Making Pretty Pictures with D3D

Or…

A hand-wavey tour through some case studies of how to bring a graphics card to its knees in the persuit of ‘greater beauty’

Alex Evans, Lionhead Studios

And Relax…

- Amid all the awesome tools available its easy to forget why we're making these engines…
  - Easy to forget simple things, e.g.
    - there's more to the rendering style spectrum than just toon shading or hyper-realism

I promise this rant will be quick!

- It's time for games to make more an effort to look different from each other
  - and not just driven by the actual art assets themselves
- Shaders are opening up our way of thinking about PC graphics coding

Lots can be done with very little!

- It doesn't have to be complex to look great
  - E.g. Lots of alpha layers
    - Blooming, radial blur, particles
  - Hiding polygon edges
    - Silhouette polygons, blur, post-processing, noise
- So let's get down to business...
    - 4 different case studies coming up, ranging from simple to just-plain-weird…

#1: DOF in 2.5D games

- Aesthetic / design decision:
  - Limit ourselves to 2D billboards
    - Placed in a full '3D' space
  - Blurring is as simple as choosing a mip-map to render with
    - Just set D3DSAMP_MIPMAPLODBIAS
  - This trick is as old as the hills!
    - But it looks great...

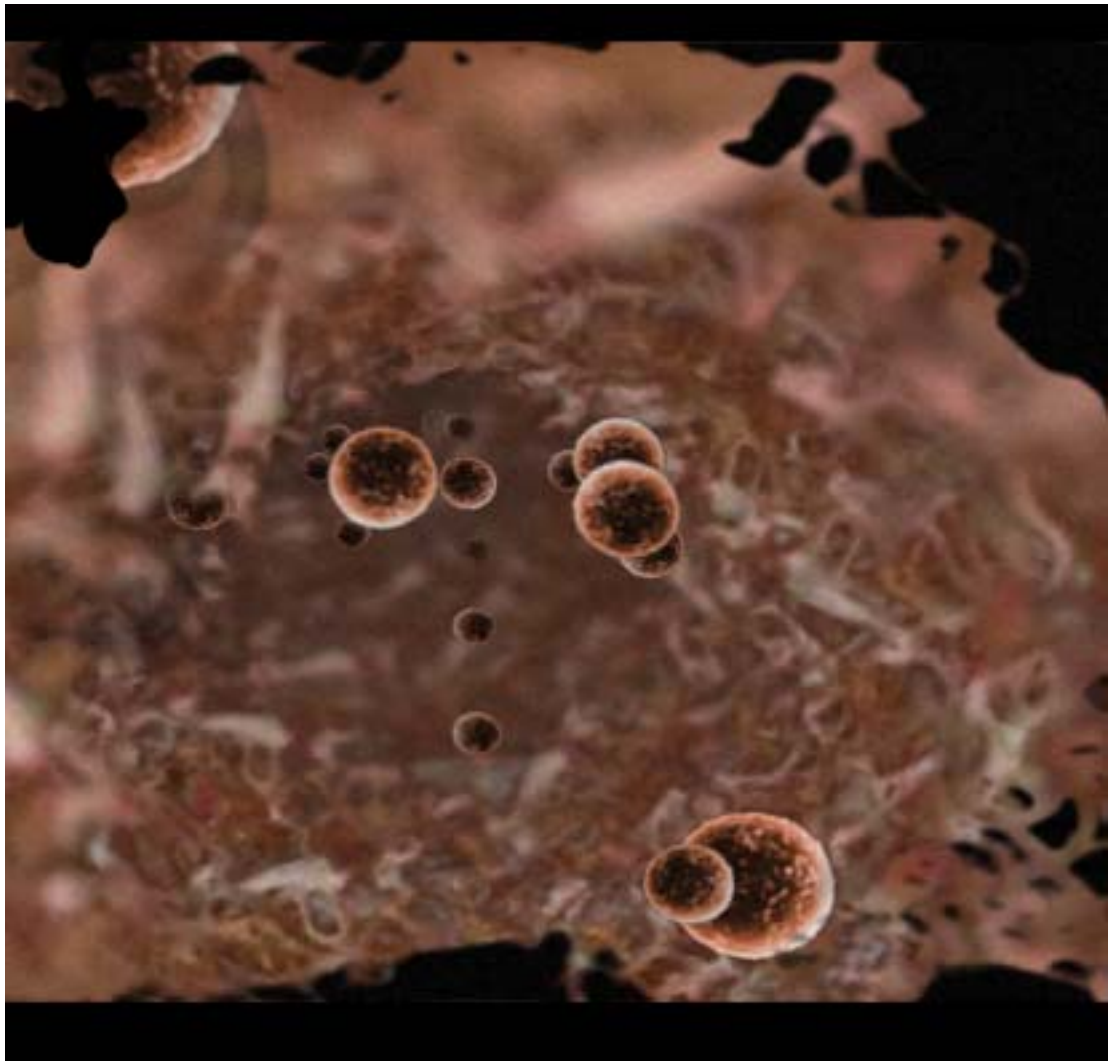


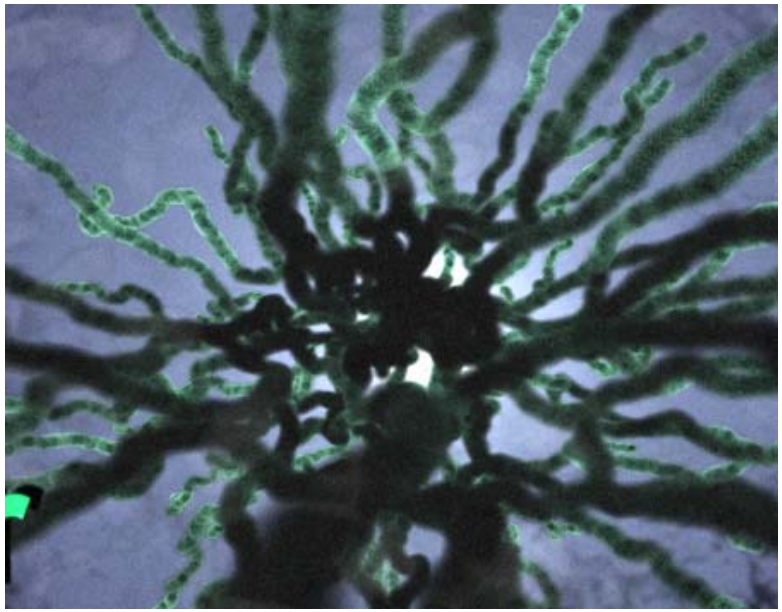
Image from ‘we cell’ by kewlers

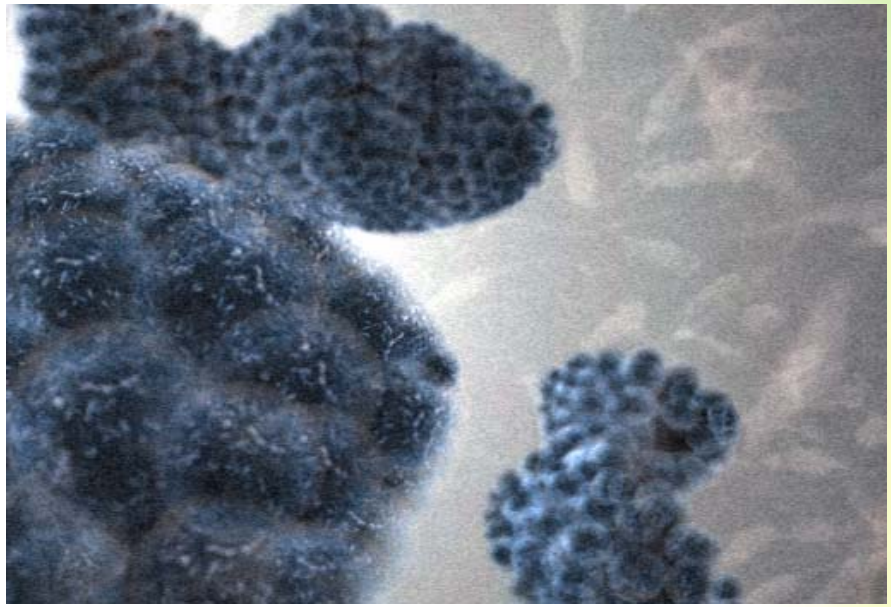
2D DOF problems

- Must keep big alpha 0 borders
- Box filter is ugly
  - Could use something better
  - Can force the borders to alpha 0 here too

More with less cont…

- When you combine tricks like this with the massive fill-rate of modern cards…
  - You get some lovely results
  - Best example I've seen recently:
    - 'We Cell' by kewlers
    - Check out their awesome work
      - http://kewlers.scene.org





- Images from ‘we-cell’ by kewlers
- No polygons (other than billboards…)
- No Shaders!

#2: Lighting in 2.5D Games

- Art-led engine feature request:
  - How to shade these billboards under changing lighting conditions?
  - Could use normal maps…
    - But they are not always an ideal fit – it imposes a lighting model that must be evaluated at run time.
    - And what about self-shadowing?
  - What if we could just pre-render every possible light direction?

Lots of different light dirs…

Lighting continued…

- Every possible direction is too memory heavy, but is flexible
  - (Lossy) Compression please!
- The pixel colours are generally slowly varying with light direction
  - Diffuse / glossy assumption here
  - In ‘3D-world’, Spherical Harmonics have been helping us out here…

‘Circular Harmonics’

- Project source images onto basis:

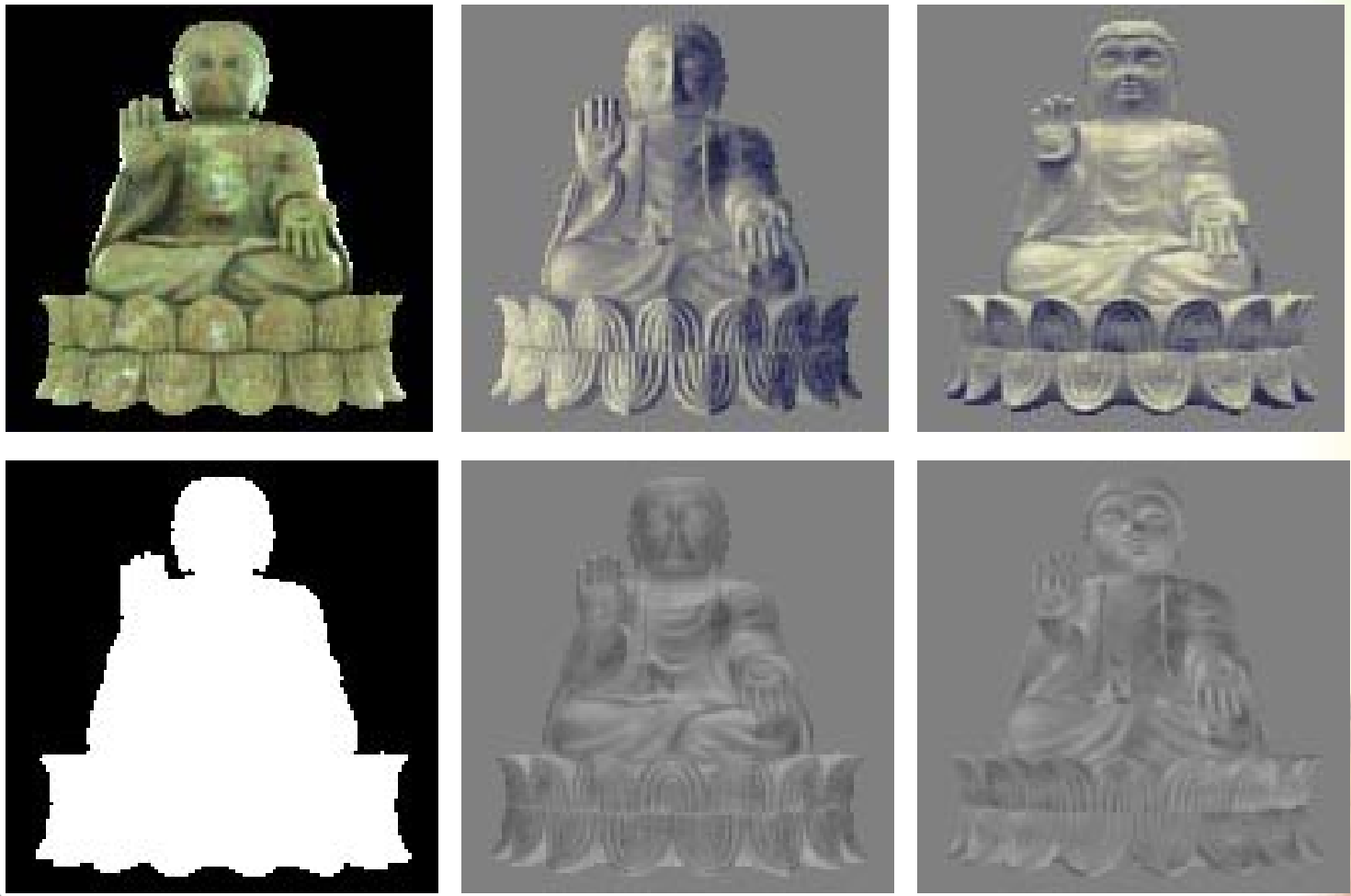
$$B_0 := \theta \rightarrow \sqrt{\frac{1}{2\pi}}$$

$$B_{2\,i} := \theta \rightarrow \frac{\sin(\theta\, i)}{\sqrt{\pi}} \qquad B_{2\,i+1} := \theta \rightarrow \frac{\cos(\theta\, i)}{\sqrt{\pi}}$$

- Gives us some small number of Y_i
  - For each pixel, for each channel

$$Y_i := \int_0^{2\pi} X B_i(\theta)\, d\theta$$

Circular Harmonics Pictures



Rendering it on PS_1_1

- C(++) code computes lighting and projects onto our basis, as before
- Uploads as 5 constants:
  - `c0 = B0r, B0g, B0b, 1`
  - `c1 = B1r, B1g, B1b, 1`
  - `c2 = B2r, B2g, B2b, 1`
  - `c3 = 0, 0, 0, B3`
  - `c4 = 0, 0, 0, B4`

$$Y_i := \int_0^{2\pi} X B_i(\theta)\, d\theta$$

Rendering it on PS_1_1

```
ps.1.1
tex t0
tex t1
tex t2
mul r0,t0,c0
mad r0.rgb,t1_bx2,c1,r0
mad r0.rgb,t2_bx2,c2,r0
mad r0.rgb,t1_bx2.a,c3.a,r0
mad r0.rgb,t2_bx2.a,c4.a,r0
mul r0,r0,v0
```

Rendering it on PS_1_1

```
ps.1.1
tex t0
tex t1
tex t2
mul r0,t0,c0
mad r0.rgb,t1_bx2,c1,r0
mad r0.rgb,t2_bx2,c2,r0
mad r0.rgb,t1_bx2.a,c3.a,r0
mad r0.rgb,t2_bx2.a,c4.a,r0
mul r0,r0,v0
```

r0 = lighting B0 * first texture rgb

Rendering it on PS_1_1

```
ps.1.1
tex t0
tex t1
tex t2
mul r0,t0,c0
mad r0.rgb,t1_bx2,c1,r0
mad r0.rgb,t2_bx2,c2,r0
mad r0.rgb,t1_bx2.a,c3.a,r0
mad r0.rgb,t2_bx2.a,c4.a,r0
mul r0,r0,v0
```

```
r0 += lighting B1 * second texture rgb
```

Rendering it on PS_1_1

```
ps.1.1
tex t0
tex t1
tex t2
mul r0,t0,c0
mad r0.rgb,t1_bx2,c1,r0
mad r0.rgb,t2_bx2,c2,r0
mad r0.rgb,t1_bx2.a,c3.a,r0
mad r0.rgb,t2_bx2.a,c4.a,r0
mul r0,r0,v0
```

```
r0 += lighting B2 * third texture
```

Rendering it on PS_1_1

```
ps.1.1
tex t0
tex t1
tex t2
mul r0,t0,c0
mad r0.rgb,t1_bx2,c1,r0
mad r0.rgb,t2_bx2,c2,r0
mad r0.rgb,t1_bx2.a,c3.a,r0
mad r0.rgb,t2_bx2.a,c4.a,r0
mul r0,r0,v0
```

```
r0 += lighting B3 * second texture alpha
```

Rendering it on PS_1_1

```
ps.1.1
tex t0
tex t1
tex t2
mul r0,t0,c0
mad r0.rgb,t1_bx2,c1,r0
mad r0.rgb,t2_bx2,c2,r0
mad r0.rgb,t1_bx2.a,c3.a,r0
mad r0.rgb,t2_bx2.a,c4.a,r0
mul r0,r0,v0
```

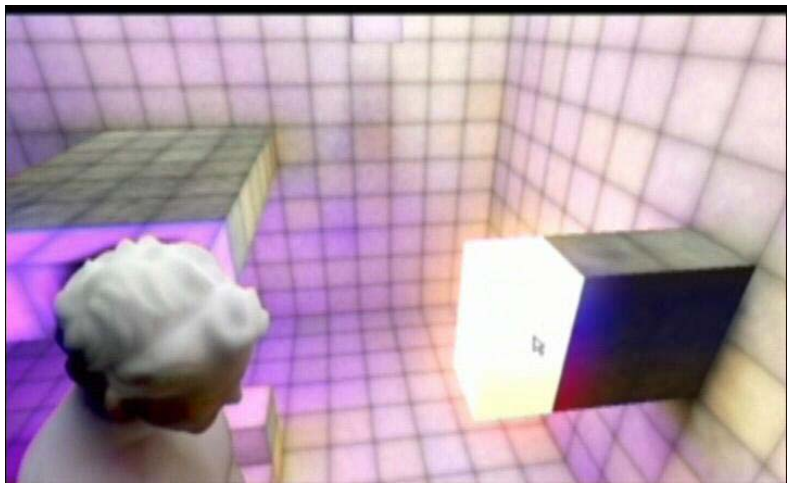
```
r0 += lighting B4 * third texture alpha
```

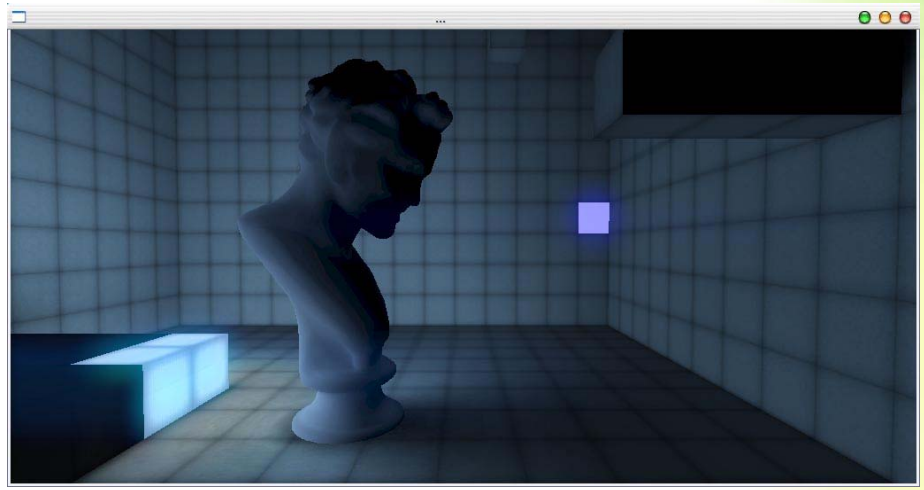
Rendering it on PS_1_1

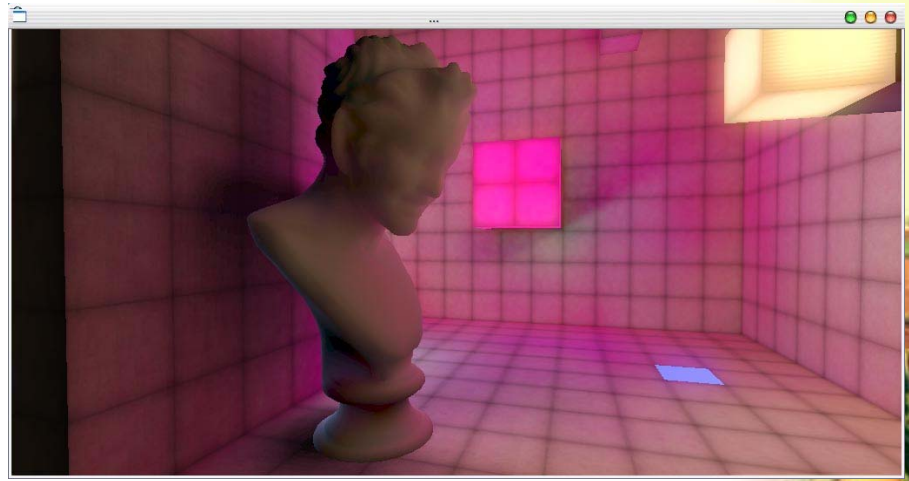
```
ps.1.1
tex t0
tex t1
tex t2
mul r0,t0,c0
mad r0.rgb,t1_bx2,c1,r0
mad r0.rgb,t2_bx2,c2,r0
mad r0.rgb,t1_bx2.a,c3.a,r0
mad r0.rgb,t2_bx2.a,c4.a,r0
mul r0,r0,v0
```

Modulate by vertex colour and output

#3: ‘GI’ on the GPU (cube version)







‘GI’ on the GPU (cube version)

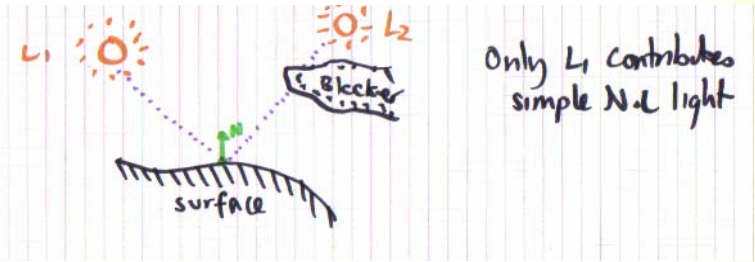
- Implementation of soft shadowing
  - with loads of area lights
  - and totally dynamic scenes
  - ‘traditional’ algorithms are cost O(N) in the number of lights
  - Recast the problem so it’s O(1) in the number of lights
    - but a very big 1 ☺

Demo!

- Look at it go!
  - Was written on a Radeon 9700
    - Makes heavy use of MRTs, 16bit targets
      - The e.g. X800 flies along in comparison
      - But volume texture reads are mem b/w limited
- Dynamic shadows of the character coming up in a second…

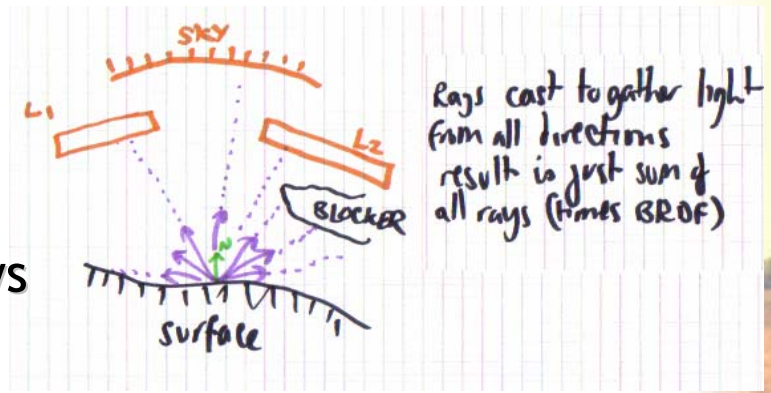
Path Tracing 101

- **Traditionally, to shade a point, you 'cast rays'**
  - **But only to point light sources. Then accumulate**



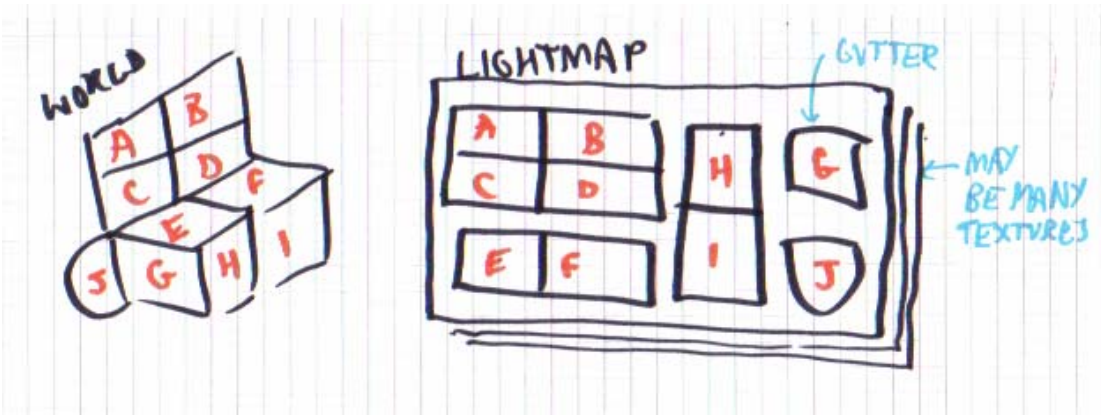
In Path tracing / Distribution ray tracing, light is gathered from all directions

- allowing bounce light, area lights and therefore soft shadows



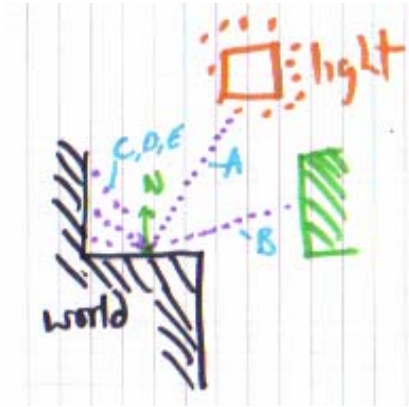
Doing it on the GPU

- Need to cast zillions of rays
  - Imagine if we could trace one ray for every pixel computed by a shader
    - Step 1: Cover the world in a lightmap

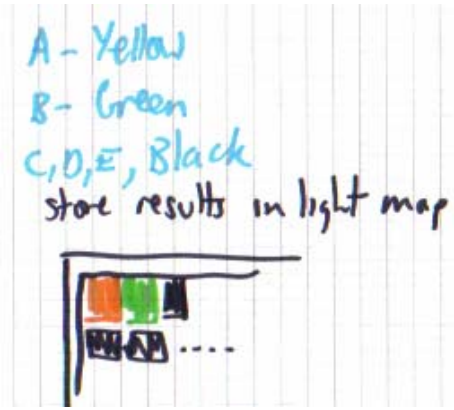


GPU GI cont…

- **Step 2: imagine casting rays out in random directions with some magic pixel shader**

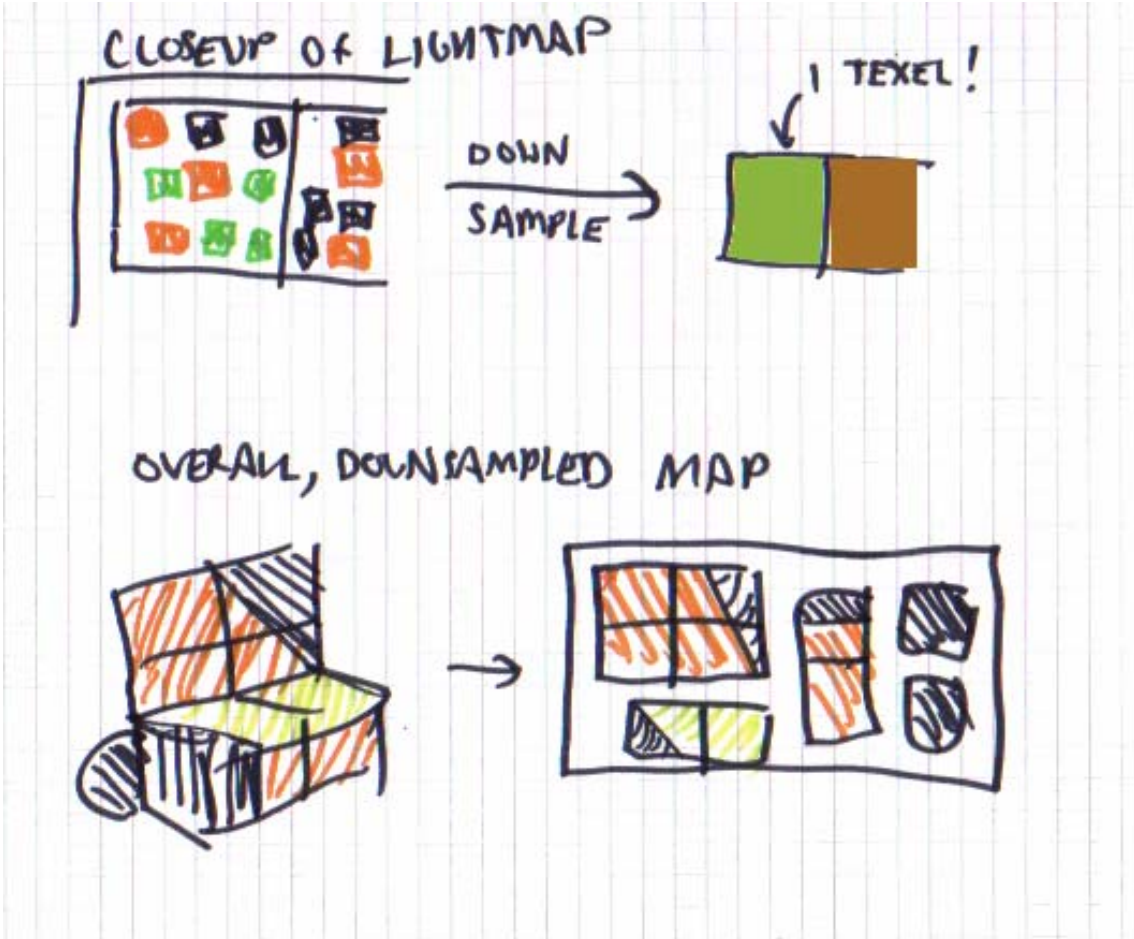


Step 3: The output of the pixel shader is the colour of the first surface or light that is hit by that ray



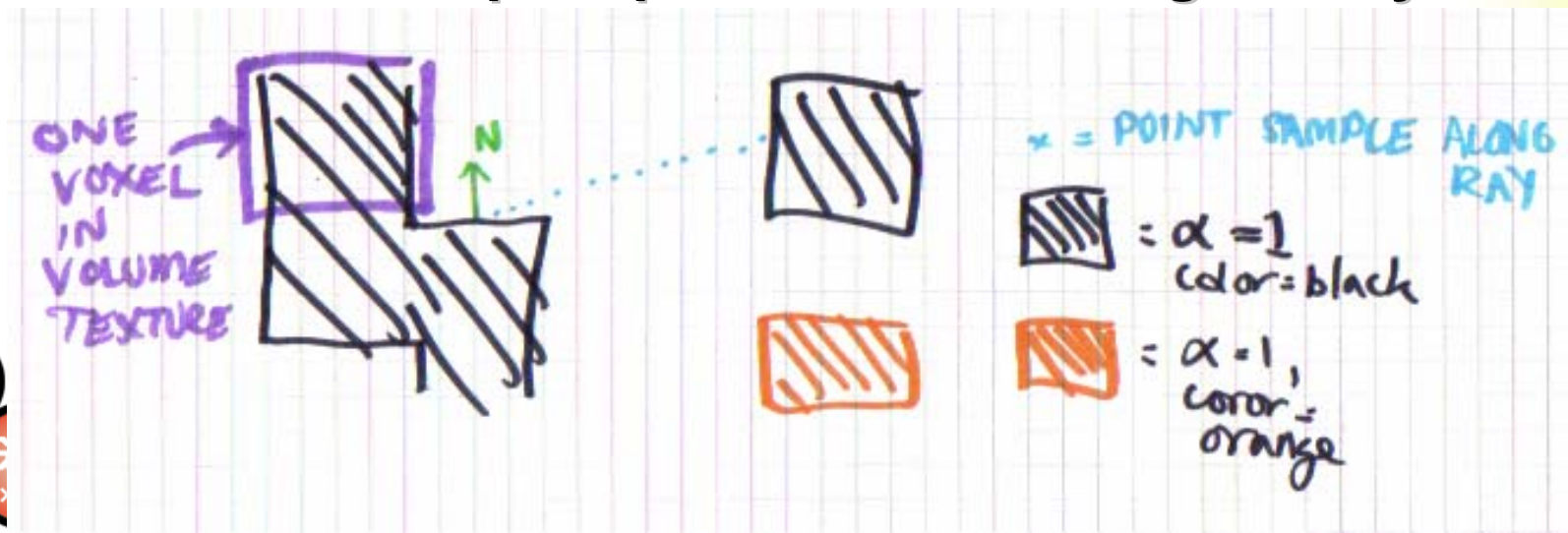
GPU GI cont…

Step 4: downsample to average the results of many rays



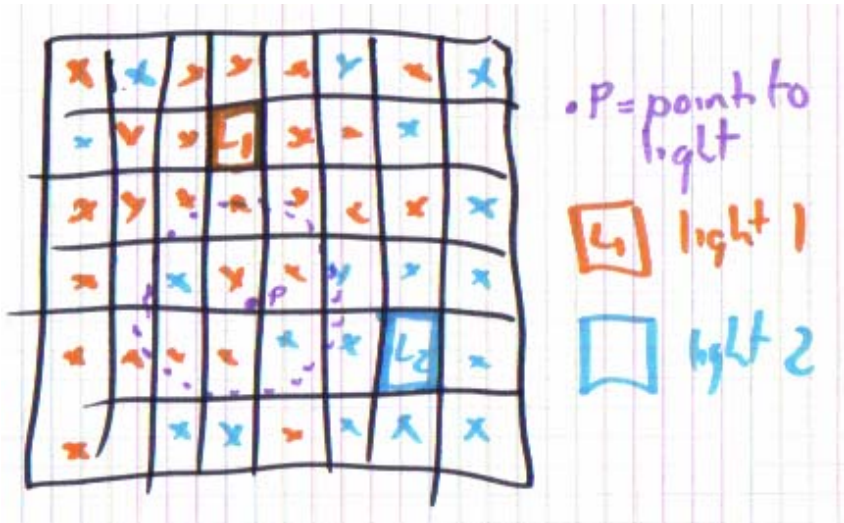
Making it Practical

- Imagine a world made entirely of cubes
  - Store the whole world in a volume texture
  - DXT compressed, 1 bit alpha = solid/not solid
  - RGB channel stores emissiveness (=lights)
- Tracing a ray now just becomes sampling
  - Can sample up to 20 times along the ray



Making it Practical 2

- There are many more details to the actual implementation
  - Getting good random numbers into the PS is hard
  - Stochastic sampling, importance sampling
    - This is implemented using a technique akin to dithering, but in 3D space over light directions.



But it all fits in PS_2_0!

```
const static int numloops = 4;
float4 steppattern = tex2D(steptex, posincell); // each element is 'nearly 1'
float4 destposx = tex2D(desttex, posonpage) ;
float3 destpos=float3(destposx.x,destposx.y,destposx.z);
float3 d = destpos - (posinworld-Zero);
float4 temp;
float outa=1.f;
for (int loopy=0;loopy<numloops;loopy++)
{
            temp=tex3D(boxtex, posinworld+d*steppattern.x);
            if (temp.a>0) outa=steppattern.x;
            temp=tex3D(boxtex, posinworld+d*steppattern.y);
            if (temp.a>0) outa=steppattern.y;
            temp=tex3D(boxtex, posinworld+d*steppattern.z);
            if (temp.a>0) outa=steppattern.z;
            temp=tex3D(boxtex, posinworld+d*steppattern.w);
            if (temp.a>0) outa=steppattern.w;
            steppattern-=0.25;
}
float4 outcol = tex3D(boxtex, posinworld+d*outa) ;
d*=worldscale;
float dlen=1.f/length(d);
float3 dnorm = d*dlen;
float dotty=dot(normal,dnorm);
dotty=max(dotty+0.1,0); // clamp to > 0 so we get some ambient
dotty*=A*dlen*dlen+B; // 1/r2 falloff
return outcol * dotty / destposx.w;
```

The down-sides…

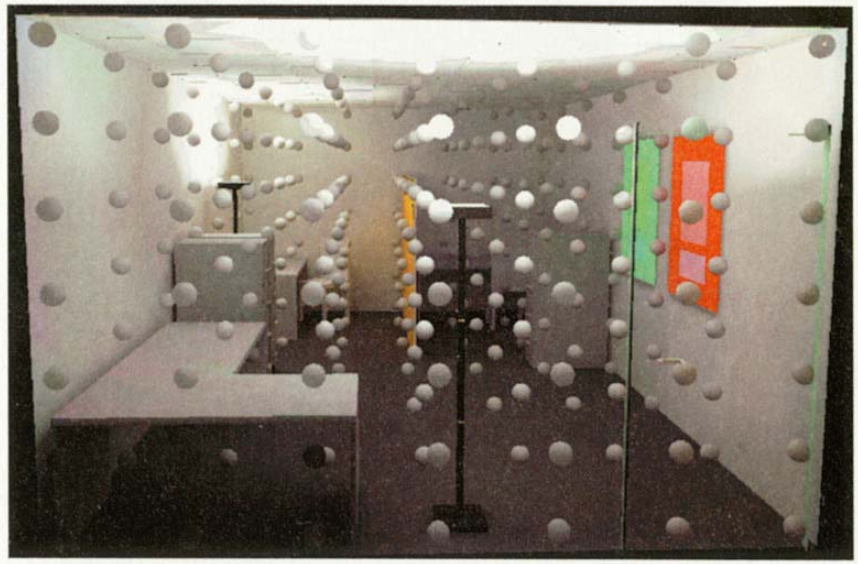
- Why was that algorithm slow?
  - A lot of rays were cast
    - But they're all independent
  - And it was limited to worlds represented as cubes
  - Offline renderers solve this issue by 'caching' the results of similar rays
    - Photon Mapping
    - Irradiance Caches
  - But that's another talk…
    - Or try different representations of your world
      - Eg Mike Bunnell, GPU Gems 2

Dynamic object shadows in cube-world

- The fuzzy shadows cast by the marble bust in the cubeworld demo…
  - Works independently of the GPU algorithm just described
  - But it is chosen carefully to ‘match’ – it is based on SH techniques that are of (approximately) fixed cost regardless of the number or size of the area lights in the scene.

SH Volume Transfer

- The Irradiance Volume [Greger98] says:
  - Imagine storing the incident lighting at every point in space projected onto the SH basis…
- Surfaces nearby have an effect on the light at every point
  - They bounce and occlude light
  - This has some relation to [Greger98], which Natalya covered already in her talk today. (but have that pic one more time!)
- What we're going to do is store *occlusion* information in a volume texture



SH Volume transfer

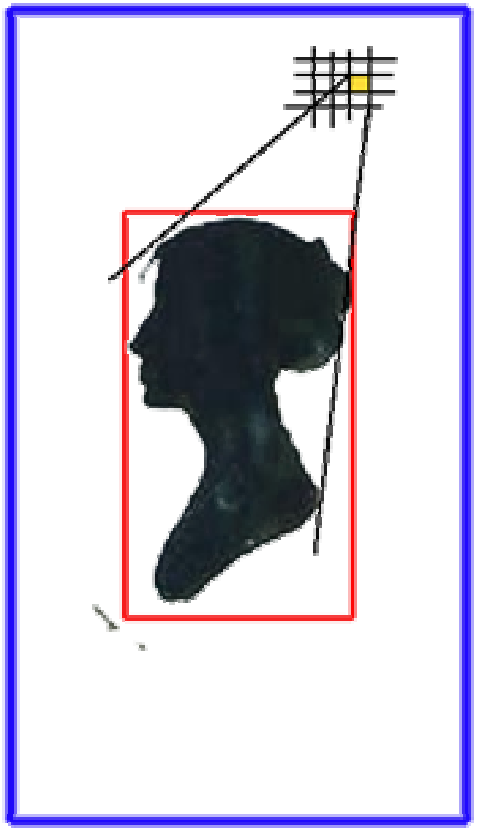
- When the lighting is stored as truncated SH
  - The influence of an object can be precomputed for all points in space
    - As a transfer matrix which maps incoming radiance expressed in SH basis, to outgoing radiance taking into account the object's effect on lighting.
    - See Sloan et al first paper on SH
      - Section 8!

too hard for GPUs?

SH Volume Transfer the 'Wrong' Way

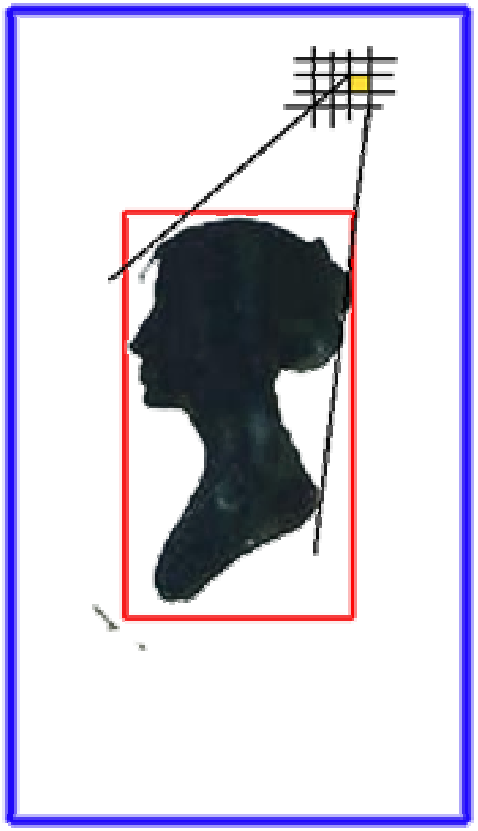
- Preprocess step embeds shadow caster in double size bounding box
- The processor is going to output a 32x32x32 volume of SH coefficient VECTORS (not matrices)
  - The demo used PCA compression to reduce the number of coefficients per voxel to 16
  - We're effectively assuming the bottom rows of the matrix are all 0s.

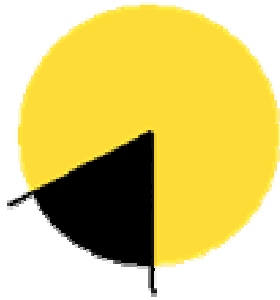
SH Volume Transfer preprocess…

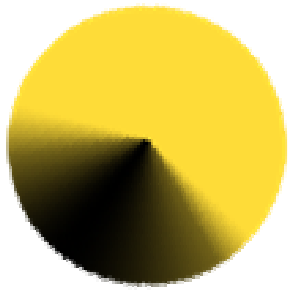


- Each voxel casts rays to determine which directions are occluded
  - The results are projected onto the SH basis
  - This is like a much simplified light transport (PRT) step
    - Except we don't bother with bounce light, the cosine term (or any BRDF)
    - And it's over all of space, not just the surface of the mesh.

SH Volume Transfer preprocess…



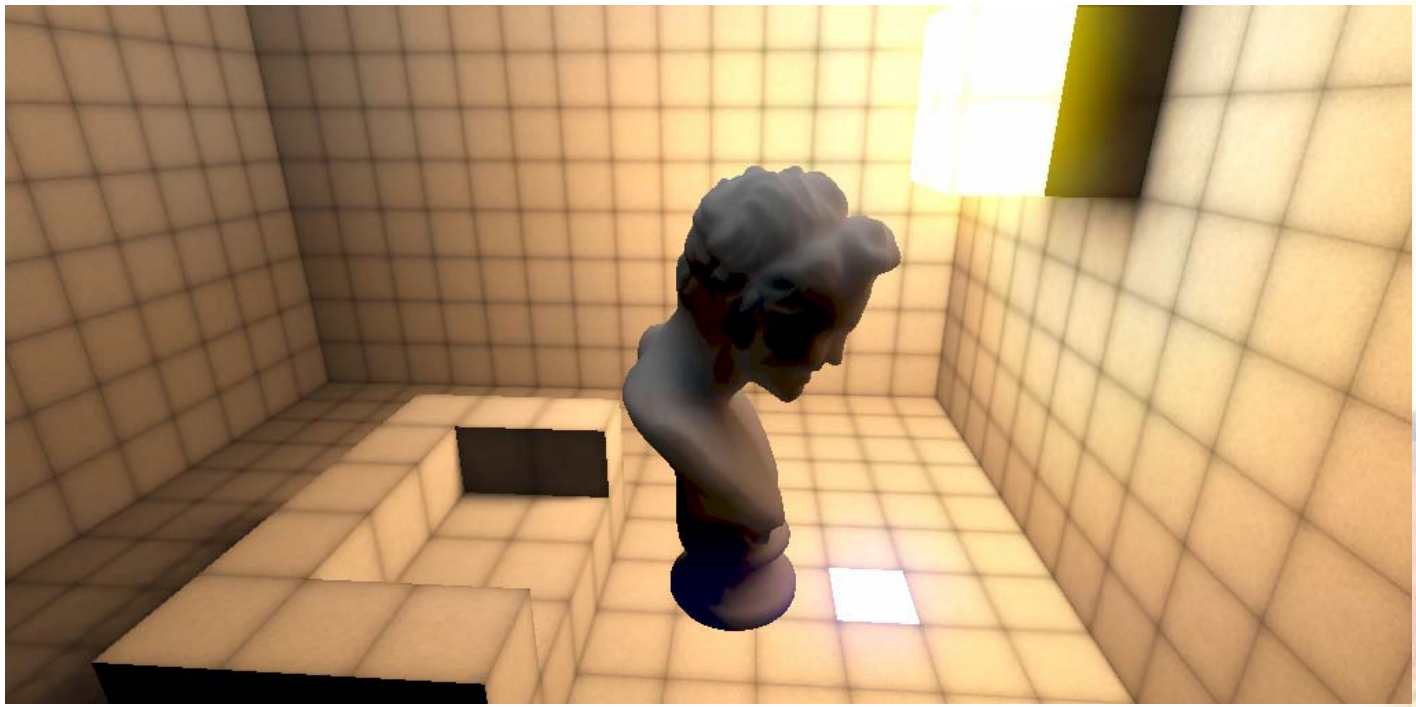




- Example voxel and the result of the SH ‘compression’ - a sort of fuzzy occlusion map

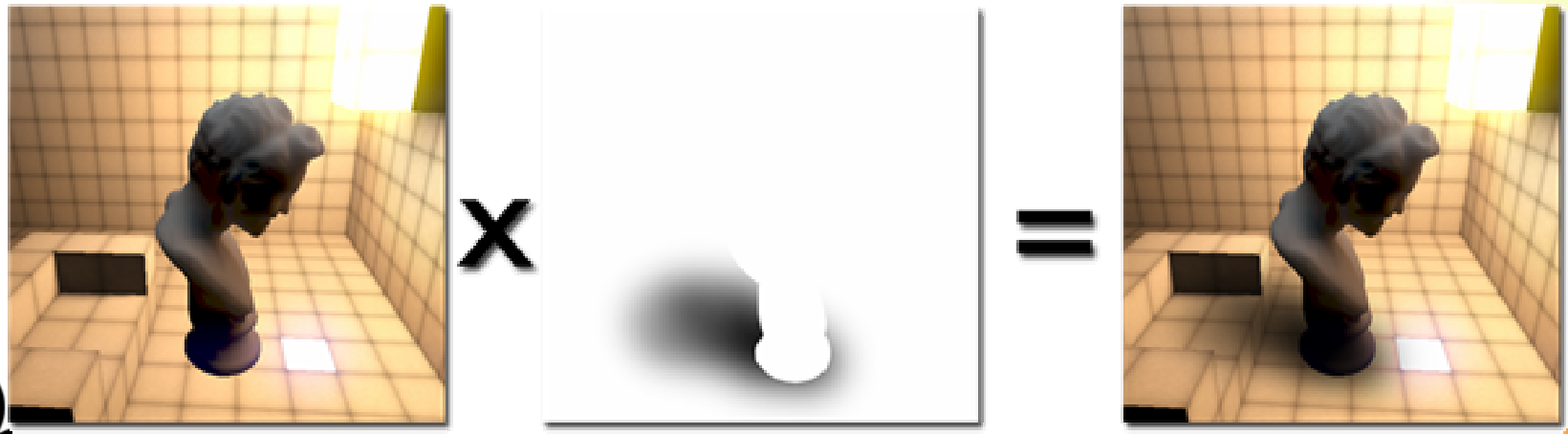
SH Volume transfer at runtime…

- After the background has been lit and the dynamic objects rendered…
  - I happen to light the dynamic objects using the standard SH PRT technique



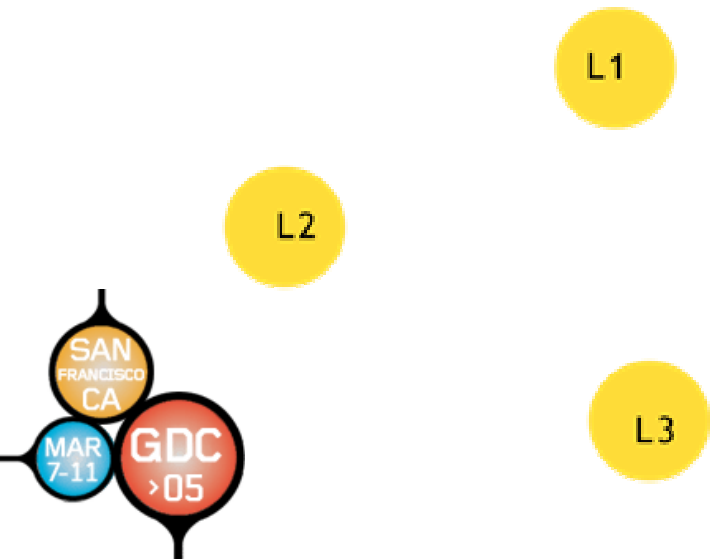
…shadows are added…

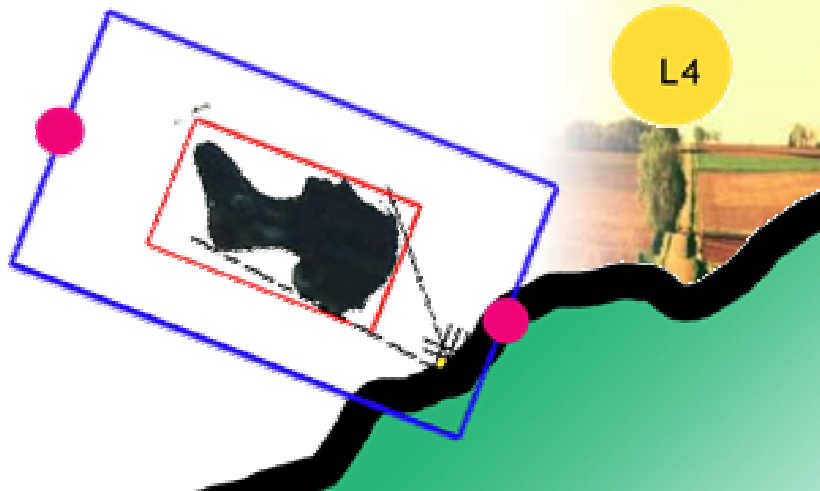
- …in a ‘deferred’ way
  - By rendering a fullscreen quad over the screen with a big pixel shader
    - For each pixel, the shader computes the occlusion of that object, and outputs a scalar value to darken the screen.



SH Volume transfer runtime…

- The CPU computes the incident lighting for the object at 2 candidate points
  - Chosen at ends of object's longest axis
    - Improves locality of lighting
    - This is a general technique applicable to all lighting approximations that rely on an assumption of 'distant lighting'





SH Volume transfer runtime…

- To compute the lighting at a point, the pixel shader:
  - Rotates the point to be shaded into the object's local space
  - Interpolates the lighting environments uploaded by the CPU, according to the position along the longest axis
  - Dots the lighting environment with the occlusion data looked up from the precomputed volume texture(s)

But it all fits in PS_2_0!

```
float4 main(float2 t0: TEXCOORD0) : COLOR
{
        float4 worldpos=tex2D(postex,t0);
        float id=worldpos.w;
        worldpos.w=1;
        worldpos=mul(worldpos,worldmat); // rotate world pos into local object
space

        float zlerp=saturate(worldpos.z+0.5);

        float4 a=abs(worldpos);
        float funkiest=max(1,max(a.x,max(a.y,a.z)));
        worldpos=worldpos*(0.5f/funkiest);

        float4 aa=tex3D(shadowtex0,worldpos);
        float4 bb=tex3D(shadowtex1,worldpos);
        float4 cc=tex3D(shadowtex2,worldpos);
        float4 dd=tex3D(shadowtex3,worldpos);

        float outcol1 = dot(float4(dot(aa,lighting[0]),dot(bb,lighting[1]),
                                   dot(cc,lighting[2]),dot(dd,lighting[3])),1);
        float outcol2 = dot(float4(dot(aa,lighting2[0]),dot(bb,lighting2[1]),
                                   dot(cc,lighting2[2]),dot(dd,lighting2[3])),1);
        return lerp(outcol1,outcol2,zlerp);
}
```

Help! I've been blinded by science!

- And who would want to make a renderer that can only do cubes?
  - ☺ There is interesting stuff you can do by enabling trilinear filtering, effectively creating a 'signed distance function'…
- There is *so* much headroom left on current cards
  - Even using PS1.1 … or PS0!
  - Restrictions in the game design are *great* for inspiring novel (non-general) solutions.
- Exploring 'whacky' rendering approaches, or hybrids of existing approaches
  - Allows the engine to be more tightly tailored to the aesthetics of the game/application
  - And will lead to more variety and imagination (?) in the visual style of games.

But…

- ‘Real Life Game Making’ militates against this
  - But the march of time can only make things better
    - As we collectively get more used to algorithms and tools such as SH, we ‘internalise’ them and feel less afraid to experiment.
    - ‘Realtime Cinematic Rendering’ doesn’t just mean great looking surfaces
      - hopefully it will bring to games the same breadth of visual styles that are evident today in film.
- “And with a little effort and careful design, a lot is possible today.”
  - Go Jerry! Go Jerry! Go Jerry!

References

- [Greger98] - Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg, "The Irradiance Volume", IEEE Computer Graphics and Applications, 18(2):32--43, March 1998
- [Malzbender01] - Tom Malzbender, Dan Gelb, Hans Wolters "Polynomial Texturemaps", SIGGRAPH 2001
- [Sloan02] -Peter-Pike Sloan, Jan Kautz, and John Snyder, Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments, SIGGRAPH 2002,July, 2002
- [Bunnell04] – Mike Bunnell, "Dynamic Ambient Occlusion and Indirect Lighting", Chapter 14 GPU Gems 2
- [Kewlers05] - Kewlers, "We Cell", http://kewlers.scene.org

Questions?

- aevans@lionhead.com

- Thanks:
  - Mark Healey, Dave Smith @ LH
  - Jason Mitchell, Mike Smith, Kevin Strange & Richard Huddy @ ATI