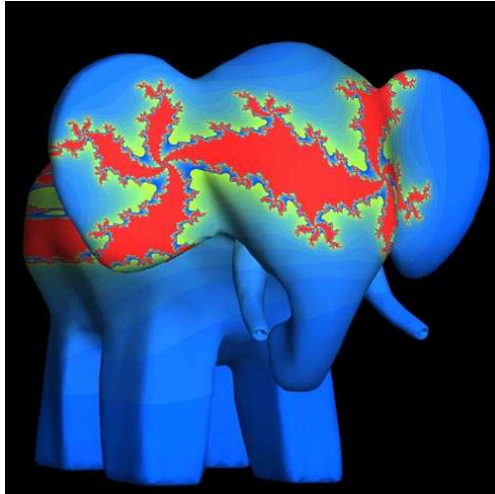


Intro to OpenGL Shading Language (GLSL)



Why should we care?

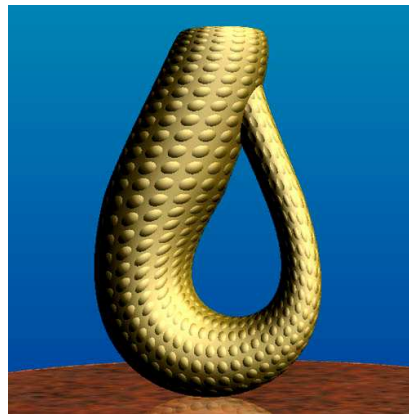
• We can do lots of really cool stuff in real-time, without overworking the CPU

•Some Examples:

- Phong Shading
- Bump Mapping
- 3D/Procedural Textures
- Particle Systems
- Animation
- Much much more!

•Beyond real-time graphics:

- Image Processing
- Scientific Data Processing
- Parallel Computing



OpenGL Logical Diagram

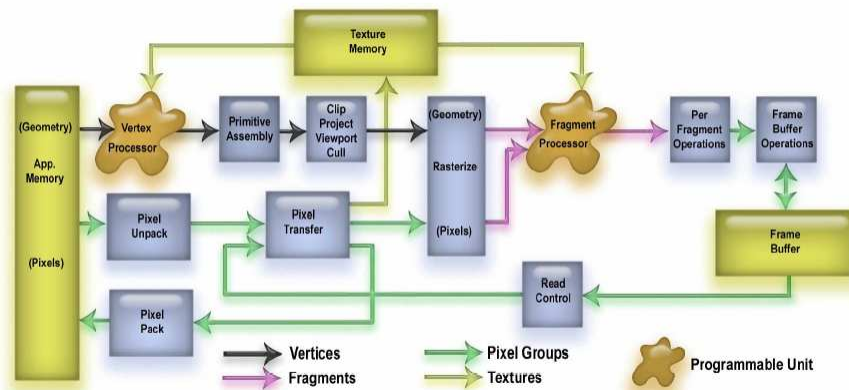


Image Source: Randi Rost Tokyo ACM SIGGRAPH 2004

Our GLSL Programs

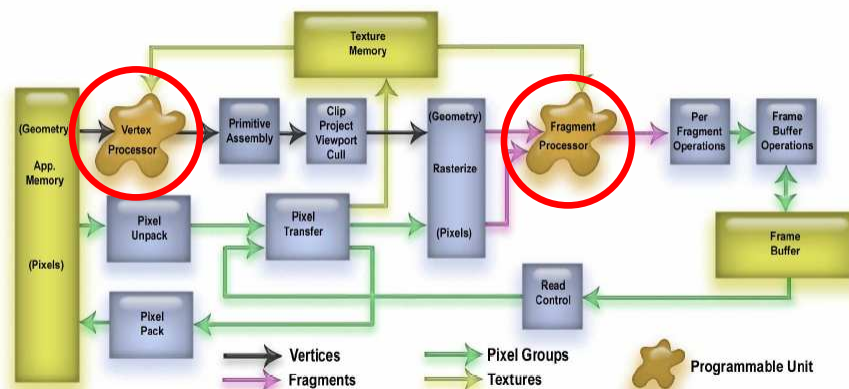


Image Source: Randi Rost Tokyo ACM SIGGRAPH 2004

Graphics Card Issues

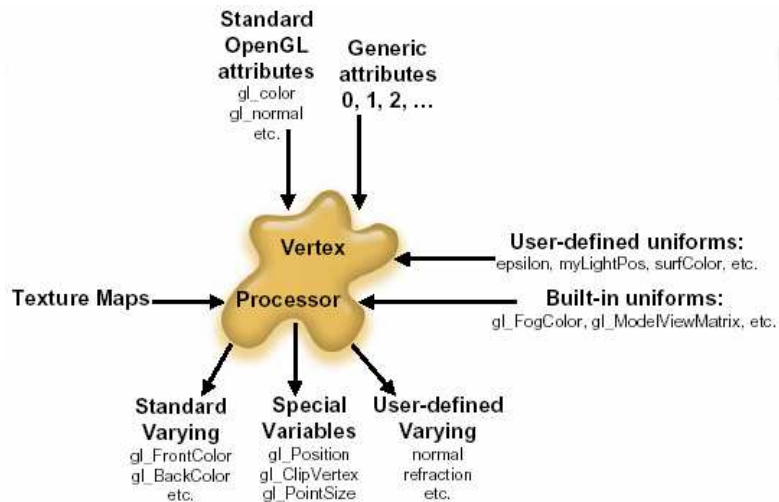
- Different graphics cards = different levels of support
- Floating point precision
- Knowing your card's specifications can help avoid headaches
- **Example: GeForce FX cards**
 - Support vertex shaders
 - Support fragment shaders
 - Fragment shaders DO NOT support "real" branching in hardware



GLSL Program Syntax

- **Similar to C**
 - Entry point: `void main() {...}`
- **Scalars**
 - `float, int, bool`
- **Vectors natively supported**
 - `vec[2|3|4]` -> vector of 2, 3, or 4 floats
 - `ivec[2|3|4], bvec[2|3|4]`
- **Matrices also natively supported**
 - `mat2, mat3, mat4` -> 2x2, 3x3, 4x4 floats
- **structs, functions, arrays and a lot more available**
 - Syntax often slightly different from C/C++

Diagram (From Randi Rost's slides)



Vertex Shaders

- **Per-vertex calculations performed here**
 - Without knowledge about other vertices (parallelism)
- **When writing a vertex shader you take responsibility for:**
 - Vertex transformation
 - Normal transformation
 - (Per-Vertex) Lighting
 - Color material application and color clamping
 - Texture coordinate generation
- **This means that you can do really cool stuff**
- **This also means you lose some automatic functionality**
 - Example: Texture coordinate generation for sphere mapping
- **You also need to set up data for fragment shaders**

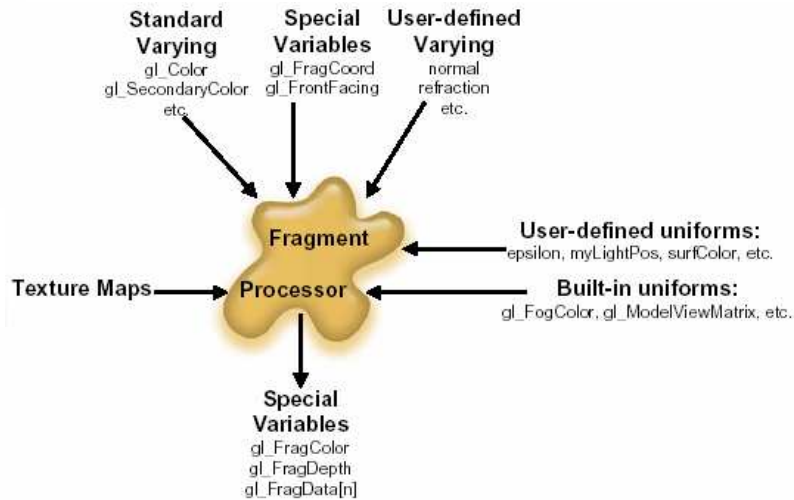
gl_Position and ftransform()

- `gl_Position` is mandatory
- `fttransform()` provides *positional invariance*
- Important in multipass rendering, when we need the same exact vertex position in clip coordinates
- `gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex`
 - Compiler optimizations and hardware differences lead to a loss of positional invariance

Simple Vertex Shader Program

```
varying vec3 ecNormal; //shared with fragment shader
varying vec3 ecPosition; //shared with fragment shader
uniform vec3 LightPosition; // From our program
void main(void)
{
    // Transform the vertex by the model-view matrix
    ecPosition = vec3 (gl_ModelViewMatrix * gl_Vertex);
    // Transform the normal
    ecNormal = normalize(gl_NormalMatrix * gl_Normal);
    // Find a vector from our position to the light
    // (just because we can)
    vec3 lightVec = normalize(LightPosition - ecPosition);
    // Get the same position as if OpenGL calculated it
    gl_Position = ftransform();
}
```

Diagram (From Randi Rost's slides)



Fragment Shaders

• What is a fragment?

- Cg Tutorial says: "You can think of a fragment as a 'potential pixel'"

Perform per-pixel calculations

- Without knowledge about other fragments (parallelism)

• Responsibilities and opportunities

- Operations on interpolated values
- Texture access and application
- Fog
- Color lookup

- Usually when you finished, you've assigned your fragment a color

(Very) Simple Fragment Program

```
varying vec3 ecPosition; // From vertex shaders,
                        // interpolated for us
varying vec3 ecNormal;   // From vertex shaders,
                        // interpolated for us
uniform vec3 LightPosition; // From our program

void main(void)
{
    // if we wanted to use our normal, we should normalize it
    ecNormal=normalize(ecNormal);

    // but, we will just set our color to be red
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Sharing Data Between Shaders

- Fragment shaders often need data that's been computed in a vertex shader
- In our vertex shader, declare these variables as `varying`
- Automatically interpolated between vertices
- We can only share in one direction vertex to shader
- Cannot share data between vertex shaders
- Some commonly shared pieces of data:
 - Position, normals, light intensity, light direction vector, texture coordinates (these variables shared automatically)

Getting Data From Our (Main) Program

- We can get data to our shaders from several places
- Data we get automatically
 - Vertex positions, normals, material properties, modelview matrix, texture coordinate, etc. (think glFoo calls)
- Data we send in
 - Infrequently changing data: uniform
 - `glUniform{1|2|3|4}{type}ARB(Gluint location, TYPE v)`
 - Frequently changing data: attribute
 - If it's set per-vertex, use attribute
 - `glVertexAttrib{1|2|3|4}{type}ARB(Gluint index, TYPE v)`

Getting Shaders into Programs

- Shader objects are created with:
 - `shaderID = glCreateShaderObjectARB(shaderType);`
- Shader source code is supplied to Shader source code is supplied to OpenGL using:
 - `glShaderSourceARB(shaderID, numStrings, strings, lengths)`
- Shader objects are compiled with:
 - `glCompileShaderARB(shaderID);`
- To determine whether the shader object was compiled successfully:
 - `glGetObjectParameterfARB(shaderID, GL_OBJECT_COMPILE_STATUS_ARB, status);`

Continued

- **Program objects are created with:**
 - `programID =glCreateProgramObjectARB();`
- **The “link recipe” is created using:**
 - `glAttachObjectARB(programID, shaderID)`
 - `glDetachObjectARB (programID,shaderID)`
- **A program object is linked with:**
 - `glLinkProgramARB(programID)`
- **A program object is made current with:**
 - `glUseProgramObjectARB(programID)` (call with 0 to disable)
- **To determine whether the program object was linked successfully:**
 - `glGetObjectParameterfARB(programID, GL_OBJECT_LINK_STATUS_ARB, status);`

Some Useful Books

- **The Orange Book: OpenGL Shading Language**
- **The Cg Tutorial**
- **GPU Gems I & II**



Useful Websites

- <http://3dshaders.com/>
 - Official site for the orange book
- <http://developer.3dlabs.com/downloads/>
- <http://developer.3dlabs.com/documents/index.htm>
 - Find Randi Rost's slides (the contents of which were used on several of the slides in this presentation)
- <http://developer.nvidia.com/>
 - A lot of information is Cg specific, but has general GPU programming information as well
- <http://www.opengl.org/>