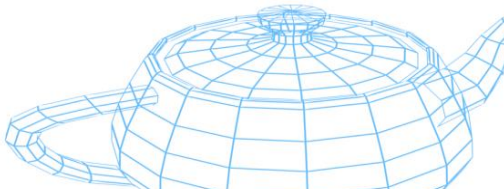
SUPSI

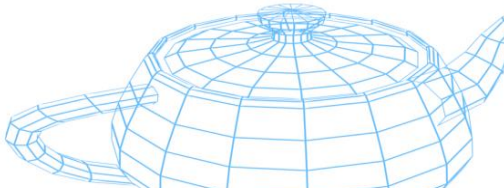
Computer Graphics

OpenGL: an overview

Achille Peternier, lecturer




Applications
Graphics engine
OpenGL
Drivers



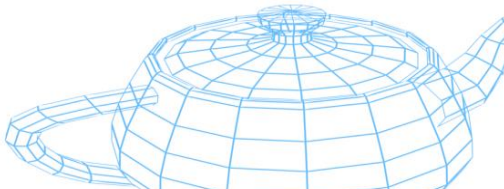
OpenGL overview

- **Open G**raphics **L**ibrary.
- Cross-platform and -language API for 2D/3D computer graphics.
- Abstract API:
  - Can be implemented both in hardware and software.
- Maintained by the non-profit group Khronos.

OpenGL overview

- Rendering only:
  - No audio.
  - No image file processing.
  - No user input.
  - No GUI.
  - No timing.
  - No animations.
  - No multithreading.

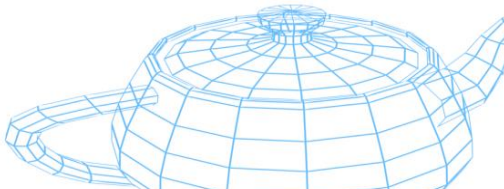
Just raw primitive drawing!



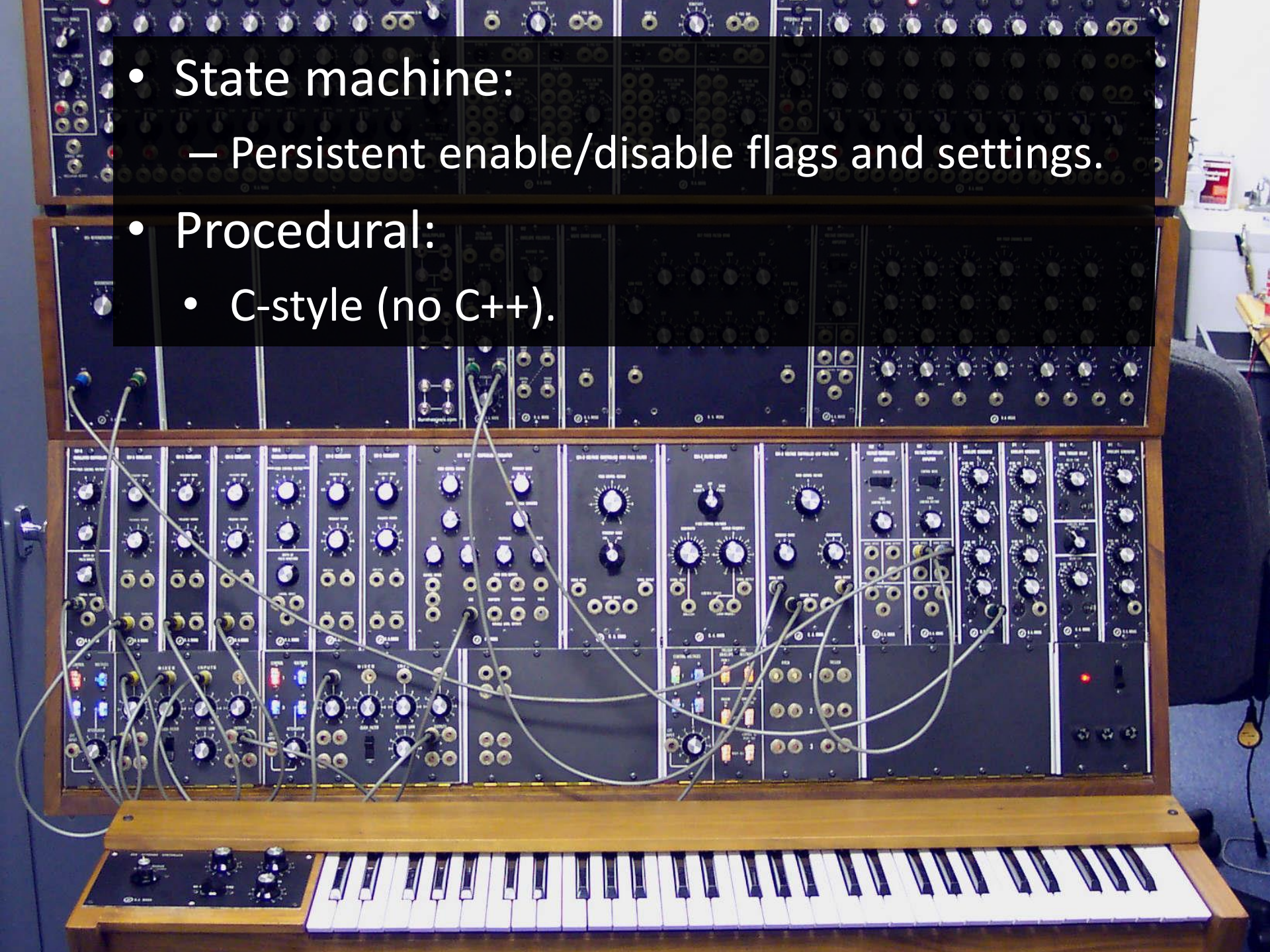
OpenGL overview

- Open specification:
  - Not *open-source*: there's nothing to download (except the document defining the specs).
  - Opposed to ISO standards, where you have to pay for using them.
  - You can read the specs and make your device accessible through your own implementation of the specs:
    - Mesa 3D is an open-source *implementation* of the open *specification.*

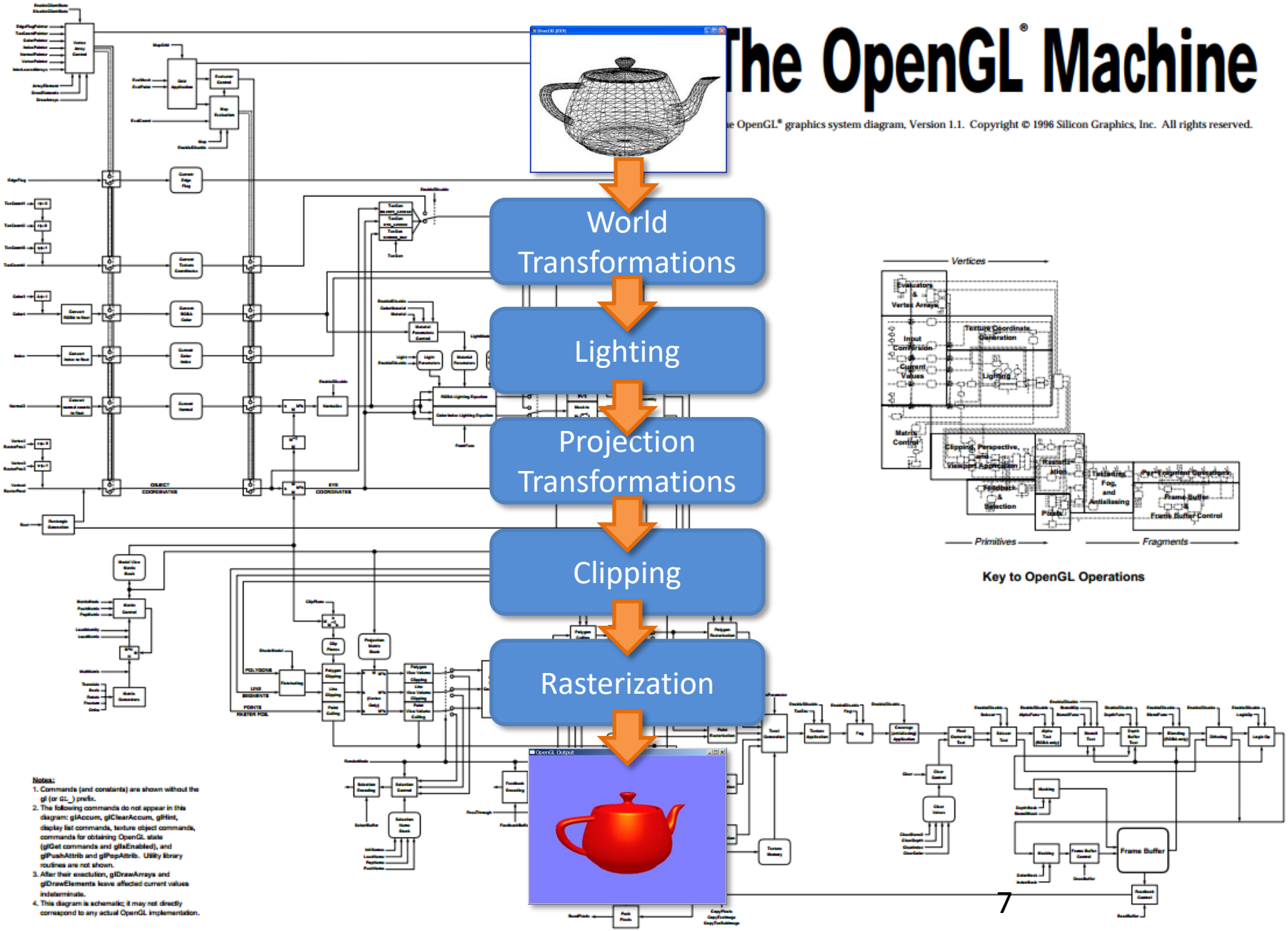
- The specification is maintained and regularly updated by the Khronos Group:
  - Formerly, the Architectural Review Board (ARB) was responsible for that.



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

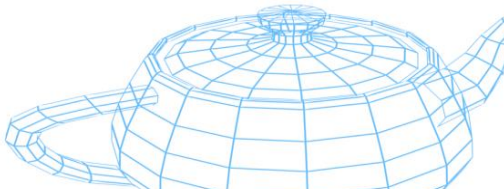


The OpenGL® Machine



Using OpenGL

- Requires the initialization of a **context**:
  - A context is a container allocated by the driver to store the state of the API and the graphic resources used by a windowed application.
  - Context creation is platform-specific.
  - This part is not portable:
    - Software needs one specific initialization procedure on each platform.
    - Auxiliary libraries are often used to deal with this aspect.

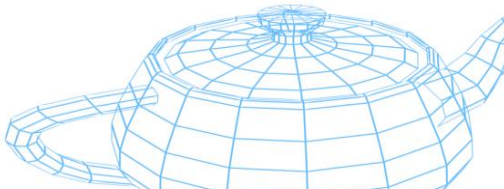


Using OpenGL

- Include **`GL/gl.h`**

- On Windows:
  - Link your code to **opengl32.lib** (also when compiling in x64):
    - **`opengl32.dll`** loads and checks for a real OpenGL driver:
      - The driver is typically installed when you install your full graphic drivers (and not with the default drivers installed by Windows).
    - **`opengl32.dll`** belongs to Microsoft and features OpenGL 1.1:
      - For more recent versions, use extensions or (much better) an external library that does the job for you (e.g.: Glee, GLEW).

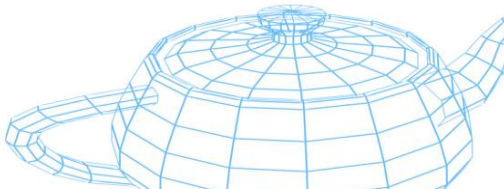
- On Linux:
  - Link your code to **libGL.so**



Using OpenGL

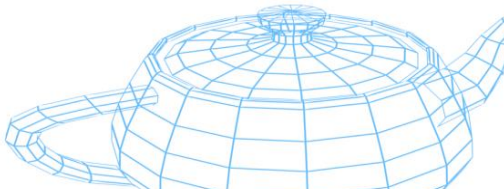
- On Windows:
  - Without a proper driver installed, a software emulation of OpenGL 1.1 is provided (reported as “Windows GDI”).
  - Use **`glinfo2.exe`** for a quick test (available on iCorsi):

- On Linux:
  - Without proprietary drivers, a software, open-source implementation of OpenGL (up to recent versions, depending on the available hardware) is provided (reported as “Mesa”).
  - Use **`glxinfo`** for a report about the available OpenGL configuration.



Using OpenGL

- Some notebooks have more than one graphics card:
  - Integrated GPU (e.g., Intel HD graphics);
    - Inferior 3D performance, but longer battery life.
  - Discrete GPU:
    - Ideal for 3D gaming and CAD, but more power-hungry.
  - Nvidia Optimus and AMD Dynamic Switchable Graphics use this technology: make sure that you run your OpenGL application with the correct profile.



Context creation example (Windows)

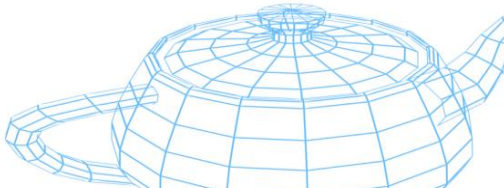
```
PIXELFORMATDESCRIPTOR pfd =
   {
       sizeof(PIXELFORMATDESCRIPTOR),
       1,
       PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,  //Flags
       PFD_TYPE_RGBA,            // The kind of framebuffer: RGBA or palette
       32,                       // Colordepth of the framebuffer
       0, 0, 0, 0, 0, 0,
       0,
       0,
       0,
       0, 0, 0, 0,
       24,                       // Number of bits for the depthbuffer
       8,                        // Number of bits for the stencilbuffer
       0,                        // Number of Aux buffers in the framebuffer
       PFD_MAIN_PLANE,
       0,
       0, 0, 0
   };
```

(see full example at http://www.opengl.org/wiki/Creating_an_OpenGL_Context_(WGL))

Context creation example (Windows)

```
// Add error-checking everywhere…

// Get device context:
deviceContext = GetDC(hWnd);

// Find a pixel format matching the PFD previously described:
PixelFormat = ChoosePixelFormat(deviceContext, &pfd);

// Set pixel format for this device context:
SetPixelFormat(deviceContext, PixelFormat, &pfd);

// Get a rendering context for this device:
renderingContext = wglCreateContext(deviceContext);

// Apply the chosen rendering context to the device:
wglMakeCurrent(deviceContext, renderingContext);
```

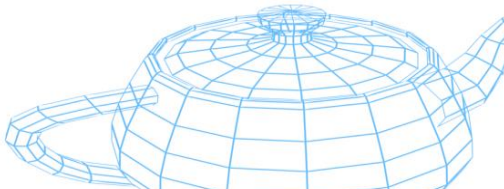
Win-specific methods (*wgl**)



Context creation example (Windows)

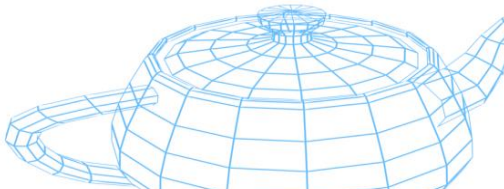
- "HelloWorld" example ~400 lines of code:
  - More than 50% is Windows-specific.
  - Similar case on other platforms (e.g.: X11 through the **`glx*`** and MacOS through the **`agl*`** extensions).

- Source code example:
  http://nehe.gamedev.net/tutorial/creating_an_opengl_window_(win32)/13001/



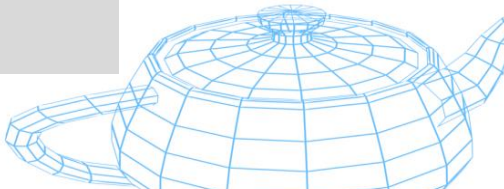
OpenGL syntax

- 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.



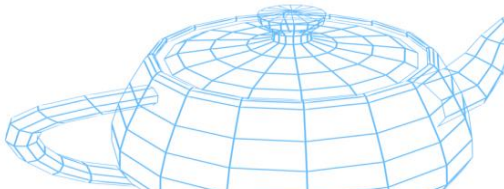
OpenGL syntax

```
#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 syntax

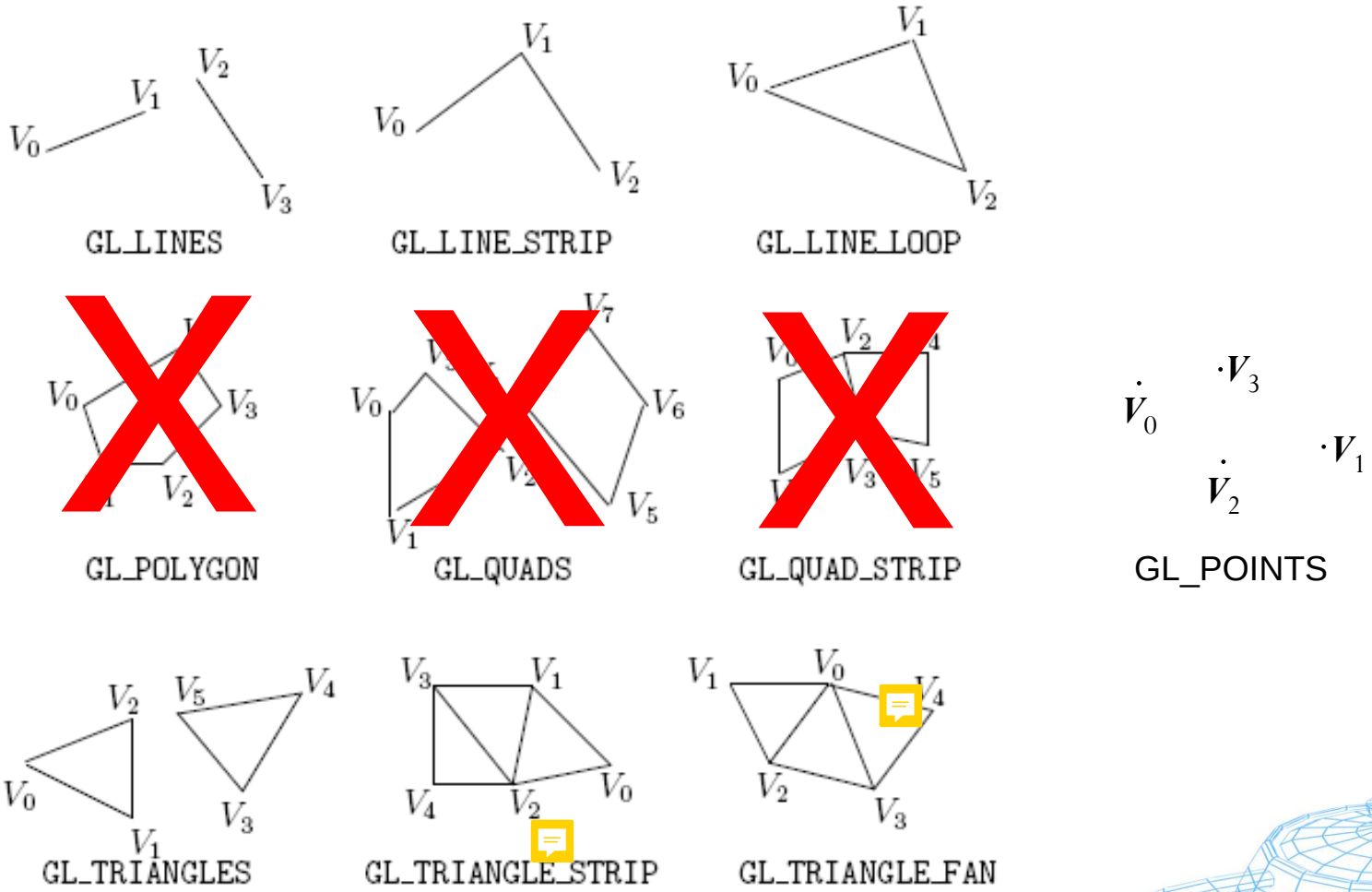
- Once the context is initialized, use **`glGetString()`** to get NULL terminating strings with several pieces of information, like:
  - **`GL_VERSION`** OpenGL supported version.
  - **`GL_VENDOR`** driver's implementer.
  - **`GL_RENDERER`** renderer used (usually the name of the GPU).
  - **`GL_EXTENSIONS`** a (long) list with all the supported extensions.



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~~	~~Draws quadrilaterals (4 – sided shapes) on screen. Every four v[illegible]fied 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 extensions

- Extensible through an extension mechanism that allows modifying the official specification through specific features:
  - Introduced as proprietary extensions, often available only on some platforms, e.g.:
    `GL_NV_point_sprite`
    `GL_ATI_texture_float`
    …
  - When selected extensions meet consensus, they are promoted to the **`GL_ARB_*`** or **`GL_KHR_*`** group.
    - **`GL_ARB_*`** and **`GL_KHR_*`** extensions become mainstream in the next major release of OpenGL and must be supported by everyone wants to comply with the specs..

OpenGL extensions

- Loaded by accessing extended API functions through their function pointers (dynamically, at runtime):
  - Dynamic Link Library magic.

- **`wglGetProcAddress();`** (on Windows)
  `glXGetProcAddress();` (on Linux)

OpenGL extensions

1. Check the string returned by **`glGetString(GL_EXTENSIONS)`** for the required extension (e.g., “GL_ARB_window_pos”):
   - Read the extension specs for information on how to use it: http://www.opengl.org/registry/
   - In our case: http://www.opengl.org/registry/specs/ARB/window_pos.txt

2. If the extension is supported and the specs say that it implements new methods, get the required function pointers, e.g.:

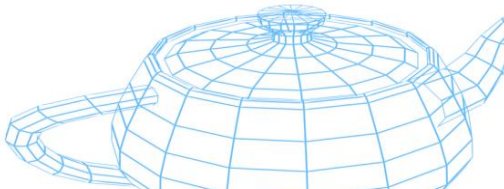
```
PFNGLWINDOWPOS2IPROC glWindowPos2iARB;
glWindowPos2iARB = (PFNGLWINDOWPOS2IPROC) wglGetProcAddress("glWindowPos2iARB");
```

Function prototypes are defined in **`glext.h`** (regularly updated with the latest extensions):
http://www.opengl.org/registry/api/GL/glext.h

(more than 12’000 lines!)

Auxiliary libs

- GLUT (Open**GL** **U**tility **T**oolkit)
  - Features a series of common functionalities required by small CG projects using OpenGL for rendering.
- FreeGLUT ✓
  - Same as GLUT but still supported and with a less restrictive license.
- GLM (Open**GL** **M**ath) ✓
  - Math library replacing deprecated OpenGL features and much more.
- GLEW (Open**GL** **E**xtension **W**rangler)
  - Extension manager to automatically take advantage from the latest versions of OpenGL.
- GLFW (Open**GL** **F**rame**W**ork)
  - Open**GL** **F**rame**W**ork, similar to FreeGLUT but without the callback boilerplate.
- SDL (**S**imple **D**irectMedia **L**ayer)
  - Supports input, audio, timing, threads. Also similar to GLUT and GLFW, but more generic (without being explicitly done for OpenGL)
- *And many more…*



OpenGL *et similia*

	OpenGL	OpenGL|ES	WebGL	OpenGL|SC
Since	1992	2003	2011	2004
Current version	4.6	3.2	2.0	2.0
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);
```

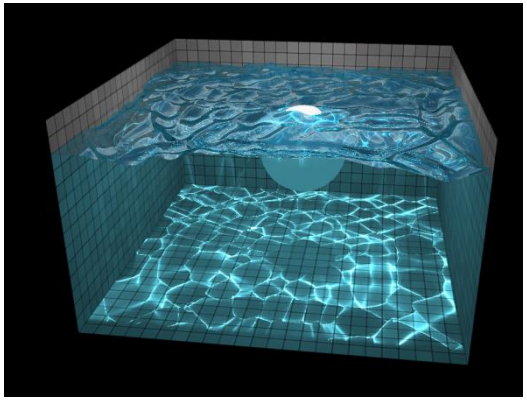
WebGL syntax example (JavaScript)

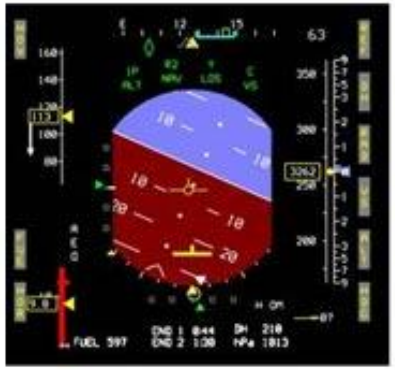
```
// Clear screen to black:
gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

// 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

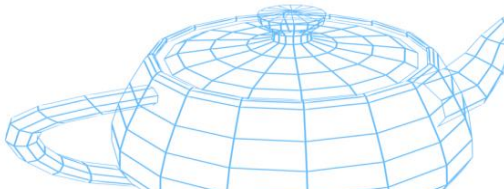
A brief history of OpenGL

- IRIS GL (pre-OpenGL):
  - Proprietary graphics library written by SGI.
  - Used in their graphics workstations.

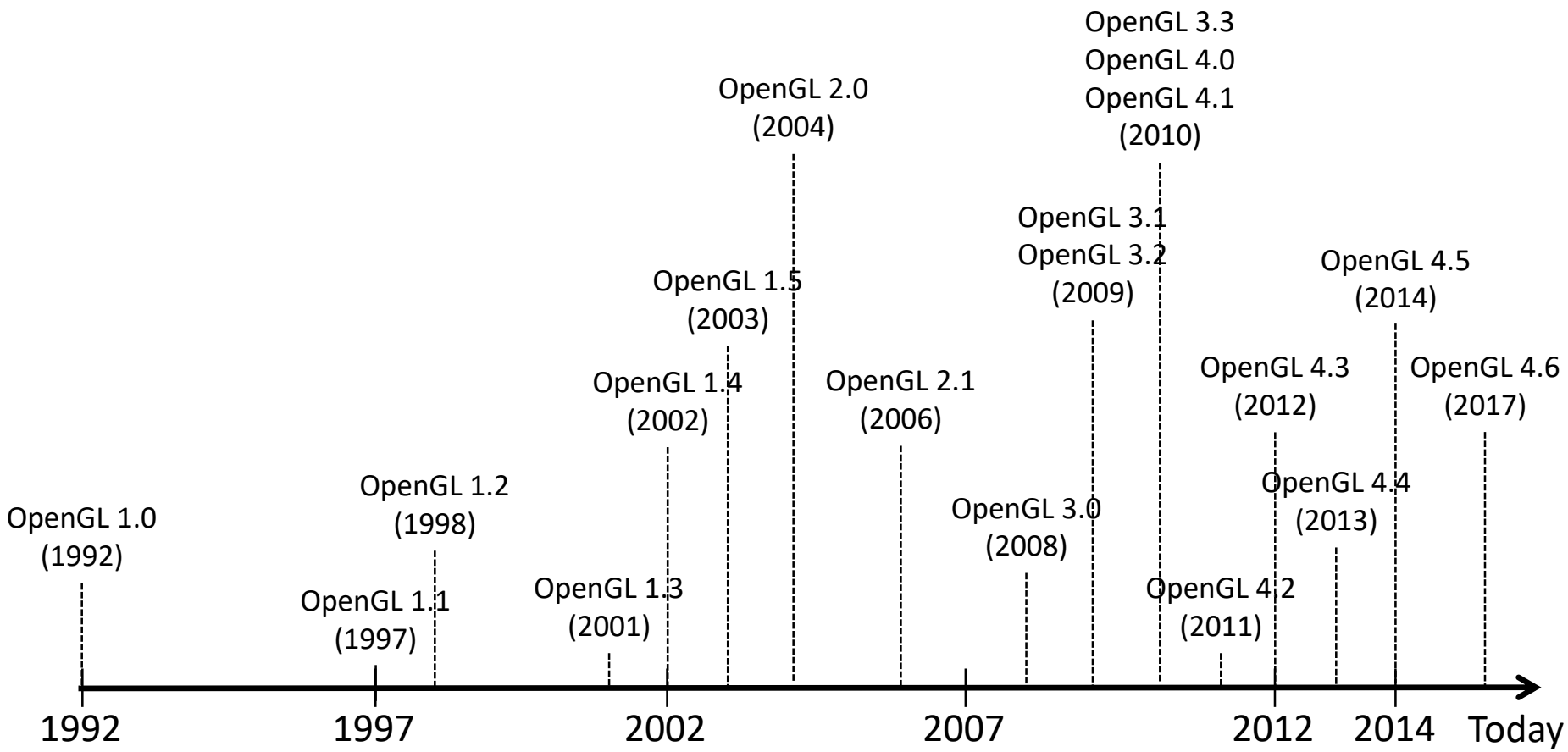
- IRIS GL 1 (1983) to IRIS GL 4 (1991).

- OpenGL specs (1991).

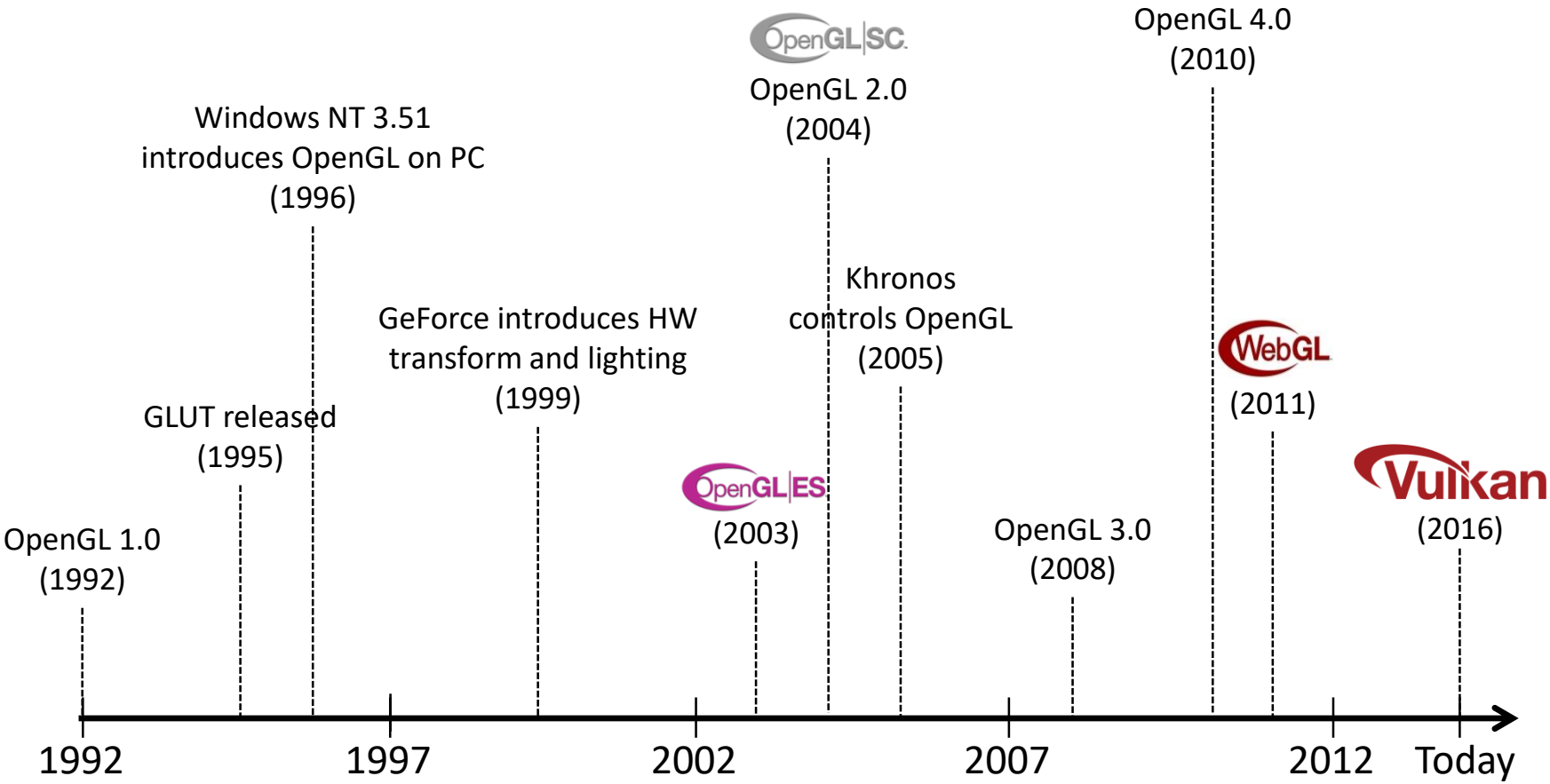
- OpenGL release (1992).



A brief history of OpenGL

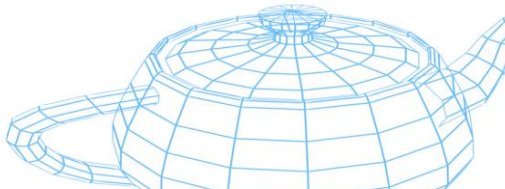


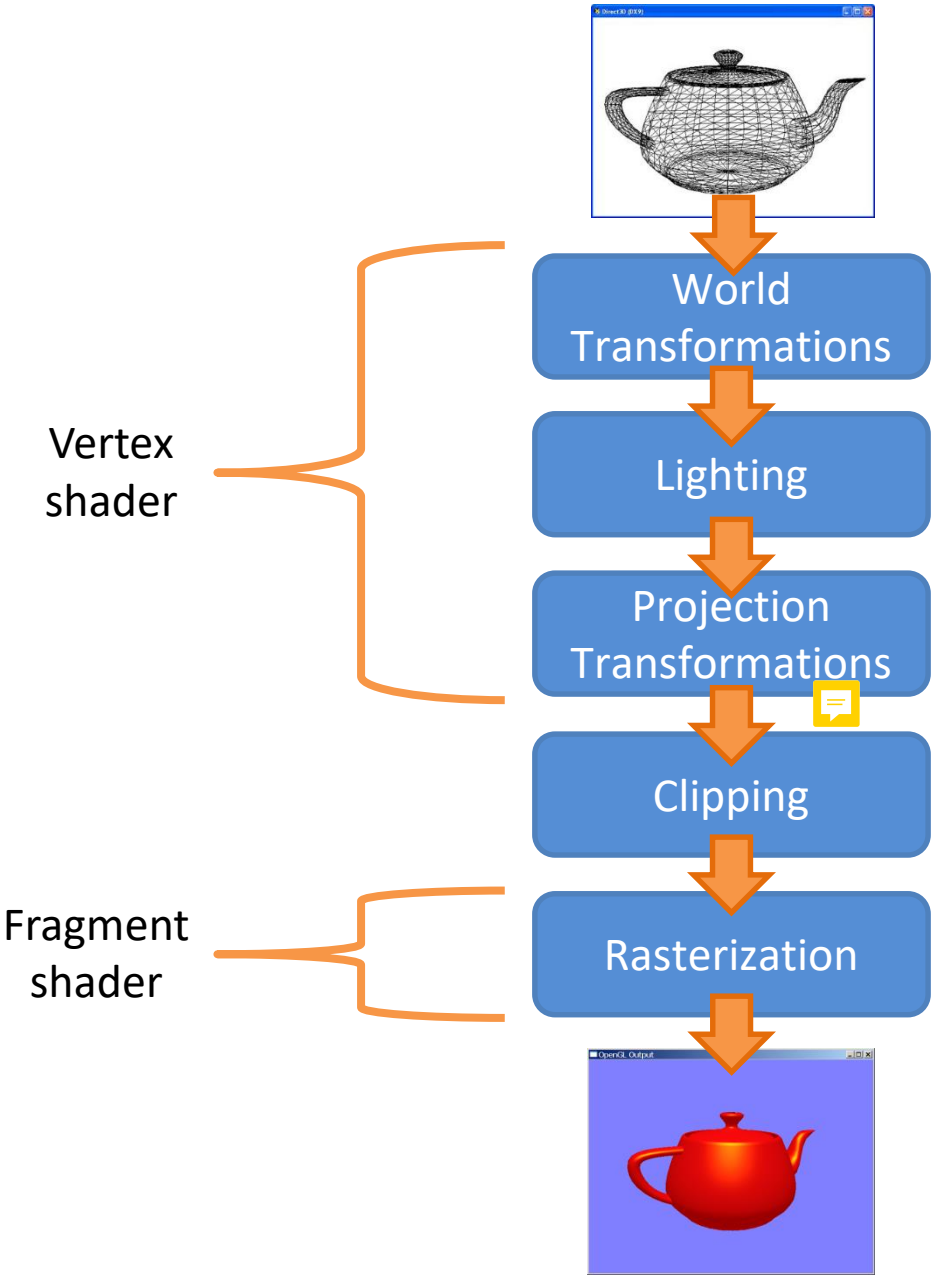
A brief history of OpenGL



A brief history of OpenGL

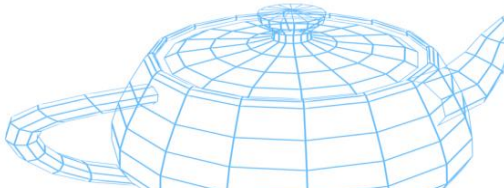
- OpenGL 2.0:
  - Vertex and fragment shaders for improved pipeline programmability:
    - The pipeline was *fixed* (hardcoded) before 2.0.
    - On modern OpenGL implementations, the fixed pipeline is emulated through shaders for backward compatibility.
  - From OpenGL 3.0, OpenGL ES 2.0 and WebGL 1.0 onwards, the fixed pipeline has been removed:
    - Shaders are the only way to go.






```
void main()
{
    // Transform the vertex:
    gl_Position =
        gl_ModelViewProjectionMatrix *
        gl_Vertex;
}
```

```
void main()
{
    // Set each pixel to red:
    gl_FragColor =
        vec4(1.0, 0.0, 0.0, 1.0);
}
```



A brief history of OpenGL

- From OpenGL 1.0 to 2.x:
  - New features added at each release.
  - Full backward-compatibility.

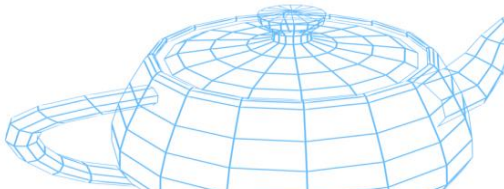
- From OpenGL 3.0 on:
  - Deprecation model adopted for old/vintage functions:
    - Extensions are used to promote new features into the specifications, while deprecation is used for removing obsolete functionalities from OpenGL.
    - The driver complexity issue.
  - “core” and “compatibility” profiles introduced:
    - Core profiles are no longer backward-compatible.
  - Complete removal of the fixed pipeline.
  - Entirely focused on the rendering:
    - No math, no built-in variables -> use your own lib (e.g.: GLM).

A brief history of OpenGL

- Application-generated object names
- Color index mode
- Shading language 1.10 and 1.20
- Begin/End primitive specification
- Edge flags
- Fixed function vertex processing
- Client-side vertex arrays
- Rectangles
- Current raster position
- Two-sided color selection
- Non-sprite points
- Wide lines and line stripple
- Quadrilateral and polygon primitives
- Separate polygon draw mode
- Polygon stripple
- Pixel transfer modes and operations
- Pixel drawing
- Bitmaps
- Legacy OpenGL 1.0 pixel formats

Deprecated OpenGL functionalities (since OpenGL 3.0):

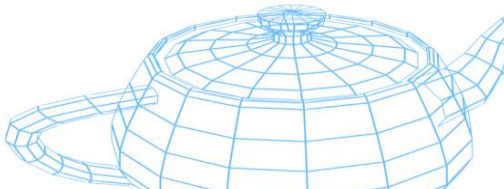
- Legacy pixel formats
- Depth texture mode
- Texture wrap mode CLAMP
- Texture borders
- Automatic mipmap generation
- Fixed function fragment processing
- Alpha test
- Accumulation buffers
- Context framebuffer size queries
- Evaluators
- Selection and feedback mode
- Display lists
- Hints
- Attribute stacks
- Unified extension string
- glTranslate/glRotate/glMultMatrix, etc.



A brief history of OpenGL

- Current version: 4.6

- Future versions?
  - New proprietary standards emerged (and faded…):
    - AMD Mantle.
    - Apple Metal.
  - OpenCL, CUDA, etc.:
    - Interoperability.
  - Next Generation OpenGL initiative:
    - SIGGRAPH 2014 https://www.khronos.org/assets/uploads/developers/library/2014-siggraph-bof/OpenGL-Ecosystem-BOF_Aug14.pdf
    - Led to the creation of Vulkan.



Documentation

- OpenGL SDK home: http://www.opengl.org/sdk/
- OpenGL reference cards (only recent versions):

- Tons of online tutorials, examples, forums, etc.