OpenGL4

http://www.opengl-tutorial.org/ **OpenGL 3.3** and later !

Tutorial 1 : Opening a window

- Introduction
- Prerequisites
- Forget Everything
- Building the tutorials
    - Building on Windows
    - Building on Linux
    - Building on Mac
    - Note for Code::Blocks
- Running the tutorials
- How to follow these tutorials
- Opening a window

Miscellaneous:

http://www.opengl-tutorial.org/miscellaneous/math-cheatsheet/

http://www.opengl-tutorial.org/miscellaneous/clicking-on-objects/

webGL2

https://webgl2fundamentals.org/

Opening a window

https://webgl2fundamentals.org/webgl/lessons/webgl-setup-and-installation.html

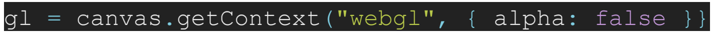
WebGL is JavaScript instead of C
WebGL is designed to run in an Internet browser

webGL has less functionality than OpenGL 3.3+
(no Tessellation, Compute, Geometry Shaders)

Miscellaneous:

https://webgl2fundamentals.org/webgl/lessons/webgl-picking.html

https://webgl2fundamentals.org/webgl/lessons/webgl-and-alpha.html



```
gl = canvas.getContext("webgl", { alpha: false }}
```

Three.js (not used)

Threejs.org

https://threejs.org/docs/#manual/en/introduction/Creating-a-scene

3D graphics using JavaScript, without having to learn WebGL

GPU memory and (vertex) shaders

Shader = GPU program (compiled at the start of the CPU OpenGL program!)

B = Buffer = chunk of GPU memory
VS = Vertex Shader = GPU program modifying vertices
FS = Fragment Shader = GPU program modifying pixels
VA = VertexAttribute = input to VS
Attribute index j = j th input argument to VS

Example structure:
http://openglbook.com/chapter-3-index-buffer-objects-and-primitive-types.html

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-9-vbo-indexing/
https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL)

VS Attributes (VAO)

Attributes explain what is in the buffer

- ➢ ask for a handle (=name=int) to a VA: Gen.VA (1,&handle)
- ➢ (state:) for this spot in GPU memory: Bind.VA (buffertype,handle)
- ➢ of the j th Vertex Shader input: Enable.VA.array(j)
- ➢ define the input (buffer) format of kth argument of VS: VA.Pointer(k, length etc)
- ➢ Disable.VA.array(j)

Buffer initialization (VBO)

Buffer = data

- ➢ ask for a handle (=name=int) to a buffer: B.GenObject(1,&handle) ⇒
- ➢ (state:) for this spot in GPU memory: Bind.B.(B-type,handle)
- ➢ allocate memory for CPU data at currently bound buffer of this type:
  B.Data(B-type,size,CPU data source,usage)
- ➢ unbind: Bind.B(B-type,0)

VAO VBO

To avoid messy binding/unbinding of buffers and passing all the settings for each vertex attribute, best practice is to organize the code as follows.

initialization :
for each batch

- ➢ generate , store , bind VAO
- ➢ bind all buffers VBO for a draw call
- ➢ unbind VAO

main loop / whenever you render :
for each batch

- ➢ bind VAO
- ➢ glDrawArrays (. . .) ; or glDrawElements (. . .) ; etc.
- ➢ unbind VAO

How does the GPU allocation of memory and passing of data mirror that of C++ on the CPU?
What is a uniform?