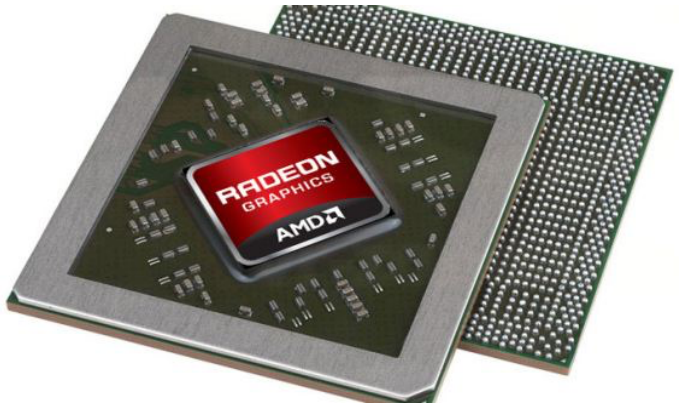


GPU, Shaders & OpenGL >= 3.2

CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

http://www.pling.org.uk/cs/cgv.html

nVidia GTX 280

Graphics Hardware

http://images.nvidia.com/products/geforce_gtx_280/GeForce_GTX_280_low_3qtr.png

Performance

Limited by fragment drawing rate

#fragments / second

= |Image| x FPS x depth complexity

1440 x 900 x 60 x 4 ~= 297 Megafragments/sec

Note: depth complexity is average "overdraw"

bandwidth

= #fragment/second * #bytes/fragment

297 Megafragments/sec * 4 ~= 1.2 GB/ s

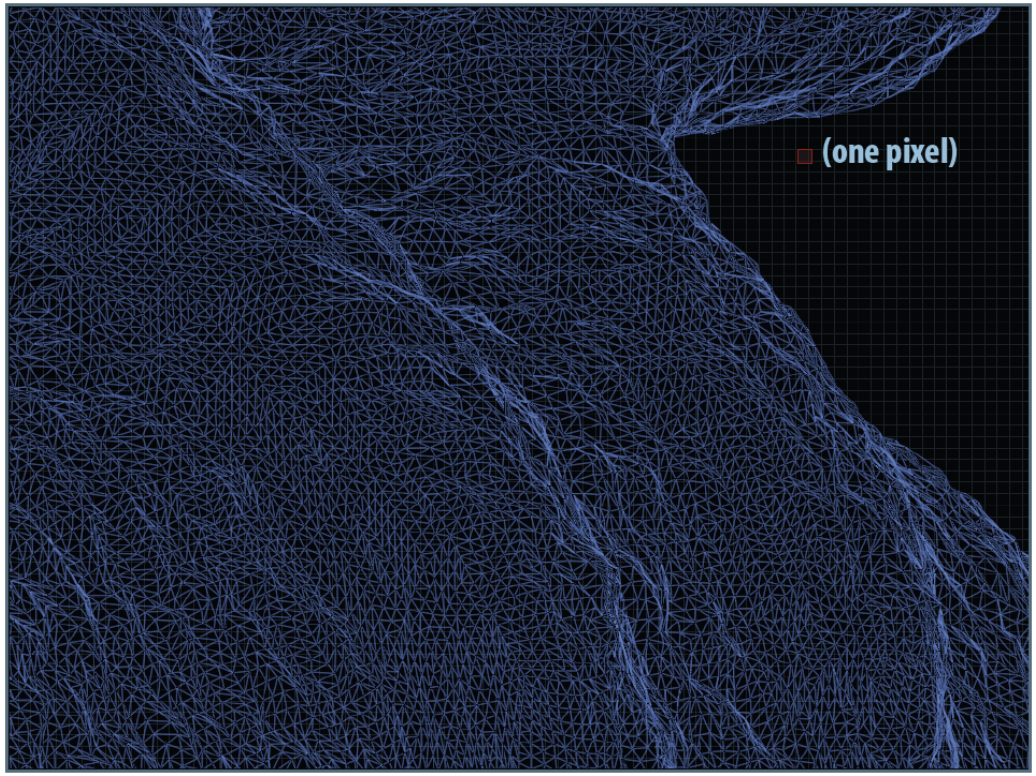
Performance

#triangles/sec
= (#fragments/sec) / avg(|triangle|)

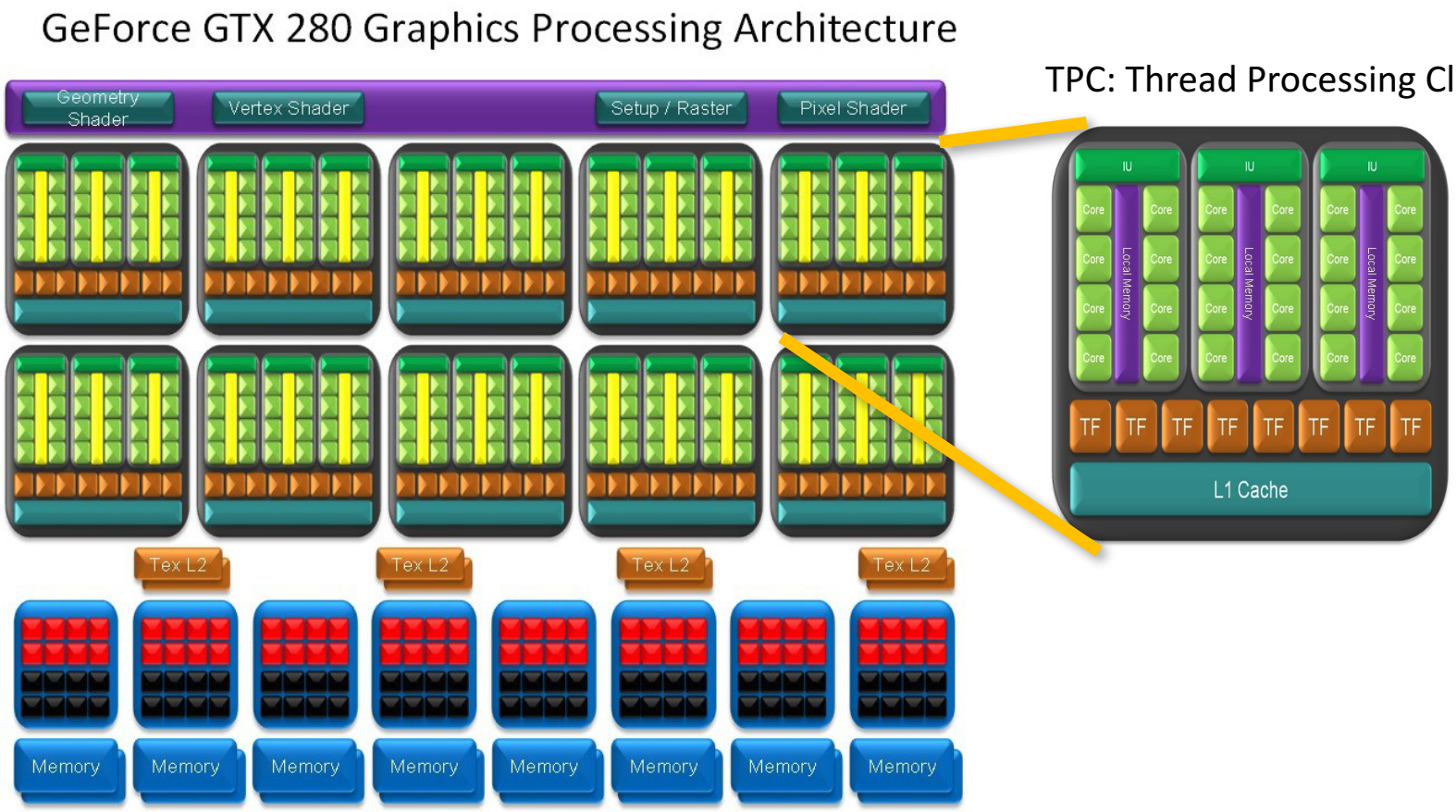
Triangles are small!

assuming 16 pixels/triangle

~18.6 million triangles/sec

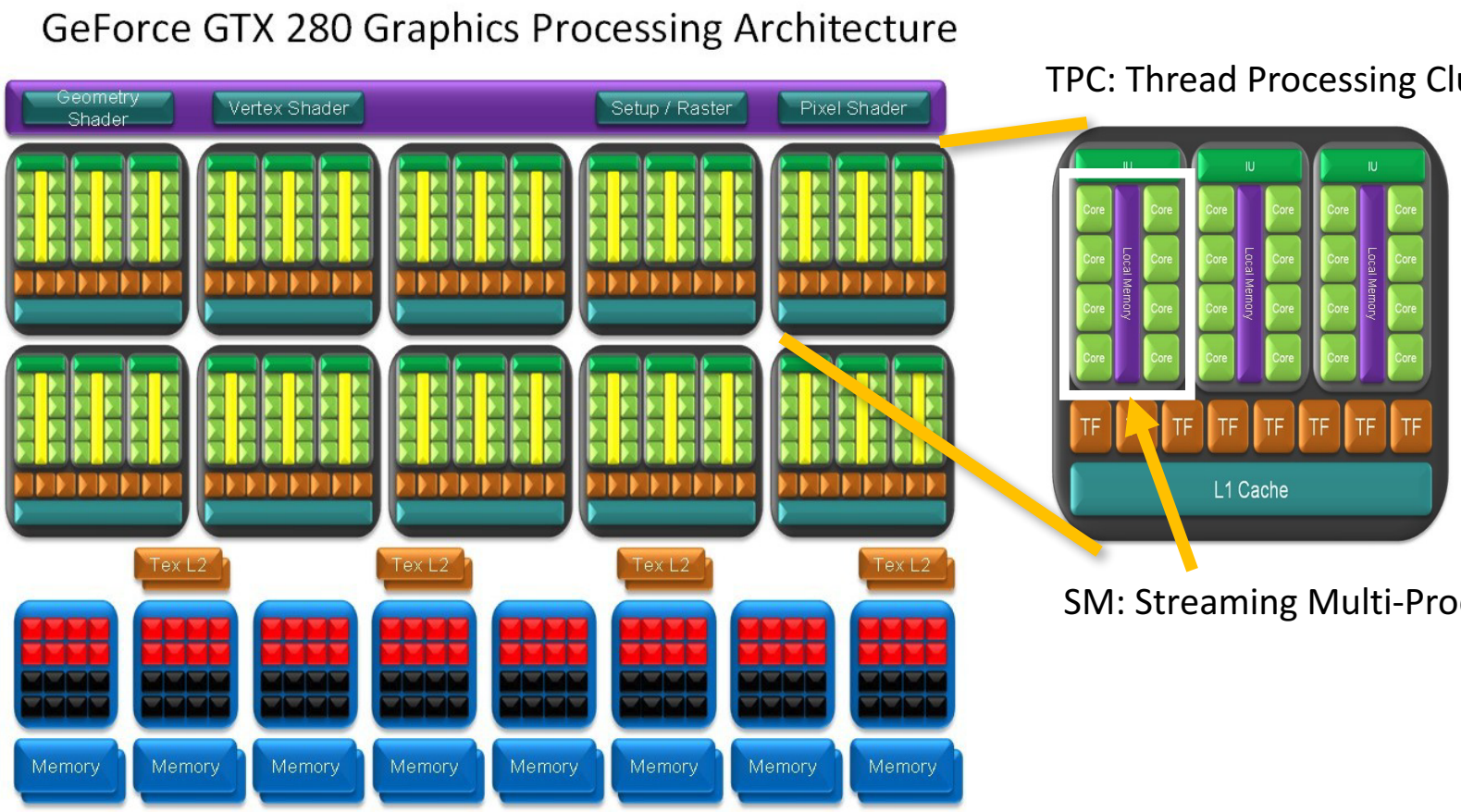


Modern GPUs



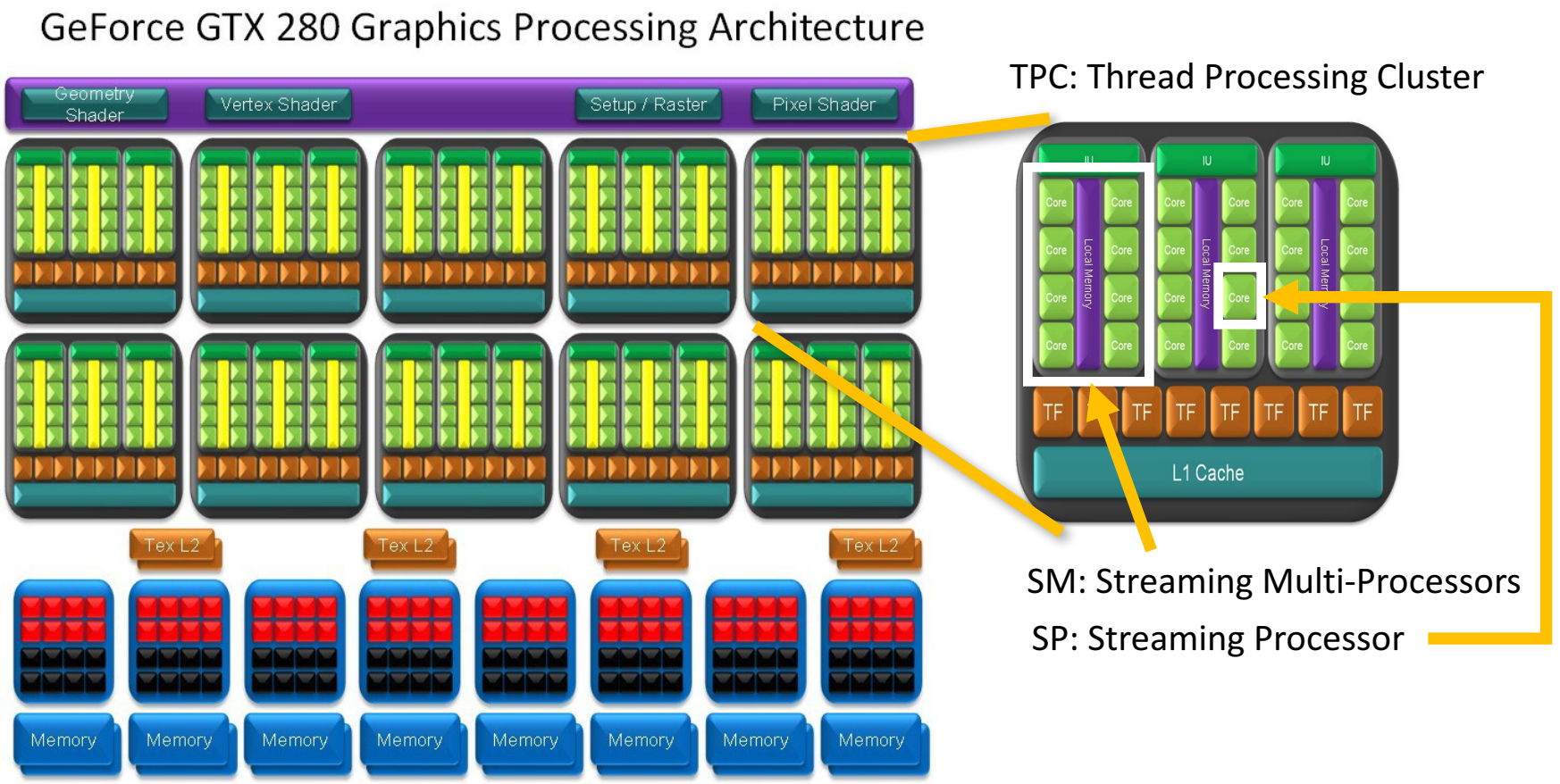
Geforce GTX 200 Technical Brief by nVidia

Modern GPUs



Geforce GTX 200 Technical Brief by nVidia

Modern GPUs



Unified Architecture!

Geforce GTX 200 Technical Brief by nVidia

Modern GPUs

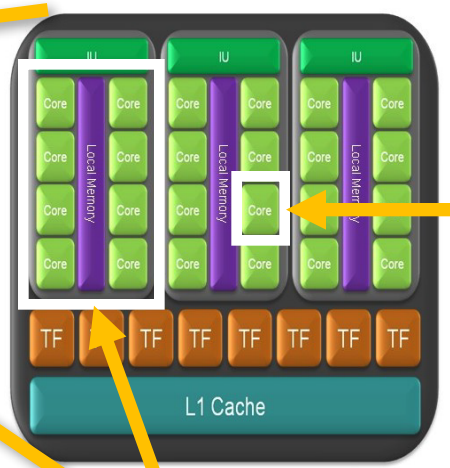
Specifications

Note: The below specifications represent this GPU as incorporated into NVIDIA's
Clock specifications apply while gaming with medium to full GPU utilization. Gra
vary by Add-in-card manufacturer. Please refer to the Add-in-card manufacturer
specifications.

GPU Engine Specs	
CUDA Cores[1]	240
Graphics Clock (MHz)	602MHz
Processor Clock (MHz)	1296MHz
Texture Fill Rate (billion/sec)	48.2
Graphics Performance	??
Memory Specs	
Memory Clock	1107MHz
Standard Memory Config	1GB
Memory Interface Width[5]	512-bit
Memory Bandwidth (GB/sec)	141.7
Feature Support	
OpenGL	2.1
Supported Technologies[3]	3D Vision, PhysX, CUDA, SLI
SLI Options[4]	2-way
	3-way

Unified Architecture!

TPC: Thread Processing Cluster



SM: Streaming Multi-Processor

SP: Streaming Processor

10 TPCs.
240 SPs (cores) at 1296 MHz
1024 Threads per SM.

Geforce GTX 200 Technical Brief by nVidia

Key Differences: GPU vs CPU

NV GTX 1060	Intel i7-4790K	Intel Xeon E5-2699
Cores: 1280	Cores: 4 (8 Threads)	Cores: 18 (36 Threads)
Clock: 1.5 – 1.7 GHz	Clock: 4 – 4.4 GHz	Clock: 2.3 – 3.6 GHz.
Power: 400 W (120W)	Power: 88W	Power: 145W
Mem BW: 192 GB/sec	Mem BW: 25.6 GB/sec	Memory BW: 68 GB/sec
Many Simple Cores	**Few Complex Cores**	
GPU	CPU	

Key Differences: GPU vs CPU

NV GTX 1060	Intel i7-4790K	Intel Xeon E5-2699
Cores: 1280	Cores: 4 (8 Threads)	Cores: 18 (36 Threads)
Clock: 1.5 – 1.7 GHz	Clock: 4 – 4.4 GHz	Clock: 2.3 – 3.6 GHz.
Power: 400 W (120W)	Power: 88W	Power: 145W
Mem BW: 192 GB/sec	Mem BW: 25.6 GB/sec	Memory BW: 68 GB/sec
Many Simple Cores	**Few Complex Cores**	
Slower Clock	**Fast Clock**	
GPU	CPU	

Key Differences: GPU vs CPU

NV GTX 1060	Intel i7-4790K	Intel Xeon E5-2699
Cores: 1280	Cores: 4 (8 Threads)	Cores: 18 (36 Threads)
Clock: 1.5 – 1.7 GHz	Clock: 4 – 4.4 GHz	Clock: 2.3 – 3.6 GHz.
Power: 400 W (120W)	Power: 88W	Power: 145W
Mem BW: 192 GB/sec	Mem BW: 25.6 GB/sec	Memory BW: 68 GB/sec
Many Simple Cores	**Few Complex Cores**	
Slower Clock	**Fast Clock**	
Power Hungry	**Power Efficient**	
GPU	CPU	

Key Differences: GPU vs CPU

NV GTX 1060	Intel i7-4790K	Intel Xeon E5-2699
Cores: 1280	Cores: 4 (8 Threads)	Cores: 18 (36 Threads)
Clock: 1.5 – 1.7 GHz	Clock: 4 – 4.4 GHz	Clock: 2.3 – 3.6 GHz.
Power: 400 W (120W)	Power: 88W	Power: 145W
Mem BW: 192 GB/sec	Mem BW: 25.6 GB/sec	Memory BW: 68 GB/sec
Many Simple Cores	**Few Complex Cores**	
Slower Clock	**Fast Clock**	
Power Hungry	**Power Efficient**	
High Mem Bandwidth	**Lower Mem Bandwidth**	
GPU	CPU	

Key Differences: GPU vs CPU

NV GTX 1060

Cores: 1280

Clock: 1.5 – 1.7 GHz

Power: 400 W (120W)

Mem BW: 192 GB/sec

Many Simple Cores

Slower Clock

Power Hungry

High Mem Bandwidth

High-Throughput

GPU

Intel i7-4790K

Cores: 4 (8 Threads)

Clock: 4 – 4.4 GHz

Power: 88W

Mem BW: 25.6 GB/sec

Intel Xeon E5-2699

Cores: 18 (36 Threads)

Clock: 2.3 – 3.6 GHz.

Power: 145W

Memory BW: 68 GB/sec

Few Complex Cores

Fast Clock

Power Efficient

Lower Mem Bandwidth

Low Latency

CPU

Key Differences: GPU vs CPU

NV GTX 1060

Cores: 1280

Clock: 1.5 – 1.7 GHz

Power: 400 W (120W)

Mem BW: 192 GB/sec

Many

Slow

Powe

High Mem Bandwidth

High-Throughput

GPU

Intel i7-4790K

Cores: 4 (8 Threads)

Clock: 4 – 4.4 GHz

Power: 88W

Intel Xeon E5-2699

Cores: 18 (36 Threads)

Clock: 2.2 – GHz.

B/sec

Fast Clock

Power Efficient

Lower Mem Bandwidth

Low Latency

CPU

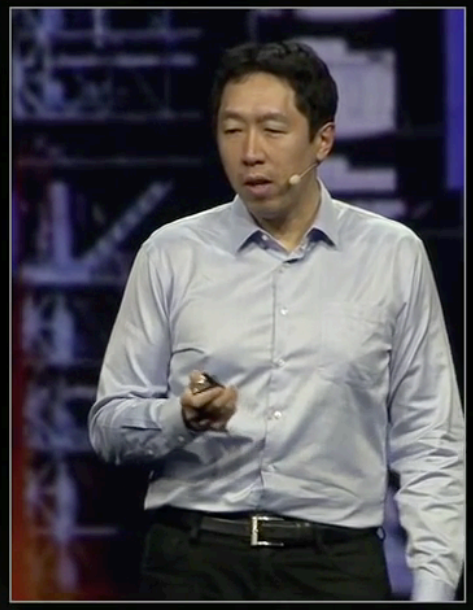
Usually for GPUs

Write Speed >> Read Speed

GPGPU

- CUDA
  - Only on nVidia hardware
  - Mature utility APIs and Tools
- OpenCL
  - Open Standard
  - Device and OS independent
  - Works across GPUs and CPUs from multiple vendors
- DirectCompute (Microsoft)
  - DirectX 10 and higher, Windows based

GPGPU



Why is Deep Learning taking off?

Baidu Research

Andrew Ng

Super Computers

Titan: Oakridge National Laboratory

Each of Titan's 18,688 nodes contains an NVIDIA Tesla K20 GPU

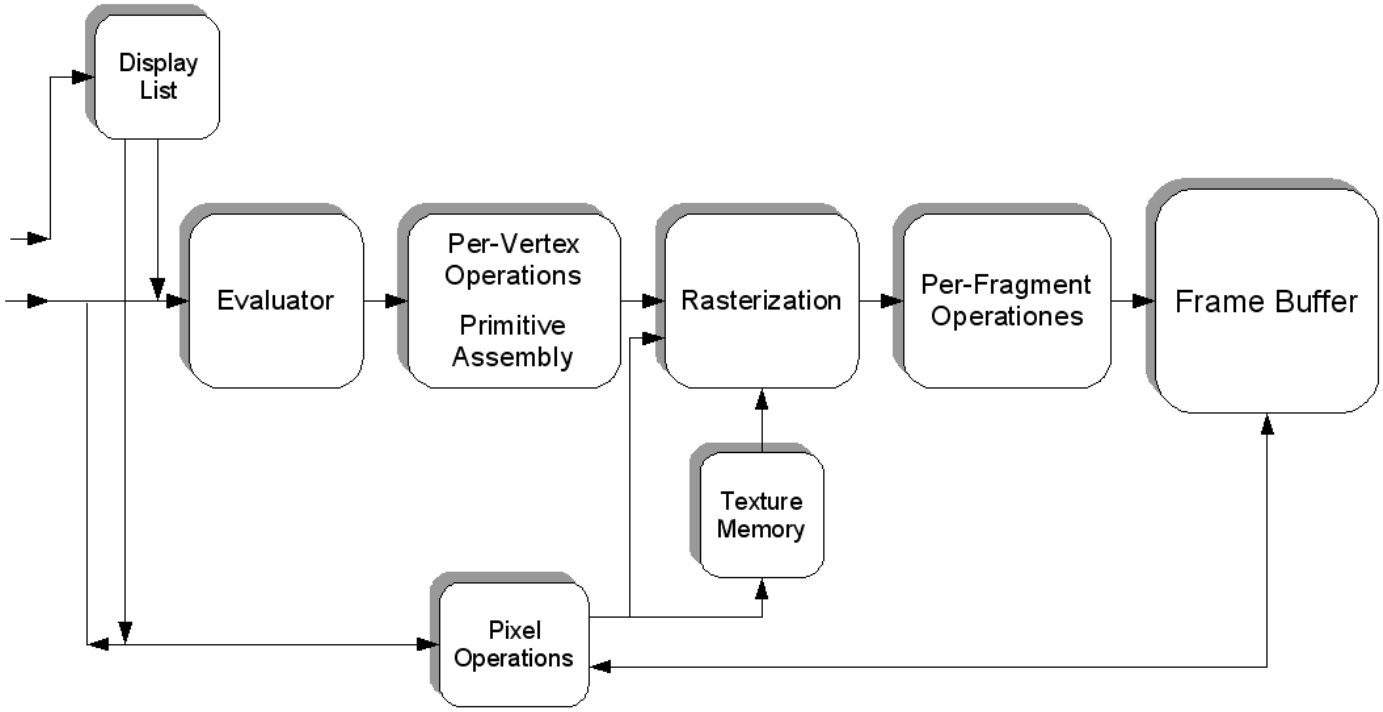
http://www.top500.org/featured/top-systems/titan-oak-ridge-national-laboratory/

OpenGL >= 3.2

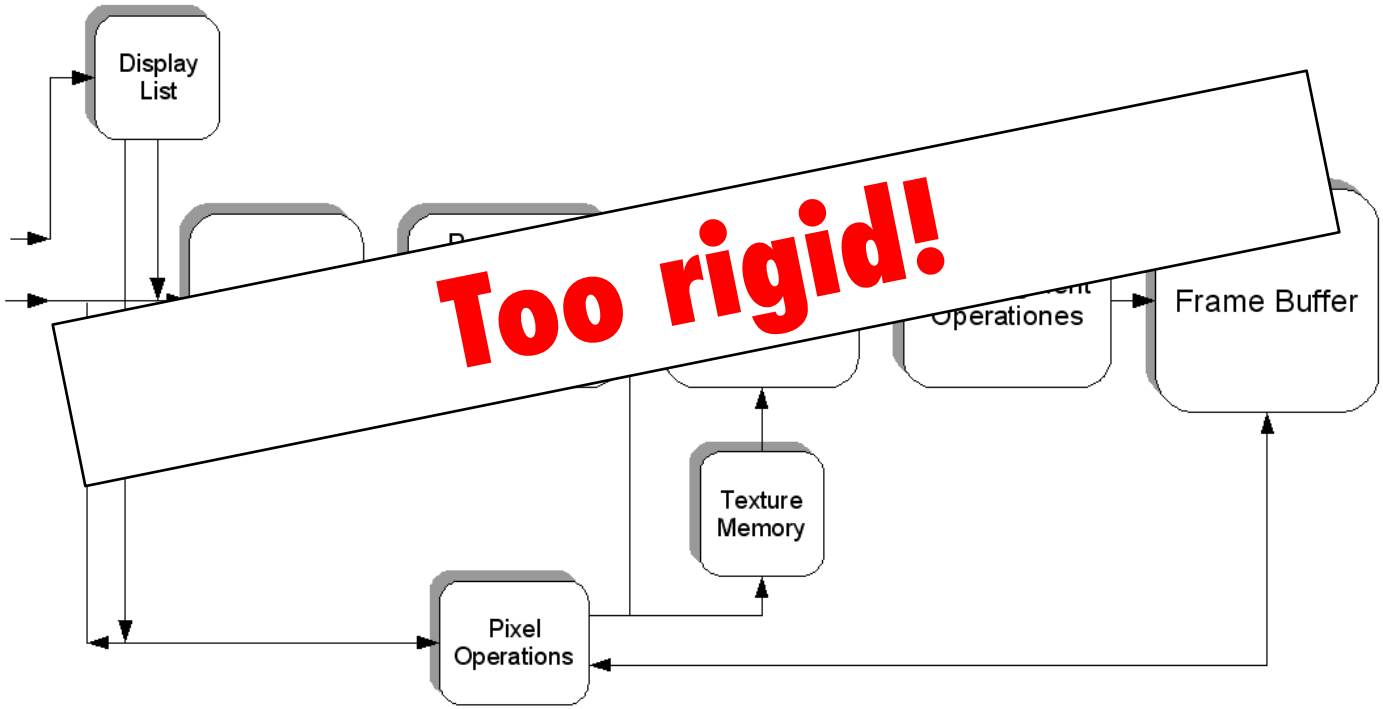
Reference Card:

https://www.khronos.org/files/opengl-quick-reference-card.pdf

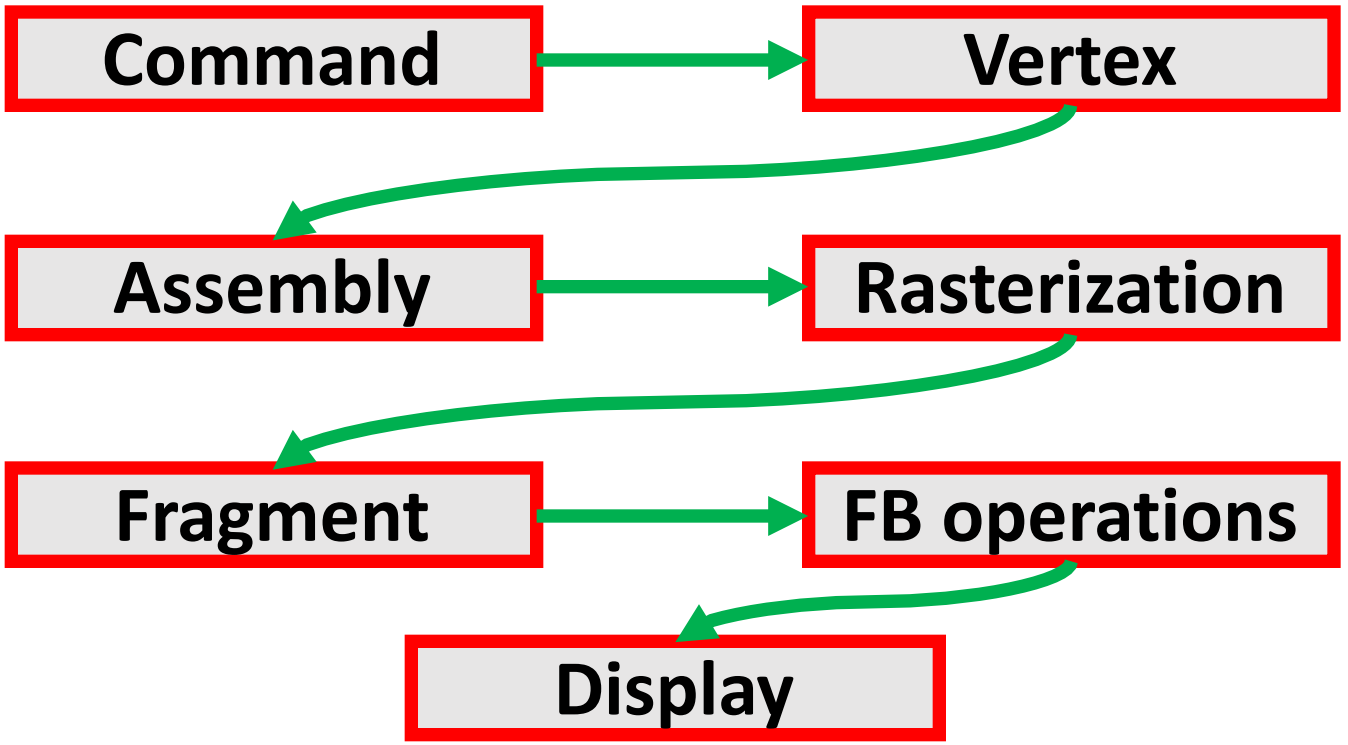
Fixed-Function Pipeline



Fixed-Function Pipeline

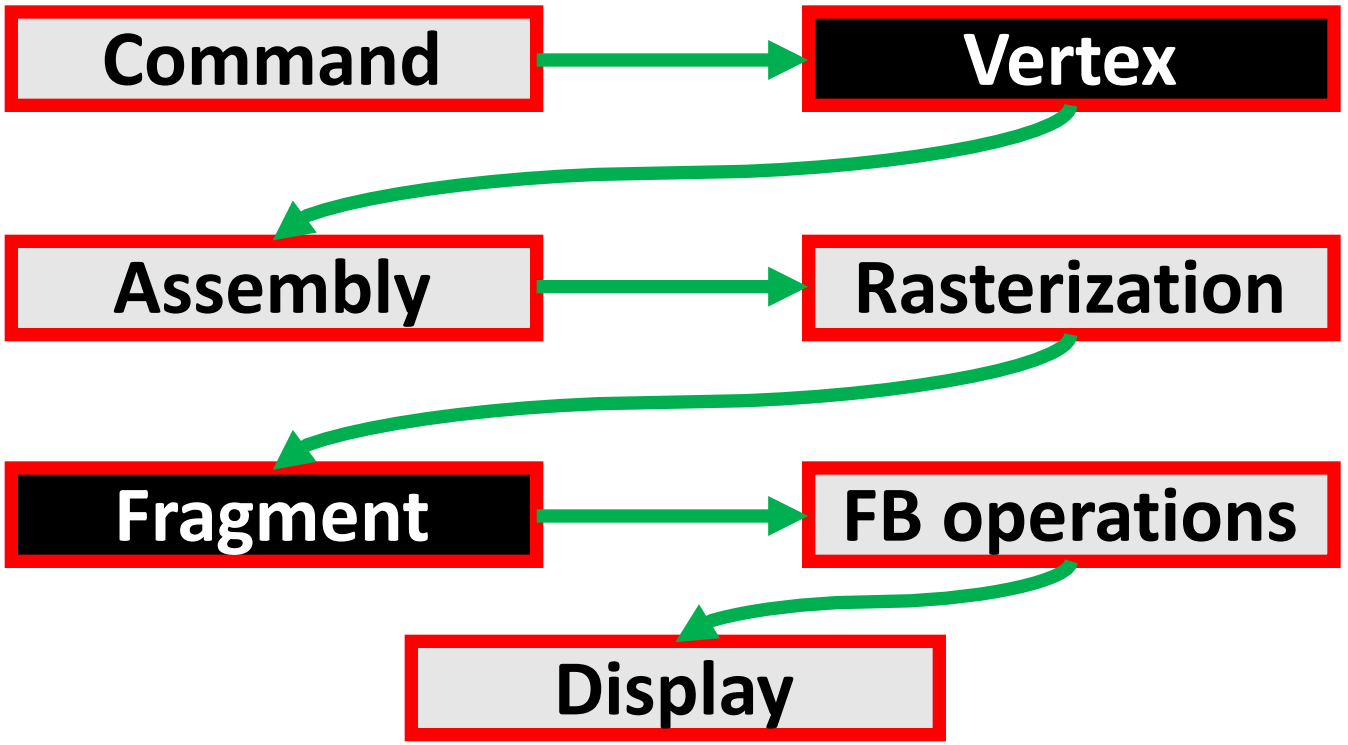


Simplistic Fixed Pipeline



http://graphics.stanford.edu/courses/cs148-11-fall/lectures/programmable.pdf

Simplistic Programmable Pipeline > 2.0

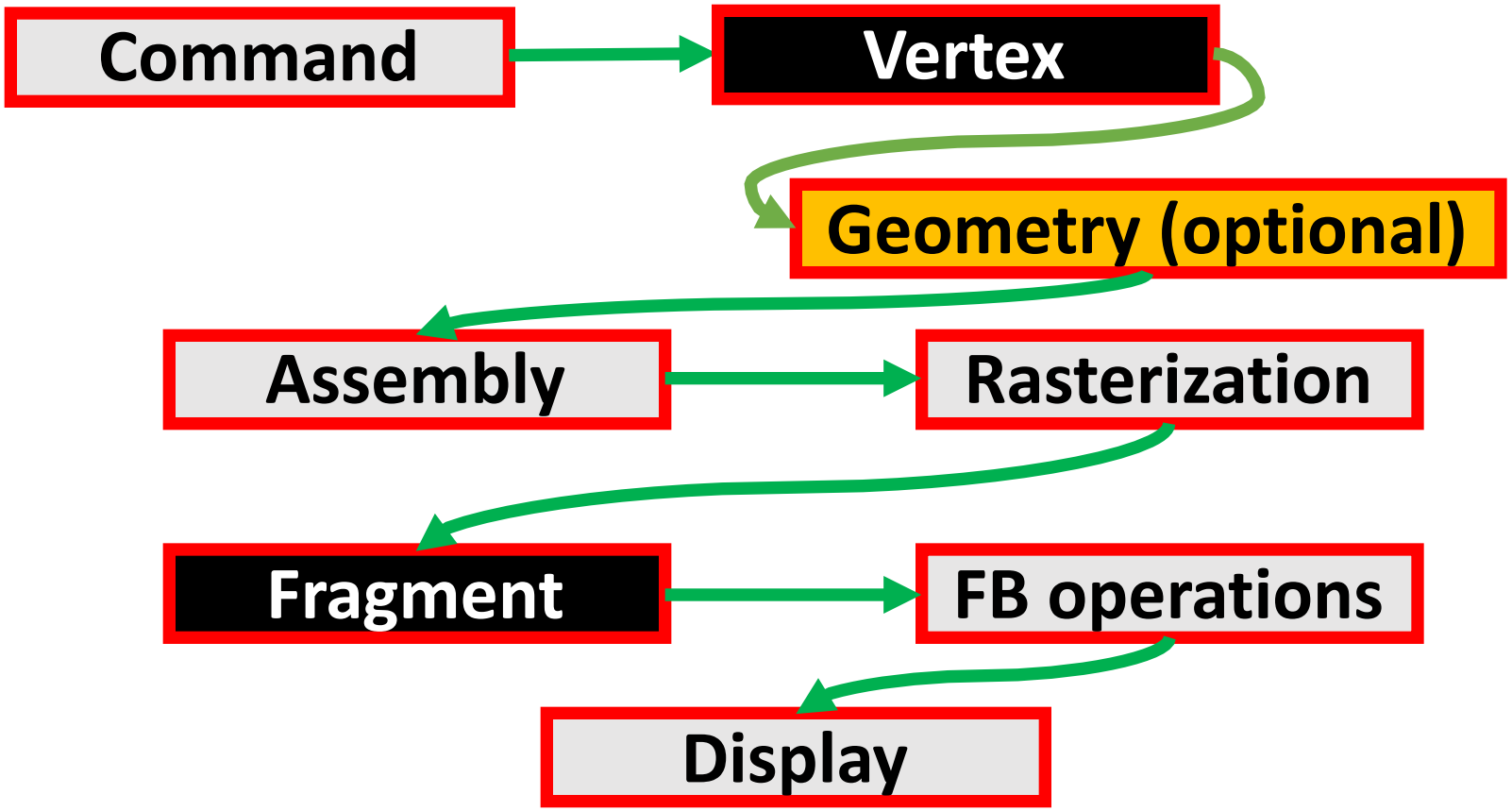


Introducing “Shaders”

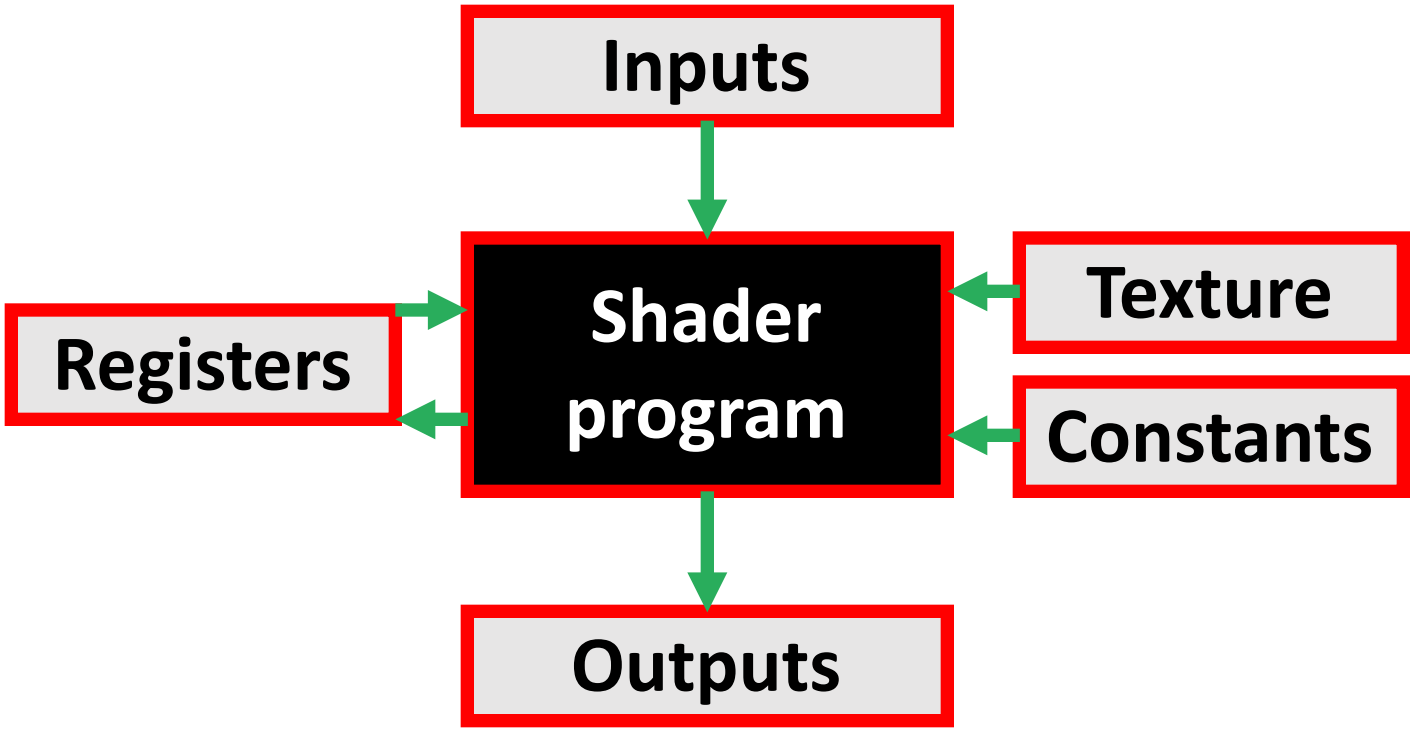
Short program customizing a part of the graphics pipeline.

Misnomer: They do more than just shade !

Simplistic Programmable Pipeline > 3.2

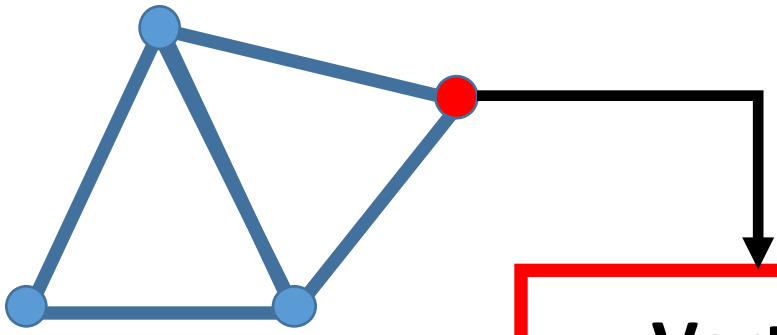


Shader Architecture



http://graphics.stanford.edu/courses/cs148-11-fall/lectures/programmable.pdf

Vertex Program/Shader



Input: Single Vertex with attributes, e.g.

- Position
- Normal
- Texture Coordinate
- Other programmer defined attribs

Vertex program

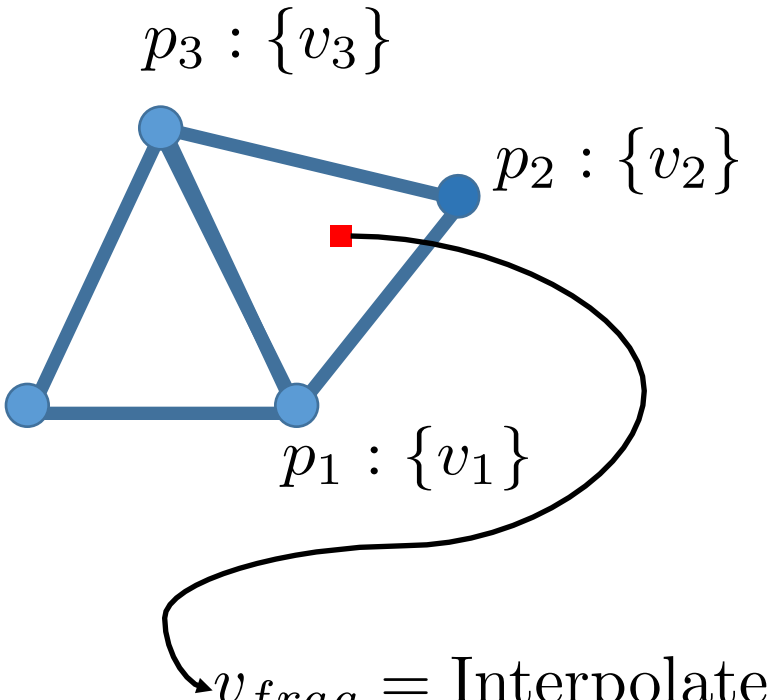
Input: Uniforms - remain constant during the entire draw call, e.g.

- Model matrix
- View/Camera matrix
- Projection matrix
- Other programmer defined parameters

Output:

- Transformed positions (NDC)
- Point size (optional)
- Other programmer defined **varying** data (input for Fragment program)

“Varying” output



Output attributes (marked with the keyword “**out**”) of the *Vertex Program* are interpolated by the hardware during the rasterization process and set as an input attributed for the *Fragment Program* (see later).

Historically called **“varying”** attributes.

$$v_{frag} = \text{Interpolate}_{p_1,p_2,p_3}\left(v_1, v_2, v_3\right)$$

$$p_i : \text{Transformed positions}$$

$$v_j : \textbf{varying} \text{ output attributes}$$

A Simple Vertex Program

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

A Simple Vertex Program

declare GLSL version 3.3 – matches with OpenGL version

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

A Simple Vertex Program

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

Uniforms – remains constant during the entire draw call

A Simple Vertex Program

Per vertex attributes – position, color, normal, etc.

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

A Simple Vertex Program

Varying output ("out" keyword)– will be interpolated and fed as an input attribute to the *Fragment Program.* There can be many such "out" attributes.

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

A Simple Vertex Program

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPos                    ute position 0
layout (location = 1) in vec3 vCol                    position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

”gl_Position” must contain the transformed position in NDC.

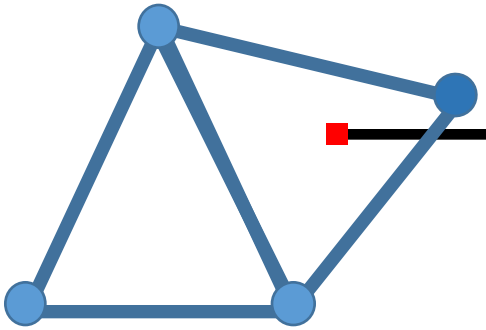
Note: Use of vec4 to extend 3D to 4D homogeneous coordinate.

A Simple Vertex Program

Set the varying output attribute. For now just passing on the color attribute from the input. You can do whatever you want though.

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

Fragment Program/Shader



Input: Single rasterized fragment with interpolated "out" attributes from the *Fragment Program*.

Fragment Program

Input: Uniforms - remain constant during the entire draw call, e.g.

- Material properties
- Textures (1D, 2D, 3D)
- Light positions
- etc - whatever else you wish

Output:

- Final color (with maybe Alpha)
- Depth value (optional)
- **Discard** – discard a fragment from further processing

A Simple Fragment Program

Varying ("out") data from the Vertex Program

```
#version 330 core

in vec3 outColor;
in vec2 outTexCoord;

out vec4 finalColor;

uniform InputMaterial {
    vec4  matDiffuse;
    vec4  matSpecular;
    float matShininess;
    vec4  matAmbient;
} material;

uniform sampler2D diffuseTexture;

void main()
{
    finalColor = vec4(outColor, 1.0f) *
                 material.matDiffuse  *
                 texture(diffuseTexture,outTexCoord);
}
```

A Simple Fragment Program

The final output color

```
#version 330 core

in vec3 outColor;
in vec2 outTexCoord;

out vec4 finalColor;

uniform InputMaterial {
    vec4  matDiffuse;
    vec4  matSpecular;
    float matShininess;
    vec4  matAmbient;
} material;

uniform sampler2D diffuseTexture;

void main()
{
    finalColor = vec4(outColor, 1.0f) *
                 material.matDiffuse  *
                 texture(diffuseTexture,outTexCoord);
}
```

A Simple Fragment Program

```
#version 330 core

in vec3 outColor;
in vec2 outTexCoord;

out vec4 finalColor;

uniform InputMaterial {
    vec4  matDiffuse;
    vec4  matSpecular;
    float matShininess;
    vec4  matAmbient;
} material;

uniform sampler2D diffuseTexture;

void main()
{
    finalColor = vec4(outColor, 1.0f) *
                 material.matDiffuse  *
                 texture(diffuseTexture,outTexCoord);
}
```

Can use C struct like syntax

A Simple Fragment Program

```
#version 330 core

in vec3 outColor;
in vec2 outTexCoord;

out vec4 finalColor;

uniform InputMaterial {
    vec4  matDiffuse;
    vec4  matSpecular;
    float matShininess;
    vec4  matAmbient;
} material;

uniform sampler2D diffuseTexture;

void main()
{
    finalColor = vec4(outColor, 1.0f) *
                 material.matDiffuse  *
                 texture(diffuseTexture,outTexCoord);
}
```

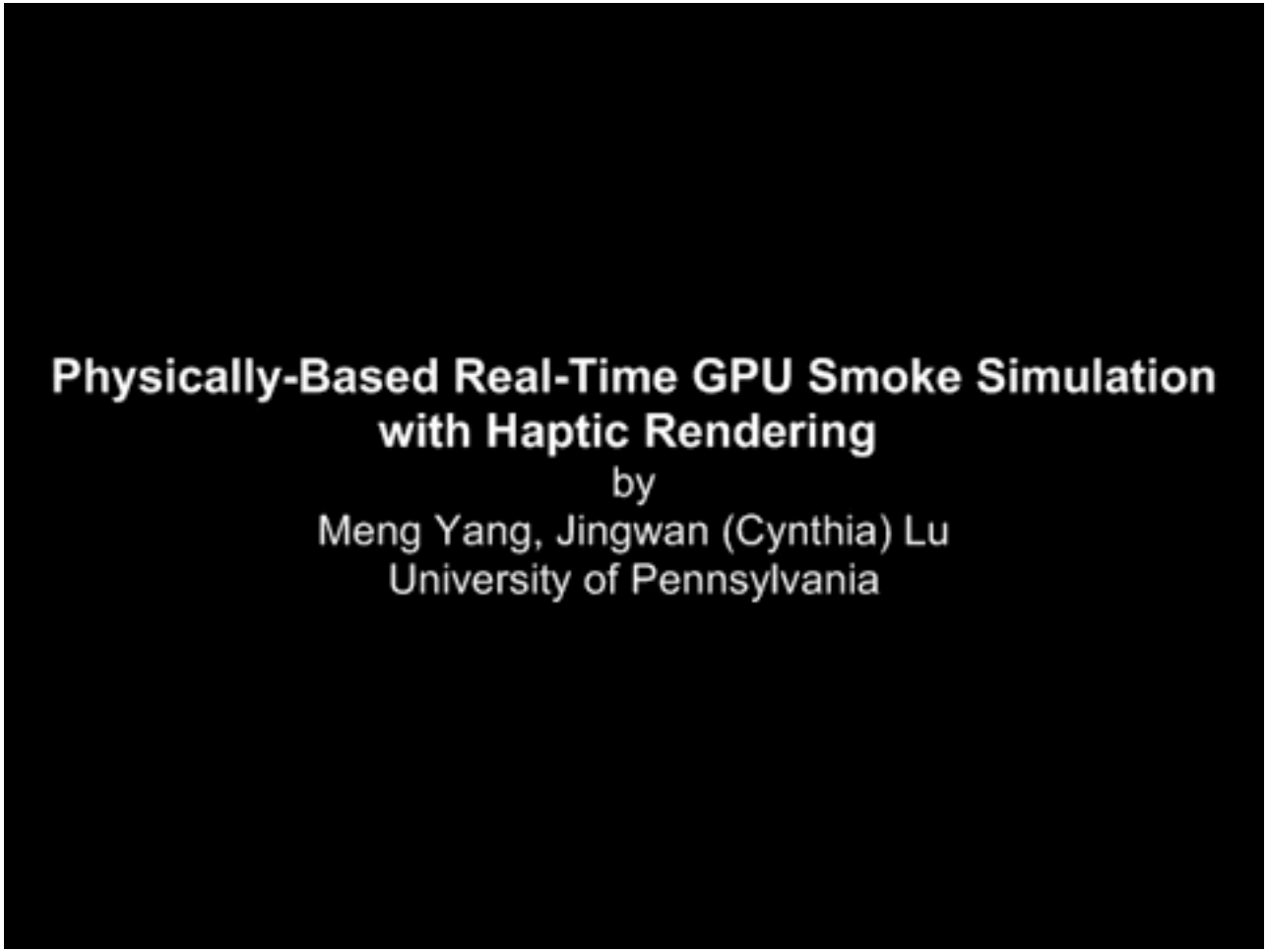
Uniform for a texture (in this case a 2D texture)

A Simple Fragment Program

```
#version 330 core

in vec3 outColor;
in vec2 outTexCoord;

out vec4 finalColor;

uniform InputMaterial {
    vec4  matDiffuse
    vec4  matSpecular
    float matShininess
    vec4  matAmbient
} material;

uniform sampler2D diffuseTexture;

void main()
{
    finalColor = vec4(outColor, 1.0f) *
                 material.matDiffuse  *
                 texture(diffuseTexture,outTexCoord);
}
```

Compute the final color. Can get arbitrarily complex. Note that the multiplications are "component wise."

Shader Demo


Physically-Based Real-Time GPU Smoke Simulation
with Haptic Rendering
by
Meng Yang, Jingwan (Cynthia) Lu
University of Pennsylvania

More References and Tutorials

- http://learnopengl.com
- OpenGL 3.3 Quick Reference Card: https://www.khronos.org/files/opengl-quick-reference-card.pdf

Shader Considerations

- Single-precision arithmetic
  - > GTX 200 supports Double-precision
- Branching, loops expensive
- No access to neighboring fragments/vertices

Shader Considerations

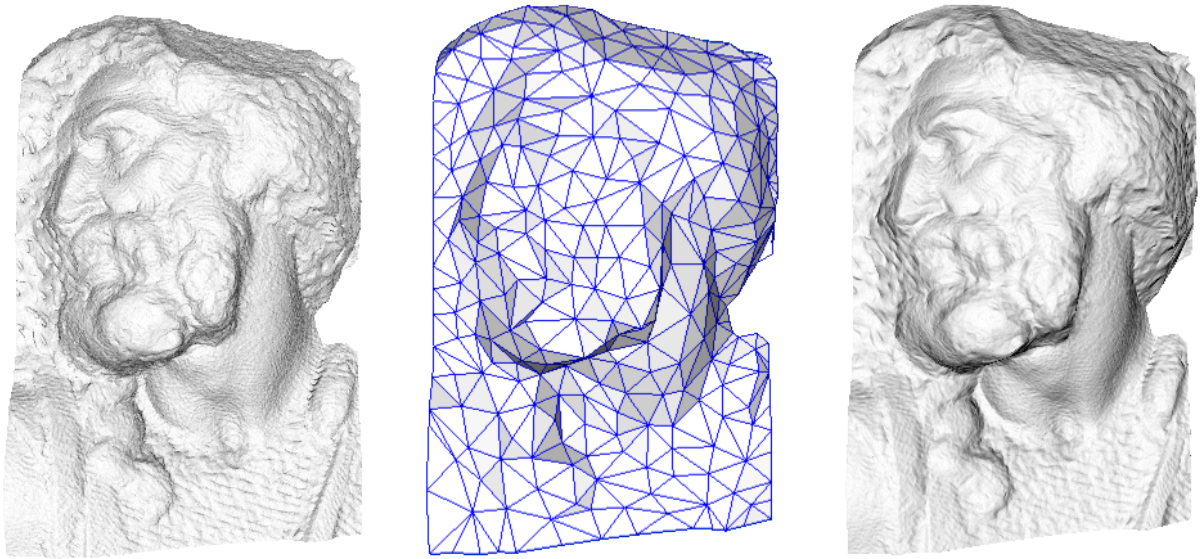
- Limited stack/instruction count
- Timeout possibility
- Support, debugging not consistent

Shader Language Options

- **Assembly**
  Issue commands directly to GPU
- **GLSL**
  OpenGL
- **Cg**
  OpenGL/Direct3D
- **HLSL**
  Direct3D

Common Shader Tasks

Bump/Normal Mapping

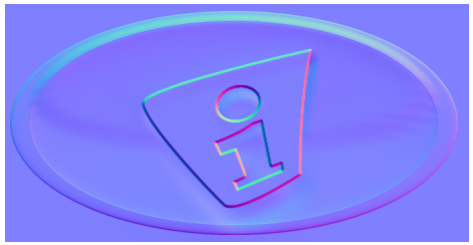
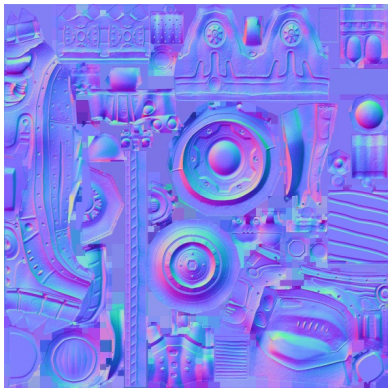
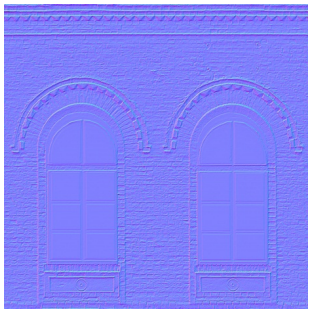
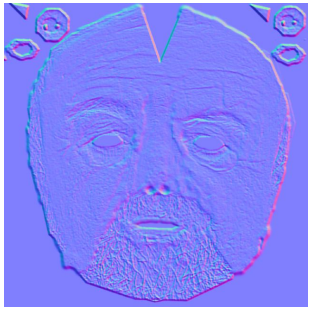
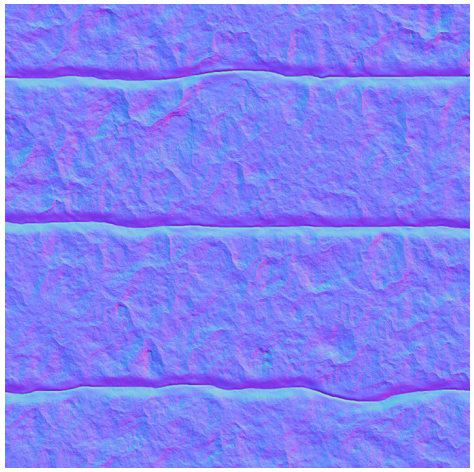


4M triangles

500 triangles

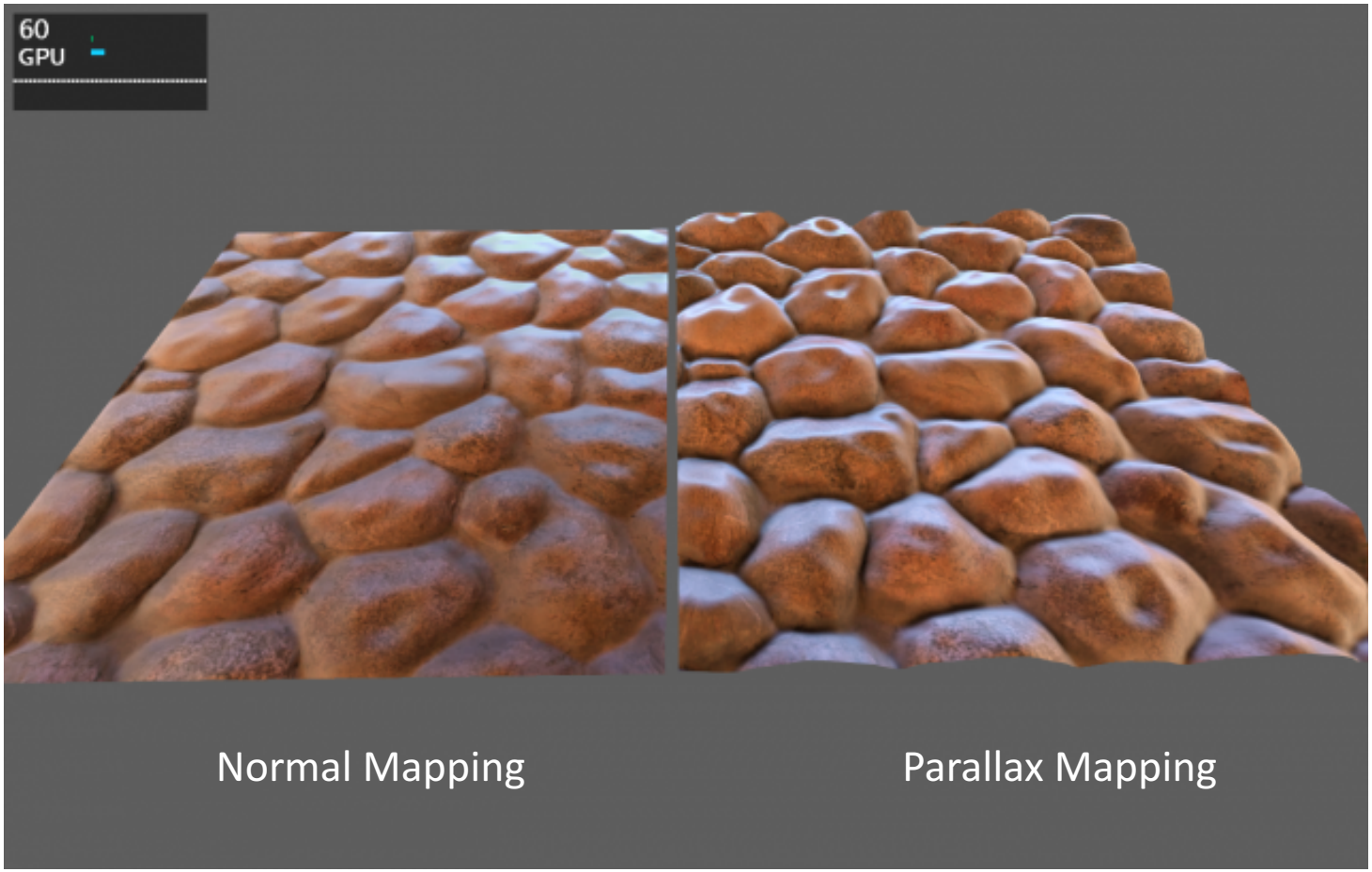
http://upload.wikimedia.org/wikipedia/commons/3/36/Normal_map_example.png

Bump/Normal Mapping

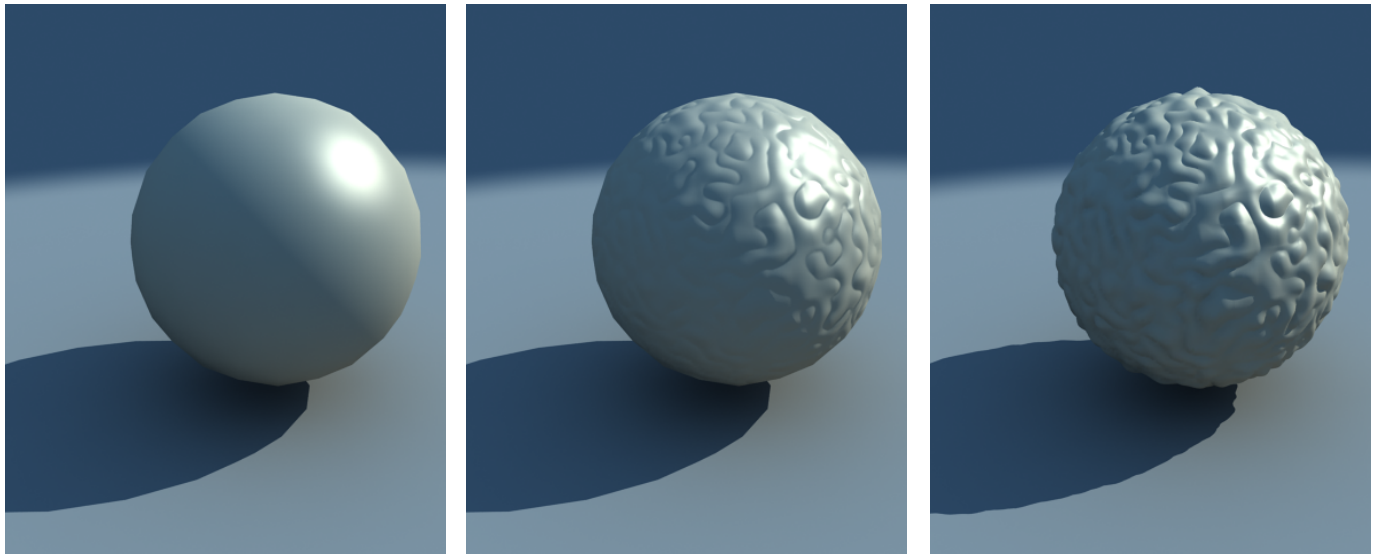
http://www.kxcad.net/lightwave/lightwave_3d_9/normalmap_normal.png
http://charhut.info/files/cs280/CliffNormal.png
http://imageshack.us/photo/my-images/412/gahumanefacenorce9.png/sr=1
http://users.tkk.fi/~mliukka/textures/Brickwall_windows_01_pom_ddn.jpg
http://www.ericspitler.com/images/2d/normalmap.jpg

Parallax Mapping



https://vvvv.org/sites/default/files/screenshot1382763194.png

Displacement Mapping



Original **Bump** **Displacement**

http://www.spot3d.com/vray/help/150SP1/tutorials_displacement.htm

Complex Material & Lighting

http://images.digitalmedianet.com/2006/Week_28/y0utm5jb/story/10c.jpg

Toon Shading

Use small set of colors
(maybe with 1D texure as a color palette)

http://www.cse.unr.edu/~mahsman/courses/cs791a/full.png

2D Image Processing

http://groups.csail.mit.edu/graphics/bilagrid/bilagrid_web.pdf

Simulation

http://www.geeks3d.com/20080812/nvidia-physx-powerpack-download/
http://pcper.com/images/reviews/245/graw_physx_3.jpg

Compiling Shaders

Shader Program

```
// Create and compile a Vertex Shader
GLuint vertexShader;
vertexShader = glCreateShader(GL_VERTEX_SHADER);

char *vertexShaderSource = loadVertexShaderFile(...);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);

// Similarly for Fragment Shader
GLuint fragmentShader;
fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);

// Create and link a shader program
GLuint shaderProgram;
shaderProgram = glCreateProgram();

glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);

// Once linked we don't need the shaders anymore
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);

glUseProgram(shaderProgram); // Enable before rendering
/*****************************
// Render something
*****************************/
glUseProgram(0); // Disable after rendering

// Destroy a shader program when completely done
glDeleteProgram(shaderProgram)
```

Shader Program – Vertex Shader

```
// Create and compile a Vertex Shader
GLuint vertexShader;
vertexShader = glCreateShader(GL_VERTEX_SHADER);

char *vertexShaderSource = loadVertexShaderFile(...);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
```

Shader Program – Fragment Shader

```
// Similarly for Fragment Shader
GLuint fragmentShader;
fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);
```

Shader Program - Linking

```
// Create and link a shader program
GLuint shaderProgram;
shaderProgram = glCreateProgram();

glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);

// Once linked we don't need the shaders anymore
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);
```

Shader Program - Running

```
// Create and compile vertex shader
GLuint vertexShader;
vertexShader = glCreateShader(GL_VERTEX_SHADER);

char *vertexShaderSource = loadVertexShaderFile(...);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);

// Similarly for Fragment Shader
GLuint fragmentShader;
fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);

// Create and link a shader program
GLuint shaderProgram;
shaderProgram = glCreateProgram();

glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);

// Once linked we don't need the shaders anymore
glDeleteShader(vertexShader);
...
glDeleteProgram(shaderProgram);
```

```
glUseProgram(shaderProgram); // Enable before rendering
/*****************************
// Render something
*****************************/
glUseProgram(0); // Disable after rendering
```

Passing Data to Shaders

Passing “Uniforms”

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

Some News!

- No *glPushMatrix/glPopMatrix/glLoadMatrix* etc
- Matrices are like any other “Uniforms”
- Manage your own matrices on the CPU side
  - Use third party libraries – e.g. glm
  - Matrix Format – Column Major Order

Passing “Uniform” - Matrix

```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition;
layout (location = 1) in vec3 vColor;
```

```
float matModel [16] = {...};
GLint matModelLocation = glGetUniformLocation(shaderProgram, "matModel");
glUniformMatrix4fv(matModelLocation, 1, GL_FALSE, matModel);
```

Similary:
glUniform{1,2,3,4}{if}{v},
glUniformMatrix{2,3,4}fv
… and many more

Passing Vertex Data

Vertex Data/Attributes

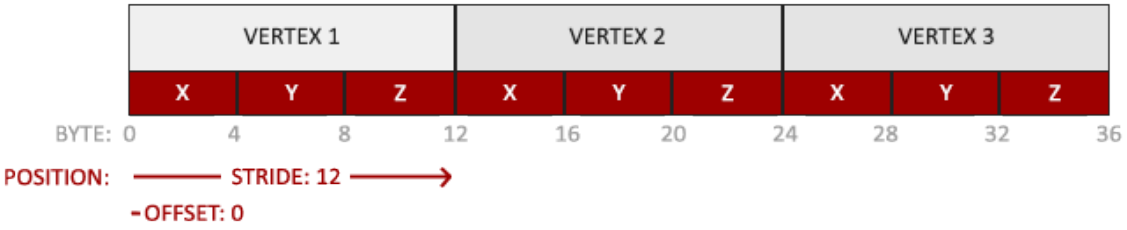
```
#version 330 core

uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; // The position variable has attribute position 0
layout (location = 1) in vec3 vColor;    // The color variable has attribute position 1

out vec3 outColor; // Output a color to the fragment shader

void main()
{
    gl_Position = matProj * matView * matModel * vec4(vPosition, 1.0);
    outColor = vColor; // Set outColor to the input color we got from the vertex data
}
```

Vertex Buffer Object (VBO)

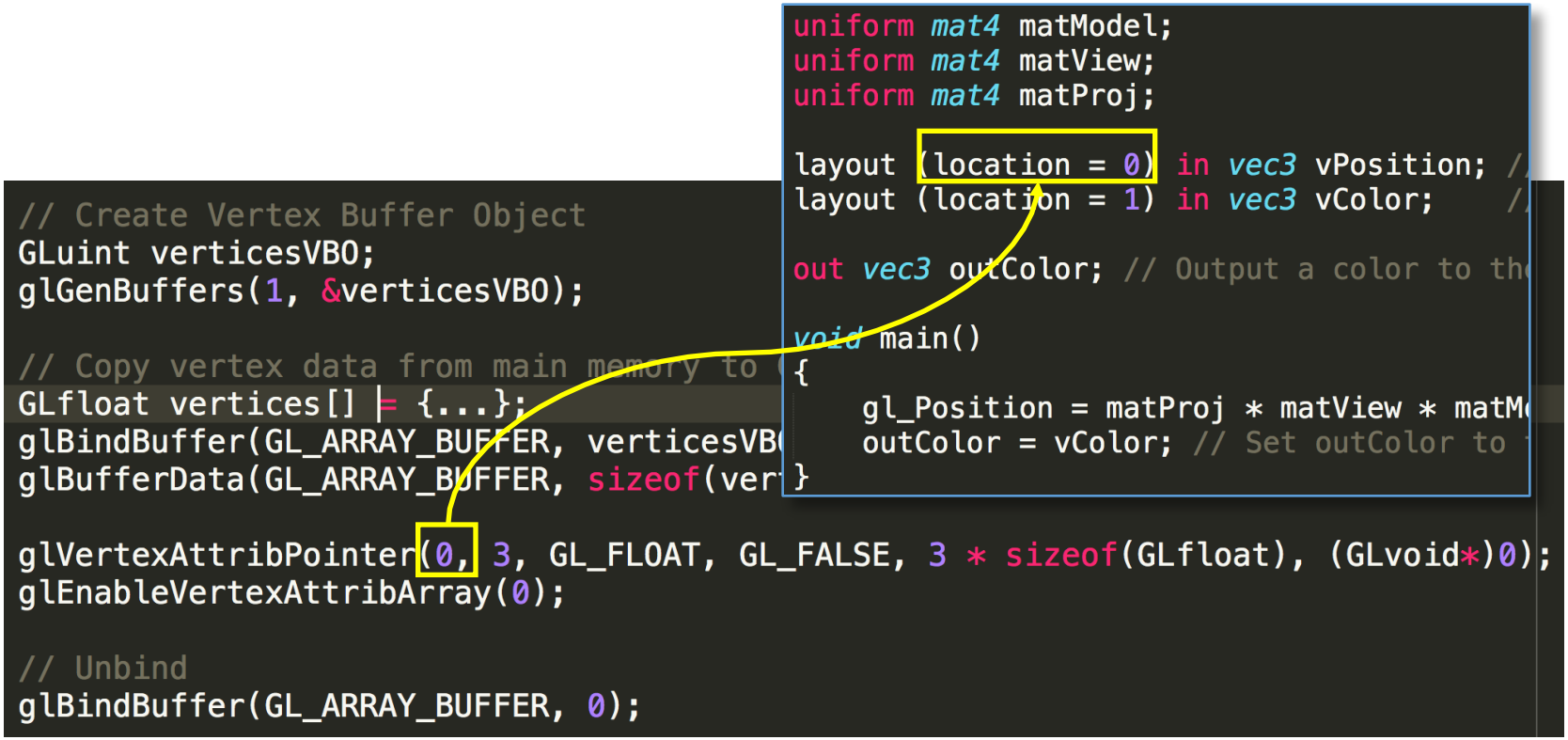
```
// Create Vertex Buffer Object
GLuint verticesVBO;
glGenBuffers(1, &verticesVBO);

// Copy vertex data from main memory to GPU memory
GLfloat vertices[] = {...};
glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Vertex Buffer Object (VBO)

```
// Create Vertex Buffer Object
GLuint verticesVBO;
glGenBuffers(1, &verticesVBO);

// Copy vertex data from main memory to GPU memory
GLfloat vertices[] = {...};
glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);

// Unbind
glBindBuffer(GL_ARRAY_BUFFER, 0);
```



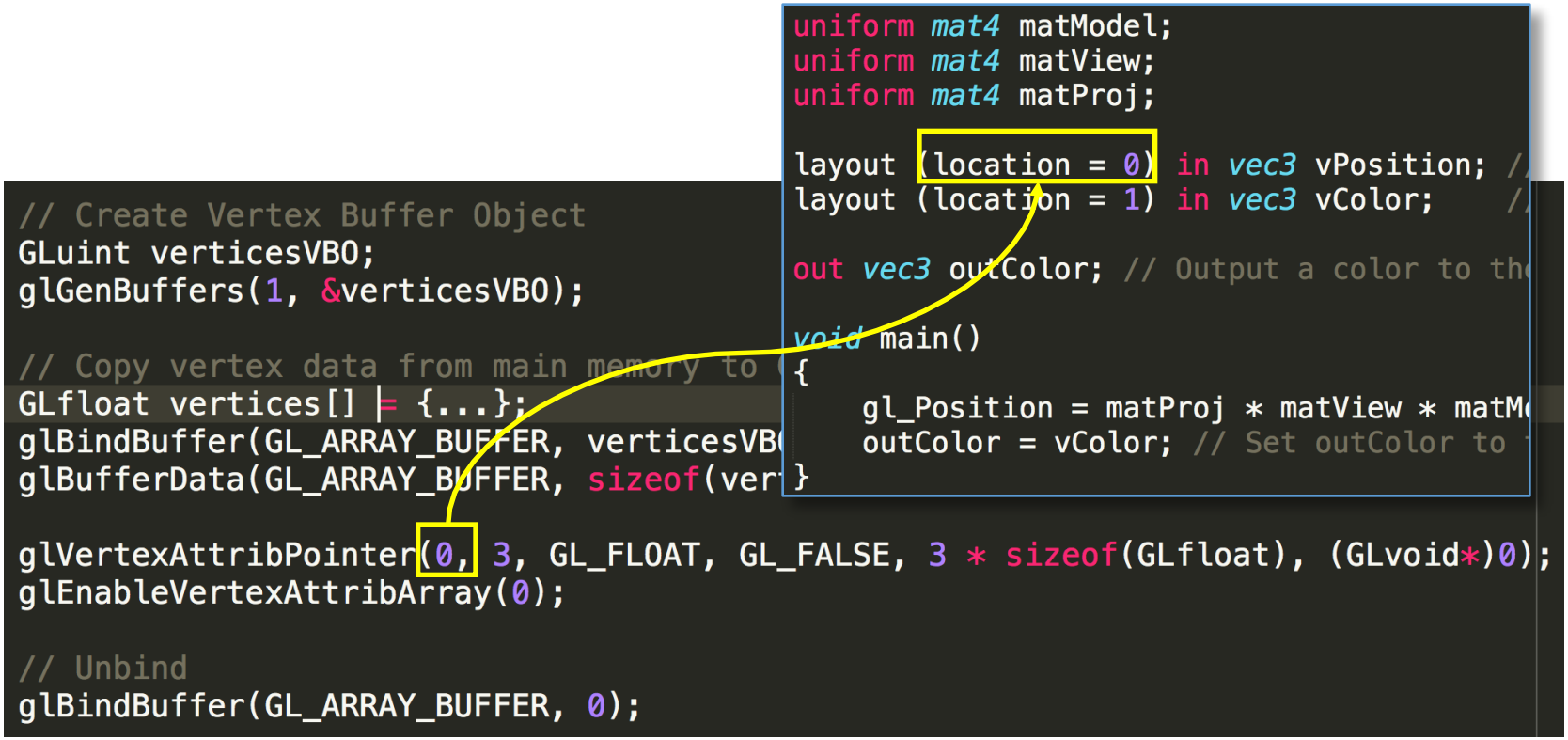
Vertex Buffer Object (VBO)



```
// Create Vertex Buffer Object
GLuint verticesVBO;
glGenBuffers(1, &verticesVBO);

// Copy vertex data from main memory to
GLfloat vertices[] = {...};
glBindBuffer(GL_ARRAY_BUFFER, verticesVB
glBufferData(GL_ARRAY_BUFFER, sizeof(ver

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);

// Unbind
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; //
layout (location = 1) in vec3 vColor;    //

out vec3 outColor; // Output a color to th

void main()
{
    gl_Position = matProj * matView * matM
    outColor = vColor; // Set outColor to
}
```

Vertex Buffer Object (VBO)



```
// Create Vertex Buffer Object
GLuint verticesVBO;
glGenBuffers(1, &verticesVBO);

// Copy vertex data from main memory to
GLfloat vertices[] = {...};
glBindBuffer(GL_ARRAY_BUFFER, verticesVB
glBufferData(GL_ARRAY_BUFFER, sizeof(ver

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);

// Unbind
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; //
layout (location = 1) in vec3 vColor;    //

out vec3 outColor; // Output a color to th

void main()
{
    gl_Position = matProj * matView * matM
    outColor = vColor; // Set outColor to
}
```

Vertex Buffer Object (VBO)

```
// Create Vertex Buffer Object
GLuint verticesVBO;
glGenBuffers(1, &verticesVBO);

// Copy vertex data from main memory to GPU memory
GLfloat vertices[] = {...};
glBindBuffer(GL_ARRAY_BUFFER, verticesVBO
glBufferData(GL_ARRAY_BUFFER, sizeof(vert

glVertexAttribPointer(0, 3, GL_FLOAT, GL_
glEnableVertexAttribArray(0);

// Unbind
glBindBuffer(GL_ARRAY_BUFFER, 0);

// Similarly for colors
GLuint colorsVBO;
glGenBuffers(1, &colorsVBO);
.
.
.
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(1);
```

```
uniform mat4 matModel;
uniform mat4 matView;
uniform mat4 matProj;

layout (location = 0) in vec3 vPosition; //
layout (location = 1) in vec3 vColor;    //

out vec3 outColor; // Output a color to th

void main()
{
    gl_Position = matProj * matView * matM
    outColor = vColor; // Set outColor to
}
```

Vertex Buffer Object (VBO)

```
GLuint vertexIndexBuffer;
glGenBuffers(1, &vertexIndexBuffer);

GLuint indices[] = {...};
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vertexIndexBuffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, , sizeof(indices), indices, GL_STATIC_DRAW);
```

Note: GL_ELEMENT_ARRAY_BUFFER

Vertex Buffer Object (VBO)

```
//Bind vertex data buffer
glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);

//Bind color data buffer
glBindBuffer(GL_ARRAY_BUFFER, colorsVBO);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(1);

//Bind index data buffer
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vertexIndexBuffer);

//Finally draw
if(useIndexBuffer)
    glDrawElements(GL_TRIANGLES, sizeof(indices), GL_UNSIGNED_INT, (void*)0);
else
    glDrawArrays(GL_TRIANGLES, 0, sizeof(vertices) );
```

Vertex Buffer Object (VBO)

```
//Bind vertex data buffer
glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);

//Bind color data buffer
glBindBuffer(GL_ARRAY_BUFFER, colorsVBO);
glVertexAttribPointer(1, 3, GL_FLOAT,                    ), (GLvoid*)0);
glEnableVertexAttribArray(1);

//Bind index
glBindBuffer(                    , vertexIndexBuffer);

//Finally draw
if(useIndexBuffer)
    glDrawElements(GL_TRIANGLES, sizeof(indices), GL_UNSIGNED_INT, (void*)0);
else
    glDrawArrays(GL_TRIANGLES, 0, sizeof(vertices) );
```

That's Cumbersome !

Vertex Array Object (VAO)

```
//Create and bind Vertex Array Object
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
    // All the subsequent Binds and Attributes will be stored in vao

    //Bind vertex data buffer
    glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(0);

    //Bind color data buffer
    glBindBuffer(GL_ARRAY_BUFFER, colorsVBO);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(1);

    //Bind index data buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vertexIndexBuffer);

// Unbind Vertex Array Object
glBindVertexArray(0);
```

Vertex Array Object (VAO)

```
//Create and bind Vertex Array Object
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
    // All the subsequent Binds and Attributes will be stored in vao

    //Bind vertex data buffer
    glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
    gl
    //
    gl
    gl                                                                    ;
    gl

    //Bind index data buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vertexIndexBuffer);

// Unbind Vertex Array Object
glBindVertexArray(0);
```

"VBO Binds" and "Attribute Pointer" States will be stored in VAO

Vertex Array Object (VAO)

```
//Create and bind Vertex Array Object
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
    // All the subsequent Binds and Attributes will be stored in vao

    //Bind vertex data buffer
    glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
    gl
    //
    gl
    gl                                                                    ;
    gl

    //Bind index data buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vertexIndexBuffer);

// Unbind Vertex Array Object
glBindVertexArray(0);
```

"VBO Binds" and "Attribute Pointer" States will be stored in VAO

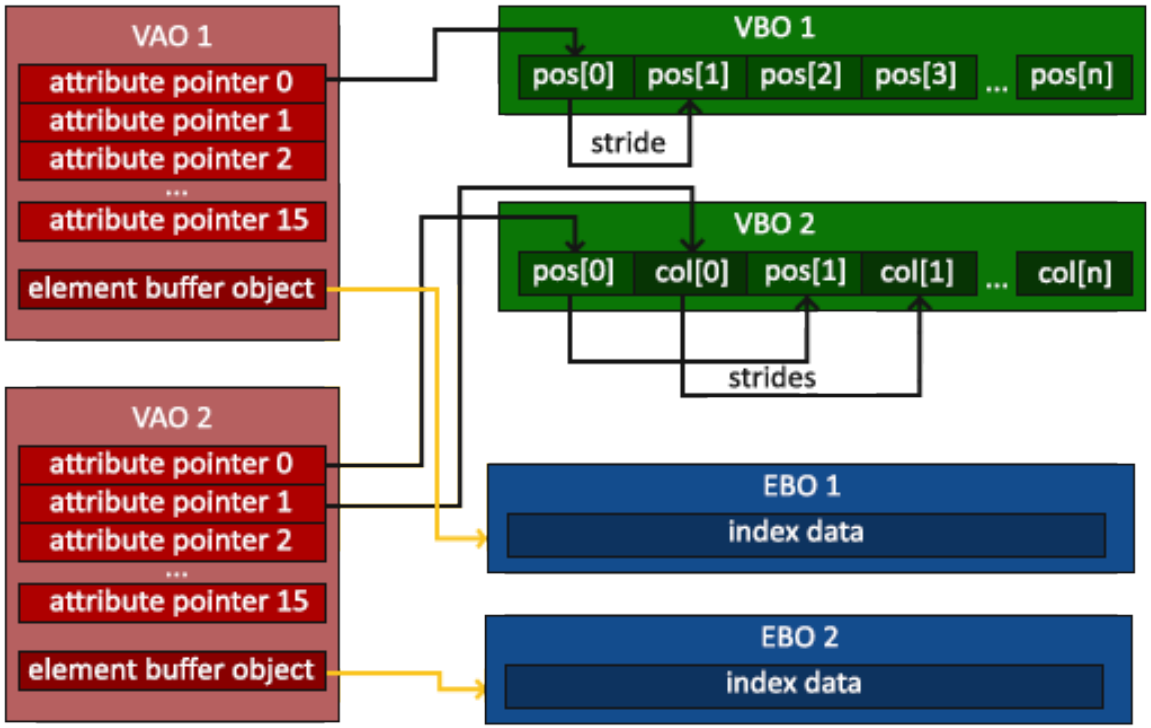
Don't unbind; VAO will remember that too!

Vertex Array Object (VAO)

```
//Create and bind Vertex Array Object
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
    // All the subsequent Binds and Attributes will be stored in vao

    //Bind vertex data buffer
    glBindBuffer(GL_ARRAY_BUFFER, verticesVBO);
    glVertexAttribPointer(0, 3, GL_FLOAT, G                    at), (GLvoid*)0);
    glEnableVertexAttribArray(0);

    //Bind color data
    glBindBuff                                  );
    glVertexAt                 LOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVe                 ay(1);

    //Bind index data buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vertexIndexBuffer);

// Unbind Vertex Array Object
glBindVertexArray(0);
```

At setup time

Vetex Array Object (VAO)

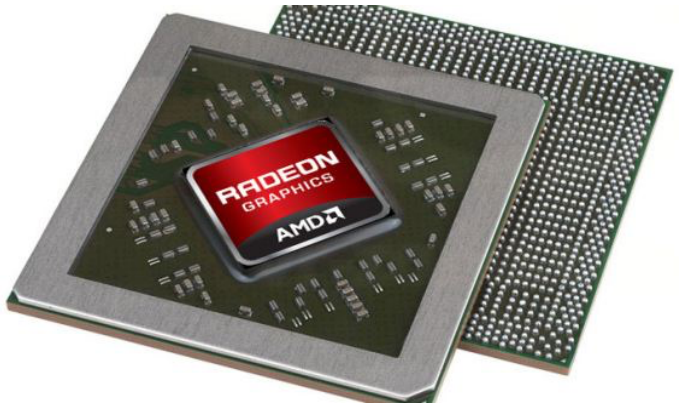


http://learnopengl.com/#!Getting-started/Hello-Triangle

Vertex Array Object (VAO)

And Finally …

```
glUseProgram(shaderProgram);
glBindVertexArray(vao);
glDrawElements(GL_TRIANGLES, sizeof(indices), GL_UNSIGNED_INT, (void*)0);
glBindVertexArray(0);
glUseProgram(0);
```



GPU, Shaders & OpenGL >= 3.2

CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

http://www.pling.org.uk/cs/cgv.html