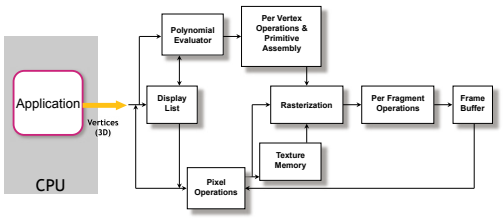
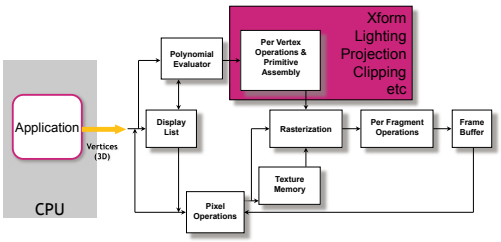
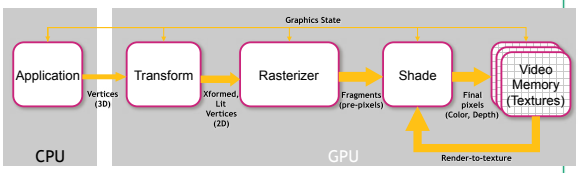
Traditional Graphics Pipeline



Traditional Graphics Pipeline



Traditional Graphics Pipeline



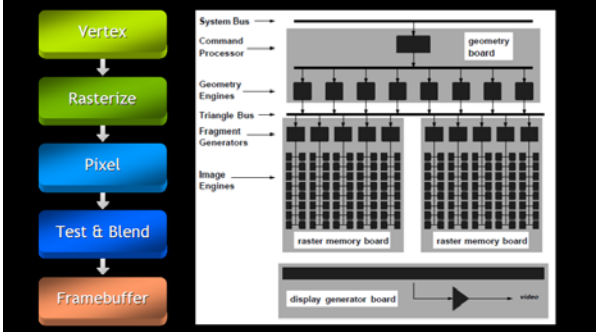
A simplified graphics pipeline
- Note that pipe widths vary
- Many caches, FIFOs, and so on not shown

The Graphics Pipeline

- Key abstraction of real-time graphics
- Hardware used to look like this
- Distinct chips/boards per stage
- Fixed data flow through pipeline

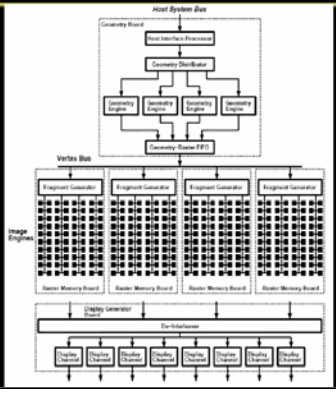
Beyond Programmable Shading: In Action

SGI RealityEngine (1993)



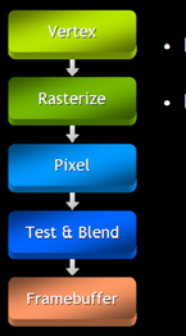
Kurt Akeley. RealityEngine Graphics. In *Proc. SIGGRAPH '93* ACM Press, 1993.

SGI InfiniteReality (1997)



Montrym, Baum, Dignam, & Migdal. InfiniteReality: A real-time graphics system. In *Proc. SIGGRAPH '97* ACM Press, 1997.

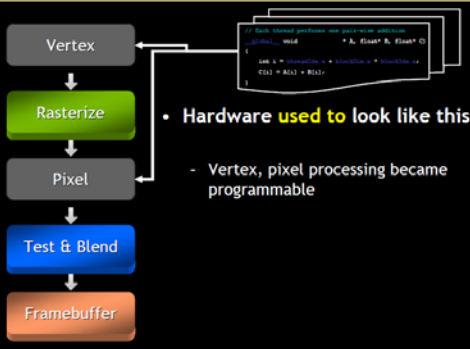
The Graphics Pipeline



- Remains a useful abstraction
- Hardware used to look like this

Beyond Programmable Shading: In Action

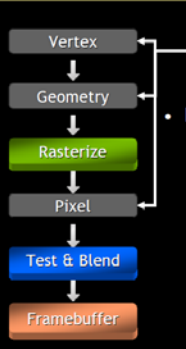
The Graphics Pipeline



- Hardware used to look like this:
  - Vertex, pixel processing became programmable

Beyond Programmable Shading: In Action

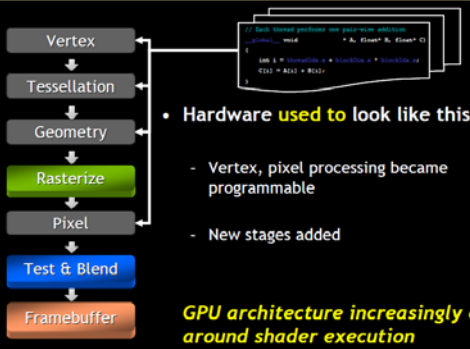
The Graphics Pipeline



- Hardware used to look like this
  - Vertex, pixel processing became programmable
  - New stages added

Beyond Programmable Shading: In Action

The Graphics Pipeline

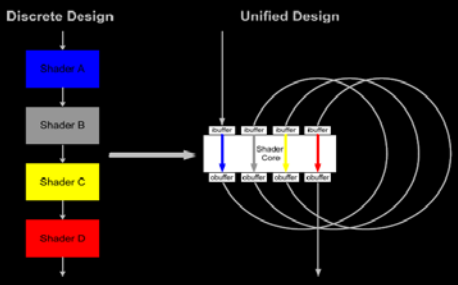


- Hardware used to look like this
  - Vertex, pixel processing became programmable
  - New stages added

GPU architecture increasingly centers around shader execution

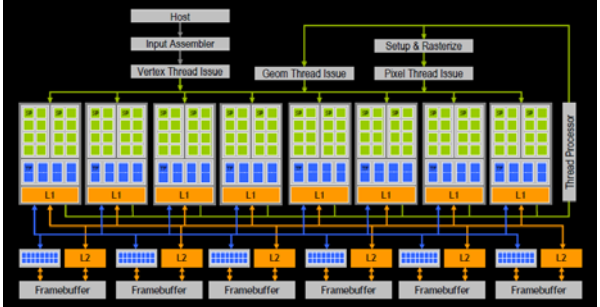
Beyond Programmable Shading: In Action

Modern GPUs: Unified Design

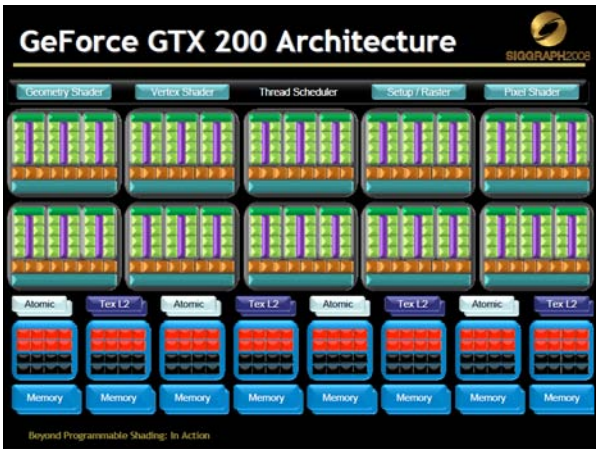


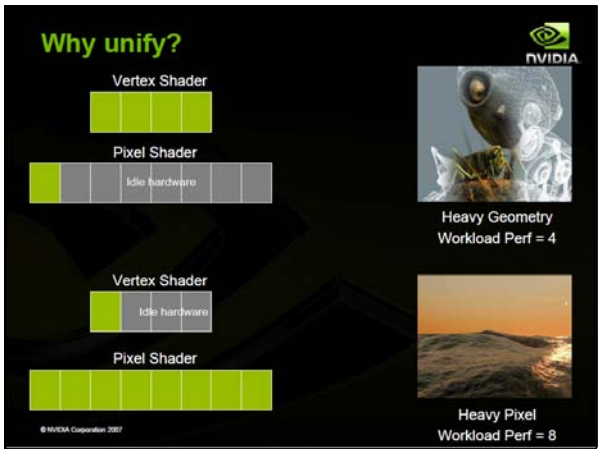
Vertex shaders, pixel shaders, etc. become *threads* running different programs on a flexible core

GeForce 8: Modern GPU Architecture

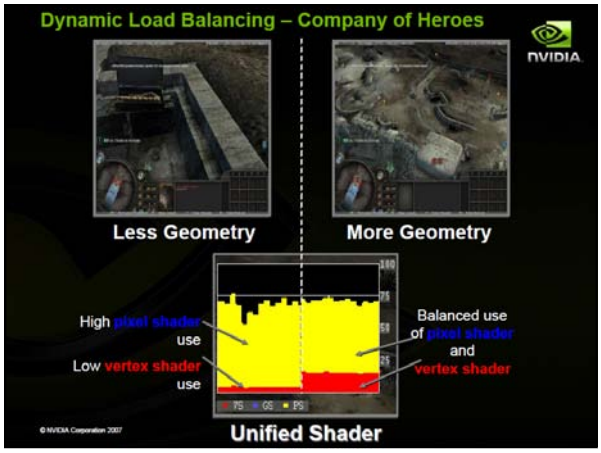


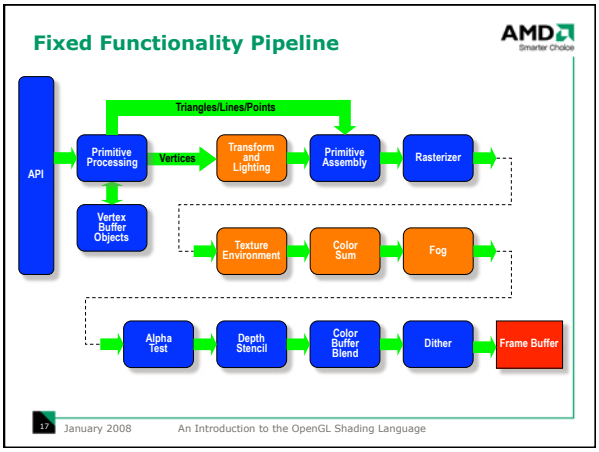
Beyond Programmable Shading: In Action

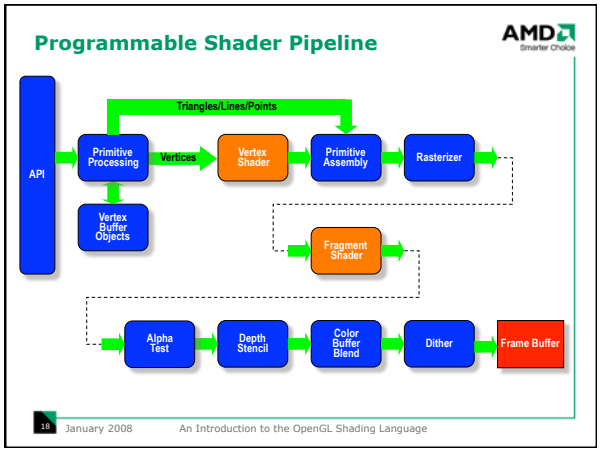

GeForce GTX 200 Architecture
SIGGRAPH2008
Geometry Shader
Vertex Shader
Thread Scheduler
Setup / Raster
Pixel Shader
Atomic
Tex L2
Memory
Beyond Programmable Shading: In Action


Why unify?
NVIDIA
Vertex Shader
Pixel Shader
Idle hardware
Heavy Geometry
Workload Perf = 4
Vertex Shader
Idle hardware
Pixel Shader
Heavy Pixel
Workload Perf = 8

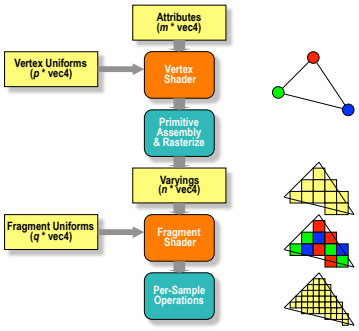

Why unify?
NVIDIA
Unified Shader
Vertex Workload
Pixel
Heavy Geometry
Workload Perf = 11
Unified Shader
Pixel Workload
Vertex
Heavy Pixel
Workload Perf = 11


Dynamic Load Balancing – Company of Heroes
NVIDIA
Less Geometry
More Geometry
High pixel shader use
Low vertex shader use
Balanced use of pixel shader and vertex shader
VS
GS
PS
Unified Shader

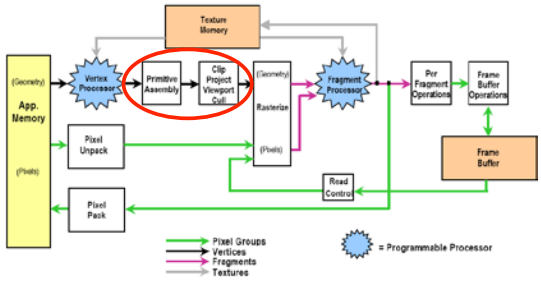

Fixed Functionality Pipeline
AMD
Smarter Choice
API
Triangles/Lines/Points
Primitive Processing
Vertices
Transform and Lighting
Primitive Assembly
Rasterizer
Vertex Buffer Objects
Texture Environment
Color Sum
Fog
Alpha Test
Depth Stencil
Color Buffer Blend
Dither
Frame Buffer
17
January 2008
An Introduction to the OpenGL Shading Language


Programmable Shader Pipeline
AMD
Smarter Choice
API
Triangles/Lines/Points
Primitive Processing
Vertices
Vertex Shader
Primitive Assembly
Rasterizer
Vertex Buffer Objects
Fragment Shader
Alpha Test
Depth Stencil
Color Buffer Blend
Dither
Frame Buffer
18
January 2008
An Introduction to the OpenGL Shading Language

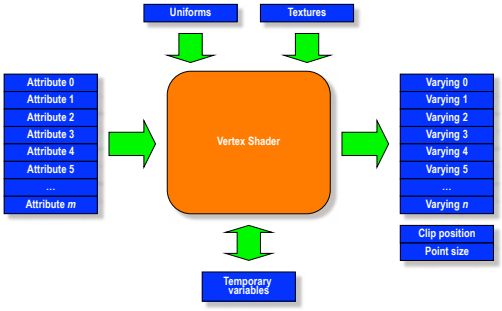
Programmer's Model



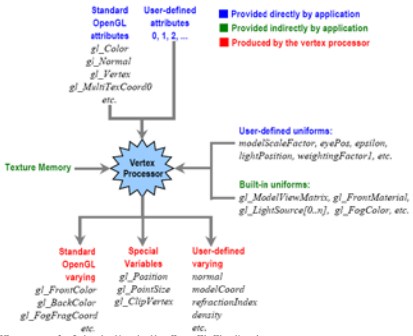
OpenGL 2.0 Logical Diagram



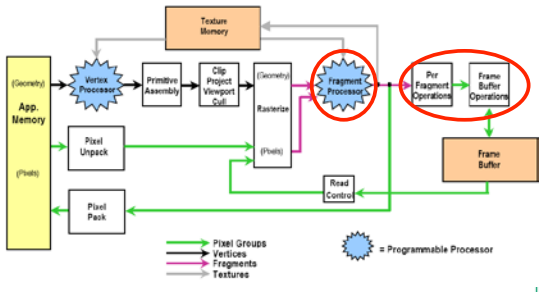
Vertex Shader Environment



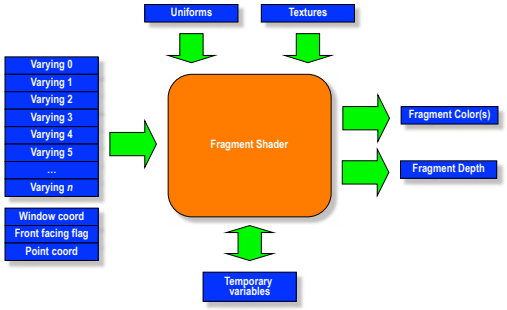
Vertex Processor Overview



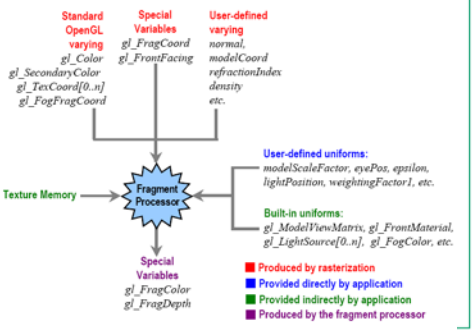
OpenGL 2.0 Logical Diagram



Fragment Shader Environment



Fragment Processor Overview



Hello World!

```
void main(void)
{
    // This is our Hello World vertex shader

    // Standard MVP transform
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}

void main(void)
{
    // This is our Hello World fragment shader

    // Set to a constant color (hint: look at it upside down)
    gl_FragColor = vec4(0.7734);
}
```


Basic method

2 basic object types

- Shader object
- Program object

Create Vertex & Fragment Shader Objects

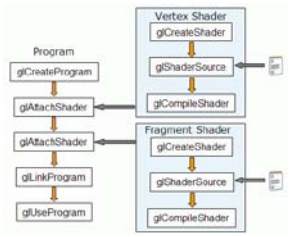
Compile both

Create program object & attach shaders

Link program

Use program

Creating Shaders



Compiling

```
void glShaderSource(GLuint shader, GLsizei nstrings, const GLchar **strings,
    const GLint *lengths)
    //if lengths==NULL, assumed to be null-terminated


void glCompileShader (GLuint shader);
```


Attaching & Linking

```
void glAttachShader(GLuint program, GLuint shader);
    //twice, once for vertex shader & once for fragment shader


void glLinkProgram(GLuint program);
    //program now ready to use


void glUseProgram(GLuint program);
    //switches on shader, bypasses FFP
    //if program==0, shaders turned off, returns to FFP
```

In short…

```
GLuint programObject;
GLuint vertexShaderObject;
GLuint fragmentShaderObject;

unsigned char *vertexShaderSource = readShaderFile(vertexShaderFilename);
unsigned char *fragmentShaderSource = readShaderFile(fragmentShaderFilename);

programObject=glCreateProgram ();
vertexShaderObject=glCreateShader (GL_VERTEX_SHADER);
fragmentShaderObject=glCreateShader (GL_FRAGMENT_SHADER);

glShaderSource (vertexShaderObject,1,(const char**)&vertexShaderSource,NULL);
glShaderSource (fragmentShaderObject,1,(const char**)&fragmentShaderSource,NULL);

glCompileShader (vertexShaderObject);
glCompileShader (fragmentShaderObject);

glAttachObject (programObject, vertexShaderObject);
glAttachObject (programObject, fragmentShaderObject);

glLinkProgram (programObject);

glUseProgram (programObject);
```


Example

```
void setShaders() {
        char *vs,*fs;

        v = glCreateShader(GL_VERTEX_SHADER);
        f = glCreateShader(GL_FRAGMENT_SHADER);

        vs = textFileRead("toon.vert");
        fs = textFileRead("toon.frag");

        const char * vv = vs;
        const char * ff = fs;

        glShaderSource(v, 1, &vv,NULL);
        glShaderSource(f, 1, &ff,NULL);

        free(vs);free(fs);

        glCompileShader(v);
        glCompileShader(f);

        p = glCreateProgram();

        glAttachShader(p,v);
        glAttachShader(p,f);

        glLinkProgram(p);
        glUseProgram(p);
}
```


Other functions

Clean-up

```
void glDetachObject (GLuint container, GLuint attached);
void glDeleteObject (GLuint object);
```

Info Log

```
void glGetInfoLog (GLuint object,      GLsizei maxLength, GLsizei
*length,      GLchar *infoLog);
```

- Returns compile & linking information, errors

Useful References

http://www.3dshaders.com/
- Home page for the "orange book" focused solely on GLSL

http://www.opengl.org/sdk/
- OpenGL SDK, including links to the below resources

http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf
- one double-sided page cheat sheet to GLSL – indispensible!

http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf
- This is the ultimate authority: the GLSL specification document

http://www.opengl.org/sdk/docs/books/SuperBible/
- Full reference and tutorial to OpenGL 2.1
- All sample code downloadable for Windows, Mac OS X, and Linux