GPU Shading

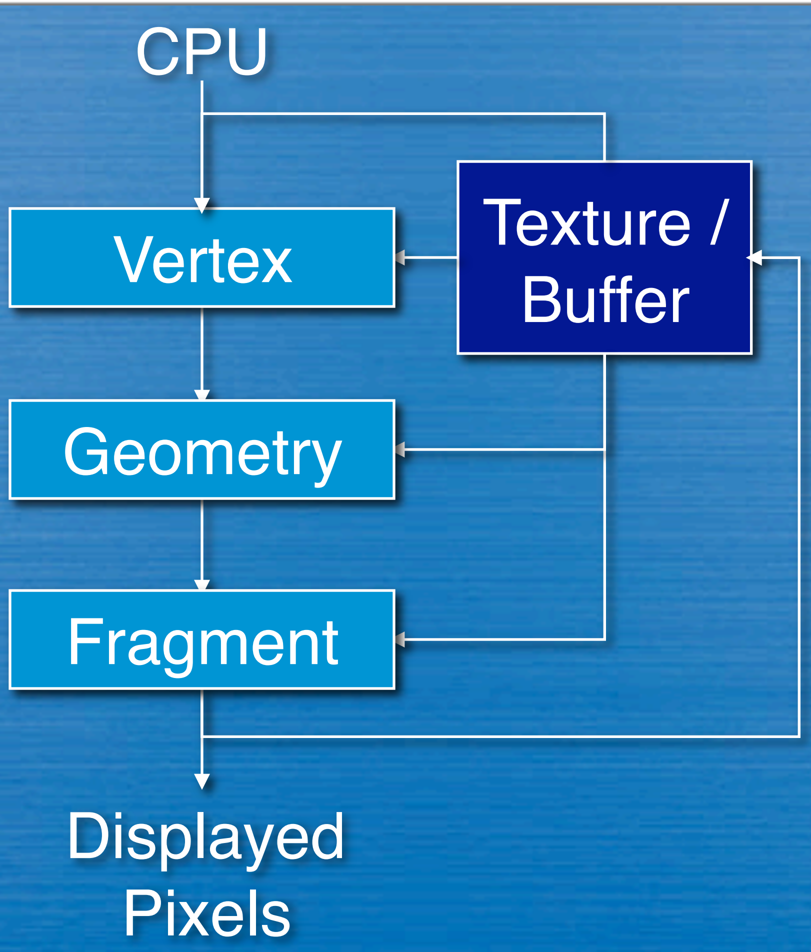
CMSC 435/634

GPU

- GPU: Graphics Processing Unit
  - Designed for real-time graphics
  - Present in almost every PC
  - Increasing realism and complexity

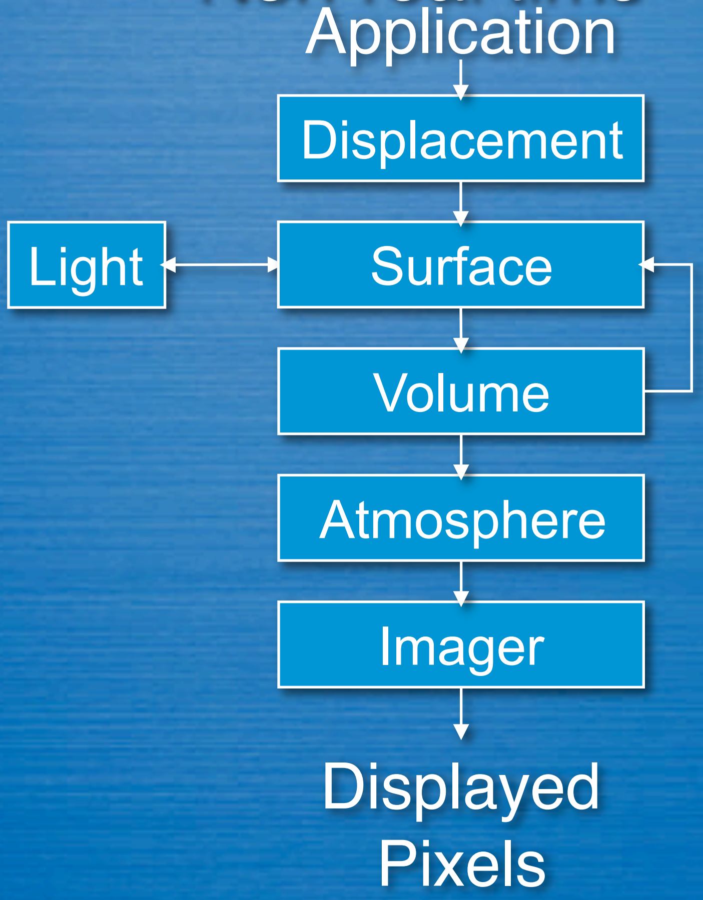
Americas Army

GPU computation



Non-real time vs. Real-time

- Non-real time



Application

Displacement

Light

Surface

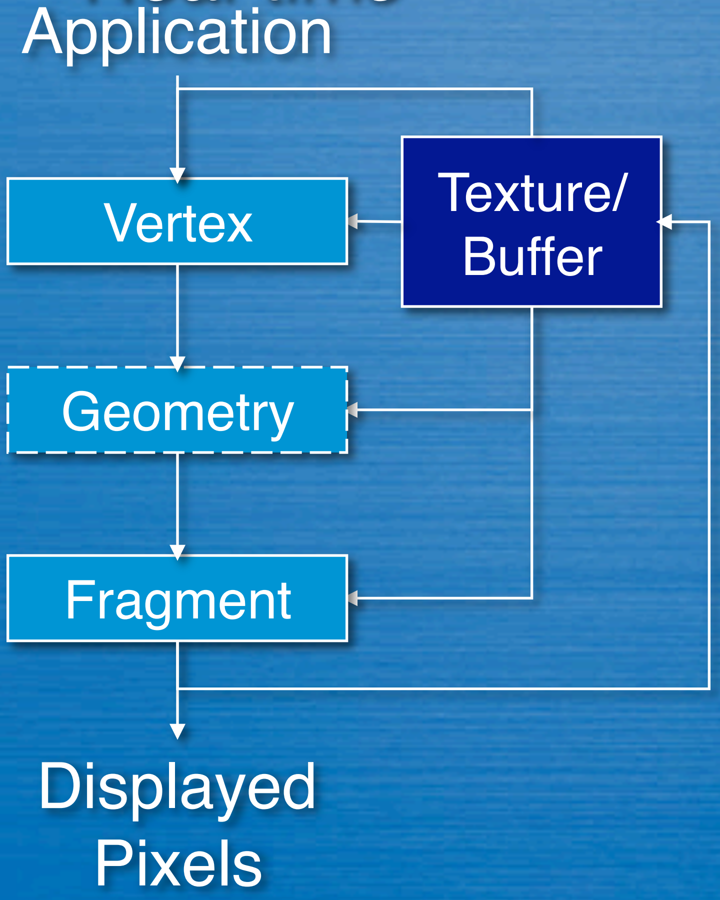
Volume

Atmosphere

Imager

Displayed Pixels

- Real-time



Application

Vertex

Texture/ Buffer

Geometry

Fragment

Displayed Pixels

Non-real time vs. Real time

- Not real-time
  - Developed from General CPU code
  - Seconds to hours per frame
  - 1000s of lines
  - “Unlimited” computation, texture, memory, …

- Real-time
  - Developed from fixed-function hardware
  - Tens of frames per second
  - 1000s of instructions
  - Limited computation, texture, memory, …

History (not real-time)

- Testbed [Whitted and Weimer 1981]
- Shade Trees [Cook 1984]
- Image Synthesizer [Perlin 1985]
- RenderMan [Hanrahan and Lawson 1990]
- Multi-pass RenderMan [Peercy et al. 2000]
- GPU acceleration [Wexler et al. 2005]

History (real-time)

- Custom HW [Olano and Lastra 1998]
- Multi-pass standard HW [Peercy et al. 2000]
- Register combiners [NVIDIA 2000]
- Vertex programs [Lindholm et al. 2001]
- Compiling to mixed HW [Proudfoot et al. 2001]
- Fragment programs
- Standardized languages
- Geometry shaders [Blythe 2006]

Choices

- OS: Windows, Mac, Linux
- API: DirectX, OpenGL, XNA
- Language: HLSL, GLSL, Cg, …
- Compiler: DirectX, OpenGL, Cg, ASHLI
- Runtime: CgFX, ASHLI, OSG (& others), sample code

Major Commonalities

- Vertex, Geometry & Fragment/Pixel
- C-like, if/while/for
- Structs & arrays
- Float + small vector and matrix
  - Swizzle & mask (a.xyz = b.xxw)
- Common math & shading functions

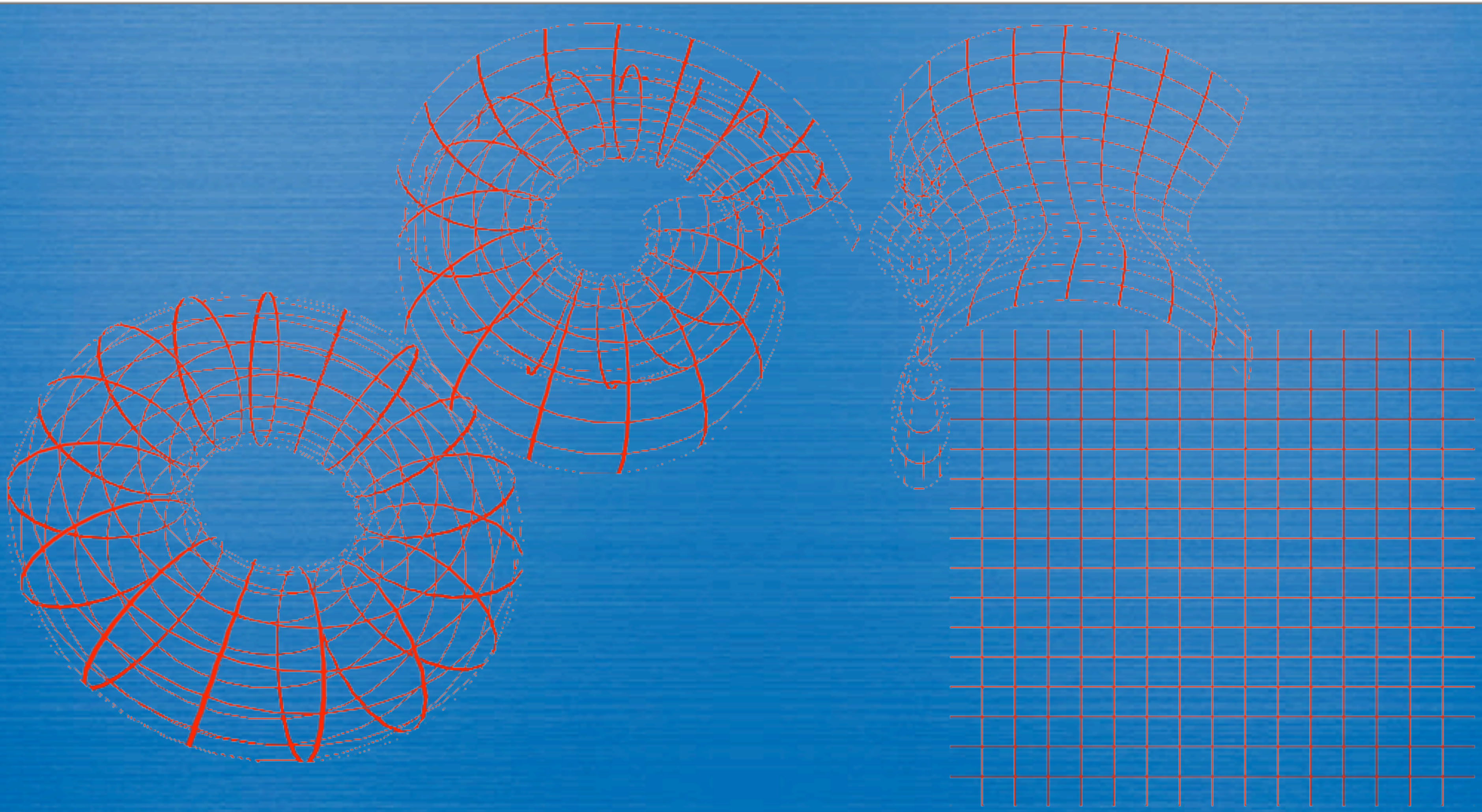
OpenGL Shading

- High level language
  - OpenGL Shading Language = GLslang = GLSL
- Integrated into OpenGL API
  - no extra run-time

API-integrated

- Compiler built into driver
  - Presumably they know your card best
  - IHV’s must produce (good) compilers
- Use built-in parameters (glColor, glNormal, …)
  - Add your own
  - Built-ins disappearing in OpenGL *“Mount Evans”*
- Other options can still produce low-level code
  - Cg, ASHLI, RapidMind, …
  - With loss of integration

Vertex Demo: Blend Positions



Vertex Shader Code

```
void main() {
    float Kin = gl_Color.r;          // key input

    // screen position from vertex and texture
    vec4 Vp = ftransform();
    vec4 Tp = vec4(gl_MultiTexCoord0.xy*1.8-.9, 0.,1.);

    // interpolate between Vp and Tp
    gl_Position = mix(Tp,Vp,pow(1.-Kin,8.));

    // copy to output
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_TexCoord[1] = Vp;
    gl_TexCoord[3] = vec4(Kin);
}
```

Main Function

```
void main() {
    float Kin = gl_Color.r;         // key input

    // screen position from vertex and texture
    vec4 Vp = ftransform();
    vec4 Tp = vec4(gl_MultiTexCoord0.xy*1.8-.9, 0.,1.);

    // interpolate between Vp and Tp
    gl_Position = mix(Tp,Vp,pow(1.-Kin,8.));

    // copy to output
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_TexCoord[1] = Vp;
    gl_TexCoord[3] = vec4(Kin);
}
```

Use Standard OpenGL State

```
void main() {
    float Kin = gl_Color.r;        // key input

    // screen position from vertex and texture
    vec4 Vp = ftransform();
    vec4 Tp = vec4(gl_MultiTexCoord0.xy*1.8-.9, 0.,1.);

    // interpolate between Vp and Tp
    gl_Position = mix(Tp,Vp,pow(1.-Kin,8.));

    // copy to output
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_TexCoord[1] = Vp;
    gl_TexCoord[3] = vec4(Kin);
}
```

Built-in Types

```
void main() {
    float Kin = gl_Color.r;          // key input

    // screen position from vertex and texture
    vec4 Vp = ftransform();
    vec4 Tp = vec4(gl_MultiTexCoord0.xy*1.8-.9, 0.,1.);

    // interpolate between Vp and Tp
    gl_Position = mix(Tp,Vp,pow(1.-Kin,8.));

    // copy to output
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_TexCoord[1] = Vp;
    gl_TexCoord[3] = vec4(Kin);
}
```

Swizzle / Channel Selection

```
void main() {
    float Kin = gl_Color.r;          // key input

    // screen position from vertex and texture
    vec4 Vp = ftransform();
    vec4 Tp = vec4(gl_MultiTexCoord0.xy*1.8-.9, 0.,1.);

    // interpolate between Vp and Tp
    gl_Position = mix(Tp,Vp,pow(1.-Kin,8.));

    // copy to output
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_TexCoord[1] = Vp;
    gl_TexCoord[3] = vec4(Kin);
}
```

Vector Construction

```
void main() {
    float Kin = gl_Color.r;          // key input

    // screen position from vertex and texture
    vec4 Vp = ftransform();
    vec4 Tp = vec4(gl_MultiTexCoord0.xy*1.8-.9, 0.,1.);

    // interpolate between Vp and Tp
    gl_Position = mix(Tp,Vp,pow(1.-Kin,8.));

    // copy to output
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_TexCoord[1] = Vp;
    gl_TexCoord[3] = vec4(Kin);
}
```

Built-in Functions

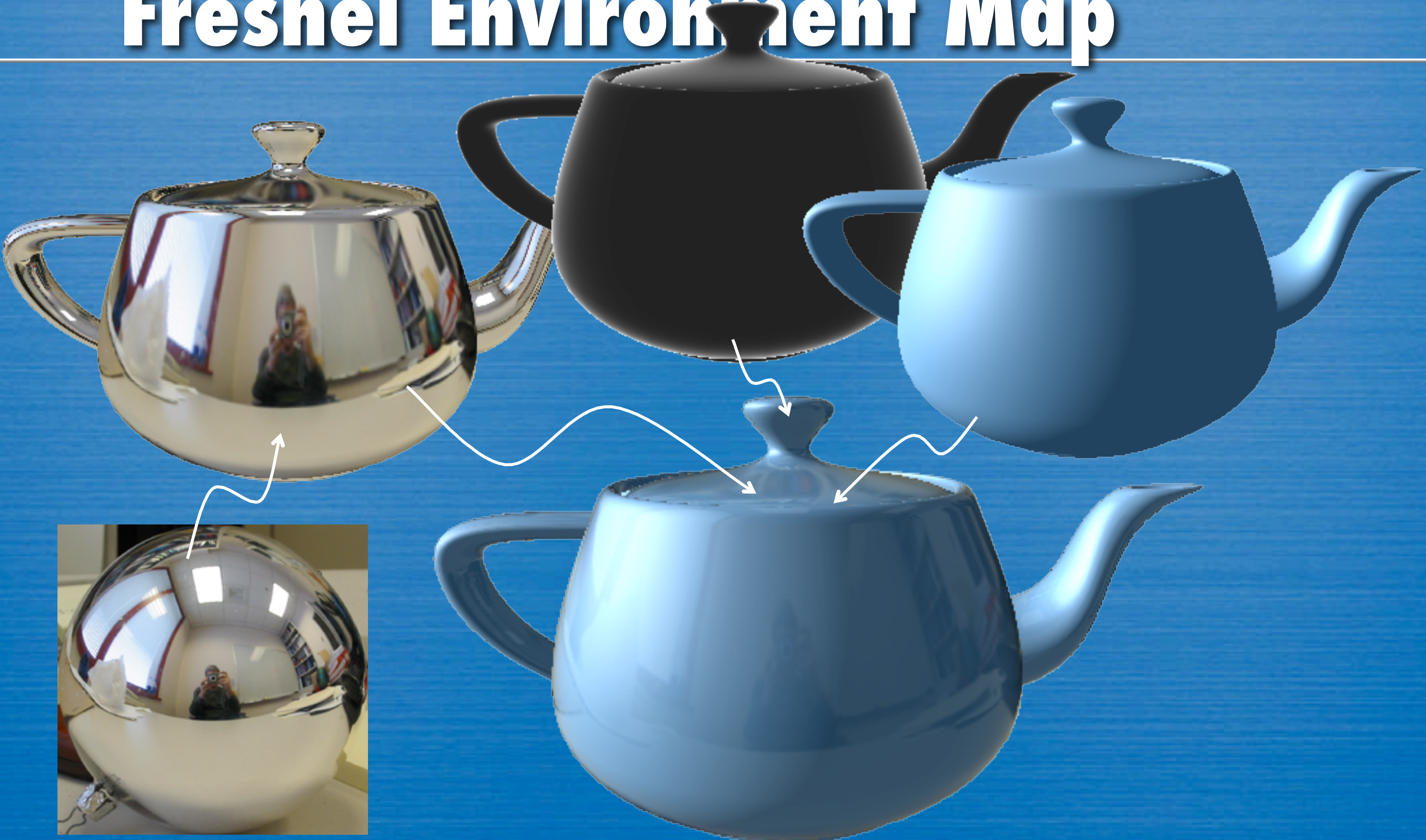
```
void main() {
    float Kin = gl_Color.r;          // key input

    // screen position from vertex and texture
    vec4 Vp = ftransform();
    vec4 Tp = vec4(gl_MultiTexCoord0.xy*1.8-.9, 0.,1.);

    // interpolate between Vp and Tp
    gl_Position = mix(Tp,Vp,pow(1.-Kin,8.));

    // copy to output
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_TexCoord[1] = Vp;
    gl_TexCoord[3] = vec4(Kin);
}
```

Vertex + Fragment Demo: Fresnel Environment Map



Using GLSL (OpenGL)

- Create shader object

```
S = glCreateShader(GL_VERTEX_SHADER)
S = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB)
```

  - Vertex or Fragment

- Load shader into object

```
glShaderSource(S, n, shaderArray, lenArray)
glShaderSourceARB(S, n, shaderArray, lenArray)
```

  - Array of strings

- Compile object

```
glCompileShader(S)
glCompileShaderARB(S)
```

Loading Shaders (OpenGL)

- `glShaderSource(S, n, shaderArray, lenArray)`
  - One string containing entire mmap'd file
  - Strings as #includes
    - Varying variables between vertex and fragment
  - Strings as lines
    - Null-terminated if lenArray is Null or length=-1

Using GLSL (OpenGL)

- Create program object

  ```
  P = glCreateProgram()
  P = glCreateProgramObjectARB()
  ```

- Attach all shader objects

  ```
  glAttachShader(P, S)
  glAttachObjectARB(P, S)
  ```

  - Vertex, Fragment or both

- Link together

  ```
  glLinkProgram(P)
  glLinkProgramARB(P)
  ```

- Use

  ```
  glUseProgramObject(P)
  glUseProgramObjectARB(P)
  ```

Using Parameters (OpenGL)

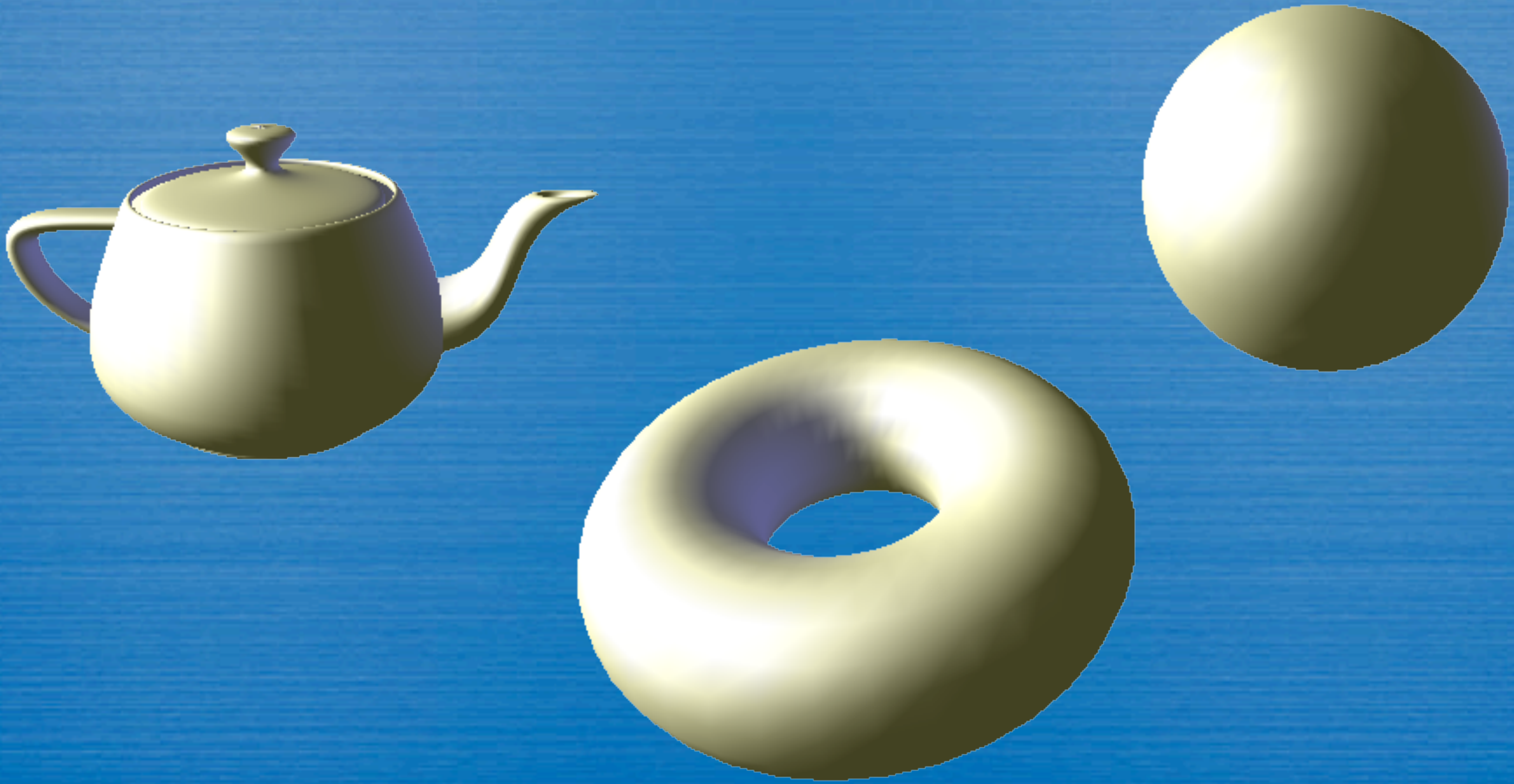
- Where is my attributes/uniforms parameter?

```
i=glGetAttribLocation(P,”myAttrib”)
i=glGetUniformLocation(P,”myAttrib”)
```

- Set them

```
glVertexAttrib1f(i,value)
glVertexAttribPointer(i,…)
glUniform1f(i,value)
```

OpenGL State Demo: Vertex Lighting



Lighting Vectors in Eye Space

```
void main() {
  // convert shading-related vectors to eye space
  vec4 P = gl_ModelViewMatrix*gl_Vertex;
  vec4 E = gl_ProjectionMatrixInverse*vec4(0,0,-1,0);
  vec3 V = normalize(E.xyz*P.w-P.xyz*E.w);
  vec3 N = normalize( gl_NormalMatrix*gl_Normal) ;
  …
```

Accumulate Each Light

```
…
// accumulate contribution from each light
gl_FrontColor = vec4(0);
for(int i=0; i<gl_MaxLights; i++) {
  vec3 L = normalize(gl_LightSource[i].position.xyz*P.w
                             - P.xyz*gl_LightSource[i].position.w);
  vec3 H = normalize(L+V);
  float diff = dot(N,L);

  gl_FrontColor += gl_LightSource[i].ambient;
  if (diff > 0.) {
    gl_FrontColor  +=     gl_LightSource[i].diffuse * diff;
    gl_FrontColor  +=     gl_LightSource[i].specular *
       max(pow(dot(N,H), gl_FrontMaterialShininess),0.);
  }
}
…
```

Standard Vertex Shader Stuff

```
  …
  // standard texture coordinate and position stuff
  gl_TexCoord[0] = gl_TextureMatrix[0]*gl_MultiTexCoord0;
  gl_Position = ftransform();
}
```

Shader Design Strategies

- Learn and adapt from RenderMan
  - Noise
  - Layers
- Multiple Passes
- *Baked* computation