights. This can be done by explicitly resolv-

ing the `accumTexture` and `revealageBuffer` into a texture using the second shader from listing 3 and then upsampling and compositing that texture, or by integrating the normalization directly into the upsampling process when each value is read. If the coverage normalization is incorrectly performed *after* upsampling, then the edges of volumes will be too dark.

Enabling alpha testing for all passes can decrease the fill rate required at the expense of disabling early depth testing and hierarchical depth testing on most GPUs.

4. Examples

Figure 1 shows the increased color and coverage accuracy of both of our new methods compared to previous blended OIT methods. The billboards in this scene are within a well-defined depth range of 1 to 5 scene units, where the far plane is around 100 units. That is an easy case to tune a depth weighting function for, and those weights allow image (e) to closely approximate the occlusion ordering of the ground truth image. We chose $w(z) = [|z| + 5]^{-2}$ as the weight function. Squaring helps with discrimination of more distant values because, for example, the difference between 1/2 and 1/3 is much greater than between 1/99 and 1/100.

Figure 4. San Miguel rendered with our blended OIT and depth weights of $w(z,\alpha) = z^{-4}$.

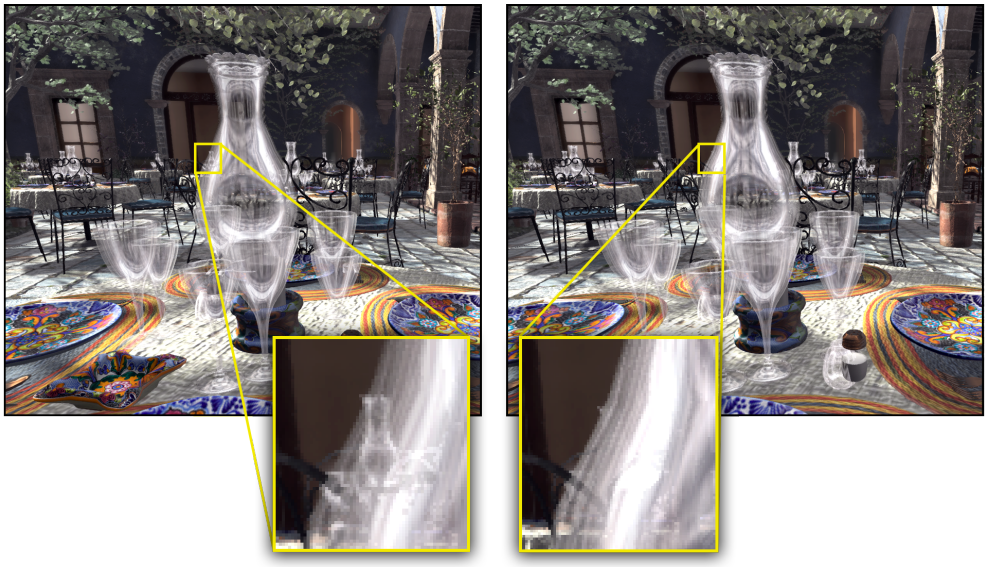


Figure 5. Left: our blended OIT. Right: our blended OIT with depth weights.

Figure 4 shows the San Miguel scene, which has many transparent surfaces. Visible in this scene are glass vases and silverware, wine glasses, window glass, water, glass hanging lanterns and glass bulbs *inside* the lanterns.

Figure 5 shows a closeup of one of the table arrangements, which has many layers of transparency (the windows in the background are also glass). The left of the figure shows our blended OIT without depth weights. In the zoomed detail it is evident that

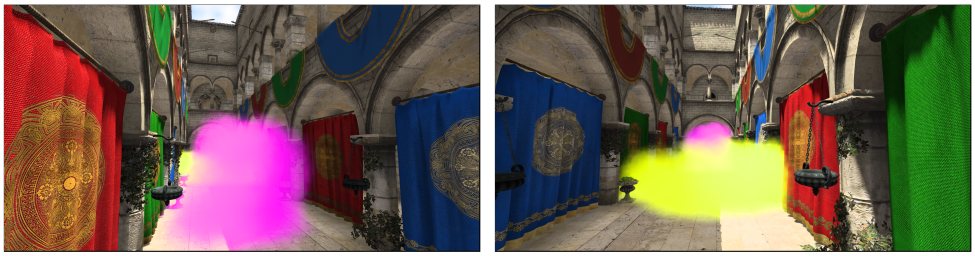


Figure 6. Two differently-colored smoke puffs in Sponza, viewed from different directions. Depth weighting by $w(z,\alpha) = \alpha^2 z^{-6}$ gives the correct occlusion cue.

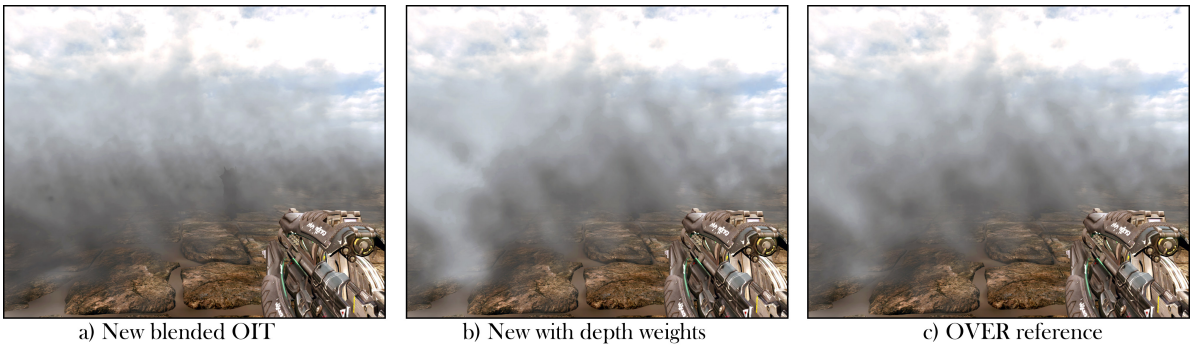
Figure 7. Fog in Sponza. Left: our blended OIT. Right: with depth weights.

the vase from a distant place setting is insufficiently occluded by the nearby glass. The right of the figure shows the same scene rendered with $w(z) = z^{-4}$. Now in the detail the reflections on the nearby glass are clearer because the distant glass has been more attenuated by the nearer one, giving a stronger proximity cue.

Figure 6 shows how depth weighting gives correct occlusion cues in a hard case of dense collections of low-coverage particles such as colored smoke. The two smoke puffs are from a single particle system. When viewed from opposite sides of the long hallway they must produce different color results.

The left image in figure 7 demonstrates a failure case of our unweighted algorithm. The figure shows the Sponza model filled with fog lit by a shaft of sunlight. Near the center of the frame a distant hanging pot and arch are incorrectly visible through the fog. The algorithm correctly computes 100% coverage for those objects. However, because the objects block some dark fog particles in the depth buffer, the blended color is brighter within their silhouettes than outside. This causes their shape to be revealed. The right image shows the same view with depth weights. The silhouettes are not visible in this case.

Figure 8a shows how that same kind of scenario can affect gameplay. The image is the player's first-person view of his or her own weapon and another character mostly obscured by a dense smoke cloud that has shading. The scene is rendered in *Unreal Engine 3*. The target character is near the center of the screen. Note that its silhouette



a) New blended OIT b) New with depth weights c) OVER reference

Figure 8. Smoke modeled as particles with a range of color values. Left: new blended OIT without depth weights. Right: Sorted OVER reference image.

a) New blended OIT *without* weights b) OVER reference

Figure 9. Smoke modeled as particles with similar color values. Left: new blended OIT without depth weights. Right: Sorted OVER reference image.

is slightly visible, revealing this "concealed" character's position. This is obviously unacceptable for games where hiding in such smoke is an intended tactic. Figure 8b shows that adding depth weighting can almost completely correct this problem. For reference, figure 8c is the ground truth image rendered by sorting and then compositing the smoke particles from back to front. The coloration is slightly different than that with depth-weighted blending, but the silhouette is equally concealed.

Another solution to the character silhouette problem is to reduce the shading variation in the transparent surfaces. Figure 9a shows the unweighted blending algorithm for a dense smoke cloud that has little shading variation. Because all smoke particle colors are similar, the color is similar inside and outside the silhouette. This enables the algorithm without weights to produce a result comparable to figure 9b, which was rendered using the reference OVER back-to-front compositing algorithm. Figure 10 shows another location in this scene, with a vehicle half-inside the smoke cloud. The upper-right and lower-left subfigures are rendered with weight $w(z,\alpha) = 10/(1+10|z\cdot k|^5)$. Note that because there is little shading variation in the smoke, changing k and even setting $w(z,\alpha) = 1$ all produce results that are comparable to the ideal sorted compositing reference image.

Introducing shading variation through smoke self-shadowing for this scene, as

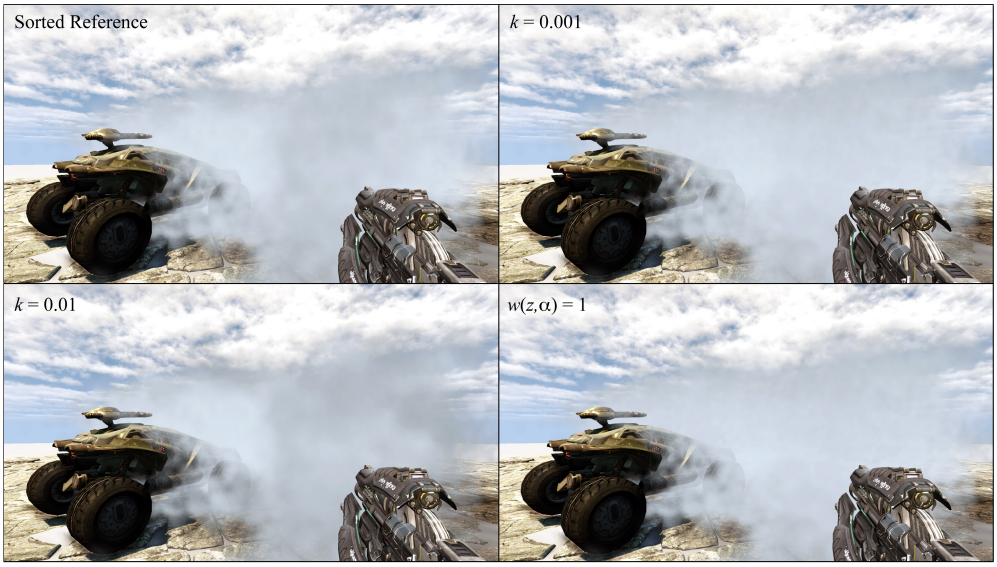


Figure 10. Smoke with little intensity difference rendered with our method and varying weighting schemes. All images are comparable to the sorted compositing reference image.

Figure 11. Smoke with contrasting self-shadowing intensities, rendered with our method under varying weighting schemes. The $k = 0.01$ result is closest to the sorted compositing reference image. The variations in the right column produce the silhouetting artifact.

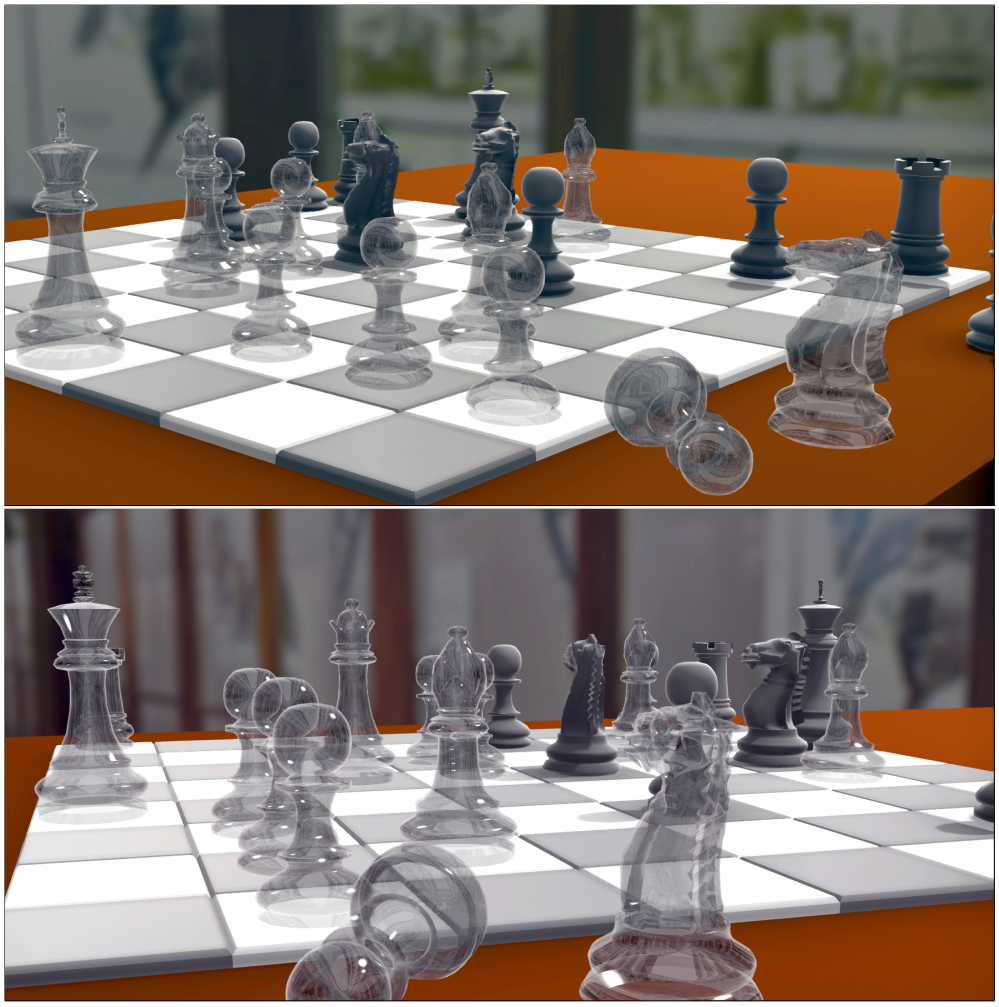


Figure 12. Two views of a chess set with glass pieces rendered using our new blended OIT with depth weight function $w(z) = |z|^{-5}$.

shown in figure 11, makes the result more sensitive to the blending function. In this case, no blending weights or improperly tuned blending weights produce the silhouette artifacts along the hood of the vehicle as seen in the right column of images. A properly tuned blending weight can still match the sorted reference image nearly perfectly, as shown in the lower left image.

Figure 12 shows two views of a scene with high depth complexity for transparent glass chess pieces. The depth weighting allows occlusion cues between the pieces, especially in the lower figure where an individual pixel may be covered by as many as six surfaces from the front and back surfaces of three pieces.

Acknowledgements

We thank the Visual Alchemy team at Vicarious Visions for discussing OIT with us and suggesting improvements to this paper. Likewise, the anonymous reviewers at JCGT provided exceptional feedback that greatly improved the version you are reading now. The Crytek Sponza scene was modeled by Frank Meinl at Crytek. The San Miguel scene was modeled by Guillermo M. Leal Llaguno. Both scenes are available from `http://graphics.cs.williams.edu/data`.

References

BAVOIL, L., AND MYERS, K. 2008. Order independent transparency with dual depth peeling. Tech. rep., NVIDIA. 124, 127

BAVOIL, L., CALLAHAN, S. P., LEFOHN, A., COMBA, J. L. D., AND SILVA, C. T. 2007. Multi-fragment effects on the GPU using the *k*-buffer. In *Proc. of SI3D'07*, ACM, New York, NY, USA, 97–104. 125

CARPENTER, L. 1984. The A-buffer, an antialiased hidden surface method. *SIGGRAPH'84 18*, 3 (Jan.), 103–108. 124, 125

ENDERTON, E., SINTORN, E., SHIRLEY, P., AND LUEBKE, D. 2010. Stochastic transparency. In *Proc. of SI3D '10*, 157–164. 125, 128

EVERITT, C. 2001. Interactive order-independent transparency. Tech. rep., NVIDIA. 124

FUCHS, H., KEDEM, Z. M., AND NAYLOR, B. F. 1980. On visible surface generation by a priori tree structures. In *SIGGRAPH'80*, ACM, New York, NY, USA, SIGGRAPH '80, 124–133. 124

JANSEN, J., AND BAVOIL, L. 2010. Fourier opacity mapping. In *Proc. of SI3D'10*, ACM, New York, NY, USA, I3D '10, 165–172. 125

JOUPPI, N. P., AND CHANG, C.-F. 1999. Z^3: an economical hardware technique for high-quality antialiasing and transparency. In *Proc. of Graphics Hardware*, ACM, New York, NY, USA, HWWS '99, 85–93. 125

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *SIGGRAPH '00*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 385–392. 125

LORACH, T. 2007. Soft particles. Tech. rep., NVIDIA, Jan. 131

MARK, W. R., AND PROUDFOOT, K. 2001. The F-buffer: a rasterization-order FIFO buffer for multi-pass rendering. In *Proc. of Graphics Hardware '01*, ACM, New York, NY, USA, HWWS '01, 57–64. 124

MAULE, M., COMBA, J., TORCHELSEN, R., AND BASTOS, R. 2013. Hybrid transparency. In *Proc. of SI3D'13*, ACM, New York, NY, USA, I3D '13, 103–118. 125

MCGUIRE, M., AND ENDERTON, E. 2011. Colored stochastic shadow maps. In *Proc. of SI3D'11*. 123, 131

MESHKIN, H., 2007. Sort-independent alpha blending, March. Perpetual Entertainment, GDC Session. 126

MOORE, J., AND JEFFERIES, D., 2009. Covering new ground: Foliage rendering rendering in pure in pure, July. SIGGRAPH 2009 Advances in Real-Time Rendering in 3D Graphics and Games Course. 127

MYERS, K., AND BAVOIL, L. 2007. Stencil routed A-buffer. In *ACM SIGGRAPH 2007 sketches*, ACM, New York, NY, USA, SIGGRAPH '07. 125

NGUYEN, H., Ed. 2007. August, ch. 23. 132

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *SIGGRAPH'84*, ACM, New York, NY, USA, vol. 18, 253–259. 123

SALVI, M., MONTGOMERY, J., AND LEFOHN, A. 2011. Adaptive transparency. In *Proc. of HPG'11*, ACM, New York, NY, USA, HPG '11, 119–126. 124, 125

SINTORN, E., AND ASSARSSON, U. 2009. Hair self shadowing and transparency depth ordering using occupancy maps. In *Proc. of SI3D'09*, ACM, New York, NY, USA, I3D '09, 67–74. 125

SMITH, A. R., AND BLINN, J. F. 1996. Blue screen matting. In *SIGGRAPH'96*, ACM, New York, NY, USA, SIGGRAPH '96, 259–268. 123

TATARCHUK, N., TCHOU, C., AND VENZON, J., 2013. Mythic science fiction in real-time: Destiny rendering engine, July. SIGGRAPH 2013 Advances in Real-Time Rendering in Games Course. 132

YOUNG, P. 2006. Coverage sampled antialiasing. Tech. rep., NVIDIA, October. 125

Author Contact Information

Morgan McGuire
NVIDIA
47 Lab Campus Drive
Williamstown, MA 01267
USA
momcguire@nvidia.com

Louis Bavoil
NVIDIA
12 avenue de l'Arche
92400 Courbevoie
France
lbavoil@nvidia.com

McGuire and Bavoil, Weighted Blended Order-Independent Transparency, *Journal of Computer Graphics Techniques (JCGT)*, vol. 2, no. 2, 122–141, 2013
http://jcgt.org/published/0002/02/09/

Received:	2013-10-22		
Recommended:	2013-11-29	Corresponding Editor:	Andrew Glassner
Published:	2013-12-19	Editor-in-Chief:	Morgan McGuire