Computer Graphics labs

Lab 0 - Introduction to Computer Graphics

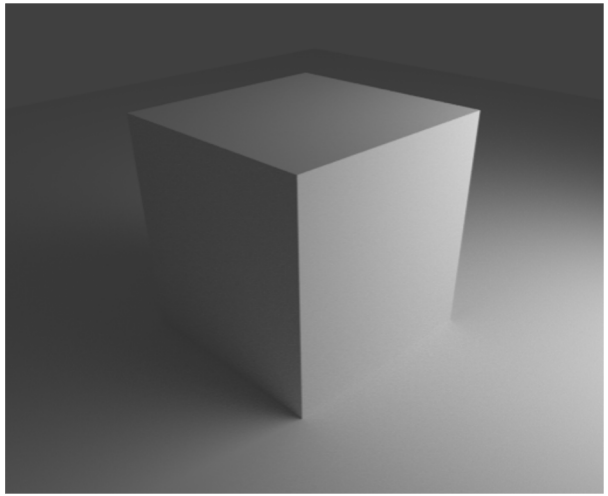
Teaching Assistants

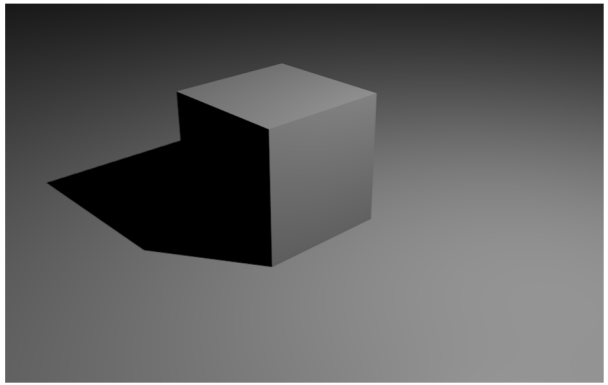
- Yahia Zakaria
  - Email: yahiazakaria13@gmail.com
  - Alternative Email: yzetman@eng.cu.edu.eg (checked less regularly)
  - Office Hours:
    - Tuesday 12-2

Computer Graphics

- In Games.
- In Movies.
- In CAD (Computer-aided design).

```
# cube.obj
#

mtllib cube.mtl
o cube

v -0.500000 -0.500000 0.500000
v 0.500000 -0.500000 0.500000
v -0.500000 0.500000 0.500000
v 0.500000 0.500000 0.500000
v -0.500000 0.500000 -0.500000
v 0.500000 0.500000 -0.500000
v -0.500000 -0.500000 -0.500000
v 0.500000 -0.500000 -0.500000

vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 1.000000 1.000000

vn 0.000000 0.000000 1.000000
vn 0.000000 1.000000 0.000000
vn 0.000000 0.000000 -1.000000
vn 0.000000 -1.000000 0.000000
vn 1.000000 0.000000 0.000000
vn -1.000000 0.000000 0.000000

g cube
usemtl cube
s 1
f 1/1/1 2/2/1 3/3/1
f 3/3/1 2/2/1 4/4/1
s 2
f 3/1/2 4/2/2 5/3/2
f 5/3/2 4/2/2 6/4/2
s 3
f 5/4/3 6/3/3 7/2/3
f 7/2/3 6/3/3 8/1/3
s 4
f 7/1/4 8/2/4 1/3/4
f 1/3/4 8/2/4 2/4/4
s 5
f 2/1/5 8/2/5 4/3/5
f 4/3/5 8/2/5 6/4/5
s 6
f 7/1/6 1/2/6 5/3/6
f 5/3/6 1/2/6 3/4/6
```





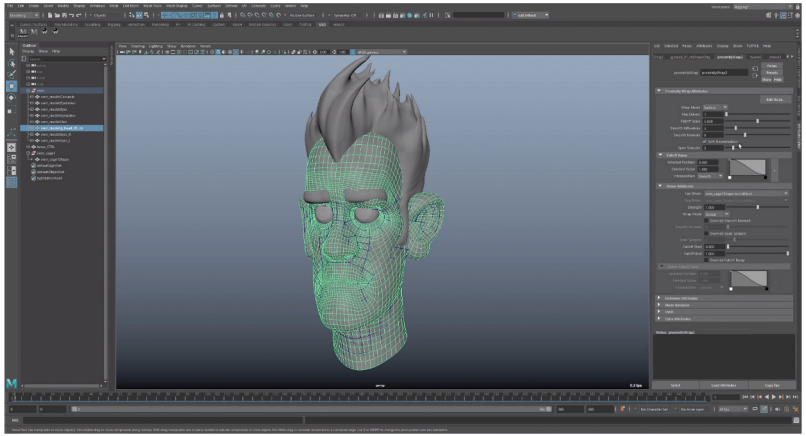
Graphics Engineer ≠ Game Developer

A game developer could use an Engine to produce a game with no knowledge about graphics.

Graphics Engineer ≠ Game Developer

A graphics engineer creates different types of applications like:

- Game Engines.
- Rendering Softwares for animation movies.
- Medical Applications.
- ... etc

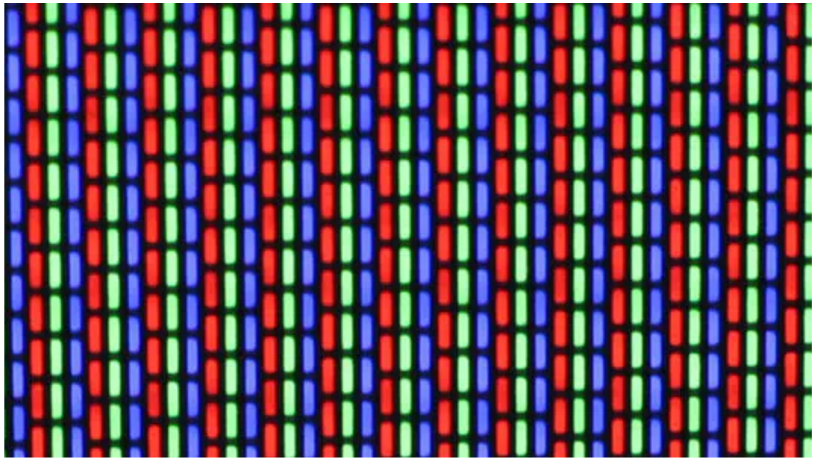


Maya

Why Graphics Processing Units GPUs?

Some applications are inherently parallel.

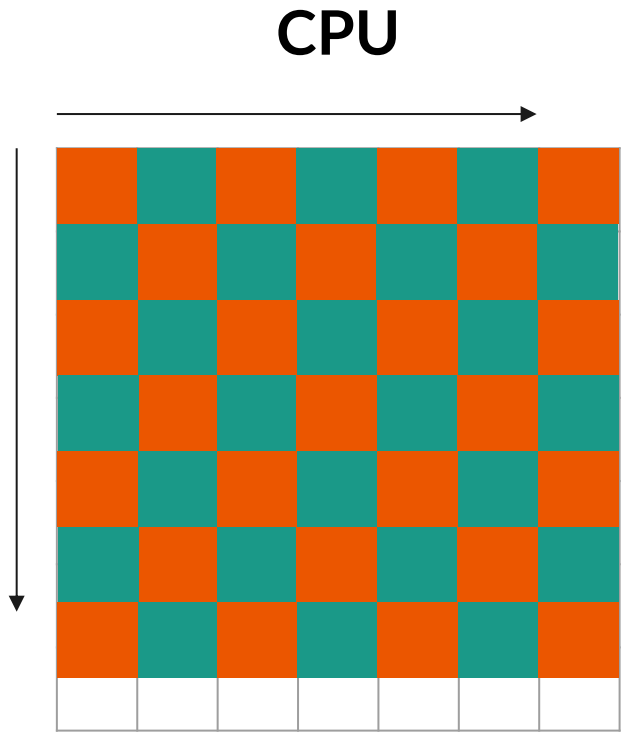
Like drawing pixels on the screen.



CPU vs GPU

CPU executes operations sequentially.

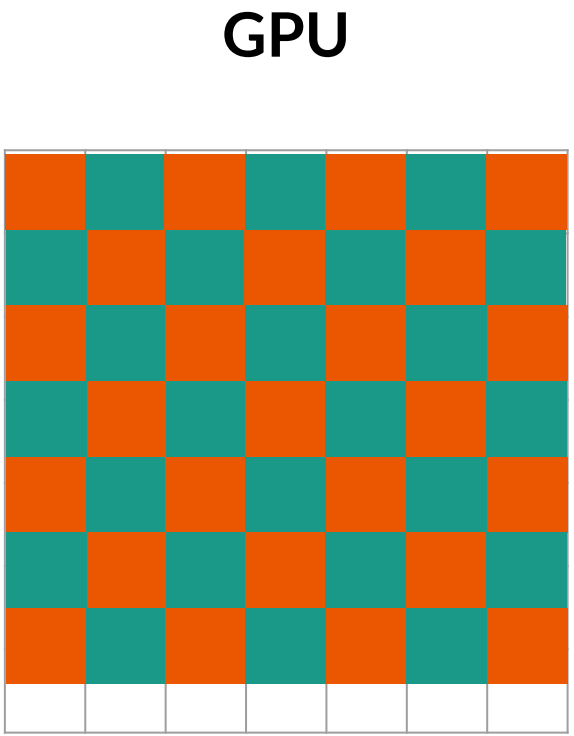
A CPU will draw the screen pixel by pixel.

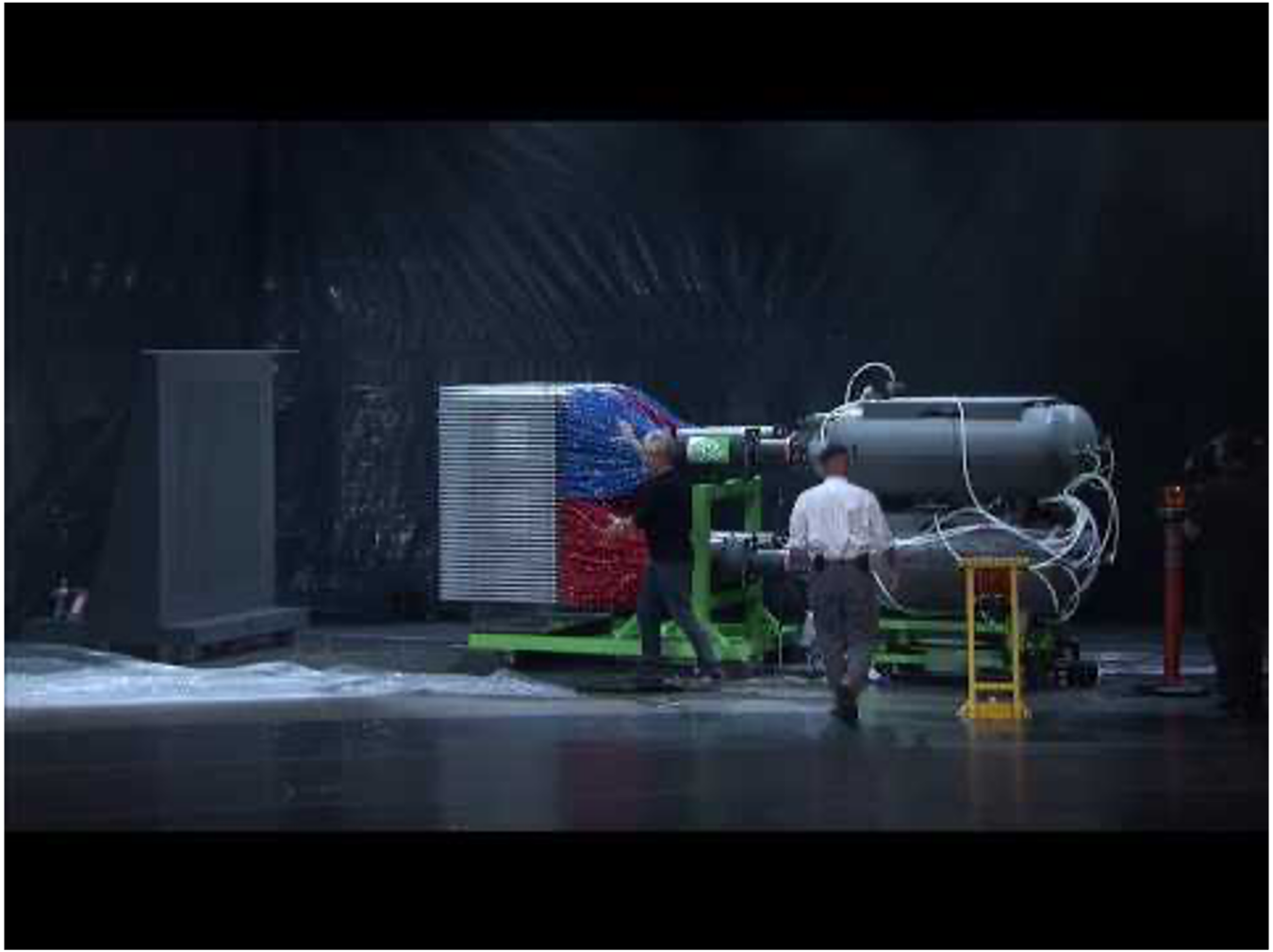


CPU vs GPU

GPU executes operations in parallel.

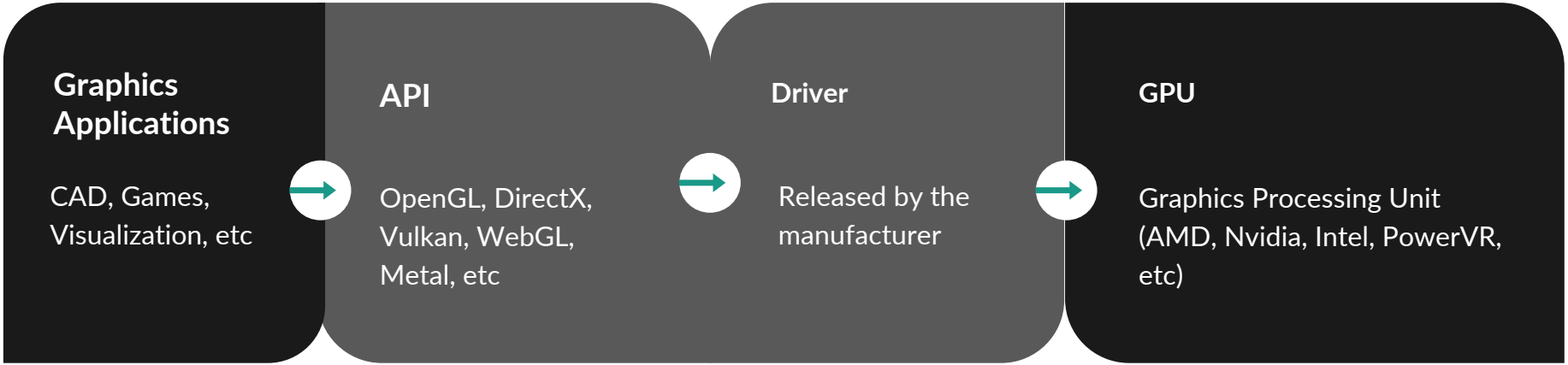
A GPU could draw the screen at once (Theoretically).






UNLOCKED
FX
PROCESSOR
AMD
ENTERTAINMENT EDITION
ENTERTAINMENT EDITION
RADEON
RADEON

But How can we talk to the GPU?



Released in 1992

Released in 1995

Released in 2003

Released in 2011

Released in 2016

Released in 2015

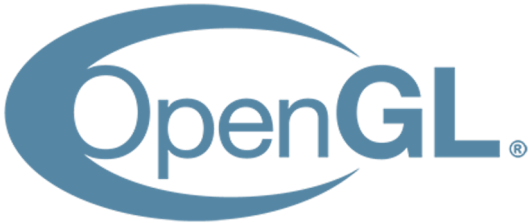
Released in 2014

Not Released Yet

This course

- We will start by exploring openGL with C++.
- And explore some rendering examples and use cases through the labs.

Note: OpenGL is a C-API which contains only functions and constants. We will use C++ for convenience and to use helper libraries written in C++.



Why OpenGL?

- Ease of use.
- Cross platform:
  - Windows
  - Linux
  - Mac (It was deprecated by Apple in 2018).
- Up-to-date with modern gpu features.

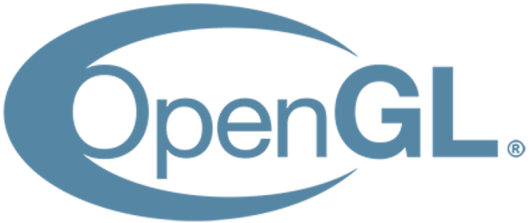
OpenGL version history

Version	Release Date	Features
1.1	March 4, 1997	Texture objects
1.2	March 16, 1998	3D textures, BGRA and packed pixel formats,[22] introduction of the *imaging subset* useful to image-processing applications
1.2.1	October 14, 1998	A concept of ARB extensions
1.3	August 14, 2001	Multitexturing, multisampling, texture compression
1.4	July 24, 2002	Depth textures, GLSlang[23]
1.5	July 29, 2003	Vertex Buffer Object (VBO), Occlusion Queries[24]
2.0	September 7, 2004	GLSL 1.1, MRT, Non Power of Two textures, Point Sprites,[25] Two-sided stencil[24]
2.1	July 2, 2006	GLSL 1.2, Pixel Buffer Object (PBO), sRGB Textures[24]
3.0	August 11, 2008	GLSL 1.3, Texture Arrays, Conditional rendering, Frame Buffer Object (FBO)[26]
3.1	March 24, 2009	GLSL 1.4, Instancing, Texture Buffer Object, Uniform Buffer Object, Primitive restart[27]
3.2	August 3, 2009	GLSL 1.5, Geometry Shader, Multi-sampled textures[28]
3.3	March 11, 2010	GLSL 3.30, Backports as much function as possible from the OpenGL 4.0 specification
4.0	March 11, 2010	GLSL 4.00, Tessellation on GPU, shaders with 64-bit precision[29]
4.1	July 26, 2010	GLSL 4.10, Developer-friendly debug outputs, compatibility with OpenGL ES 2.0[30]
4.2	August 8, 2011[31]	GLSL 4.20, Shaders with atomic counters, draw transform feedback instanced, shader packing, performance improvements
4.3	August 6, 2012[32]	GLSL 4.30, Compute shaders leveraging GPU parallelism, shader storage buffer objects, high-quality ETC2/EAC texture compression, increased memory security, a multi-application robustness extension, compatibility with OpenGL ES 3.0[33]
4.4	July 22, 2013[34]	GLSL 4.40, Buffer Placement Control, Efficient Asynchronous Queries, Shader Variable Layout, Efficient Multiple Object Binding, Streamlined Porting of Direct3D applications, Bindless Texture Extension, Sparse Texture Extension[34]
4.5	August 11, 2014[7][35]	GLSL 4.50, Direct State Access (DSA), Flush Control, Robustness, OpenGL ES 3.1 API and shader compatibility, DX11 emulation features
4.6	July 31, 2017[36][37]	GLSL 4.60, More efficient geometry processing and shader execution, more information, no error context, polygon offset clamp, SPIR-V, anisotropic filtering

Summary

OpenGL is a cross-language, cross-platform **application programming interface (API)** for rendering 2D and 3D vector graphics.

The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.



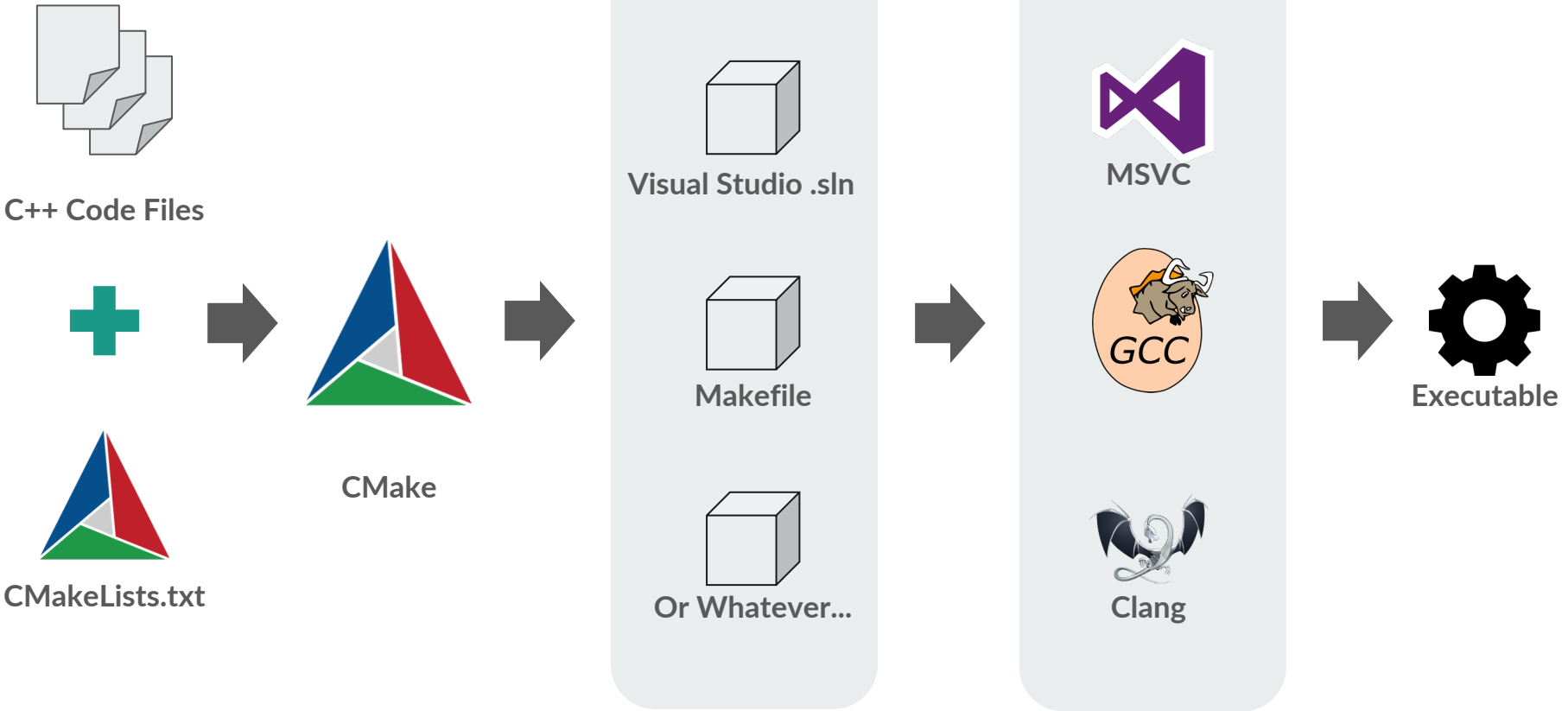
TOOLS WE NEED

CMake

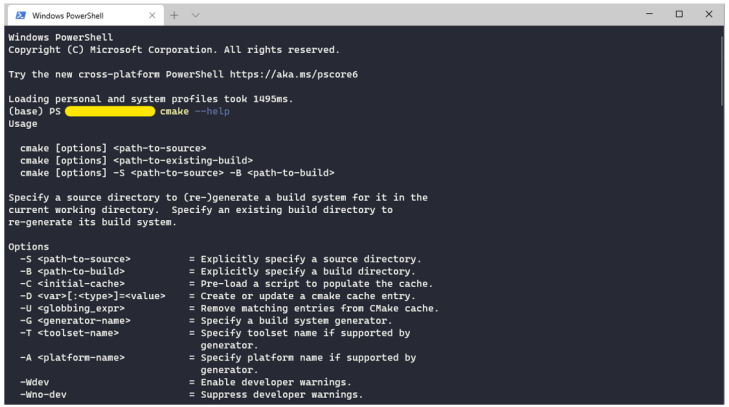
CMake is a **cross-platform** free and open-source software tool for **managing the build process** of software using a **compiler-independent** method.

Link: https://cmake.org/

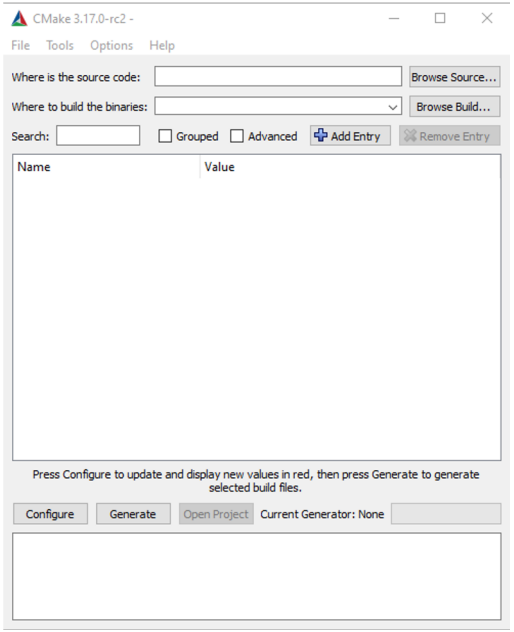
Version: 3.17+


C++ Code Files
CMakeLists.txt
CMake
Visual Studio .sln
Makefile
Or Whatever...
MSVC
GCC
Clang
Executable

Using CMake: Method 1 - Directly (Unpreferred)



OR



Using CMake: Method 2 - Via an IDE

1. CLion (Student/Teacher edition is free).
2. Visual Studio 2017+ (2019 preferred) **[windows only]** with CMake support installed.
3. Visual Studio Code with the following extensions:
    a. C/C++ by Microsoft
    b. CMake by twxs
    c. CMake Tools by Microsoft
4. QtCreator.

Which Compilers can we use?

Basically any compiler with C++17 support. This includes:

- MSVC 19.15 (Comes with Visual Studio 2017 15.8). Newer versions are preferred.
  - Note: For Windows only. Can be found on https://visualstudio.microsoft.com/
- GCC 10+
  - For windows, you can find it on WinLibs.com
  - For linux, it can be installed using the **apt-get** command.
- Clang 5+
  - Can be installed as part of LLVM or alongside GCC from WinLibs.com

What else do we need?

- All the needed libraries will be supplied with lab examples.
- Make sure that your gpu driver is up to date.

Optional (Recommended) Tools

- An extension for the **GLSL** language:
  - Visual Studio: **GLSL language integration** by **Daniel Scherzer.**
  - Visual Studio Code: **Shader languages support for VS Code** by **slevesque.**
  - CLion: **GLSL Support** plugin.
- RenderDoc for debugging OpenGL state and api calls.
- You can find other debugging tools on: https://www.khronos.org/opengl/wiki/Debugging_Tools

CMake Helping Material

- CMake Projects in Visual Studio
- CMake Tools for Visual Studio Code
- CMake Projects in CLion
- CMake Projects in QtCreator
- CMake Tutorial

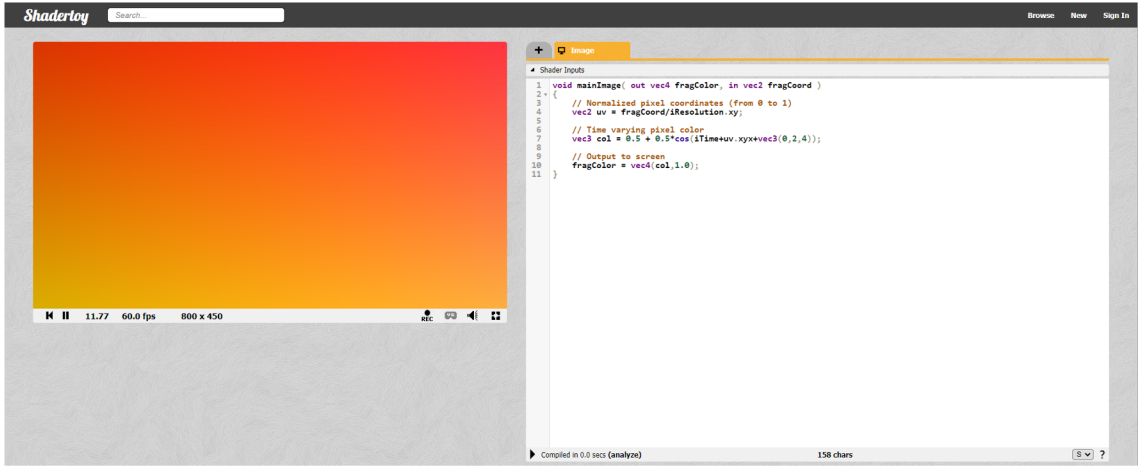
Game engine

Then we will implement what we learned in a simple game-engine and create a game with it.

GPUs and drawing

Shaders are simple programs that describe the traits of either a vertex or a pixel.

Vertex shaders describe the attributes (position, texture coordinates, colors, etc.) of a vertex, while **fragment shaders** describe the traits (color, z-depth and alpha value) of a pixel (or a pixel fragment to be more precise).



Using fragment shader to color the screen.

Shadertoy example...

Thank you