CSCD396

Beginning Graphics

Today’s Topic

- Instancing
- Uniform variables

Instancing

- We may need to draw the same object many times in virtual workspace; for example, a forest full of same kind of trees, a field full of grass etc.
- It needs thousands copies of identical sets of geometry, modified only slightly from instance to instance i.e. different position, orientation, color etc)
- OpenGL address this through instance rendering, draw many copies of the same geometry with one draw command;
- Instance rendering needs instance positions, orientation, color etc.

Instancing

glDrawArraysInstanced(GLenum mode, GLint first, GLsizei count, GLsizei primCount);

Draws ***primCount*** instances of the geometric primitives specified by ***mode***, ***first***, and ***count*** as if specified by individual calls to ***glDrawArrays()***. The built-in variable ***gl_InstanceID*** is incremented for each instance, and new values are presented to the vertex shader for each instanced vertex attribute.

Instancing

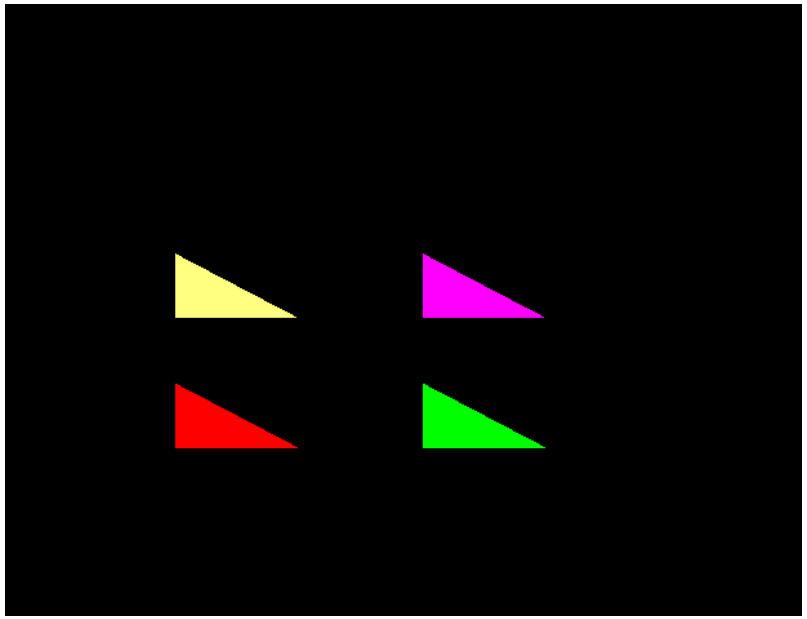
- ***void glVertexAttribDivisor(GLuint index, GLuint divisor);***

- '*divisor*' specifies the rate at which new values of the instanced vertex attribute at *index* are presented to the vertex shader during instanced rendering.

- If divisor is '1', the value of the attribute is updated every instance; if '2', then update is performed every second instance etc.

Instancing

- An object drawn several times with instanced position and color:



Instancing

```
GLfloat tri_vertices[] ={-1.0f, -1.0f, 0.0f, 1.0f,
                          1.0f, -1.0f, 0.0f, 1.0f,
                         -1.0f, 1.0f, 0.0f, 1.0f};
```

```
GLfloat instance_colors[] ={ 1.0f, 0.0f, 0.0f, 1.0f,
                             0.0f, 1.0f, 0.0f, 1.0f,
                             1.0f, 0.0f, 1.0f, 1.0f,
                             1.0f, 1.0f, 0.5f, 1.0f };
```

- GLfloat instance_positions[] ={-2.0f, -2.0f, 0.0f, 1.0f,
  2.0f, -2.0f, 0.0f, 1.0f,
  2.0f, 2.0f, 0.0f, 1.0f,
  -2.0f, 2.0f, 0.0f, 1.0f};

Instancing

```
glVertexAttribPointer(position_loc, 4, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(position_loc);


glVertexAttribPointer(color_loc, 4, GL_FLOAT, GL_FALSE, 0, (Glvoid *)sizeof(tri_vertices));
glEnableVertexAttribArray(color_loc);
glVertexAttribDivisor(color_loc, 1); // every instance of color will change


glVertexAttribPointer(instance_position_loc, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid *)(sizeof(tri_vertices) + sizeof(instance_colors)));
glEnableVertexAttribArray(instance_position_loc);
glVertexAttribDivisor(instance_position_loc, 1); //every instance of position will change
```

Instancing

- Lastly, **glDrawArraysInstanced** is called with the number of the instance of triangle to be created

```
glDrawArraysInstanced(GL_TRIANGLES, 0, 3, 5);
```

- You can use instancing with **glDrawElements** the same way by adding the number of instances in the parameter list, i.e., **glDrawElementsInstanced**.

Uniform Variables

- Declared as global variables passed to shader from the application
- The same uniform variable can be accessed by both vertex shader and fragment shader;
- Values can only be changed by application;
- Cannot be written or changed by a shader,
- Constant across a given primitive;

Uniform Variables

- GLint **glGetUniformLocation**(GLuint *program*,const char* *name*)
  - Returns the index of the uniform variable name associated with the shader program;
  - Once you have the associated index for the uniform variable, you can set the value of the uniform variable using the **glUniform*()** or **glUniformMatrix*()** routines.

```
GLint timeLoc; /* Uniform index for variable "time" in shader */
GLfloat timeValue; /* Application time */
timeLoc = glGetUniformLocation(program, "time");
glUniform1f(timeLoc, timeValue);
```

Uniform Variables

- void **glUniform**{1234}{fdi ui}(GLint *location*, TYPE *value*);
- void **glUniform**{1234}{fdi ui}v(GLint *location*, GLsizei *count*, const TYPE * *values*);
- void **glUniformMatrix**{234}{fd}v(GLint *location*, GLsizei *count*, GLboolean *transpose*, const GLfloat * *values*);
- void **glUniformMatrix**{2x3,2x4,3x2,3x4,4x2,4x3}{fd}v(
- GLint *location*, GLsizei *count*,GLboolean *transpose*, const GLfloat * *values*);

Uniform Variables in the Applocation

- //get the locations of the uniform variables

- glm::vec3 scale(1.0f, 1.0f, 1.0f); ← uses **glm** library
- scale_loc = glGetUniformLocation(prog, "scale");
- glUniform3fv(scale_loc, 1,(GLfloat*)&scale[0]);

Uniform Variables in Shader

- layout (location = 0) in vec4 position;
- layout (location = 1) in vec4 instance_color;
- layout (location = 2) in vec4 instance_position;

- uniform vec3 scale;

Using GLM Library

- OpenGL Mathematics Library: A header only library
- Download from the following website:
  - http://glm.g-truc.net/0.9.7/index.html

- **For windows**
  - After extracting, you just need to place the GLM folder (that contains header files) to
  - …..\Microsoft Visual Studio 14.0\VC\include\glm\ ← VS2015
  - C:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\VC\Tools\MSVC\14.10.25017\include\glm ← VS2015

- **For Linux**
  - Should already be installed if you use the OpenGL installation commands (Lecture 1 Week 1)

Using GLM Library

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define GLM_FORCE_RADIANS

#include <glm/mat4x4.hpp>
#include <glm/gtc/matrix_transform.hpp>
```