The Graphics Pipeline and OpenGL III:
OpenGL Shading Language

Dr. Sheng Bin (盛斌)
Shanghai Jiao Tong University
Lecture 4

Lecture Overview

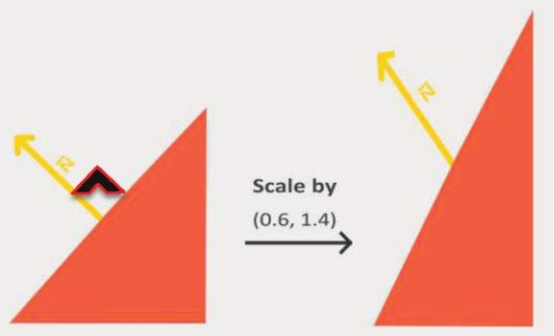
- Normal Matrix
- Review of graphics pipeline
- OpenGL shader types
- OpenGL Shading Language (GLSL)

Normal Matrix

- During lighting calculations you need to transform all vectors into one coordinate system
- Homogeneous transforms don't apply to normals
- Normals are only direction vectors and do not represent a specific position in space
- Translations do and should not affect normal vectors
  - To transform normal vector with model matrix:
    - remove the translation by only taking the upper left 3x3 matrix of the model matrix
    - Can also set w component of 4D normal vector to 0

Normal Matrix

- Non-uniform scaling in model matrix -> normal vector not perpendicular to surface anymore



$$M = \begin{pmatrix} 0.6 & 0 & 0 \\ 0 & 1.4 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(upper left 3x3
matrix of model view)

Normal Matrix

- So if we can't apply the modelview matrix in all cases, what matrix do we apply?
- N = normal vector, T = tangent vector
- $N \cdot T = 0 \rightarrow N' \cdot T' = 0$
- We know that M preserves tangents, so we can apply it to T
- Let's assume there is some matrix G such that the following equation holds:

$$N' \cdot T' = (GN) \cdot (MT) = 0$$

$$(GN) \cdot (MT) = (GN)^T (MT) = N^T G^T MT$$

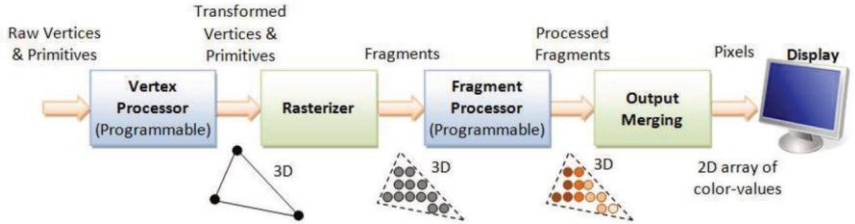
- If $G^T M = I$, then $N' \cdot T' = N \cdot T = 0$
- Therefore:

$$G^T M = I \Rightarrow G = (M^{-1})^T$$

G = Normal Matrix

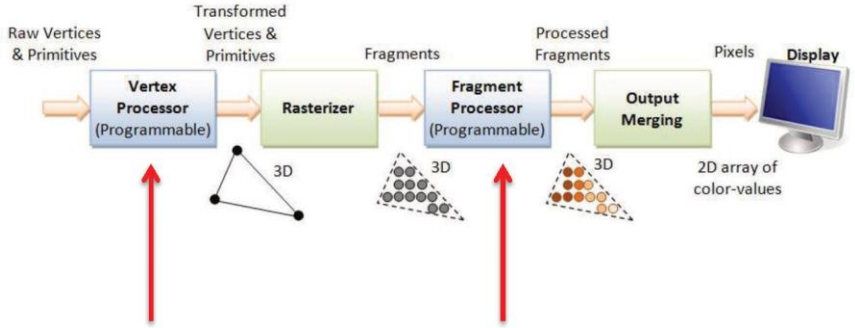
Questions?

Reminder: The Graphics Pipeline



https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

Reminder: The Graphics Pipeline



vertex shader

- transforms
- (per-vertex) lighting

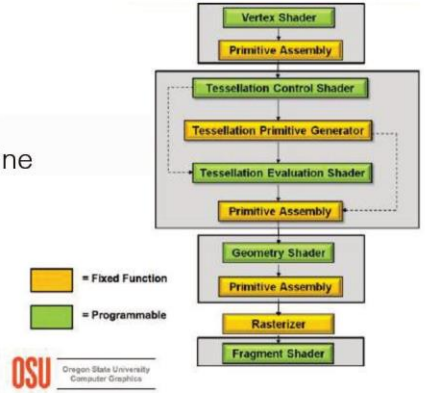
fragment shader

- texturing
- (per-fragment) lighting

https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

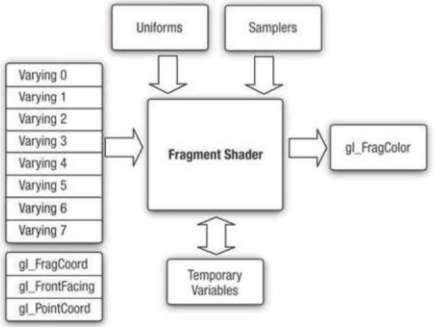
Shaders

- Shaders are little programs that rest on GPU
  - Isolated
  - Transform inputs to outputs
- Run for each specific section of graphics pipeline
- Replaces Fixed Function Pipeline
  - Hard codes lighting and texture mapping
- OpenGL uses GLSL shader Language
- Direct3D uses HLSL shader Language



Types of Shaders

- 2D Shaders
  - Fragment (Pixel) Shaders
    - Output color and other attributes of each fragment
    - Have access to scene geometry and nearby fragments
      - Lighting, bump maps, shadows, depth of field blur, etc.



Types of Shaders

- 2D Shaders
  - Fragment (Pixel) Shaders
    - Output color and other attributes of each fragment
    - Have access to scene geometry and nearby fragments
      - **Lighting**, bump maps, shadows, depth of field blur, etc.

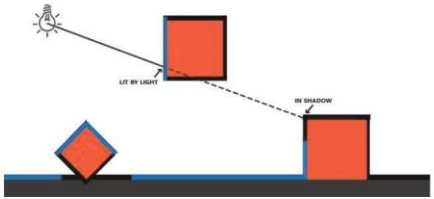
Types of Shaders

- 2D Shaders
  - Fragment (Pixel) Shaders
    - Output color and other attributes of each fragment
    - Have access to scene geometry and nearby fragments
      - Lighting, **normal maps**, shadows, depth of field blur, etc.

Learnopengl.com

Types of Shaders

- 2D Shaders
  - Fragment (Pixel) Shaders
    - Output color and other attributes of each fragment
    - Have access to scene geometry and nearby fragments
      - Lighting, normal maps, **shadows**, depth of field blur etc.



Types of Shaders

- 2D Shaders
  - Fragment (Pixel) Shaders
    - Output color and other attributes of each fragment
    - Have access to scene geometry and nearby fragments
      - Lighting, normal maps, **shadows**, depth of field blur etc.

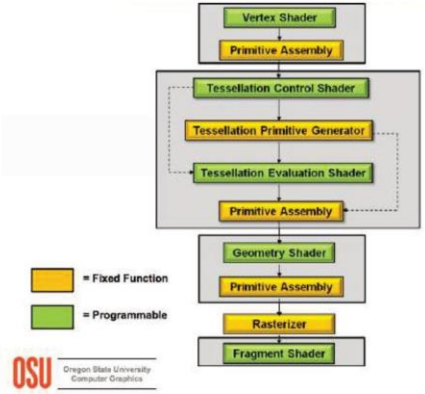
http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter17.html

Types of Shaders

- 2D Shaders
  - Fragment (Pixel) Shaders
    - Output color and other attributes of each fragment
    - Have access to scene geometry and nearby fragments
      - Lighting, normal maps, shadows, **depth of field blur**, etc.

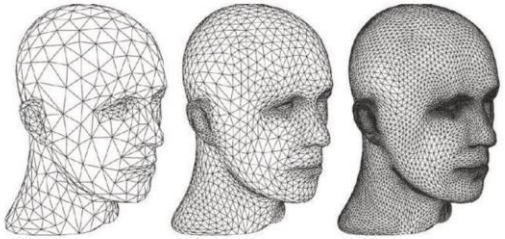
Types of Shaders

- 3D Shaders
  - Vertex Shader
    - Transform each vertex's 3D position to 2D coordinate
    - Cannot add new vertices
  - Tessellation Shader (optional)
    - Patches of vertex data are subdivided into smaller primitives
  - Geometry Shader (optional)
    - Can add, remove, modify primitives
    - Last stage that can modify geometry



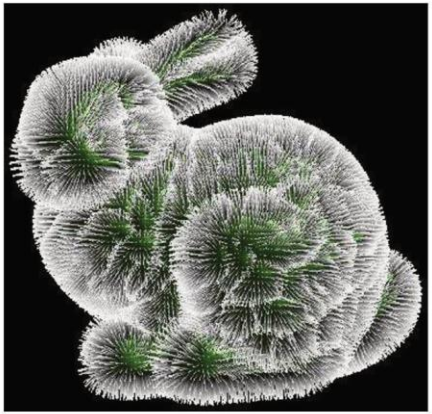
Types of Shaders

- 3D Shaders
  - Vertex Shader
    - Transform each vertex's 3D position to 2D coordinate
    - Cannot add new vertices
  - **Tessellation Shader (optional)**
    - Patches of vertex data are subdivided into smaller primitives
  - Geometry Shader (optional)
    - Can add, remove, modify primitives
    - Last stage that can modify geometry



Types of Shaders

- 3D Shaders
  - Vertex Shader
    - Transform each vertex's 3D position to 2D coordinate
    - Cannot add new vertices
  - Tessellation Shader (optional)
    - Patches of vertex data are subdivided into smaller primitives
  - **Geometry Shader (optional)**
    - Can add, remove, modify primitives
    - Last stage that can modify geometry



http://blog.twodee.org/

Questions?

GLSL

- OpenGL Shading Language
- Based on ANSI C
- C extended with vector and matrix types (similar to glm::vec3 and glm::mat4)
- Overloading functions based on argument type supported (from C++)
- Ability to declare variables where needed, instead of at beginning of blocks

GLSL

```
#version version_number

in type in_variable_name;
in type in_variable_name;

out type out_variable_name;

uniform type uniform_name;

void main()
{
  // Process input(s) and do some weird graphics stuff
  ...
  // Output processed stuff to output variable
  out_variable_name = weird_stuff_we_processed;
}
```

Learnopengl.com

Vectors and Matricies

- Vectors
  - 1,2,3, or 4 component container for any of types above
  - vecn, bvecn, ivecn, uvecn, dvecn
  - n is the number of components
  - Allows for swizzling:

```
vec2 someVec;
vec4 differentVec = someVec.xyxx;
vec3 anotherVec = differentVec.zyw;
vec4 otherVec = someVec.xxxx + anotherVec.yxzy;
```

- Matricies:
  - Column-major ordering (also default for GLM)

Inputs/outputs

- Shaders can't communicate directly, only though inputs / outputs
- *in* and *out* keywords
- Where the output of one shader **matches** the input of the next one, the data is passed along
  - Case sensitive
  - "out vec3 vertexPosition;" matches to "in vec3 vertexPosition;"
  - "out vec3 vertexPosition;" will NOT match "in vec3 vertexposition;"
- Vertex shader is first stage, and gets data directly from vertex data
  - Input specifies how data is organized with *layout* keyword

Example

Vertex Shader

```
#version 420 core
layout (location = 0) in vec3 position;         // The position variable has attribute position 0

out vec4 vertexColor;                           // Specify a color output to the fragment shader

void main()
{
    gl_Position = vec4(position, 1.0);          // See how we directly give a vec3 to vec4's constructor
    vertexColor = vec4(0.5f, 0.0f, 0.0f, 1.0f); // Set the output variable to a dark-red color
}
```

Fragment Shader

```
#version 420 core
in vec4 vertexColor; // The input variable from the vertex shader (same name and same type)

out vec4 color;

void main()
{
    color = vertexColor;
}
```

Uniforms

- Another way to pass data from CPU to shaders on GPU from vertex attributes
- Uniforms are global
  - Unique per shader program object
  - Can be accessed from any shader at any stage in the shader program
- Uniforms will keep value until they are reset or updated

```
uniform vec4 ourColor; // We set this variable in the OpenGL code.
```

- Can define them in any shader -> no need to pass uniform data from vertex shader to fragment shader

Uniforms

- If you declare a uniform that isn't used anywhere in your GLSL code the compiler silently removes the variable from the compiled code which will cause "uniform not found" errors when you try to attach it in your OpenGL code
- How to attach uniform:
  - Get location using:

```
GLint glGetUniformLocation(GLuint program, const GLchar *name);
```

  - Attach using *glUniform*, which sets uniform on **active** shader program
  - OpenGL is core C library, it does not support type overloading so you have a different function for every different type uniform you want to pass
    - glUniform1f, glUniform2f, glUniform1i, glUniformMatrix4fv, etc

Compiling a shader

- For each shader type:

```
GLuint vertexShader;
glCreateShader(GL_VERTEX_SHADER);       // Create shader
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL); // Load shader source code into shader object
glCompileShader(vertexShader);                              // Compile shader
```

- Need link shader objects together

```
GLuint shaderProgram;
shaderProgram = glCreateProgram();              // Create program
glAttachShader(shaderProgram, vertexShader);    // Attach vertex shader
glAttachShader(shaderProgram, fragmentShader);  // Attach fragment shader
glLinkProgram(shaderProgram);                   // Link
```

- Active shader program:

```
glUseProgram(shaderProgram);
```

Remember OpenGL is a state machine, every rendering call after glUseProgram will now use this program object (and thus shaders), until changed

Questions?

Summary

- Normal Matrix: Used for transforming normal vectors correctly
- Shaders:
  - 2D shaders: Fragment
  - 3D shaders: Vertex, geometry, tesselation
- GLSL: OpenGL shader language

Further Reading

- good overview of OpenGL (deprecated version) and graphics pipeline (missing a few things) :

  https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

- textbook: Shirley and Marschner “Fundamentals of Computer Graphics”, AK Peters, 2009

- definite reference: “OpenGL Programming Guide” aka “OpenGL Red Book”