SGI OpenGL Shader

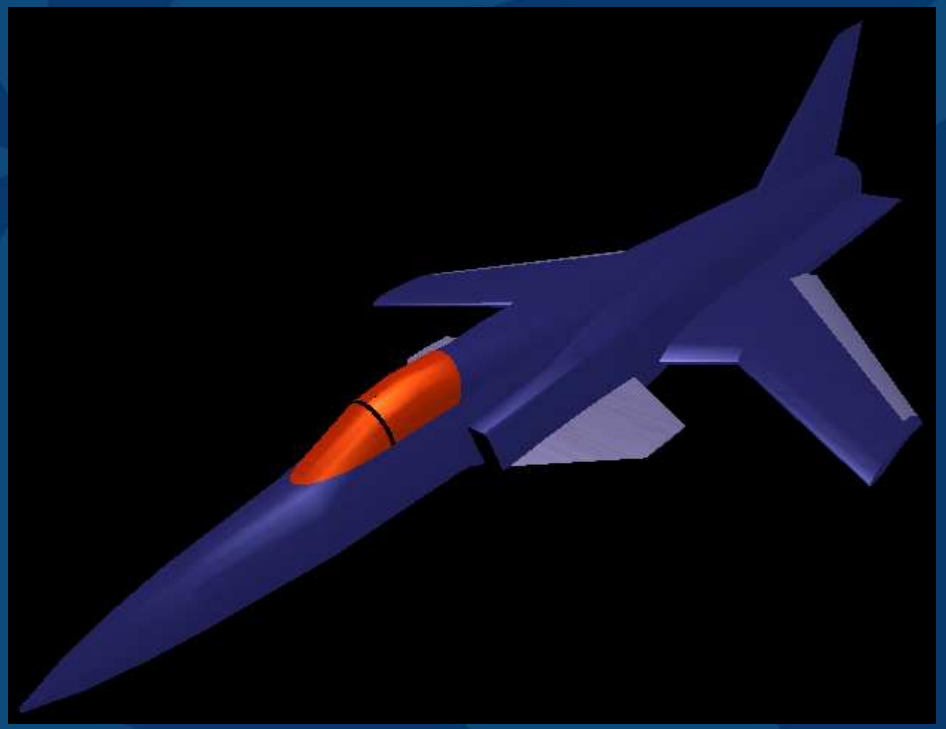
Marc Olano
SGI

Interactive Rendering

SAN ANTONIO SIGGRAPH 2002

Illusion of Presence

- 10 – 30 – 60 frames per second
- Immediate response
- Simple appearance

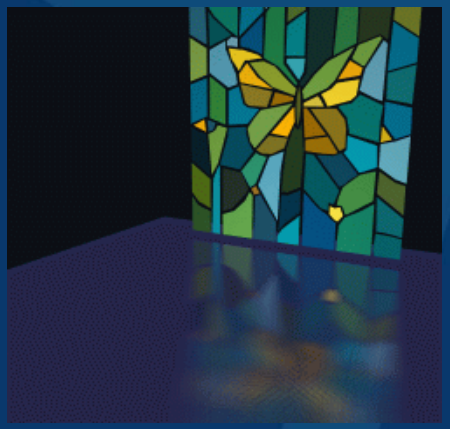


Multi-pass Rendering

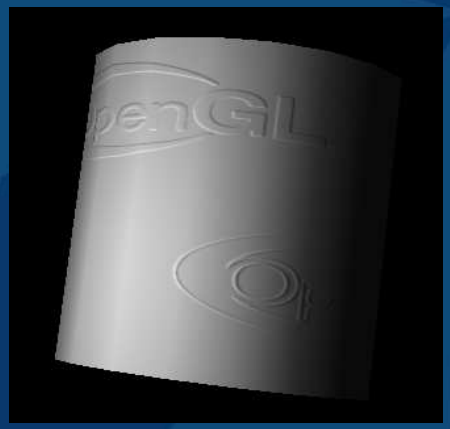
SAN ANTONIO SIGGRAPH 2002

Improved appearance

- Build effects
- Per-frame or per-object
- Still interactive



[Diefenbach97]



[Peercy97]

[Cabral99]

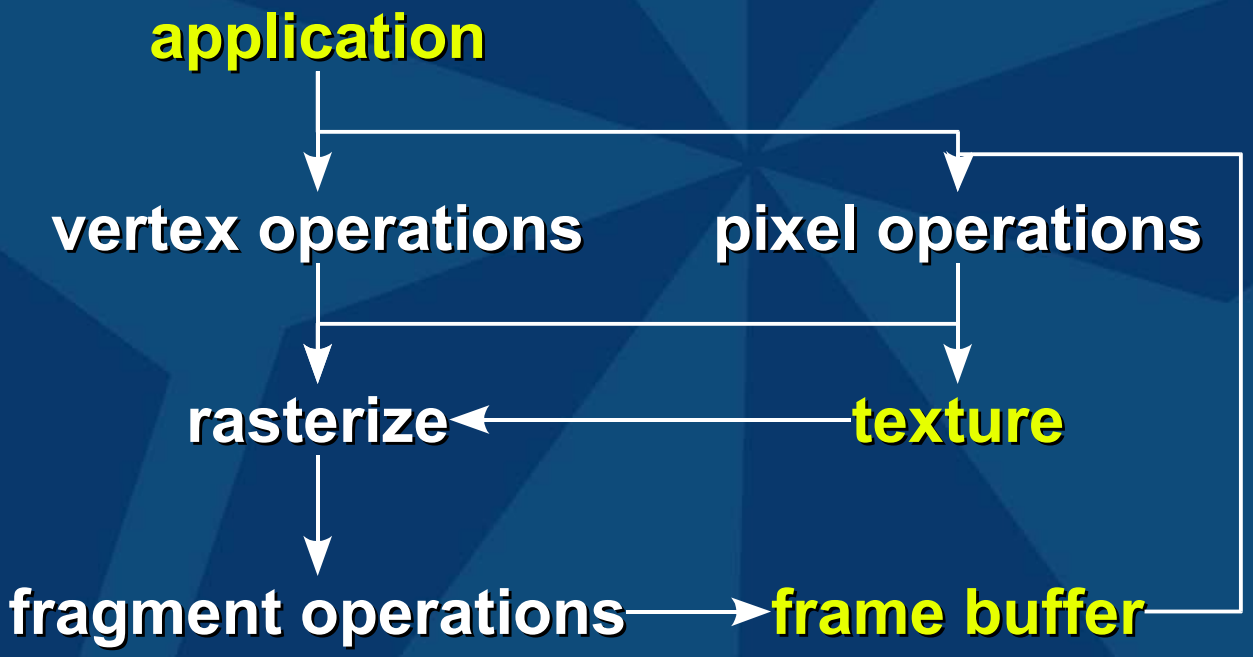
[Kautz99]

What's in a Pass?

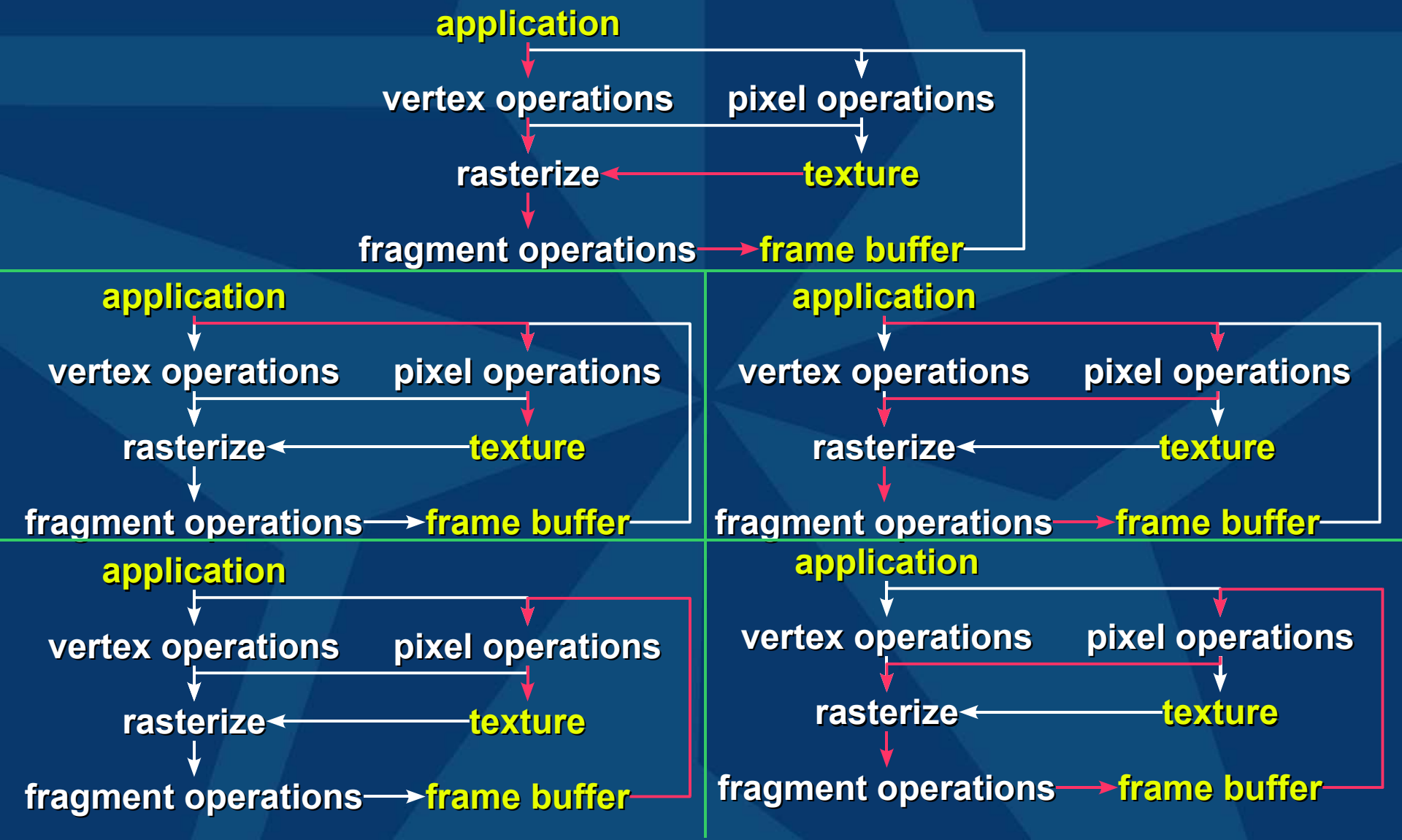
SAN ANTONIO SIGGRAPH 2002

Graphics hardware

- (as seen through OpenGL)



Rendering Passes



Multi-Pass = SIMD

Single Instruction, Multiple Data

Classic SIMD

- Thousands/millions of processors
- Thinking Machines, PixelFlow, …
- Not small-scale SIMD (MMX, etc.)

Shading languages use SIMD model

- Describe shading for one point
- Apply for every point on surface

Multi-Pass = SIMD

General SIMD	OpenGL
Shared Control	Application
Processor Array	Pixel Array
Per-PE ALU	Fragment ops
Per-PE Memory	FB / Texture
Per-PE Conditionals	Alpha / Stencil

What's it Mean?

We can create a compiler

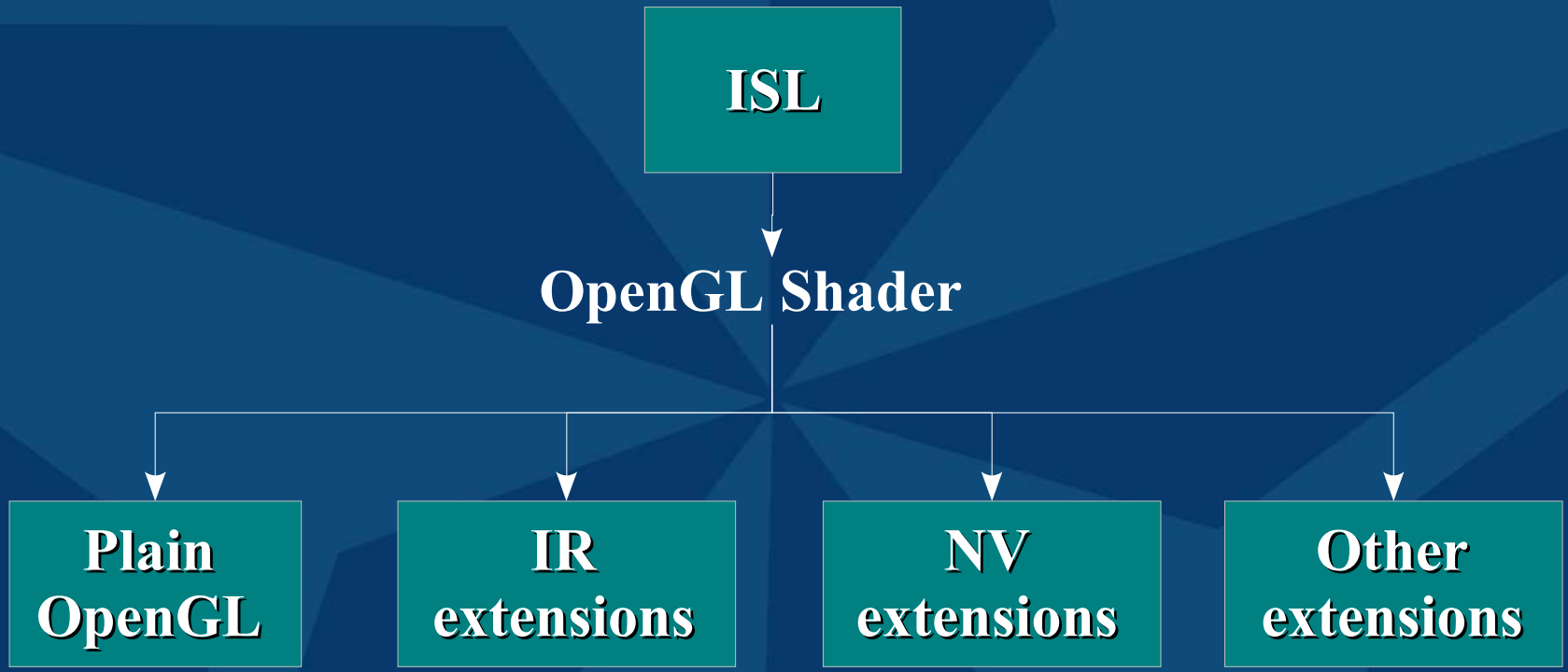
- High-level language in
- OpenGL out

Isn't That Slow?

No!

- Like drawing a few extra objects
- Optimize to compress passes
- Target hardware extensions

OpenGL Shader



About ISL

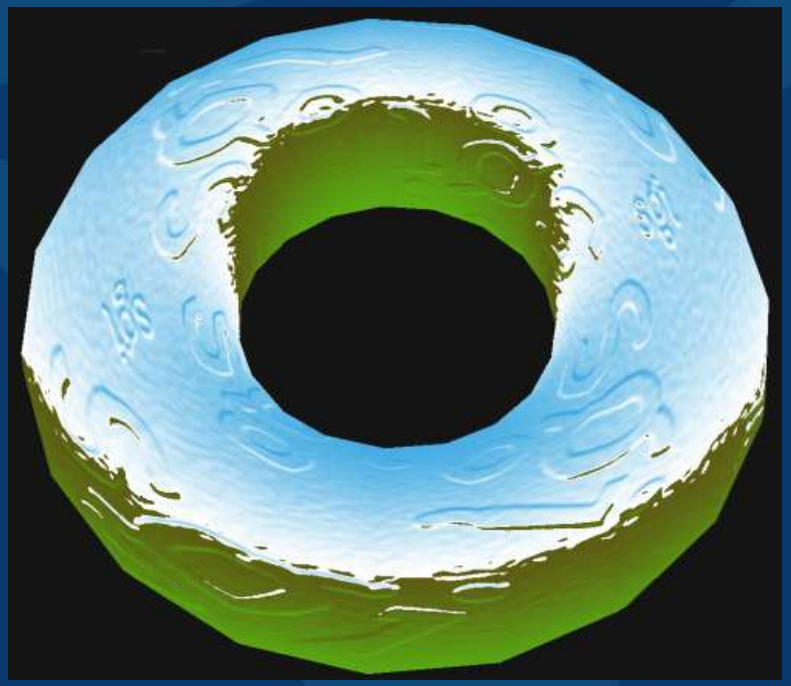
Things exposed in ISL

- Pass count: passes <= statements
  - Optimize to fewer
- Range: clamped 0 – 1
- Texturing limits
  - No per-pixel computed texture coordinates
  - Can use per-vertex texture coordinates

Example 1

Shiny Bump Map

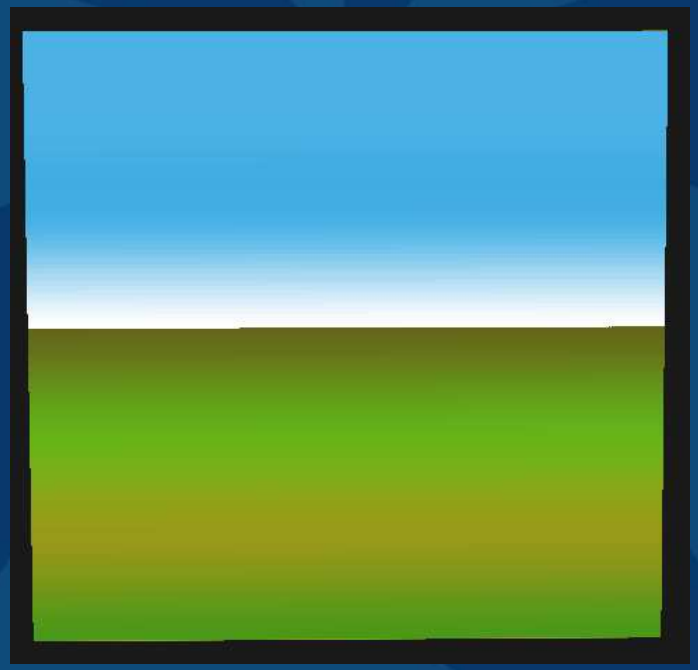
- Dependent texturing?
- ISL lookup function



Lookup-base Environment

ISL's run anywhere philosophy

- 1D environment
- Internally, same as texture lookup



Procedural Environment

Can build with ISL code

```
repeat(h) {
   // Fresnel component
   uniform float f = fresnel(2*i/(n-1) - 1, refract);

   // color spline for ground
   groundsky[i] = spline(i/(h-1),{
      color(.3,.6,.1,f),
      color(.3,.6,.1,f),
      color(.6,.6,.1,f),
      color(.4,.7,.1,f),
      color(.4,.4,.1,f),
      color(.3,.3,.1,f)});
   i = i+1;
}
```

Procedural Environment

And the sky...

```
repeat(h) {
   // Fresnel component
   uniform float f = fresnel(2*i/(n-1) - 1, refract);

   // color spline for ground
   groundsky[i] = spline((i-h)/(h-1),{
      color(1.,1.,1.,f),
      color(1.,1.,1.,f),
      color(.3,.7,.9,f),
      color(.3,.7,.9,f),
      color(.3,.7,.9,f),
      color(.3,.7,.9,f)});
   i = i+1;
}
```

Bump Map

Makes smooth surface appear bumpy

Several choices

- Evaluate bump math per-fragment
- “Embossed” bumps
- Normal map

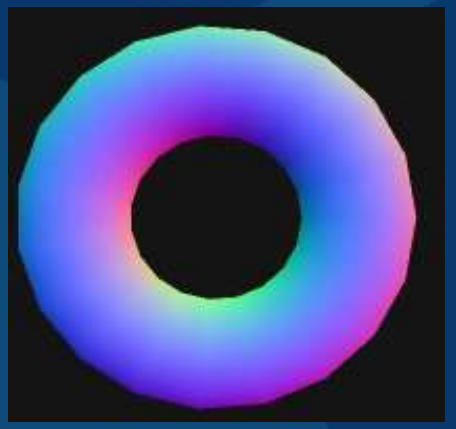
Normal Map

Normal range –1 to 1

Color range 0 to 1

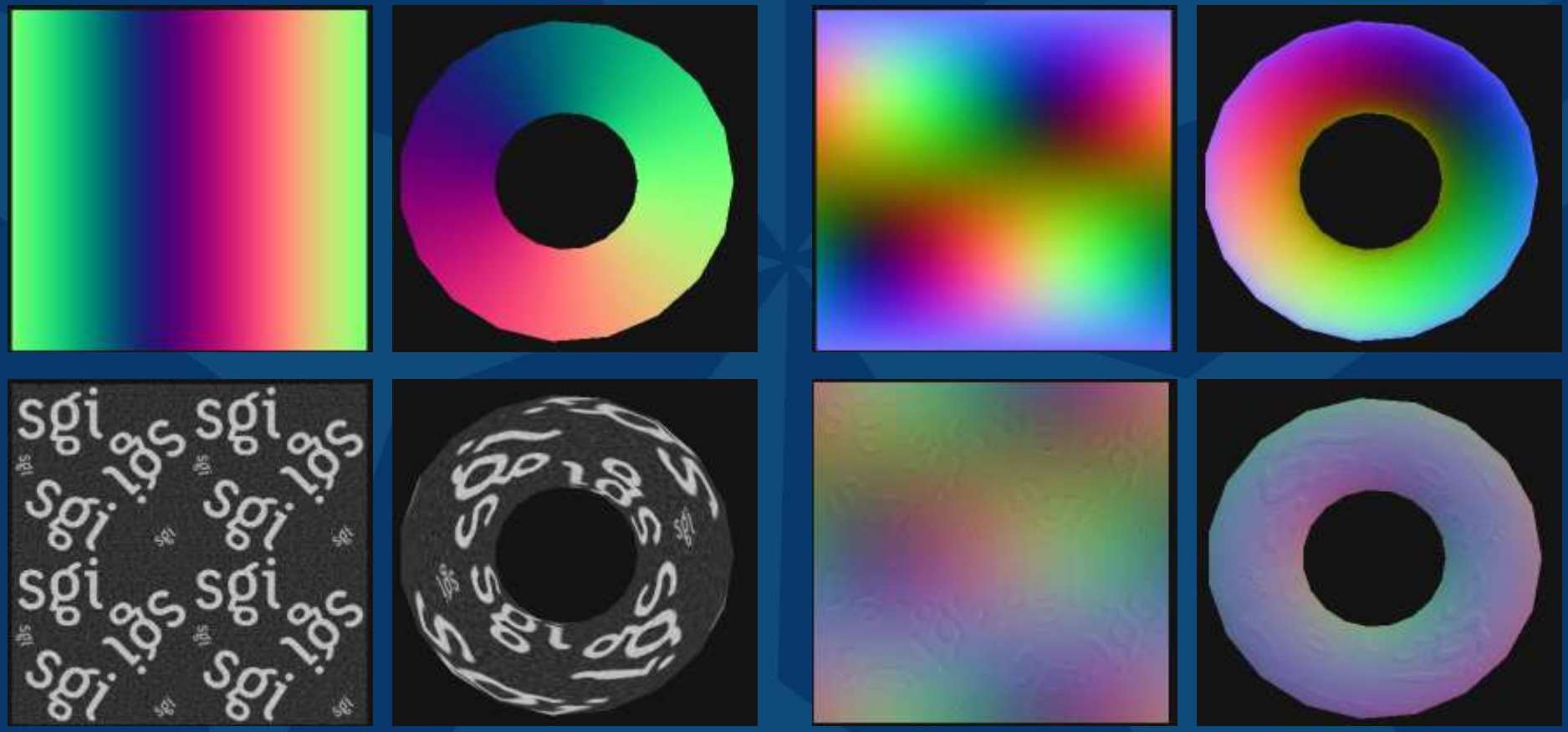
Normal map is just a texture

- $R = .5\ N_X + .5$
- $G = .5\ N_Y + .5$
- $B = .5\ N_Z + .5$



Bumped Normal Map

Perturb in tangent directions + renormalize



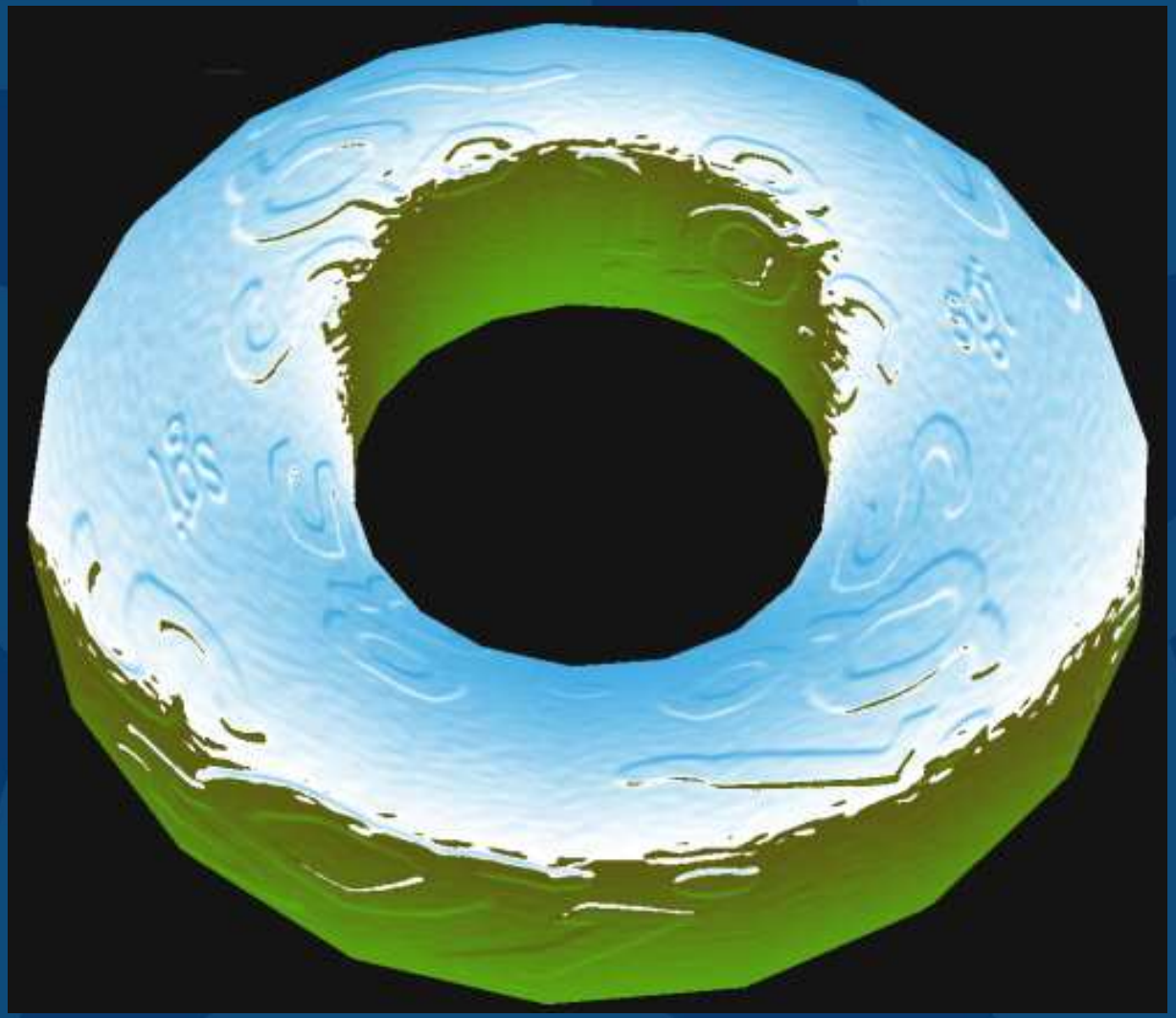
Scale & Transform Normal


```
// rescale normal vectors from 0..1 to -1..1 and back
uniform matrix nScale = translate(-.5,-.5,-.5)
                        *scale(2,2,2);
uniform matrix nUnscale = scale(.5,.5,.5)
                        *translate(.5,.5,.5);

// transform -1..1 normal from object to world space
parameter matrix nm = inverse(affine(shadermatrix));

// set rgb to y (vertical) component and alpha to z
// so one lookup can do both environment map and Fresnel
uniform matrix gggb = matrix(0,  0,  0,  0,
                             1,  1,  1,  0,
                             0,  0,  0,  1,
                             0,  0,  0,  0);
```

Actual shading code


```
FB = texture(nmap);
FB = transform(nScale * nm * nUnscale * gggb);
FB = lookup(groundsky);
```

Demo

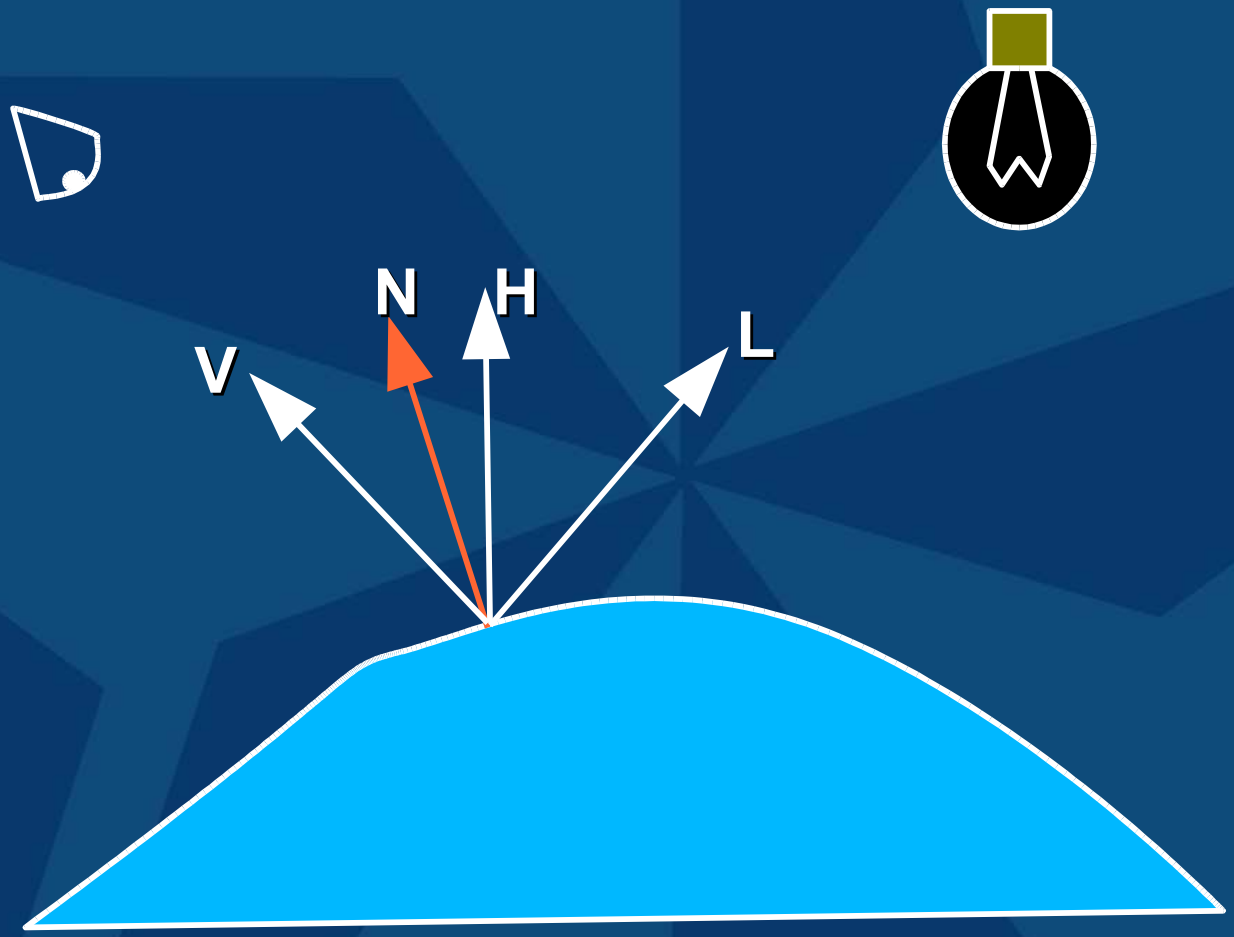


Example 2

SAN ANTONIO SIGGRAPH 2002

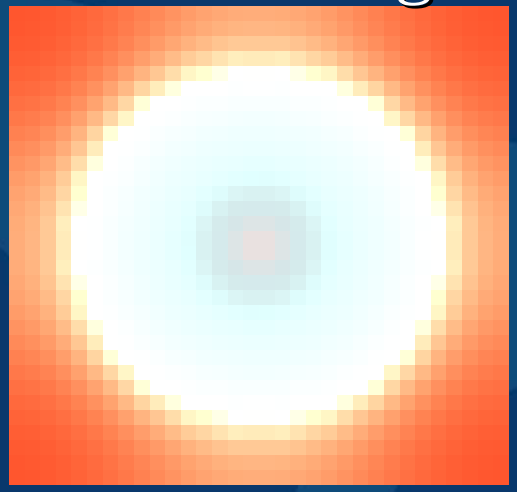
Homomorphic BRDF Factorization

Factorization

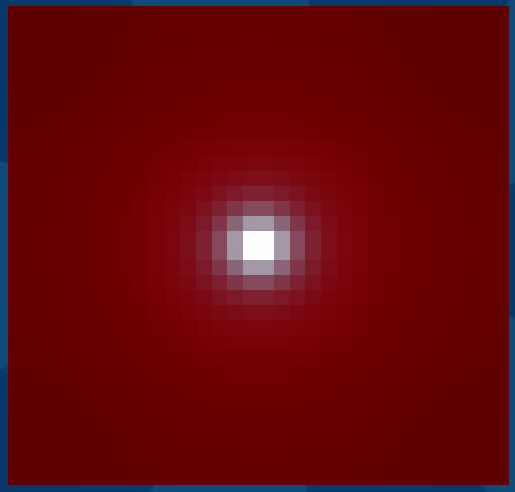


Factored Textures

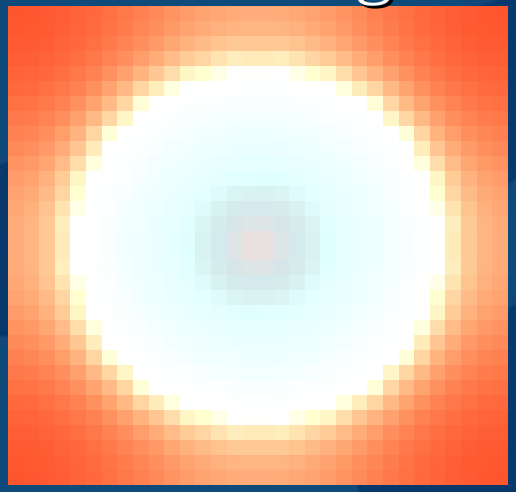
L
Shadowing



N
Microfacet



V
Masking



The Shader


```
surface BRDF(
    uniform string brdfP = "brdf p.rgb";
    uniform string brdfQ = "brdf q.rgb";
    uniform color brdfC = color(1,1,1,1))
{
    FB = diffuse();
    FB *= texture(brdfP, 1, 1);
    FB *= texture(brdfQ, 1, 2);
    FB *= texture(brdfP, 1, 3);
    FB *= brdfC;
}
```

Texture Coordinates


```
texture("texture", matrix, texcoord_set)
```

Passed to application draw callback

- Per-vertex application code
- Vertex programs

OpenGL Shader 3.0

- includes vertex program emulation

Demo

SAN ANTONIO SIGGRAPH 2002

Homomorphic BRDF Factorization

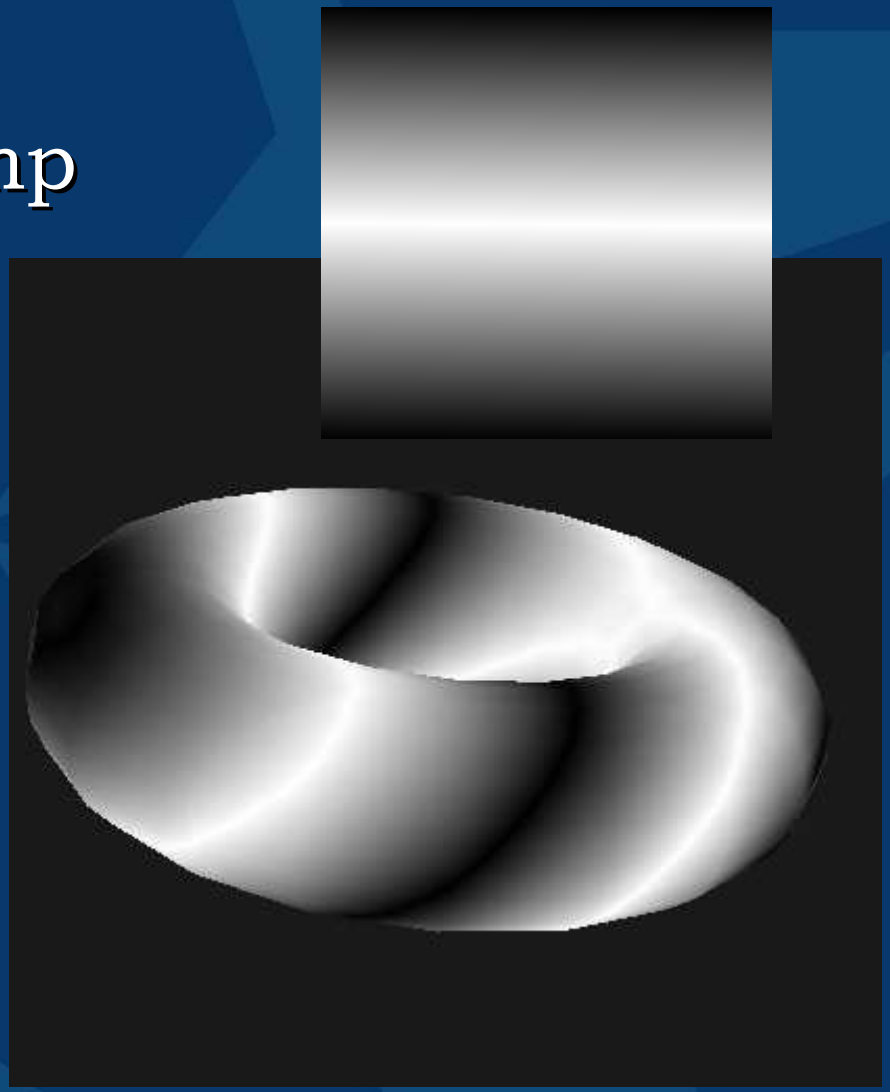
Example 3

Parameterized Wood

Need Some Bands

Start with a simple ramp

```
project("wave.bw",
    inverse(shadermatrix)*
    ringCenterXlate*
    ringAxisRotate*
     ringScale*
    textureCenterXlate);
```



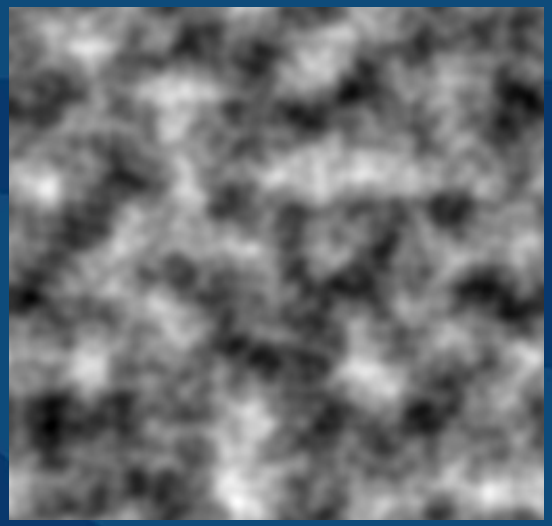
Turn into Rings

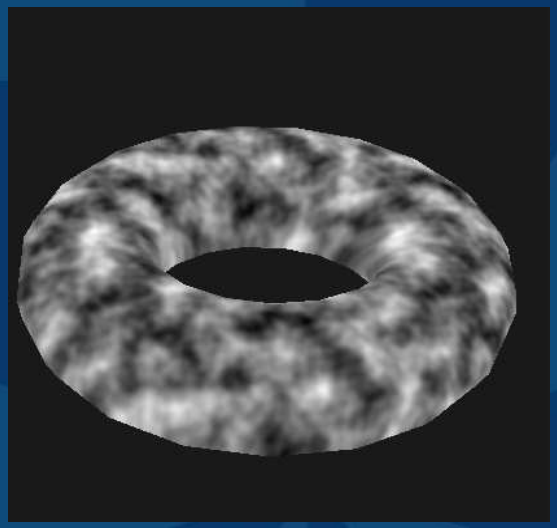
```
if (FB[0] < lightToDark)
   FB = darkWood;
else
   FB = lightWood;
```

Differing Specular

```
FB = diffuse();
varying color dif=FB;

FB = environment("highlight.bw");
varying color spec=FB;

if (FB[0] < lightToDark) {
    FB = darkWood;
    FB *= dif;
    varying color a = FB;

    FB = darkGloss;
    FB *= spec;
    FB += a;
}
```

Turbulent Rings

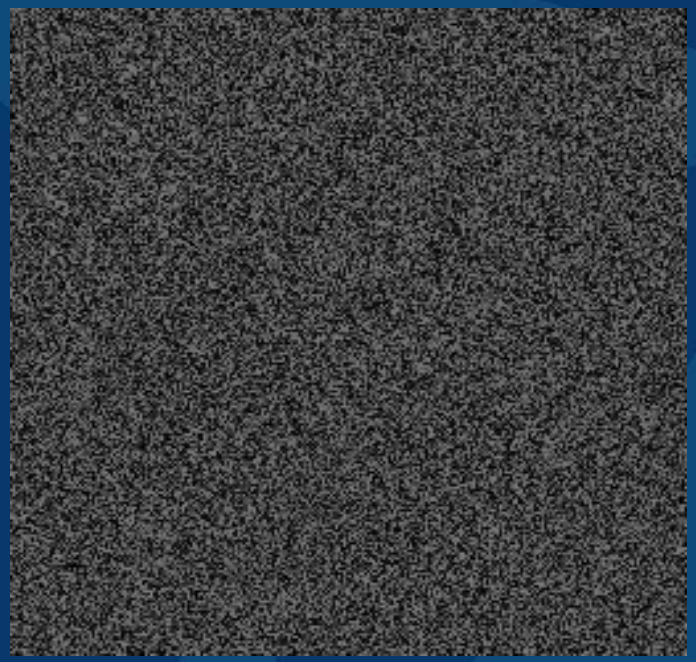




```
FB = project("turbulence.bw",...);
FB *= ringNoiseScale;
FB += project("wave.bw",...);
```

Fine Grain

```
FB = darkGloss;
FB.a = project("noise.bw",...);
FB = over(darkGrainGloss);
FB *= spec;
FB += a;
```



Demo

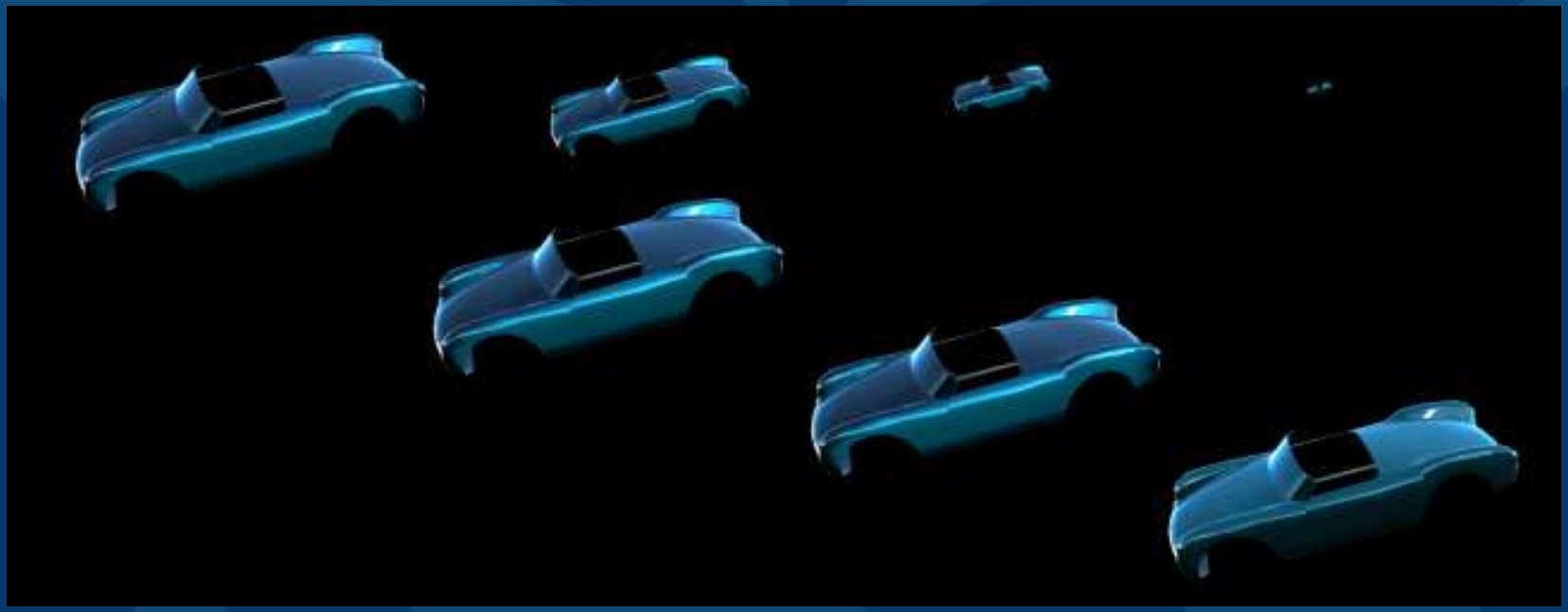
Bonus Example

Level-of-detail Shaders

Level-of-detail Shaders

Add conditionals to adjust complexity

- Distance
- Importance
- Time
- Available texture



Level-of-detail

Automatic

- Add conditionals
- Change “hardware mapping” rules in each branch

Semi-automatic

- Use LOD building blocks

Manual

- Add conditionals
- Hand-code levels

Demo