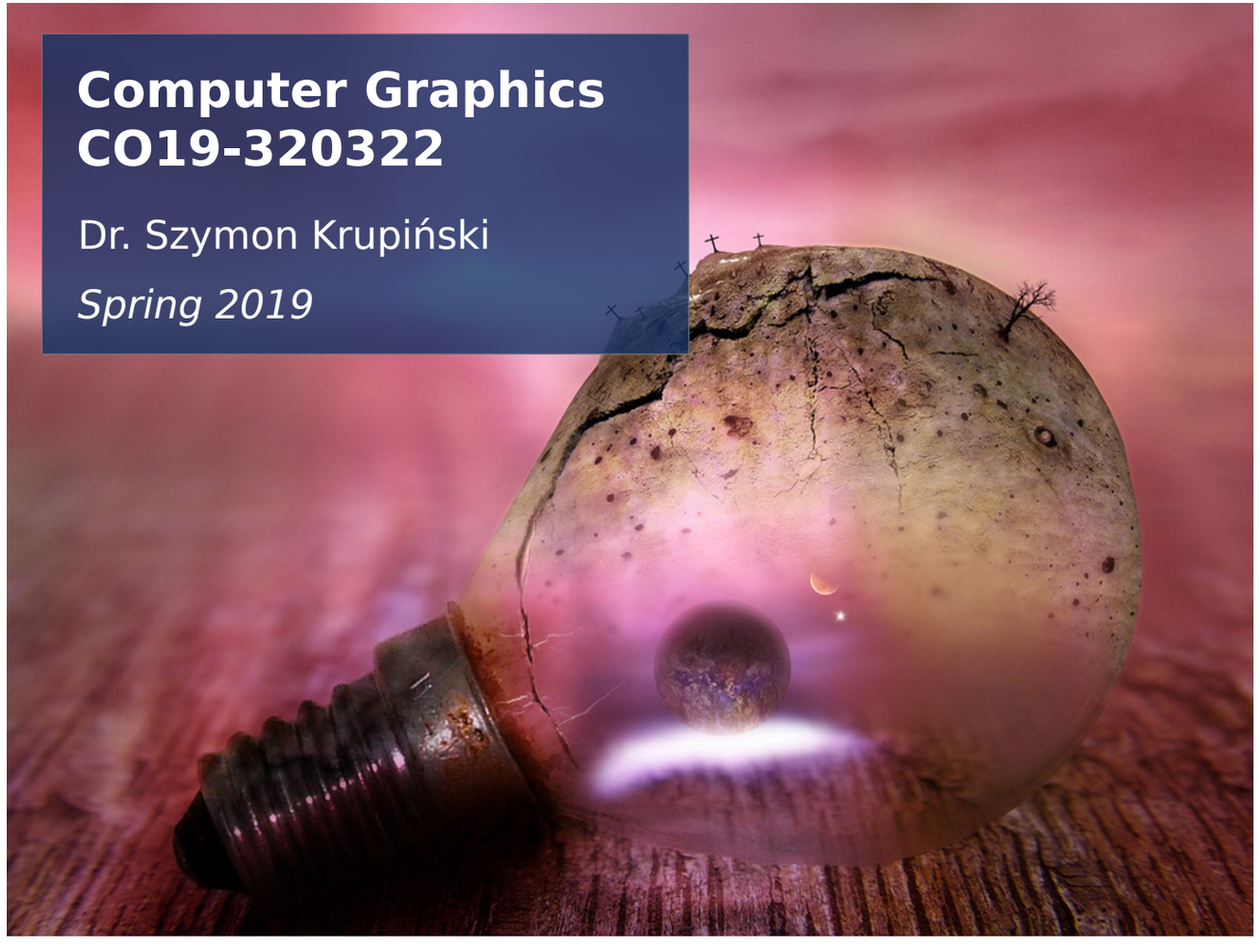
Lecture 7: OpenGL Primitives

JACOBS UNIVERSITY



Computer Graphics
CO19-320322

Dr. Szymon Krupiński

Spring 2019

In the last episode...

- More detailed look into the OpenGL API
  - With many GLUT/GLU references
- The backbone of every OpenGL program
  - intialisation
  - callbacks (including the display)
  - main loop
- “old style” pipeline (matrix stack management)

Focus on the display part

- From last time:
  (myfirstcode.c)

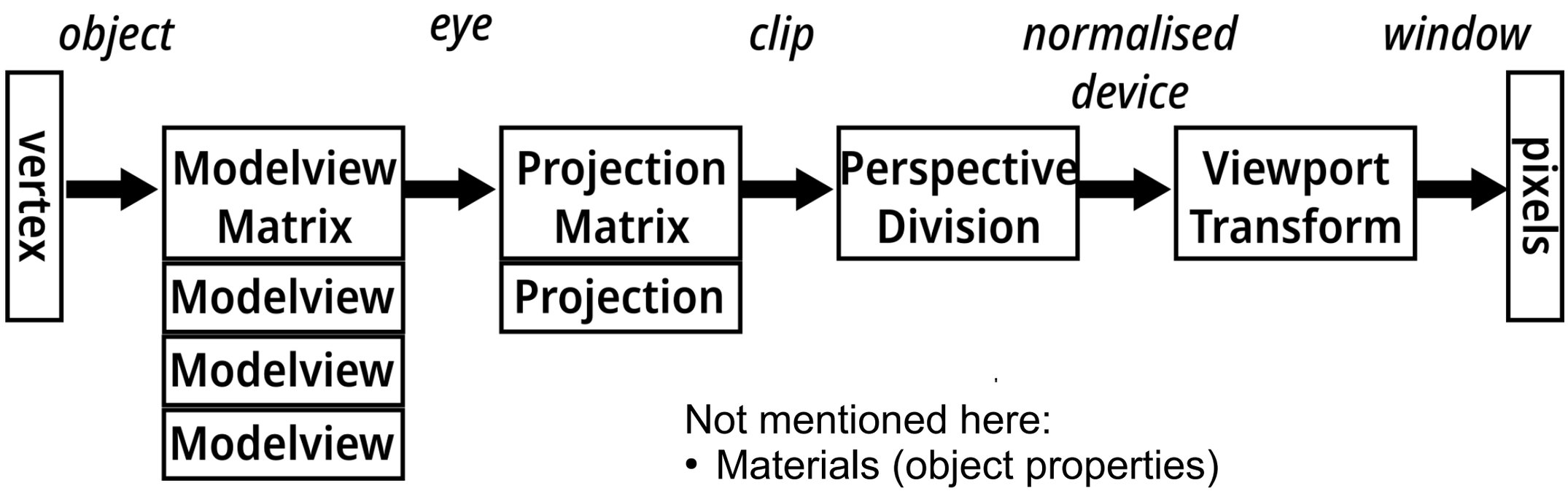
```
void display(void) {
    // we will be working on the model view matrix stack
    glMatrixMode(GL_MODELVIEW);
    // clear the drawing buffer.
    glClear(GL_COLOR_BUFFER_BIT);
    // clear the identity matrix.
    glLoadIdentity();
    // traslate the drawing plane by z = -4.0
    glTranslatef(0.0,0.0,-10.0);
    // changing in transformation matrix.
    // rotation about X axis
    glRotatef(xRotated,1.0,0.0,0.0);
    // rotation about Y axis
    glRotatef(yRotated,0.0,1.0,0.0);
    // rotation about Z axis
    glRotatef(zRotated,0.0,0.0,1.0);
    // scaling transfomation
    glScalef(1.0,1.0,1.0);
```

```
// set lights
float lightZeroPosition[] = {10.0, 4.0, 10.0, 1.0};
float lightZeroColor[] = {0.8, 1.0, 0.8, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, lightZeroPosition);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
glEnable(GL_LIGHTING);
float ambColor[] =  {1.0, 0.0, 0.0, 1.0};
float difColor[] =  {1.0, 0.0, 0.0, 1.0};
glMaterialfv(GL_FRONT, GL_AMBIENT, ambColor);
glMaterialfv(GL_FRONT, GL_DIFFUSE, difColor);
```

```
    // built-in (glut library) function, draws a sphere.
    glutSolidSphere(radius,20,20);
    // Swap and flush buffers to screen
    glutSwapBuffers();
}
```

Light and material-related function calls were not covered in class yet.

What do we need to set up in order to display?



Not mentioned here:

- Materials (object properties)
- Shade model
- Polygon rendering mode
- Polygon culling
- Clipping

} OpenGL options

Primitives

- OpenGL geometry is defined through special structures called primitives
- They are collections of vertices
- There exist a world beyond triangles
  - ...but we will mostly use them
  - working with polygons is deprecated
- Let’s talk about specific conventions

Data types

- You will often see strange looking basic types
- They are there to assure compatibility between systems
- from GL/gl.h:

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

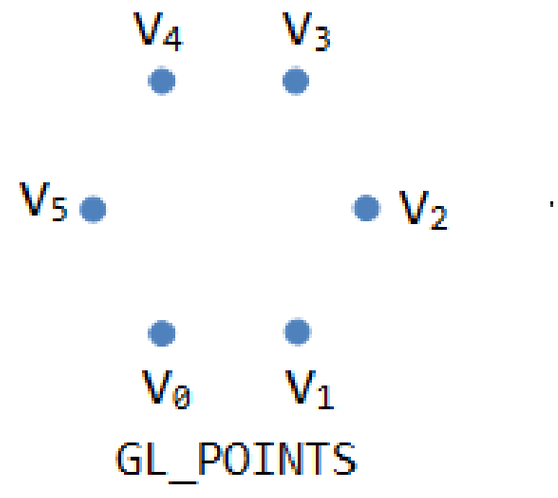
Primitives

- Matrix convention
  - All affine operations are matrix mutliplications
  - All matrices are stored column-major in OpenGL
  - Matrices are always post-multiplied
    - Hence, the product of matrix and vector is $\mathbf{M}\vec{v}$

$$\mathbf{M} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

Primitives

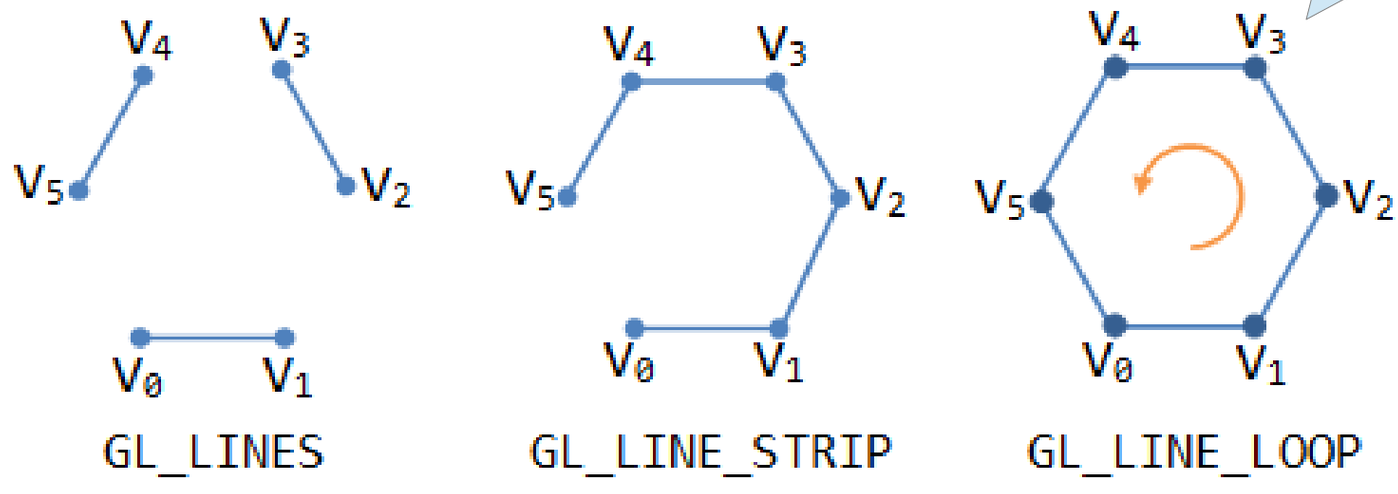
- The very basic: GL_POINTS, a collection of unconnected vertices



Primitives

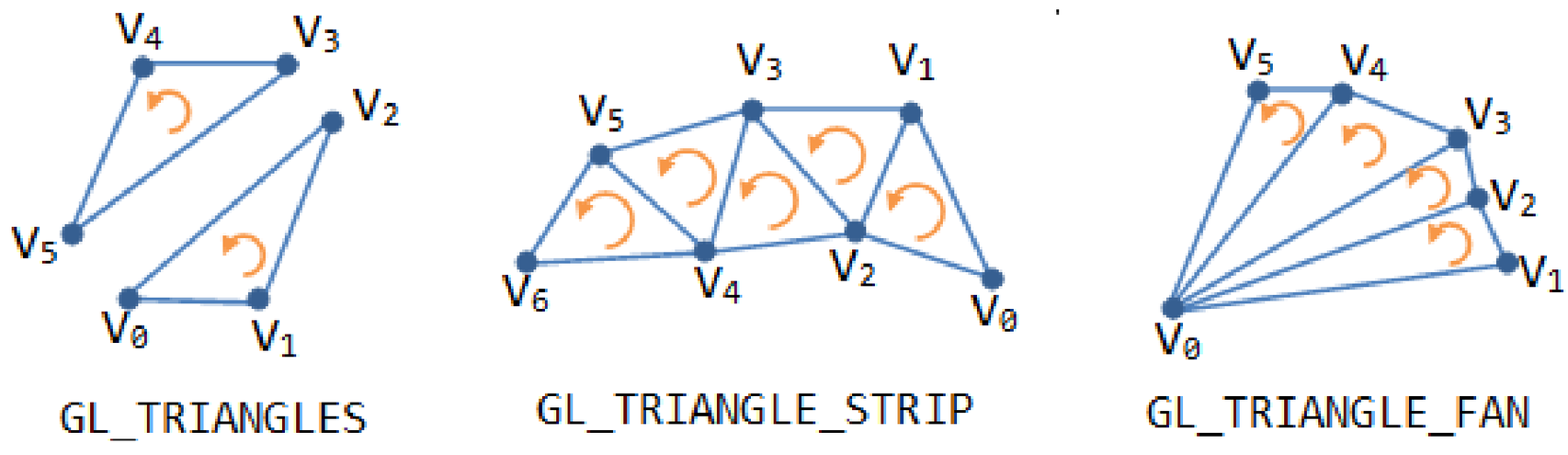
- Edges automatically inserted between (some) vertices:
  - GL_LINES: between every consecutive pair
  - GL_LINE_STRIP: pairwise between all consecutive points
  - GL_LINE_LOOP: the same but with last element connected to the first

What is this arrow? The order in which OpenGL connects vertices decides which side of the figure will be the “front”. It indicates the Winding Order.



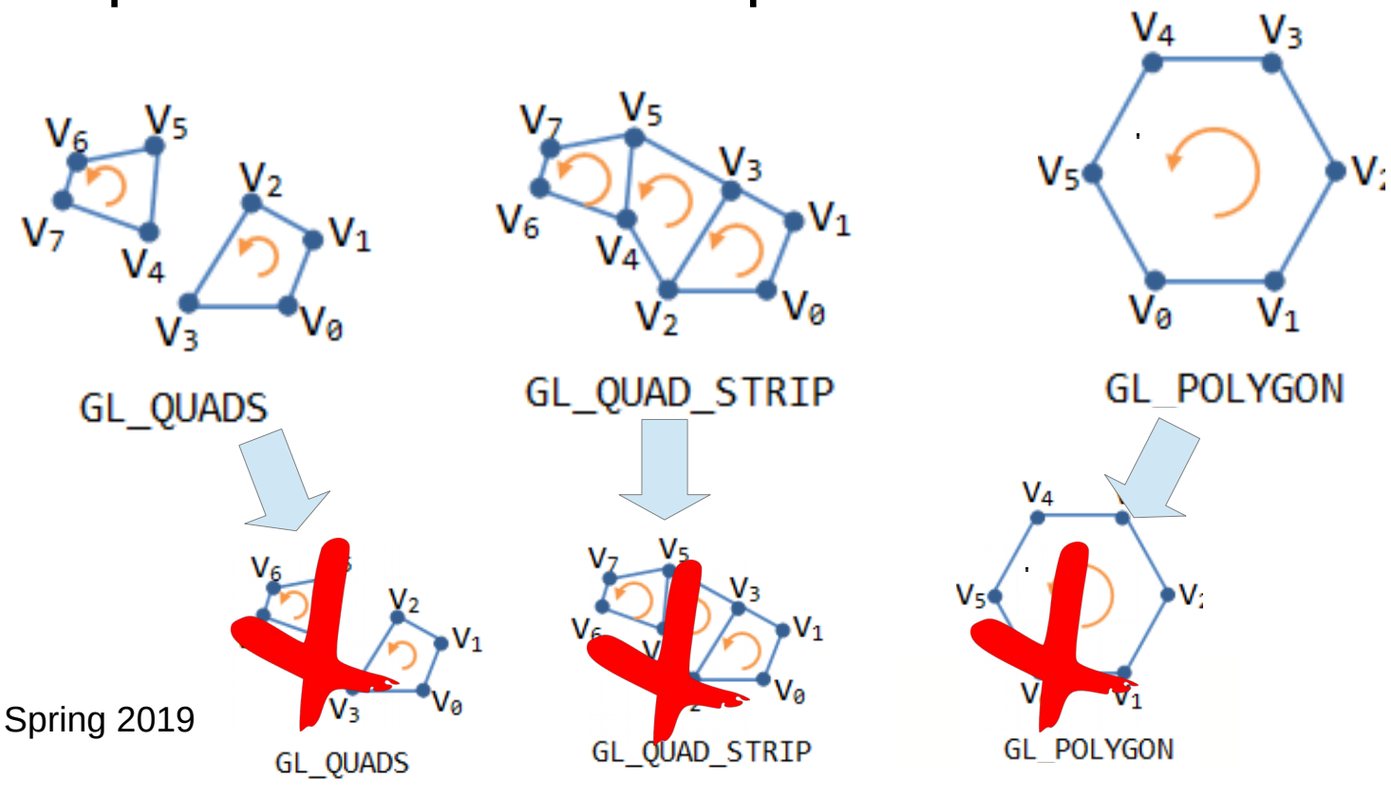
Primitives

- Triangles formed between certain vertices:
  - GL_TRIANGLES: between every consecutive set of three vertices
  - GL_TRIANGLES_STRIP: each consecutive vertex creates a triangle with two preceding ones
  - GL_TRIANGLE_FAN: the first vertex serves as a common point of all triangles, each new vertex creates a triangle with it and the previous one



Primitives

- GL_QUADS and GL_POLYGONS: four- and multiple vertex shapes (no mechanism to guarantee that they are planar)
- Attention: using shapes like quads and polygons is discourages, as they are deprecated in newer OpenGL versions



Primitives

- All primitives need vertices in their definition but it's the type of primitive which decides how/if these vertices will be connected

```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(4.0, 3.0);
    glVertex2f(6.0, 1.5);
    glVertex2f(4.0, 0.0);
glEnd();
```

```
glBegin(GL_POINTS);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(4.0, 3.0);
    glVertex2f(6.0, 1.5);
    glVertex2f(4.0, 0.0);
glEnd();
```

GL_POLYGON

GL_POINTS

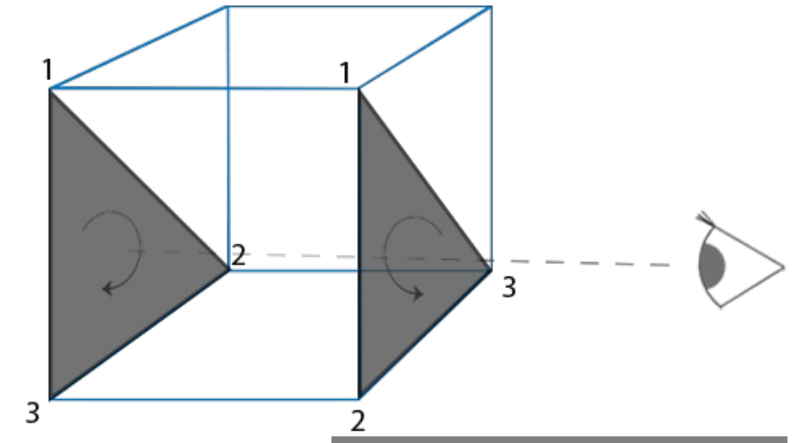
Primitives

- This way of defining primitives gives us a couple of interesting possibilities:

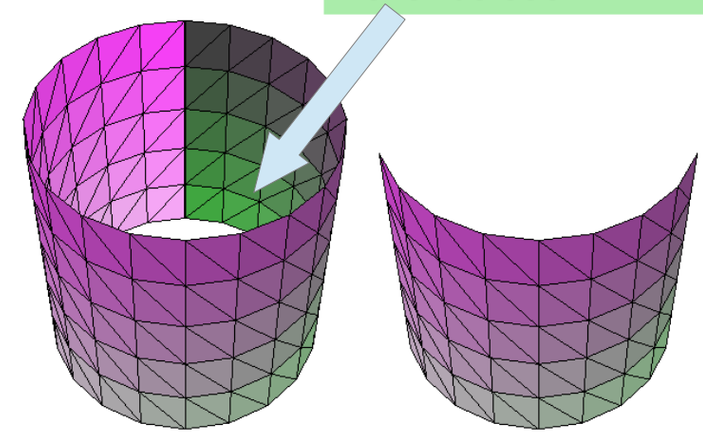
```
GLfloat red, green, blue;
GLfloat coords[3];
glColor3f(red, green, blue);
glBegin( GL_TRIANGLE_STRIP );
    for ( i=0; I < nVerts; i++ ) {
        glVertex3fv( coords );
    }
glEnd();
```

Primitives

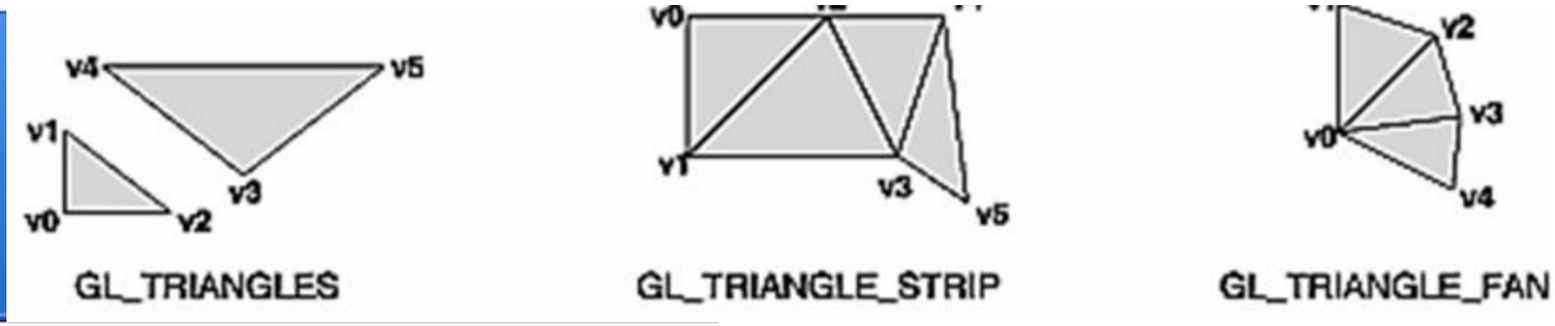
- The Winding Order determines the facing of a triangle. The triangle's facing can be used to cull faces based on whether it is the front or back face
- The general order can be changed by setting the GL_CW or GL_CCW flag during initialisation
- Facing only matters for triangle primitive rasterization. All non-triangle primitive types are considered to rasterize the front face, and face culling only works on triangles



Am I seeing front or back of the triangles..? In case of a closed shape (coherently) constructed of triangles, the back (= inside) might never be seen.



Primitives

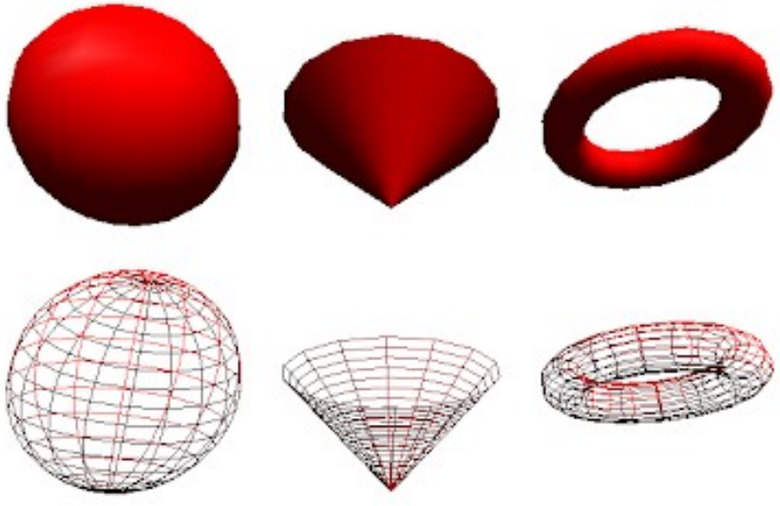


- Points can be interconnected into lines
- The triangles in the triangle arrangements are interconnected
- Let's look at a couple of possiblilities:

```
GL_TRIANGLES:

Indices:     0 1 2 3 4 5 ...
Triangles:  {0 1 2}
                 {3 4 5}

GL_TRIANGLE_STRIP:

Indices:     0 1 2 3 4 5 ...
Triangles:  {0 1 2}
              {1 2 3}  drawing order is (2 1 3) to maintain proper winding
                {2 3 4}
                  {3 4 5}  drawing order is (4 3 5) to maintain proper winding

GL_TRIANGLE_FAN:

Indices:     0 1 2 3 4 5 ...
Triangles:  {0 1 2}
            {0} {2 3}
            {0}   {3 4}
            {0}     {4 5}
```

Further OpenGL

- Shape drawing functions GLUT
  - glutWireTorus(innerRadius,outerRadius, nsides, rings);
  - glutWireCube(side_len);
  - glutWireSphere(radius, slices, stacks);
  - glutSolidTorus(innerRadius,outerRadius, nsides, rings);
  - ...glutSolidTeapot();
  - …

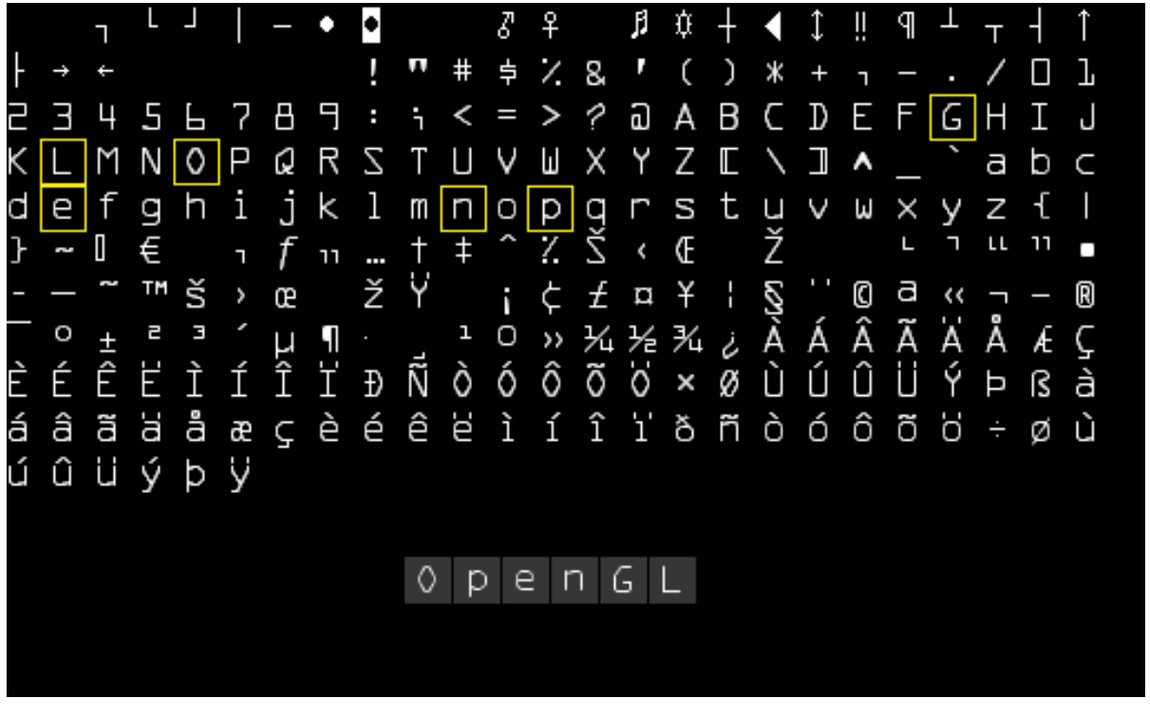


- All relevant documentation:

https://www.opengl.org/resources/libraries/glut/spec3/spec3.html

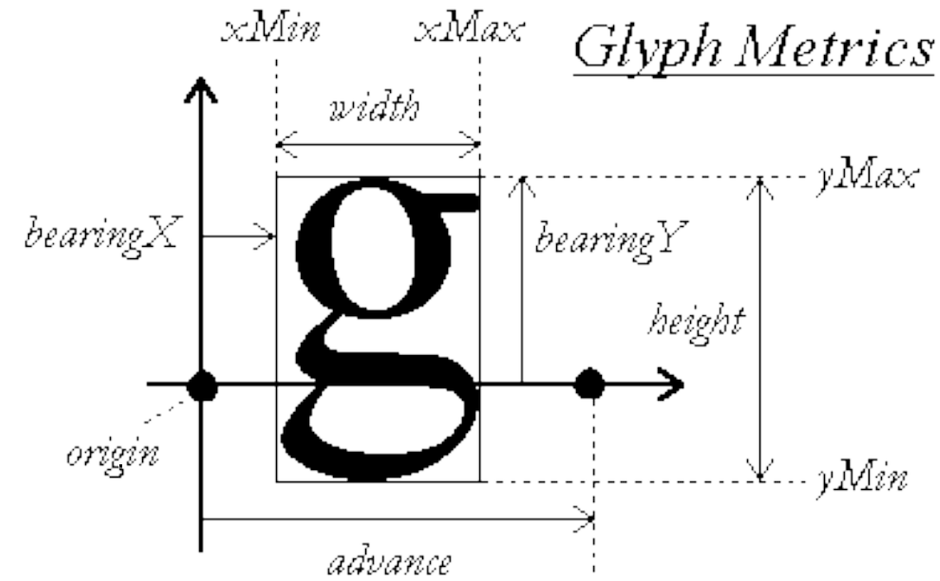
Text

- Text Rendering in OpenGL?
  - None…
- Old fashion way of solving this:
  - Bitmap fonts
  - All letter prototypes collected in one file
  - Rendering quads and overlaying them with texture from the right spot in the font file
  - `Monospace font`



Text

- How modern fonts work?
- FreeType – characters defined by mathematical formulas (vector drawing)
- They can “slide” on the baseline and be fitted just with enough space
  - Normal font



Exercises

- http://cs.lmu.edu/~ray/notes/openglexamples/
- GLUT interactive drawing
- (uses C++ and .cpp)

Easier geometry work?

- Object loading function!
- OBJ (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies
- Can easily be produced in blender!
- Example at:

http://kixor.net/dev/objloader/

Thank you!

- Questions?