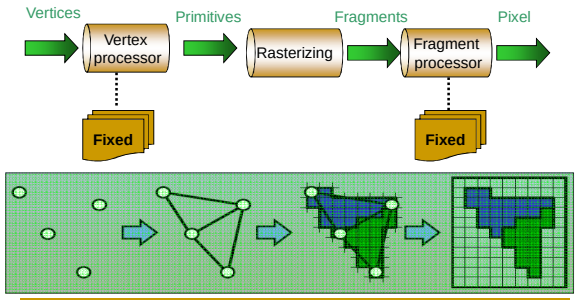
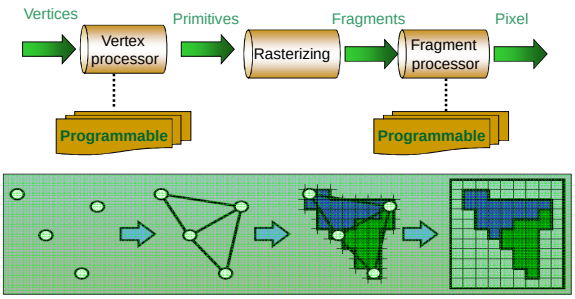
GPU Programming Using GLSL

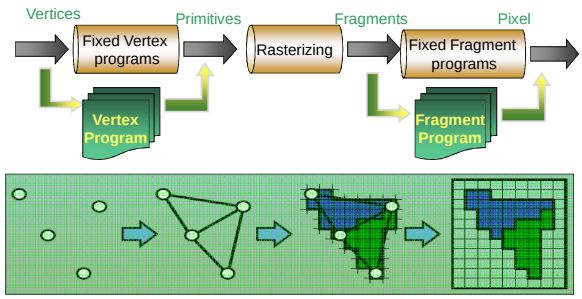
Traditional Graphics Pipeline



Programmable Graphics Pipeline



Programmable Graphics Pipeline



What Is Shader

- A simple program runs on a GPU to process a vertex, a pixel, or a geometric primitive
- There are three types of shaders
  - Vertex shader
    - for processing a single vertex
  - Pixel shader (or fragment shader)
    - for processing a single pixel
  - Geometry shader (DX10)
    - for processing a primitive or a primitive with adjacency information

Why Shader Programming

— fixed graphics pipeline

- Until the advent of programmable GPUs, all rendering pipelines were "fixed"
  - It automatically transforms each input vertex with various transformations, such as the world-to-view and the projection transformations
  - Then the vertices are assembled into polygons like triangles, which are then rasterized into fragments
  - All these operations are hard-coded in GPU
    - Limited number of algorithms
    - Limited functionality
    - Programmers have not much power to control the process once vertices are fed in the pipeline

Why Shader Programming

— programmable graphics pipeline

- With programmable GPUs, programmers can replace hard-coded functions with a shader
  - Allow programmers to do anything they want
  - Various shading techniques are readily achievable
  - New rendering ideas can be easily experimented and implemented as long as the underlying algorithms can be developed properly

Why Now?

- Programmable HW is available
- Various high level shader programming languages have been developed
  - more intuitive
  - easier to write
- Various shader IDEs have been developed by different HW venders and shader language developers
- GPU programming has becoming easier and easier

Where Can Shaders Be Used For?

- In 3D computer graphics, a shader is used to determine the final surface properties of an object or image
- This can include arbitrarily complex descriptions of
  - light absorption and diffusion,
  - texture mapping,
  - reflection and refraction,
  - shadowing,
  - surface displacement
  - and post-processing effects.
  - … …

How to Write a Shader

- Low-level
  - Assembly shader language
- High-level: C-like languages
  - CG:
    - http://www.nvidia.com/cg
  - HLSL
    - http://www.microsoft.com/windows/directx
  - GLSL
    - http://www.3dlabs.com/support/developer/ogl2/whitepapers/ index.html

What Is CG

- Cg, abbreviation of phrase “C for Graphics
- Broadly applicable across APIs and platforms (Windows, Linux, and Mac OS)
- A key enabler of cinematic computing
- Resourse:
  - Tutorial: http://developer.nvidia.com/CgTutorial
  - Demos and documents: http://developer.nvidia.com/page/cg_main.html

What Is HLSL

- HLSL: High Level Shading Language
- Part of DirectX 9 and Direct 10
- Syntactically, HLSL and Cg are almost identical
  - **Cg** will run on any GPU that supports programmable vertex or pixel processing
  - **HLSL** will run primarily on Windows platforms

A Simple Vertex Shader in HLSL (CG)

```
float4x4 matViewProj;

struct VS_IN
{
   float4 Pos : POSITION0;
};

struct VS_OUT
{
   float4 oPos : POSITION0;
};
```

```
VS_OUT vs_main(VS_IN inV)
{
  VS_OUT outV;

  outV.oPos
       = mul(matViewProj, inV.Pos);

  return(outV);
}
```

What Is GLSL

- GLSL: GL Shading Language
- Often referred as *glslang*
- Defined by the Architectural Review Board of OpenGL, the governing body of OpenGL
- Resources
  - Specification
    - http://www.opengl.org/documentation/glsl
  - Tutorial
    - http://www.lighthouse3d.com/opengl/glsl/

A Simple Pixel Shader in GLSL

```
varying vec2 Texcoord;

void main( void )
{
   gl_Position = ftransform();
   Texcoord    = gl_MultiTexCoord0.xy;
}
```

The Advantages of Using Higher Level Shader Programming Languages

- Implementation can be thought at algorithm level
- Need not worry about hardware details
- Also have usual advantages of a high level language
  - More intuitive
  - Easy code reuse
  - Improved readability

The Limitation of Vertex Shader and Pixel Shader

- Vertex shader
  - Vertices can neither be generated nor destroyed
  - Process vertices one by one, so
    - No information about topology or ordering of vertices is available
- Pixel shader
  - Fragments cannot be generated
  - Position of fragments cannot be changed
  - Access to neighboring fragments is not allowed
- Geometry shader
  - Not specified in GLSL

What To Be Learnt

- GLSL programming
  - How to integrate GLSL in an application will not be introduced
    - Refer to GLSL tutorial provided in the following webpage for detailed information http://www.lighthouse3d.com/opengl/glsl/
- How this part be assessed
  - ACW: 20%
    - A set of graphics written in GLSL and displayed in RenderMonkey. No c++ application required.
  - Exam: 25%

GLSL Tools

- Here are some commonly used GLSL shader programming IDE for developing graphics effects
  - Rendermonkey
    - http://ati.amd.com/developer/rendermonkey/
    - Used for our labs
  - Shader Designer
    - http://www.opengl.org/sdk/tools/ShaderDesigner/
  - Lumina
    - http://lumina.sourceforge.net/

RenderMonkey IDE: Key Features

- Created as a shader content creation tool
  - Allows easy shader prototyping and development
    - flexible and powerful for **programmers**
  - Artist Interface
    - familiar and intuitive for **artists**
  - A single environment
    - for managing the shaders and all of the associated visual resources

Sample Previous Students' ACW (2006)

Sample Previous Students' ACW (2008)

Sample Previous Students' ACW (2005)

Sample Previous Students' ACW(2007)