Light Shafts

Rendering Shadows in Participating Media

Jason Mitchell
3D Application Research Group
ATI Research, Inc.

Overview

- A few examples in content
  - TV, film and games
- Real-time technique
  - Volume sampling
  - Non-uniform density
  - Antialiasing
  - Future directions

The Effect

- We see light that reaches our eyes, so how can we see shafts of light?
- The light is scattering off of some particles suspended in the media through which it is passing (or the media itself)
- Shadows in this scenario, especially dynamic ones, have a really dramatic look
- This is used very frequently in film as well as intros and logos of all sorts from games to movies
- Games already try to do this in-game
- There is also some academic and industry research in the area

Stage Lighting

- Does not necessarily have to cast shadows
- Can be distant “decoration”

Large scale rays through particles in atmosphere

Small Non-shadowing Shafts

Some In-Game Examples

- Practically every game does this
- Here are some representative games in which I've noticed this
  - *Zelda: The Wind Waker*
  - *Splinter Cell*
  - *Tomb Raider: The Angel of Darkness*
  - *I.C.O*

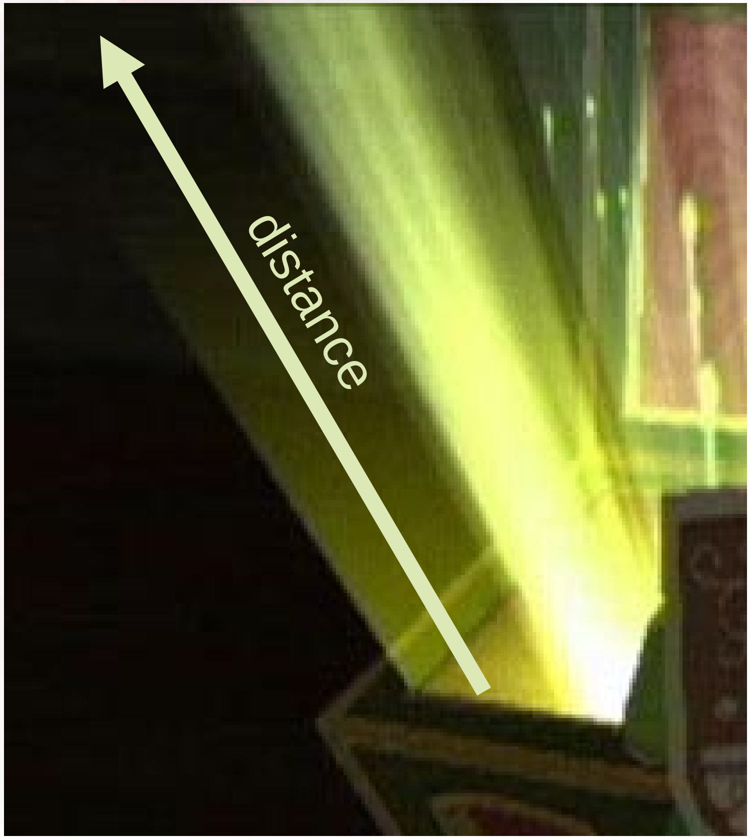
From *Zelda: The Wind Waker*

What are they doing in Zelda?

- Probably the simplest thing you could think of
- Additive blending of polygons extruded from the light source
- They’re drawn last and just z-buffered against the scene
- Attenuating brightness with distance

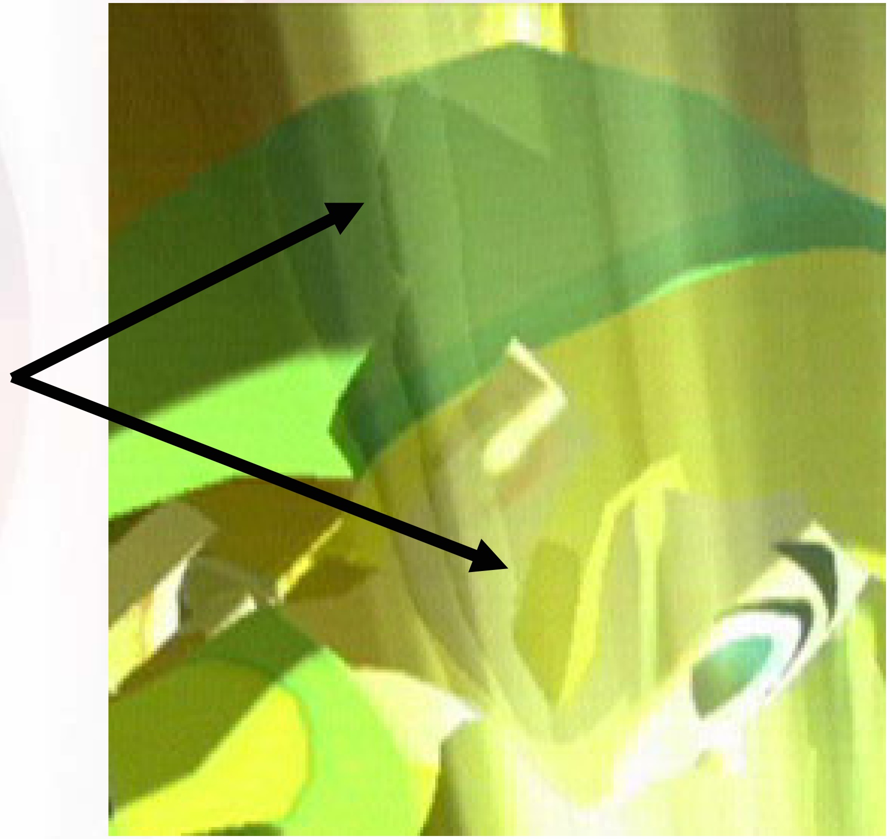
Distance Attenuation

- Probably a good idea no matter what technique is used
- We model this for lights even when we ignore scattering due to particles in media
- Can be an efficiency win



Z-Buffering Against Scene

- Scene is rendered before light shaft geometry
- Planes of light shafts leave obvious lines where they intersect scene geometry



Clearly an important visual cue

- From *Tomb Raider: Angel of Darkness*

I.C.O.

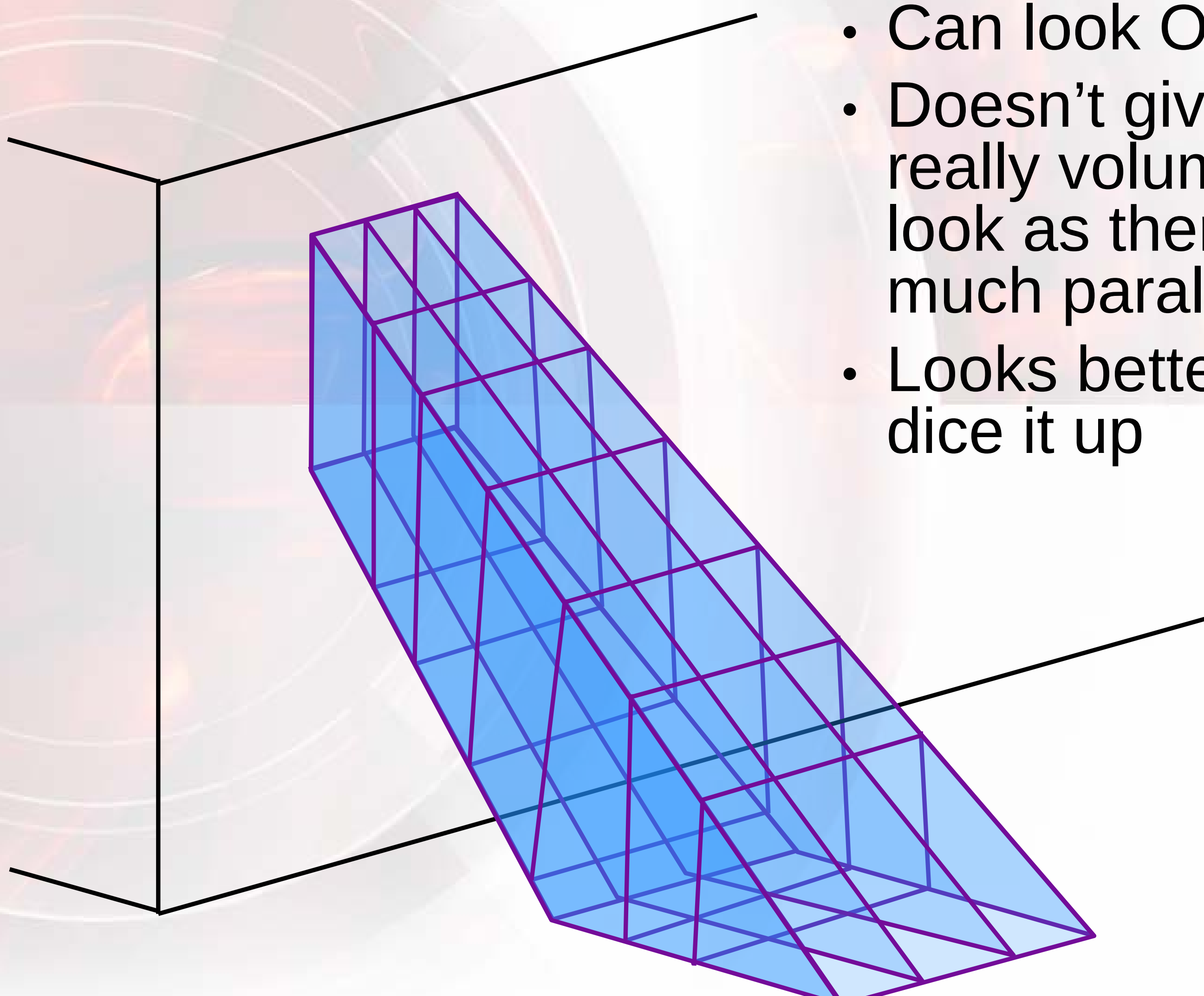
Splinter Cell

Splinter Cell

How are these drawn?

- Most games extrude some simple hull of a window or light source shape
  - Depending on viewing angle, the extrusion can sometimes end up looking obvious
  - Also hard to give a really volumetric feel or vary the color
- Particle systems can also sometimes give an acceptable look
- Often difficult to get decent shadowing with either approach

Hull Extrusion



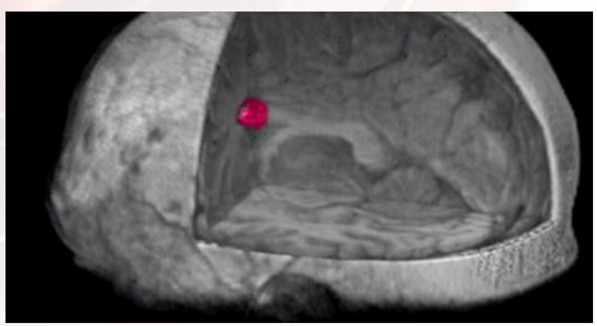
- Can look OK if faint
- Doesn't give a really volumetric look as there isn't much parallax
- Looks better if you dice it up

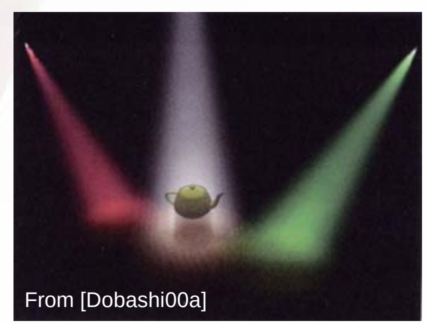
Hull Extrusion

- Each pixel of the light shafts gets light scattered from the near and far sides of the shaft
- There are some techniques which compute distance through the shaft/shape and compute an integral of scattered light
  - Radomír Mech, “Hardware-accelerated Real-time Rendering of Gaseous Phenomena,” Journal of Graphics Tools, 6(3):1-16, 2001
  - Greg James “Rendering Objects as Thick Volumes” in *ShaderX*^{2} and in GDC Direct3D tutorial last year

Volume Visualization Approach

- Here, we'll discuss an approach based on slice based volume rendering
- This technique is commonly used in volume visualization for medical applications
- Dobashi and Nishita have applied this approach to rendering of shafts of light
- Here, we'll present our implementation, with some extensions to take advantage of shader hardware and to address aliasing issues





From [Dobashi00a]

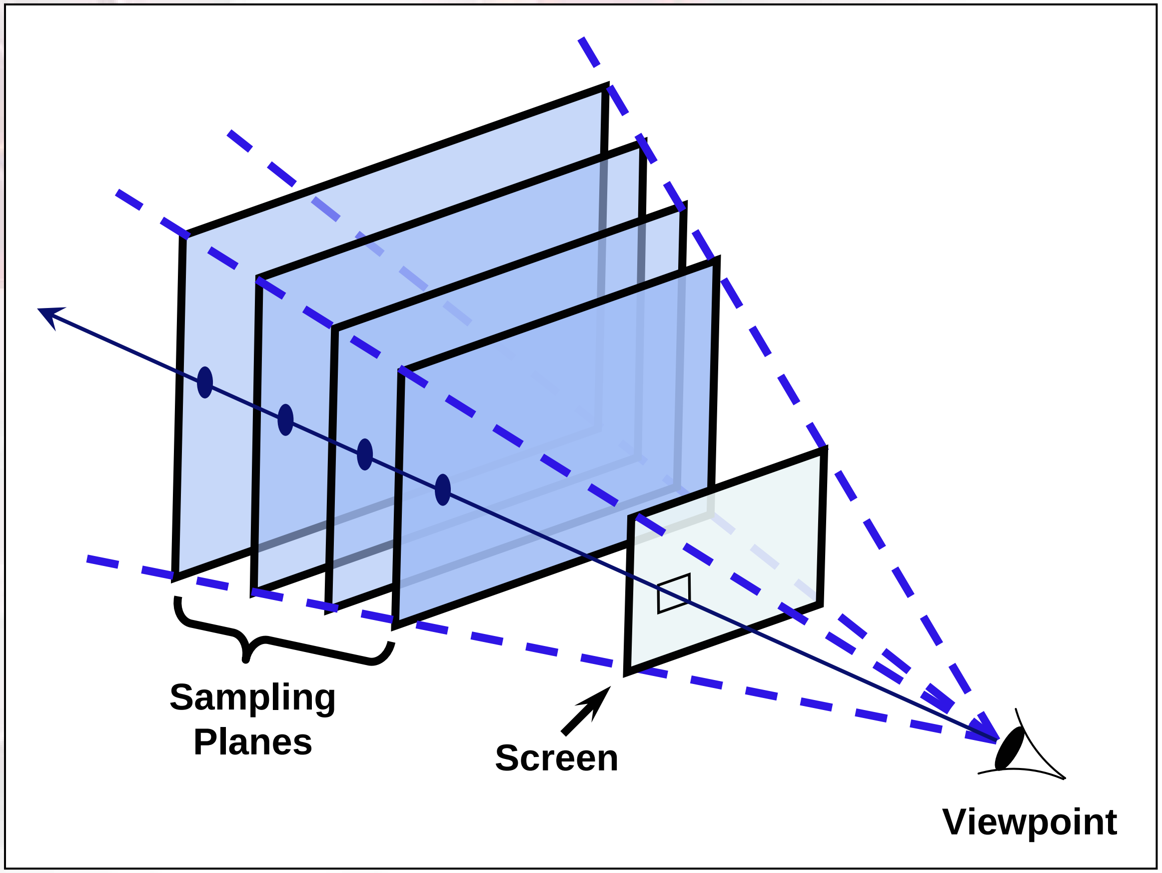
Dobashi and Nishita Volviz Results

From [Dobashi02]

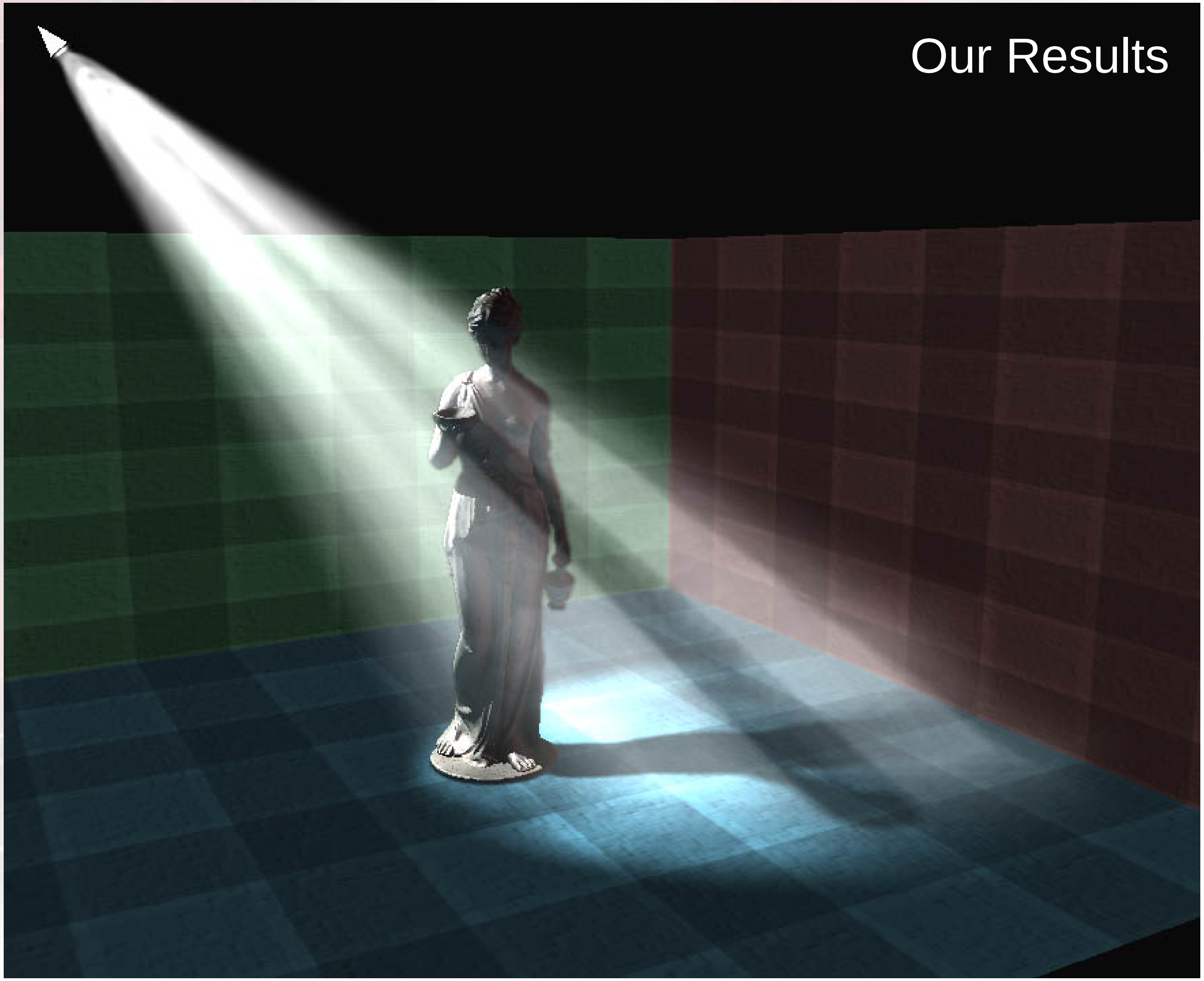
Dobashi and Nishita Volviz Results

Sampling the Light Shafts

- Technique developed in several papers by Dobashi and Nishita
- Shade sampling planes in light space
- Composite into frame buffer to approximate integral along view rays



Light Shaft Rendering



Sampling Plane Vertex Shading

- Automatic positioning using vertex shader
  - Static VB stores parametric position. Shader trilerps to fill view-space bounds of light frustum:

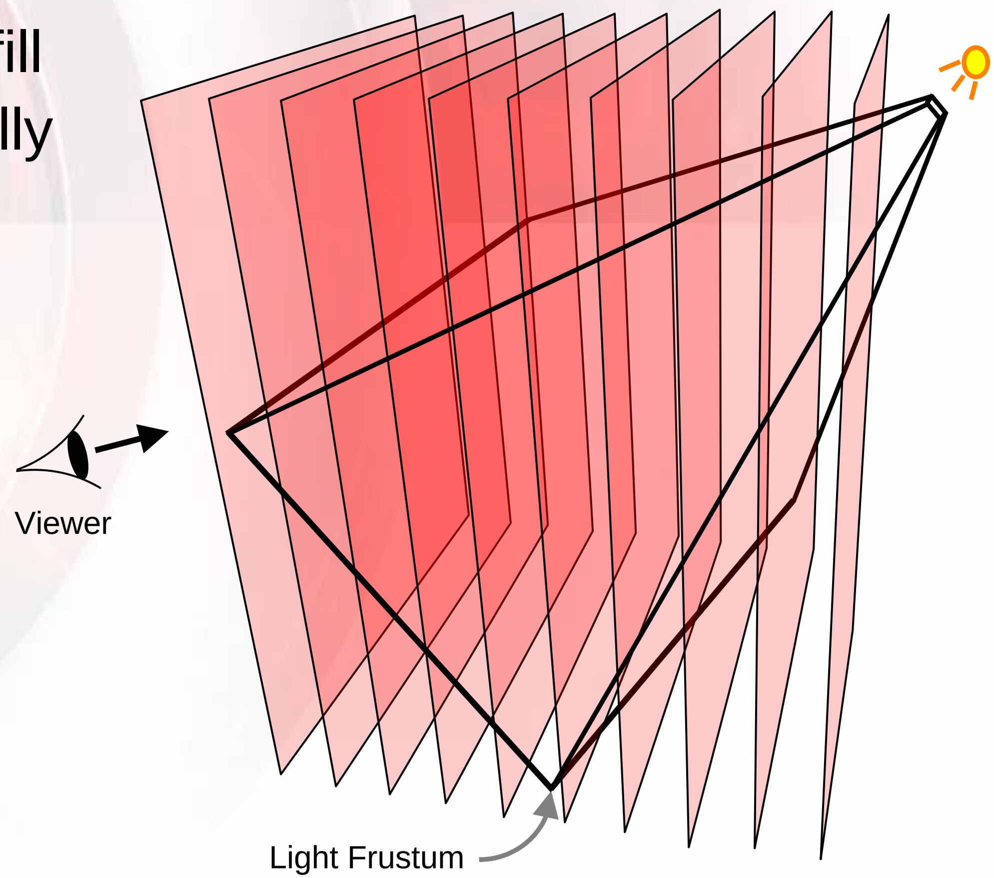
```
// Trilerp position within view-space-AABB of light's frustum
float4 pos = vMinBounds * vPosition + vMaxBounds * (1 - vPosition);
pos.w = 1.0f;

// Output clip-space position
Out.Pos = mul (matProj, pos);
```

- Clip to light frustum with user clip planes
- Only one quad per sampling plane
  - Dobashi and Nishita tessellate their sampling planes to evaluate low-frequency portion of scattering
  - Keep in mind that it's just a quad as you implement (i.e. interpolation position and compute $dist^2$ per-pixel)

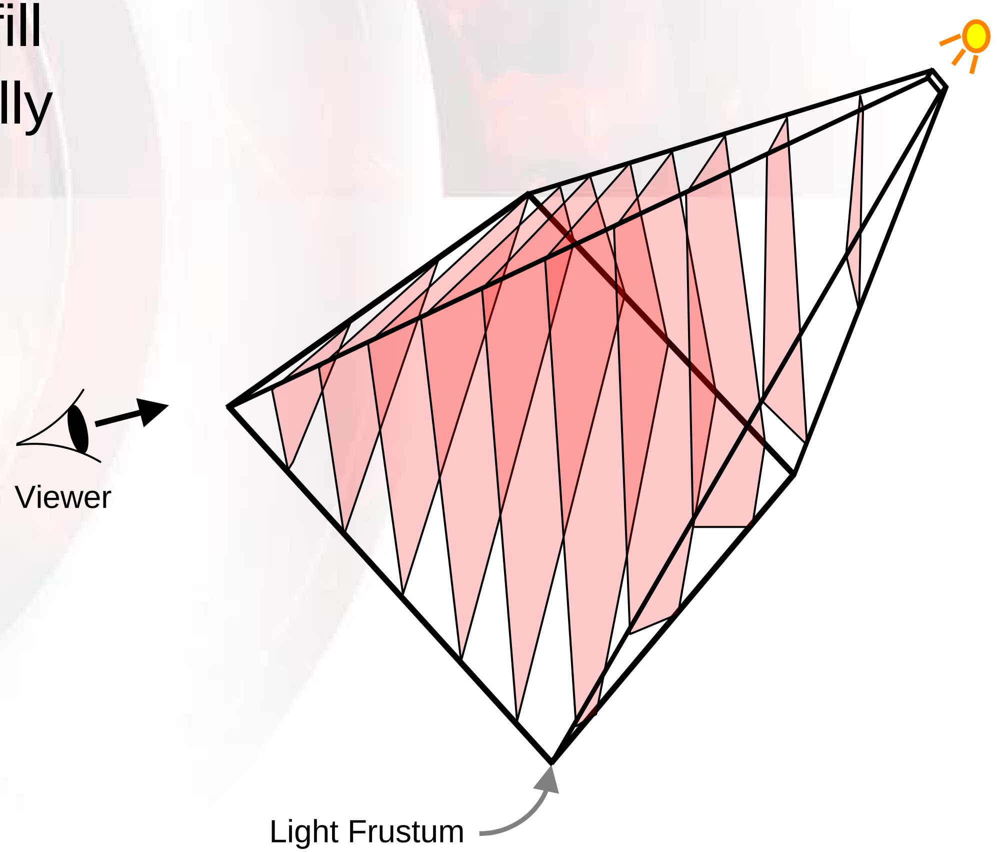
Clipping to Light Frustum

- Clipping planes to light frustum drastically reduces fill
- Worth doing manually on devices which don't support user clip planes
- Or just don't do this effect on such devices



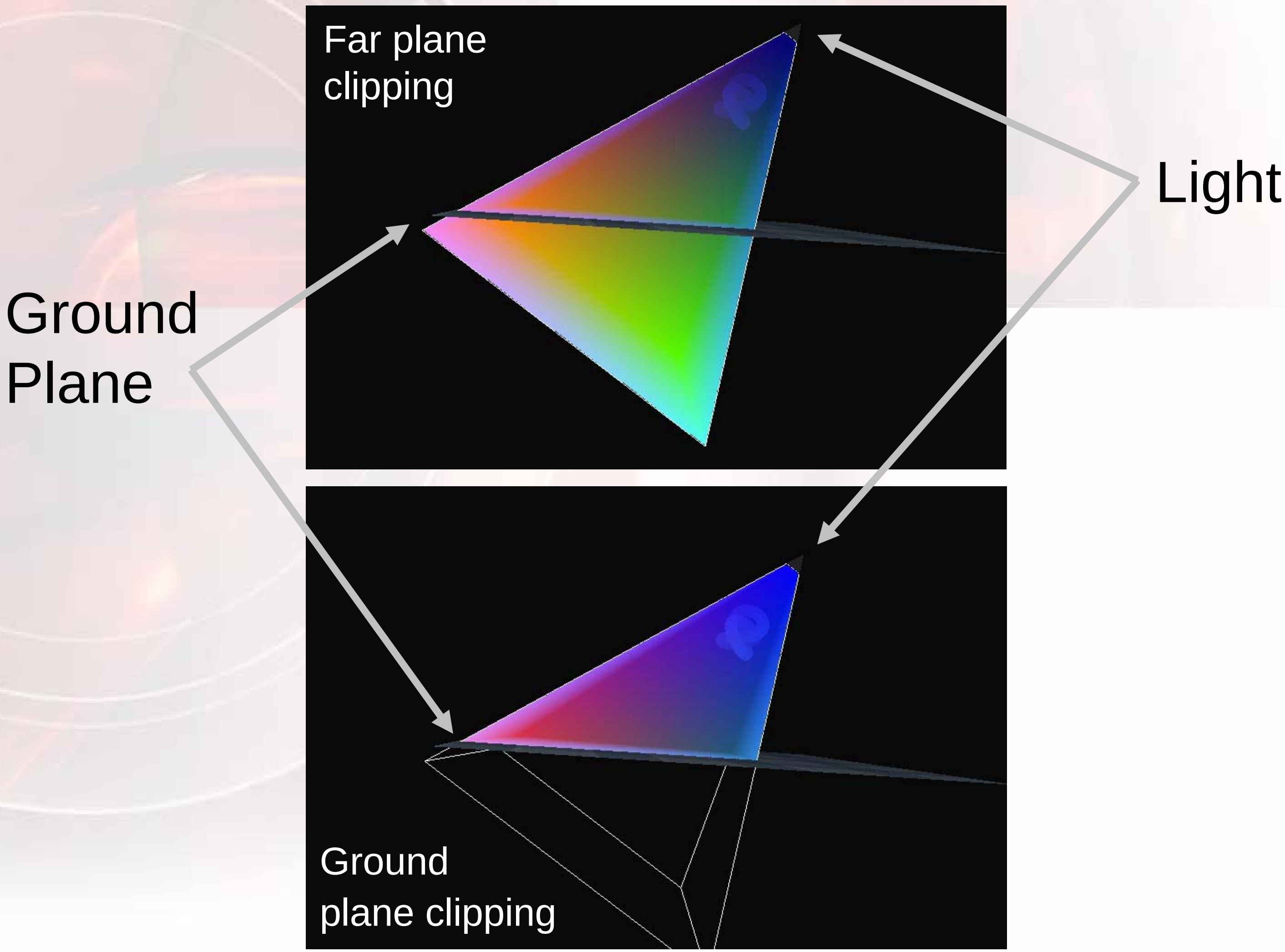
Clipping to Light Frustum

- Clipping planes to light frustum drastically reduces fill
- Worth doing manually on devices which don't support user clip planes
- Or just don't do this effect on such devices



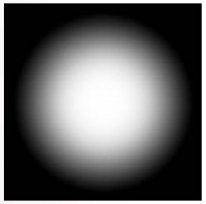
Clip at the ground plane

- Just an additional level of optimization



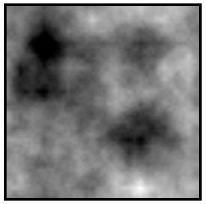
Sampling Plane Pixel Shading

- Distance attenuation (1/d$^{2)}$
- Four projective lookups from 3 2D textures projected onto sampling planes and surrounding scene:
  1. Cookie texture
  2. Shadow map
     - Can use different shadowing method on scene if desired
  3. Tiling noise map (sampled twice)
- Clipping to light frustum handles any back-projection on sampling planes. Need to keep track of this for scene geometry
- Color Mask effectively routes data to one of the four channels (more on this later)
- Alpha is set to sum of colors so that black pixels (no light) can be alpha tested away

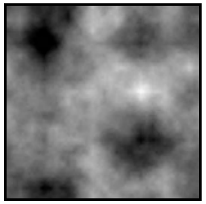


Cookie

Shadow



$Noise_1$



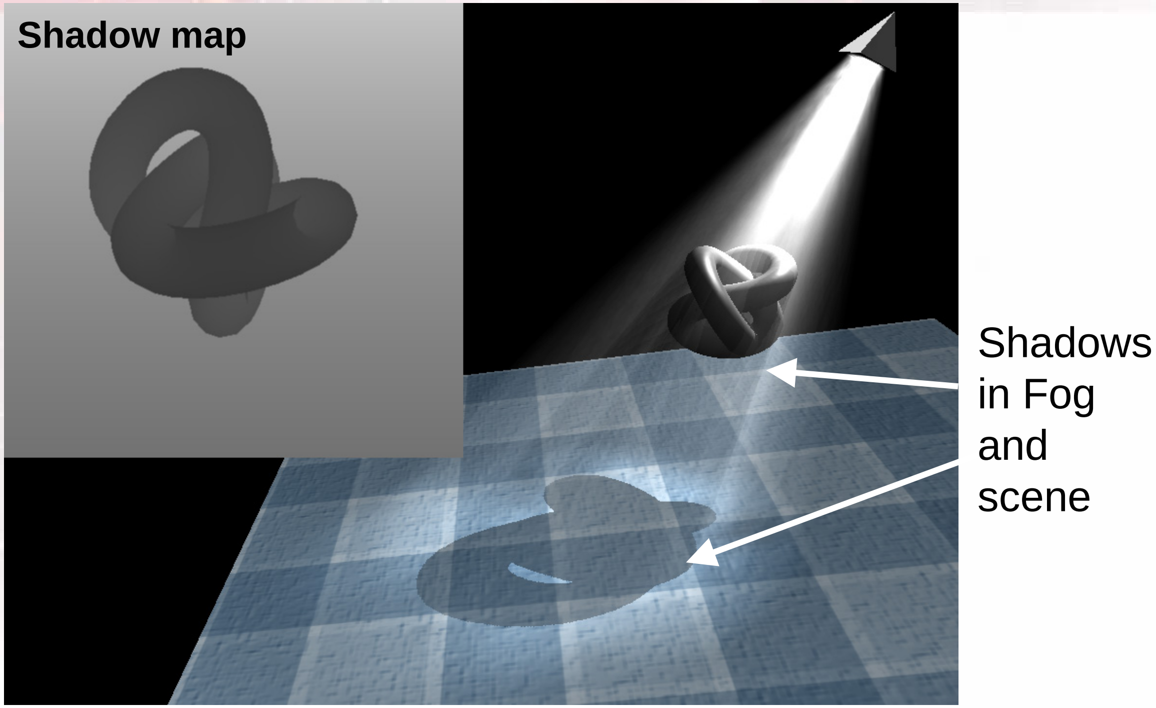
$Noise_2$

Non-uniform Particle Density

- Scroll a pair of scalar 2D noise maps in light’s projective space
- Composite together
- Modulate with other lighting terms
- Looks really nice, especially when the scene is otherwise static
- Can help hide aliasing

Shadow Mapping

- Project onto sampling planes
- Not necessarily on scene if you have another shadowing solution for scene
- Can be static (i.e. not regenerated every scene)



Sampling Plane Pixel Shader

```
float4 ps_main (float4 tcProj        : TEXCOORD0, float4 tcProjScroll1 : TEXCOORD1,
                float4 tcProjScroll2 : TEXCOORD2, float4 lsPos_depth : TEXCOORD3, float4 ChannelMask : COLOR0,

         uniform bool   bScrollingNoise,  //
         uniform bool   bShadowMapping,   // Uniform inputs to generate shader permutations
         uniform bool   bCookie) : COLOR  //
{
   float compositeNoise = 0.015f;
   float shadow = 1.0f;
   float4 cookie = {1.0f, 1.0f, 1.0f, 1.0f};

   float shadowMapDepth;
   float4 output;

   if (bCookie) {
      cookie = tex2Dproj(CookieSampler, tcProj);      // Sample the cookie
   }

   if (bScrollingNoise) {
      float4 noise1 = tex2Dproj(ScrollingNoiseSampler, tcProjScroll1);
      float4 noise2 = tex2Dproj(ScrollingNoiseSampler, tcProjScroll2);

      compositeNoise = noise1.r * noise2.g * 0.05f;
   }

   shadowMapDepth = tex2Dproj(ShadowMapSampler, tcProj);

   if (bShadowMapping) {
      if (lsPos_depth.w < shadowMapDepth)
         shadow = 1.0f; // The pixel is in light
      else
         shadow = 0.0f; // The pixel is occluded
   }

   float atten = 0.25f + 20000.0f / dot(lsPos_depth.xyz, lsPos_depth.xyz); // Compute attenuation 1/(s^2)

   float scale = 9.0f / fFractionOfMaxShells;

   output.rgb = compositeNoise * cookie.rgb * lightColor * scale *  atten  * shadow * ChannelMask;
   output.a = saturate(dot(output.rgb, float3(1.0f, 1.0f, 1.0f))); // Alpha is the sum of the color channels

   return output;
}
```

Undersampling and Quantization

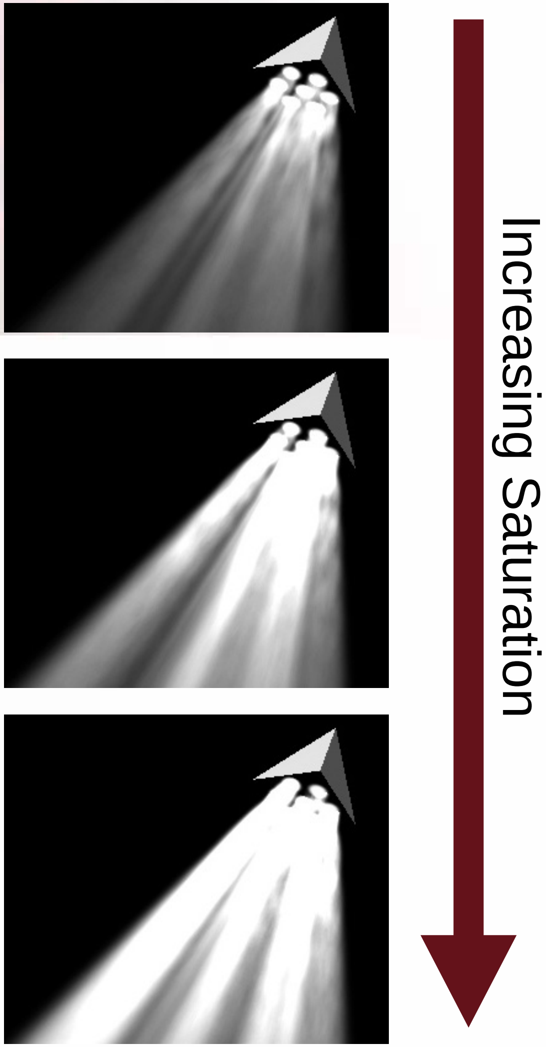
- **Undersampling**
  - This technique is a discrete approximation to integrals along rays through the volume
  - If you undersample, and the cookie texture has high spatial frequencies, you'll get aliasing. Even worse given the relatively hard-edged (high spatial frequency) shadows from the shadow map
  - Hence, you want **many sampling planes** in order to capture the high frequency content of the cookie and shadow maps
- **Quantization**
  - The discrete approximations to the integrals along rays through the volume are computed by additive blending with the frame buffer.
  - Current hardware can only blend to 8 bit per channel surfaces
  - Hence, you want **few sampling planes** so that each addition has at least a few bits of precision in the value added to the frame buffer
- **Conflicting goals!**

Dobashi Volviz Experiments

Comparison of images under different numbers of virtual planes

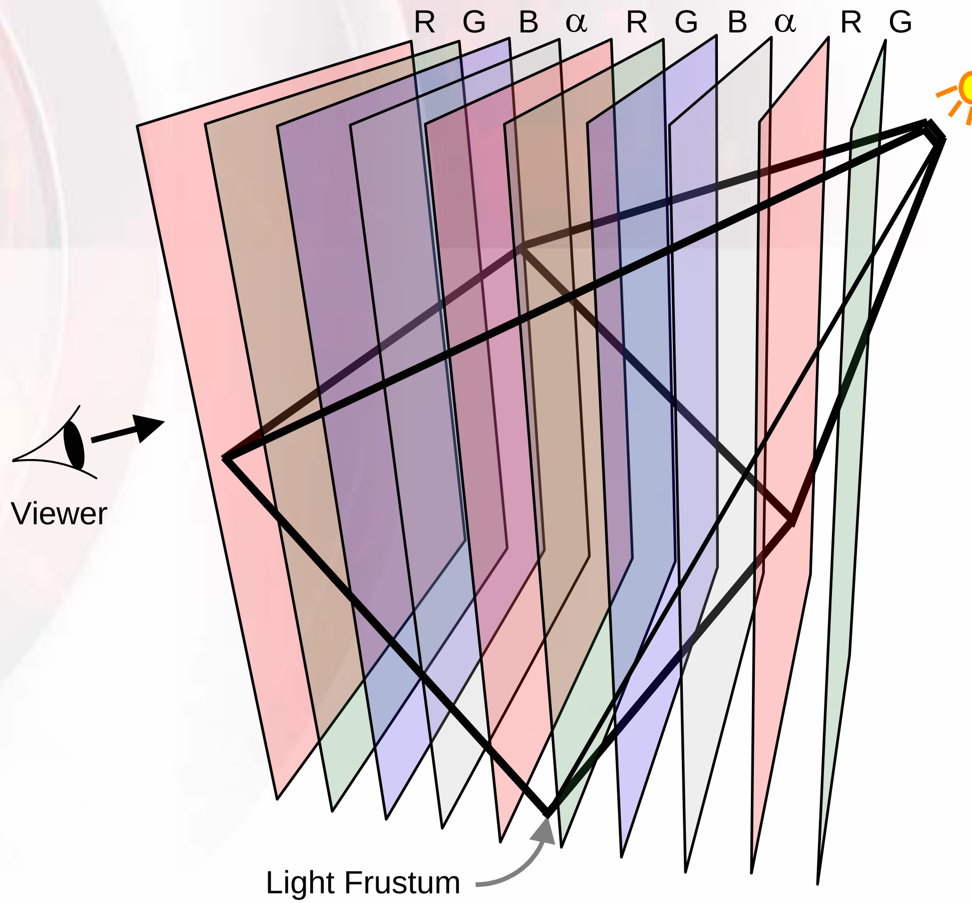
Hacking at the aliasing problem

- Aliasing tends to be more visible near the light source
- Try to smooth out the high frequency cookie details near the light source
  - Over-blur and brighten the small mip-levels of the cookie? Seems to help some.
  - Tune the attenuation term to cause lots of saturation near the light source?



Increasing Destination Precision

- Draw every fourth plane into a different channel of an offscreen RGBA texture



Increasing Destination Precision

- Draw every fourth plane into a different channel of an offscreen RGBA texture
- Clip to light frustum
- When subsequently compositing with back buffer, combine these four channels

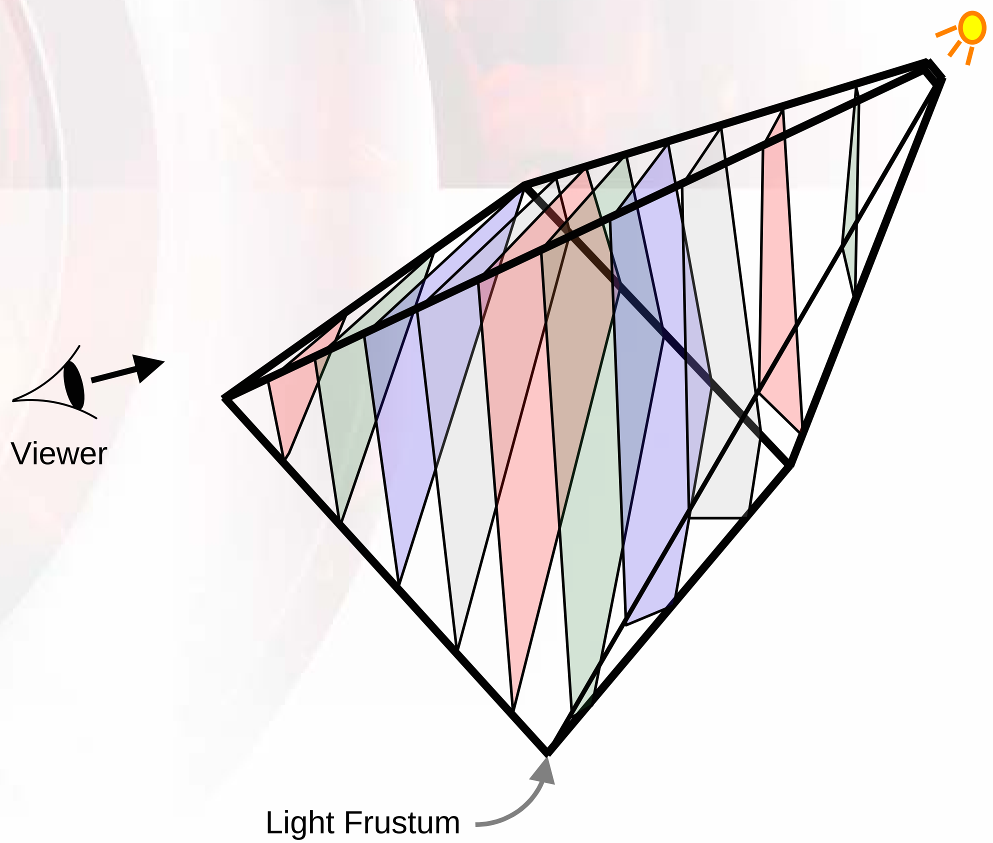


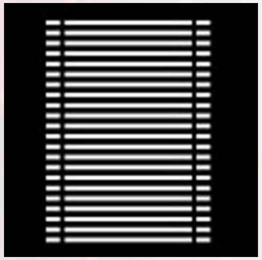
Image Space Glow / Blur

- Clearly, light scattered from particles in the air should undergo blurring just like any other light that reaches our eye
- Currently doing this a little bit now
- Helps hide aliasing due to undersampling

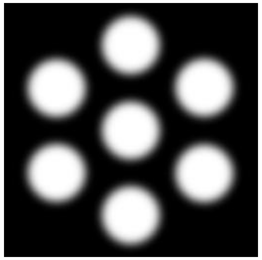
What works well?

- Positioning / Geometry
  - Avoid large depth extent in view space if you can
  - Reduce the volume of light frustum
    - Keep light FOV low (flashlights in E.T.)
- Low spatial frequency cookie
- If you use one buffer in order to get a color cookie, vary the cookie/gobo color
  - Tends to saturate less, since a given pixel is likely to have different channels hit as the sampling planes are drawn

Bad:

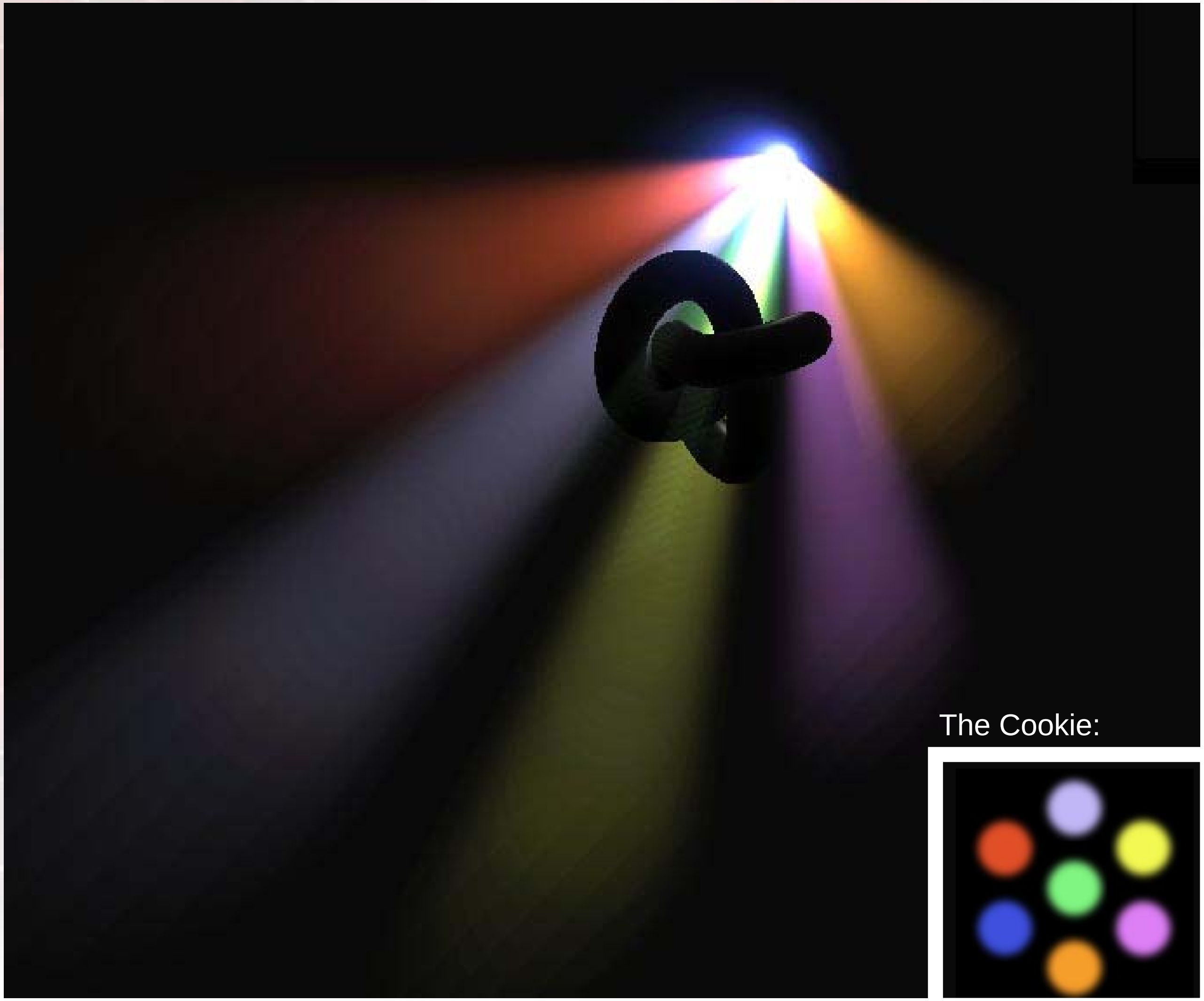


Good:



Great:

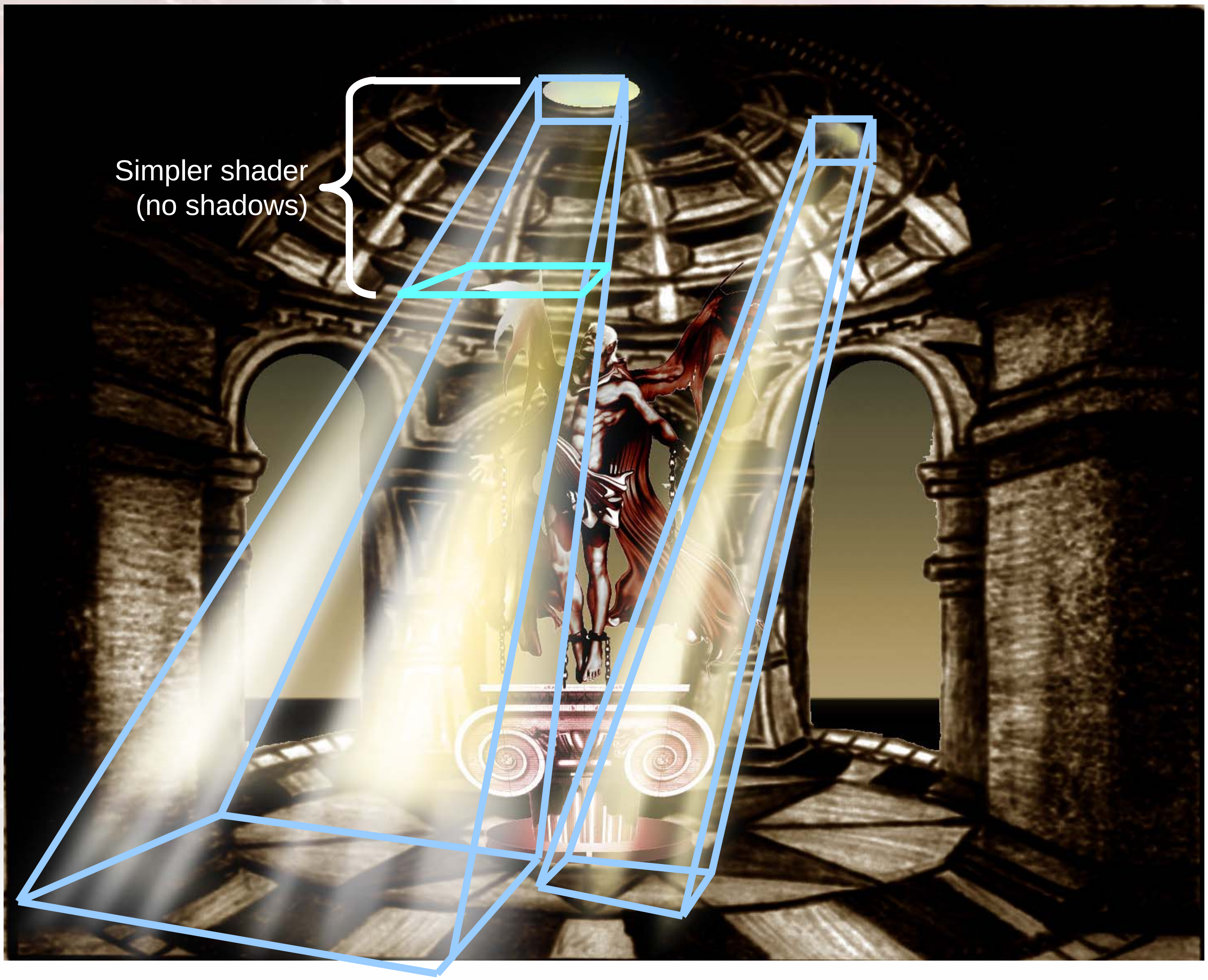
With a Colored Cookie



Optimization: Minimize the fill!

- Minimize the number of sampling planes
  - Scale based upon depth extent of light frustum / clipping volume
  - Currently doing this based on depth and capping at a max of 100 sampling planes
- Minimize the number of pixels shaded
  - Aggressive clipping and partitioning done now
  - Potential future optimization with zequal test as done in [Krüger03]
    - Sort light frusta and draw them last with z-writes on
    - For every sampling plane
      - Draw with a simple shader (color writes off) to determine coverage
      - Render it again with complex shader (color writes on) and zequal test
- Minimize the cost of the pixel shader
  - Scrolling noise hurts
  - Shadow mapping hurts
  - Potentially clip the light volume above the shadowing objects and draw that part of frustum with simpler shader?
- Minimize the number of pixels written out (α test)
  - May hurt more than help if your scene has a lot of occluders, since this turns off early-z test

Partitioning the light shafts



Pros and Cons of Volviz Approach

- Pro
  - Inherently “soft”
  - Easy to fake non-uniform density of particles
  - Easy to color it for stained-glass or other effects
- Con
  - Fillrate-heavy
  - Cost of shadow map rendering pass (if dynamic)
  - Possible shadow map filtering
  - High fillrate required
  - Could undersample volume
    - Especially due to hard occlusion info from shadow map
  - Quantization errors due to accumulation in 8-bit per channel render target
  - Did I mention that it requires a lot of fill?

Future Directions

- Better scattering models
  - Plenty of literature on this
- Better shadow map filtering
  - Percentage-closer filtering
- Virtual sub-planes
  - Use pixel shader to evaluate multiple samples rather than just one
  - Dobashi does something similar but more costly with "sub-planes"
  - Dobashi put more of these near the viewer than far from the viewer
- Interleaved Sampling
  - Vary positions of planes and/or virtual subplanes between neighboring pixels in screen-space. See [Keller01]

References

- **[Dobashi00_b]** Yoshinori Dobashi, Tsuyoshi Yamamoto , Tomoyuki Nishita, "Interactive Rendering Method for Displaying Shafts of Light," Proc. Pacific Graphics 2000, pp. 31-37 (2000).
- **[Dobashi00_c]** Yoshinori Dobashi, T. Okita, Tomoyuki Nishita, "Interactive Rendering of Shafts of Light Using a Hardware-accelerated Volume Rendering Technique," Proc. SIGGRAPH 2000 Technical Sketches, pp. 219, New Orleans (USA), July 2000.
- **[Nishita01]** Tomoyuki Nishita and Yoshinori Dobashi, “Modeling and Rendering of Various Natural Phenomena Consisting of Particles,” Proc. Computer Graphics International 2001
- **[Keller01]** Alexander Keller and Wolfgang Heidrich, “Interleaved Sampling” Eurographics Workshop on Rendering Techniques 2001
- **[Dobashi02]** Yoshinori Dobashi, Tsuyoshi Yamamoto and Tomoyuki Nishita, “Interactive Rendering of Atmospheric Scattering Effects Using Graphics Hardware,” Graphics Hardware 2002.
- **[Krüger03]** Jens Krüger and Rüdiger Westermann, “Acceleration Techniques for GPU-based Volume Rendering” IEEE Visualization 2003.