SAN ANTONIO

SIGGRAPH 2002

SAN ANTONIO SIGGRAPH 2002

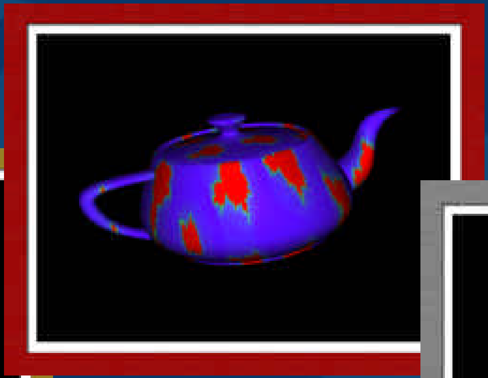
Course 17: State-of-the-Art in Hardware Rendering

The OpenGL Shading Language

Randi Rost

3Dlabs, Inc.

The OpenGL Shading Language



OpenGL 2.0

- "OpenGL 2.0" is a vision of where we think OpenGL should be headed
- An ARB GL2 working group has been formed and is meeting weekly
- Extensions have been proposed to support the OpenGL Shading Language
- 3Dlabs, ATI, and others are implementing the proposed extensions
- OpenGL 2.0 is more than shading stuff, but the shading stuff is most important

OpenGL Shading Language Goals

Define a vertex/fragment language that

- Is specific for OpenGL
- Exposes flexibility of (near) future hardware
- Provides hardware independence
- Is easy to use
- Will stand the test of time

OpenGL Shading Language Summary

- Integrated intimately with OpenGL 1.x
- Incremental replacement of fixed functions
- C-based, vector and matrix types
- Virtualizes (some) pipeline resources
- Same language for vertex and fragment
- Shaders are treated as objects
- Shaders are attached to a program object

Vertex Processor

A programmable unit that replaces:

- Vertex transformation
- Normal transformation, normalization, and rescaling
- Lighting
- Color material application
- Color clamping
- Texgen & texture coordinate transformation

Vertex Processor

Does NOT replace:

- Perspective projection and viewport mapping
- Frustrum and user clipping
- Backface culling
- Primitive assembly
- Two sided lighting selection
- Polygon mode & offset

Fragment processor

Replaces:

- Operations on interpolated values
- Texture access and application
- Fog
- Color sum
- Pixel zoom
- Scale and bias
- Color table lookup
- Convolution
- Color matrix

Fragment processor

Does NOT replace:

- Shading model
- Coverage
- Histogram
- Minmax
- Pixel packing and unpacking
- Stipple

Fragment Processor

Does NOT replace:

- Pixel ownership test
- Scissor test
- Alpha test, Stencil test, Depth test
- Blending
- Dithering
- Logical ops
- (But these could be done programmably)

Shader Management

Shaders and Programs

- OpenGL processor source code is a “shader”
- Shaders are defined as an array of strings
- A container for shader objects is a “program”

Shader Management

```
// Create a shader (shaderID != 0 if success)
shaderA = glCreateShaderObjectGL2( shaderType );
// Load source code
glLoadShaderGL2( shaderA, numStrings, strings );
glAppendShaderGL2( shaderA, string );

// Compile (status = TRUE if success)
status = glCompileShaderGL2( shaderA );
// Get information string from compile
infolog = glGetInfoLogGL2( shaderA );
```

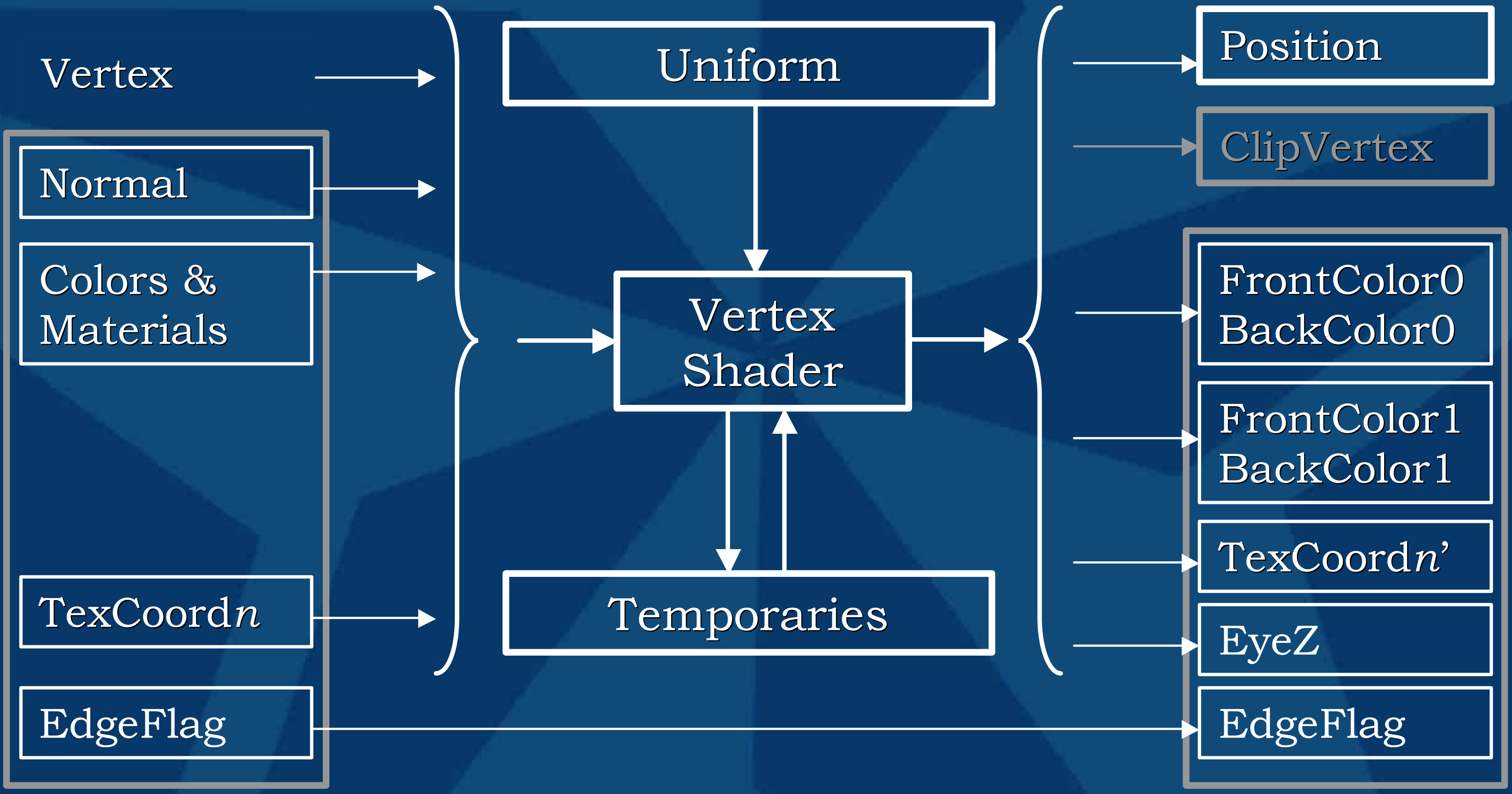
Shader Management

```
// Create a program object
programZ = glCreateProgramObjectGL2();
// Attach shader object(s)
status =
  glAttachShaderObjectGL2( programZ, shaderA );
status =
  glAttachShaderObjectGL2( programZ, shaderB );
// Link the program
status = glLinkProgramGL2( programZ );
```

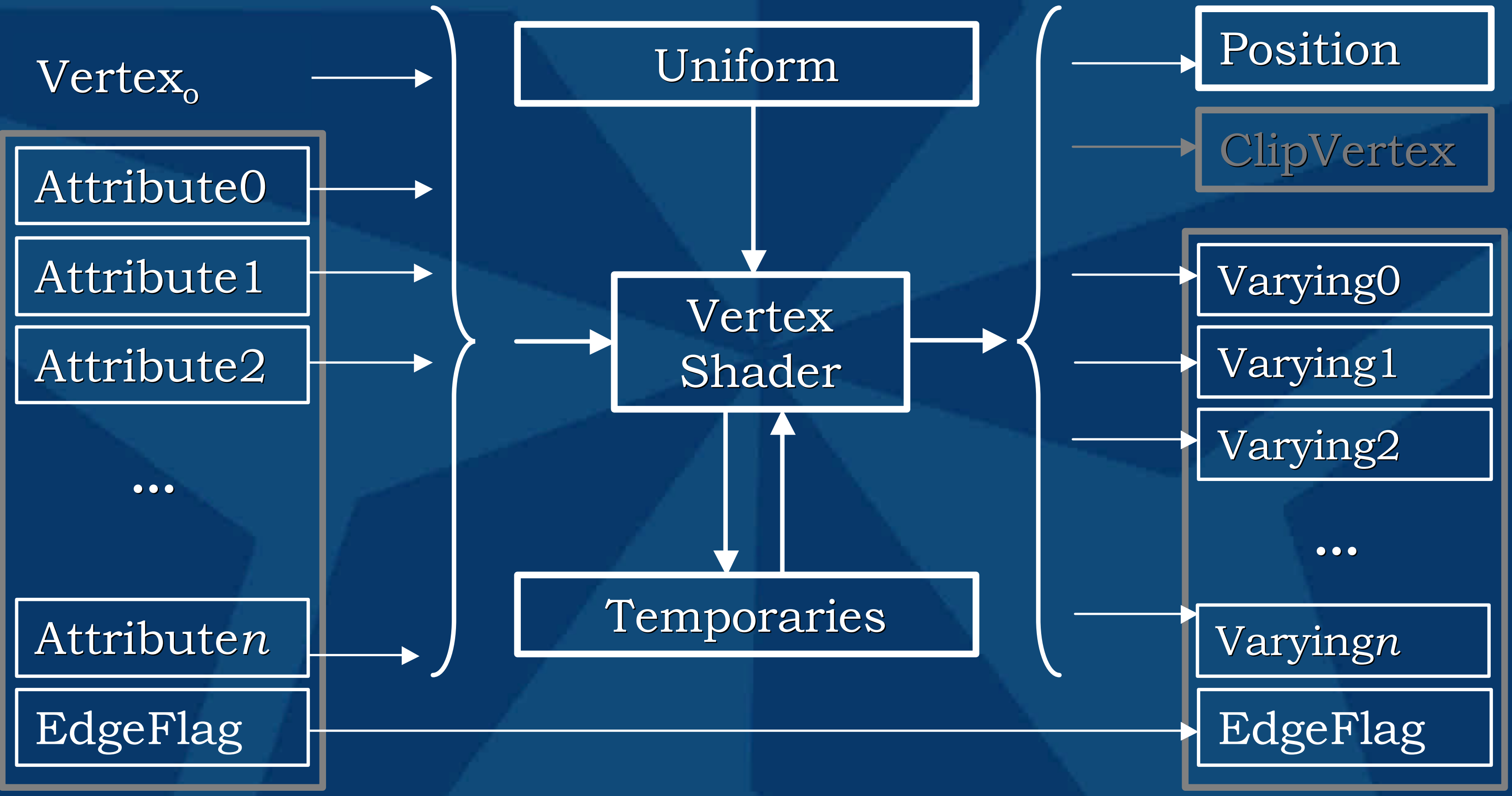
Shader Management

```
// get information on link
infolog = glGetInfoLogGL2( programZ );
glGetObjectParameterfvGL2( programZ,
              GL_SHADER_RELATIVE_SIZE_GL2, size );
// Make program current
status = glUseProgramObjectGL2( programZ );
```

Vertex Processor

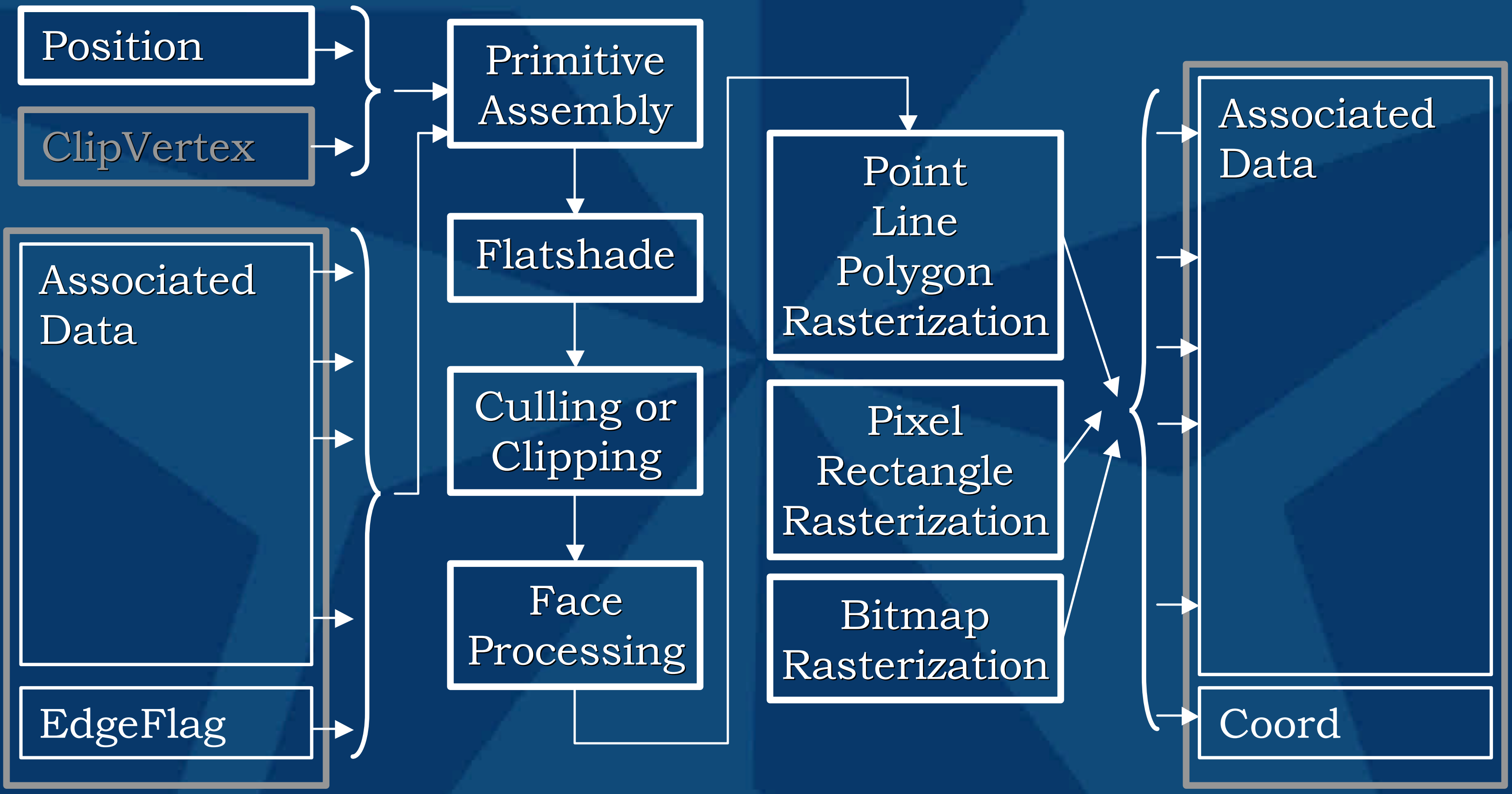


Vertex Processor



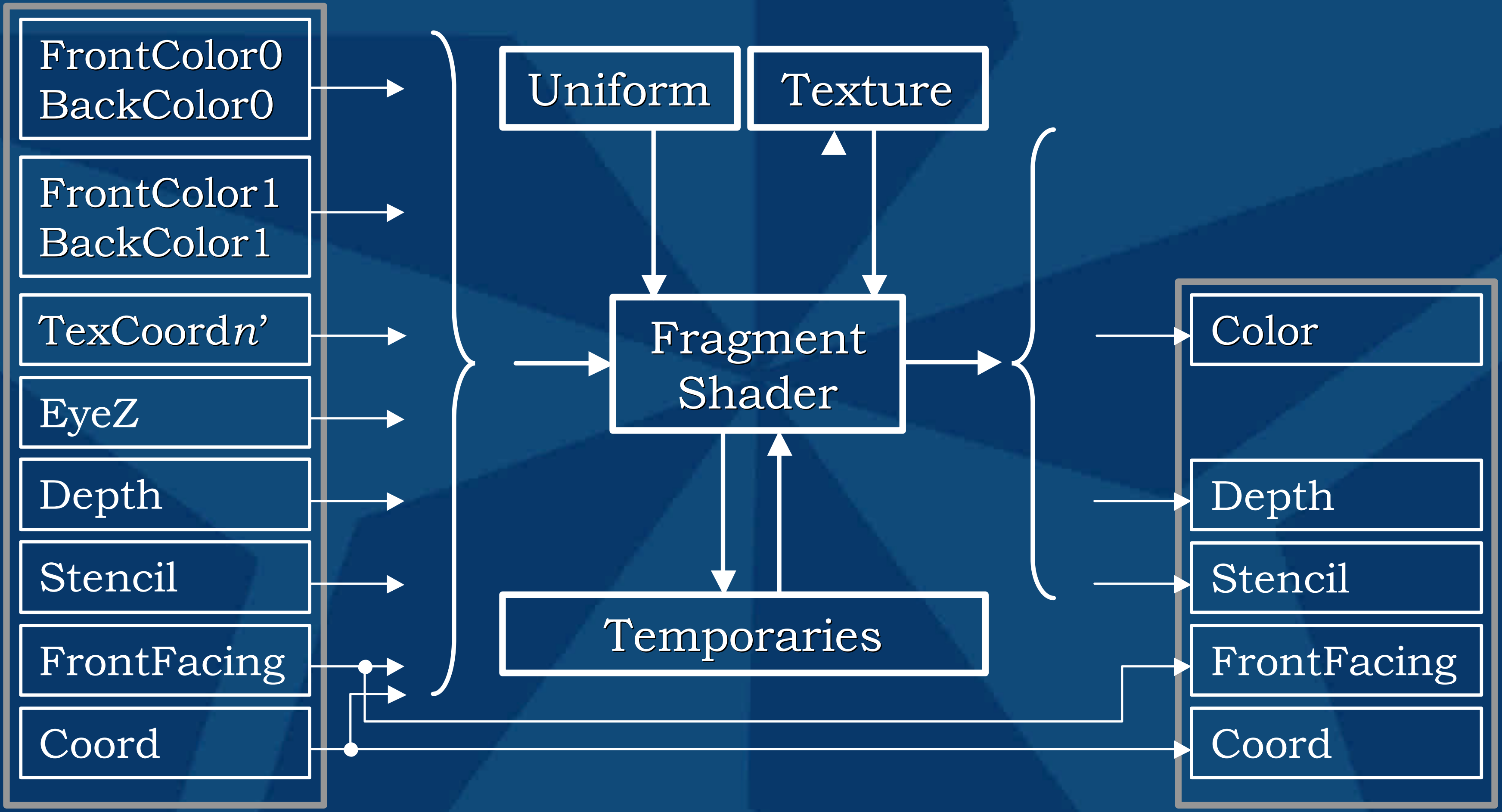
Rasterization

SAN ANTONIO SIGGRAPH 2002



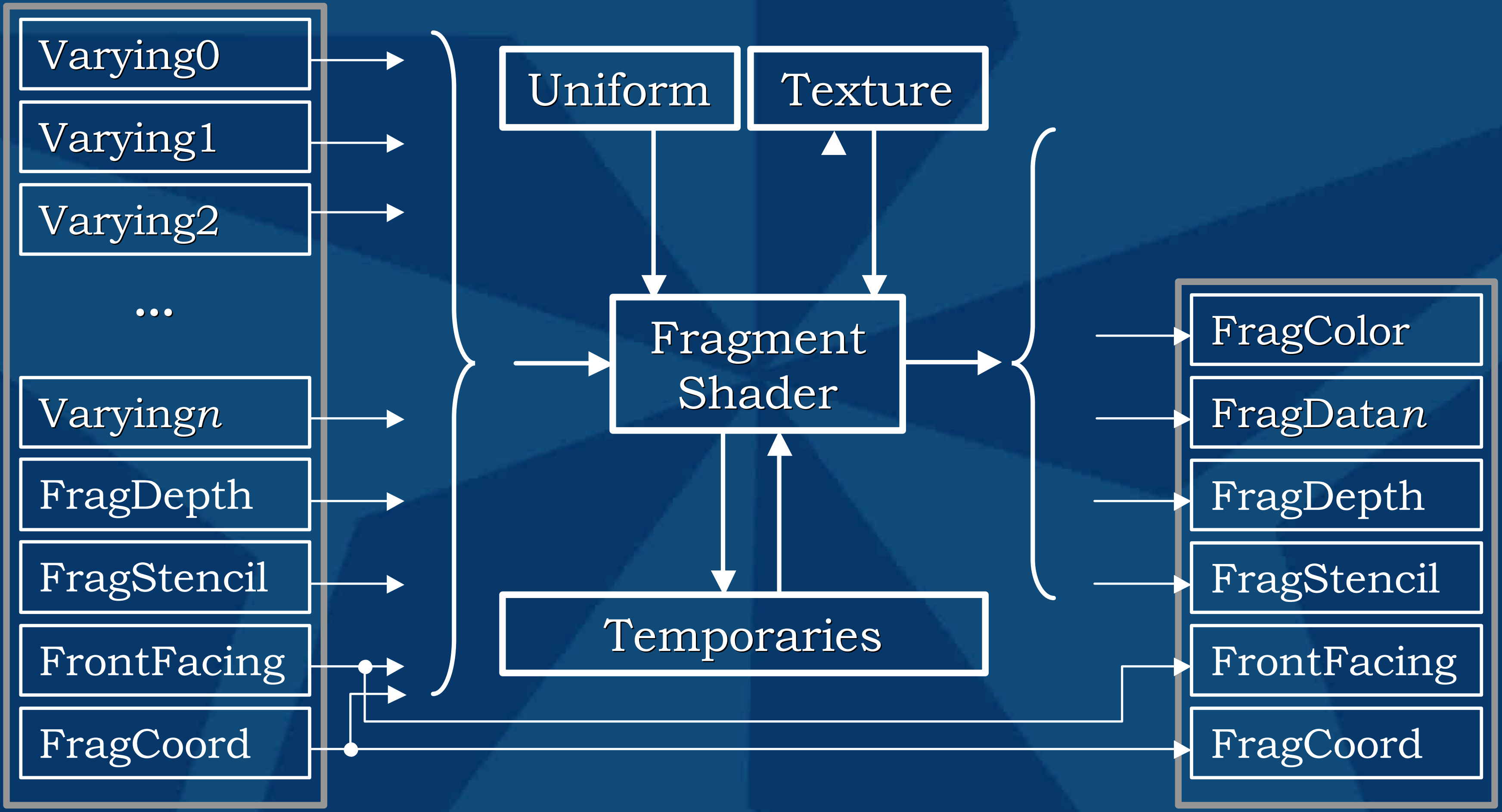
3Dlabs

Fragment Processor



Fragment Processor

SIGGRAPH 2002 San Antonio



3Dlabs

Data Types

- float
- vec2, vec3, vec4
- mat2, mat3, mat4
- int
- bool
  - As in C++, true or false
- arrays
  - [] to access array element

Type Qualifiers

- const
- attribute
- uniform
- varying

Expressions

- constants
- variables
- scalar/vector/matrix operations
- unary and binary ops: +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=, ++, --, &&, ||, ^^, <=, >=, ==, !=, <<, >>, ,, =, (), [], |, ., !, -, ,+, *, /, %, <, >, |, ^, &, ?

Expressions

- **component accessors for vectors and matrices**
  - {x,y,z,w},{r,g,b,a},{s,t,p,q},{0,1,2,3}
  - .rc (matrix element)
- **row and column accessors for matrices**
  - .r_ (matrix row)
  - ._c (matrix column)

Flow Control

- expression ? trueExpression : falseExpression
- if, if-else
- for, while, do-while
- return, break, continue, kill
- function calls

User-defined Functions

- Pass by reference
- Output keyword
- Function overloading
- One return value (can be any type)
- Scope rules same as C

SAN ANTONIO SIGGRAPH 2002

Built-in Functions

- sin, cos, tan, asin, etc…
- pow, log2, exp2, sqrt, inversesqrt
- abs, sign, floor, ceil, frac, mod
- min, max, clamp
- mix, step, smoothstep
- length, distance, normalize, dot, cross
- texture, lod, dPdx, dPdy, fwidth, kill, noise

3Dlabs

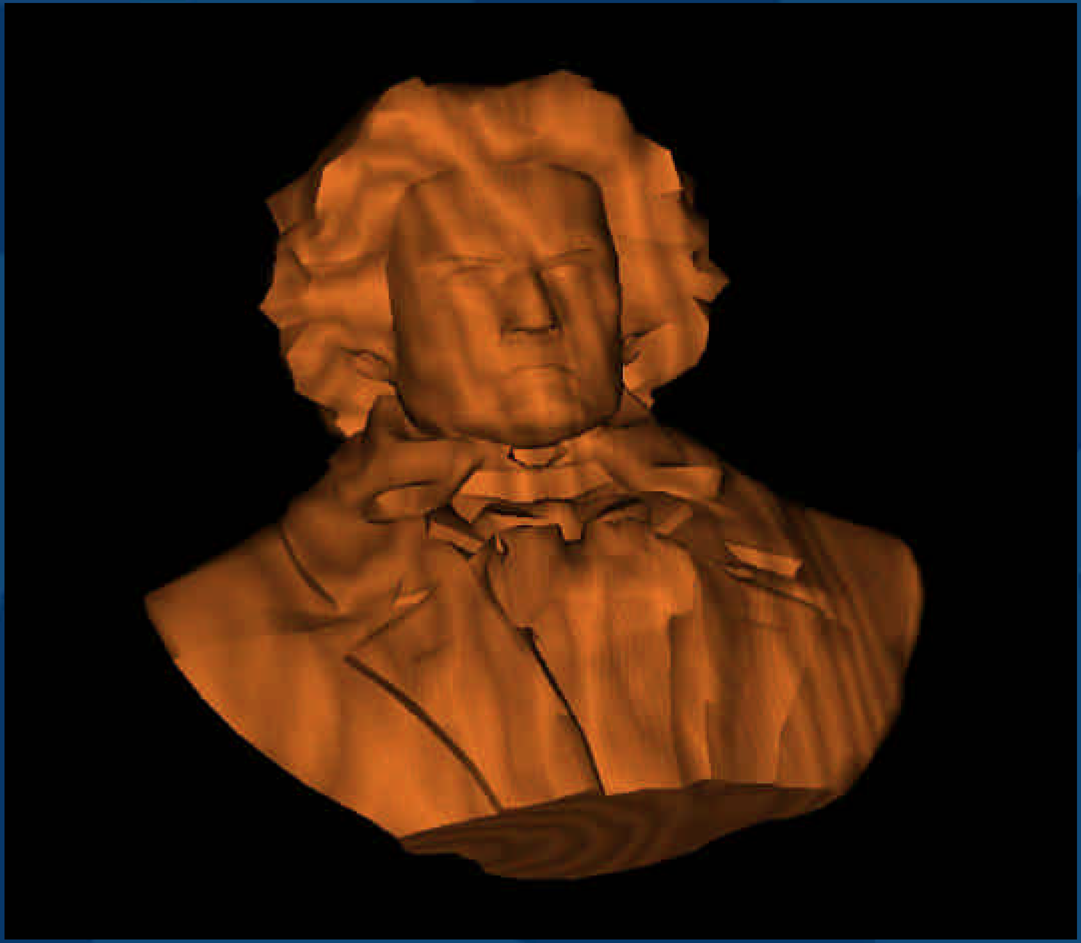
Preprocessor

- #define
- #ifdef … #else … #elif … #endif
- #pragma

About These Examples

- OpenGL Shading Language implementations are just emerging
- We describe implemented, working shaders rather than theoretical ones
- We do not claim these are the “best” examples for each category
- Focus is on illustrating the language rather than describing the algorithm
- Demo hardware is the 3Dlabs Wildcat VP870

Example 1: Wood Shader



Wood Shader Overview

- Part of ogl2demo developed by 3Dlabs
- Uses a tileable, 4-octave noise function that is precomputed and stored in a 3D texture
- Vertex shader is trivial
- Fragment shader indexes the 3D texture and uses the result to procedurally generate a 3D wood texture

“Theory” of Wood

- The wood is composed of light and dark areas alternating in concentric cylinders surrounding a central axis.
- Noise is added to warp the cylinders to create a more natural-looking pattern
- The center of the “tree” is taken to be the y-axis
- Throughout the wood there is a high-frequency grain pattern

Wood Vertex Shader

```
varying float lightIntensity;
varying vec3 Position;

uniform vec3 LightPosition;
uniform float Scale;

void main(void)
{
        vec4 pos = gl_ModelViewMatrix * gl_Vertex;
        Position = vec3(gl_Vertex) * Scale;
        vec3 tnorm = normalize(gl_NormalMatrix * gl_Normal);
        lightIntensity = abs(dot(normalize(LightPosition –
                             vec3(pos)), tnorm) * 1.5);
        gl_Position = gl_ModelViewProjectionMatrix *
                          gl_Vertex;
}
```

Wood Fragment Shader (1 of 3)

```
varying float lightIntensity;
varying vec3 Position;

uniform vec3  LightWood;
uniform vec3  DarkWood;
uniform float RingNoise;
uniform float RingFreq;
uniform float LightGrains;
uniform float DarkGrains;
uniform float GrainThreshold;
uniform vec3  NoiseScale;
uniform float Noisiness;
uniform float GrainScale;
```

Wood Fragment Shader (2 of 3)

SAN ANTONIO SIGGRAPH 2002

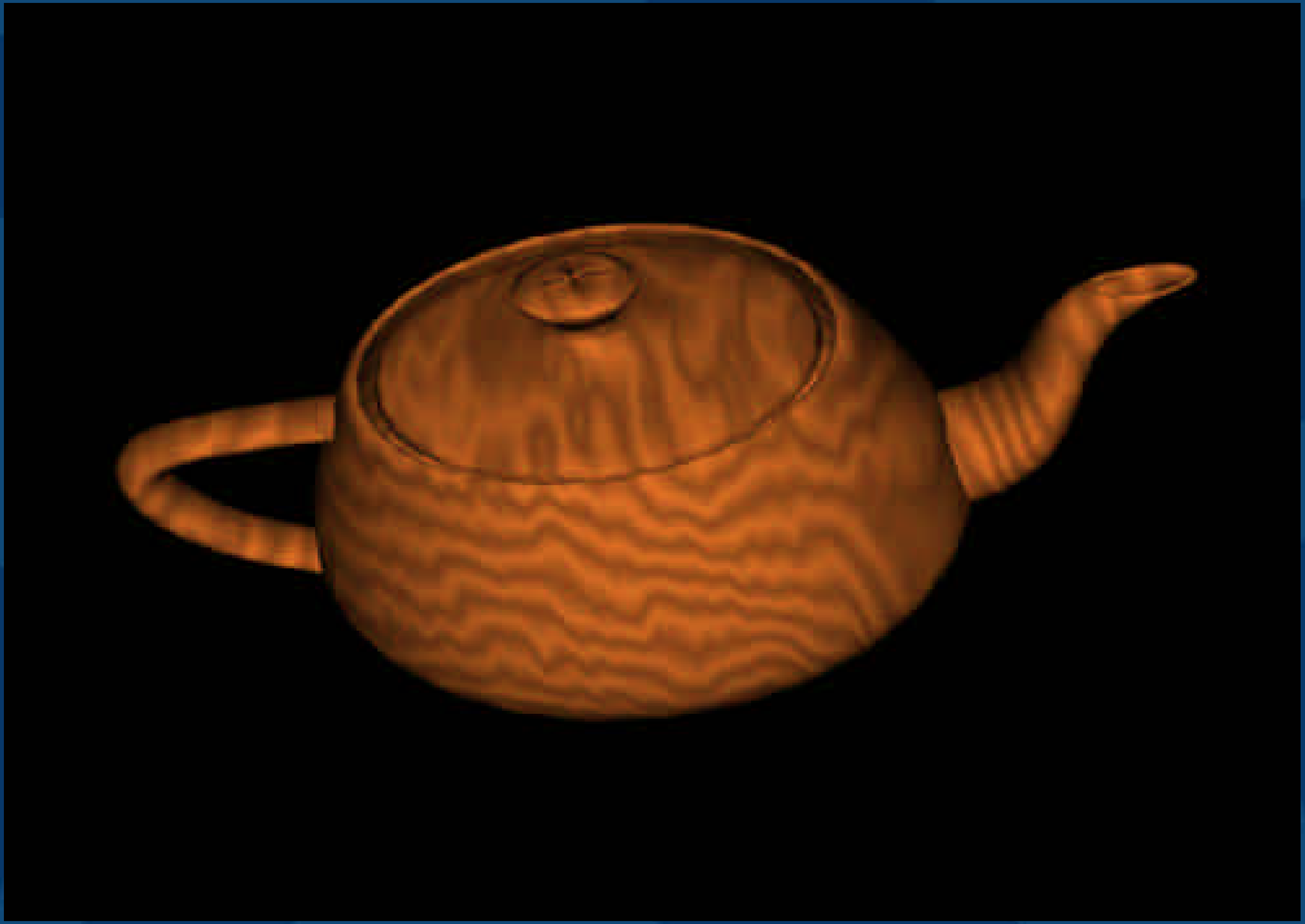
```
void main (void)
{
    vec3 location;
    vec3 noisevec;
    vec3 color;
    float dist;
    float r;

    noisevec = texture3(6, Position * NoiseScale) * Noisiness;

    location = vec3 (Position + noisevec);

    dist = sqrt(location.x * location.x +
                location.z * location.z) * RingFreq;

    r = fract(dist + noisevec.0 + noisevec.1 + noisevec.2) * 2.0;
```

3Dlabs

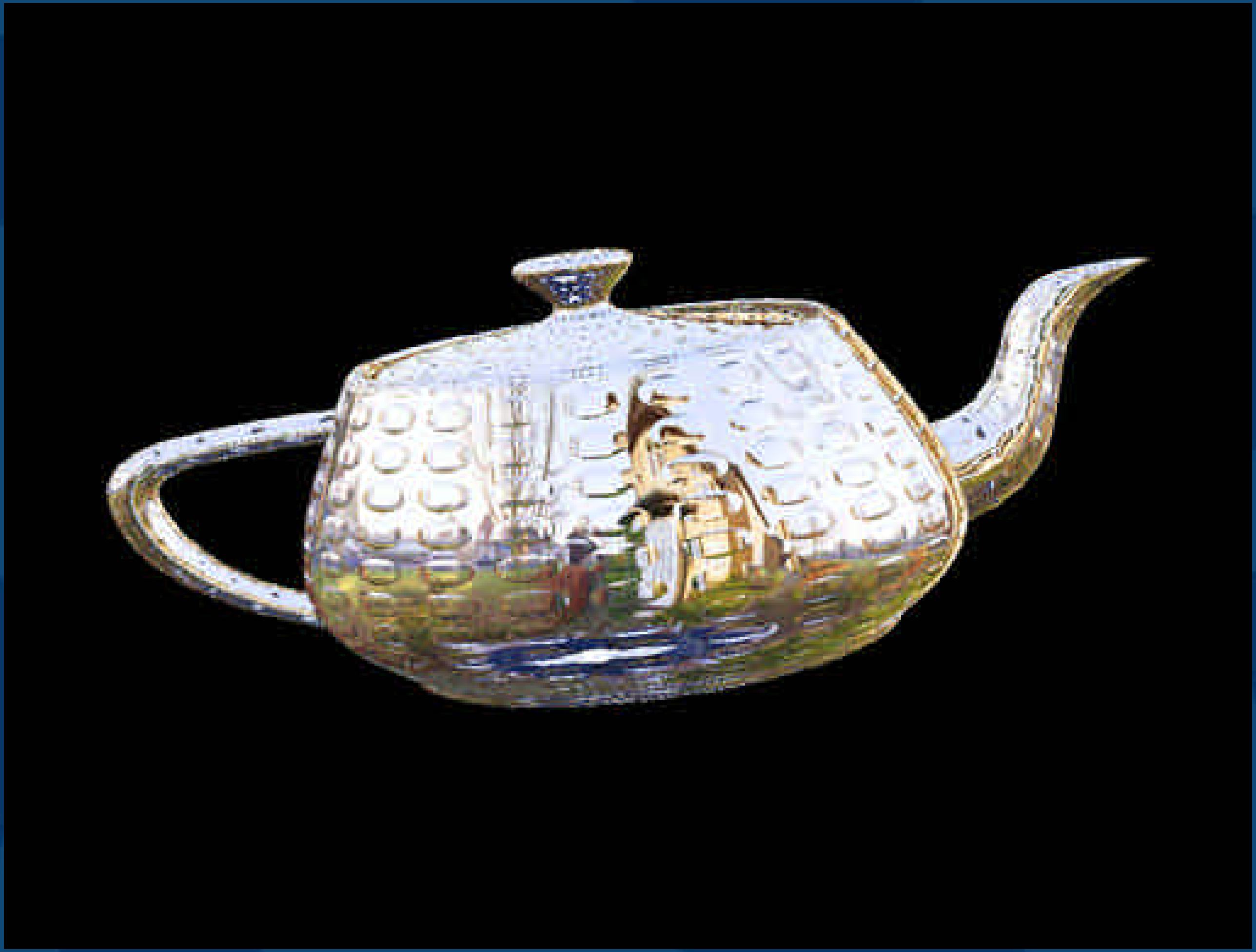
Wood Fragment Shader (3 of 3)

```
    if (r > 1.0)
        r = 2.0 - r;

    color = mix(LightWood, DarkWood, r);

    r = fract((Position.x + Position.z) * GrainScale + 0.5);
    noisevec.2 *= r;
    if (r < GrainThreshold)
       color += LightWood * LightGrains * noisevec.2;
    else
        color -= LightWood * DarkGrains * noisevec.2;

    color = clamp(color * lightIntensity, 0.0, 1.0);

    gl_FragColor = vec4(color, 1.0);
}
```

Wood Shader Demo



Example 2: Bumpy/Shiny Shader



Bumpy/Shiny Shader Overview

- Part of ogl2demo developed by 3Dlabs
- Vertex shader is trivial
- Fragment shader computes procedural bumps and the final index into the environment map
- Environment maps courtesy of Jerome Dewhurst, jerome@photographica.co.uk, http://www.photographica.co.uk

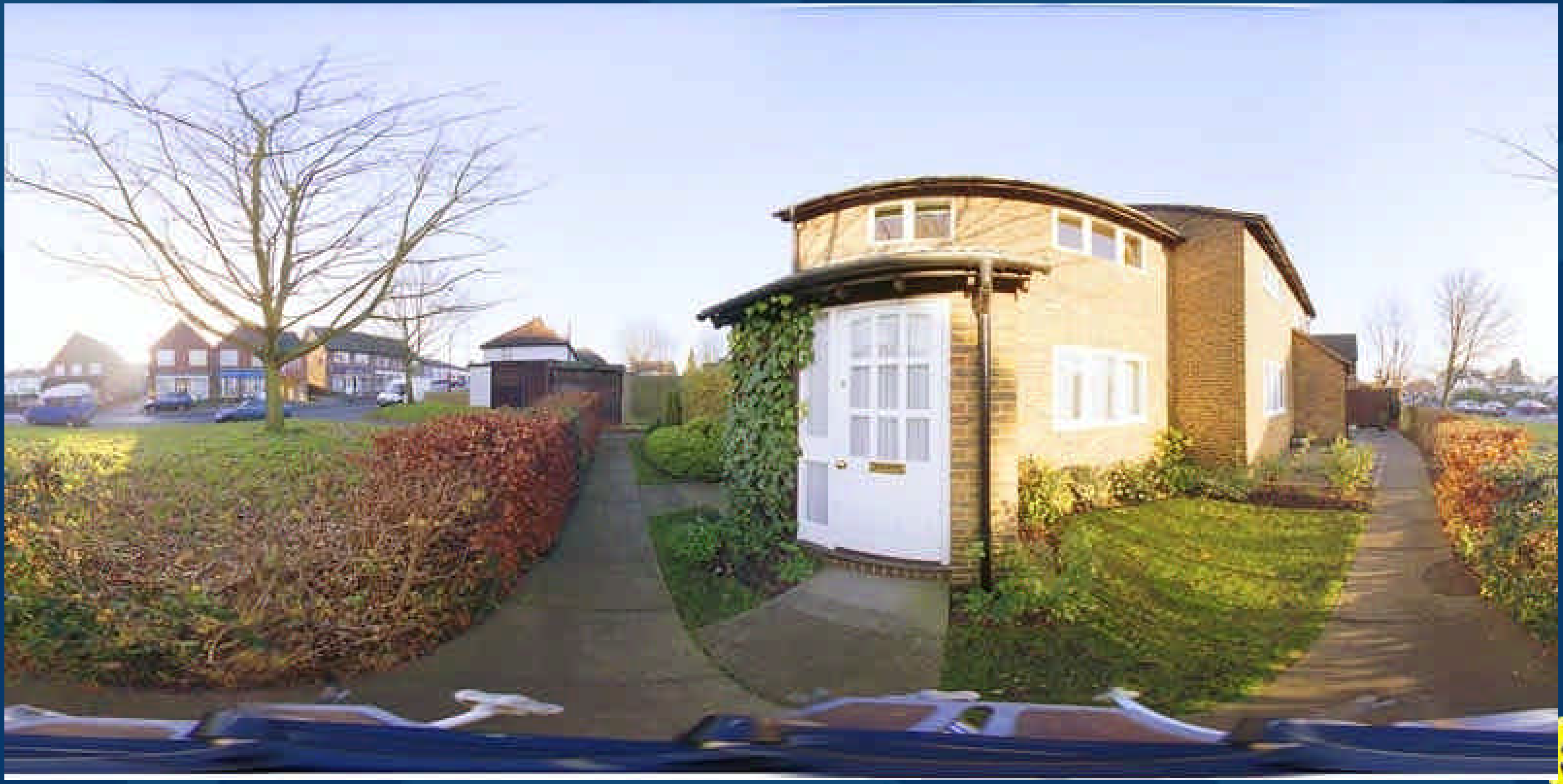
Bumpy/Shiny Algorithm

- Compute normalized eye vector and normalized normal in vertex shader
- Compute reflection direction R in fragment shader
- Use R to obtain altitude and azimuth angles at each fragment
- Use altitude as environment map vertical index, azimuth as horizontal index
- Use a cheesy hack to perturb this lookup if the fragment is within a “bump”

Environment Map

SAN ANTONIO SIGGRAPH 2002

- A single 2D texture containing an equirectangular image (360º horiz, 180º vert)



3Dlabs

Bumpy/Shiny Vertex Shader

```
varying vec3 Normal;
varying vec3 EyeDir;

void main(void)
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    gl_TexCoord0 = gl_MultiTexCoord0;
    Normal = normalize(gl_NormalMatrix * gl_Normal);
    EyeDir = normalize(vec3(gl_ModelViewMatrix * gl_Vertex));
}
```

Bumpy/Shiny Fragment Shader (1 of 3)

```
uniform vec3 Xunitvec;
uniform vec3 Yunitvec;

varying vec3 Normal;
varying vec3 EyeDir;

const float Density = 6.0;
const float Coeff1 = 5.0;
float Size = 0.15;

void main (void)
{
    // Compute reflection vector. Should really normalize
    // EyeDir and Normal, but this is OK for some objects
    vec3 reflectDir = reflect(EyeDir, Normal);
```

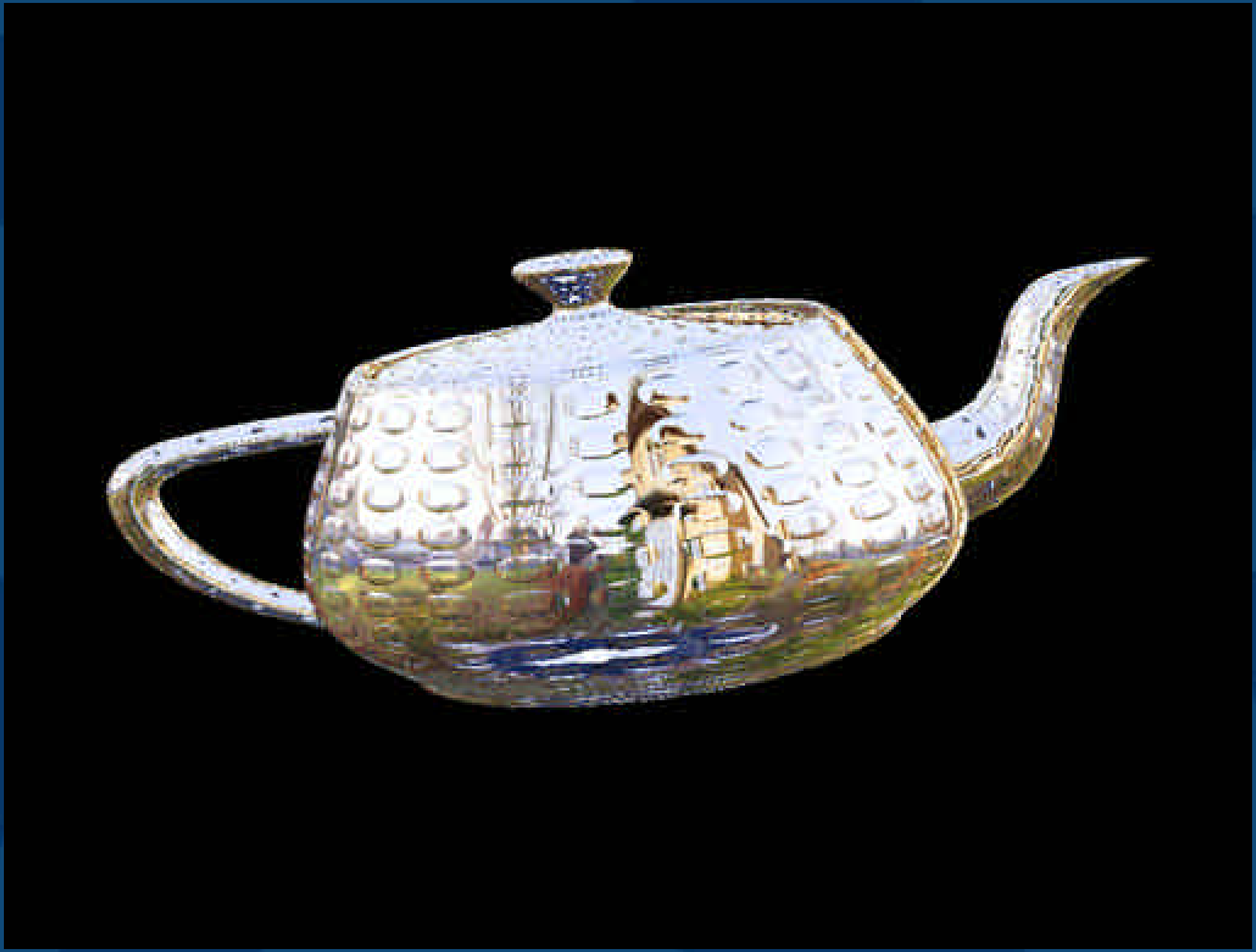
Bumpy/Shiny Fragment Shader (2 of 3)

```
// Compute cosine of altitude and azimuth angles
vec2 index;
index.1 = dot(normalize(reflectDir), Yunitvec);
reflectDir.y = 0.0;
index.0 = dot(normalize(reflectDir), Xunitvec);

// Compute bump locations
vec2 c = Density * vec2(gl_TexCoord0);
vec2 offset = fract(c) - vec2(0.5);

float d = offset.x * offset.x + offset.y * offset.y;
if (d >= Size)
    offset = vec2(0.0);
```

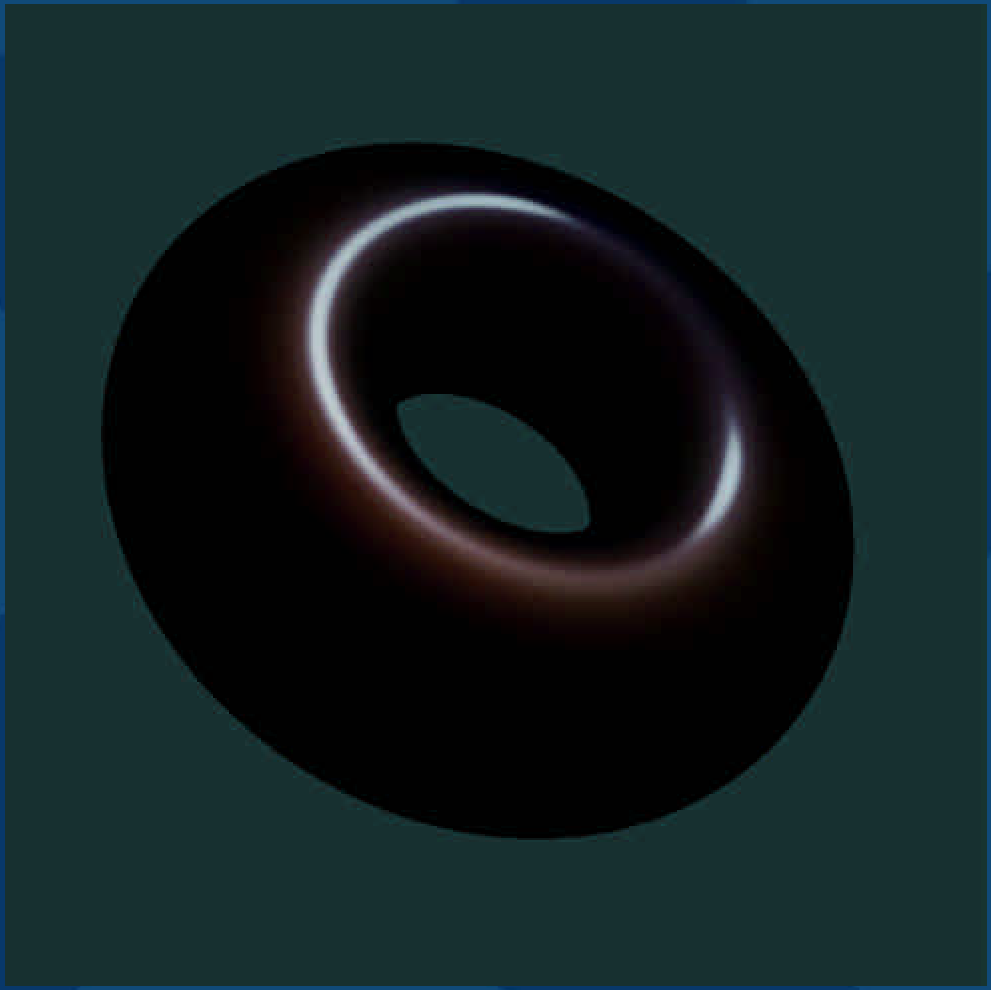
Wood Fragment Shader (3 of 3)

```
    // Translate index to (0.5, 0.5) and make offset non-linear
    index = (index + 1.0) * 0.5 –
            Coeff1 * offset * offset * offset;

    // Do a lookup into the environment map
    vec3 envColor = texture3(4, index);

    gl_FragColor = vec4(envColor, 1.0);
}
```

Bumpy/Shiny Shader Demo



Example 3: BRDF Shader



Rendering BRDF Data With PTMs

- BRDF data is from Cornell
- Samples are automotive paints from Ford
- BRDF data was encoded in a polynomial texture map (PTM) by Hewlett-Packard
- Shaders and textures used were provided courtesy of Hewlett-Packard
- Example shown is mystique lacquer, which changes color with viewing angle
- Realism is the goal

Creating a BRDF PTM

- **For each texel in the PTM:**
  - **Sample the BRDF data varying view angle and light position to obtain the data points to be represented**
  - **Perform a biquadric least squares curve fit**
  - **Store the coefficient values of the resulting polynomial in the texture(s)**

e.g., $f(u,v) = Au^2 + Bv^2 + Cuv + Du + Ev + F$

This process is partly black magic…

SAN ANTONIO SIGGRAPH 2002

Application Setup

- At each vertex, application passes vertex, normal, tangent, and binormal
- For RGB PTMs and BRDF RGB PTMs, there are separate coefficient textures for each of R, G, and B
- Application enables red channel, binds red coefficient textures, renders geometry
- Repeat for green and blue channels using same shaders

3Dlabs

BRDF PTM Vertex Shader Overview

- Infinite viewer, local light source
- Goal of vertex shader is to produce
  - 2D parameterization of light position
  - Texture coordinate used to reference PTM
  - Texture coordinate used to reference light factor
- Half angle/difference vector formulation is used to achieve better quality
- Use user-defined attributes to pass tangent and binormal

BRDF PTM Vertex Shader (1 of 4)

```
// VP_PTM_3DLABS_HuHvDuDv_PosInf
uniform vec3 LightPos;
uniform vec3 EyeDir;

attribute vec3 Tangent;
attribute vec3 Binormal;

varying float LdotT;
varying float LdotB;
varying float LdotN;

void main(void)
{
        vec3 lightTemp;
        vec3 halfAngleTemp;
        vec3 tPrime;
        vec3 bPrime;
```

BRDF PTM
Vertex Shader (2 of 4)

```
// Transform vertex
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
lightTemp = normalize(LightPos - vec3(gl_Vertex));

// Calculate the Half Angle vector
halfAngleTemp = normalize(EyeDir + lightTemp);

// Multiply the Half Angle vector by NOISE_FACTOR
// to avoid noisy BRDF data
halfAngleTemp = halfAngleTemp * 0.9;

// Calculate T' and B'
//    T' = |T - (T.H)H|
tPrime = Tangent – (halfAngleTemp *
                        dot(Tangent, halfAngleTemp));
tPrime = normalize(tPrime);
```

BRDF PTM Vertex Shader (3 of 4)

```
//    B' = H x T'
bPrime = cross(halfAngleTemp, tPrime);

// Hu = Dot(Light, T')
// Hv = Dot(Light, B')
LdotT = dot(lightTemp, tPrime);
LdotB = dot(lightTemp, bPrime);

// Du = Dot(HalfAngle, T)
// Dv = Dot(HalfAngle, B)
// Remap [-1.0..1.0] to [0.0..1.0]
gl_TexCoord0.s = dot(Tangent,  halfAngleTemp) * 0.5 + 0.5;
gl_TexCoord0.t = dot(Binormal, halfAngleTemp) * 0.5 + 0.5;
```

BRDF PTM
Vertex Shader (4 of 4)

```
    gl_TexCoord0.q = 1.0;

    // "S" Text Coord3: Dot(Light, Normal);
    LdotN = dot(lightTemp, gl_Normal) * 0.5 + 0.5;
}
```

BRDF PTM Fragment Shader Overview

- Goal of fragment shader is
  - Look up polynomial coefficients A, B, C, D, E, F based on incoming texture coordinate
  - Apply coefficient scale and bias factors
  - Compute the polynomial
  - Use LdotN value to look up a light factor
  - Multiply and clamp to produce final color

BRDF PTM Fragment Shader (1 of 3)

```
// PP_PTM_3DLABS_RGB
const int ABCtexture   = 0;
const int DEFtexture   = 1;
const int lighttexture = 3;

uniform vec3 ABCscale, ABCbias;
uniform vec3 DEFscale, DEFbias;

varying float LdotT; // passes the computed L*Tangent value
varying float LdotB; // passes the computed L*Binormal value
varying float LdotN; // passes the computed L*Normal value

void main(void)
{
       vec3    ABCcoef, DEFcoef;
       vec3    ptvec;
       float   ptval;
```

BRDF PTM Fragment Shader (2 of 3)

```
// Read coefficient values and apply scale and bias factors
ABCcoef = (texture3(ABCtexture, gl_TexCoord0, 0.0)
                - ABCbias) * ABCscale;
DEFcoef = (texture3(DEFtexture, gl_TexCoord0, 0.0)
                - DEFbias) * DEFscale;

// Compute polynomial
ptval = ABCcoef.0 * LdotT * LdotT +
        ABCcoef.1 * LdotB * LdotB +
        ABCcoef.2 * LdotT * LdotB +
        DEFcoef.0 * LdotT +
        DEFcoef.1 * LdotB +
        DEFcoef.2;
```

BRDF PTM Fragment Shader (3 of 3)

```
    // Turn result into a vec3
    ptvec = vec3 (ptval);

    // Multiply result * light factor
    ptvec *= texture3(lighttexture, LdotN);

    vec3 fragc = clamp(ptvec, vec3 (0.0), vec3 (1.0));

    // Clamp result and assign it to gl_FragColor
    gl_FragColor = vec4 (fragc, 1.0);
}
```

BRDF Demo

Other Demos

- Time permitting…

Feedback

SAN ANTONIO SIGGRAPH 2002

“We're at the point where we can apply an OGL2 shader through our (Houdini) interface and (given an equivalent VEX shader) watch the software renderer (Mantra) draw the same thing but much, much slower :-). It's one of those jaw dropping "wow" moments actually, so we thank you for making that happen! . . . It rocks. Having read the original white paper still did not prepare us to see it actually working. The ease with which we can now define & adjust OGL2 shaders is astonishing. ”

Unsolicited email from Paul Salvini, CTO, Side Effects Software

3Dlabs

More Info

SAN ANTONIO SIGGRAPH 2002

- www.3dlabs.com/developer/ogl2
- These slides will be posted there in a few days
- Updated “course notes” will also be there eventually
- OpenGL BOF Wednesday 4:30 p.m.
- Various ISVs will be speaking in the 3Dlabs booth Tue-Thu: LightWork Design, Vital Images, Volume Graphics, Terrex, University of Magdeburg

3Dlabs