Real-Time Shadows Through Semitransparent Objects

Anthony LaRosa

Philip Cioni

1. Abstract

Shadows are an integral component of photorealistic graphics. However, some rendering systems do not account for translucency when casting a shadow from an object. Naive approaches shade a pixel under two categories: inside a shadow, or not. As such, the shadows appear hard and uniform in darkness, causing an appearance of flatness. Semitransparent shadows give shadows depth, and provide volume to particle-based objects such as smoke and clouds.

Furthermore, graphics often rely on photorealistic rendering of shadows for smoke and particles, which is not a trivial computation. Raytracing methods struggle to cover the area in between each particle in the particle system, doubly so if the particles are reflective. For this reason, we opted to create a shaded method for creating semitransparent shadows. Shaders are low-cost compared to raytracing, and can be applied to any material in a scene.

Our paper presents a method for casting shadows through semitransparent objects in real-time using shaders. Semitransparent objects are objects with materials that have a varying alpha value between 1 and 99 percent transparency. That is, an object that is a variable level of translucency. Our paper addresses real-time shadows being cast from semitransparent materials, with the shadows having a varying darkness depending on the transparency and depth of a particle system or material.

2. Related Works

One method shown by *Lindbeck* in *"Rendering Light and Shadows for Transparent Objects*" displays a method for sending the GPU organized data in a linked list. The linked list is constructed with a Deep Shadow Map. A Deep Shadow Map is a linear probe of shadow depth within a particle cloud. A beam of light is traced through the cloud, and at preset distance intervals, the intensity of the light and the depth are recorded and appended to the end of the current linked list for this specific beam of light. When the light passes a certain distance or drops its intensity to 0, the linked list is sent to the GPU for processing. The resulting shadows are tiered in intensity, depending on how deep the Deep Shadow Map is.[1]

Other methods attempt to create semitransparent shadows through particle systems. Shadows are shaded according to the depth, density, and volume of a particle cloud. *Larsson* presented a method for shadow mapping particle shadows in *"Interactive Real-Time Smoke Rendering"*. Shadow maps were chosen since no polygonal data was available for shadow volumes. The shadows were created over nine passes, consisting of depth calculations, particle generation, normal mapping, and raytracing. This method also casts shadows on any objects between the light and the particle system. The problem was mitigated by targeting the domain of Racing Games, where fast-paced action leaves little time for the eye to discern irregularities.[2]

A method presented by *Anderson* in *"Smoke and Shadows: Rendering and Light Interaction of Smoke in Real-Time Rendered Virtual Environments"* augmented shadow maps for use with smoke shadows. His method updates the particle system to perform distance checks between a particle and the lightsource. The program then compares the distance to the light to the depth in the shadow to map to find if the pixel is in shadow, and render the pixel in the appropriate color. Our method uses an augmentation to shadow maps, but checks for differences in the alpha value of a given pixel rather than its distance from a lightsource.[3]

As for the smoke simulation part of the project, we took a lot of our inspiration from *"Real-Time Rendering of Cartoon Smoke and Clouds" (McGuire, M., & Fein, A. (n.d.))*[4]. In that paper, the authors used billboarded sprites preprocessed from meshes before rendering as particles. Shadows are rendered from proxy geometry that is not rendered to the viewer, and whole shadow volumes based on meshes are avoided.

3. Semitransparent Objects

The semitransparent materials themselves are used to generate shadows of variable darkness, giving the appearance of depth to objects and particle systems. Semitransparent objects are represented by either a particle system of varying depth and density, or a object shaded to have varying transparency. Some shadow methods, such as shadow maps, only visualize shadows in one of two categories: a shadow is present, or not. Additionally, fully transparent objects, such as glass, cast a shadow, but still reflect noticeable amounts of light. We aim to improve the categorization of shadow maps to include variable levels of darkness in shadows, and to incorporate reflectivity into the intensity of shadows.

4. Implementation

4.1 Cutout Rendering

To create our transparent material, we calculate the transparency of each pixel (fragment). The transparency in this case is the product of the color tint and the texture's alpha value at a given point(x,y). In order to create this array of transparencies, we use a transparency map: a solid color texture with fading, smooth noise in the alpha channel. The transparency map is user-defined and can be changed prior to blending.

A non-standard transparency map with more than two colors on a gradient will function, but may provide unexpected results. To create our transparency maps, we took patterned textures and zeroed their saturation.

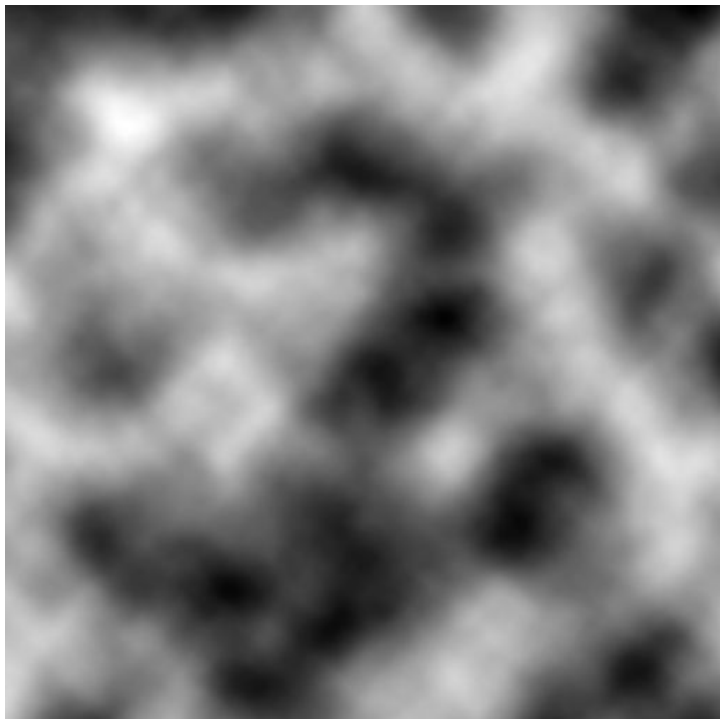

Figure 1: Example transparency map resembling noise[5]

A pixel is then determined as either fully opaque, or fully transparent. A user-defined alpha cutoff value determines what category a pixel falls in. These dynamic alpha values feign a “flowing” fluid smoke effect, especially when combined with a continuous noise function to generate your alpha maps. This allows us to create a semitransparent object out of any visible material by simply attaching the shader to it.

Figure 2: Varying Alpha Cutoffs

4.2 Fading vs. Transparency

Cutout shadows are per pixel, which means the edges are aliased and there is no smooth interpolation between transparency levels. Naively fading the edges is fine for single objects, but overlapping materials will render incorrectly on top of eachother. Additionally, cutout shadows do not correctly represent solid partially reflective objects, such as glass. Glass is fully transparent, but also contains highlights and reflections. If an object is both reflective and transparent, we will see both what’s behind it, and the reflections. However, the same light cannot pass through an object and be reflected. If an object is more reflective, less light will pass through it. We represent this by adjusting the alpha value prior to color blending, in a process known as premultiplied alpha blending.

We adjust the alpha value prior to GPU blending. If a material has 0% reflectivity, its alpha remains unchanged. If it reflects 100% of light, it’s alpha becomes effectively one.

Given an original alpha value α and a percent reflectivity *r*, the modified alpha is represented as:

$$1 - (1 - \alpha)(1 - r)$$

4.3 Cutout Shadows

The shadows of our transparent objects are still being cast as if the objects are made of a solid material. To attain semitransparent shadows, we have to access the alpha value in the shadow caster shader pass. To this, we sample the albedo texture. First, we cut holes in the shadows with the same method that we used in 4.1. Next, pipe the color tint, albedo texture, and alpha cutoff settings into the shadow shader. We can then retrieve the alpha value from the fragment shader, and use it to clip the “cut” parts of the shadow.

Figure 3: Cutout Shadows

4.4. Partial Shadows and Dithering

As discussed earlier, standard shadow maps are calculated using the distance between surfaces and their shadows. With this method, light is either blocked at a point, or not; there is no way to specify partially blocked light.

Building from cutout shadows, which clips pixels according to a threshold, we instead clip pixels uniformly. Depending on a surface's reflectivity, we use a checkerboard pattern to determine how much light is let through. For example, if an object lets 50% of light through, the checkerboard would be half one color, and half another color, making the resulting shadow appear half as dark as a full shadow.

Depending on the alpha value, we can choose an appropriate density checkerboard, where stronger (darker) shadows contain less “clipped” squares (squares that indicate a pixel should be clipped). This process of using two states to approximate a gradient is known as dithering.

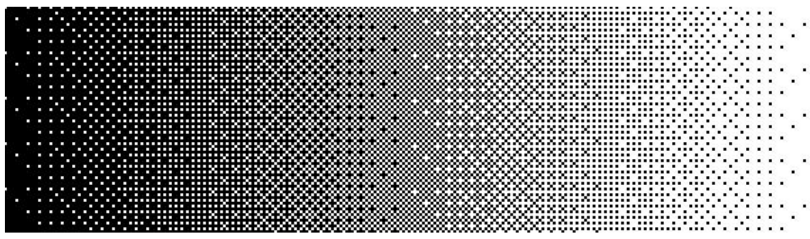

Figure 4: Example Dithering Patterns

We need to sample a shadow to dither it. UV mesh coordinates are not uniform in shadow space, so we must use screen-space coordinates of the sample pixel. The screen-space position of a pixel can be easily accessed within the fragment shader.

Instead of using a uniform dithering pattern, we base the dithering selection on the material's alpha value. Where *i* is a pixel in a shadow and *s* is the dither sample's scale, a shadow's dithering sample is found with:

PointOnTexture(DitherMaskLOD, float3(i.vpos.xy · s, alpha)).a

This will make the dithering vary based on the material opacity. Scaling down the pattern size makes the shadows appear softer and less blocky, as the dither sample cubes are smaller in size, and more dense.

4.5. Fluid System

To round out our implementation, we used a third party fluid system to represent a fluid system of particles. The fluid system had parameters for the number of particles, fluid density, gravity constants, and force over time. We used a third party open-source system in order to achieve realistic fluid dynamics far more realistic and quickly then we would have on our own.

The system simulates approximately 150 particles in real time, and for each particle we attach a particle object that contains our particle cutouts, preloaded into an object pool at start up. Particle cutouts billboard, or rotate to face, the viewport, while transparent copies of those cutouts are rotated to face light sources and cast shadows.

5. Results

Our method produces semitransparent shadows that vary in darkness depending on the transparency and reflectivity of the material casting them. Our shader is very portable in that it can be applied to almost any material, provided the material has the appropriate alpha channels and colors.

While materials will more than a binary color palette can be used, the results are unpredictable, as the alpha value cannot be accurately constrained to a bipolar gradient.

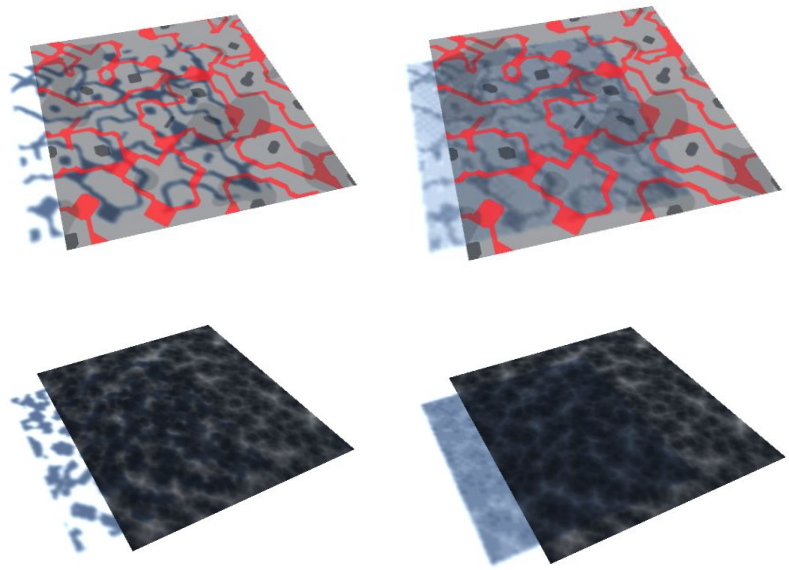

Figure 5: Cutout Shadows (Left), and Semitransparent Shadows (Right)

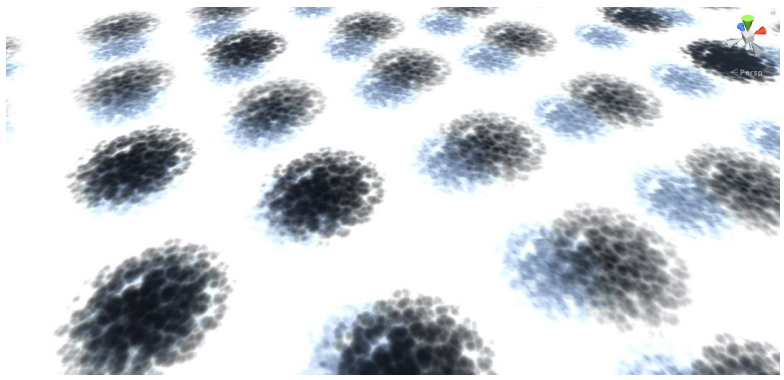

Figure 6: 100 Materials with Semitransparent Shadows

Our shader also has trace performance impacts. A scene of 10, 100, and 1000 duplicate shaded materials had infinitesimal runtime differences.

The fluid system was more tunable and realistic than what we could have done. It extending a cell based, discrete interpolated particle systems created previously. Even so, while the shading of the shadowing was successful, the particle system did not match the behavior of real smoke

5.1 Drawbacks

Dithering is not visually stable, meaning when visible geometry in the scene moves, the shadows shimmer as if behind interference. This is due to the way we sample dithering patterns. Our setup views dithering as if the pattern is repeated endless on a surface, meaning that samples are not placed onto the surface, but rather made visible underneath the surface itself, as if looking outside different angles of a window. This problem could potentially be resolved by creating a custom dithering renderer that projects patterns rather than revealing them. Additionally, the shader could "bake" the shadows onto a surface and only recompute the shadow when the object itself changes position, instead of whenever anything (namely the camera) moves.

5.2 Future Considerations

A custom dithering system would help make the shadows smoother and less “dotty”. Finer control over the shape of the dithering pattern would also help diminish the shimmering effect from moving dithered objects, as the pattern’s sample could be made static.

Our color blending mode in 4.2 is also not robust. This method does not take into account the physical volume of an object, and instead only uses the visible surface to approximate and feign volume. Precomputing the volume of an object would create a more realistic transparency effect, and provide a slight speed increase since we no longer need to adjust our alpha value.

The verisimilitude of our method must also be addressed. Our method, albeit passable, is not photorealistic. There is a distinct difference in the way the shadows appear, namely in the blocky edges of the shadow. To attain closer photorealism, we would need to find an alternative to dithering entirely, as dithering will always sample a grid regardless of resolution. As a middle ground, an antialiasing pass could be added post-dithering in order to try to soften the corners of the dither grid.

6. References

1. Lindbeck, Isak. "Rendering Light and Shadows for Transparent Objects." (2013): 1-48. *Http://lup.lub.lu.se*. Lund University, 13 Apr. 2013. Web. 20 Apr. 2017.

2. Larsson, Robert. "Interactive Real-Time Smoke Rendering." (2010): 1-62. *Http://www.cse.chalmers.se*. Chalmers University, Apr. 2010. Web. 20 Apr. 2017.

3. Anderson, Eike Falk, and Christopher J. Bass. "Smoke and Shadows: Rendering and Light Interaction of Smoke in Real-Time Rendered Virtual Environments." *Proceedings of VS-GAMES 2013: 5th International Conference on Games and Virtual Worlds for Serious Applications* (2013): 146-49. *ResearchGate*. Web. 20 Apr. 2017.

4. McGuire, M., & Fein, A. (n.d.). Real-Time Rendering of Cartoon Smoke and Clouds. Retrieved April 20, 2017, from https://pdfs.semanticscholar.org/085c/880e0029daa95475618794ee9ff013ec8379.pdf.

5. *Noise Map. Digital image. Kranwasser.de. N.p., n.d. Web. 30 Apr. 2017. <http://kranwasser.de/wp-content/uploads/2010/01/map_d.png>.*

6. *Bayer Dither. Digital image. Www.wasatch.com. N.p., n.d. Web. 30 Apr. 2017. <https://www.wasatch.com/halftoneImages/bayerDither.gif>.*