Deferred Particle Shading

- Cooler Looking Smoke For Games

  (yeehaa)

Typical Game Particles So Far

- Bunch of quads rendered on top of each other
- The effect comes from this (semi)-random representation of a volume effect...
- This works ok in practice
  - Player can't tell each particle apart.
- Typically, we use 3 blend modes to do our particles:
  - Multiply (dark smoke),
  - Additive (fiery things),
  - Blending (grayish smoke)

Limitations

- All well and good, but smoke can be a very solid thing:

- Really a volume effect. We cannot treat each particle by itself.
- Hence, deferred shading:
  - We use one (or more) offscreen buffers to accumulate the particles (*accumulation pass)*
  - Finally we render a 'fullscreen' quad onto the main buffer. *(Transfer pass)*
- *The goal is to get the nice interactions between the particles, but letting us*

Deferred - The Basics:

- We're looking at smoke - the principles outlined here can be used for lots of other effects.
- We typically care about surface normal & 'amount' of smoke.
- RGB = Normal, A = amount
  - Alphablend the normalmaps of particles into an offscreen buffer.
- Do transfer pass(es) where we calculate lighting on the accumulated particles.
  - Calculate a bumpmap style diffuse lighting
  - Add backlit inscatter - faking extinction
  - Add specular inscatter for light sources behind the smoke

Accumulating The Particles

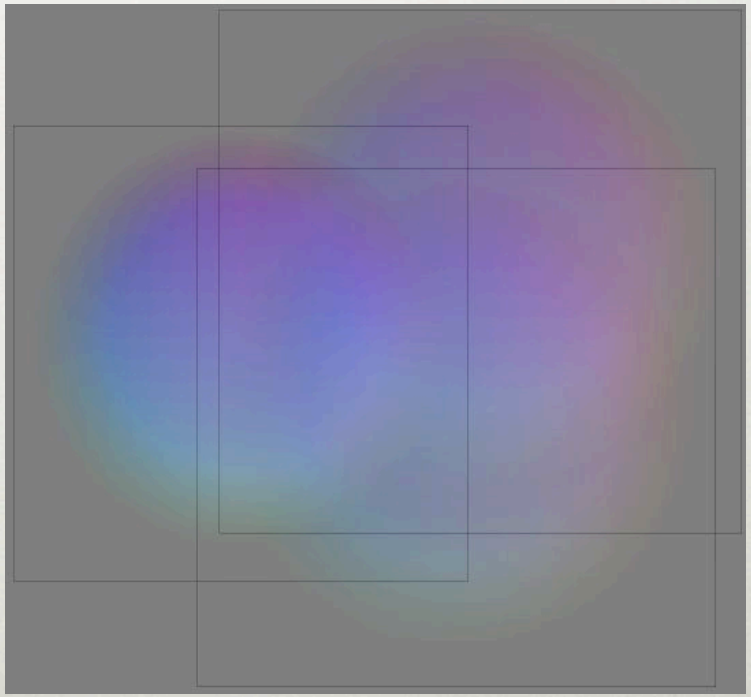
- Normal alphablending:

```
glBlend (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
    RGB = Src.rgb * Src.a + Dst.rgb * (1 - Src.a)
      A =   Src.a * Src.a + Dst.rgb * (1 - Src.a)
```

The particle

RGB

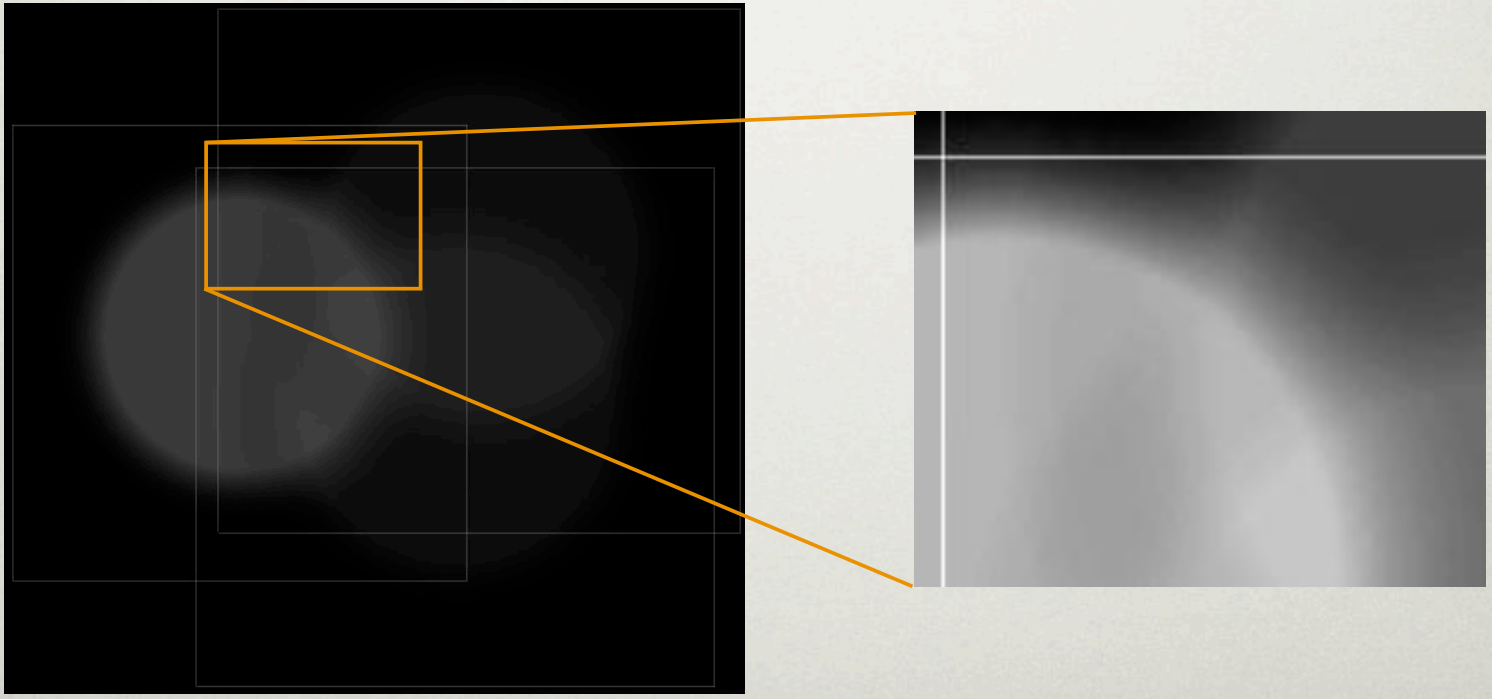
A



Accumulating The Particles

- Normal alphablending:

```
glBlend (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
    RGB = Src.rgb * Src.a + Dst.rgb * (1 - Src.a)
      A =   Src.a * Src.a + Dst.rgb * (1 - Src.a)
```



- But we want to ADD the alpha, not blend it

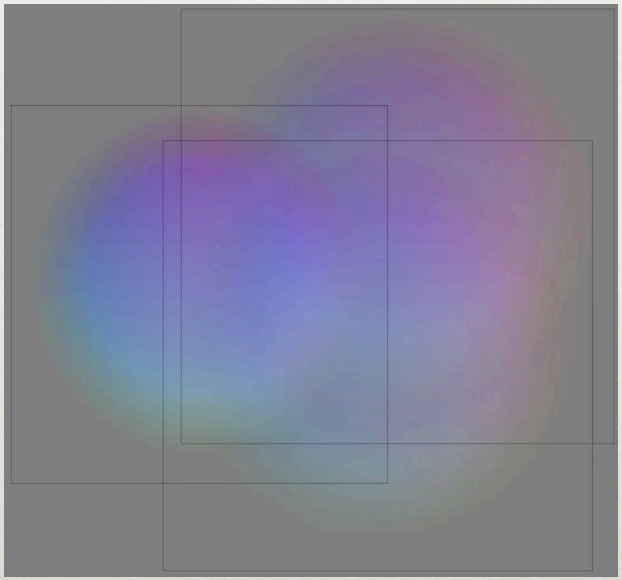
Accumulating The Particles

- Premultiply the particles RGB in a shader, then do

```
glBlend (GL_ONE, GL_ONE_MINUS_SRC_ALPHA)
```

- This gives

```
RGB = Src.rgb * Src.a + Dst.rgb * (1 - Src.a)
  A =   Src.a * ONE   + Dst.rgb * (1 - Src.a)
```



- RGB is still blended....

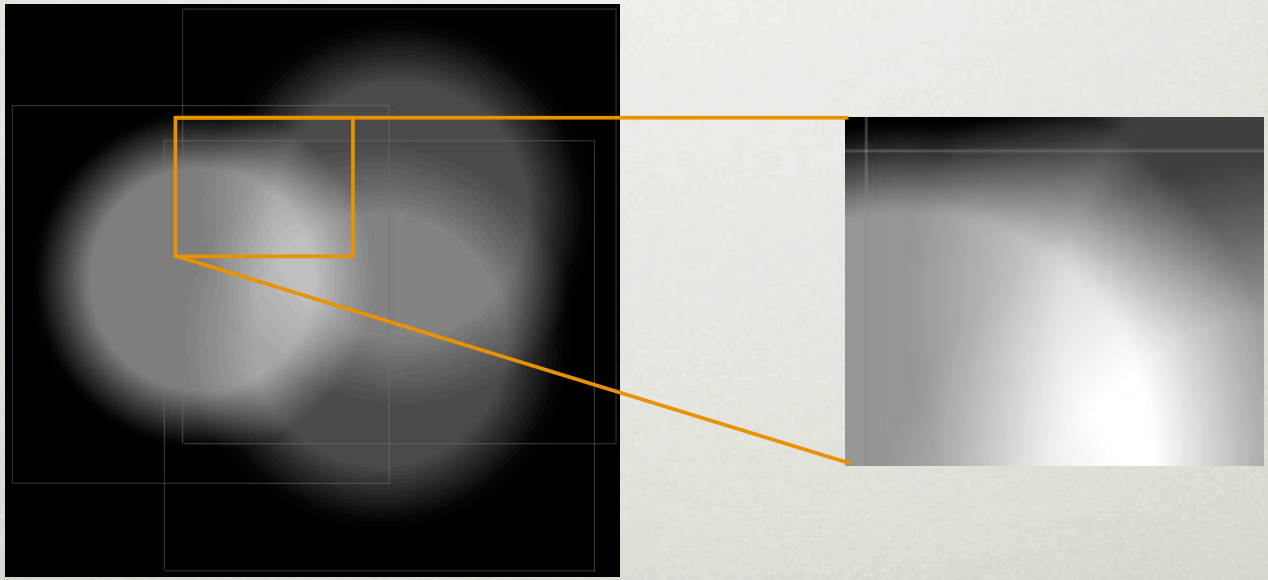
Accumulating The Particles

- Premultiply the particles RGB in a shader, then do

```
glBlend (GL_ONE, GL_ONE_MINUS_SRC_ALPHA)
```

- This gives

```
RGB = Src.rgb * Src.a + Dst.rgb * (1 - Src.a)
  A =   Src.a * ONE   + Dst.rgb * (1 - Src.a)
```

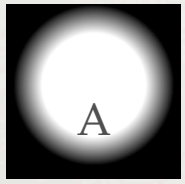


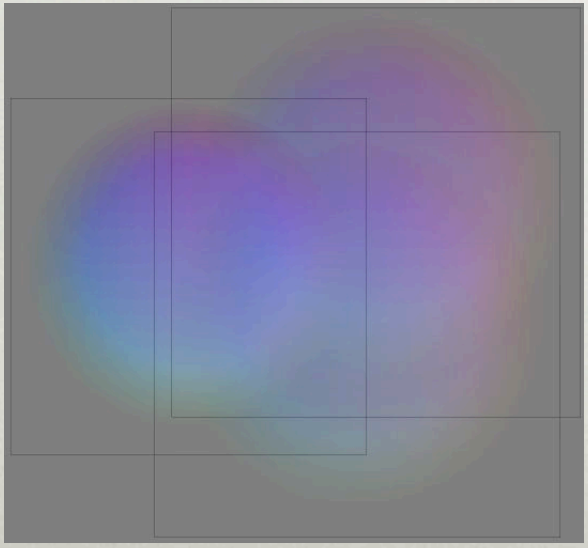
- Alpha is now oldskool AddSmooth

Accumulating The Particles

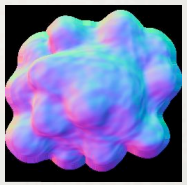
Accumulation Result

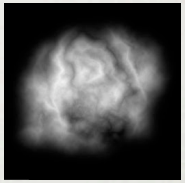
Silly Demonstration Particles

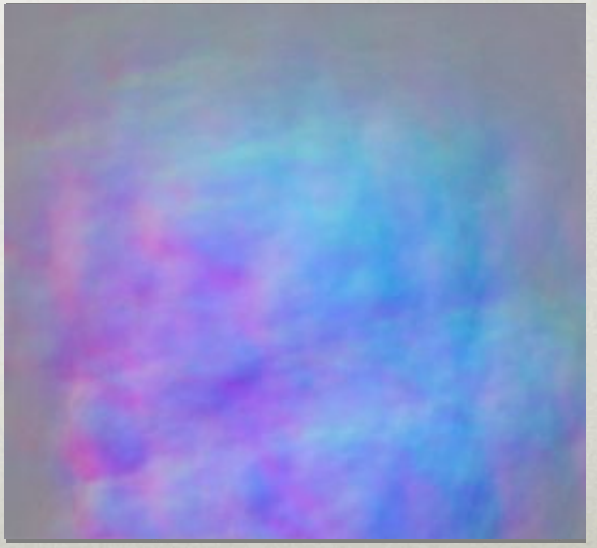




Proper Ingame Particles



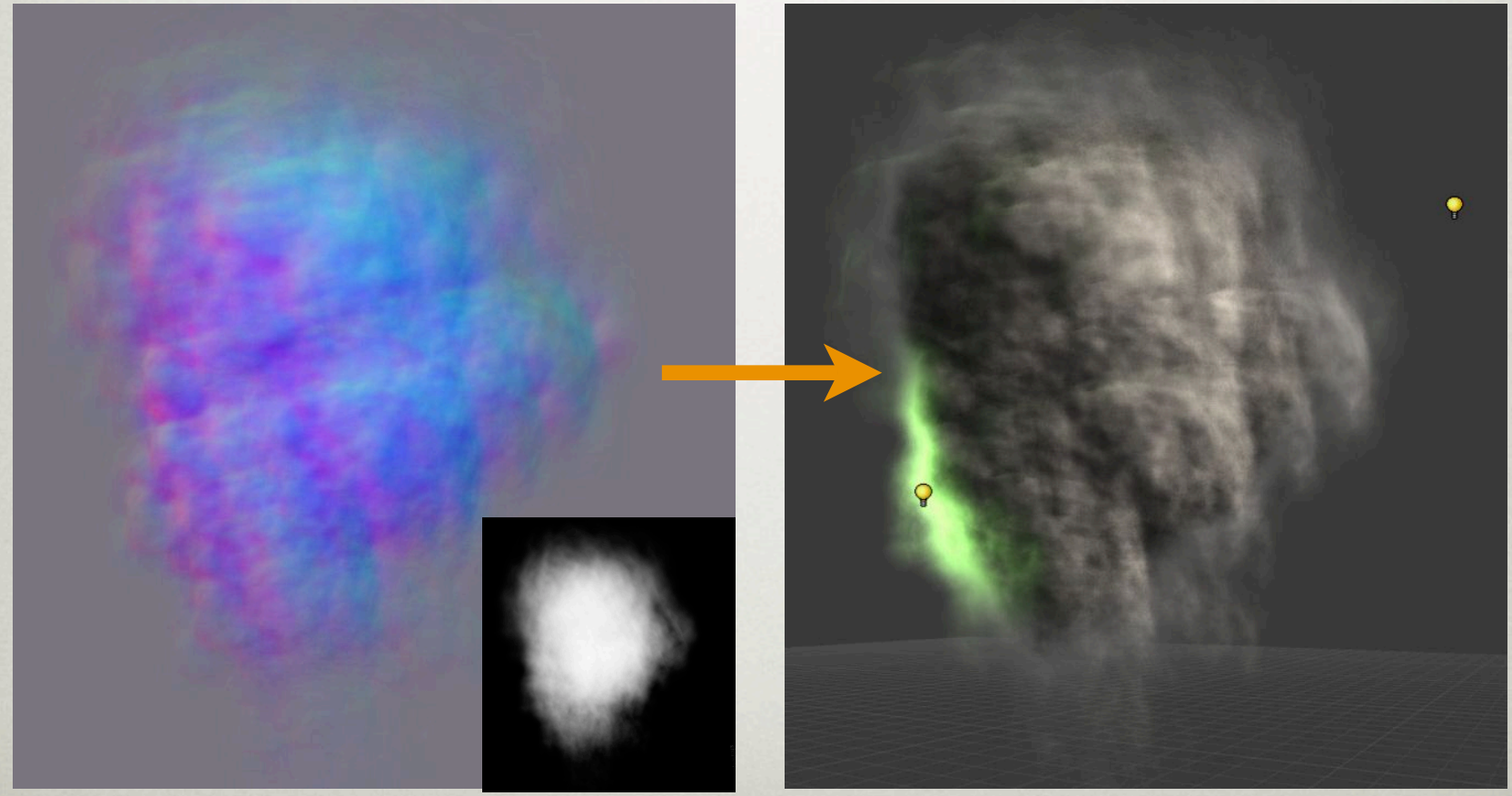




Transfer & Lighting

Accumulated Normals

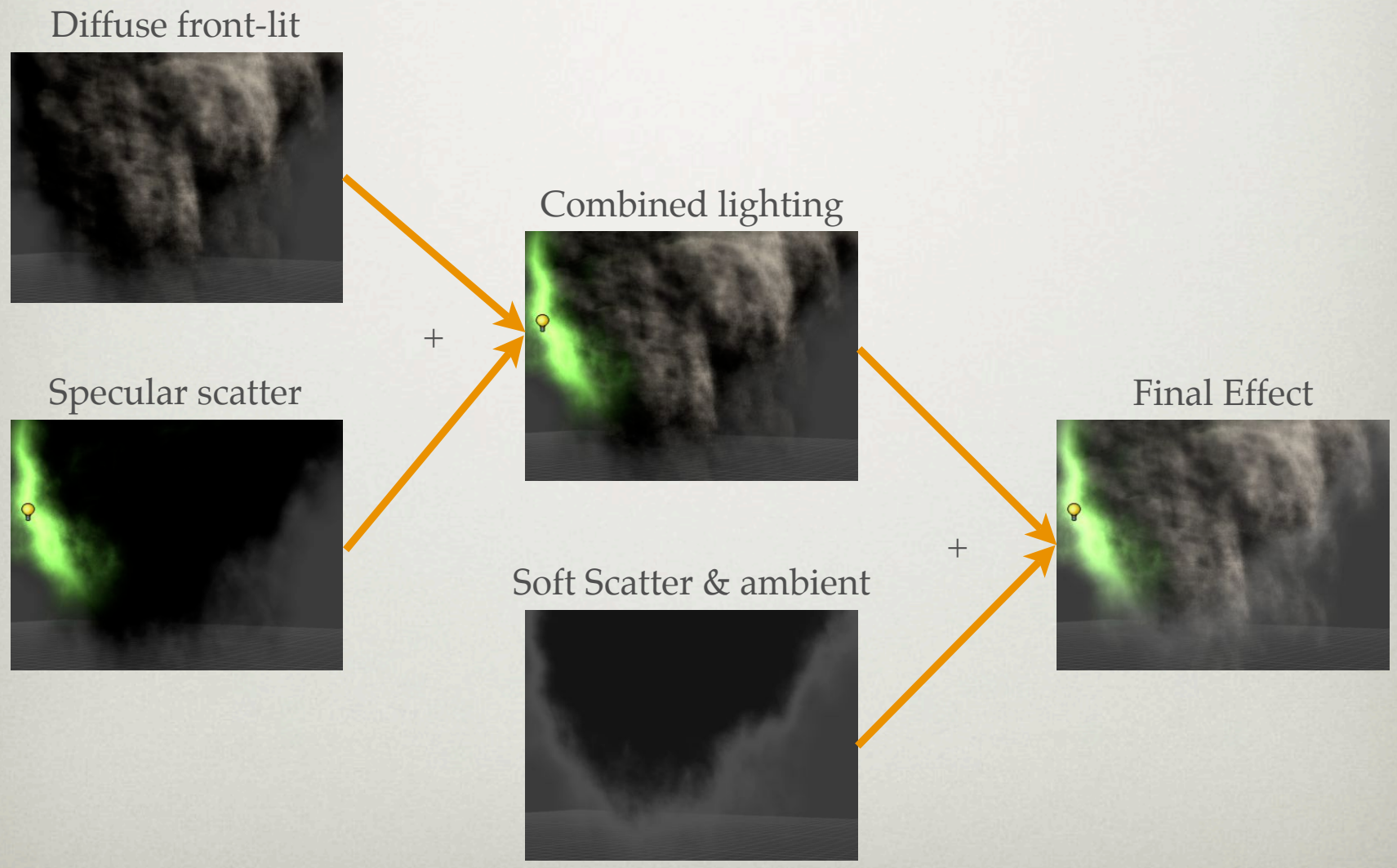
Lit & Blended



Transfer & Lighting

- We need to play nice with the depth buffer.
- Don't do a fullscreen quad - render the particle system's bounding box instead
  - Slightly tighter fit
  - Plays nicely with depth buffer of other objects in your scene
  - Needs special handling of near-plane clipping.
- To get per-particle depth testing with each particle going again, you have 2 options:
  - Render depth of bbox-intersecting objects to offscreen buffer
  - Or use FBOs to only rebind the color buffer & reuse existing main buffer depth info.

Shade Tree


Diffuse front-lit
Specular scatter
+
Combined lighting
Soft Scatter & ambient
+
Final Effect

Diffuse Lighting

- Surface lighting is just:
  surface normal • light direction
  - But we have lots of SSS going on, so a normal diffuse falloff doesn't look good.
- Instead, do the DOT, then lookup that into a ramp texture.
  - Make the gradient larger & softer.
  - Let your artist make the gradient to match the look of your game.
- We're rendering a BBox into the scene, so point light vectors will be weird
  - For shading **only**, flatten the vertices to a plane & calc light vector from that

Diffuse Lighting

CG fragment program

```
struct v2f {
    float4 pos : POSITION;
    float4 uv : TEXCOORD0;
    float3 lightnormal;
};

uniform samplerRECT _OffscreenTexture;
uniform sampler2D _DiffuseRamp;

float4 frag (v2f i) : COLOR {
    float4 col;
    float4 particle = texRECTproj (_OffscreenTexture, i.uv);

     float3 normal = normalize (particle.rgb * 2 - 1);

    float lightAmount = dot (i.lightnormal, normal);

    float2 lightUVlookup = float2(lightAmount * .5 + .5, 0);
    float3 lightFalloff =
        tex2D (_DiffuseCtrlTex, lightUVlookup).rgb;

    col.rgb = lightFalloff * _ModelLightColor.rgb;
    col.a = particle.a;

    return col;
}
```

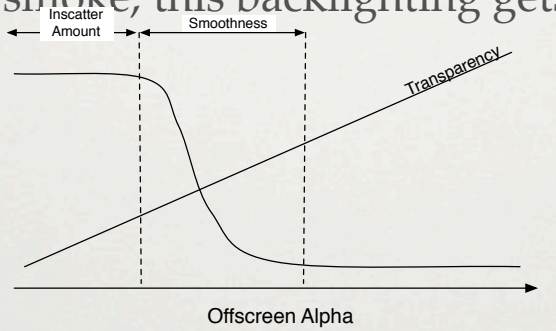
```
                                                    // Viewport space light normal


                                                    // Our offscreen texture
                                                    // Diffuse ramp texture


                                                    // Look up the offscreen texture to get normal

                                                    // Expand & normalize

                                                    // Calculate basic diffuse shading

                                                    // Do the ramp lookup


                                                    // Calculate final fragment color
```

Diffuse Lighting

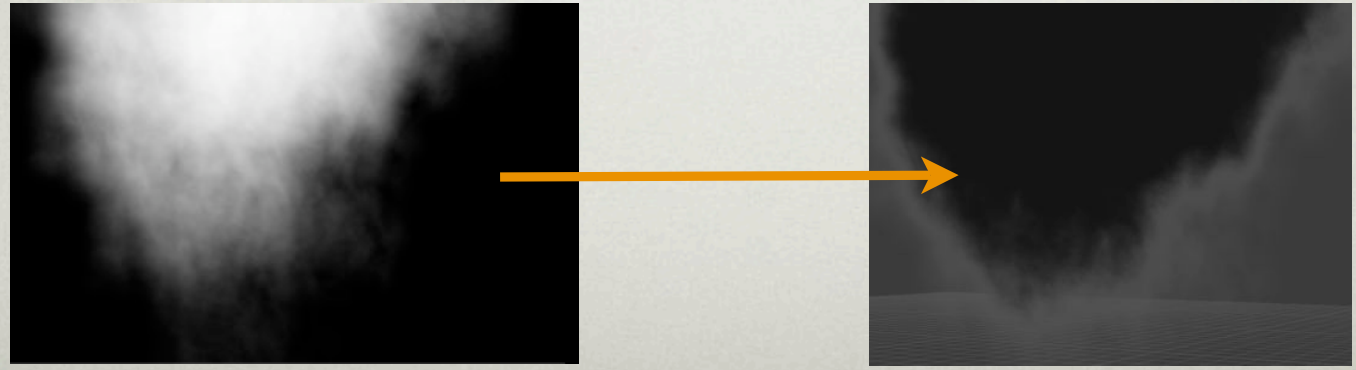
- All this is getting terribly slow.
  - Use cubemaps - saves the normalize & clamping (but requires a vector3 rotation).
  - Use motif lighting - render a small sphere with the lighting to an offscreen cubemap, then just use that in your shading pass.
  - Or use spherical harmonics?

Soft Scattering

- In thin smoke, there is a very even scattering going on
  - This is essentially ambient backlighting.
- As we get more smoke, this backlighting gets darker



- smoothstep (_ScatterAmt, _ScatterSmooth + _ScatterAmt, offscreen.a)



Diffuse Lighting

CG fragment program

```
struct v2f {
        float4 pos : POSITION;
        float4 uv : TEXCOORD0;
};

uniform samplerRECT _OffscreenTexture;

uniform float _ScatterAmt, _ScatterSmooth;
uniform float4 _ScatterColor, _AmbientColor;

float4 frag (v2f i) : COLOR{
        float4 col;
        float particle = texRECTproj (_OffscreenTexture, i.uv.xyw).a;

        float val = smoothstep (_ScatterAmt,
                _ScatterSmooth + _ScatterAmt, particle);
        col.rgb = lerp (_ScatterColor.rgb, _AmbientColor.rgb, val);

        col.a = particle.a * 3;
        return col;
}
```

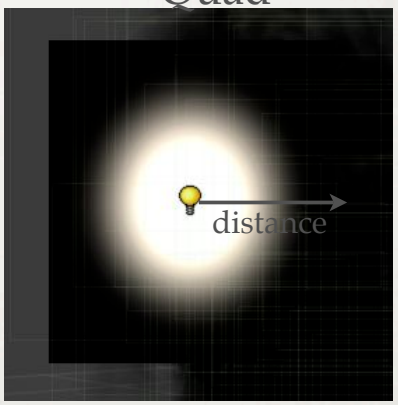
```
// Our offscreen texture

// Values described above
// The colors to lerp between

// Look up the offscreen texture to get dedpth

// Do fade from backlight to ambient

// Boost the alpha for blending
// and output the final color
```

All of this can be precalculated into a ramp texture

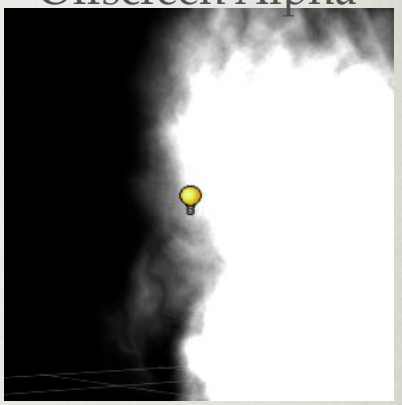
Specular Scattering

- Lights behind smoke tend to do a specular-like backlighting
- Add a quad around each light behind the smoke.
  - To make this feel like a specular effect, keep this quad a constant screen-space size.
  - Also helps fillrate
- To shade it, use another ramp texture - lookup with (distance^2, particle alpha)
- Fade out the quad as it goes through the smoke BBox.

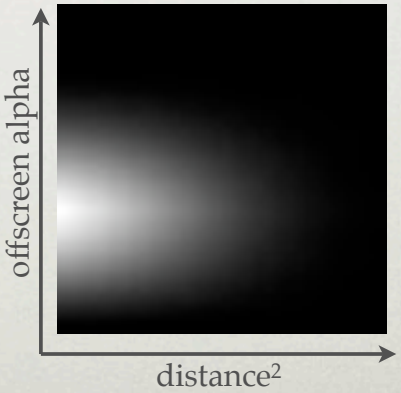
Quad



Offscreen Alpha



Ramp Texture



Final Result

Specular Scattering

CG fragment program

```
struct v2f {
        float4 pos : POSITION;
        float fog: FOGC;
        float4 uv : TEXCOORD0;
        float2 scatterUV;
        float3 color;
};

uniform samplerRECT _OffscreenTexture;
uniform sampler2D _ScatterTex;

float4 frag (v2f i) : COLOR{
        float4 col;
        float4 particle = texRECTproj (_OffscreenTexture, i.uv);

        float2 scatterLookup;
        scatterLookup.x = dot (i.scatterUV, i.scatterUV);
        scatterLookup.y = particle.a;

        col.rgb = tex2D (_ScatterTex, scatterLookup).rgb
        col.rgb *= i.color;

        return col;
}
```

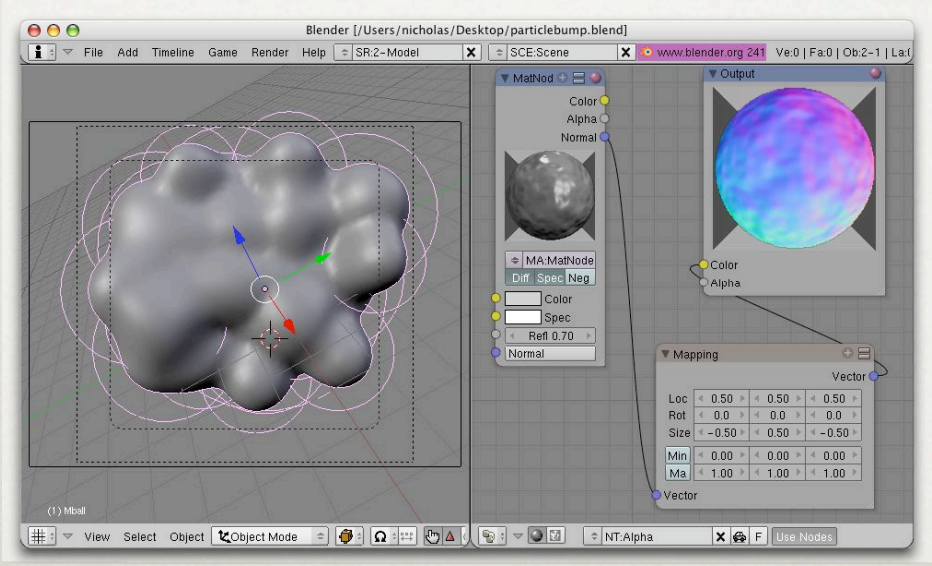
```
// deferred particles UV
// Scatter Quad in range (-1, -1) - (1, 1)
// Faded light color for the quad


// Our offscreen texture
// Diffuse ramp texture



// Look up the offscreen texture to get alpha

// Build the ramp lookup
// X is distance to particle
// Y is smoke amount

// Do the ramp lookup
// mul in faded color

// Calculate final fragment color
```

Speed

- How can this be fast?
  - Particles are inherently overdraw-heavy - we're fillrate bound.
  - We can keep the offscreen buffers at lower res to conserve fillrate. Clamp the resolution.
- We only do one fullscreen quad to get them on to the screen.
- This means the worst-case result (lots of particles right in front of the camera) gets tighter shading & fill bounds.

Details

- Getting the normalmap right is a bitch.
  - Grayscale heightmaps never get ‘curvy’ enough.
  - Get your gfx artist to ZBrush you a wispy cloud.
  - Or use metaballs, then add some procedural bump



Details

- The hard part is getting the cubemap working
  - Start simple: Get a diffuse cubemapped particle to play nicely with your lighting system!

Questions?

- Presentation & demo available

  www.otee.dk/blogs/nf
- Or write me:
  - nicholas@otee.dk