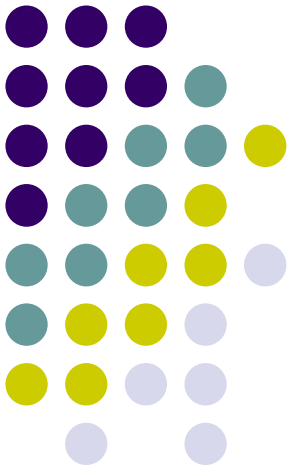
Computer Graphics (CS 543) Lecture 1 (Part 3): Introduction to OpenGL/GLUT (Part 2)

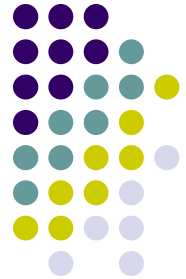
Prof Emmanuel Agu

Computer Science Dept.

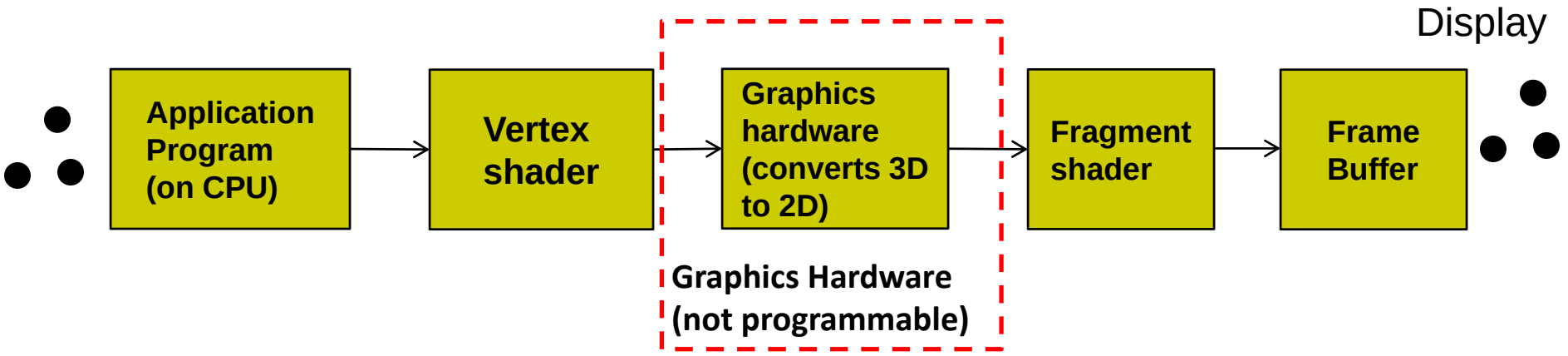
Worcester Polytechnic Institute (WPI)



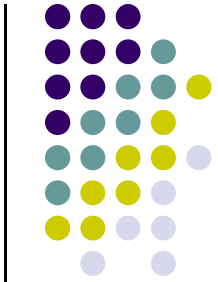
What other Initialization do we Need?



- Also set clear color and other OpenGL parameters
- Also set up shaders as part of initialization
  - Read, compile, link
- Also need to specify two shaders:
  - **Vertex shader:** program that is run once on **each vertex**
  - **Fragment shader:** program that is run once on **each pixel**
- Need to connect **.cpp file** to **vertex shader** and **fragment shader**

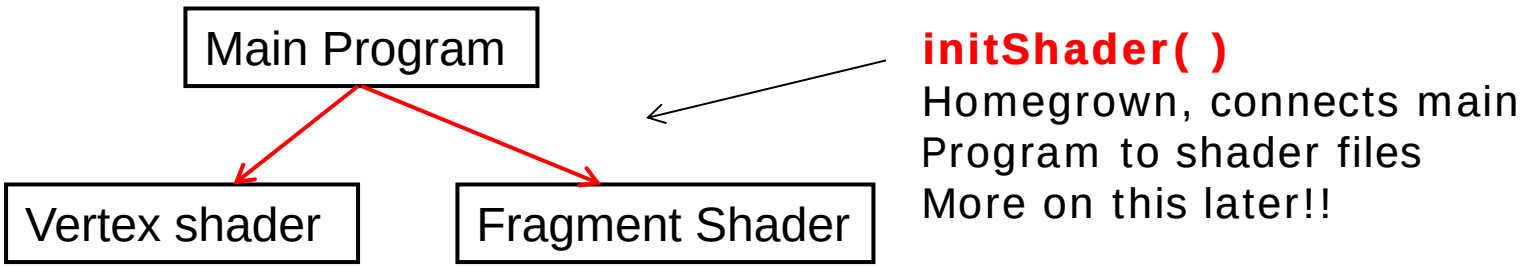


OpenGL Program: Shader Setup

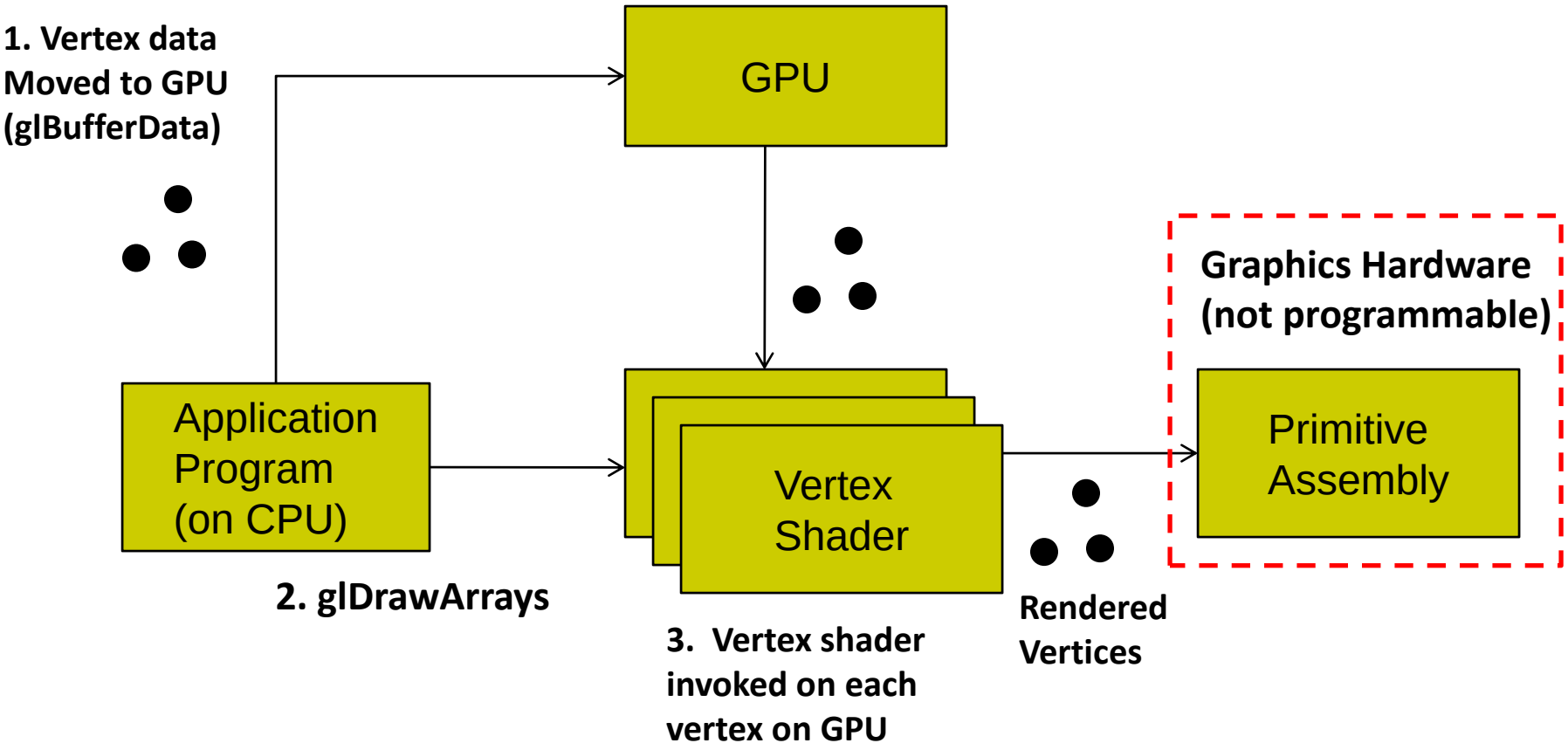


- OpenGL programs now have 3 parts:
  - Main **OpenGL program** (.cpp file), **vertex shader** (e.g. vshader1.glsl), and **fragment shader** (e.g. fshader1.glsl) in same Windows directory
  - In main program, need to link names of vertex, fragment shader
  - **initShader( )** is homegrown shader initialization function. More later

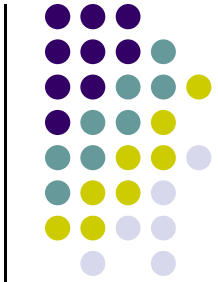
```
GLuint = program;
GLuint program = InitShader( "vshader1.glsl", "fshader1.glsl" );
glUseProgram(program);
```



Execution Model



Vertex Shader



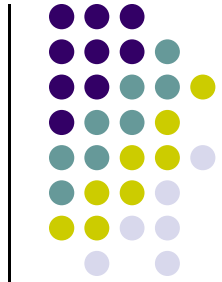
- We write a simple “pass-through” shader (does nothing)
- Simply sets output vertex position to received input position
- **`gl_Position`** is built in variable (already declared)

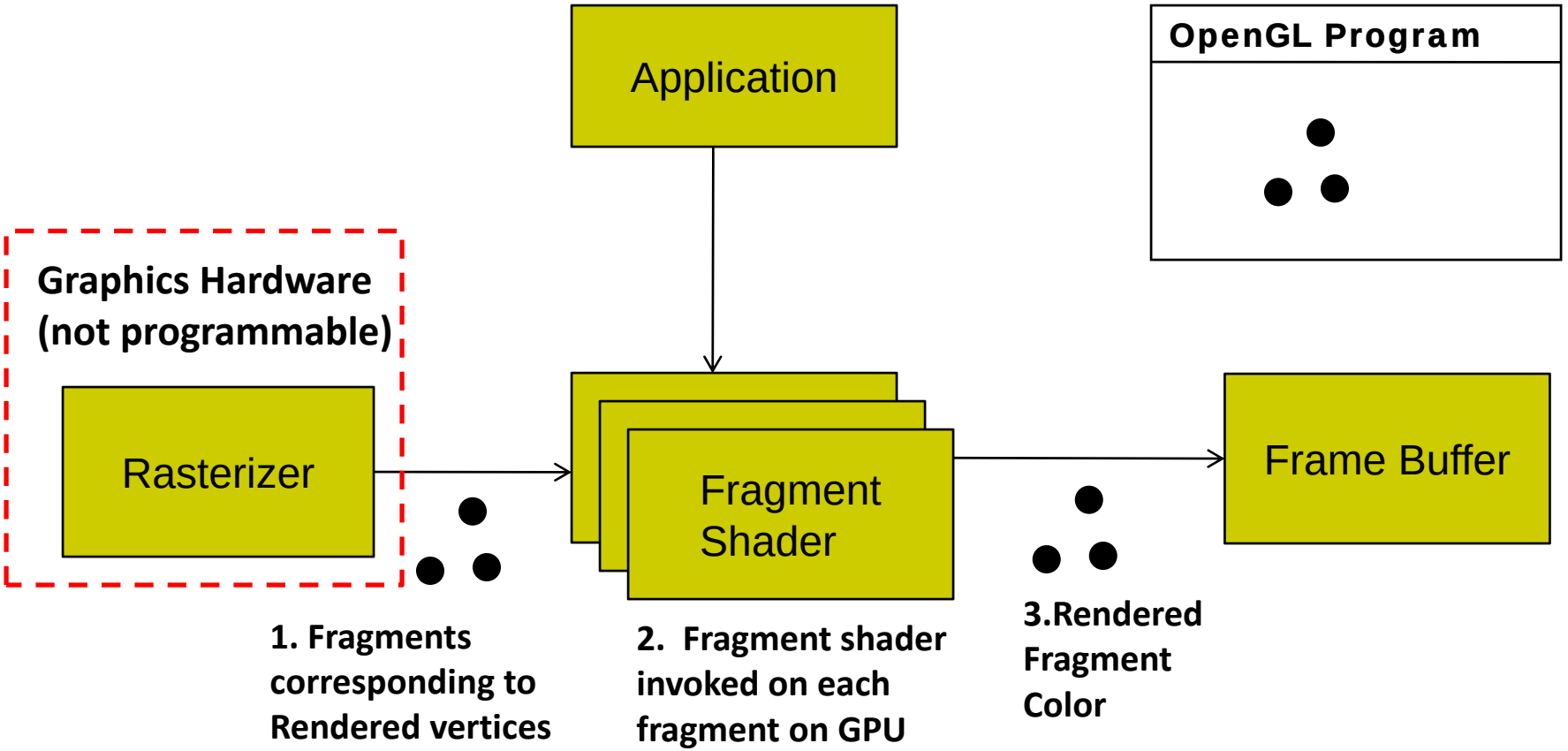
```
in vec4 vPosition

void main( )
{
      gl_Position = vPosition;
}
```

output vertex position

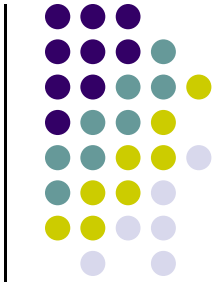
input vertex position

Execution Model



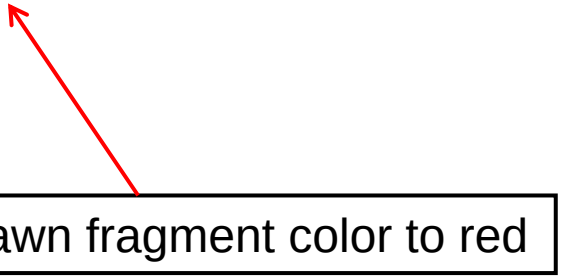


Fragment Shader

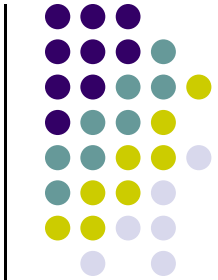


- We write a simple fragment shader (sets color to red)
- **`gl_FragColor`** is built in variable (already declared)

```
void main( )
{
      gl_FragColor = vec(1.0, 0.0, 0.0, 1.0);
}
```

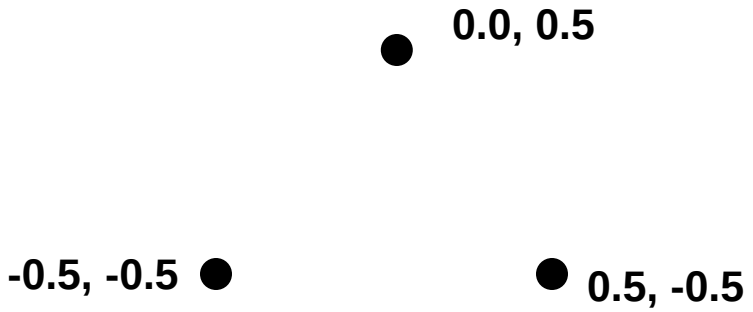


Previously: Generated 3 Points to be Drawn

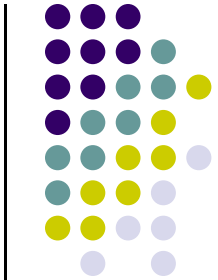


- Stored points in array **points[**], moved to GPU, draw using **`glDrawArray`**

```
point2 points[NumPoints];

points[0] = point2( -0.5, -0.5 );
points[1] = point2( 0.0, 0.5 );
points[2] = point2( 0.5, -0.5 );
```

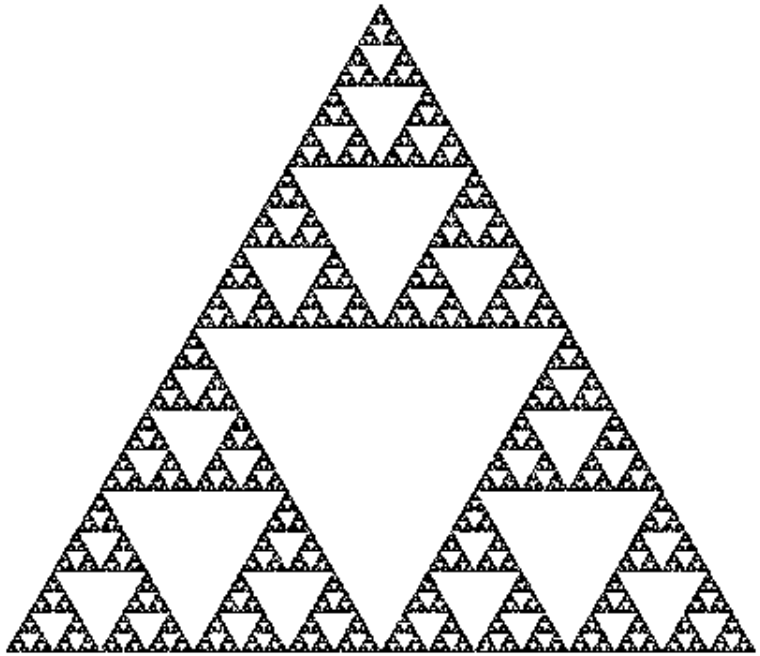


- Once drawing steps are set up, can generate more complex sequence of points algorithmically, drawing steps don't change
- Next: example of more algorithm to generate more complex point sequences

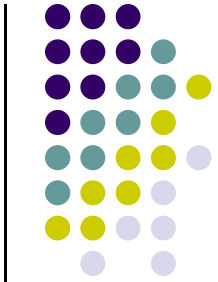


Sierpinski Gasket Program

- Any sequence of points put into array points[ ] will be drawn
- Can generate interesting sequence of points
  - Put in array points[ ], draw!!
- Sierpinski Gasket: Popular fractal



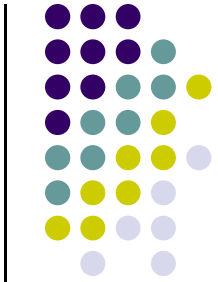
Sierpinski Gasket



Start with initial triangle with corners (*x1, y1, 0*), (*x2, y2, 0*) and (*x3, y3, 0*)

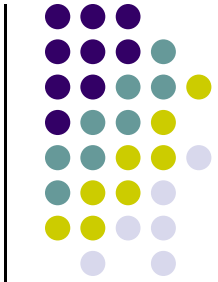
1. Pick initial point **p** = (*x, y, 0*) at random inside a triangle
2. Select on of 3 vertices at random
3. Find **q,** halfway between **p** and randomly selected vertex
4. Draw dot at **q**
5. Replace **p** with **q**
6. Return to step 2

Actual Sierpinski Code



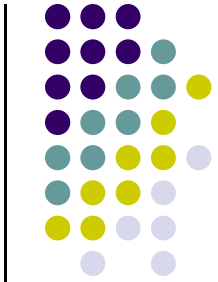
```
#include “vec.h”    // include point types and operations
#include <stdlib.h> // includes random number generator

void Sierpinksi( )
{
    const int NumPoints = 5000;
    vec2 points[NumPoints];

    // Specifiy the vertices for a triangle
    vec2 vertices[3] = {
        vec2( -1.0, -1.0 ), vec2( 0.0, 1.0 ), vec2( 1.0, -1.0 )
    };
```

Actual Sierpinski Code



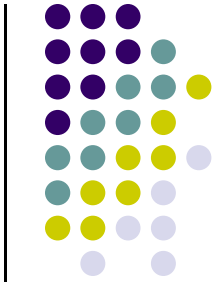
```
// An arbitrary initial point inside the triangle
points[0] = point2(0.25, 0.50);

// compute and store N-1 new points
for ( int i = 1; i < NumPoints; ++i ) {
    int j = rand() % 3;   // pick a vertex at random

    // Compute the point halfway between the selected vertex
    //   and the previous point
    points[i] = ( points[i - 1] + vertices[j] ) / 2.0;
}
```

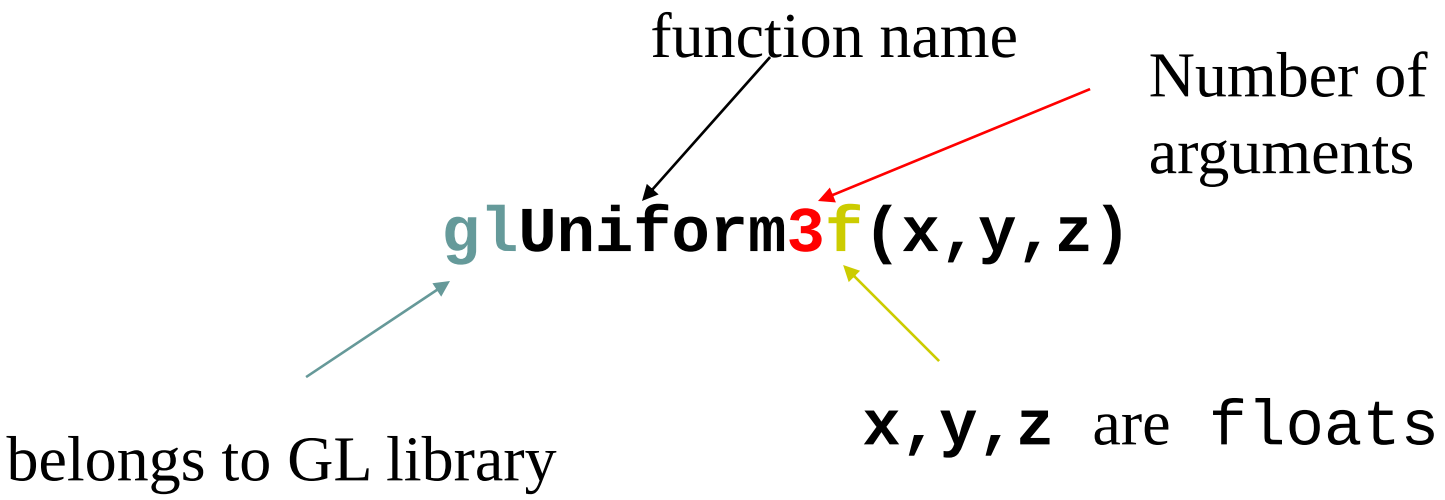
Lack of Object Orientation



- OpenGL is not object oriented
- Multiple versions for each command
  - `glUniform3f`
  - `glUniform2i`
  - `glUniform3dv`

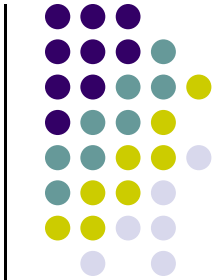
OpenGL function format






```
glUniform3fv(p)
```

Argument is array of values
p is a pointer to array



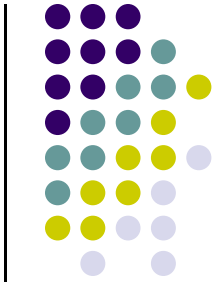
Recall: Single Buffering

- If display mode set to single framebuffers
- Any drawing into framebuffer is seen by user. How?
  - **glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);**
    - Single buffering with RGB colors
- Drawing may not be drawn to screen until call to **glFlush( )**

```
void mydisplay(void){
      glClear(GL_COLOR_BUFFER_BIT); // clear screen
      glDrawArrays(GL_POINTS, 0, N);
      glFlush( );
}
```

Drawing sent to screen

Double Buffering

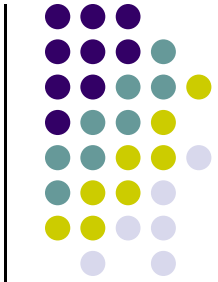


- Set display mode to double buffering (create front and back framebuffers)
  - **`glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);`**
    - Double buffering with RGB colors

- Front buffer displayed on screen, back buffers not displayed
- Drawing into back buffers (not displayed) until swapped in using **`glutSwapBuffers( )`**

```
void mydisplay(void){
      glClear(GL_COLOR_BUFFER_BIT); // clear screen
      glDrawArrays(GL_POINTS, 0, N);
      glutSwapBuffers( );
}
```

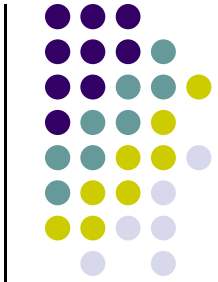
Back buffer drawing swapped in, becomes visible here



OpenGL Data Types

C++	OpenGL
Signed char	GLByte
Short	GLShort
Int	GLInt
Float	GLFloat
Double	GLDouble
Unsigned char	GLubyte
Unsigned short	GLushort
Unsigned int	GLuint

Example: Integer is 32-bits on 32-bit machine
but 64-bits on a 64-bit machine



References

- Angel and Shreiner, Interactive Computer Graphics, 6th edition, Chapter 2
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition, Chapter 2