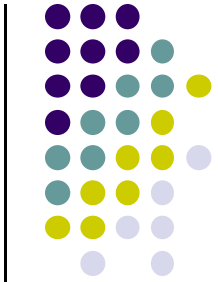
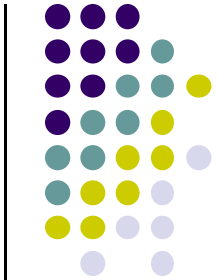
Adding Interaction



- So far, OpenGL programs just render images
- Can add user interaction
- Examples:
  - User hits ‘h’ on keyboard -> Program draws house
  - User clicks mouse left button -> Program draws table

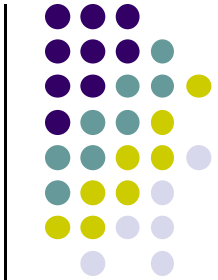
Types of Input Devices



- **String:** produces string of characters e.g. keyboard

- **Locator:** User points to position on display. E.g mouse

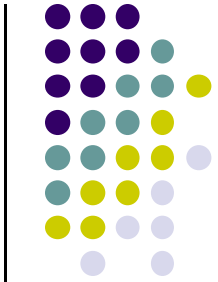
Types of Input Devices



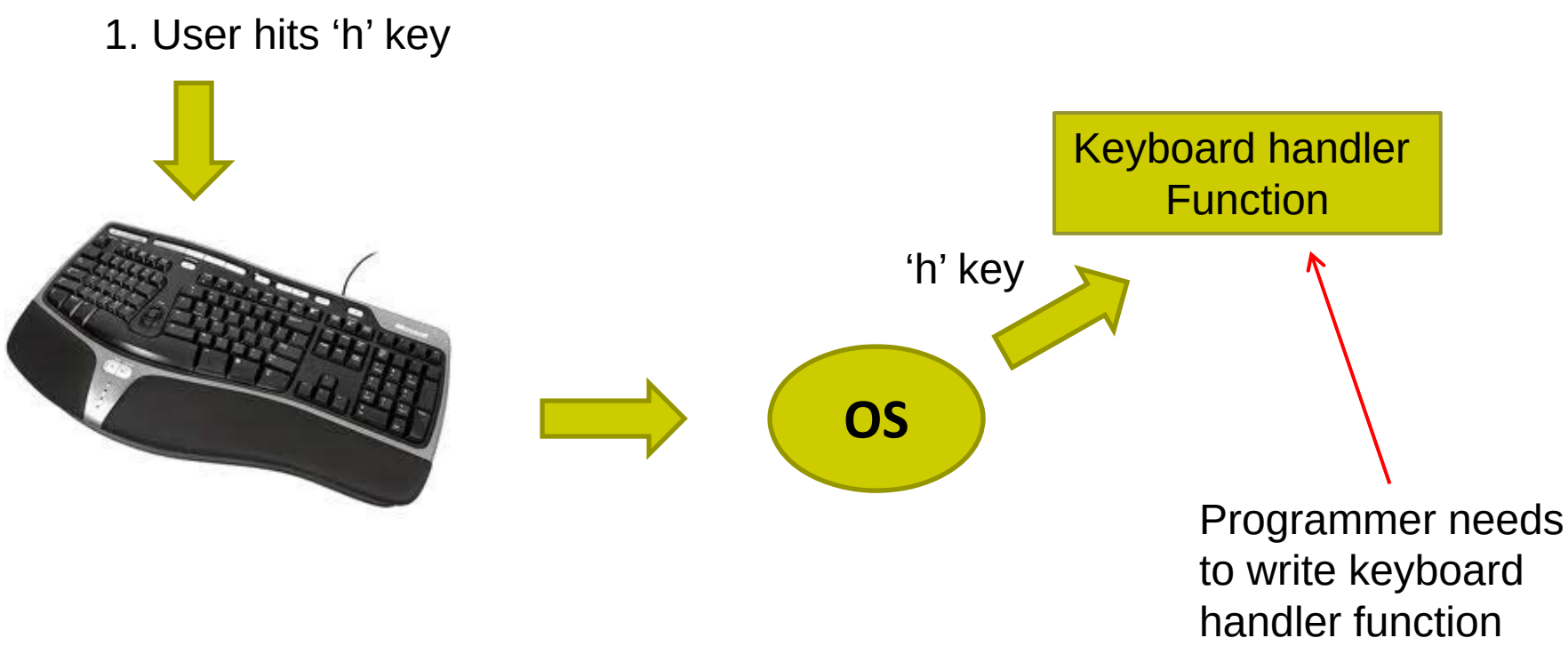
- **Valuator:** generates number between 0 and 1.0 (proportional to how much it is turned)

- **Pick:** User selects location on screen (e.g. touch screen in restaurant, ATM)

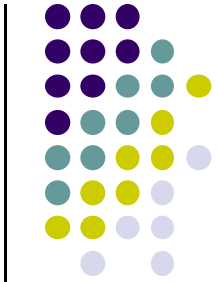
GLUT: How keyboard Interaction Works



- Example: User hits ‘h’ on keyboard -> Program draws house



Using Keyboard Callback for Interaction



```
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);   // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // … now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}
```

1. Register keyboard Function

2. Implement keyboard function

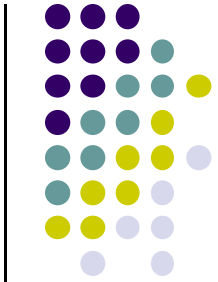
ASCII character of pressed key

x,y location of mouse

```
void myKeyboard(char key, int x, int y )
{    // put keyboard stuff here
       …….
        switch(key){   // check which key
           case 'f':
             // do stuff
           break;

           case 'k':
              // do other stuff
           break;

        }
     ……..
     }
```

Note: Backspace, delete, escape keys checked using their ASCII codes

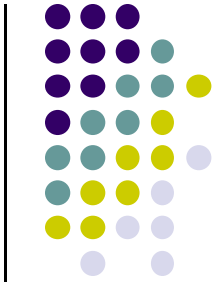
Special Keys: Function, Arrow, etc



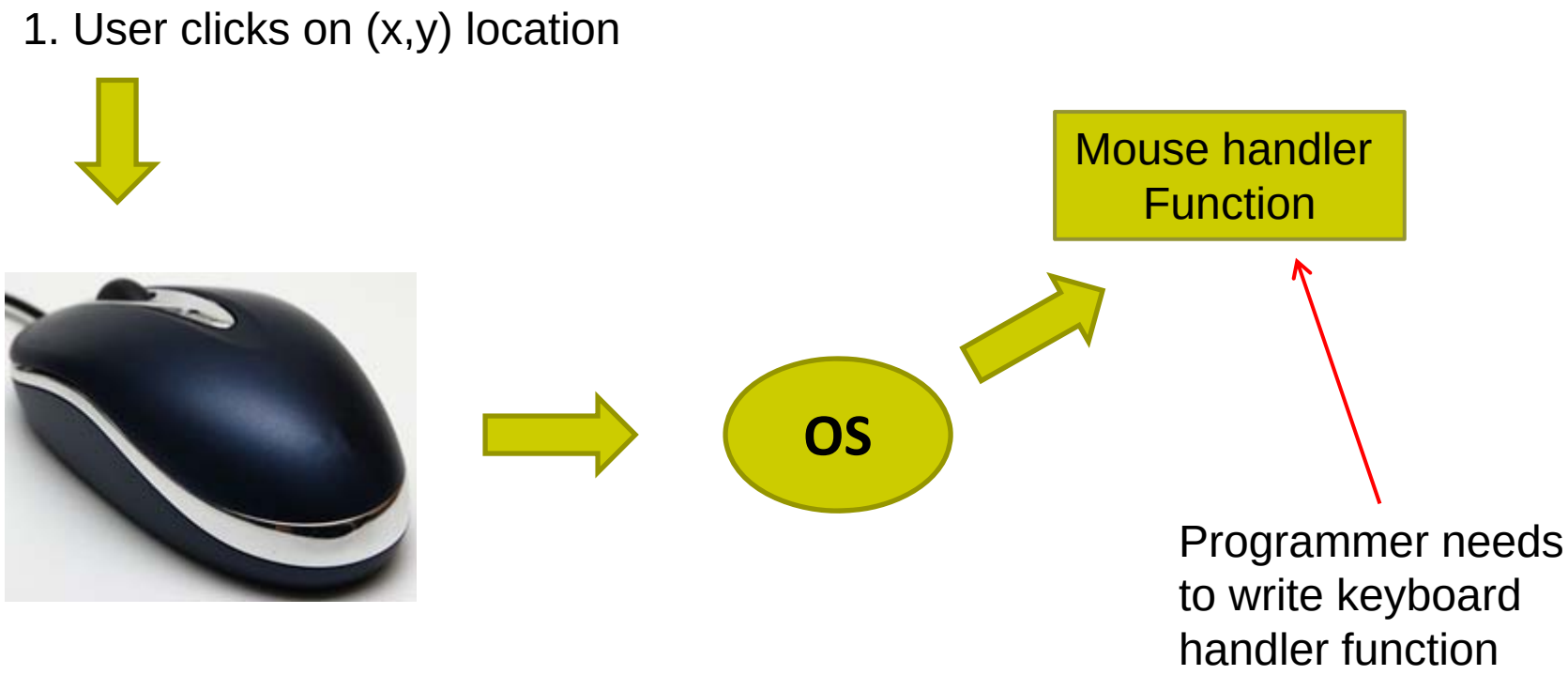
```
glutSpecialFunc (specialKeyFcn);
……

Void specialKeyFcn (Glint specialKey, GLint, xMouse,
                                        Glint yMouse)
```

- Example: if (**`specialKey == GLUT_KEY_F1)`**// F1 key pressed
  - **`GLUT_KEY_F1, GLUT_KEY_F12, ….`** for function keys
  - **`GLUT_KEY_UP, GLUT_KEY_RIGHT, ….`** for arrow keys keys
  - **`GLUT_KEY_PAGE_DOWN, GLUT_KEY_HOME, ….`** for page up, home keys
- Complete list of special keys designated in **`glut.h`**

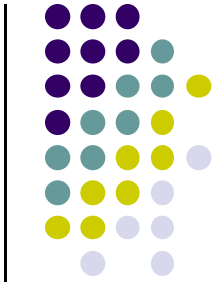
GLUT: How Mouse Interaction Works



- Example: User clicks on (x,y) location in drawing window -> Program draws a line



Using Mouse Callback for Interaction



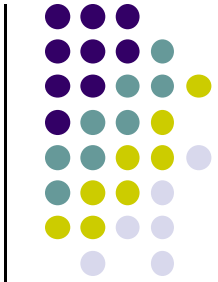
```
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);   // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // … now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}
```

1. Register keyboard Function

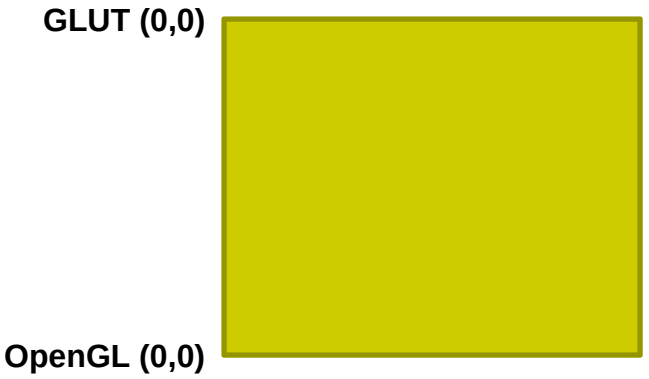
2. Implement mouse function

```
void myMouse(int button, int state, int
   x, int y)
{    // put mouse stuff here




        ……
        }
```

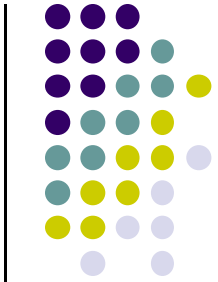
Mouse Interaction



- Declare prototype
  - **`myMouse(int button, int state, int x, int y)`**
  - **`myMovedMouse`**
- Register callbacks:
  - **`glutMouseFunc(myMouse)`:** mouse button pressed
  - **`glutMotionFunc(myMovedMouse)`:** mouse moves with button pressed
  - **`glutPassiveMotionFunc(myMovedMouse)`:** mouse moves with no buttons pressed
- Button returned values:
  - GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
- State returned values:
  - GLUT_UP, GLUT_DOWN
- X,Y returned values:
  - x,y coordinates of mouse location
  - Convert GLUT (0,0) to GLUT (0,0)? How?



Mouse Interaction Example

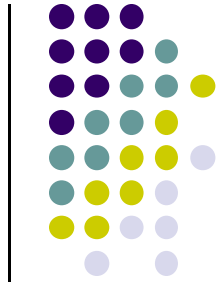


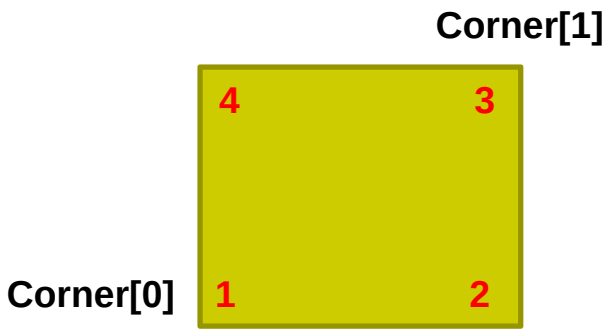
- **Example:** draw (or select) rectangle on screen
- Each mouse click generates separate events
- Store click points in **global** or **static** variable in mouse function

```
void myMouse(int button, int state, int x, int y)
{
   static GLintPoint corner[2];
   static int numCorners = 0;   // initial value is 0
   if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
   {
        corner[numCorners].x = x;
        corner[numCorners].y = screenHeight – y; //flip y coord
        numCorners++;
```

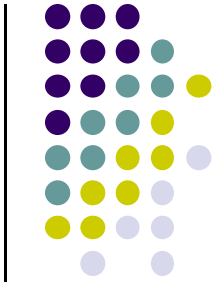
Screenheight is height of drawing window

Mouse Interaction Example (continued)






```
   if(numCorners == 2)
   {
        // draw rectangle or do whatever you planned to do
        Point3 points[4] = corner[0].x, corner[0].y,   //1
                           corner[1].x, corner[0].y,   //2
                           corner[1].x, corner[1].y,   //3
                            corner[0].x, corner[1].y);  //4

         glDrawArrays(GL_QUADS, 0, 4);

        numCorners == 0;
   }
   else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        glClear(GL_COLOR_BUFFER_BIT); // clear the window
   glFlush( );
}
```



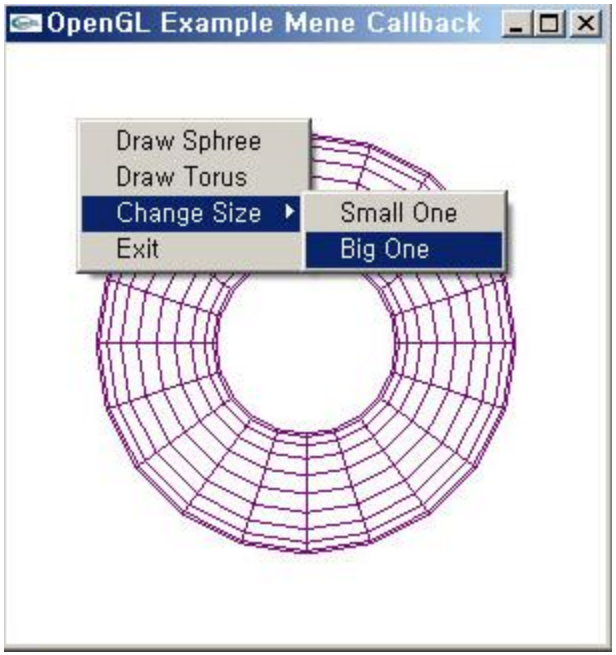
Menus

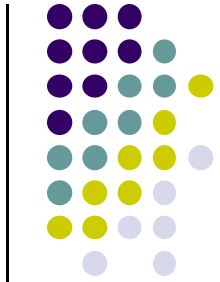
- Adding menu that pops up on mouse click

1. Create menu using **`glutCreateMenu(myMenu);`**

2. Use **`glutAddMenuEntry`** adds entries to menu

3. Attach menu to mouse button (left, right, middle) using **`glutAttachMenu`**





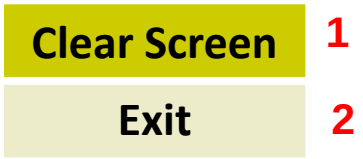
Menus

- Example:

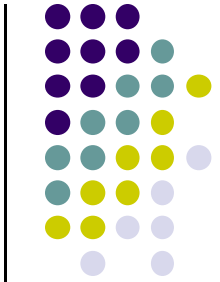
Shows on menu

Checked in mymenu

```
glutCreateMenu(myMenu);
 glutAddMenuEntry("Clear Screen", 1);
 glutAddMenuEntry("Exit", 2);
 glutAttachMenu(GLUT_RIGHT_BUTTON);

 ….

 void mymenu(int value){
    if(value == 1){
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush( );
    }
    if (value == 2) exit(0);
}
```



GLUT Interaction using other input devices



- Tablet functions (mouse cursor must be in display window)

```
glutTabletButton(tabletFcn);
…..
void tabletFcn(Glint tabletButton, Glint action, Glint
   xTablet, Glint yTablet)
```

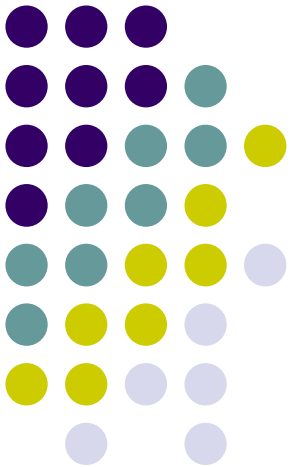
- Spaceball functions
- Dial functions
- Picking functions: use your finger
- Menu functions: minimal pop-up windows within your drawing window
- Reference: *Hearn and Baker, 3rd edition (section 20-6)*

Computer Graphics (CS 4731)
Lecture 6: Shader Setup & GLSL Introduction

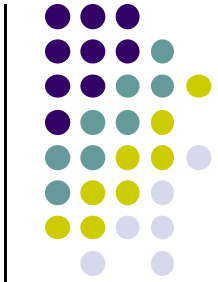
Prof Emmanuel Agu

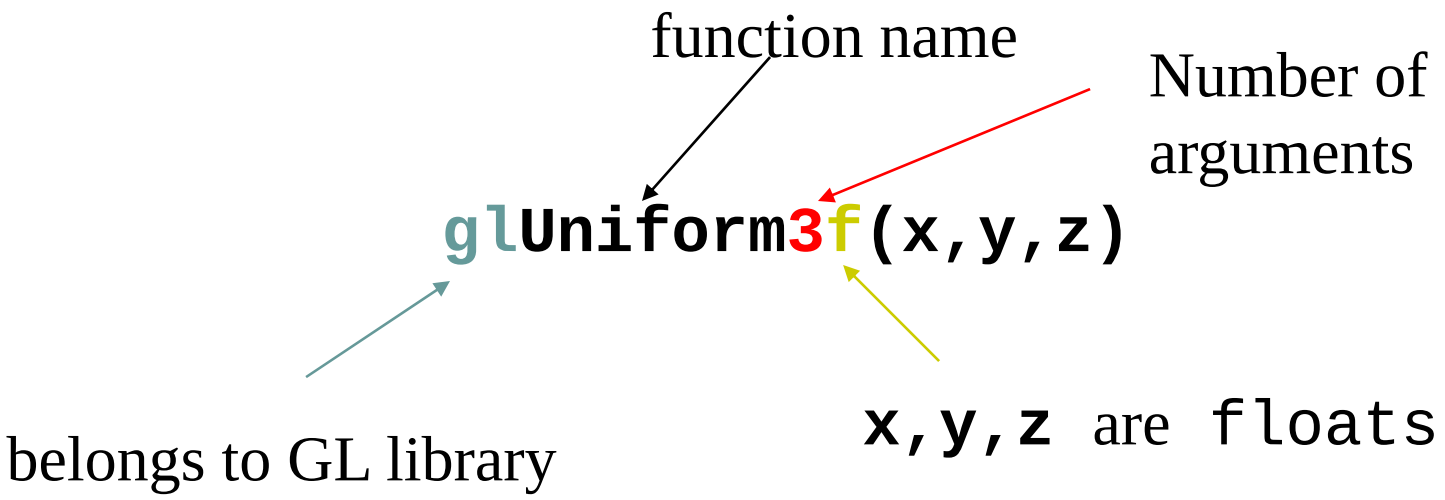
Computer Science Dept.

Worcester Polytechnic Institute (WPI)



OpenGL function format

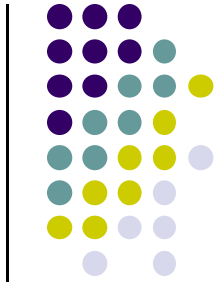




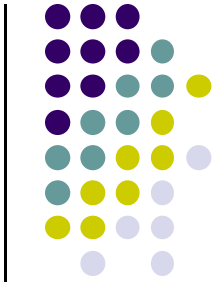

```
glUniform3fv(p)
```

Argument is array of values
p is a pointer to array

Lack of Object Orientation



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

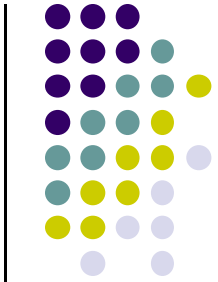


OpenGL Data Types

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

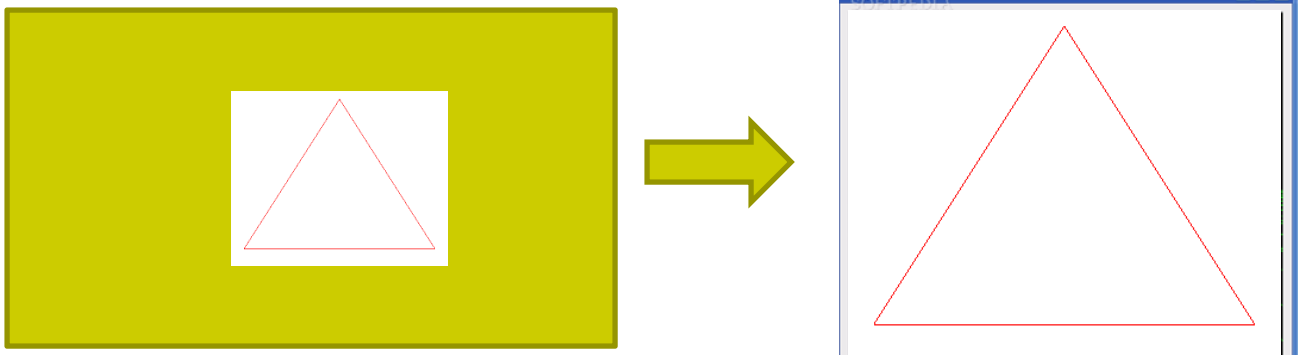
Recall: Single Buffering



- 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



Single Frame buffer

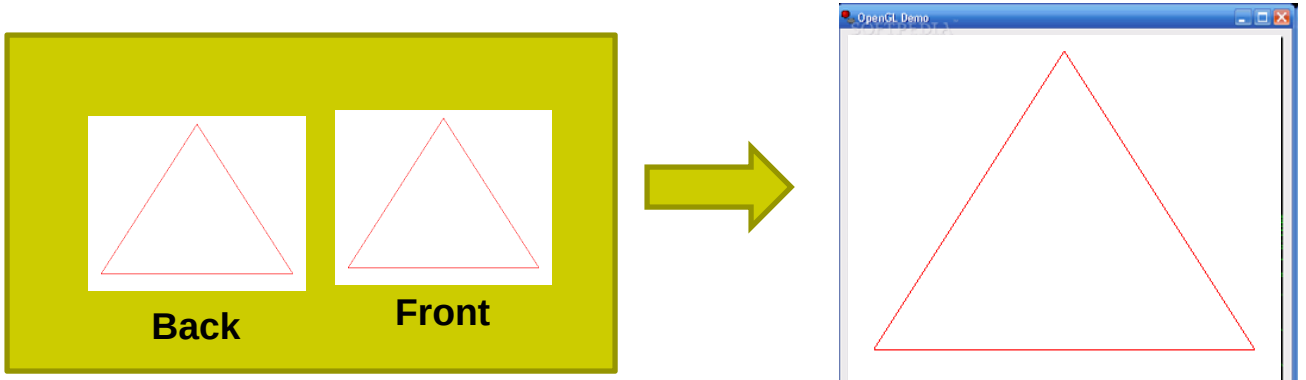
Double Buffering

- 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( )`

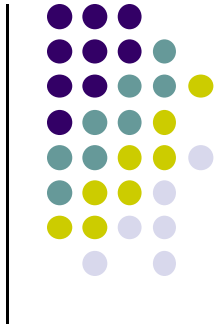
```
void mydisplay(void){
        glClear(GL_COLOR_BUFFER_BIT); // clear screen
        glDrawArrays(GL_POINTS, 0, N);
        glutSwapBuffers( );
}
```

Back buffer drawing swapped in, becomes visible here



Double Frame buffer

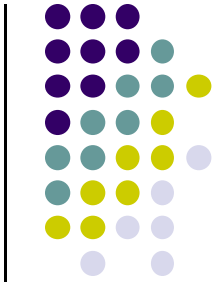
Recall: OpenGL Skeleton



```
void main(int argc, char** argv){
   glutInit(&argc, argv);   // initialize toolkit
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 150);
   glutCreateWindow("my first attempt");
   glewInit( );

   // … now register callback functions
   glutDisplayFunc(myDisplay);
   glutReshapeFunc(myReshape);
   glutMouseFunc(myMouse);
   glutKeyboardFunc(myKeyboard);

   glewInit( );
   generateGeometry( );
   initGPUBuffers( );
   void shaderSetup( );

   glutMainLoop( );
}
```

```
void shaderSetup( void )
{
   // Load shaders and use the resulting shader program
   program = InitShader( "vshader1.glsl", "fshader1.glsl" );
   glUseProgram( program );

   // Initialize vertex position attribute from vertex shader
   GLuint loc = glGetAttribLocation( program, "vPosition" );
   glEnableVertexAttribArray( loc );
   glVertexAttribPointer( loc, 2, GL_FLOAT, GL_FALSE, 0,
                                                BUFFER_OFFSET(0) );

   // sets white as color used to clear screen
   glClearColor( 1.0, 1.0, 1.0, 1.0 );
}
```

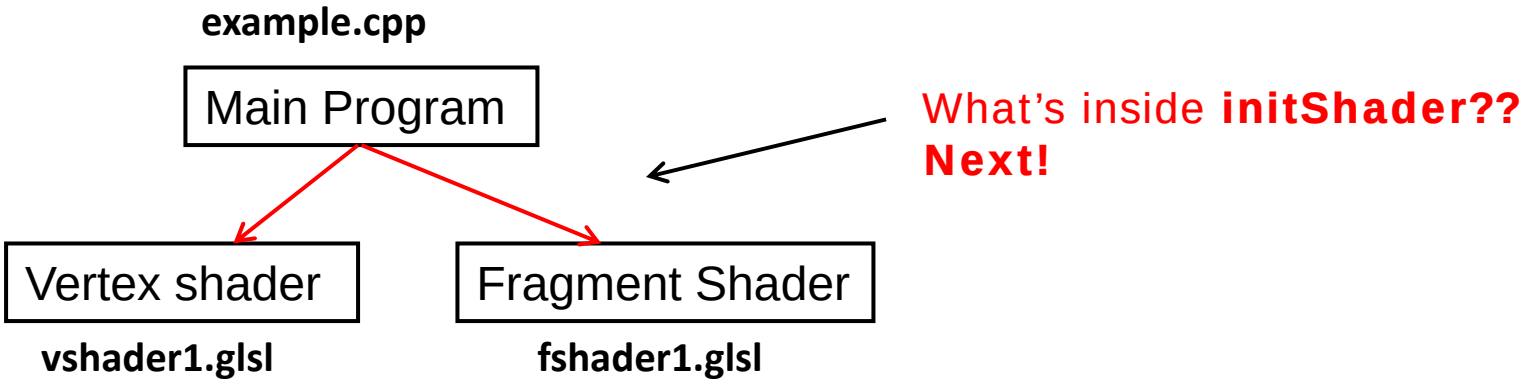
Recall: OpenGL Program: Shader Setup



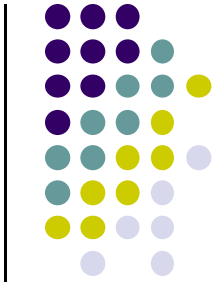
- **`initShader( )`**: our homegrown shader initialization
  - Used in main program, connects and link vertex, fragment shaders
  - Shader sources read in, compiled and linked

```
Gluint = program;
```

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

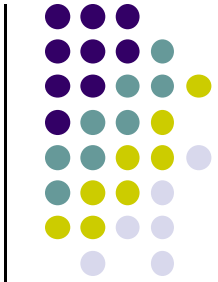


Coupling Shaders to Application (initShader function)



1. Create a program object
2. Read shaders
3. Add + Compile shaders
4. Link program (everything together)
5. Link variables in application with variables in shaders
   - Vertex attributes
   - Uniform variables

Step 1. Create Program Object



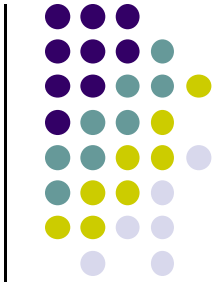
- Container for shaders
  - Can contain multiple shaders, other GLSL functions

```
GLuint myProgObj;

myProgObj = glCreateProgram();
```

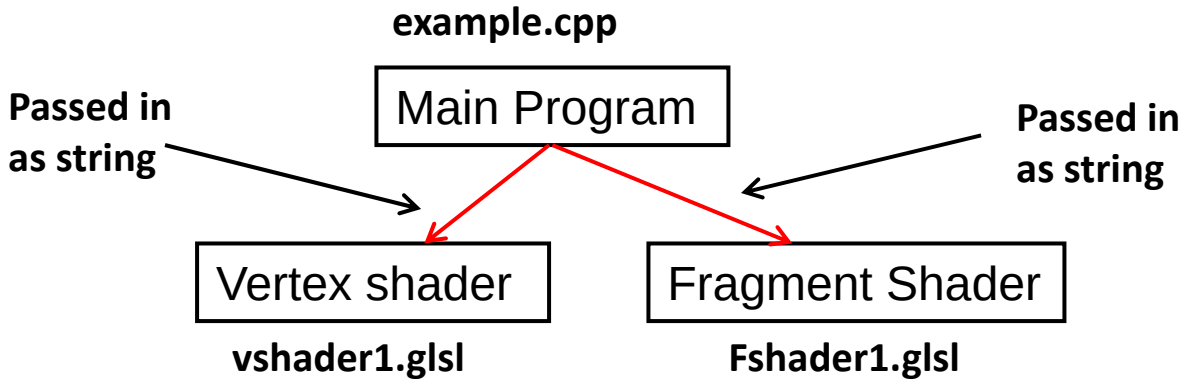
Create container called **Program Object**

Main Program

Step 2: Read a Shader

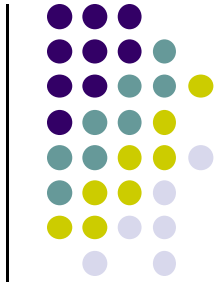


- Shaders compiled and added to program object



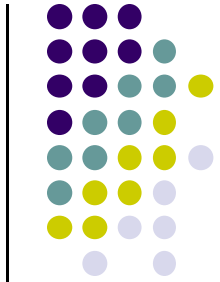
- Shader file code passed in as null-terminated string using the function **`glShaderSource`**
- Shaders in files (vshader.glsl, fshader.glsl), write function **`readShaderSource`** to convert shader file to string

Shader Reader Code?



```
#include <stdio.h>

static char* readShaderSource(const char* shaderFile)
{
    FILE* fp = fopen(shaderFile, "r");

    if ( fp == NULL ) { return NULL; }

    fseek(fp, 0L, SEEK_END);
    long size = ftell(fp);

    fseek(fp, 0L, SEEK_SET);
    char* buf = new char[size + 1];
    fread(buf, 1, size, fp);

    buf[size] = '\0';
    fclose(fp);

    return buf;
}
```

Step 3: Adding + Compiling Shaders



```
GLuint myVertexObj;
Gluint myFragmentObj;
```

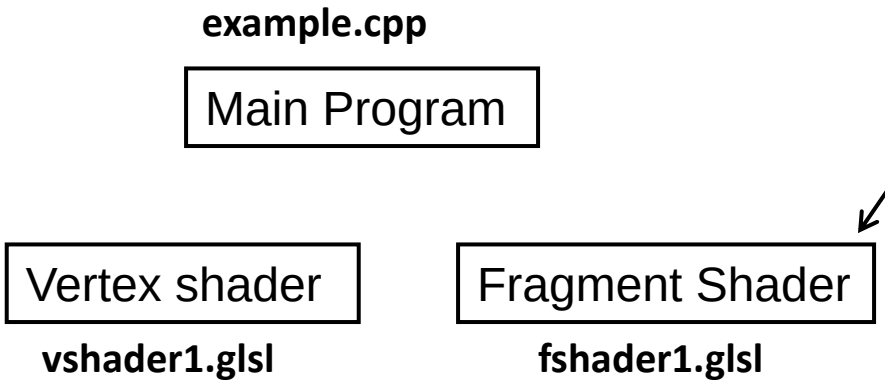
Declare shader object
(container for shader)

```
GLchar* vSource = readShaderSource("vshader1.glsl");
GLchar* fSource = readShaderSource("fshader1.glsl");
```

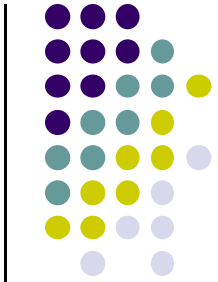
Read shader files,
Convert code
to string

```
myVertexObj = glCreateShader(GL_VERTEX_SHADER);
myFragmentObj = glCreateShader(GL_FRAGMENT_SHADER);
```

Create empty
Shader objects



Step 3: Adding + Compiling Shaders
Step 4: Link Program



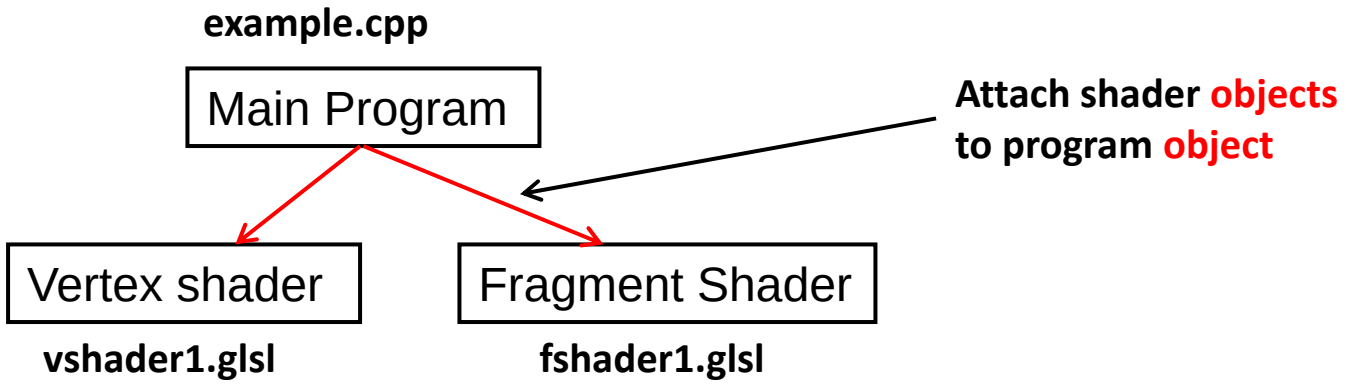
Read shader code strings into shader objects

```
glShaderSource(myVertexObj, 1, vSource, NULL);
glShaderSource(myFragmentObj, 1, fSource, NULL);

glCompileShader(myVertexObj);
glCompileShader(myFragmentObj);

glAttachShader(myProgObj, myVertexObj);
glAttachShader(myProgObj, myFragmentObj);

glLinkProgram(myProgObj);
```

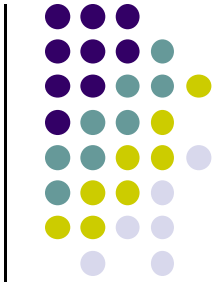
Compile shader objects

Attach shader objects to program object

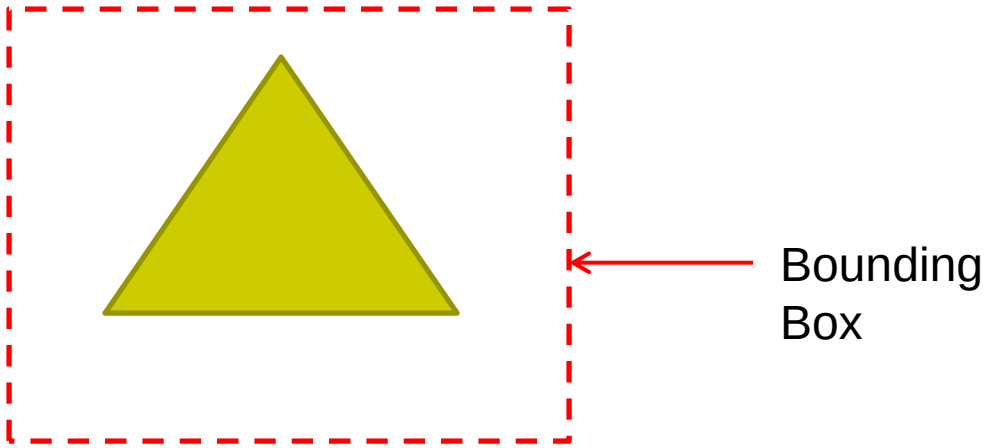
Link Program



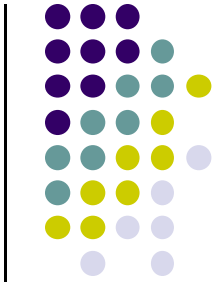
Uniform Variables



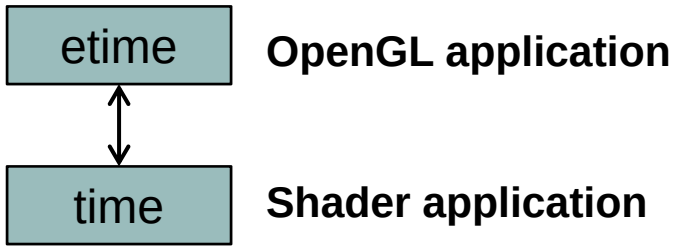
- Variables that are **constant** for an entire primitive
- Can be changed in application and sent to shaders
- Cannot be changed in shader
- Used to pass information to shader
  - **Example:** bounding box of a primitive



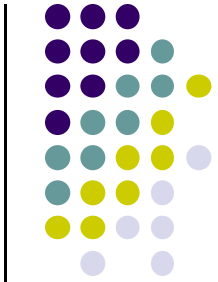
Uniform variables



- Sometimes want to connect uniform variable in OpenGL application to uniform variable in shader
- Example?
  - Check “elapsed time” variable (etime) in OpenGL application
  - Use elapsed time variable (time) in shader for calculations



Uniform variables



- First declare **etime** variable in OpenGL application, get time

```
float etime;

etime = 0.001*glutGet(GLUT_ELAPSED_TIME);
```

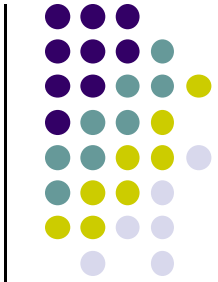
Elapsed time since program started

- Use corresponding variable **time** in shader

```
uniform float time;
attribute vec4 vPosition;

main(  ){
     vPosition.x  += (1+sin(time));
    gl_Position = vPosition;
}
```

- Need to connect **etime** in application and **time** in shader!!

Connecting **etime** and **time**



- Linker forms table of shader variables, each with an index
- Application can get index from table, tie it to application variable
- In application, find location of shader **time** variable in linker table

```
Glint timeLoc;

timeLoc = glGetUniformLocation(program, "time");
```

423	time

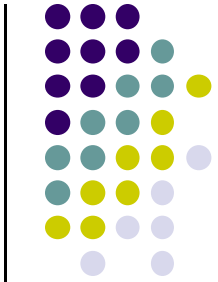
- Connect: **location** of shader variable **time** to **etime**!

```
glUniform1(timeLoc, etime);
```

423	etime

Location of shader variable time

Application variable, etime



References

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