Graphics Programming

CS7GV6 2021/2022

Lecturer:
Prof. Carol O'Sullivan, carol.osullivan@tcd.ie

Demonstrator:
Bharat Vyas, vyasb@tcd.ie

Course Content: Blackboard

Unity 3D


testscene.unity – Bootcamp – PC and Mac Standalone
Pivot
Local
Layers
Layout
Game
Free Aspect
Maximize on Play
Gizmos
Stats
Inspector
Camera
Static
Tag
Untagged
Layer
Default
Transform
Position
X 17.74164
Y 3.618703
Z 17.97578
Rotation
X 8.649139
Y 330.9547
Z 0.0009765625
Scale
X 1
Y 1
Z 1
Camera
Clear Flags
Skybox
Background
Culling Mask
Mixed ...
Projection
Perspective
Field of View
36.84956
Clipping Planes
Near 0.3
Far 115
Normalized View Port Rect
X 0
Y 0
W 1
H 1
Depth
0
Rendering Path
Use Player Settings
Target Texture
None (Render Texture)
Flare Layer
GUILayer
Audio Listener
Animation
Color Correction Curves (Script)
Mode
Simple
Red
Green
Blue
Reset
Selective
Curve Resolution
256
Bloom And Flares (Script)
Mode
Advanced
Intensity
0.3
Blur iterations
3
Blur spread
1.188793
Hierarchy
Create
Camera
Soldier
SoldierFinalPos
EndTrigger
HudWeapons
MainMenuScreen
QualityManager
Radar
SargeManager
SoldierTargetHUD
SoundManager
SoundObjectManager
aquiris
beams_columns
Project
Create
CutsceneAnimations
Editor
Fonts
Robustik
Robustik-Bold
font material
font Texture
Robustik-Regular_9
Robustik-Regular_10
Robustik-Regular_12
Robustik-Regular_14
Robustik-Regular_14
Gizmos
Locomotion System
Inspector
Scene
Textured
RGB

Unreal Engine 4

Why OpenGL?

- No ready-to-use tools
- Graphics programming is hard
- Much, much longer to create a game
- 3D programming is very time consuming!

- Mastering OpenGL will lead you towards becoming a graphics programmer
- You will have a deeper understanding of how game engines have been built

Overview

- OpenGL background
- OpenGL conventions,
- GLUT Event loop, callback registration
- OpenGL primitives, OpenGL objects
- Shaders
- Vertex Buffer Objects
- Books, resources, recommended reading

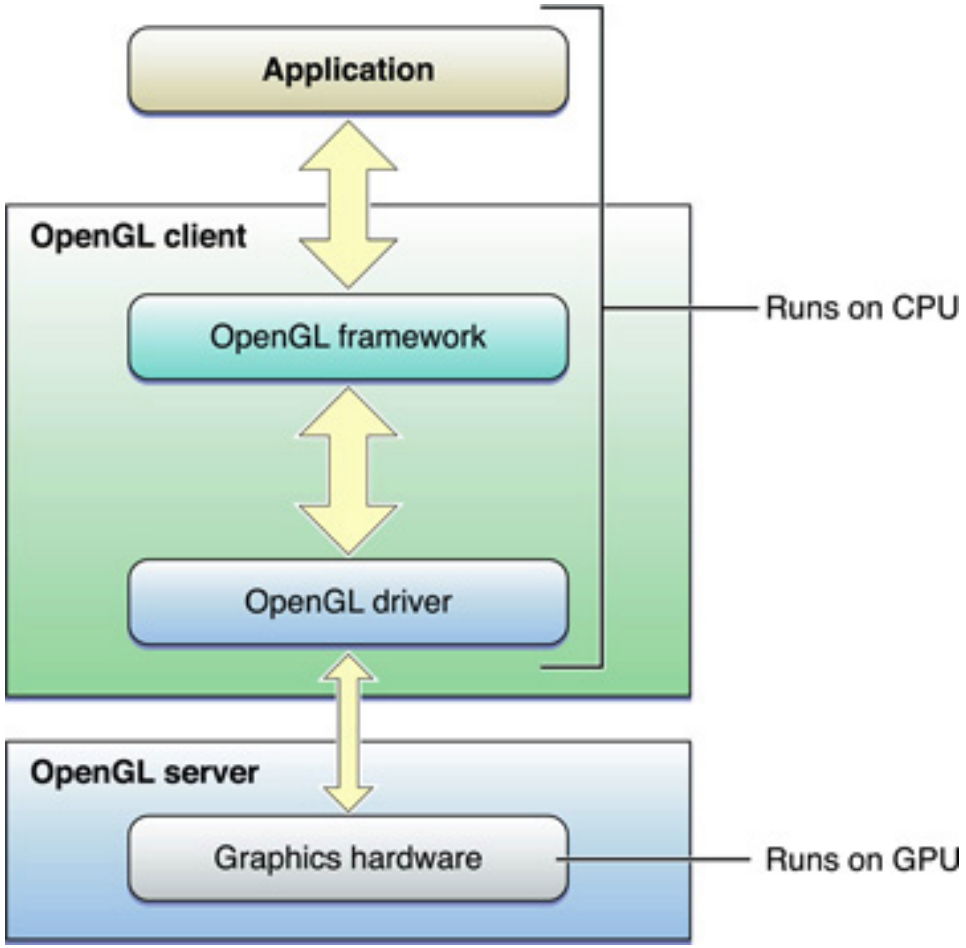
What is OpenGL?

- OpenGL = Open Graphics Library
- Application you can use to access and control the graphics subsystem of the device upon which it runs
- Developed at Silicon Graphics (SGI)
- It is *device independent*
- Cross Platform
  - (Win32, Mac OS X, Unix, Linux)
- Only does 3D Graphics. No Platform Specifics
  - (Windowing, Fonts, Input, GUI)

OpenGL

- OpenGL is a ***software library*** for accessing features in graphics hardware.
- About 500 distinct commands.
  - Not a single function relating to window, screen management, keyboard input, mouse input
- OpenGL uses a ***client-server*** model
  - Client is your application, server is OpenGL implementation on your graphics card/network graphics card
- Default language is **C/C++.**
- To the programmer OpenGL behaves like a **state machine**.
- The actual drawing operations are performed by the underlying *accelerated graphics hardware* (e.g. Nvidia, ATI, SGI etc).

Graphics API Architecture



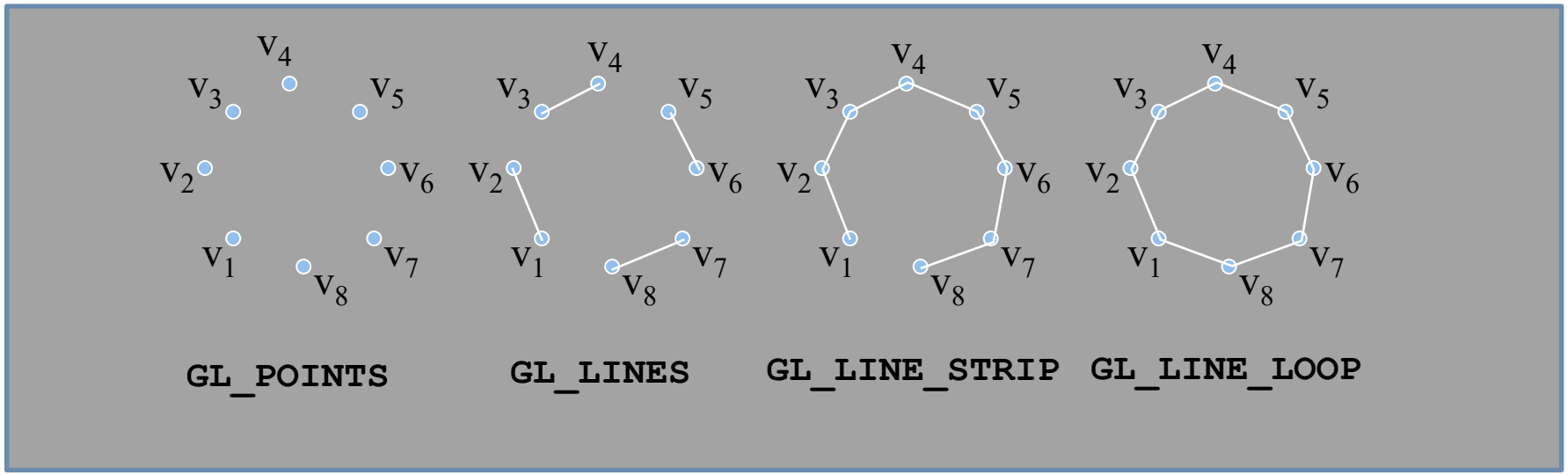
- Set-up & rendering loop run on CPU
- Copy mesh data to **buffers** in graphics hardware memory
- Write **shaders** to draw on the GPU
- CPU command queues drawing on GPU with *this* shader, and *that* mesh data
- CPU & GPU then run **asynchronously**

OpenGL Global State Machine

- Set various aspects of the state machine using the API
  - Colour, lighting, blending
- When rendering, everything drawn is affected by the current settings of the state machine
- Most **parameters are persistent**
  - Values remain unchanged until we explicitly change them through functions that alter the state
- Not uncommon to have **unexpected results** due to having one or more states set incorrectly

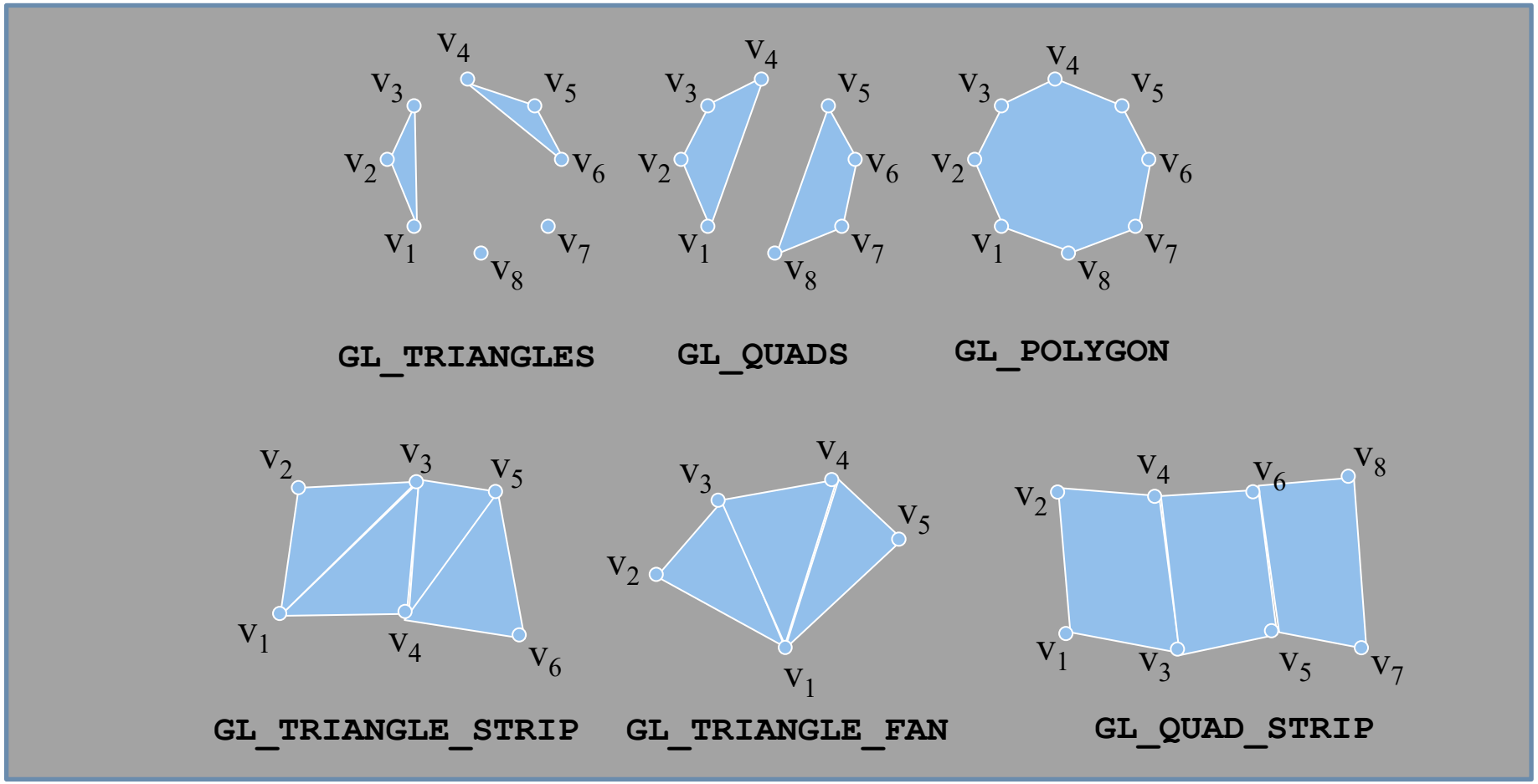
OpenGL Primitives

- All geometric objects in OpenGL are created from a set of basic *primitives*.
- Certain primitives are provided to allow optimisation of geometry for improved rendering speed.
- Line based primitives:



OpenGL® Primitives

- Polygon primitives



OpenGL Conventions

- Conventions:
  - all function names begin with **`gl`**, or **`glut`**
    - **`glBegin(…)`**
    - **`glutInitDisplayMode(…)`**
  - constants begin with **`GL_`**, **`GLU_`**, or **`GLUT_`**
    - **`GL_POLYGON`**
  - Function names can encode parameter types, e.g. **`glVertex*:`**
    - **`glVertex2i(1, 3)`**
    - **`glVertex3f(1.0, 3.0, 2.5)`**
    - **`glVertex4fv(array_of_4_floats)`**

http://www.opengl.org/sdk/docs/man/

The Drawing Process

- `ClearTheScreen();`
- `DrawTheScene();`
- `CompleteDrawing();`
- `SwapBuffers();`

- In animation there are usually two buffers. Drawing usually occurs on the background buffer.
- When it is complete, it is brought to the front (swapped). This gives a smooth animation without the viewer seeing the actual drawing taking place. Only the final image is viewed.

- The technique to swap the buffers will depend on which windowing library you are using with OpenGL.

Clearing the Window

- `glClearColor(0.0, 0.0, 0.0, 0.0);`
- `glClear(GL_COLOR_BUFFER_BIT);`

- Typically you will clear the color and depth buffers.

- `glClearColor(0.0, 0.0, 0.0, 0.0);`
- `glClearDepth(0.0);`
- `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`

- You can also clear the accumulation and stencil buffers.
  - `GL_ACCUM_BUFFER_BIT and GL_STENCIL_BUFFER_BIT`

Specifying a Colour

- It is possible to represent almost any colour by adding red, green and blue
- Colour is specified in (R,G,B,A) form [Red, Green, Blue, Alpha], with each value being in the range of 0.0 to 1.0.
  - 0.0 means "all the way off"
  - 1.0 means "all the way on"
- Examples:
  - `(red, green, blue, alpha);`

```
- (0.0, 0.0, 0.0); /* Black */
- (1.0, 0.0, 0.0); /* Red */
- (0.0, 1.0, 0.0); /* Green */
- (1.0, 1.0, 0.0); /* Yellow */
- (1.0, 0.0, 1.0); /* Magenta */
- (1.0, 1.0, 1.0); /* White */
```

Complete Drawing the Scene

- Need to tell OpenGL you have finished drawing your scene.
- **`glFinish();`**

  or

- **`glFlush();`**
- For more information see Chapter of the Red Book:
  - http://fly.srk.fer.hr/~unreal/theredbook/chapter02.html

OpenGL `GLUT` Overview

- Initialise GLUT and create a window
- GLUT enters event processing loop and gains control of the application
- GLUT waits for an event to occur & then checks for a function to process it
- Tell GLUT which functions it must call for each event

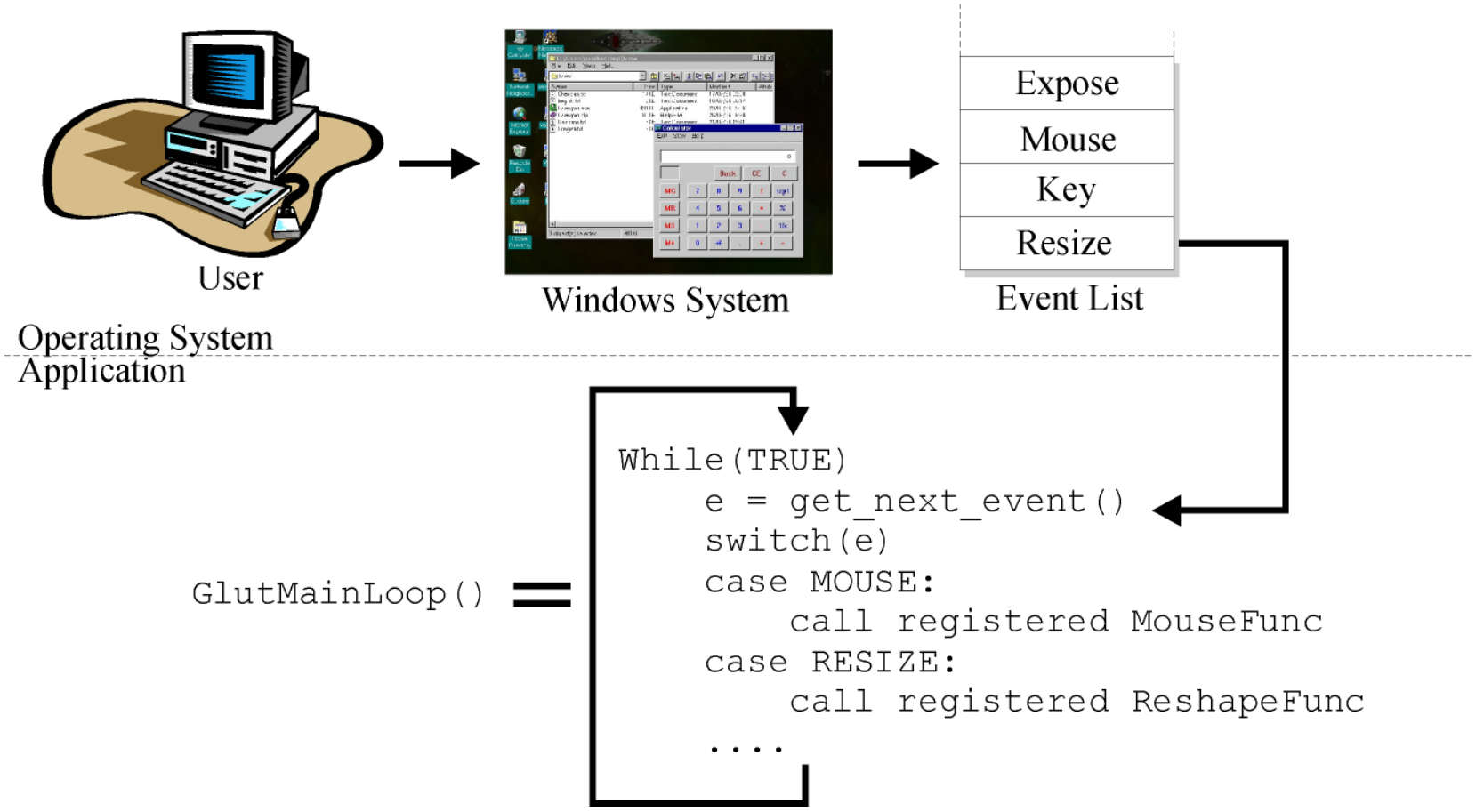
OpenGL `GLUT` Event Loop

- Interaction with the user is handled through an ***event loop***.
- Application registers ***handlers*** (or *callbacks*) to be associated with particular events:
  - mouse button, mouse motion, timer, resize, redraw
- `GLUT` provides a wrapper on the X-Windows or Win32 core event loop.
- X-Windows or Win32 manages event creation and passing, `GLUT` uses them to catch events and then invokes the appropriate callback.
- `GLUT` is more general than X or Win32 etc.

  ⇒ more portable: user interface code need not be changed.

  ⇒ less powerful: implements a common subset

OpenGL GLUT Event Loop



OpenGL `GLUT` Event Loop

- To add handlers for events we call a callback registering function, e.g:

```
void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));
```

- Takes a function (the required callback) as a parameter.
- Handlers must conform to the specification defined.
- Example:

```
void key_handler(unsigned char key, int x, int y);

glutKeyboardFunc(key_handler);
```

- In this case, **`key`** is the ascii code of the key hit and **`(x,y)`** is the mouse position within the window when the key was hit.
- The callback function is *automatically* called when a key is hit.

```
//----------------------------------------------------------------------
//
// main
//

int
main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(512, 512);
    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_CORE_PROFILE);
    glutCreateWindow(argv[0]);

    if (glewInit()) {
        cerr << "Unable to initialize GLEW ...  exiting" << endl;
        exit(EXIT_FAILURE);
    }

    init();

    glutDisplayFunc(display);

    glutMainLoop();
}
```

Creates a Window using GLUT

Call init()

Event Loop

```
//----------------------------------------------------------------------
//
// init
//

void
init(void)
{
    glGenVertexArrays(NumVAOs, VAOs);
    glBindVertexArray(VAOs[Triangles]);

    GLfloat  vertices[NumVertices][2] = {
        { -0.90, -0.90 },  // Triangle 1
        {  0.85, -0.90 },
        { -0.90,  0.85 },
        {  0.90, -0.85 },  // Triangle 2
        {  0.90,  0.90 },
        { -0.85,  0.90 }
    };

    glGenBuffers(NumBuffers, Buffers);
    glBindBuffer(GL_ARRAY_BUFFER, Buffers[ArrayBuffer]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
                 vertices, GL_STATIC_DRAW);

    ShaderInfo  shaders[] = {
        { GL_VERTEX_SHADER, "triangles.vert" },
        { GL_FRAGMENT_SHADER, "triangles.frag" },
        { GL_NONE, NULL }
    };

    GLuint program = LoadShaders(shaders);
    glUseProgram(program);

    glVertexAttribPointer(vPosition, 2, GL_FLOAT,
                          GL_FALSE, 0, BUFFER_OFFSET(0));
    glEnableVertexAttribArray(vPosition);
}
```

Set up your object's initial position

Specify Shaders

```
//----------------------------------------------------------
//
// display
//

void
display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBindVertexArray(VAOs[Triangles]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);

    glFlush();
}
```

Does the actual Drawing on the Screen

Pick current vertex array

Request that image is presented on screen