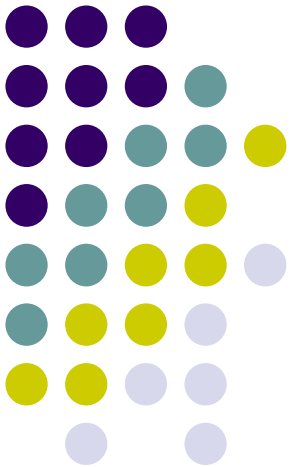
Computer Graphics (CS 4731) Lecture 2: Introduction to OpenGL/GLUT (Part 1)

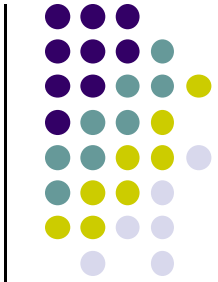
Prof Emmanuel Agu

Computer Science Dept.

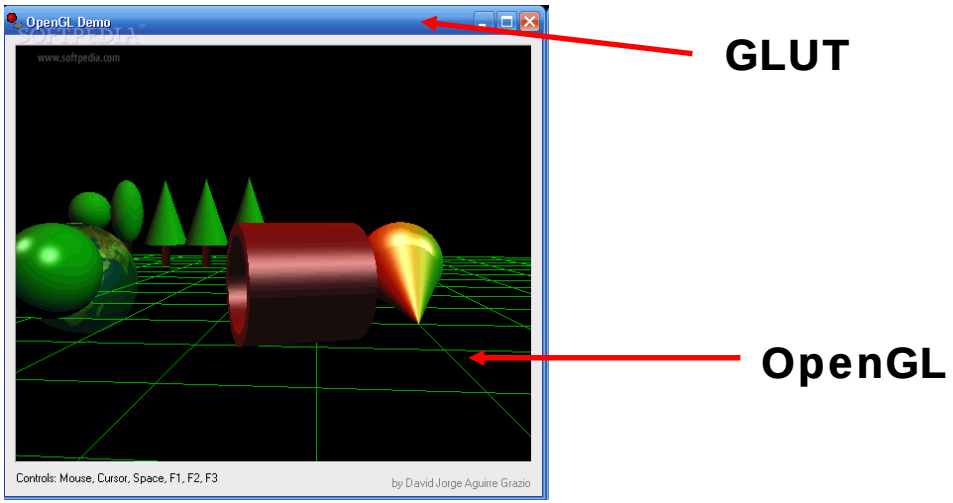
Worcester Polytechnic Institute (WPI)



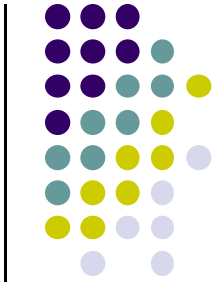
Recall: OpenGL/GLUT Basics



- OpenGL's function – Rendering (2D, 3D drawings or images)
- OpenGL does not manage drawing window
- GLUT: minimal window management

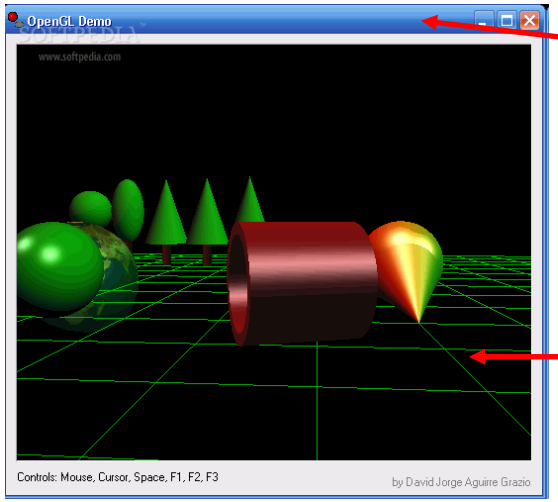


OpenGL/GLUT Installation



- **OpenGL:** Specific version (e.g. 4.3)already on your graphics card
  - Just need to check your graphics card, OpenGL version

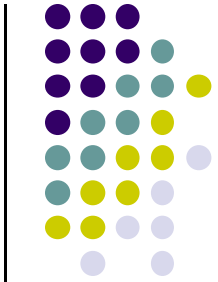
- **GLUT:** software that needs to be installed
  - already installed in zoolab machines



GLUT: install it!

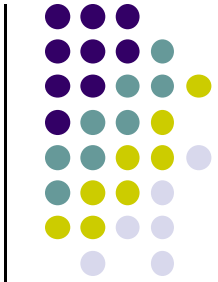
OpenGL: already on graphics card

glInfo: Finding out about your Graphics Card

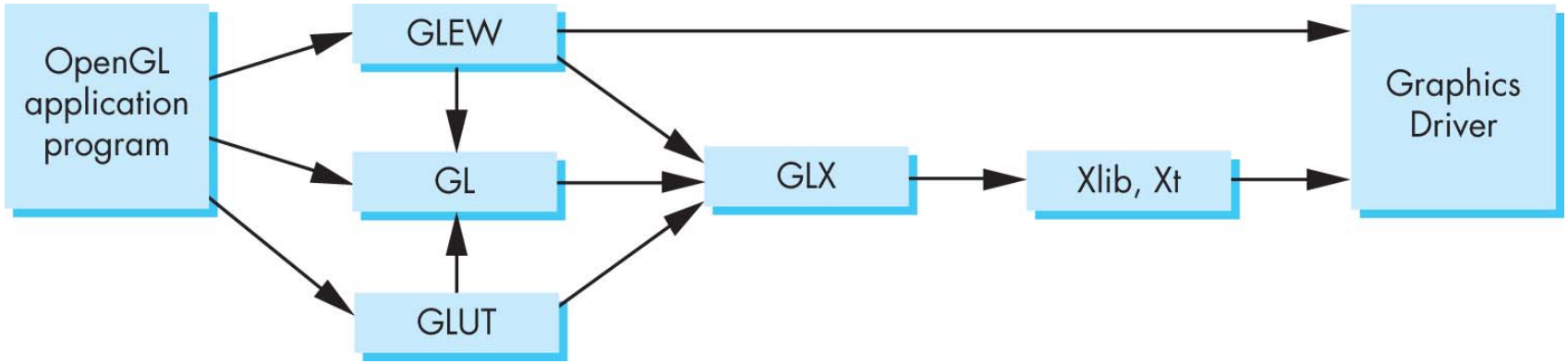


- Software tool to find out OpenGL version and extensions your graphics card supports
- This class? Need graphics card that supports OpenGL 4.3 or later

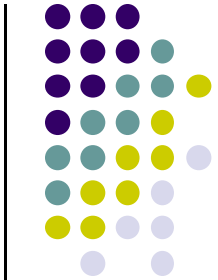
OpenGL Extension Wrangler Library (GLEW)



- **OpenGL extensions:** allows individual card manufacturers to implement new features
- **Example:** If card manufacturer maker implements new cool features after OpenGL version 4.5 released, make available as extension to OpenGL 4.5
- **GLEW:** easy access to OpenGL extensions available on a particular graphics card
- We install GLEW as well. Access to extensions on zoolab cards

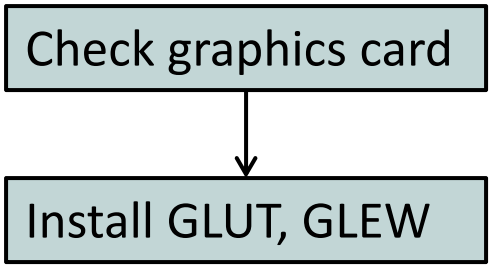


Windows Installation of GLUT, GLEW



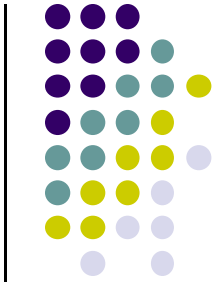
- Install Visual Studio (e.g 2010)

- Download freeglut **32-bit** (GLUT implementation)
  - http://freeglut.sourceforge.net/
- Download **32-bit** GLEW
  - http://glew.sourceforge.net/

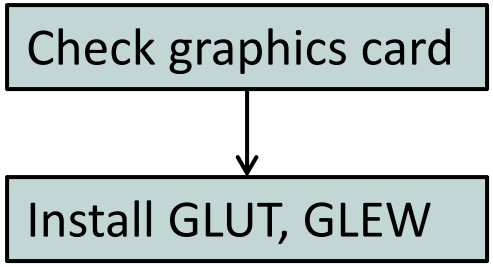


- Unzip => .lib, .h, .dll files
- E.g. download freeglut 2.8.1, files:
  - freeglut.dll
  - glut.h
  - freeglut.lib

Windows Installation of GLUT, GLEW

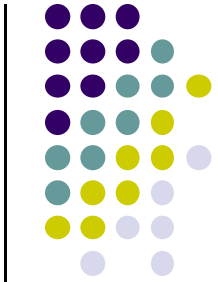


- E.g. download freeglut 2.8.1, files:
  - freeglut.dll
  - glut.h
  - freeglut.lib

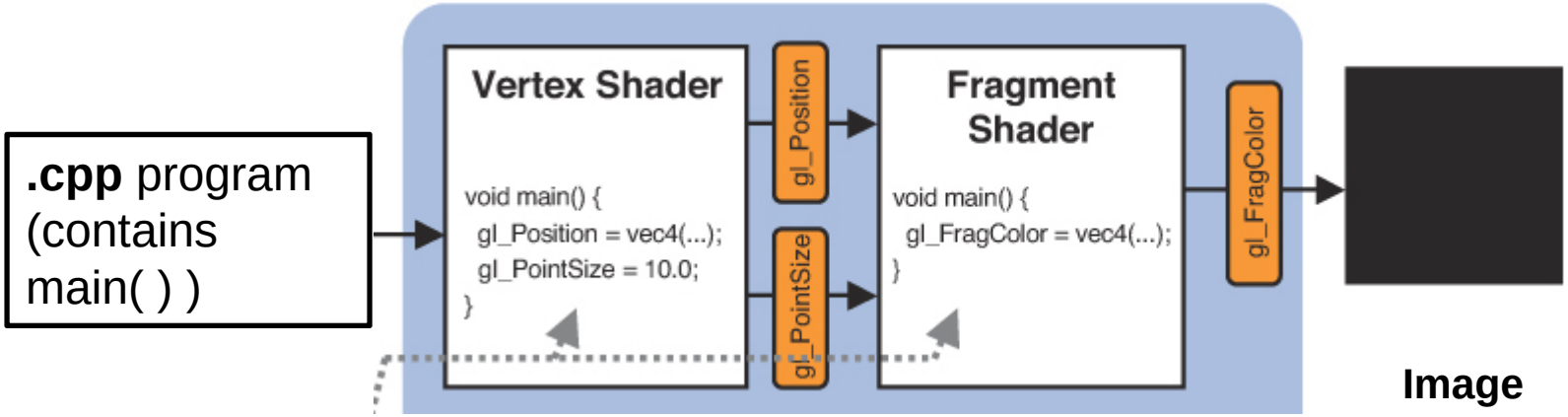


- Install files:
  - Put .dll files (for GLUT and GLEW) in C:\windows\system
  - Put .h files in c:\Visual Studio…\include\ directory
  - Put .lib files in c:\Visual Studio….\lib\ directory
- **Note:** If you have multiple versions of Visual Studio, use include directory of the highest Visual Studio version
  - E.g. if you have Visual Studio 2008 + Visual Studio 2010
  - Use include, lib directories of Visual Studio 2010

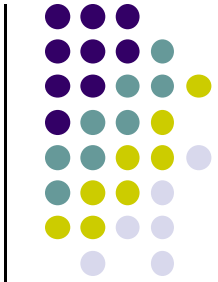
OpenGL Program?



- Usually has 3 files:
  - **Main .cpp file:** containing your main function
    - Does initialization, generates/loads geometry to be drawn
  - 2 shader files:
    - **Vertex shader:** functions to manipulate (e.g. move) vertices
    - **Fragment shader:** functions to manipulate pixels/fragments (e.g change color)

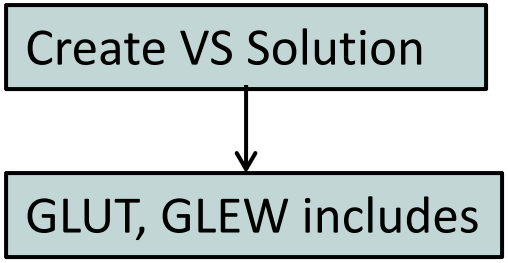


Getting Started: Writing .cpp In Visual studio



1. Create empty project
2. Create blank console application (C program)
3. Include **`glew.h`** and **`glut.h`** at top of your program

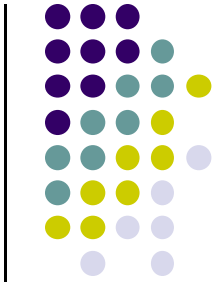
```
#include <glew.h>
#include <GL/glut.h>
```



Note: `GL/` is sub-directory of compiler **`include/`** directory

- OpenGL drawing functions in **`gl.h`**
- **`glut.h`** contains GLUT functions, also includes **`gl.h`**

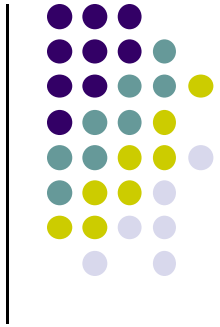
Getting Started: More #includes



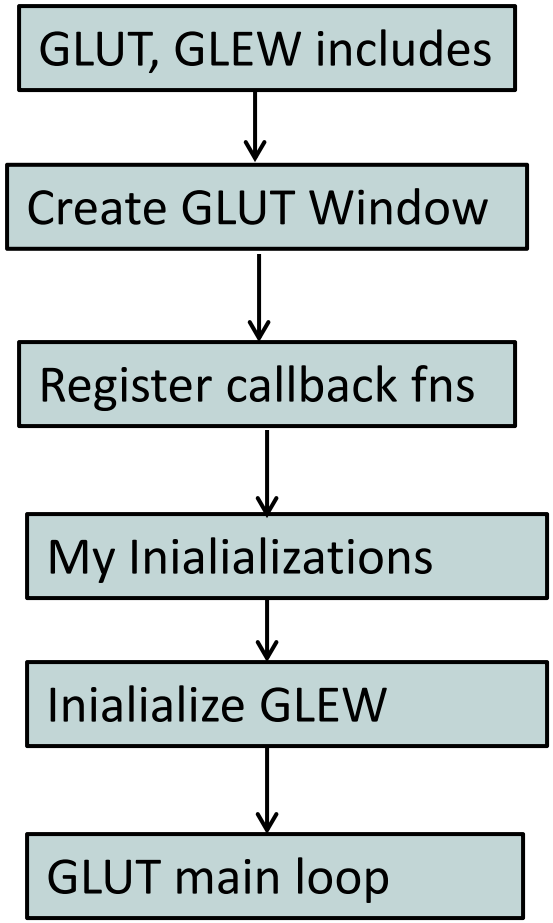
- Most OpenGL applications use standard C library (e.g **`printf`**), so

```
#include <glew.h>
#include <GL/glut.h>

#include <stdlib.h>
#include <stdio.h>
```

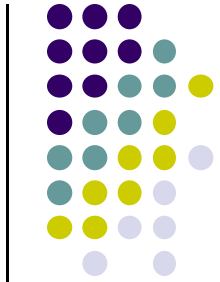
OpenGL/GLUT Program Structure



- Open window (GLUT)
  - Configure display mode, window position/size
- Register input callback functions (GLUT)
  - Render, resize, input: keyboard, mouse, etc
- My initialization
  - Set background color, clear color, etc
  - Generate points to be drawn
  - Initialize shader stuff
- Initialize GLEW
- Register GLUT callbacks
- glutMainLoop( )
  - Waits here infinitely till event

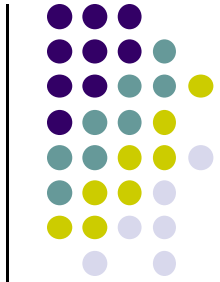


GLUT: Opening a window

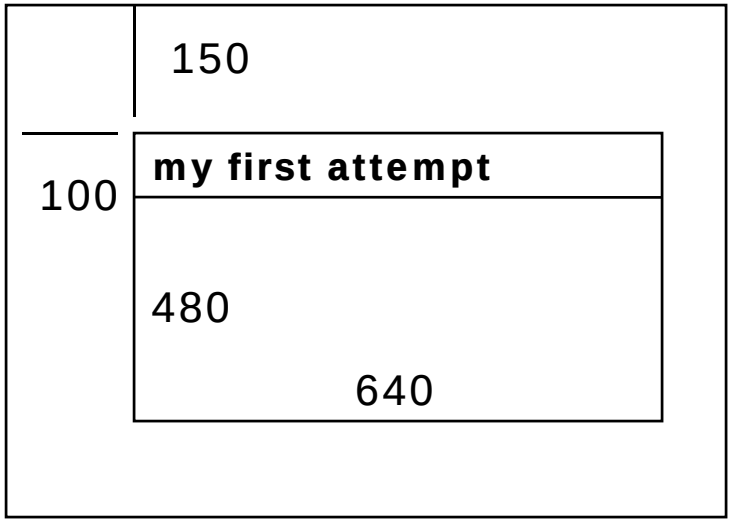


- GLUT used to create and open window
  - **`glutInit(&argc, argv);`**
    - Initializes GLUT
  - **`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);`**
    - sets display mode (e.g. single framebuffer with RGB colors)
  - **`glutInitWindowSize(640,480);`**
    - sets window size (Width x Height) in pixels
  - **`glutInitPosition(100,150);`**
    - sets location of upper left corner of window
  - **`glutCreateWindow("my first attempt");`**
    - open window with title "my first attempt"
- Then also initialize GLEW
  - **`glewInit( );`**

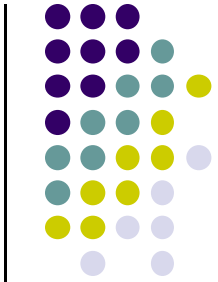
OpenGL Skeleton



```
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( );

    // … then register callback functions,
    // … do my initialization
    // .. wait in glutMainLoop for events

}
```

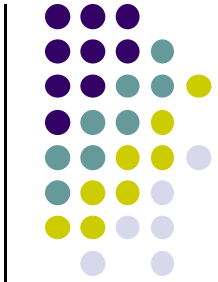


Sequential Vs Event-driven

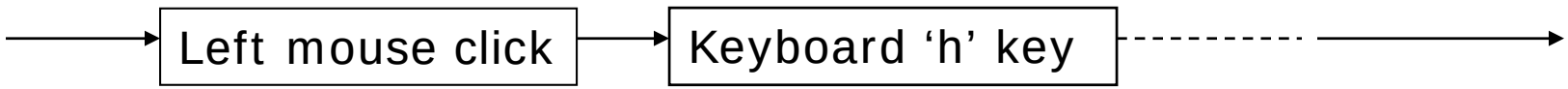


- OpenGL programs are event-driven
- Sequential program
  - **Start at main( )**
  - **Perform actions 1, 2, 3…. *N***
  - **End**
- Event-driven program
  - **Start at main( )**
  - **Initialize**
  - **Wait in infinite loop**
    - **Wait till defined event occurs**
    - **Event occurs => Take defined actions**
- What is World's most famous event-driven program?

OpenGL: Event-driven

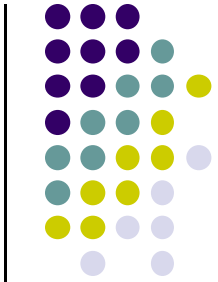


- Program only responds to events
- Do nothing until event occurs
- Example Events:
  - **mouse clicks,**
  - **keyboard stroke**
  - **window resize**
- Programmer defines:
  - Events that program should respond to
  - Actions to be taken when event occurs
- System (Windows):
  - Receives event, maintains event queue

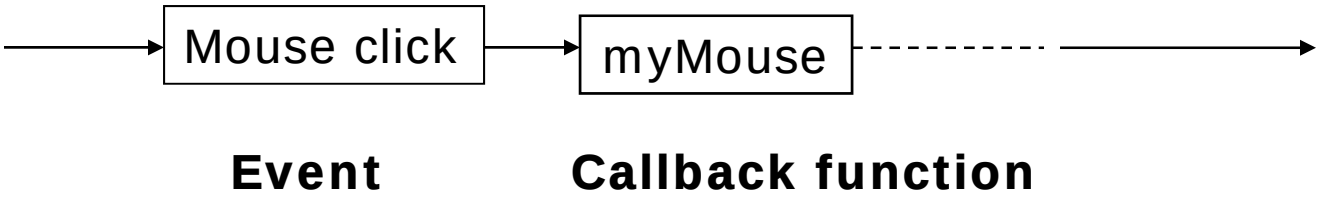


  - takes programmer-defined actions

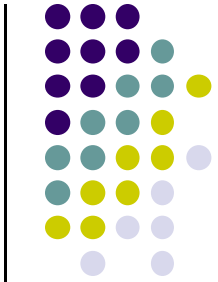
OpenGL: Event-driven



- How in OpenGL?
  - Programmer registers callback functions (event handler)
  - Callback function called when event occurs
- Example: Programmer
  1. Declare function *myMouse*, to be called on mouse click
  2. Register it: glutMouseFunc(*myMouse*);
- When OS receives mouse click, calls callback function **myMouse**

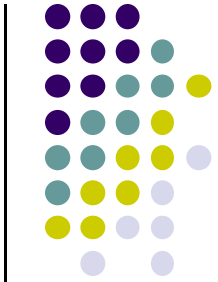


GLUT Callback Functions

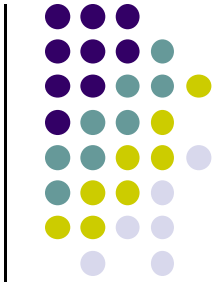


- Register callbacks for all events your program will react to
- No registered callback = no action
- Example: if no registered keyboard callback function, hitting keyboard keys generates NO RESPONSE!!

GLUT Callback Functions



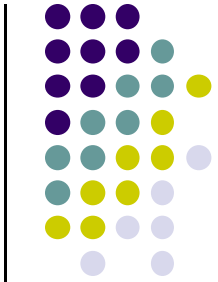
- GLUT Callback functions in skeleton
  - **`glutDisplayFunc(myDisplay);`** Image to be drawn initially
  - **`glutReshapeFunc(myReshape);`** called when window is reshaped
  - **`glutMouseFunc(myMouse);`** called when mouse button is pressed
  - **`glutKeyboardFunc(mykeyboard);`** called when keyboard is pressed or released
- **`glutMainLoop( );`**
  - program draws initial picture (by calling myDisplay function once)
  - Enters infinite loop till event



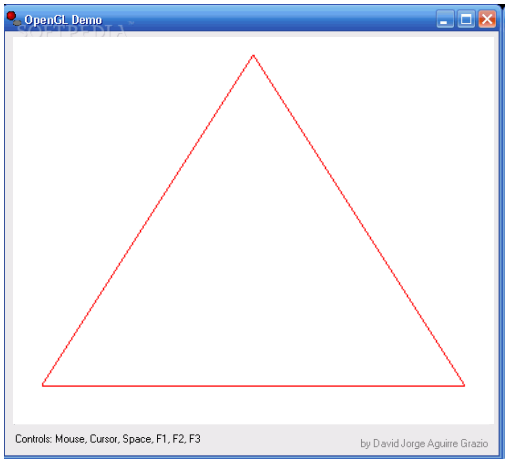
OpenGL Skeleton

```
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);  <--Next… how to draw in myDisplay
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

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

Example: Draw in function myDisplay

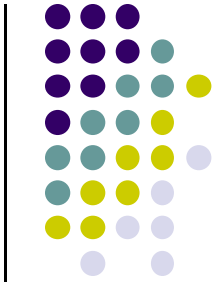


- **Task:** Draw red triangle on white background

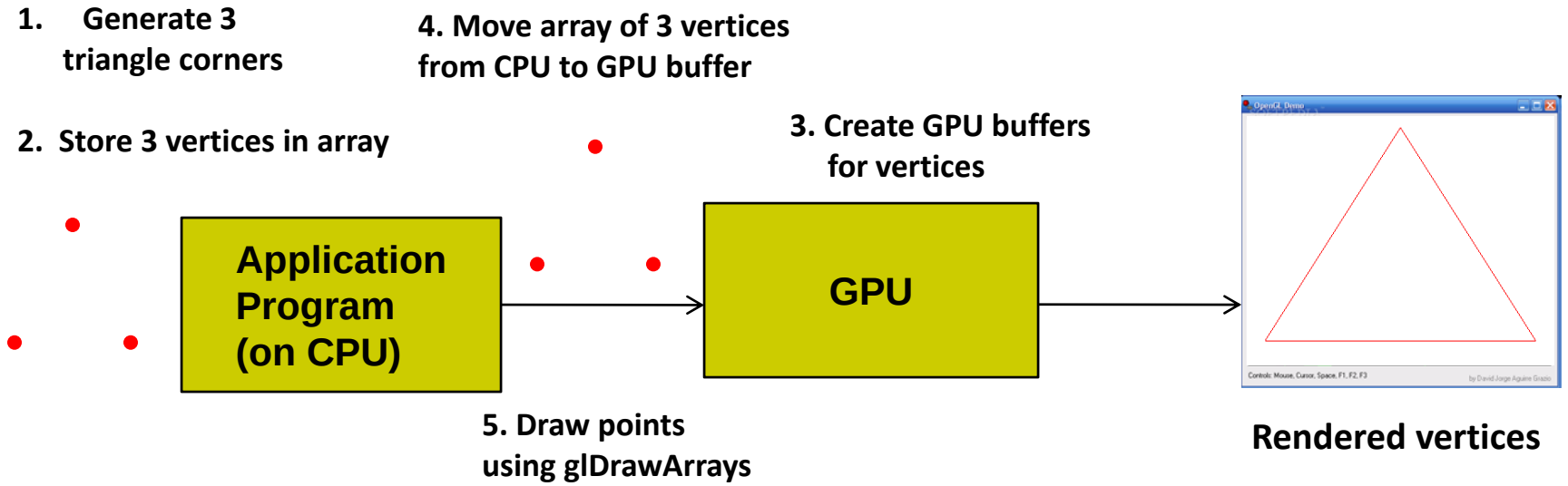


- **Rendering steps:**
  1. Generate triangle corners (3 vertices)
  2. Store 3 vertices into an array
  3. Create GPU buffer for vertices
  4. Move 3 vertices from CPU to GPU buffer
  5. Draw 3 points from array on GPU using **`glDrawArray`**

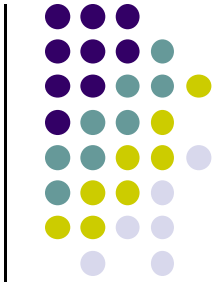
Example: Retained Mode Graphics



- **Rendering steps:**
  1. Generate triangle corners (3 vertices)
  2. Store 3 vertices into an array
  3. Create GPU buffer for vertices
  4. Move array of 3 vertices from CPU to GPU buffer
  5. Draw 3 points from array on GPU using **`glDrawArray`**
- **Simplified Execution model:**

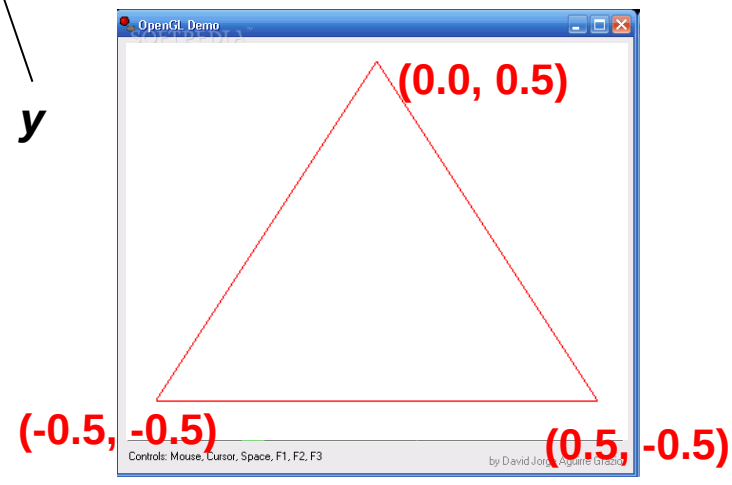


1. Generate triangle corners (3 vertices)
2. Store 3 vertices into an array

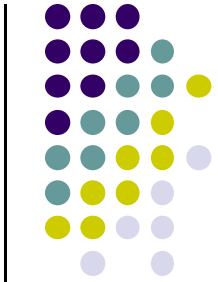


```
point2 points[3];

// generate 3 triangle vertices + store in array
void generateGeometry( void ){
      points[0] = point2( -0.5, -0.5 );
      points[1] = point2( 0.0, 0.5 );
      points[2] = point2( 0.5, -0.5 );
}
```

x *y*



Declare some Types for Points, vectors



- Useful to declare types
  - ***point2*** for (x,y) locations
  - ***vec3*** for (x,y,z) vector coordinates
- Put declarations in *header file vec.h*

```
#include "vec.h"
```
← Declares (x, y, z) coordinates of a vector

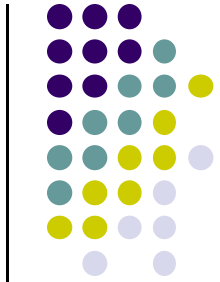
```
E.g    vec3 vector1;
```

- Can also do typedefs ← typedef (x, y) coordinates of a point

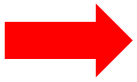
```
typedef vec2 point2;
```

- **Note:** You will be given file Angel.h, which includes vec.h

OpenGL Skeleton: Where are we?

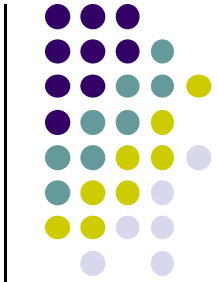


```
void main(int argc, char** argv){
   glutInit(&argc, argv);   // initialize toolkit
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 150);
   glutCreateWindow("my first attempt");
   glewInit( );

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

   glewInit( );
   generateGeometry( );

   glutMainLoop( );
}
```



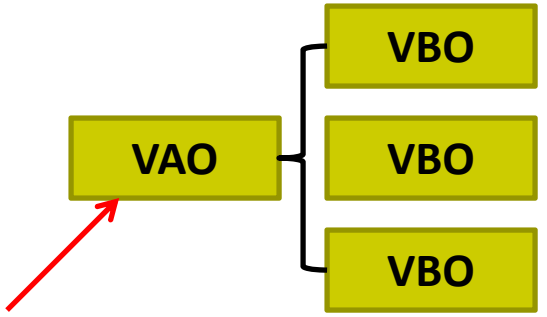
```
// generate 3 triangle vertices + store in array
void generateGeometry( void ){
          points[0] = point2( -0.5, -0.5 );
          points[1] = point2( 0.0, 0.5 );
          points[2] = point2( 0.5, -0.5 );
}
```

3. Create GPU Buffer for Vertices

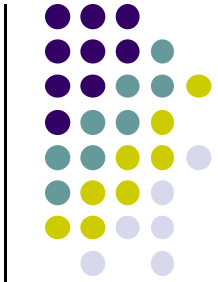


- Rendering from GPU memory significantly faster. Move data there
- Fast GPU (off-screen) memory for data called ***Vertex Buffer Objects (VBO)***
- Array of VBOs (called ***Vertex Array Object (VAO)***) usually created
- Example use: vertex positions in VBO 1, color info in VBO 2, etc

- So, first create the vertex array object




```
GLuint vao;

glGenVertexArrays( 1, &vao );   // create VAO
glBindVertexArray( vao );       // make VAO active
```



3. Create GPU Buffer for Vertices

- Next, create a buffer object in two steps
  1. Create VBO and give it name (unique ID number)

```
GLuint buffer;
glGenBuffers(1, &buffer);  // create one buffer object
```

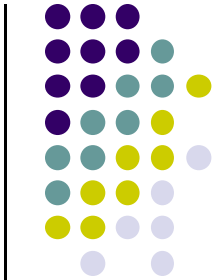
Number of Buffer Objects to return

  2. Make created VBO currently active one

```
glBindBuffer(GL_ARRAY_BUFFER, buffer);
```

Data is array of values

4. Move points GPU memory



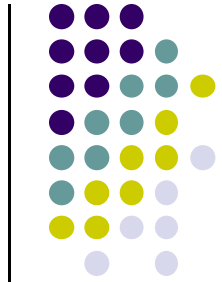
3. Move **`points`** generated earlier to VBO

```
glBufferData(GL_ARRAY_BUFFER, buffer, sizeof(points),
points, GL_STATIC_DRAW ); //data is array
```

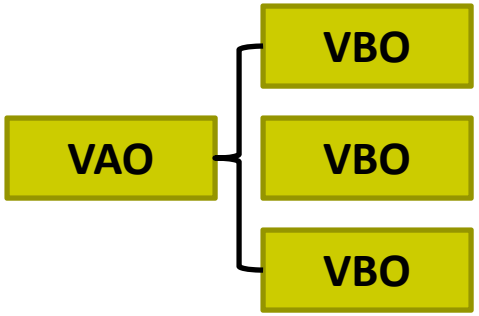
Data to be transferred to GPU memory (generated earlier)

- **`GL_STATIC_DRAW:`** buffer object data will not be changed. Specified once by application and used many times to draw
- **`GL_DYNAMIC_DRAW:`** buffer object data will be changed. Specified repeatedly and used many times to draw

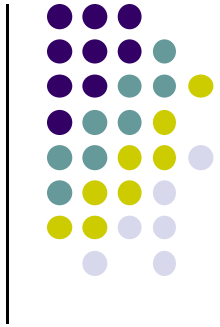
Put it Together:
3. Create GPU Buffer for Vertices
4. Move points GPU memory



```
void initGPUBuffers( void )
{
        // Create a vertex array object
        GLuint vao;
        glGenVertexArrays( 1, &vao );
        glBindVertexArray( vao );

        // Create and initialize a buffer object
        GLuint buffer;
        glGenBuffers( 1, &buffer );
        glBindBuffer( GL_ARRAY_BUFFER, buffer );
        glBufferData( GL_ARRAY_BUFFER, sizeof(points),
                                       points, GL_STATIC_DRAW );
}
```



OpenGL Skeleton: Where are we?



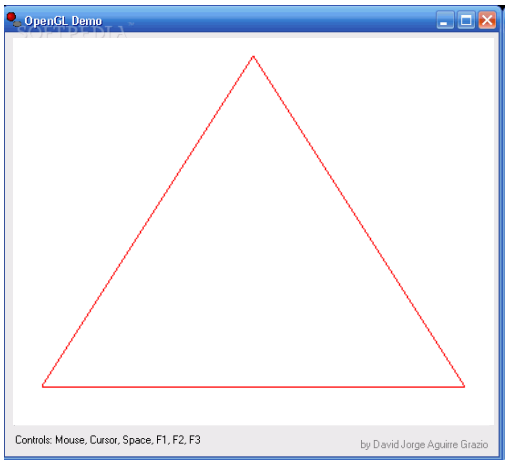
```
void main(int argc, char** argv){
   glutInit(&argc, argv);   // initialize toolkit
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 150);
   glutCreateWindow("my first attempt");
   glewInit( );

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

   glewInit( );
   generateGeometry( );
   initGPUBuffers( );

   glutMainLoop( );
}
```

```
void initGPUBuffers( void )
{
   // Create a vertex array object
   GLuint vao;
   glGenVertexArrays( 1, &vao );
   glBindVertexArray( vao );

   // Create and initialize a buffer object
   GLuint buffer;
   glGenBuffers( 1, &buffer );
   glBindBuffer( GL_ARRAY_BUFFER, buffer );
   glBufferData( GL_ARRAY_BUFFER,
          sizeof(points), points, GL_STATIC_DRAW );
}
```

5. Draw points (from VBO)

```
glDrawArrays(GL_POINTS, 0, N);
```

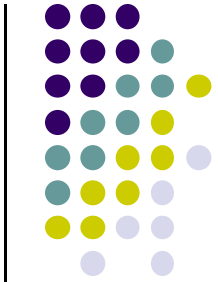
- **Render buffered data as points**
- **Starting index**
- **Number of points to be rendered**



- Display function using **`glDrawArrays:`**

```
void mydisplay(void){
    glClear(GL_COLOR_BUFFER_BIT);    // clear screen
    glDrawArrays(GL_LINE_LOOP, 0, 3);   // draw the points
    glFlush( );                      // force rendering to show
}
```

References



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