OpenGL

where it fits

what it contains

how you work with it

OpenGL

***The* cross-platform graphics library!**

Open = Open specification

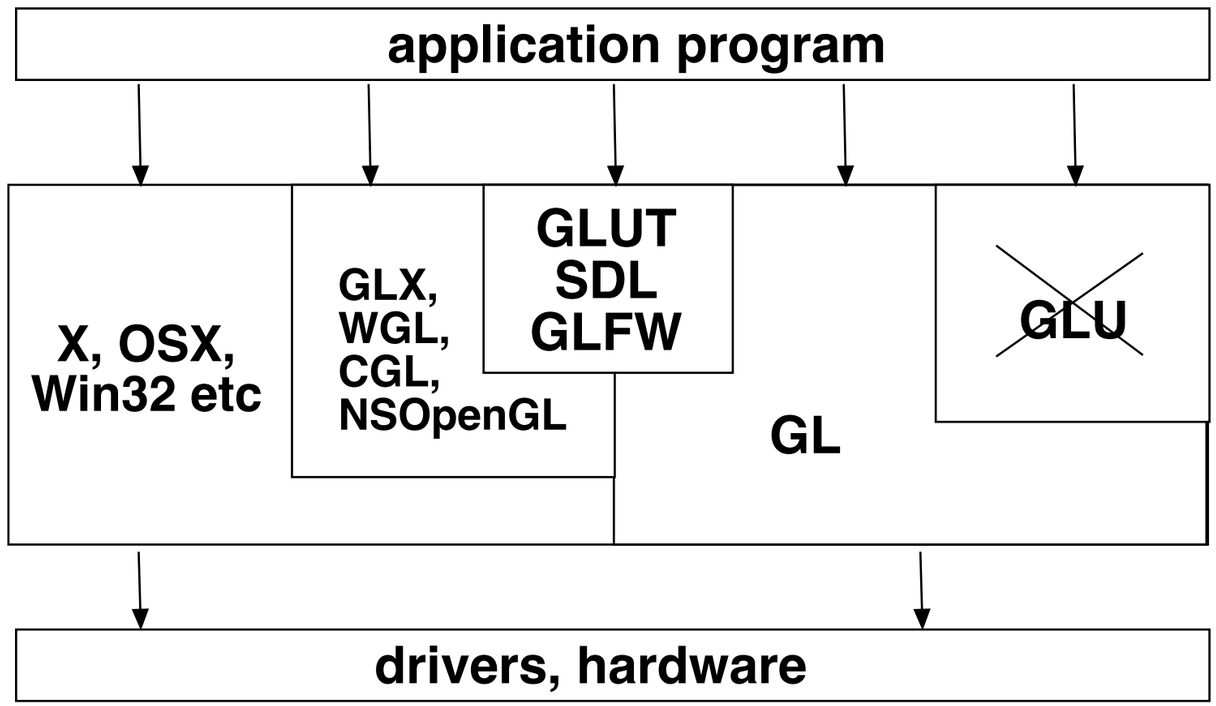
Runs everywhere - Linux, Mac, Windows...

Any language

Three branches:

OpenGL - ordinary computers
OpenGL ES - phones and tablets
WebGL - web browsers

OpenGL and Related APIs



OpenGL parts:

GL = Graphics Library (core lib)

GLU = GL Utilities (no longer supported)

GLX, WGL, CGL, NSOpenGL = System dependent libraries

GLUT = GL Utility Toolkit (optional)

FreeGLUT, MicroGlut

Also note: SDL (Simple Directmedia Layer)

GLFW (similar to GLUT)

OpenGL versions

OpenGL 1-2: Old OpenGL. Avoid!

Big leap here!

OpenGL 3.2 and up: Modern OpenGL!
Latest version 4.5.

Even hotter (but harder): Vulkan (released 2016)

Simple OpenGL example

A single triangle

- upload to the GPU once
- draw any time you want

but also

- upload shader to specify transformations and colors

Vital concepts

- **VAO, VBO**
- **Vertex shader**
- **Fragment shader**

VAO and VBO

VBO = Vertex Buffer Object

A reference to an array of vertices on the GPU

VAO = Vertex Array Object

A reference to a set of buffers

What is a shader?

Small program kernel that runs on the GPU!

Gives you great freedom to specify certain operations.

Run in parallel on multiple kernels (hundreds!) in the GPU! Extremely efficient!

Vertex shader

Specifies transformations on each vertex

Translations, rotations...

Short program with data sent from the main program

In the example: Pass-through

Fragment shader

Specifies color of each pixel

Short program with data sent from the main program or vertex shader.

In the example: Set-to-white

Sample main program (in C)

```
void main( int argc, char** argv )
                    {
        glutInit(argc, argv));
glutCreateWindow("GL3 white triangle example");
        glutDisplayFunc( display );
                    init();
                glutMainLoop();
                    }
```

Looks different for different ”wrapper” libraries

Sample shaders

Minimal, doing pretty much nothing

Vertex shader:
Specifying positioning
(pass-through)

```
#version 150

in  vec3 in_Position;

void main(void)
{
 gl_Position = vec4(in_Position, 1.0);
}
```

Fragment shader:
Specifying pixel colors
(set-to-white)

```
#version 150

out vec4 out_Color;

void main(void)
{
 out_Color = vec4(1.0);
}
```

Name conventions

All calls prefixed by gl

All constants prefixed by GL

Some calls suffixed by number, f,d,v

Example

glVertexAttrib2fv(index, v)

Number of components

2 - (x,y)

3 - (x,y,z)

4 - (x,y,z,w)

Data type

b - byte
ub - unsigned byte
s - short integer
us - unsigned short
i - int (integer)
ui - unsigned int
f - float
d - double

Vector

If present: arguments are array

If omitted: arguments as separate scalars:

glUniform1f(loc, time)

OpenGL Initialization
Set up whatever state you're going to use

```
void init( void )
{
glClearColor( 0.0, 0.0, 0.0, 1.0 );
glClearDepth( 1.0 );
glEnable( GL_DEPTH_TEST );
glEnable( GL_CULL_FACE );
}
```

OpenGL type names

Standard types redefined prefixed GL
(for compatibility reasons)

From gl.h:

```
typedef unsigned long GLenum;
typedef unsigned char GLboolean;
typedef unsigned long GLbitfield;
typedef signed char GLbyte;
typedef short GLshort;
typedef long GLint;
typedef long GLsizei;
typedef unsigned char GLubyte;
typedef unsigned short GLushort;
typedef unsigned long GLuint;
typedef float GLfloat;
typedef float GLclampf;
typedef double GLdouble;
typedef double GLclampd;
typedef void GLvoid;
```

GLUT (FreeGLUT, MicroGLUT) Callback Functions

Routines for GLUT to call when something happens

- window resize or redraw
- user input
- animation

"Register" callbacks with GLUT

```
glutDisplayFunc( display );
   glutIdleFunc( idle );
glutKeyboardFunc( keyboard );
```

Rendering Callback. Do all of our drawing here!

```
glutDisplayFunc( display );
    void display( void )
                {
    void display(void)
                {
    // clear the screen
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Select VAO and draw
        glBindVertexArray(vertexArrayObjID);
glDrawBuffer(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, NULL);

        glutSwapBuffers();
                }
```

Timer Callbacks
Use for animation and continuous update

```
glutTimerFunc( time, idle );

void timerFunc( ignoredValue )
{
t +=dt;
glutTimerFunc(20, timerFunc, 0);
glutPostRedisplay();
}
```

Simple Example

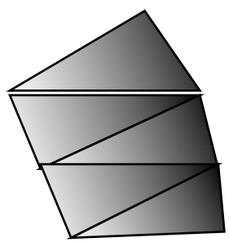
A shape can be hard-coded in an array

```
GLfloat vertices[] = { -0.5f,-0.5f,0.0f,
                       -0.5f,0.5f,0.0f,
                       0.5f,-0.5f,0.0f };
GLfloat colors[] = { 1.0f, 0.0f, 0.0f,
                     0.0f, 1.0f, 0.0f,
                     0.0f, 0.0f, 1.0f };
```

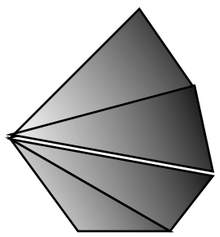
But usually you read data from files.

OpenGL Geometric Primitives

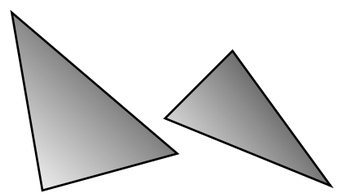
All geometric primitives are specified by vertices:



GL_TRIANGLE_STRIP

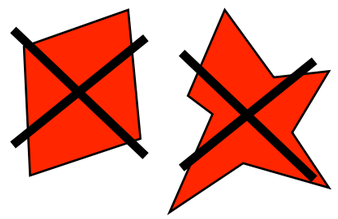


GL_TRIANGLE_FAN



GL_TRIANGLES

Why not QUAD or POLYGON?



Uploading to the GPU is a bit hairy.

```
// Allocate and activate Vertex Array Object
glGenVertexArrays(1, &vertexArrayObjID);
glBindVertexArray(vertexArrayObjID);

// VBO for vertex data
glGenBuffers(1, &vertexBufferObjID);
glBindBuffer(GL_ARRAY_BUFFER, vertexBufferObjID);

// Data to VBO
glBufferData(GL_ARRAY_BUFFER, 9*sizeof(GLfloat), vertices, GL_STATIC_DRAW);
glVertexAttribPointer(glGetAttribLocation(program, "in_Position"), 3,
                      GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(glGetAttribLocation(program, "in_Position"));
```