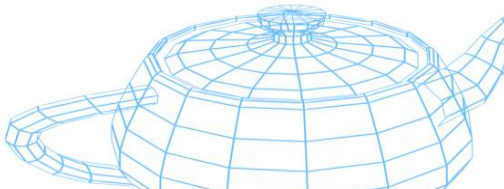
SUPSI

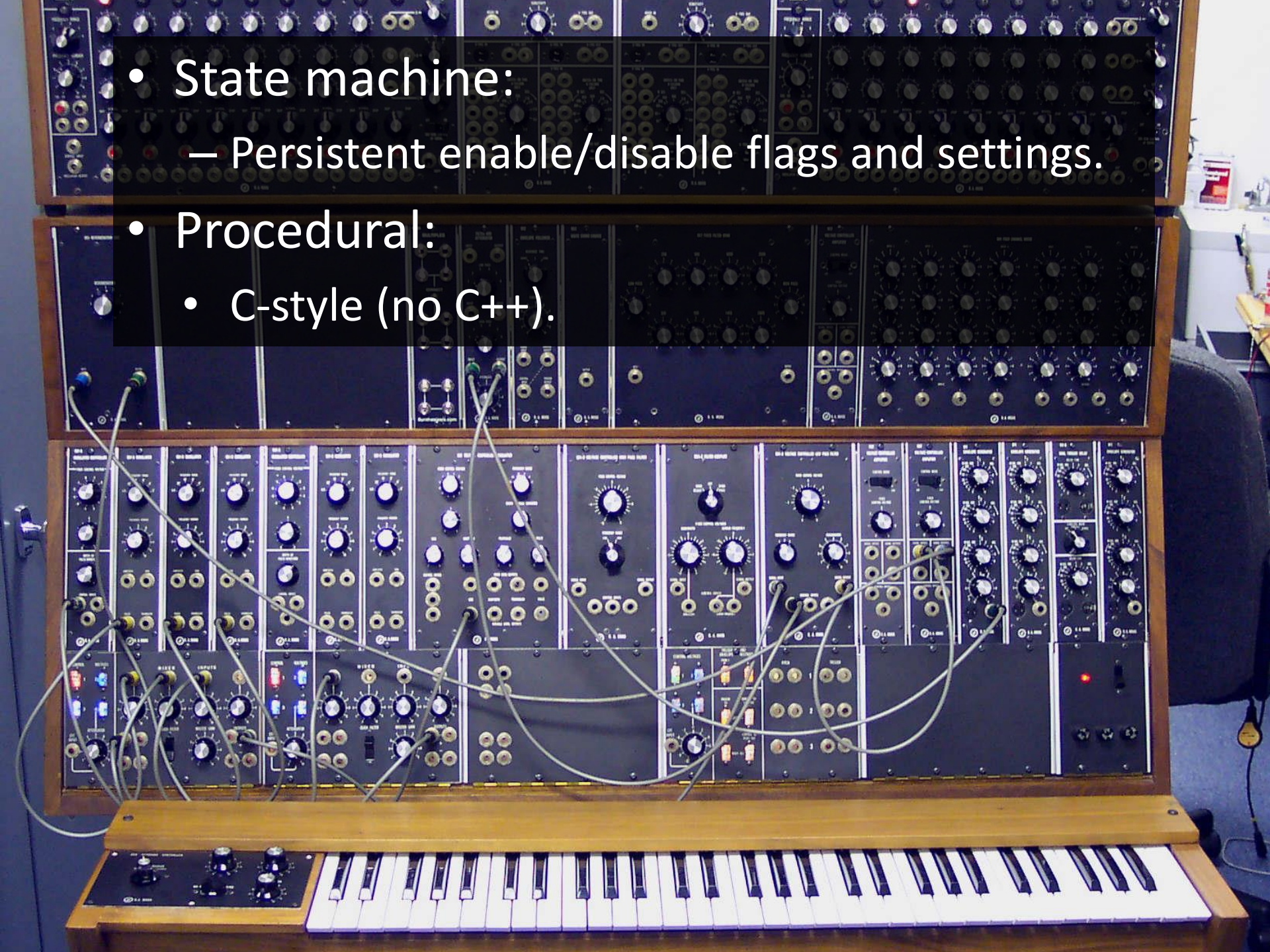
Computer Graphics

OpenGL (2): the first triangle

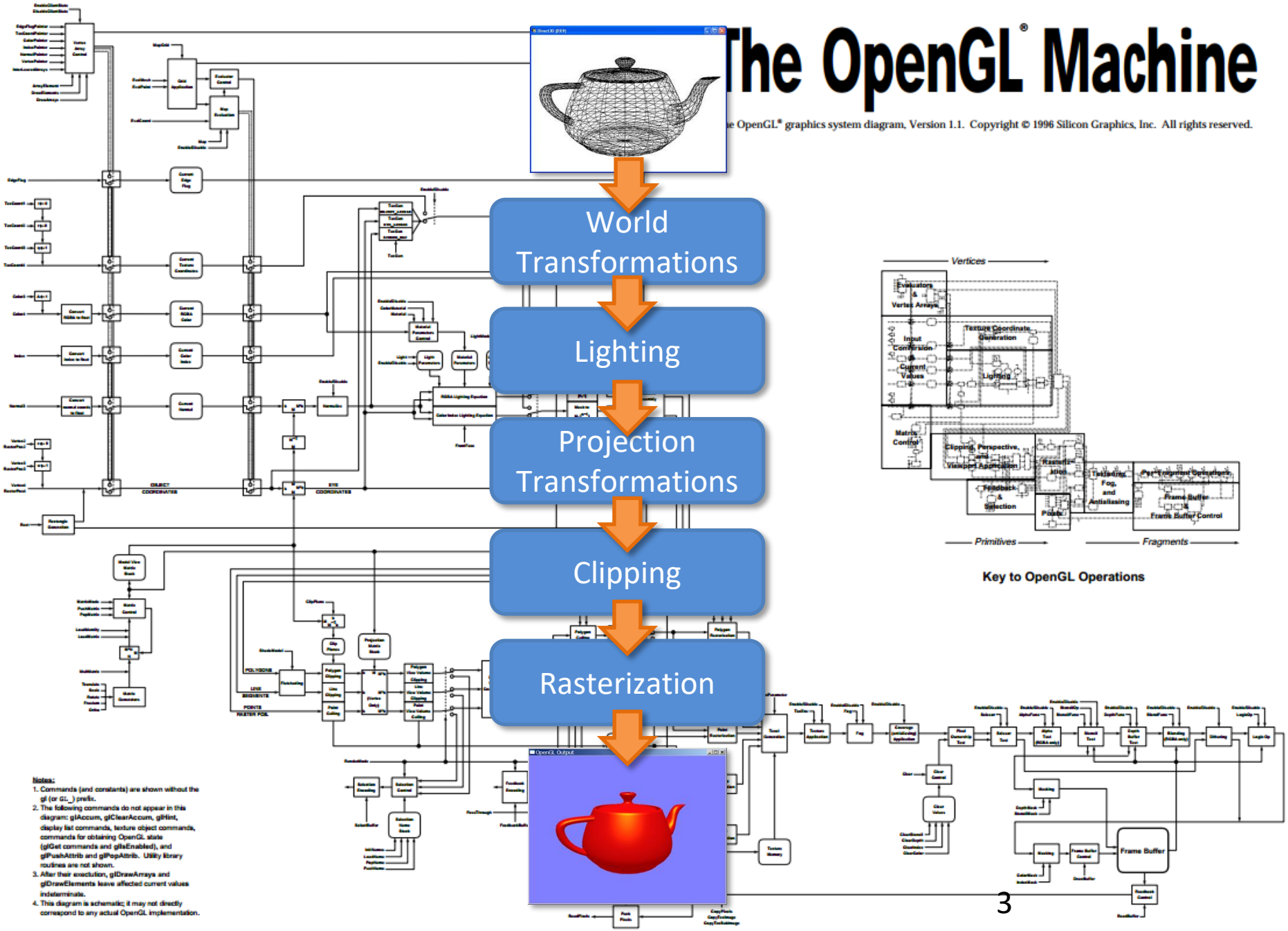
Achille Peternier, adjunct professor



- State machine:
  - Persistent enable/disable flags and settings.
- Procedural:
  - C-style (no C++).

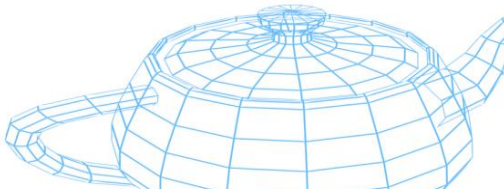


The OpenGL® Machine



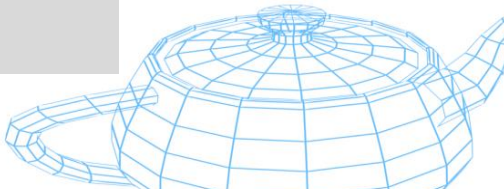
OpenGL syntax

- The API is written in C.
- Methods begin with **`gl`**, constants with **`GL_`**.
- Some methods specify the number of arguments and their type, e.g.:
  - **`glVertex4f(GLfloat x, GLfloat y, Glfloat z, GLfloat w);`**
  - **`glColor3b(GLbyte r, GLbyte g, GLbyte b);`**
- "v" means vector (array), e.g.:
  - **`glVertex3fv(const GLfloat *v);`**
- No primitives for vectors, matrices, quaternions, …
  - Use GLM instead.
  - You find them only in GLSL.



OpenGL syntax

```
#include <GL/gl.h>
#include <glm/gtc/type_ptr.hpp>

// Clear screen to black:
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

// Set object position:
glMatrixMode(GL_MODELVIEW);
glm::mat4 modelMat(…);
glLoadMatrixf(glm::value_ptr(modelMat));

// Set object vertices:
glBegin(GL_TRIANGLES);
   glVertex3f(0.0f, 0.0f, -20.0f);
   glVertex3f(10.0f, 0.0f, -20.0f);
   glVertex3f(5.0, 5.0, -20.0f);
glEnd();
```



OpenGL *et similia*

	OpenGL	OpenGL|ES	WebGL	OpenGL|SC
Since	1992	2003	2011	2004
Current version	4.6	3.2	2.0	2.0.1
Target	Any suitable, but mainly PC and PC-like products (such as graphics workstations, rendering clusters, or gaming consoles)	Embedded and mobile devices (mobile phones, gaming consoles, …)	Web browsers through a JavaScript API (no plugin required)	Security critical systems (avionics, medical, military, etc.). DO-178B certification
Support	All the main operating systems	All the main mobile operating systems	Almost all web browsers, including mobile versions	Vendor-specific
Remarks			- based on OGL ES 2.0 - no fixed pipeline API - HTML5 canvas elem.	- based on a subset of OpenGL 1.3 specs - minimum driver size and complexity

OpenGL|ES syntax example (C)

Enable shader

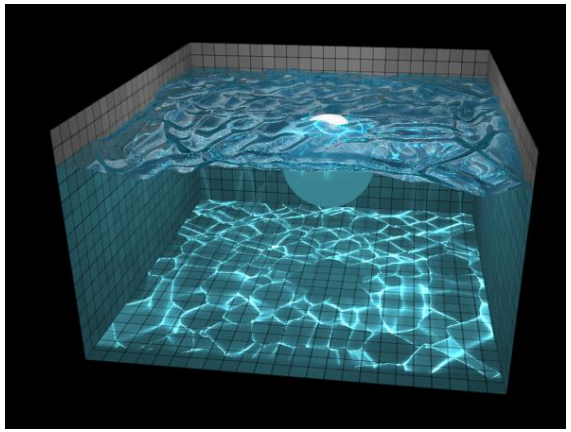
```
// Clear screen to black:
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

// Activate a custom shader:
glUseProgram(shaderProgram);

// Load vertex data:
GLfloat vertices[] = {0.0f, 0.0f, -20.0f,
                      10.0f, 0.0f, -20.0f,
                      5.0f, 5.0f, 0.0f};
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, vertices);
glEnableVertexAttribArray(0);

// Draw the array:
glDrawArrays(GL_TRIANGLES, 0, 3);
```

WebGL syntax example (JavaScript)

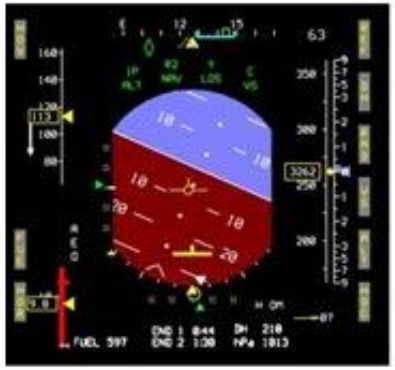
```
// Clear screen to black:
gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

// Activate a custom shader:
gl.useProgram(shaderProgram);

// Load vertex data:
triangleBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, triangleBuffer);
var vertices = [0.0, 0.0, -20.0,
                10.0, 0.0, -20.0,
                 5.0, 5.0, -20.0];
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
              gl.STATIC_DRAW);

// Draw the array:
gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT,
                       false, 0, 0);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

WebGL demos

http://www.spacegoo.com/wingsuit/#



http://madebyevan.com/webgl-water/



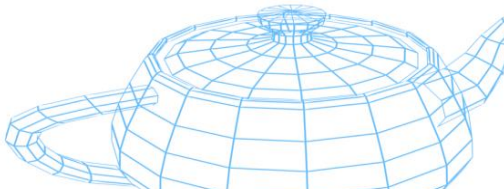

GermanCarFans.com
120

Per-vertex information

- Vertex position (as seen so far):
  - x, y, z[, w] (usually as *float*)

- Vertex color (RGB or RGBA):
  - r, g, b[, a] (usually as *byte*)

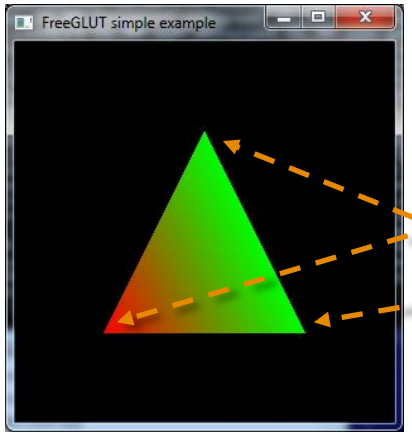
- …we will see additional per-vertex data later in the course (like normal vectors and texture coordinates).



Immediate mode

- Generates primitives according to the type specified and the number of vertices passed.

- A new vertex is generated when **`glVertex*()`** is called, using the last color values specified.

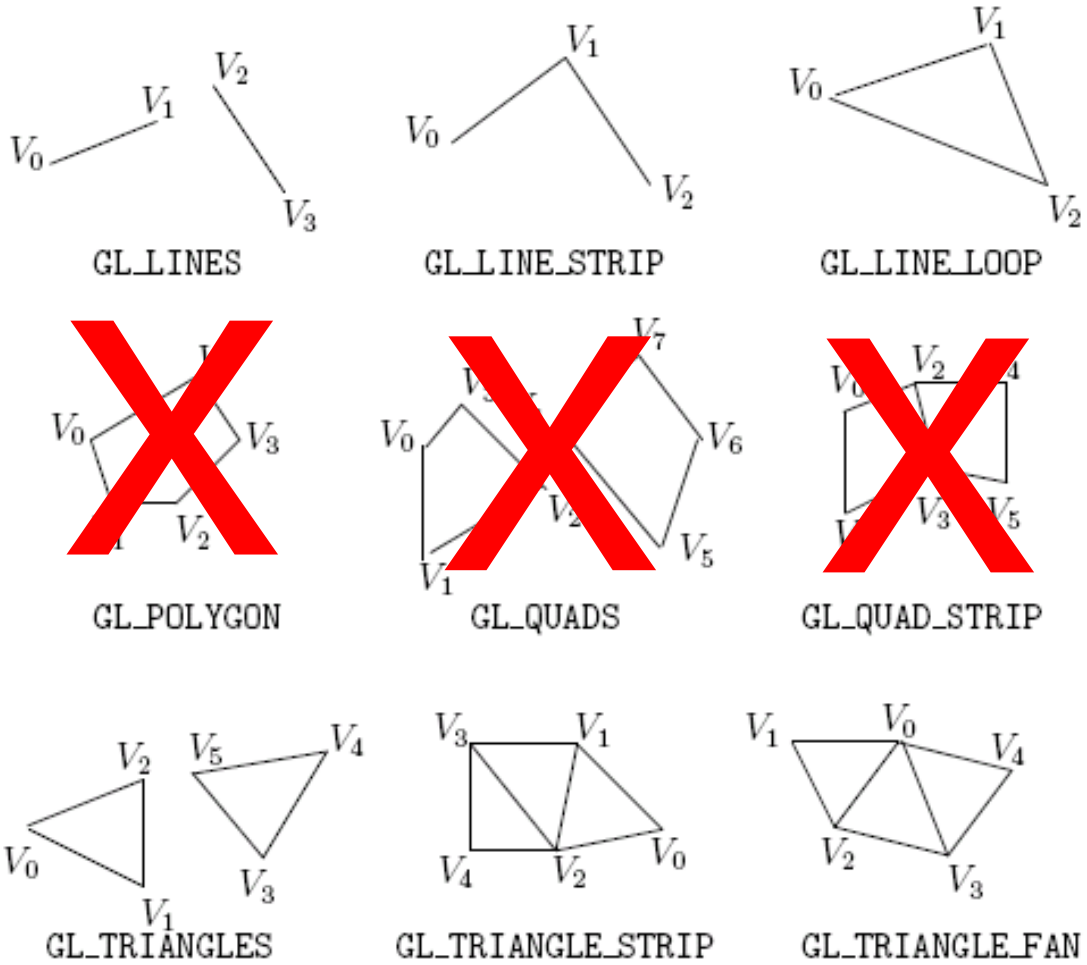


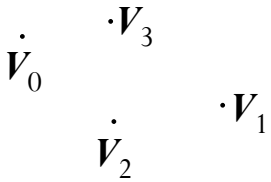

```
glBegin(GL_TRIANGLES);
      glColor3f(1.0f, 0.0f, 0.0f);
         glVertex3f(0.0f, 0.0f, 0.0f);
      glColor3f(0.0f, 1.0f, 0.0f);
         glVertex3f(10.0f, 0.0f, 0.0f);
         glVertex3f(5.0f, 5.0f, 0.0f);
glEnd();
```

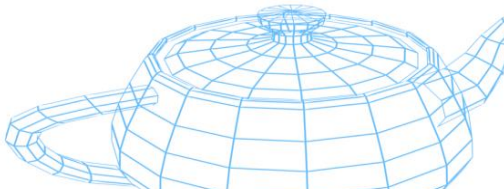
OpenGL primitives

GL_POINTS	Draws points on screen. Every vertex specified is a point. E.g.: point cloud.
GL_LINES	Draws lines on screen. Every two vertices specified compose a line.
GL_LINE_STRIP	Draws connected lines on screen. Every vertex specified after first two are connected.
GL_LINE_LOOP	Draws connected lines on screen. The last vertex specified is connected to first vertex. E.g.: a perimeter.
GL_TRIANGLES	Draws triangles on screen. Every three vertices specified compose a triangle.
GL_TRIANGLE_STRIP	Draws connected triangles on screen. Every vertex specified after first three vertices creates a triangle.
GL_TRIANGLE_FAN	Draws connected triangles like **GL_TRIANGLE_STRIP**, except draws triangles in fan shape.
GL_Q[illegible]	Draws quadrilaterals (4 – sided shapes) on screen. Every four [illegible] compose a quadrilateral.
GL_QUAD_STRIP	Dra[illegible]aterals on screen. Every two vertices specified after first [illegible]pose a conne[illegible]lateral.
GL_[illegible]	Draws a polygon on screen. Polygon can be composed of as [illegible] you want.

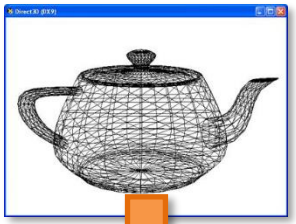
OpenGL primitives







OpenGL Matrices



World Transformations

Lighting

Projection Transformations

Clipping

Rasterization

```
glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(m);
```

*(where m is typically cameraMat^{-1} * transMat * rotMat * scaleMat)*

at will and in any order…

```
glBegin(…)
    glVertex*(…)
glEnd()
```

```
glMatrixMode(GL_PROJECTION);
glLoadMatrixf(m);
```

(where m is typically computed by glm::ortho or glm::perspective)

```
glViewport(startX, startY, width, height);
```

$$\begin{bmatrix} clip_x \\ clip_y \\ clip_z \\ clip_w \end{bmatrix} = \overbrace{\text{projMat}}^{\texttt{GL\_PROJECTION}} * \overbrace{\text{cameraMat}^{-1} * \underbrace{\text{transMat} * \text{rotMat} * \text{scaleMat}}_{\textit{at will and in any order…}}}^{\texttt{GL\_MODELVIEW}} * \overbrace{\begin{bmatrix} obj_x \\ obj_y \\ obj_z \\ 1 \end{bmatrix}}^{\texttt{glVertex*()}}$$

WARNING

Each time you resize the window, don't forget to update the projection matrix and `glViewport()` values accordingly!

Matrices

- OpenGL stores, for each mode, the current matrix.
- Current matrix is set by passing a **`glm::mat4`** pointer to the method **`glLoadMatrixf(float *);`**

```
#include <glm/gtc/type_ptr.hpp>

glMatrixMode(GL_MODELVIEW);
glm::mat4 mv = cameraInv * translation * rotation;
glLoadMatrixf(glm::value_ptr(mv));

glMatrixMode(GL_PROJECTION);
glm::mat4 pj = glm::perspective(…)
glLoadMatrixf(glm::value_ptr(pj));
```

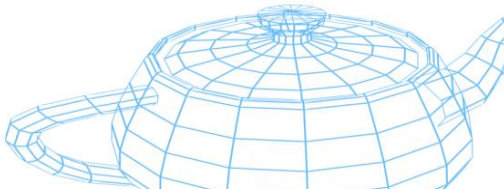
Matrices

- For any given matrix used to position an object in world coordinates:

$$\mathbf{M} = \begin{array}{c} \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \begin{array}{cccc} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{t} \end{array} \end{array}$$

where **x**, **y**, **z** and **t** are column vectors representing the object local:

  - Right direction (**x**).
  - Up direction (**y**).
  - Forward direction (**z**).
  - Position (**t**).

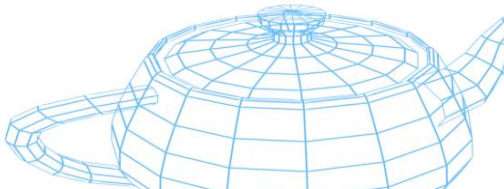


LookAt

- Commodity method for computing the camera matrix from a set of given parameters (eye, center and up):
  - **eye** (vec3): is the position of the camera.
  - **center** (vec3): is the position of the point the camera is looking at.
  - **up** (vec3): is a vector indicating the orientation of the world (typically 0, 1, 0).

- Available through the **`glm::lookAt()`** method:

```
glm::vec3 eye = glm::vec3(0.0f, 0.0f, 10.0f);
glm::vec3 center = glm::vec3(0.0f, 0.0f, 0.0f);
glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);

glm::mat4 viewMat = glm::lookAt(eye, center, up);
```



Tutorial

Advanced FreeGLUT