Volumetric Effects

Bo Yang and Taylor Infuso

What are Volumetric Effects?

Volumetric effects are effects that require rendering volume for authenticity.

Some examples would be:

- Fog
- God Ray
- Halos
- Light Shafts
- Particle Systems
- ...and more


Iniciar batalha


Flask
XBOX 360
Flask
PS4 Pro


Bilateral Filters ON
128 steps

Halos

Halos are spherical lighting effects where the highest concentration is in the center of the sphere with lower amounts of light further from the center.

How Do We Generate Halos?

Halos require us to measure the level of brightness of every pixel. How do we do this? RAYCASTS!

Basic Calculations for Light Density at a pixel:

Light Density = 1 - (distance from halo center)^2 / (radius of halo)^2

(this is a simplified version of the function)

Mathematical Concepts

Generating a halo requires vector calculus. We need to determine what brightness each pixel should be based off of the ray entering the halo. To do this, we:

1. Determine the distance between our camera and the point of the bounding geometry our ray points to. We need a density function which is determined by where this ray intersects the halo.
2. We find a pixel on the ray and use z-buffering to determine its depth.
3. We need to integrate. That means clamping an interval between the entrance and exit point of the ray through the halo. We then determine the light density and normalize the integral (equation will be shown on the next slide).

```
uniform TextureRect    structureBuffer;
uniform float3         cameraPosition, cameraView;
uniform float          R2, recipR2, recip3R2, normalizer;

float CalculateHaloBrightness(float3 pobject, float2 pixelCoord)
{
    float3 vdir = cameraPosition - pobject;
    float v2 = dot(vdir, vdir);
    float p2 = dot(pobject, pobject);
    float pv = -dot(pobject, vdir);
    float m = sqrt(max(pv * pv - v2 * (p2 - R2), 0.0));

    // Read z0 from the structure buffer.
    float2 depth = texture(structureBuffer, pixelCoord).zw;
    float t0 = 1.0 + (depth.x + depth.y) / dot(cameraView, vdir);

    // Calculate clamped limits of integration.
    float t1 = clamp((pv - m) / v2, t0, 1.0);
    float t2 = clamp((pv + m) / v2, t0, 1.0);
    float u1 = t1 * t1;
    float u2 = t2 * t2;

    // Evaluate density integral, normalize, and square.
    float B = ((1.0 - p2 * recipR2) * (t2 - t1) + pv * recipR2 * (u2 - u1)
              - v2 * recip3R2 * (t2 * u2 - t1 * u1)) * normalizer;
    return (B * B * v2);
}
```

Light Shaft

A light shaft is described as a lightsource that extends light over a long distance generally used in a scene with low visibility/darkness to emphasize a light source.

The Process of Creating Light Shafts

1. Generate a halo. Halos use a very similar setup to light shafts.

2. We generate the brightness with a function that looks like this: light_density = (location of source of light - pixel in light shaft) / height of light shaft

   (again, this is simplified)

Mathematical Concepts

To generate shaft, it's rendered by the pixel shader after the intersection parameter t1, and t2 have been calculated. To do it

1. Using inside of a shaft, by linear density function, we have a formula $\rho(z)=\sigma z+\rho_{0.}$
2. As with halos, the brightness of a shaft by integrating this density function along the ray between two parameters where it intersects the shaft's surface.
3. Find the minimum or maximum parameter value of t. This value represent the range limit.

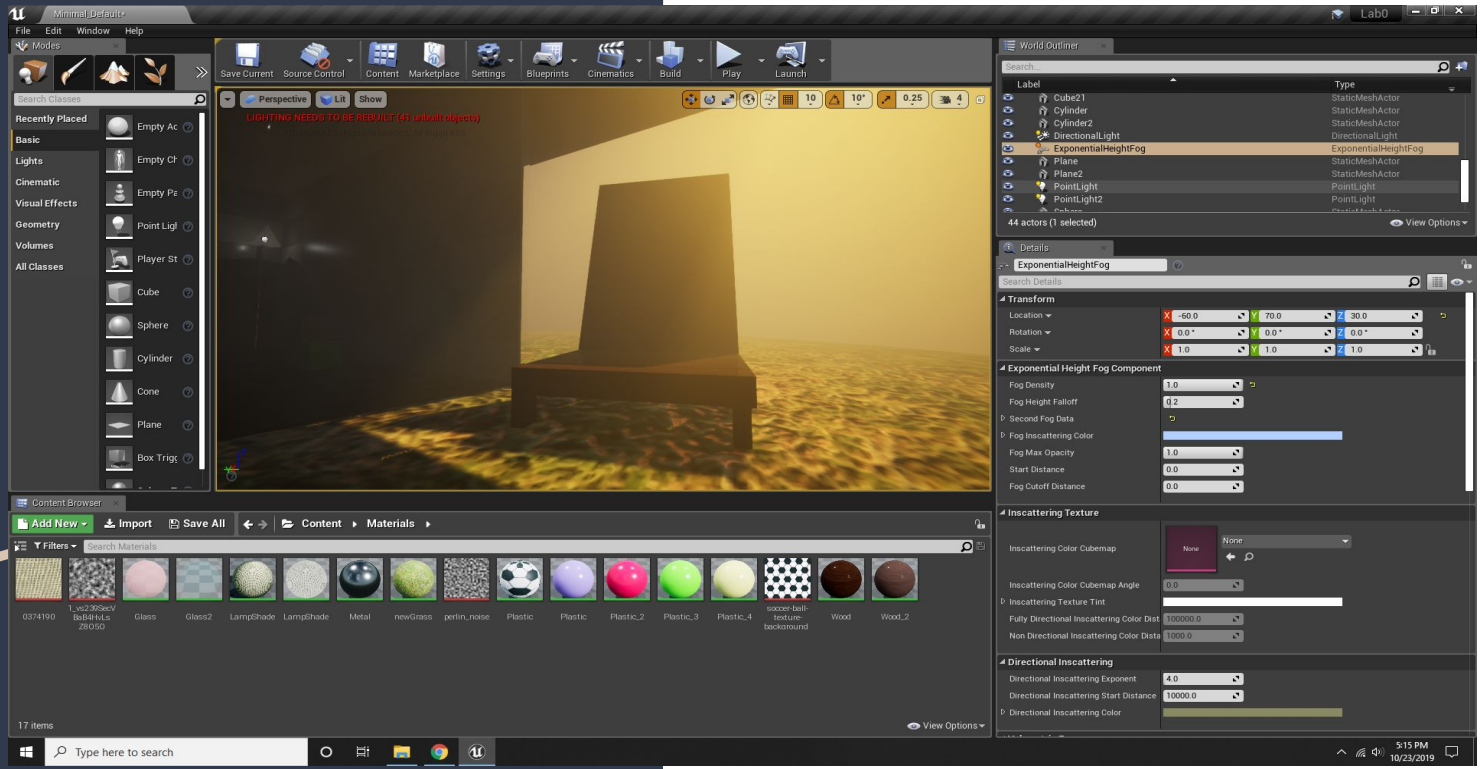
```
uniform TextureRect    structureBuffer;
uniform float3         cameraView;
uniform float          shaftSigma, shaftRho0, shaftTau, normalizer;

float CalculateShaftBrightness(float pz, float3 vdir, float2 pixelCoord,
                               float t1, float t2)
{
   // Read z0 from the structure buffer, calculate t0, and clamp to [t0,1].
   float2 depth = texture(structureBuffer, pixelCoord).zw;
   float t0 = 1.0 + (depth.x + depth.y) / dot(cameraView, vdir);
   t1 = clamp(t1, t0, 1.0); t2 = clamp(t2, t0, 1.0);

   // Limit to range where density is not negative.
   float tlim = (shaftTau - pz) / vdir.z;
   if (vdir.z * shaftSigma < 0.0) {t1 = min(t1, tlim); t2 = min(t2, tlim);}
   else {t1 = max(t1, tlim); t2 = max(t2, tlim);}

   // Evaluate density integral, normalize, and square.
   float B = (shaftSigma * (pz + vdir.z * ((t1 + t2) * 0.5)) + shaftRho0)
           * (t2 - t1) * normalizer;
   return (B * B * dot(vdir, vdir));
}
```

Atmosphere Buffer

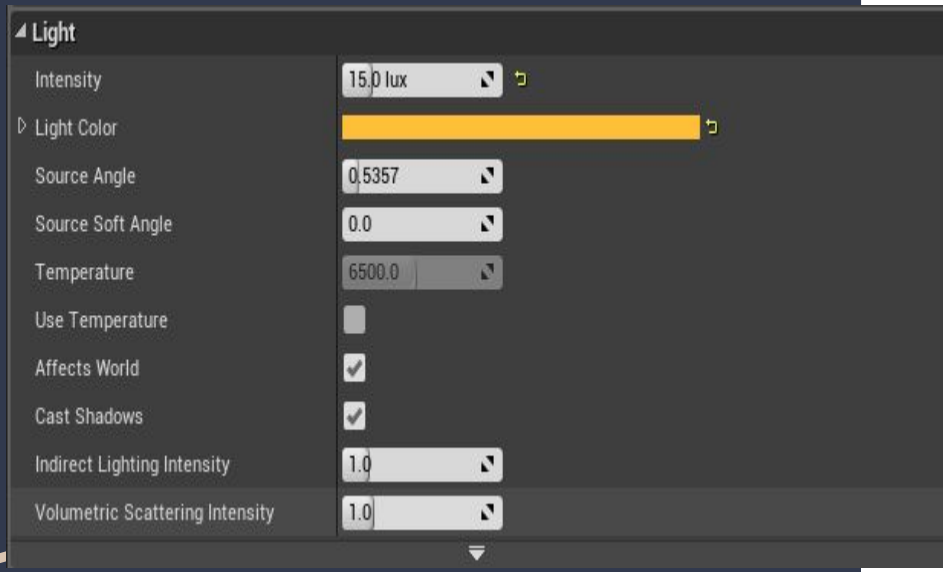
Atmosphere buffering is a term referring to the layer we use to store light contributions. These contributions come from pixels outside of shadow. For atmospheric shadowing, the basic idea is to generate a ray for each pixel's location on the projection plane and sample it at many points within some fixed range of distances from the camera.

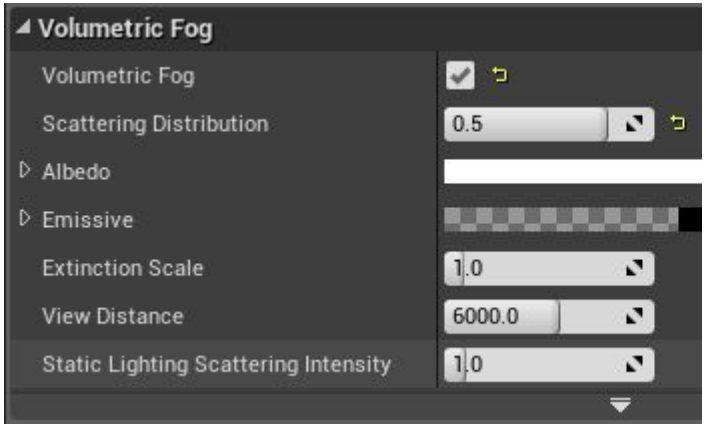
Taking a greater number of samples per ray produces better results, but doing so comes at the cost of performance. Since the ray render do not need to include sharp details, one easy step we can take to reduce the overall computational cost is to make the atmosphere buffer half the resolution of the main frame buffer so we render only one quarter the number of pixels.

How We Do This In Unreal?



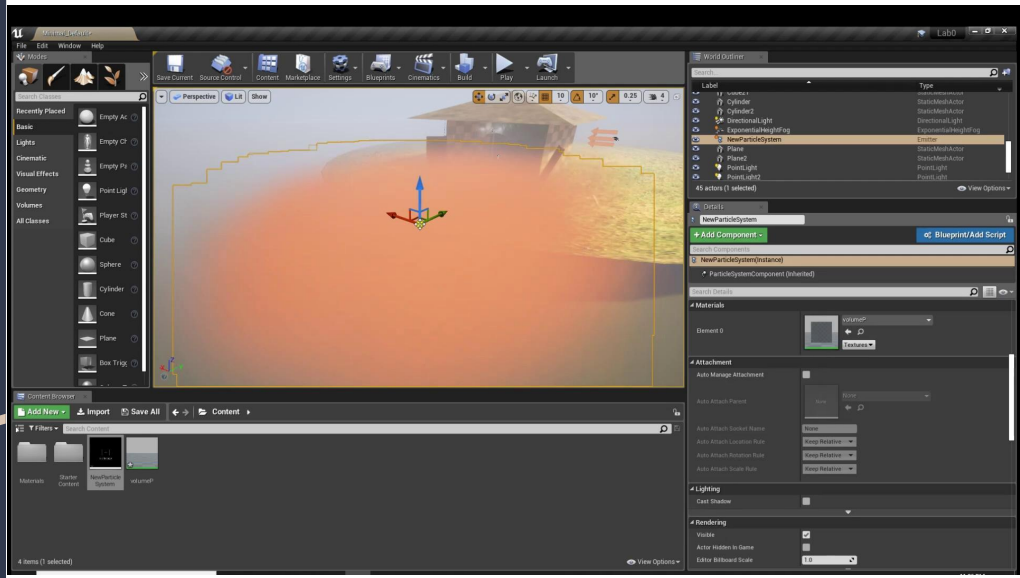
In-Editor Options

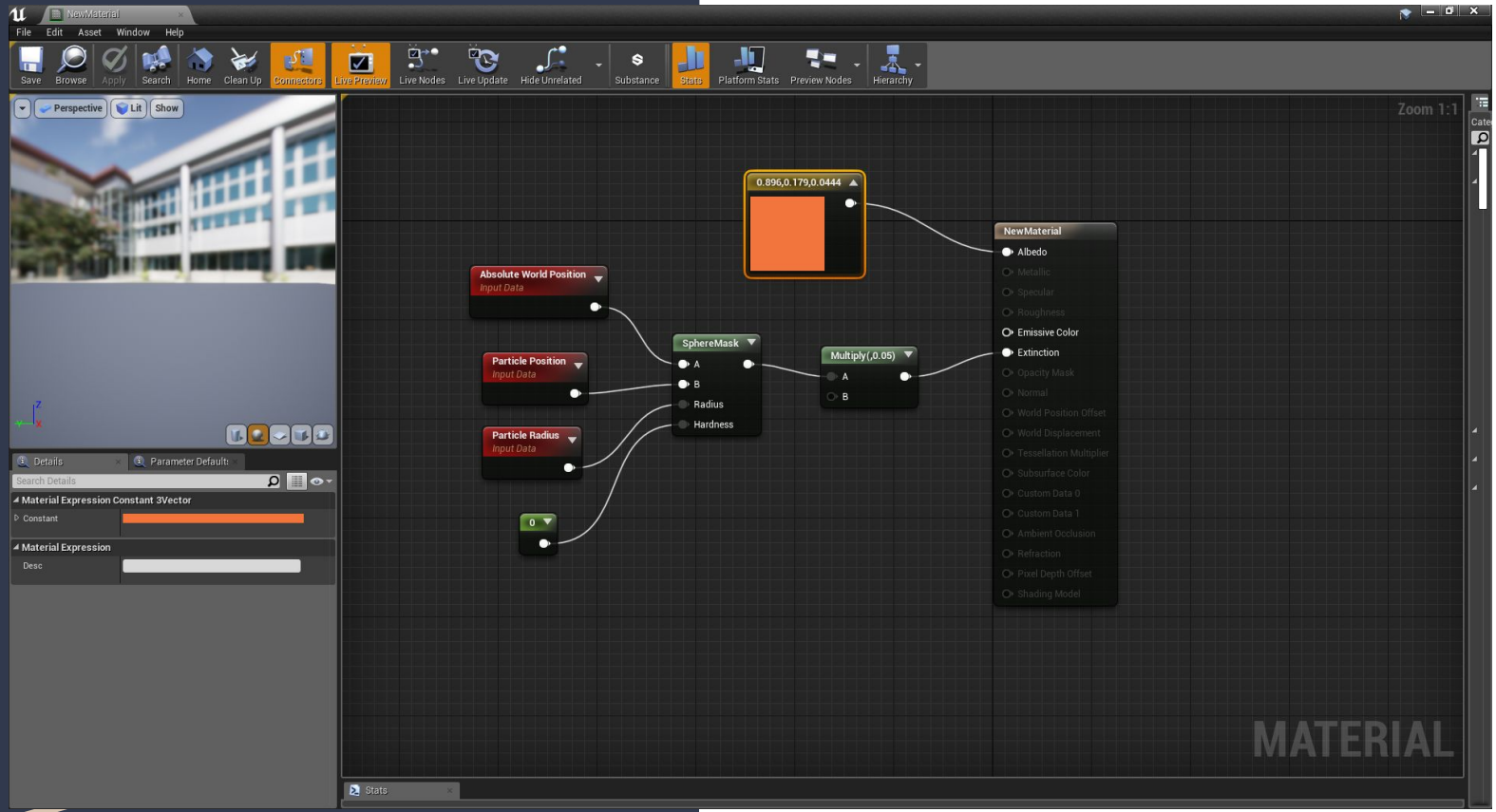


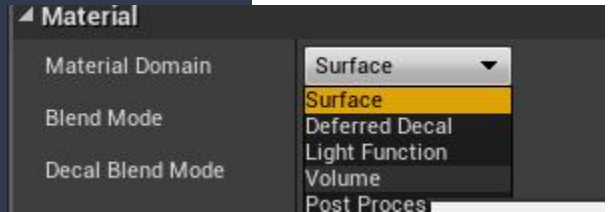


Unreal Engine Demo (inspired by GDC 2018 Unreal Engine Presentation)

Volumetric Particles (inspired by GDC 2018 Unreal Engine Presentation)




NewMaterial
File
Edit
Asset
Window
Help
Save
Browse
Apply
Search
Home
Clean Up
Connectors
Live Preview
Live Nodes
Live Update
Hide Unrelated
Substance
Stats
Platform Stats
Preview Nodes
Hierarchy
Perspective
Lit
Show
Zoom 1:1
0.896,0.179,0.0444
Absolute World Position
Input Data
Particle Position
Input Data
Particle Radius
Input Data
0
SphereMask
A
B
Radius
Hardness
Multiply(,0.05)
A
B
NewMaterial
Albedo
Metallic
Specular
Roughness
Emissive Color
Extinction
Opacity Mask
Normal
World Position Offset
World Displacement
Tessellation Multiplier
Subsurface Color
Custom Data 0
Custom Data 1
Ambient Occlusion
Refraction
Pixel Depth Offset
Shading Model
MATERIAL
Details
Parameter Defaults
Search Details
Material Expression Constant 3Vector
Constant
Material Expression
Desc
Stats


Material
Material Domain
Surface
Blend Mode
Decal Blend Mode
Surface
Deferred Decal
Light Function
Volume
Post Proces

Information We Do Not Fully Understand

1. Actually designing a atmosphere buffer is incredibly complicated. Requires knowledge on multi light ray deflection combined with elements of randomization.
2. Determining how light rays should appear when the camera is outside of their range.

Works Cited

Akenine-Moller, Tomas. Haines, Eric. Hoffman, Naty. Pesce, Angelo. Iwanicki, Michal. Hillaire Sebastien. *Real-Time Rendering*. Boca Raton. CRC Press, 2018. Print.

Kostack Studio. “God Rays - Blender Volume Lighting.” *Youtube*, 15 October 2012. https://www.youtube.com/watch?v=p3GGgy0Ulcg

stoopdapoop. “Guerilla Games: 8 Sample Volumetric Lighting.” *Youtube*, 4 September 2014. https://www.youtube.com/watch?v=0MilN7jKK9c

Unreal Engine. “Volumetric Fog and Lighting in Unreal Engine 4 | GDC 2018 | Unreal Engine.” *Youtube*, 27 March 2018. https://www.youtube.com/watch?v=Xd7-rTzfmCo