

Computer Graphics (CS 543)

Lecture 1b: Introduction to OpenGL/GLUT (Part 1)

Prof Emmanuel Agu

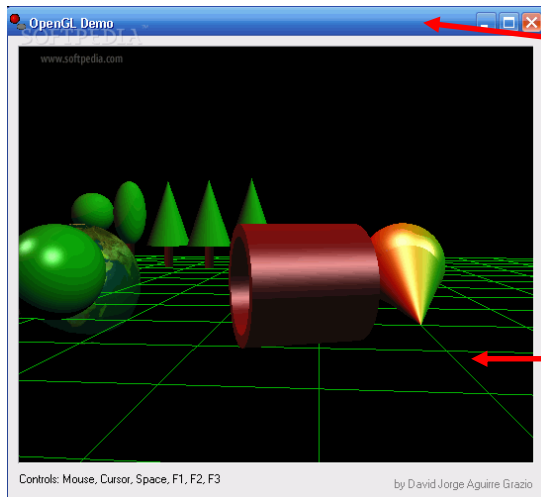
*Computer Science Dept.
Worcester Polytechnic Institute (WPI)*





OpenGL/GLUT Installation

- **OpenGL:** Specific version (e.g. 4.3) on your graphics card hardware
 - Just check OpenGL version on your graphics card
- **GLUT:** software that needs to be installed



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 with OpenGL 4.3 or later

The screenshot shows the GLInfo application window with a blue header and a dark blue body. The header contains the text "GLInfo" in yellow. The body is divided into two main sections. The left section is a sidebar with a yellow background and contains the following items: "Driver info", "Extension lists" (highlighted with a white box and a red arrow), "Implementation specifics", "Extension specifics", "Reports", and "About". The right section has a dark blue background and contains the following text: "Driver version: Unknown", "Vendor: Intel", "Renderer: Intel(R) HD Graphics", and "OpenGL version: 2.1.0 - Build 8.15.10.2202" (highlighted with a white box and a red arrow). A red arrow points from the text "OpenGL extensions on Graphics card" to the "Extension lists" box. Another red arrow points from the text "OpenGL version on Graphics card" to the "OpenGL version" text.

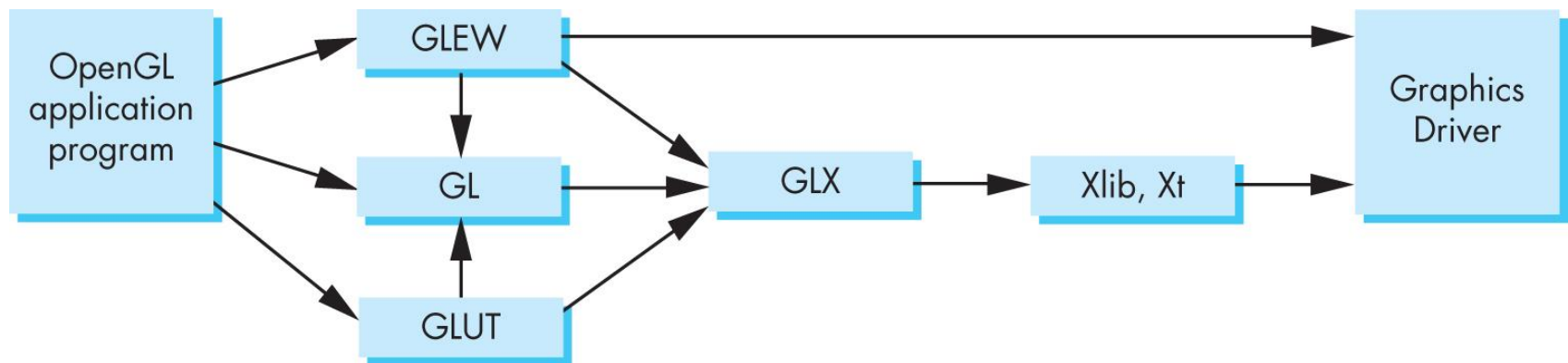
OpenGL extensions on Graphics card

OpenGL version on Graphics card

OpenGL Extension Wrangler Library (GLEW)



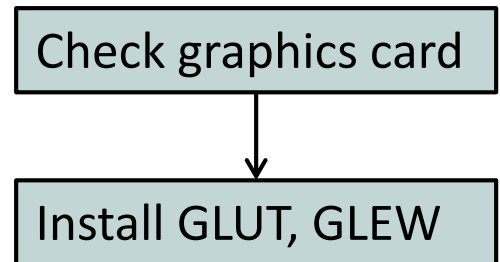
- **OpenGL extensions:** card manufacturers may implement new proprietary features after latest OpenGL version released
 - Published, made available as extensions to latest OpenGL
- **GLEW:** library to access OpenGL extensions on a graphics card
- We will install GLEW as well



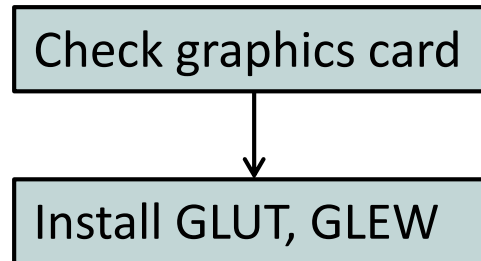


Windows Installation of GLUT, GLEW

- **Note:** GLUT, GLEW already installed in zoolab. Can just go there
- The following instructions only useful if you want to install on your home machine
 1. Install Visual Studio (e.g 2017)
 2. Download freeglut **32-bit** (GLUT implementation)
 - <http://freeglut.sourceforge.net/>
 3. Download **32-bit** GLEW
 - <http://glew.sourceforge.net/>
 4. Unzip GLUT, GLEW => .lib, .h, .dll files
 - E.g. freeglut 3.0.0, files: freeglut.dll, glut.h, freeglut.lib



Windows Installation of GLUT, GLEW



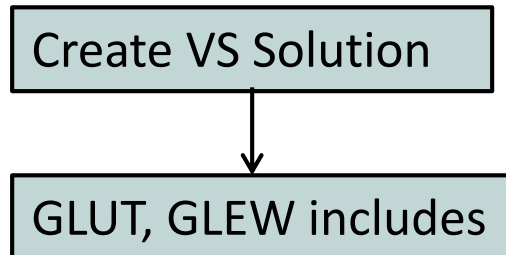
- Install .lib, .h, .dll files (for GLUT and GLEW) from zip files. Put
 - **.dll** files in **c:\windows\system**
 - **.h** files in c:\Visual Studio... **\include** directory
 - **.lib** files in c:\Visual Studio.... **\lib** directory



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>
```

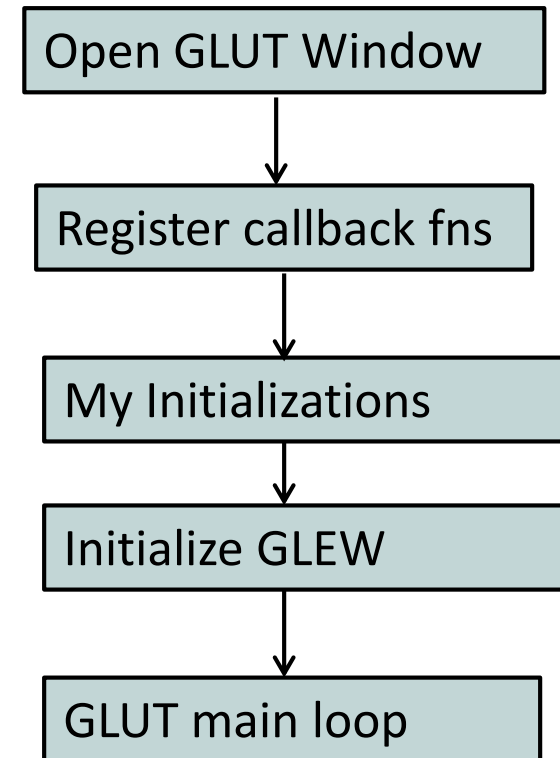
```
#include <stdlib.h>
```

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



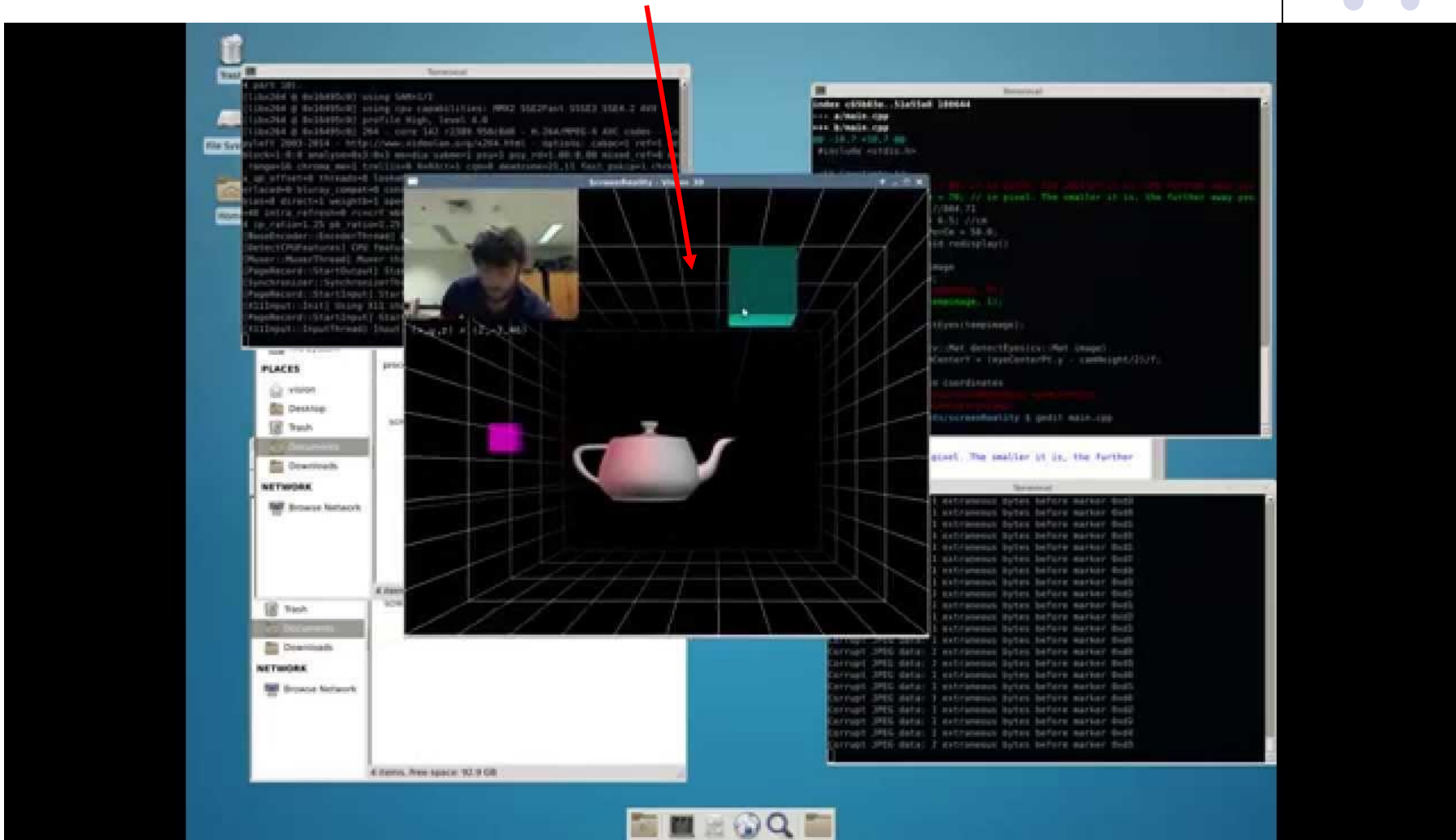

OpenGL/GLUT Program Structure

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





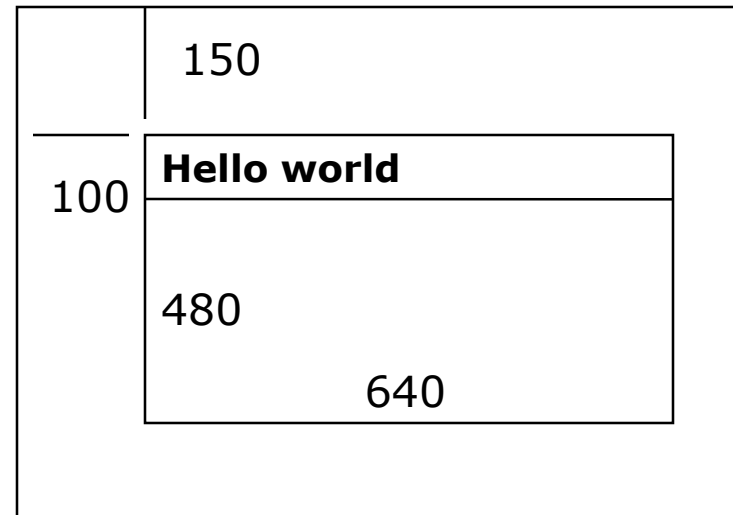
OpenGL Drawing Window?





Opening a GLUT window

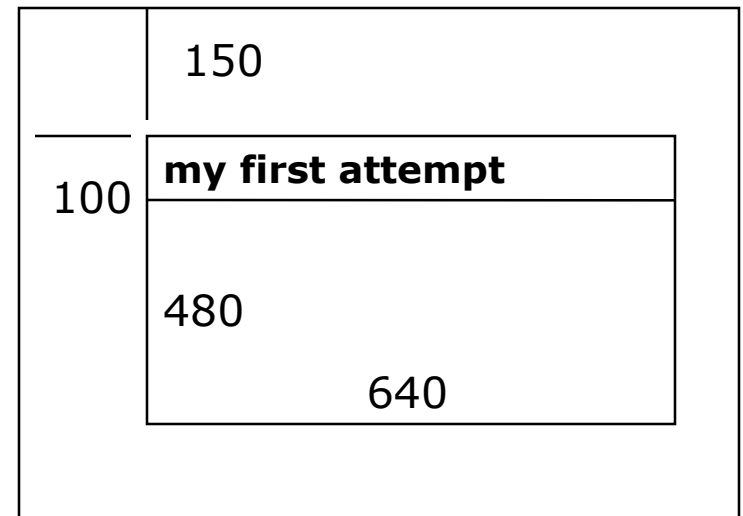
- GLUT Commands used
 - `glutInit(&argc, argv);`
 - Initializes GLUT
 - `glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);`
 - sets display mode (e.g. single framebuffer with RGB colors)
 - `glutInitWindowSize(640, 480);`
 - sets window size (Width x Height) in pixels
 - `glutInitPosition(100, 150);`
 - sets location of upper left corner of window
 - `glutCreateWindow("Hello world");`
 - open window with title "Hello world"
- Then 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  
  
}
```





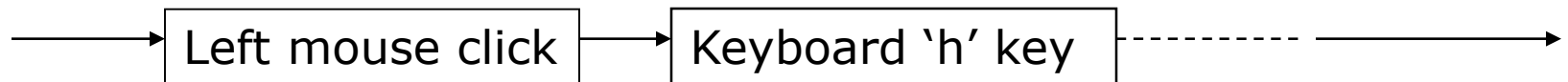
Sequential Vs Event-driven

- 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 widely used event-driven program?



OpenGL: Event-driven

- Program only responds to events
- Do nothing until event occurs
- Example Events:
 - Redraw “scene” in OpenGL window
 - mouse clicks,
 - keyboard stroke
- Programmer defines:
 - Events that program should respond to
 - Actions to be taken when event occurs
- Operating system (e.g. Windows):
 - Receives event, maintains event queue

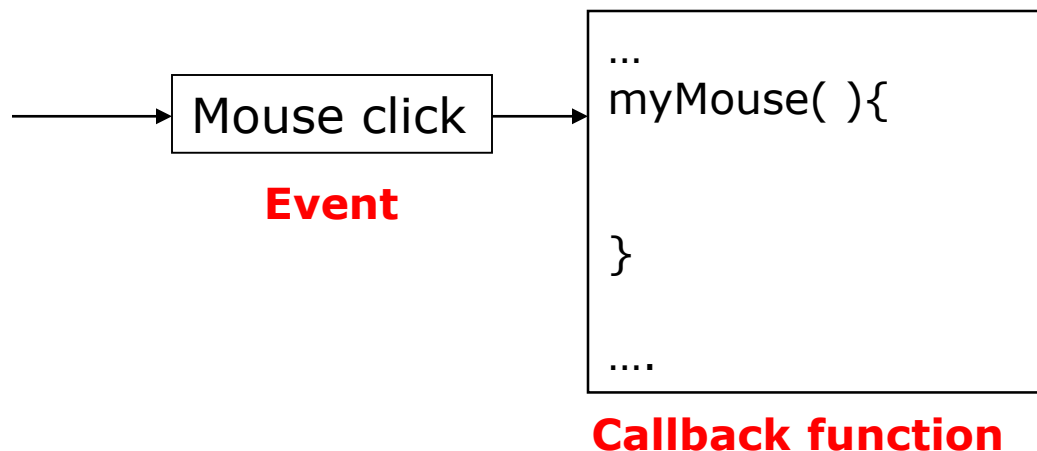


- Calls functions defined by programmer for specific event



OpenGL: Event-driven

- How in OpenGL?
 - Programmer declares, 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)** : Initial drawing put here
 - **glutReshapeFunc (myReshape)** : called when window is reshaped
 - **glutMouseFunc (myMouse)** : called when mouse button 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 occurs



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( );

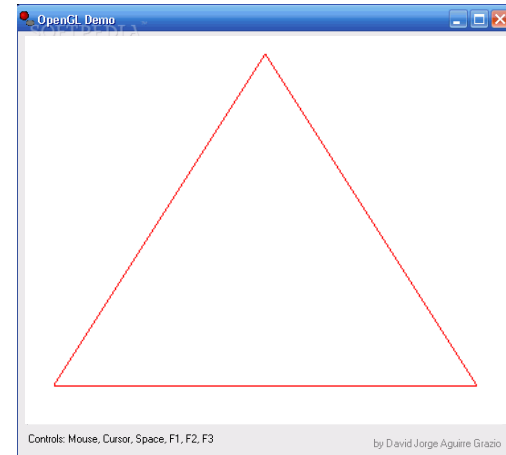
    // ... now register callback functions
    glutDisplayFunc(myDisplay);    ←--Next... how to draw in myDisplay
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

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



Example: Draw in function myDisplay

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



- **Rendering steps:**

1. Generate triangle corners (3 vertices), initially on CPU
2. Create GPU buffer for vertices
3. Move array of 3 vertices from CPU to GPU buffer
4. Draw 3 points from array on GPU using **glDrawArray**



Example: Retained Mode Graphics

- **Rendering steps:**

1. Generate triangle corners (3 vertices)
2. Create GPU buffer for vertices
3. Move array of 3 vertices from CPU to GPU buffer
4. Draw 3 points from array on GPU using `glDrawArray`

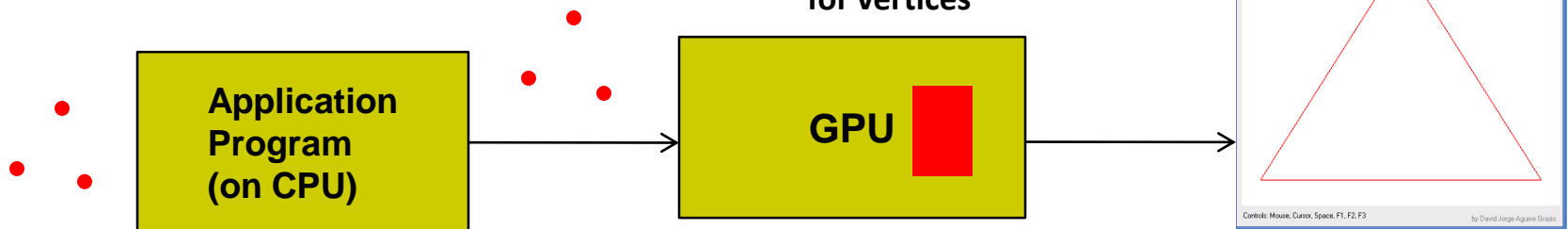
- **Simplified Execution model:**

1. Generate 3 triangle corners

3. Move array of 3 vertices from CPU to GPU buffer

2. Create GPU buffers for vertices

4. Draw points using `glDrawArrays`



Rendered vertices



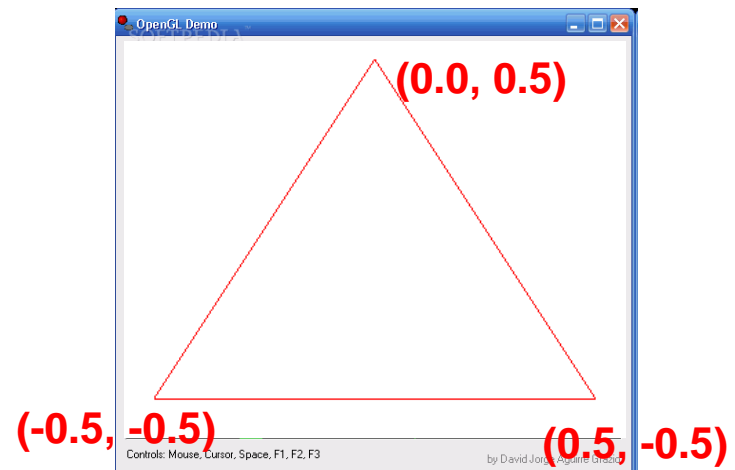
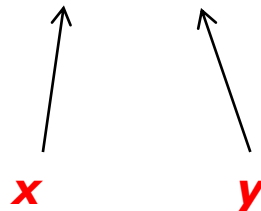
1. Generate triangle corners (3 vertices)

```
// declare array of 3 points
```

```
point2 points[3];
```

```
// generate 3 triangle vertices + store in points 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 );  
}
```





Declare some Types for Points, vectors

- Useful declarations (homegrown) in *header file* **vec.h** from book
 - **point2** for (x,y) locations
 - **vec3** for (x,y,z) vector coordinates
- Need to include header “vec.h”
- Example usage:

```
#include "vec.h"
```

```
vec3 vector1; ← Declares (x, y, z) coordinates of a vector
```

- **Note:** You will be given file Angel.h from book. It 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 );
}
```



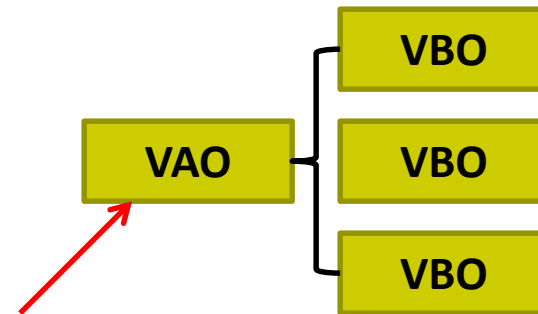
2. Create GPU Buffer for Vertices

- 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 VAO

```
GLuint vao;
```

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





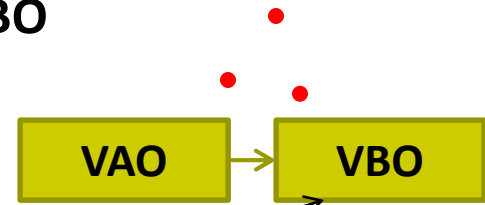
2. Create GPU Buffer for Vertices

- 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 VBO

Number of Buffer Objects to return



2. Make created VBO currently active one

glBindBuffer(GL_ARRAY_BUFFER, buffer);

Data is array of values



3. Move points GPU memory

3. Move points to VBO

Destination
GPU buffer

```
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:

2. Create GPU Buffer for Vertices

3. Move points GPU memory

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

    // Create GPU buffer object, move points to GPU
    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 );
}
```



4. Draw points (from VBO)

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

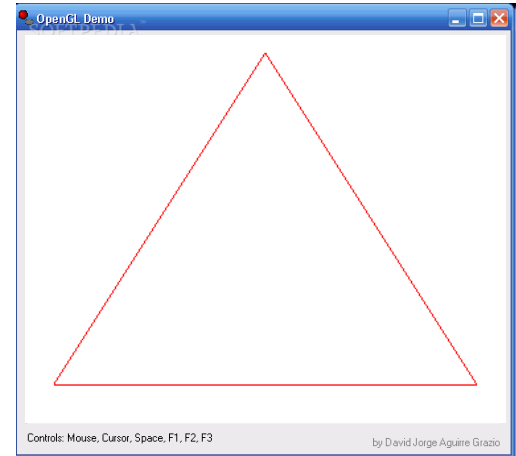
Render buffered data as points

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  
}
```





References

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