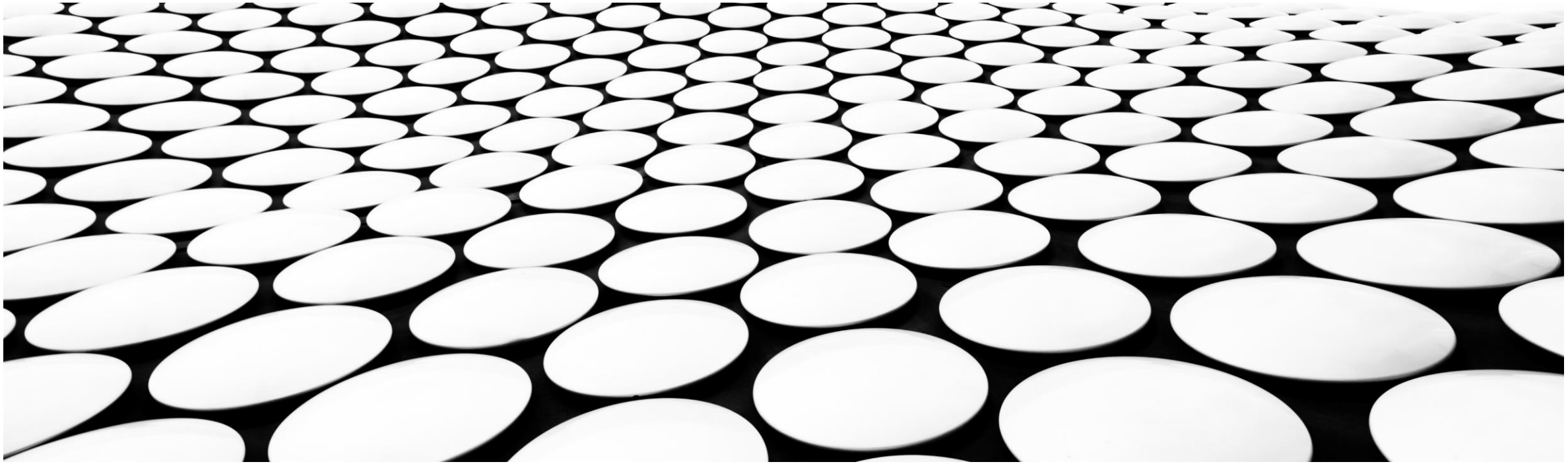
INFO-H-502 – VIRTUAL REALITY EXERCISES 01

ELINE SOETENS – LAURIE VAN BOGAERT

GAUTHIER LAFRUIT – DANIELE BONATTO



IMPORTANT RESSOURCES

- C++ / OpenGL
  - http://www.learnopengl.com/
  - http://ogldev.atspace.co.uk/
- Javascript / WebGL
  - https://webglfundamentals.org/
- Specification
  - https://www.opengl.org/registry/
- Most of the images in the slides and several shaders
  - **Shamelessly token and adapted from all around the web**
- Try to read the slides and work the exercises!

TEACHING METHOD – OPENGL PIPELINE

First session

- Abstract away the pipeline and work only on the fragment shader

Next sessions

- Work with the whole pipeline

3D GRAPHICS API

OpenGL
Cross-platform
Desktop Only

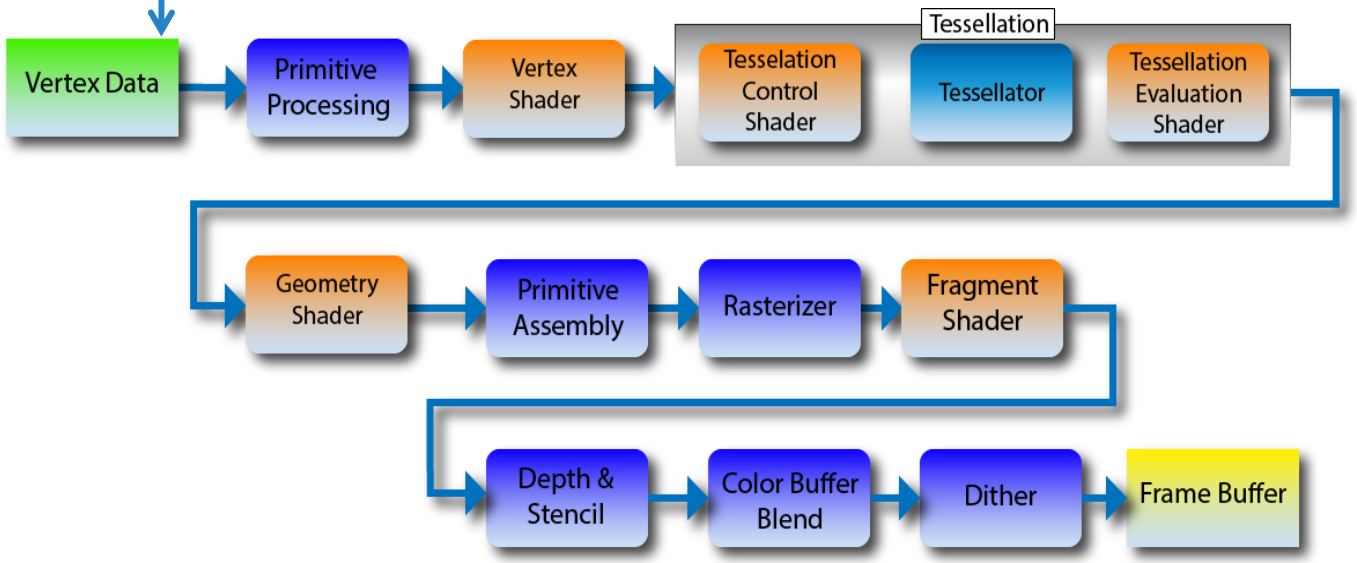
OpenGL ES
Embedded
Devices

WebGL
Web

Vulkan
Cross-platform

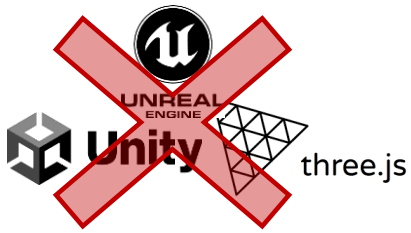
Metal
(Apple / iOS)
https://developer.apple.com/metal/

Microsoft DirectX
(Windows)



Why OpenGL (or WebGL) ?:
- Industry standard
- Cross platform
- Still in use in research
- Not too verbose
- Share similarities with other APIs

Game engines (Unity, Unreal Engine, Godot, ...) or higher-level libraries uses some of these APIs as a back-end
→ But we won't use them for the practical sessions



three.js

Game engines (Unity, Unreal Engine, Godot, …) or higher-level libraries (three.js) uses some of these APIs as a back-end

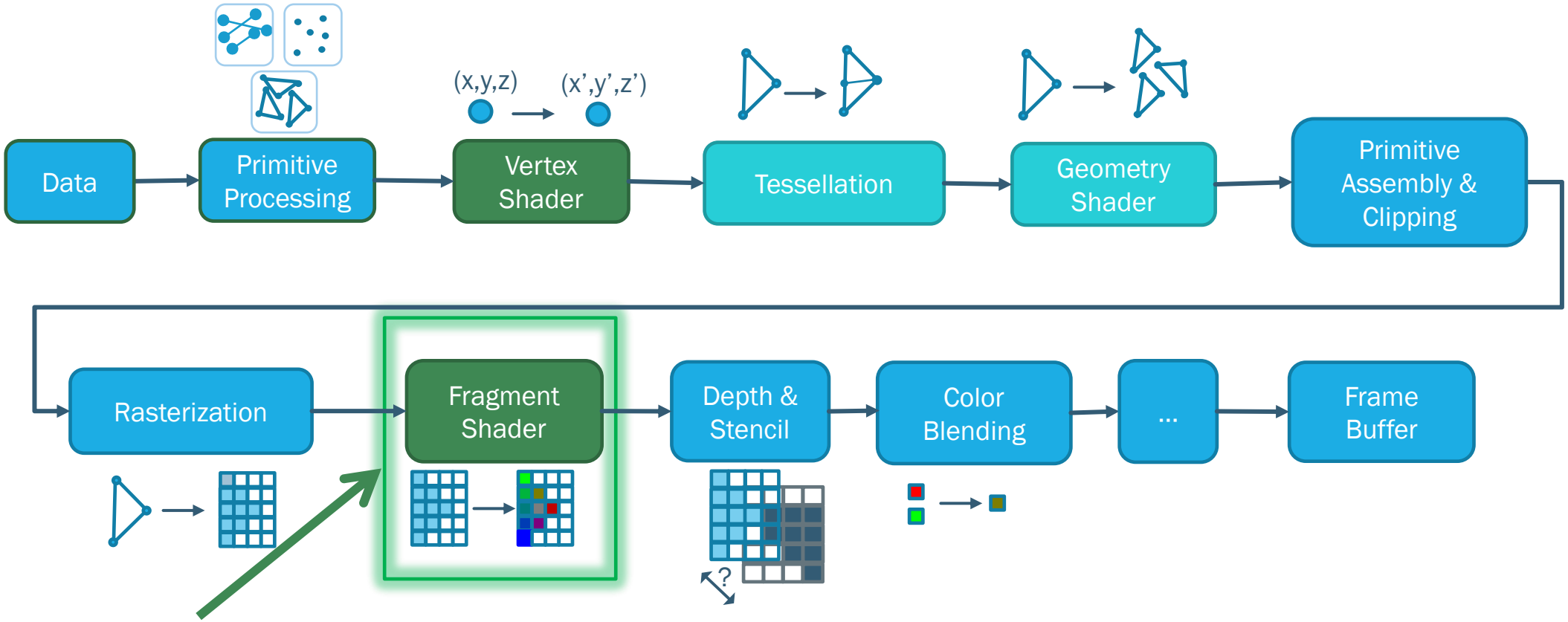
→ But we won't use them for the practical sessions

--> cf. The recent Unity controverse: https://www.theguardian.com/games/2023/sep/13/unity-seeks-to-clarify-new-game-engine-charges-amid-outrage-from-developers

How to learn this stuff?

Main difficulty: You need to understand the whole to make an application

RENDERING PIPELINE



Today: Overview + Fragment Shader

Vertex Normal Others

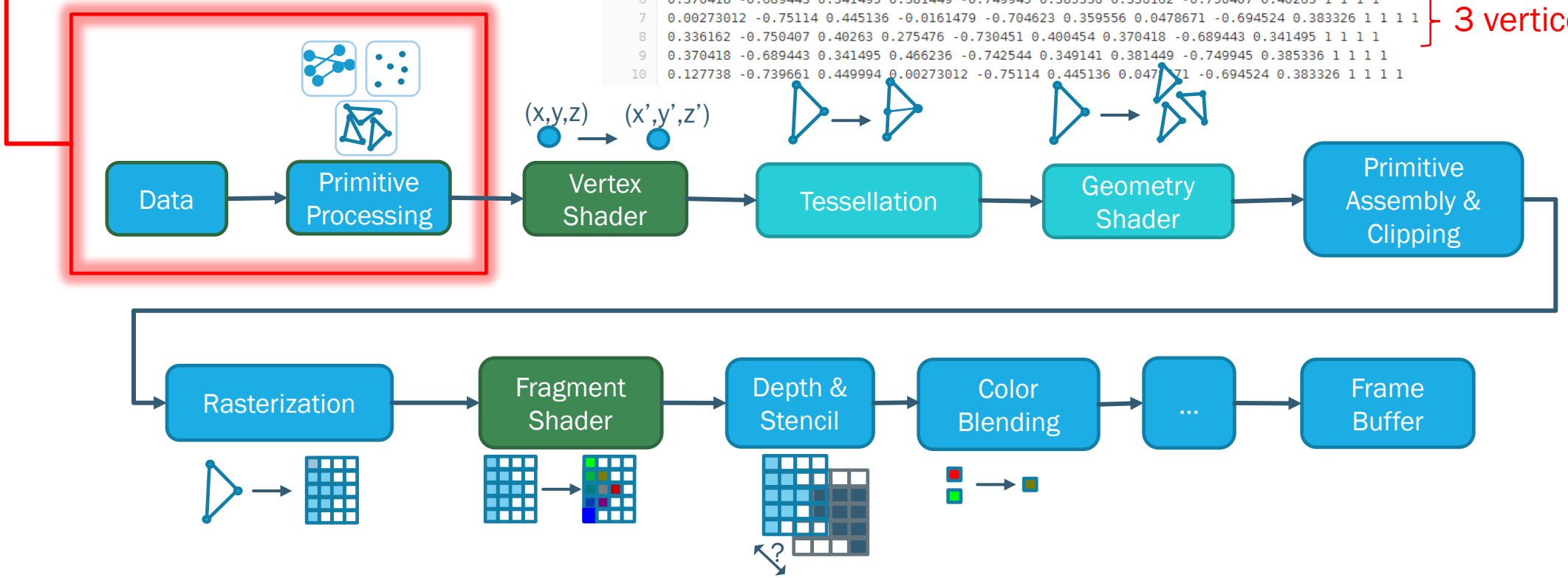
bunny.mesh 447 KB

```
1   1876
2   0.00273012 -0.75114 0.445136 -0.0934739 -0.751342 0.42608 -0.0161479 -0.704623 0.359556 1 1 1 1
3   0.19653 -0.748571 0.439299 0.127738 -0.739661 0.449994 0.190269 -0.690756 0.389317 1 1 1 1
4   0.275476 -0.730451 0.400454 0.269782 -0.69383 0.331669 0.370418 -0.689443 0.341495 1 1 1 1
5   0.370418 -0.689443 0.341495 0.453109 -0.690704 0.301089 0.466236 -0.742544 0.349141 1 1 1 1
6   0.370418 -0.689443 0.341495 0.381449 -0.749945 0.385336 0.336162 -0.750407 0.40263 1 1 1 1
7   0.00273012 -0.75114 0.445136 -0.0161479 -0.704623 0.359556 0.0478671 -0.694524 0.383326 1 1 1 1
8   0.336162 -0.750407 0.40263 0.275476 -0.730451 0.400454 0.370418 -0.689443 0.341495 1 1 1 1
9   0.370418 -0.689443 0.341495 0.466236 -0.742544 0.349141 0.381449 -0.749945 0.385336 1 1 1 1
10  0.127738 -0.739661 0.449994 0.00273012 -0.75114 0.445136 0.047[illegible]71 -0.694524 0.383326 1 1 1 1
```

3 vertices = one triangle

Input data: Vertices

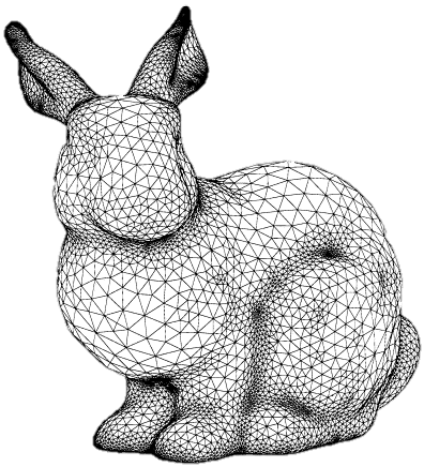
RENDERING PIPELINE



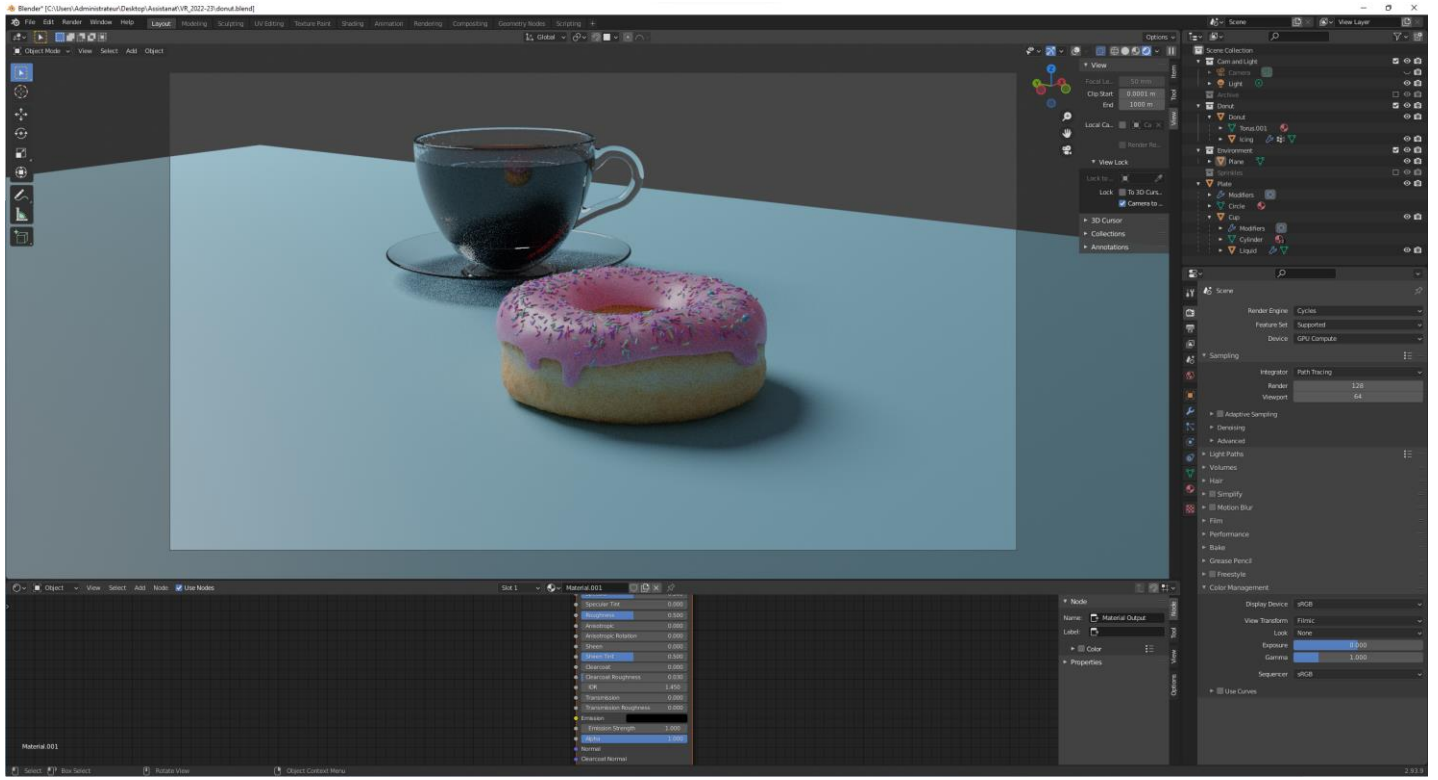
How to generate this data?

RENDERING PIPELINE

Vertices usually comes from 3D softwares



```
MeshVersionFormatted 1
Dimension 3
Vertices
5433
-0.0260146 0.112578 0.0363871 0
-0.0321783 0.174119 -0.00263321 0
-0.080718 0.152855 0.0302446 0
-0.0231294 0.112186 0.0386436 0
0.0164928 0.122293 0.0314234 0
-0.0248365 0.156574 -0.00377469 0
0.0452628 0.0863563 0.0200367 0
-0.071339 0.155715 0.00712639 0
-0.0100758 0.125949 0.0297379 0
-0.0720133 0.154518 0.0303984 0
```



- Free
- Open-source
- Easy to use
- Tutorials!

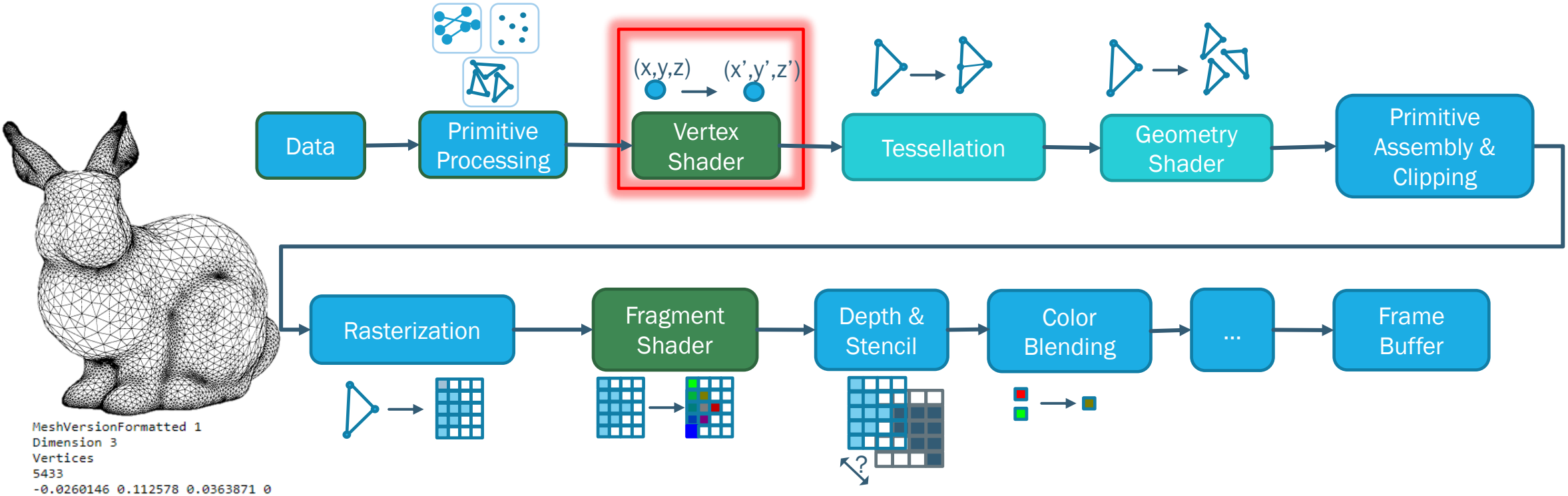
Very extensive donut tutorial :
https://www.youtube.com/playlist?list=PLjEaoINr3zgEq0u2MzVgAaHEBt--xLB6U

RENDERING PIPELINE

Model, Camera, Projection spaces

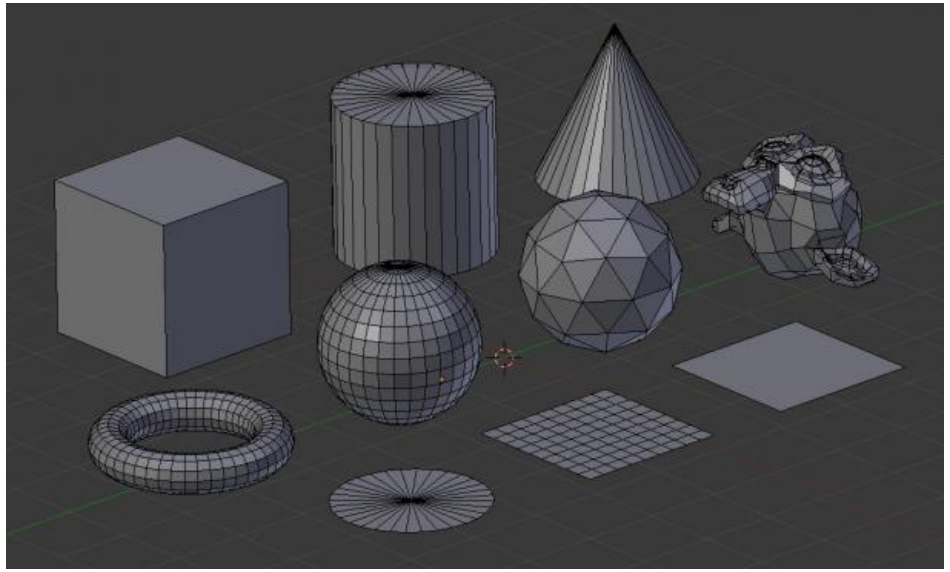
Linear Algebra: Matrix transformations

But why?



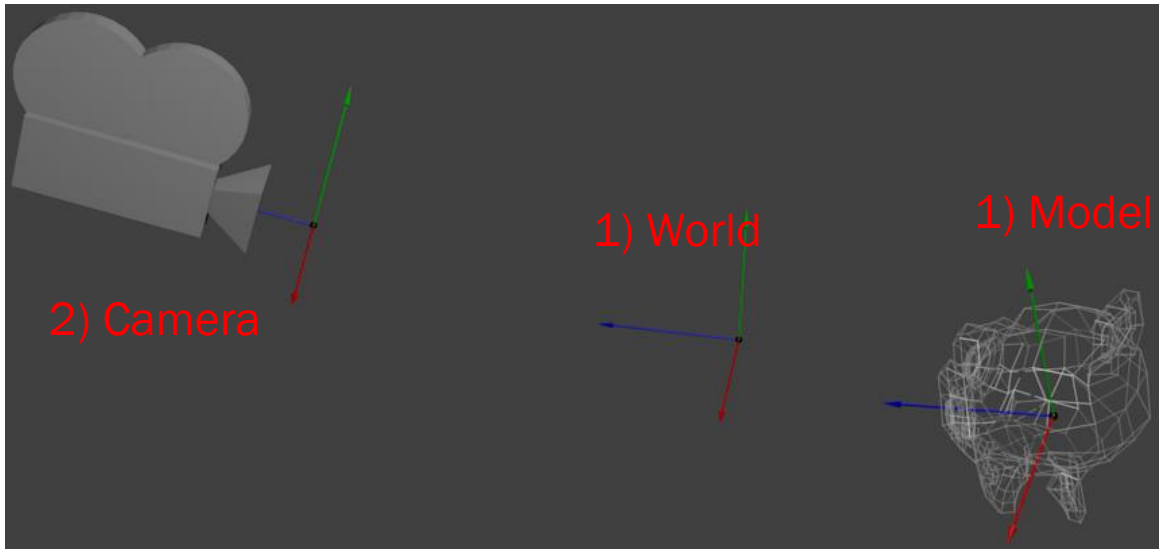

```
MeshVersionFormatted 1
Dimension 3
Vertices
5433
-0.0260146 0.112578 0.0363871 0
-0.0321783 0.174119 -0.00263321 0
-0.080718 0.152855 0.0302446 0
-0.0231294 0.112186 0.0386436 0
0.0164928 0.122293 0.0314234 0
-0.0248365 0.156574 -0.00377469 0
0.0452628 0.0863563 0.0200367 0
-0.071339 0.155715 0.00712639 0
-0.0100758 0.125949 0.0297379 0
-0.0720133 0.154518 0.0303984 0
```

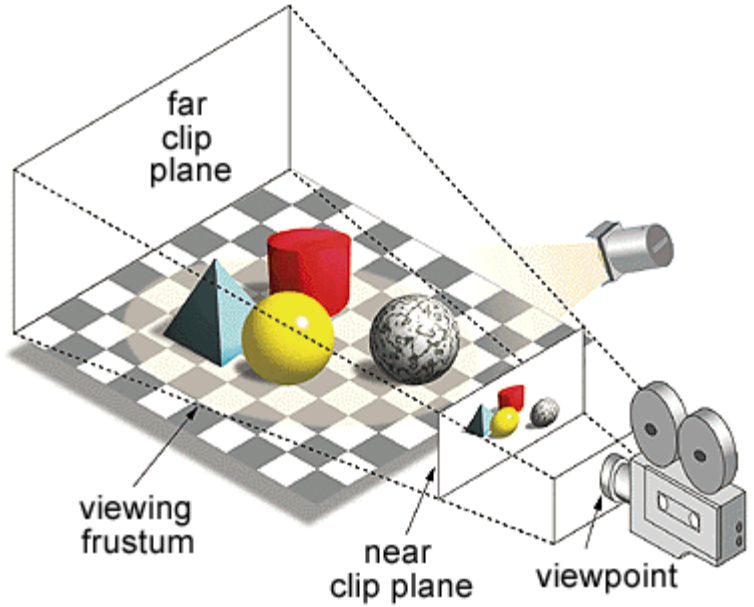
RENDERING PIPELINE



1) We need to move the objects to make a scene

2) We want to look with the camera

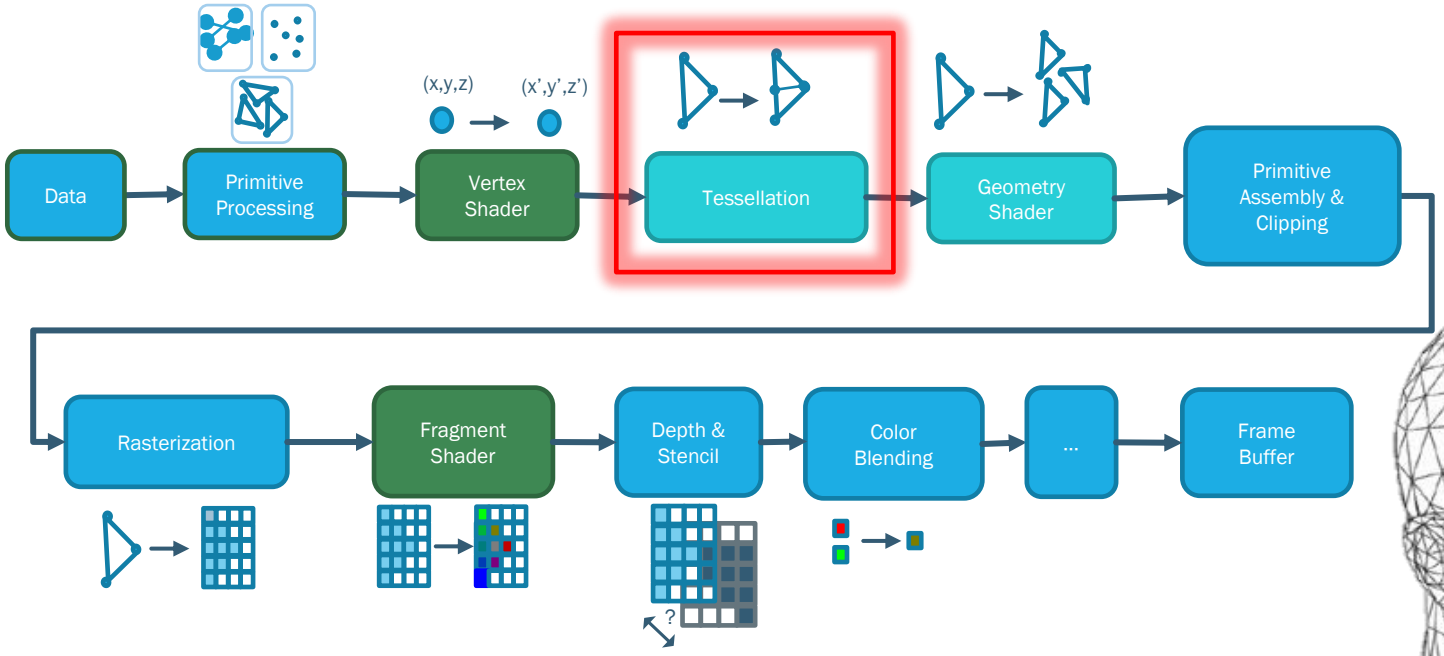


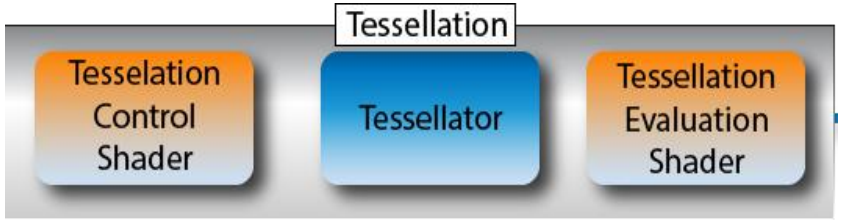


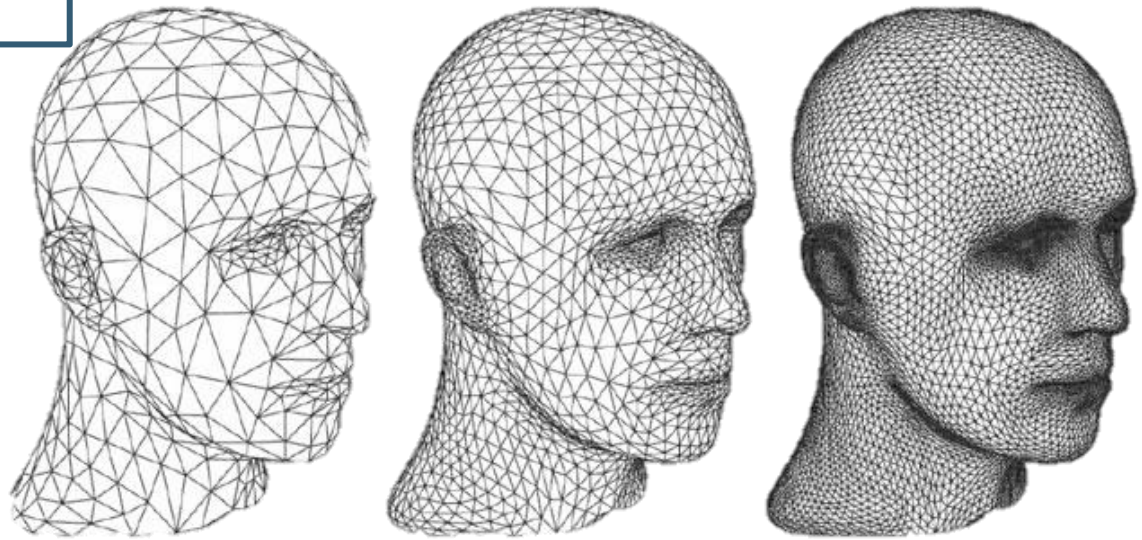
3) We need to add a projection to make them realistic for humans

Break the triangles into
Smaller triangles (primitives)

RENDERING PIPELINE



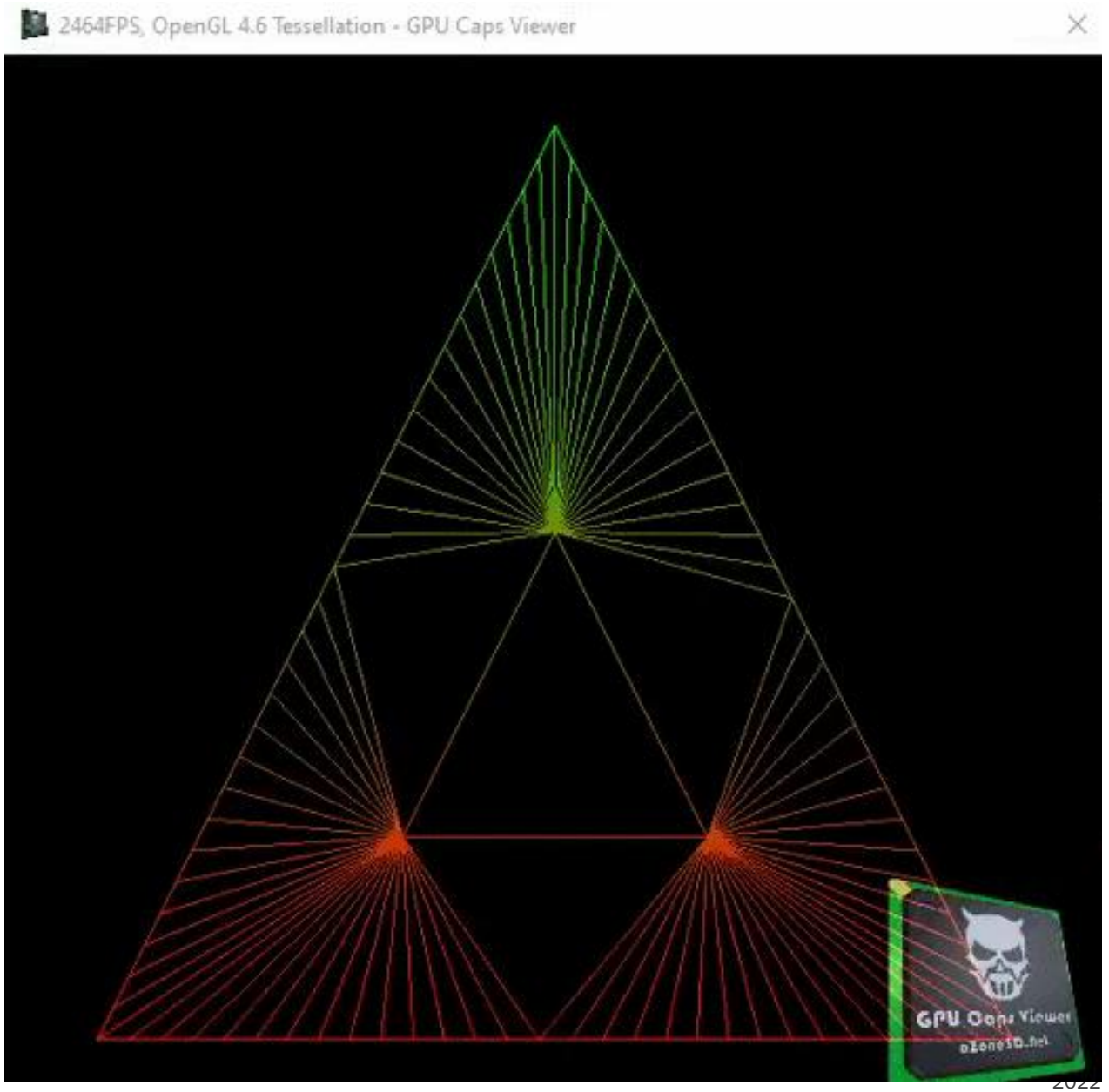




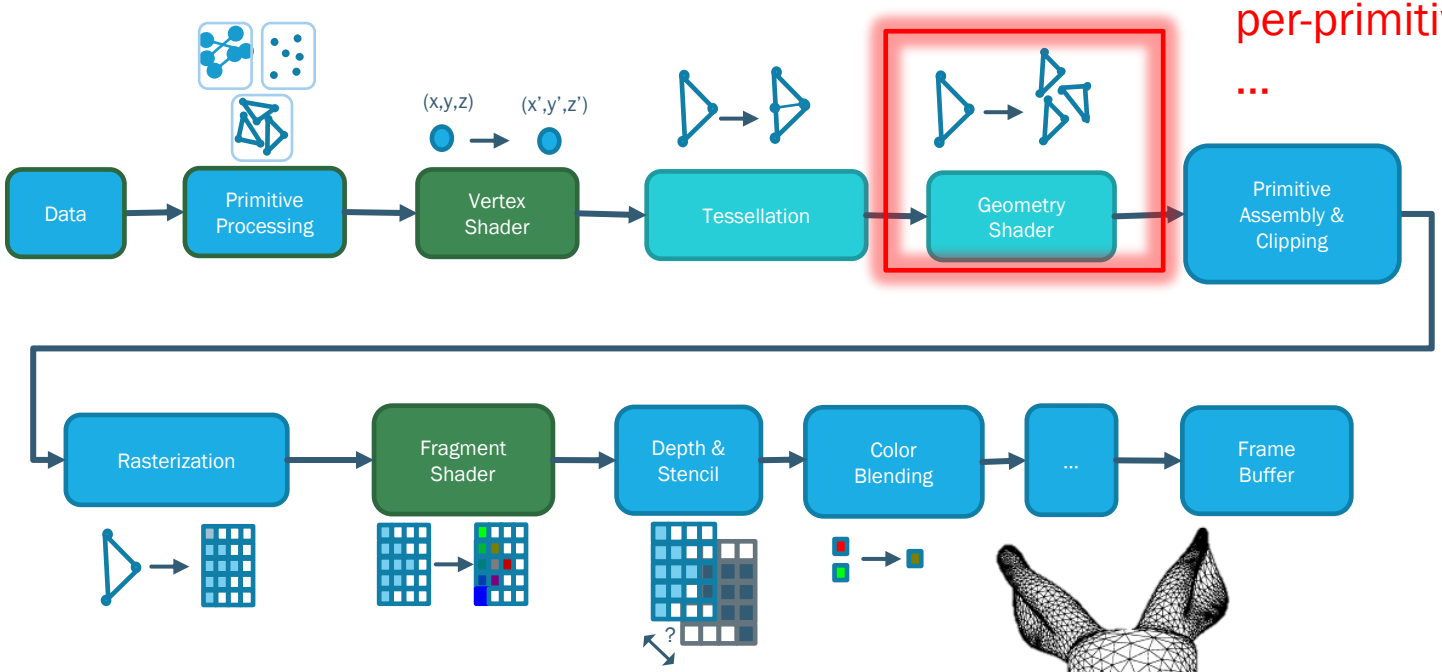
Tessellation

Source: https://www.realtimerendering.com/

RENDERING PIPELINE

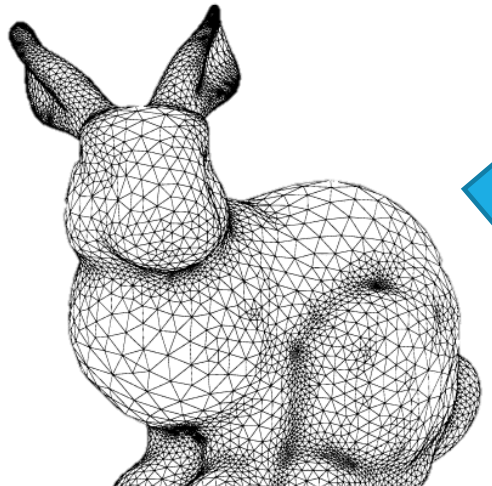


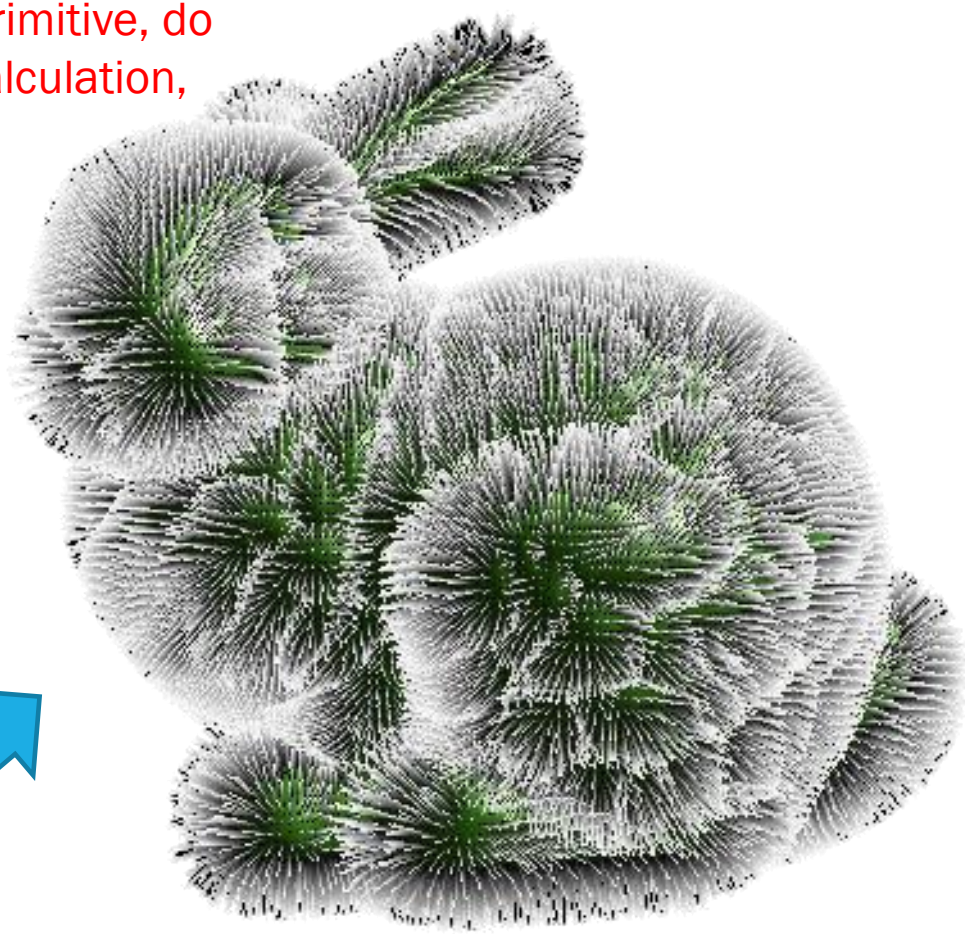
RENDERING PIPELINE



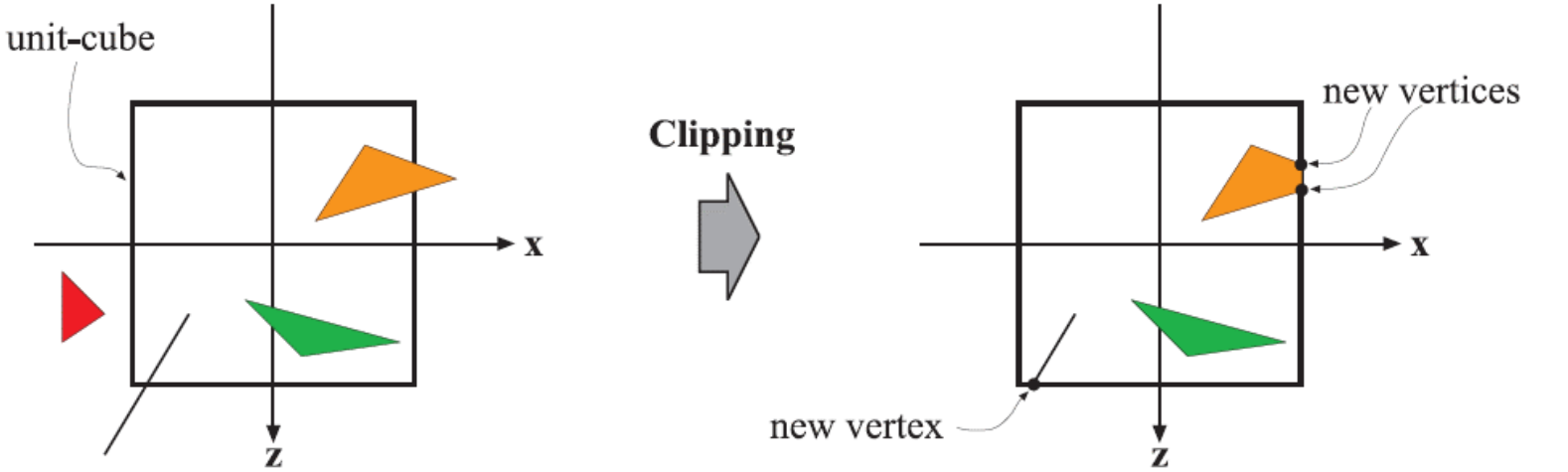
Geometry Shader:

For each **PRIMITIVE**:
create a new primitive, do per-primitive calculation,
...

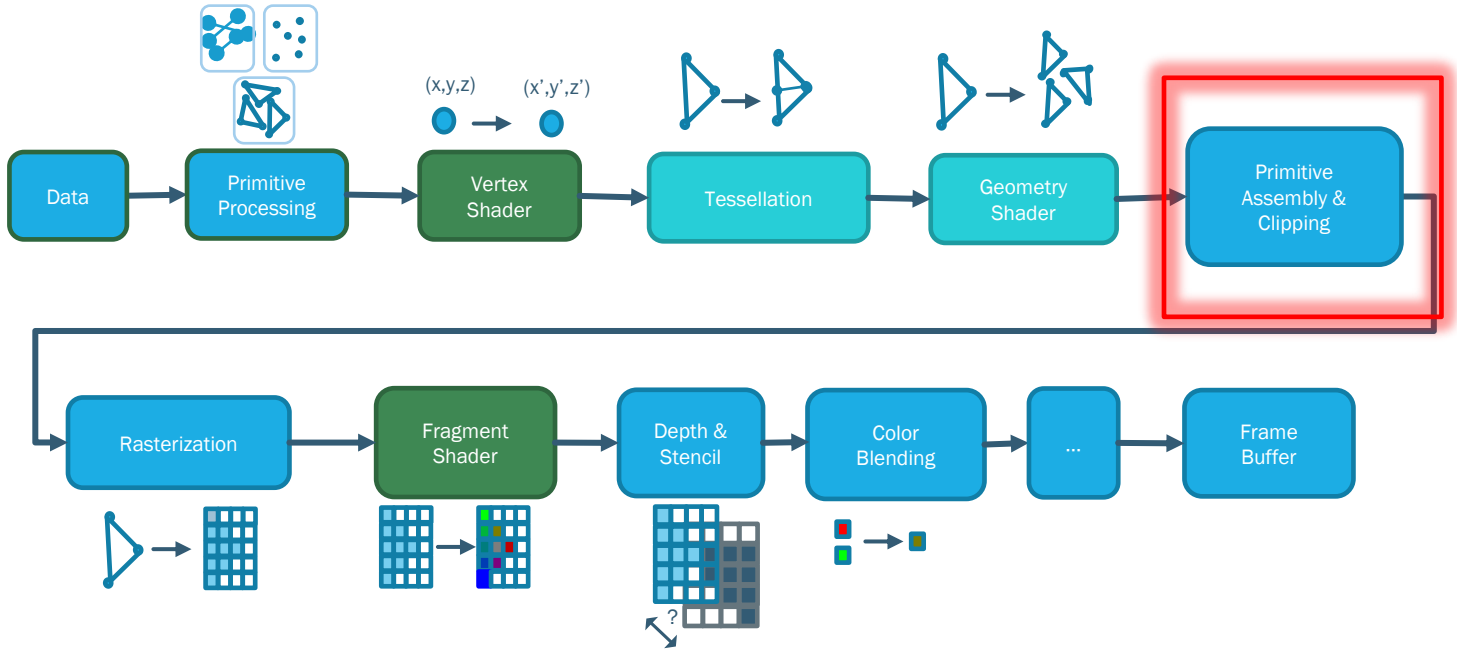




RENDERING PIPELINE



Source: https://www.realtimerendering.com/



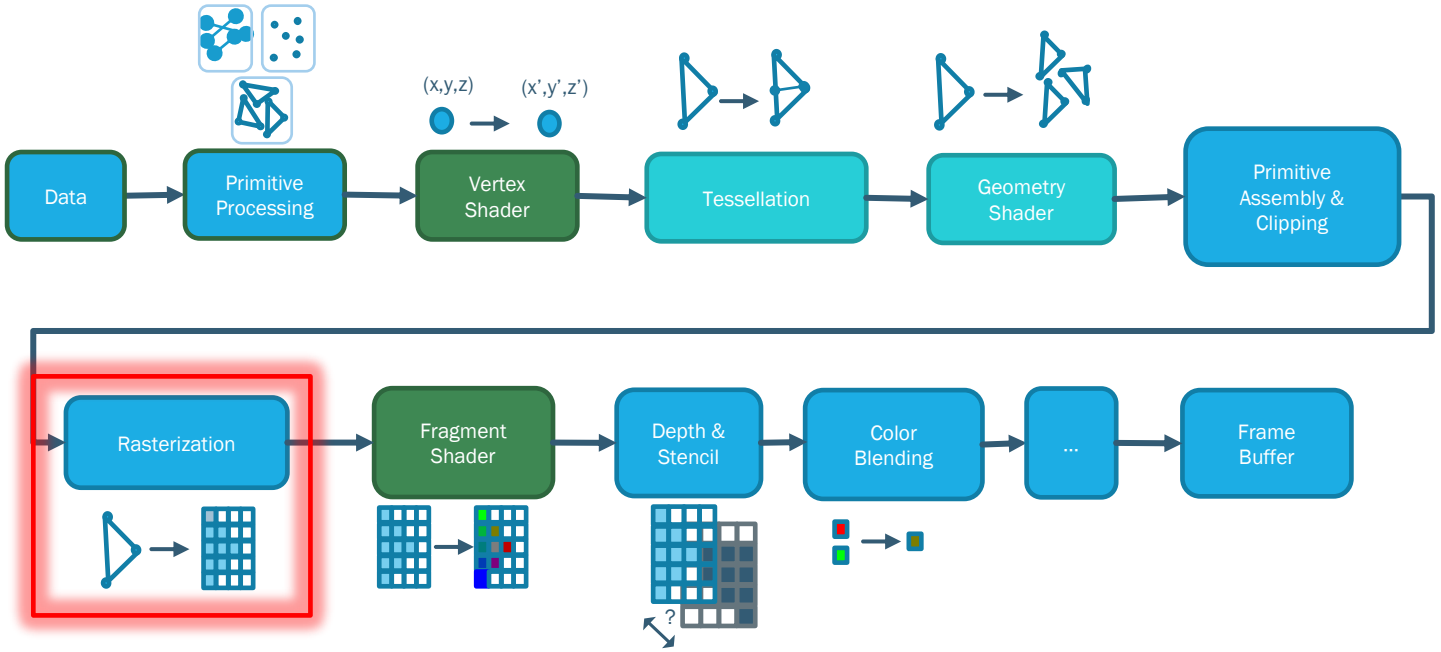
Primitive Assembly

Link all the points!

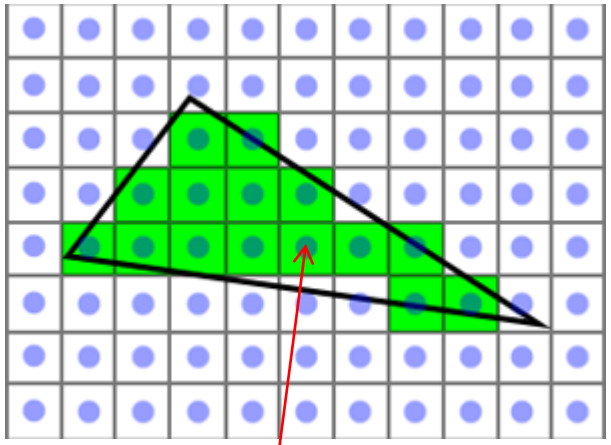
Clipping:
Discard what's not on screen

RENDERING PIPELINE

Triangles (or other primitives)
Are broken down into
Fragments



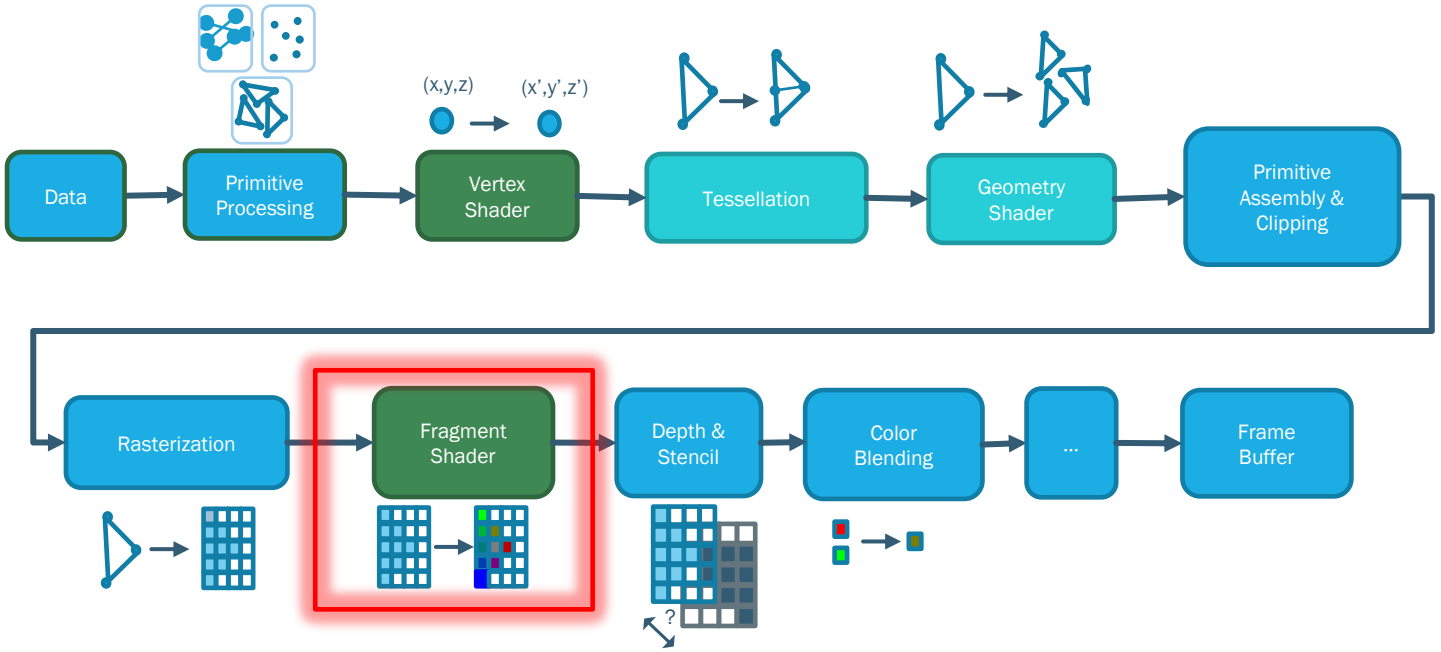
Computer Screen
=
array of pixels

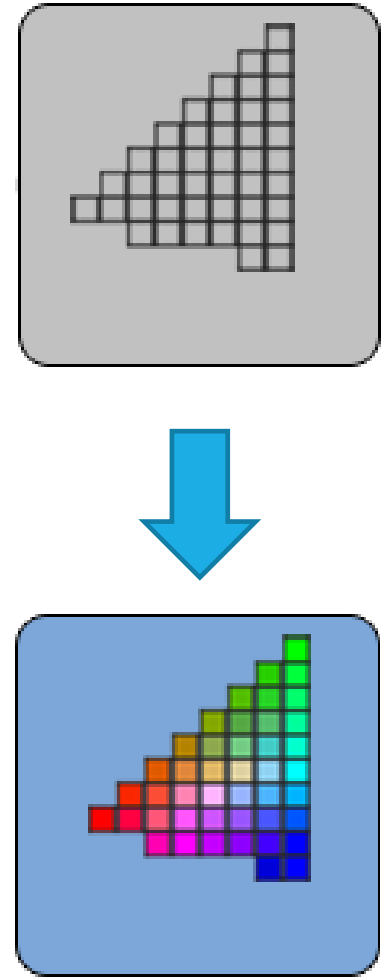


Fragment

EACH fragment is colorized
Independently (GPU threads)

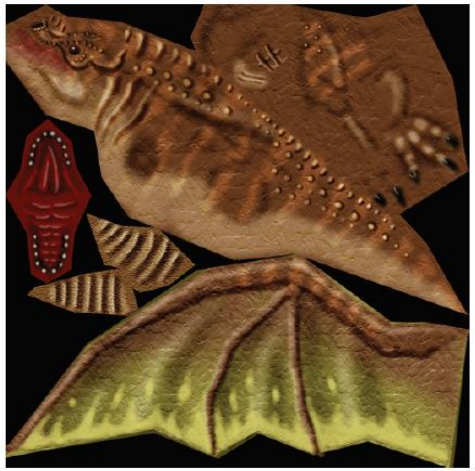
RENDERING PIPELINE



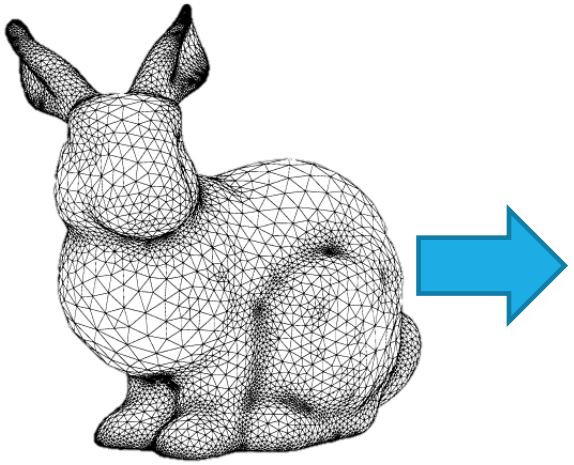


where Magic happens!

RENDERING PIPELINE



Source: https://www.realtimerendering.com/

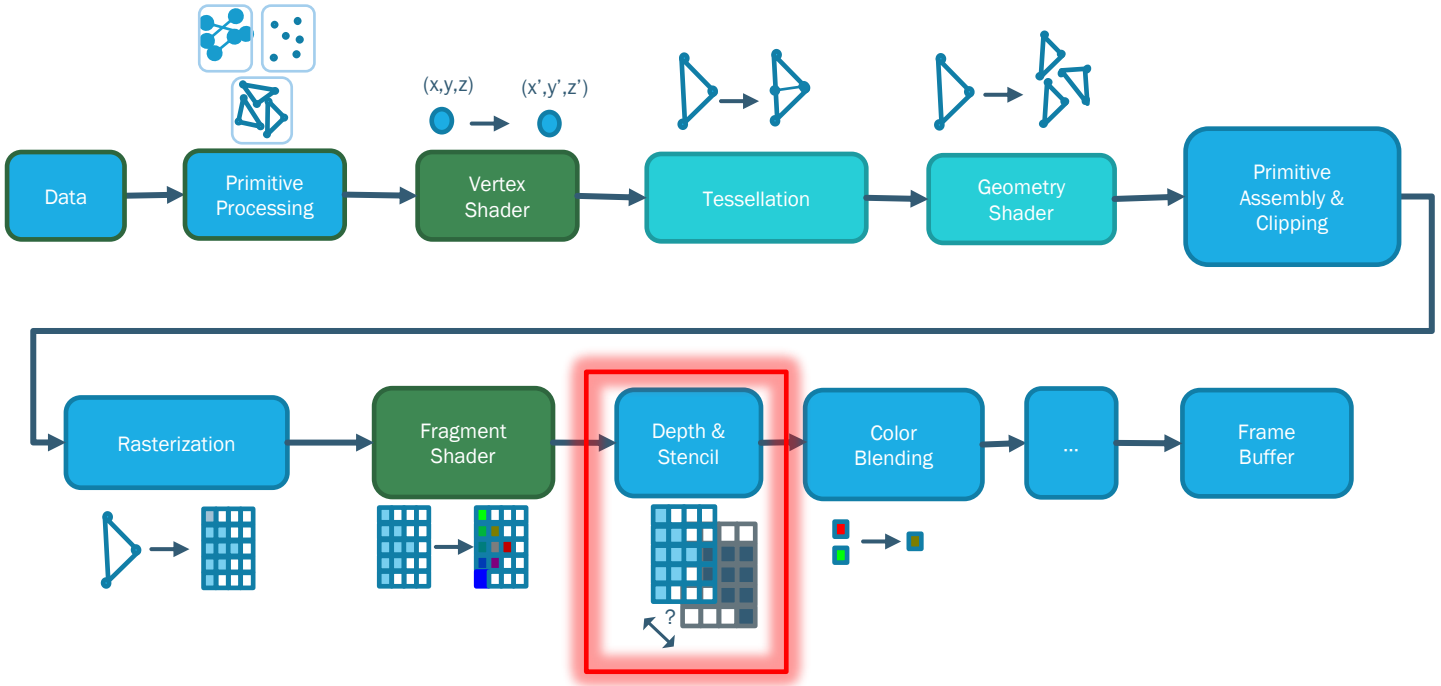


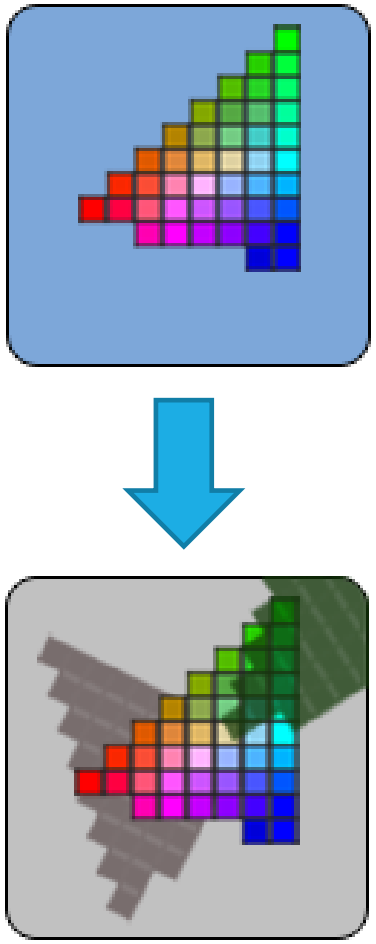
http://photorealizer.blogspot.com/

Not exactly true, you need raytracing for this kind of very high quality results (multiple refractions and subsurface scattering), but we can go very close!

If there were overlapping triangles
We select the one to show

RENDERING PIPELINE

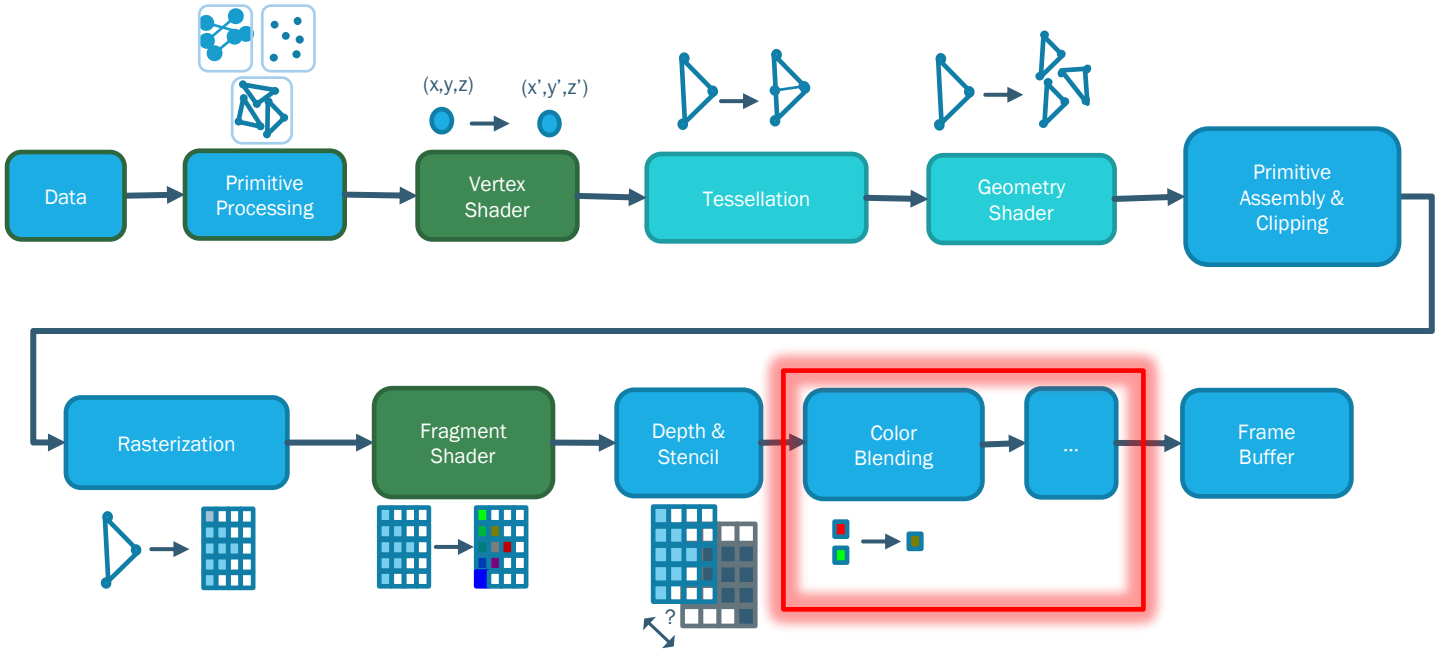




http://photorealizer.blogspot.com/

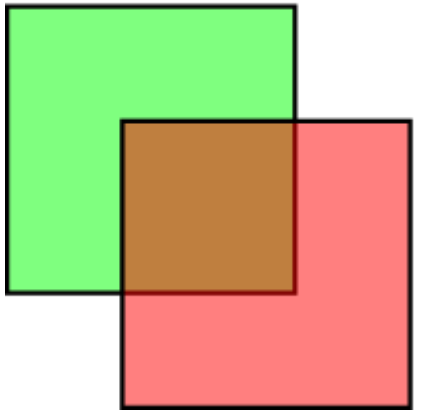
RENDERING PIPELINE

Small post processing
Eg:
- Normalization of the colors
- Alpha channel blending

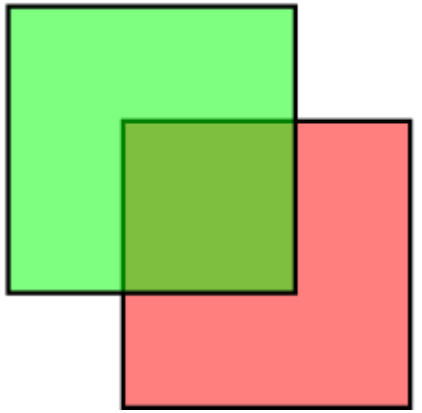


Color = R8G8B8A8

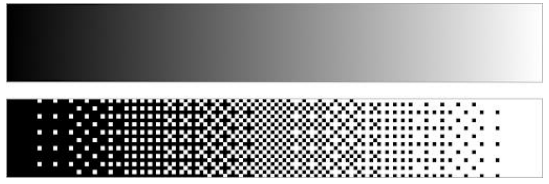
Red on top



Green on top

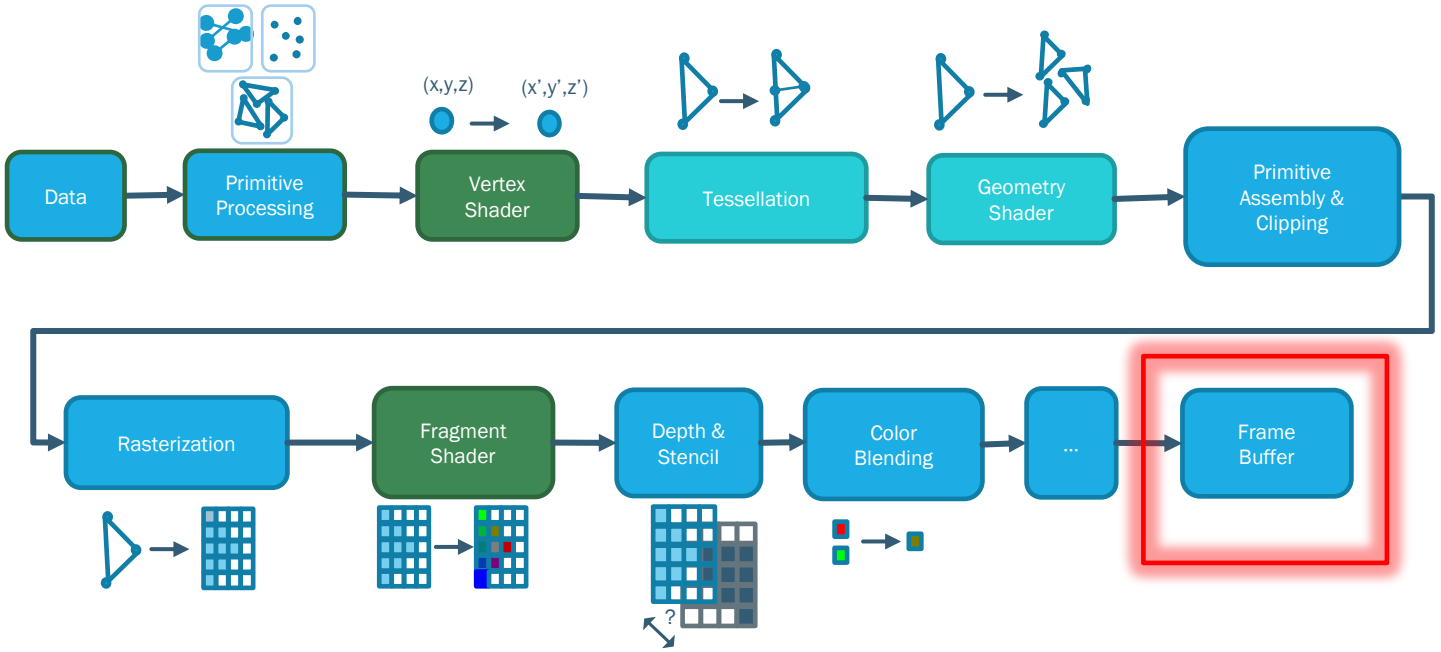


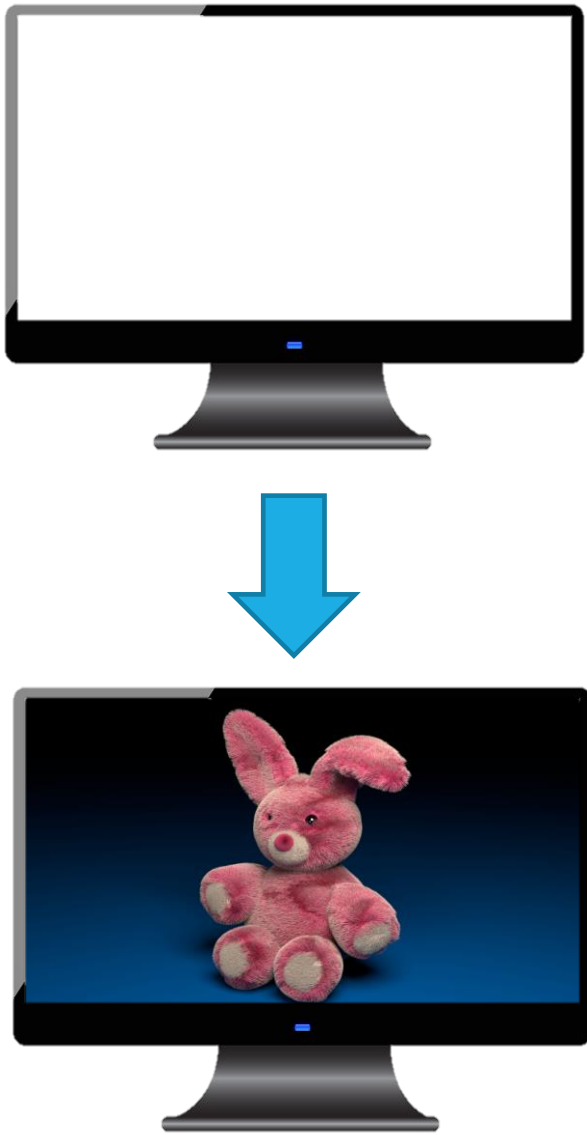
Dither
Patterns to trick the eyes to see more colors



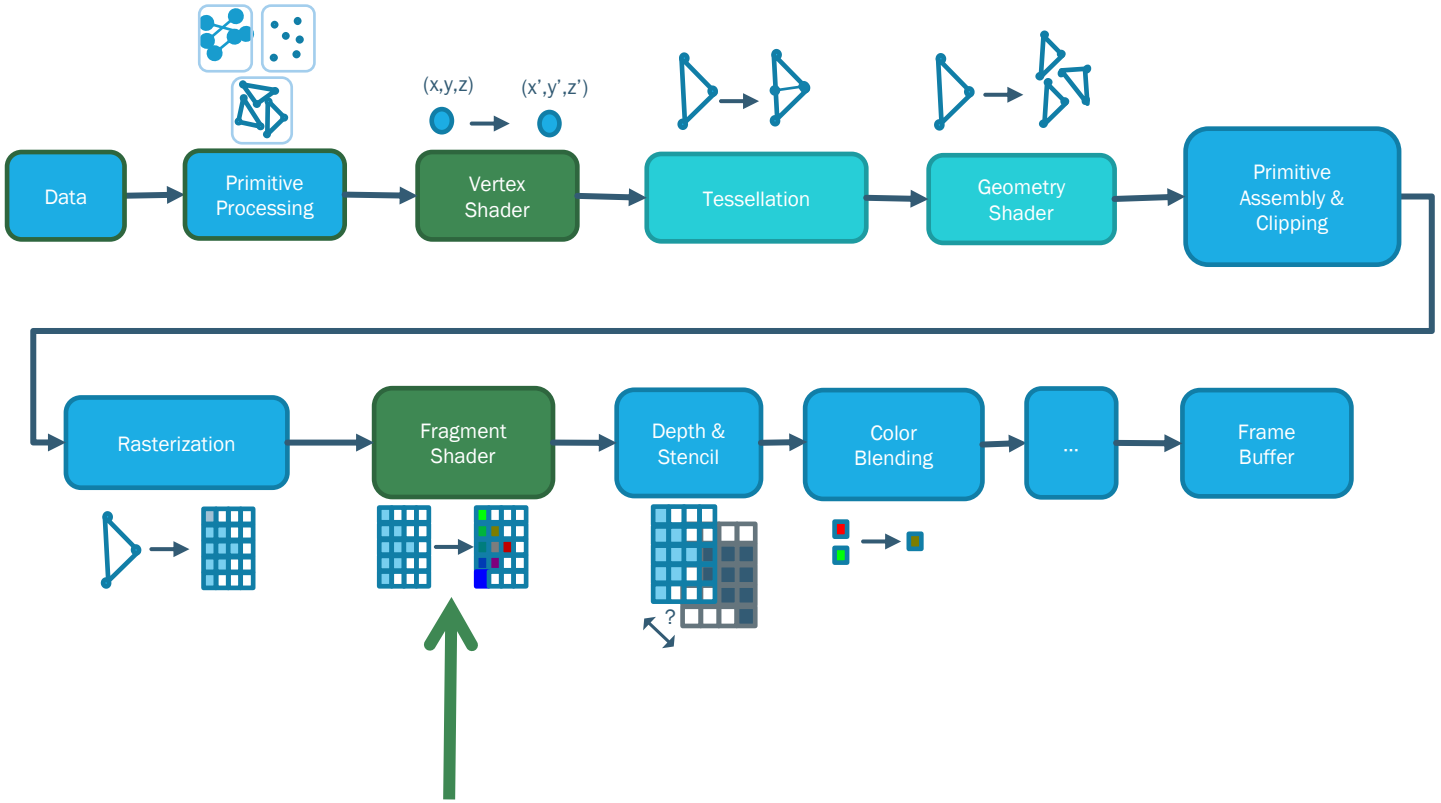
RENDERING PIPELINE

A special “Buffer” (matrix)
Which will be displayed on screen
(or used for effects!)





RENDERING PIPELINE

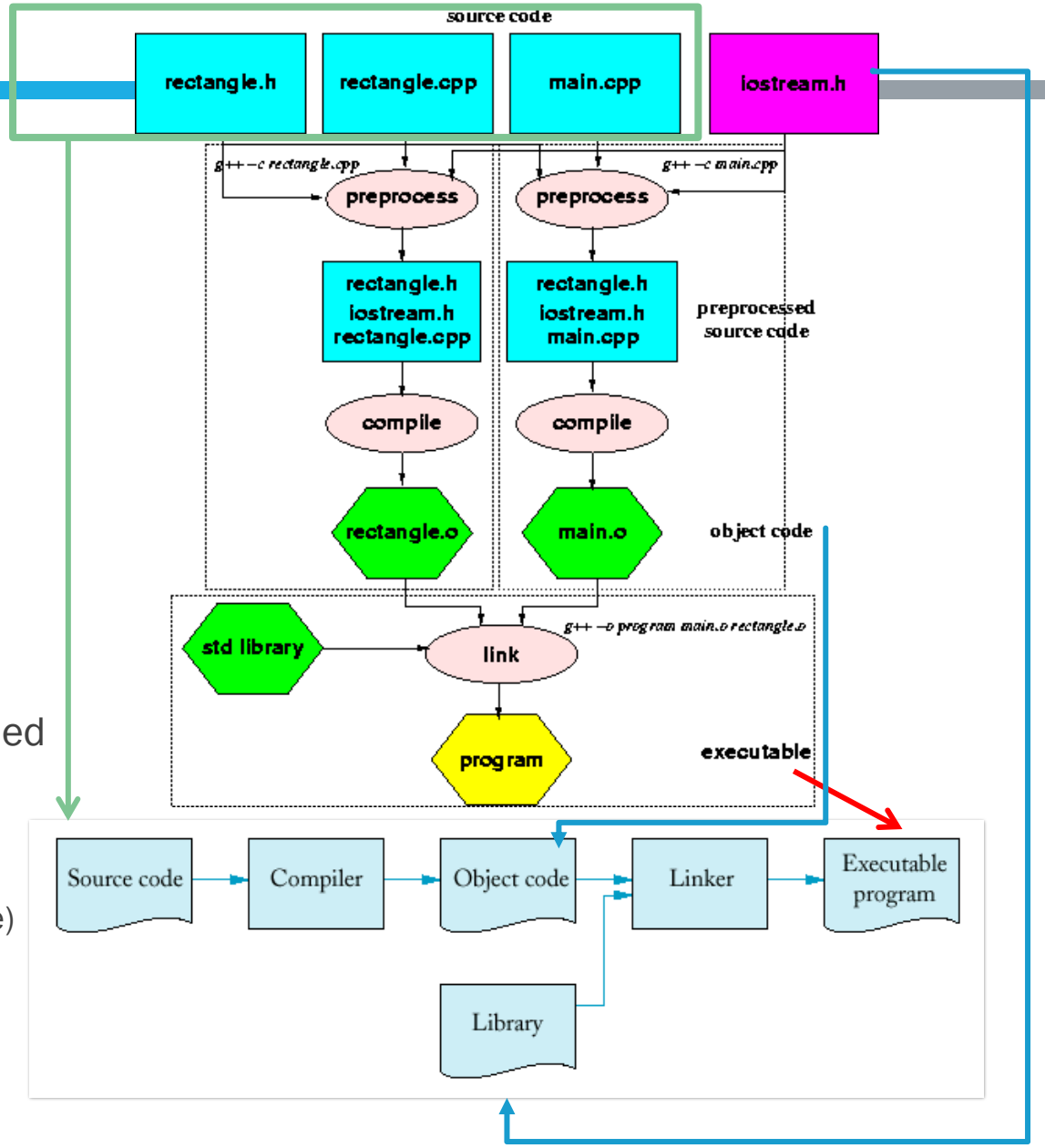


Green stages MUST be programmed
Light blue are optional and unavailable on some framework (WebGL, ...)

A few other programmable in the appendices

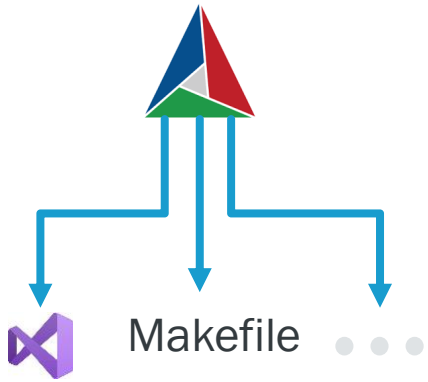
C++ AND C++ PROJECTS

- Object-oriented
- Efficient
- Plenty of C++ tutorials and libraries available
- Compiled language not interpreted like Python or Java
- Before to be able to run, your code need to be transformed into machine code by the compilator
- C++ projects need to be « configured »
  - Need to specify which files you want to compile (the source code)
  - Which libraries you need and where to find them
  - The C++ standard you want, ect …
- C++ is « cross-platform » but …
  - All libraries aren't
  - All build system and compilers aren't

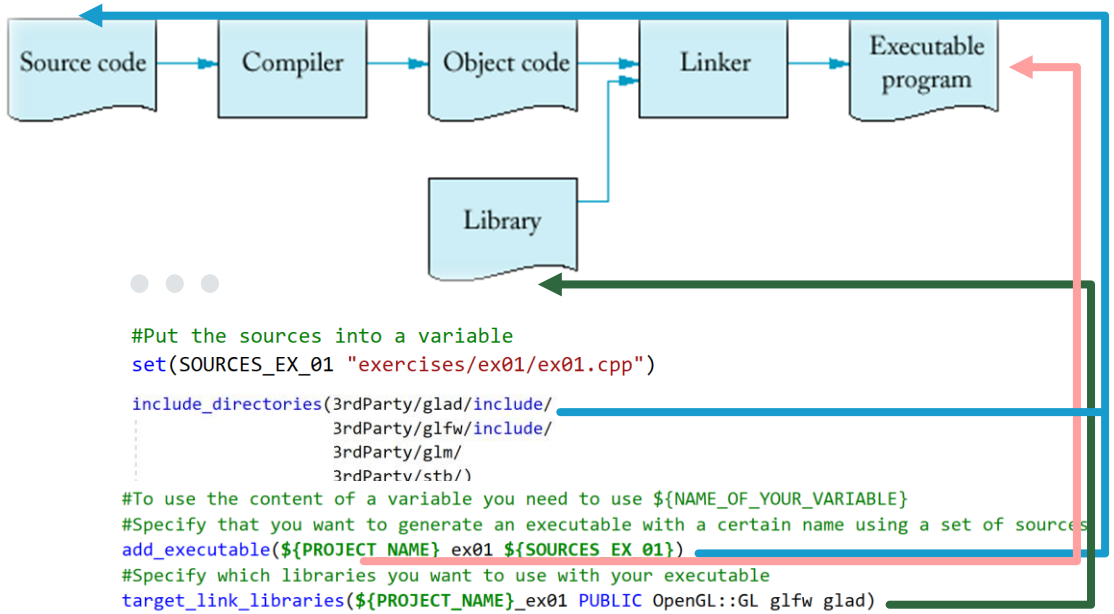


→ How can we make the exercices cross-platform?

CMAKE



- CMake is an open-source cross-platform tool that allow you to generate platform and compiler independent configuration files
- Some IDE directly support it (via extensions)
- Not material for the course but a useful tool to have for future projects
- Not mandatory for the project but recommended
  - Especially if you plan to be on different OS …
- Don’t hesitate to go read the annexes and the cmake files given to understand how it works
- It takes time to be used to it
- Don’t hesitate to ask questions !



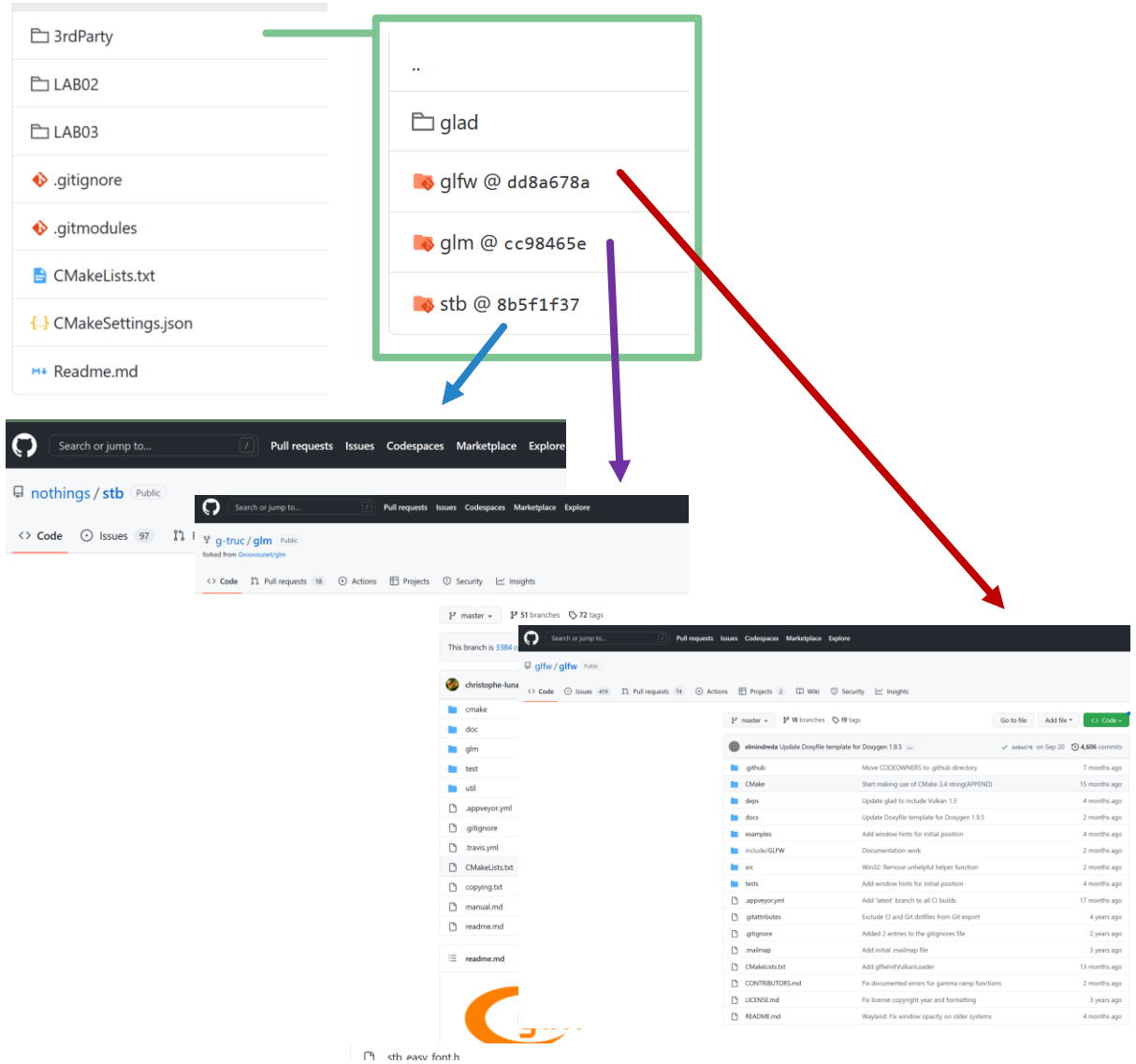

```
#Put the sources into a variable
set(SOURCES_EX_01 "exercises/ex01/ex01.cpp")

include_directories(3rdParty/glad/include/
                    3rdParty/glfw/include/
                    3rdParty/glm/
                    3rdParty/stb/)
#To use the content of a variable you need to use ${NAME_OF_YOUR_VARIABLE}
#Specify that you want to generate an executable with a certain name using a set of sources
add_executable(${PROJECT_NAME}_ex01 ${SOURCES_EX_01})
#Specify which libraries you want to use with your executable
target_link_libraries(${PROJECT_NAME}_ex01 PUBLIC OpenGL::GL glfw glad)
```

→ But often before that you may need to find where your libraries are

→So where can we locate the libraries that we will use today ?

GIT

- Git is a powerful open-source version control system
- Exercises are on gitlab : https://gitlab.com/lisa-vr-course/info-h502_202324
- The usage of git is strongly recommended for the project
- Git submodules → https://git-scm.com/book/en/v2/Git-Tools-Submodules
  - Allow you inside a git to link to another git repository
  - Very useful for project that depend on other projects
- For convenience we use git submodules to get some libraries that we will use later (glfw, glm, stb)
  - Like this you don't have to install these ☺
- When you clone it, use --recusive flag to download the source of the libraries



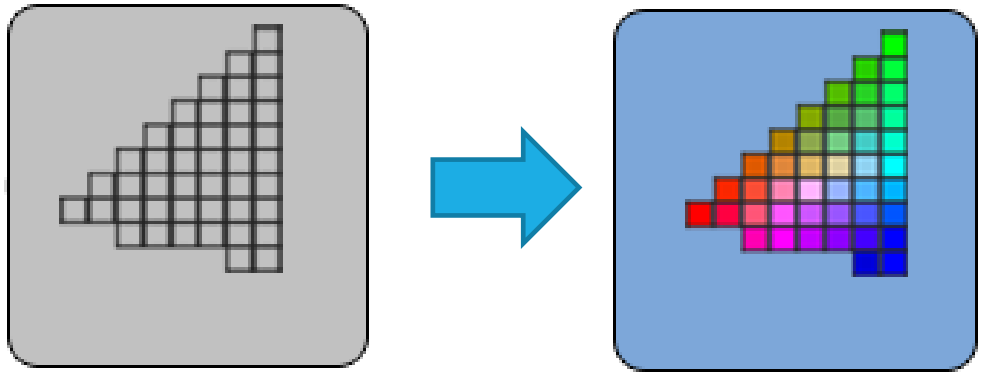
RECOMMENDED IDE

- Possible to use text-bloc or vim to edit your code but …
- Programmers like IDEs (advanced text editors) because of:
  - Automatic code color
  - Syntax completition
  - Documentation integrated (sometimes)
  - Debug
  - Etc..
- We recommend one that directly support cmake (sometime with some add-on) for exercises (and your project):
  - Microsoft visual studio (Windows only) → https://visualstudio.microsoft.com/fr/vs/community/ (See Appendix B)
  - Visual studio code (cross-platform) → https://code.visualstudio.com/ (see Appendix C)
  - Qt creator (cross-platform) → https://www.qt.io/download-open-source (see Appendix D)
  - CLion → https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html
  - …
- You can use other IDE but you may need to generate the configuration file of your project before …

WHAT DO WE DO TODAY ?

- Play with the fun part → Fragment Shader !
- Clone the project & run the first exercise
- Open the file LAB01.frag → this is the code for the fragment shader
- If your IDE is not yet downloaded use : https://www.shadertoy.com/new → online version

HOW TO « PROGRAM » THE PIPELINE?



- We use GLSL a « Shading » language

Main function name depends on the language!

```
void main() {
    vec2 fragCoord = gl_FragCoord.xy;
    vec3 color = vec3(0.6, 0.2, 0.8);
    float alpha = 1.0;

    vec4 pixel = vec4(color, alpha);
    fragColor = pixel;
}
```

This function is run by ALL the pixels on your screen in parallel !
Later when we will work with the vertex shader, it will be run by every fragments of every visible triangle!

GLSL = Open**GL** **S**hading **L**anguage
We use OpenGL
https://www.khronos.org/files/opengl4-quick-reference-card.pdf
The online version, Shadertoy uses WebGL :
https://www.khronos.org/files/webgl20-reference-guide.pdf
Few differences between the two shading languages

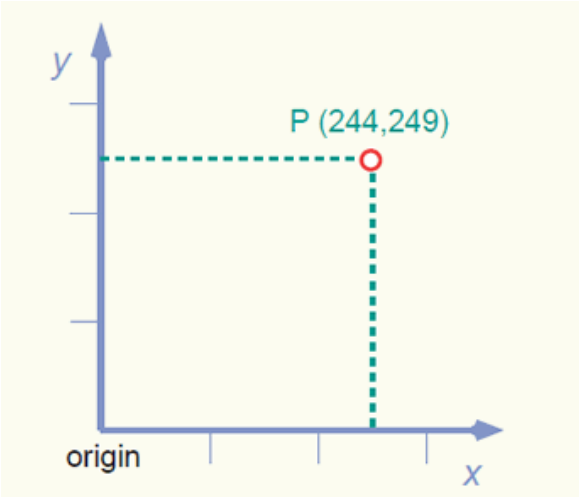
HOW TO « PROGRAM » THE PIPELINE?

- We use GLSL a « Shading » language

We can acces the « Fragment Coordinate »
= which pixel we are ON the window

```
void main() {
    vec2 fragCoord = gl_FragCoord.xy;
    vec3 color = vec3(0.6, 0.2, 0.8);
    float alpha = 1.0;

    vec4 pixel = vec4(color, alpha);
    fragColor = pixel;
}
```

! Coordinate center at the
LEFT BOTTOM corner !

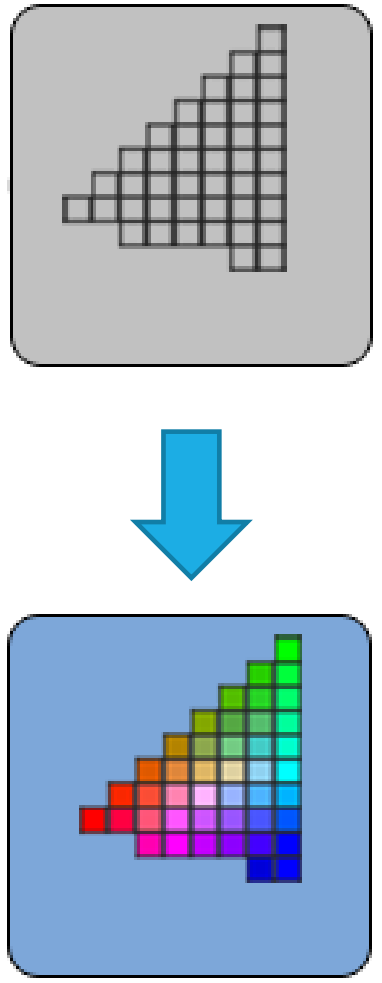


HOW TO « PROGRAM » THE PIPELINE?

- We use GLSL a « Shading » language

```
#version 330 core
precision mediump float;
uniform vec2 iResolution;
uniform float iTime;
uniform vec2 iMouse;

out vec4 fragColor;

void main() {
    vec2 fragCoord = gl_FragCoord.xy;
    vec3 color = vec3(0.6, 0.2, 0.8);
    float alpha = 1.0;

    vec4 pixel = vec4(color, alpha);
    fragColor = pixel;
}
```

We always output the color



HOW TO « PROGRAM » THE PIPELINE?

- We use GLSL a « Shading » language

```
#version 330 core
precision mediump float;
uniform vec2 iResolution;
uniform float iTime;
uniform vec2 iMouse;

out vec4 fragColor;

void main() {
    vec2 fragCoord = gl_FragCoord.xy;
    vec3 color = vec3(0.6, 0.2, 0.8);
    float alpha = 1.0;

    vec4 pixel = vec4(color, alpha);
    fragColor = pixel;
}
```

The color is a RGB variable

Usage:

color.x = color.r = 0.0

color.y = color.g = 1.0

color.z = color.b = 1.0

pixel.w = 1.0

HOW TO « PROGRAM » THE PIPELINE?

- We use GLSL a « Shading » language

```
#version 330 core
precision mediump float;
uniform vec2 iResolution;
uniform float iTime;
uniform vec2 iMouse;

out vec4 fragColor;

void main() {
    vec2 fragCoord = gl_FragCoord.xy;
    vec3 color = vec3(0.6, 0.2, 0.8);
    float alpha = 1.0;

    vec4 pixel = vec4(color, alpha);
    fragColor = pixel;
}
```

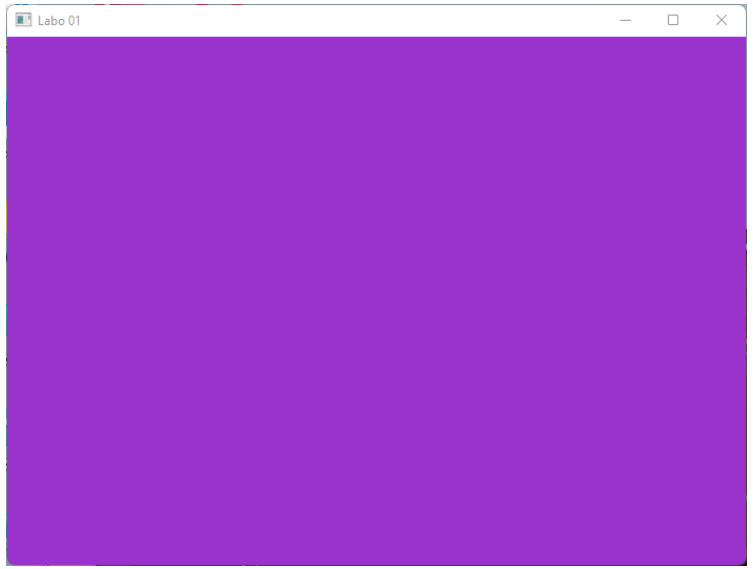
The output color is always
In 4D with the alpha channel

HOW TO « PROGRAM » THE PIPELINE?

- We use GLSL a « Shading » language
- The « programmable » parts of the pipeline use « shaders » programs to describe what they do

```
void main() {
    vec2 fragCoord = gl_FragCoord.xy;
    vec3 color = vec3(0.6, 0.2, 0.8);
    float alpha = 1.0;

    vec4 pixel = vec4(color, alpha);
    fragColor = pixel;
}
```

We finally output the color



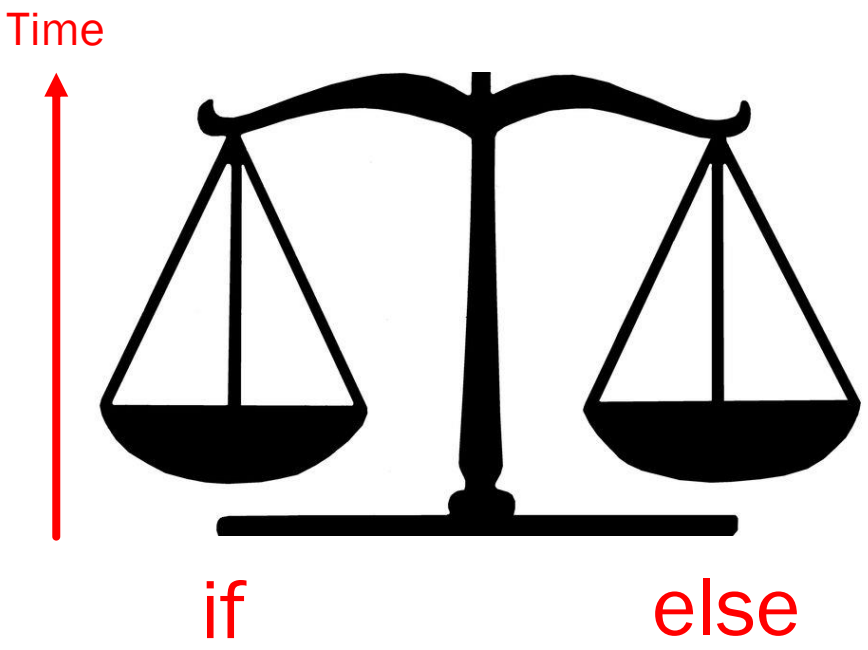
ARE SHADERS POWERFUL?

Done with Shadertoy !

(See appendix A)

IMPORTANT KEYWORDS

- fragCoord:
  - The « fragCoord » variable say at which pixel we are on screen
  - GPU work in parallel, we cannot use values of other pixels in one fragment!
- iResolution
  - xy variable (width, height) of the screen
- Program structure
  - Conditions (if, else, etc.)
  - Loops (for)
- Math functions
  - All classics: sin, cos, mod, …



Danger here!
Use them with balance
All threads should terminate at approximate the same time

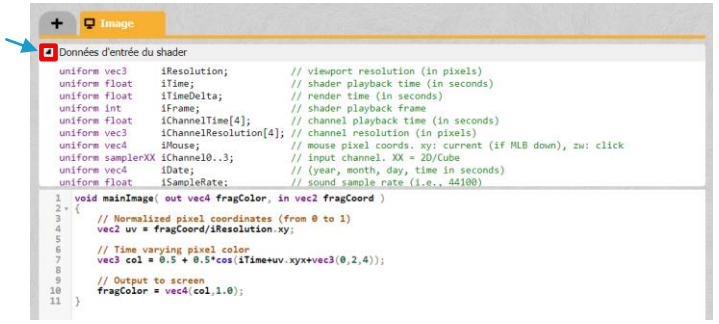
OTHER IMPORTANT FACTS

- We cannot print in the terminal! GPUs would need to transfer the text to the CPU which is extremely expensive
- GPU play extremely nice with float values! Cast into int only if really needed (casting: int(3.5) = 3))
- GPU need to render FAST, all the operations have to be of the same type $*: float \times float \rightarrow float$
  (100*r.x) does not work while (100.0*r.x) works!
- All the accessible variables are at the top of the shader!
- Don't feel limited by the content of the slides
  Internet has all the GLSL functions descriptions!

https://registry.khronos.org/OpenGL-Refpages/gl4/index.php

```
precision mediump float;
uniform vec2 iResolution;
uniform float iTime;
uniform vec2 iMouse;

out vec4 fragColor;
```

For people using shadertoy : accessible variable are also at the top of the shader

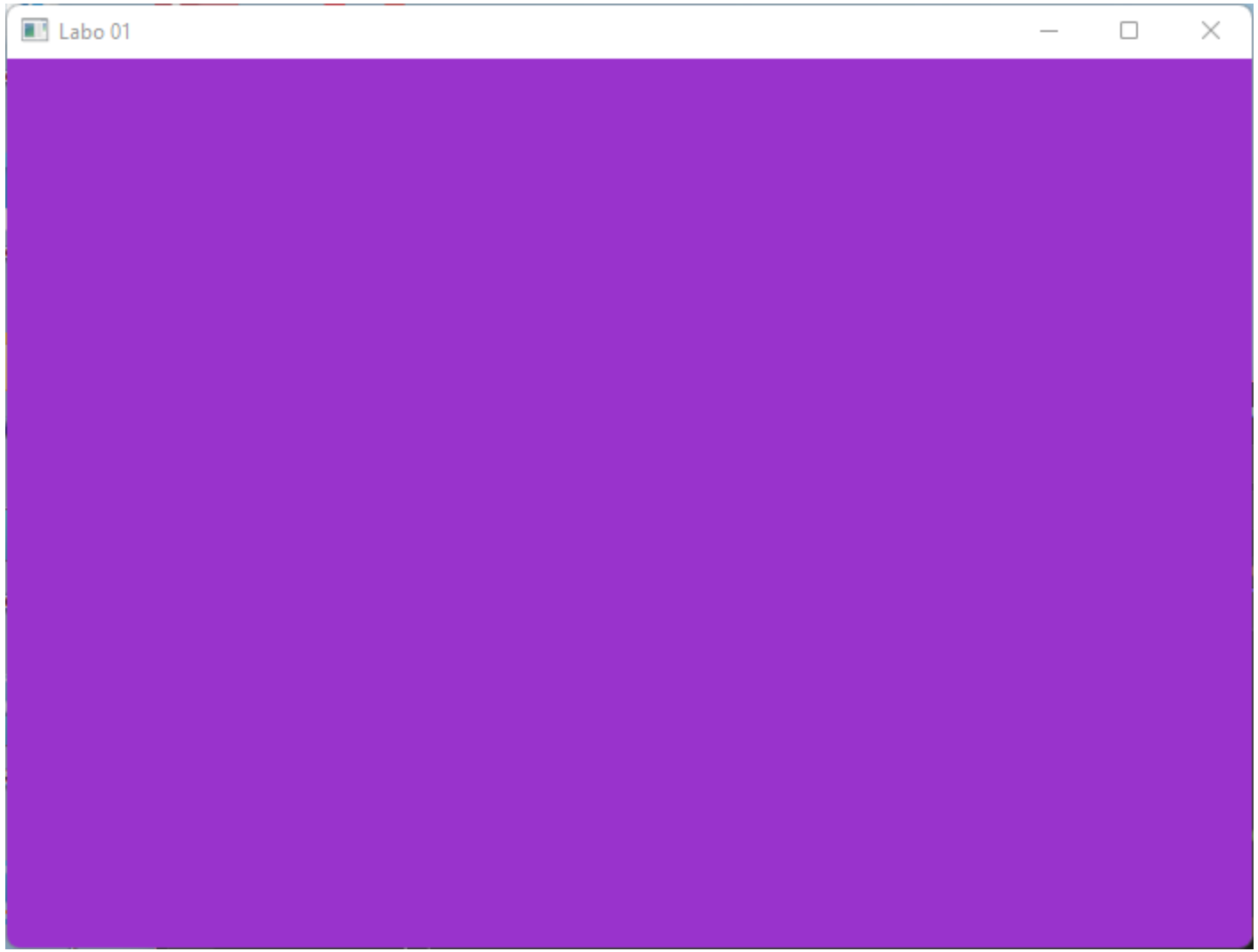


LAST REMARKS

- The solutions for the exercises are in the solution folder
- BUT
  - Look at them only if you have really tried hard !
  - OpenGL require practice!
- If the exercises are too difficult for 4h, we will continue them at the next session, take your time to understand and master every concept.
  You need to be fluent with the content of this exercise session for the project(s)

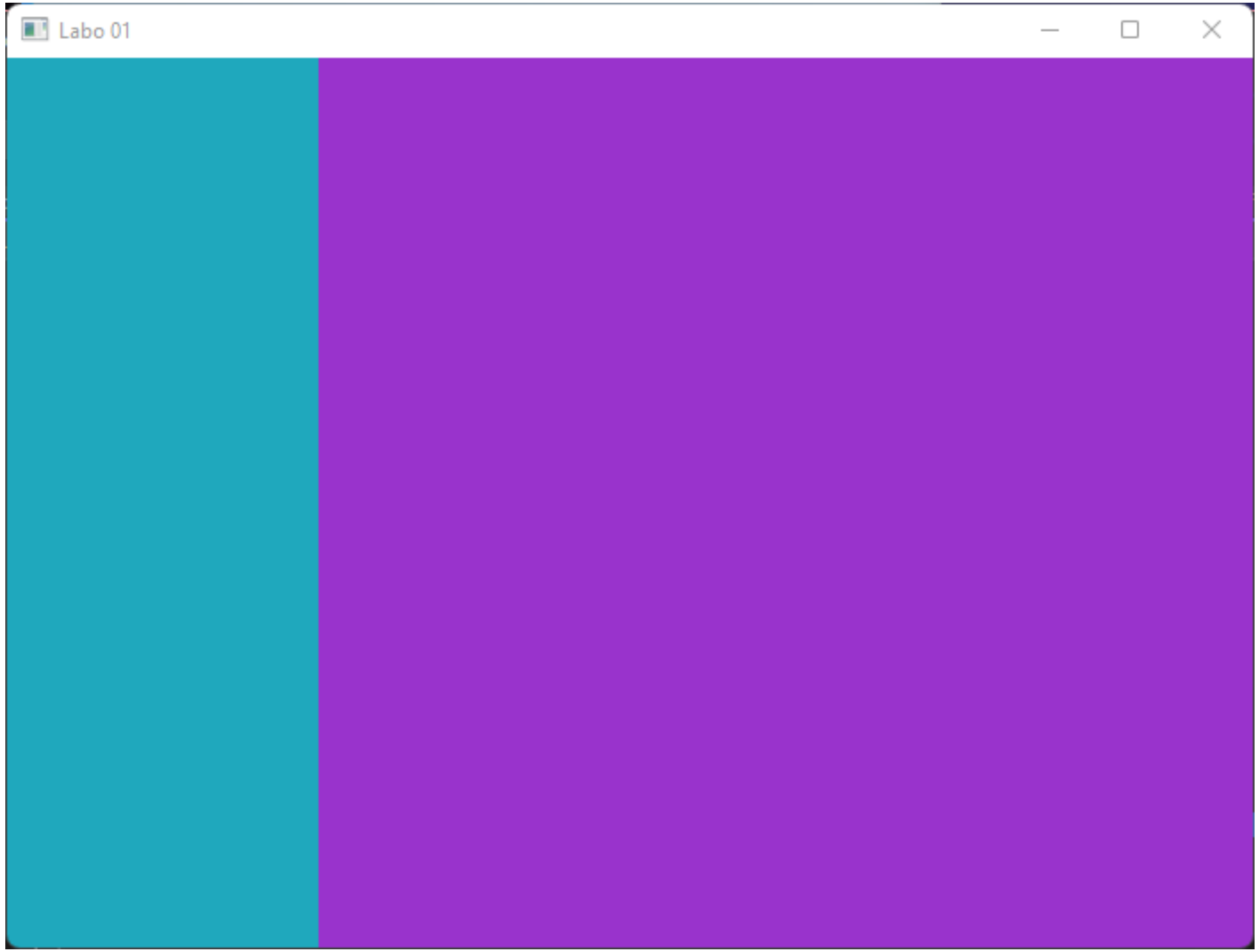
EXERCISES

- 1. Colorize the screen



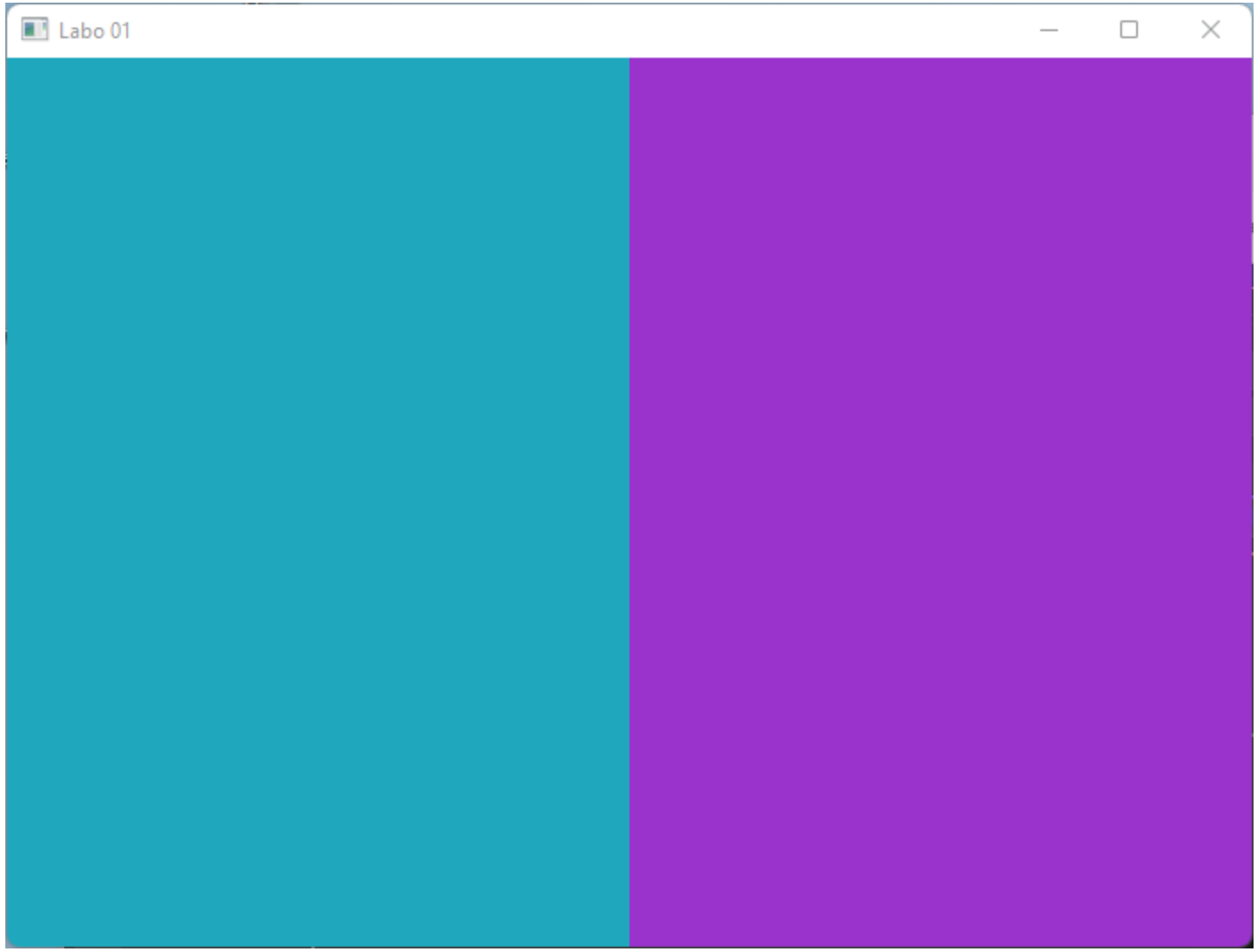
EXERCISES

- 2. Divide the screen



EXERCISES

- 3. When you click on « Fullscreen »
- The division is not right
- Normalize your coordinates to screen space

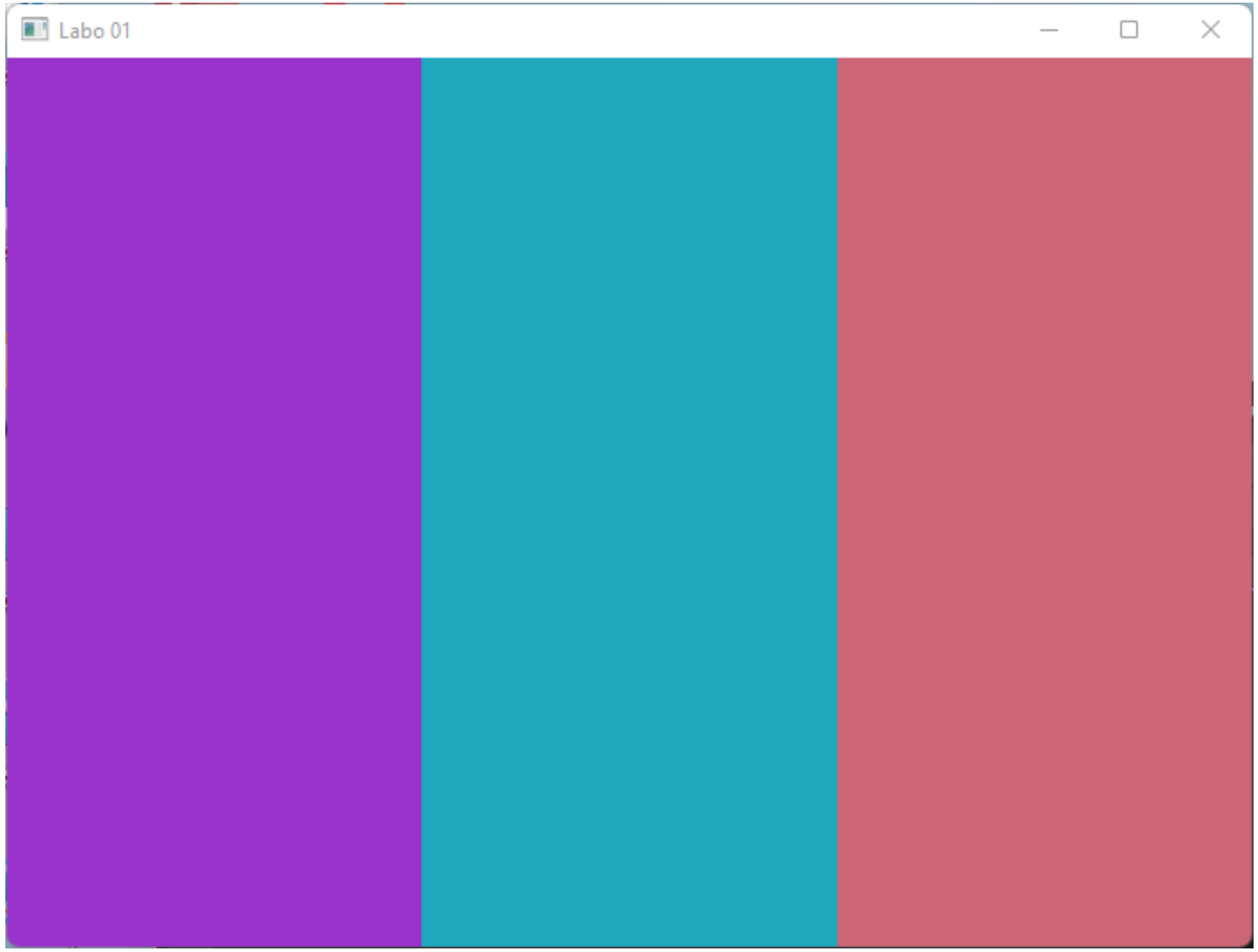


Hint: Before you had a threshold which does NOT depends on the width of the screen!

EXERCISES

- 4. Define a vector « r » in screen coordinates

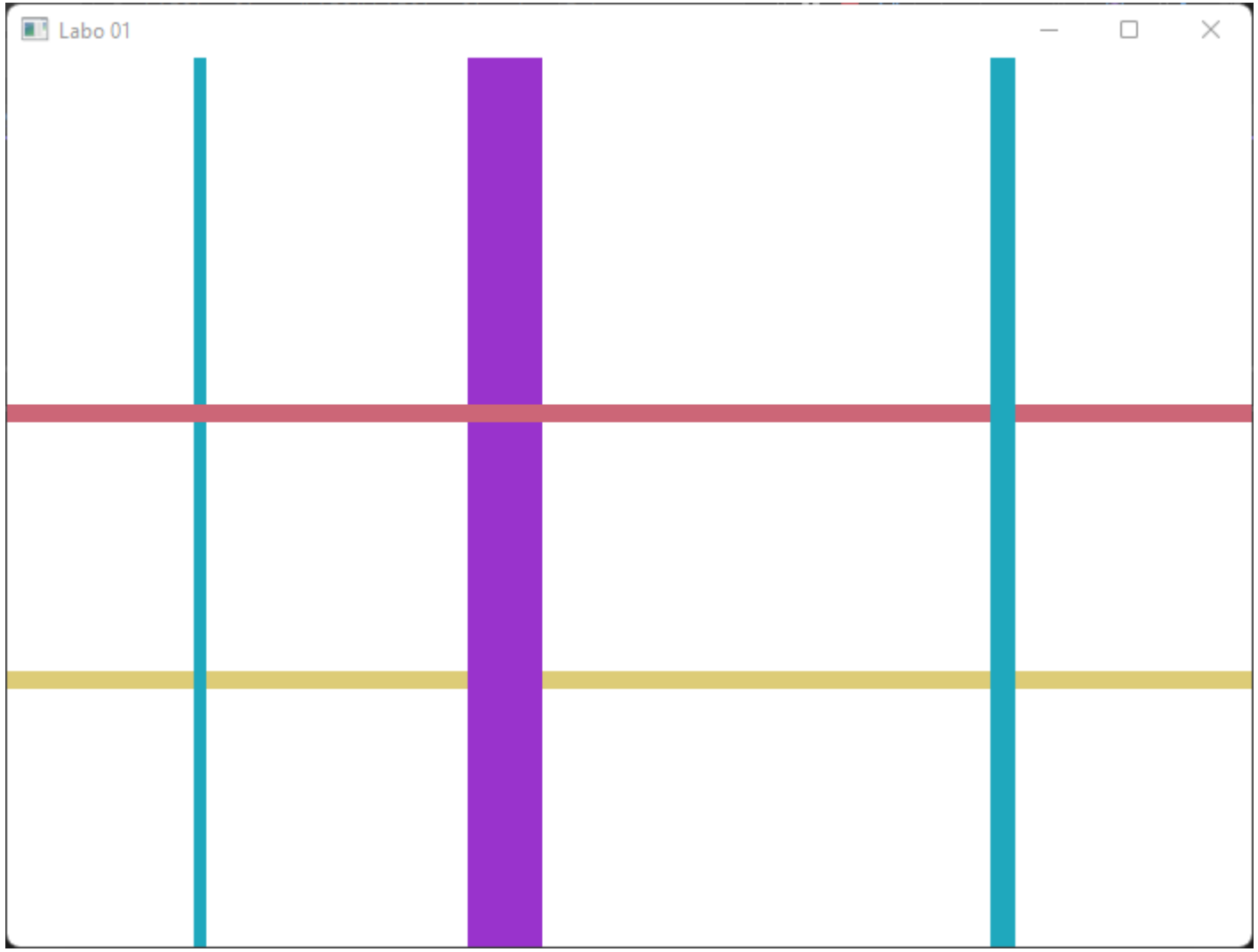
- Draw this:



Hint: as with the previous exercise, define $r \in [0,1] \times [0,1]$ a vector which let you write using 0->1 values!

EXERCISES

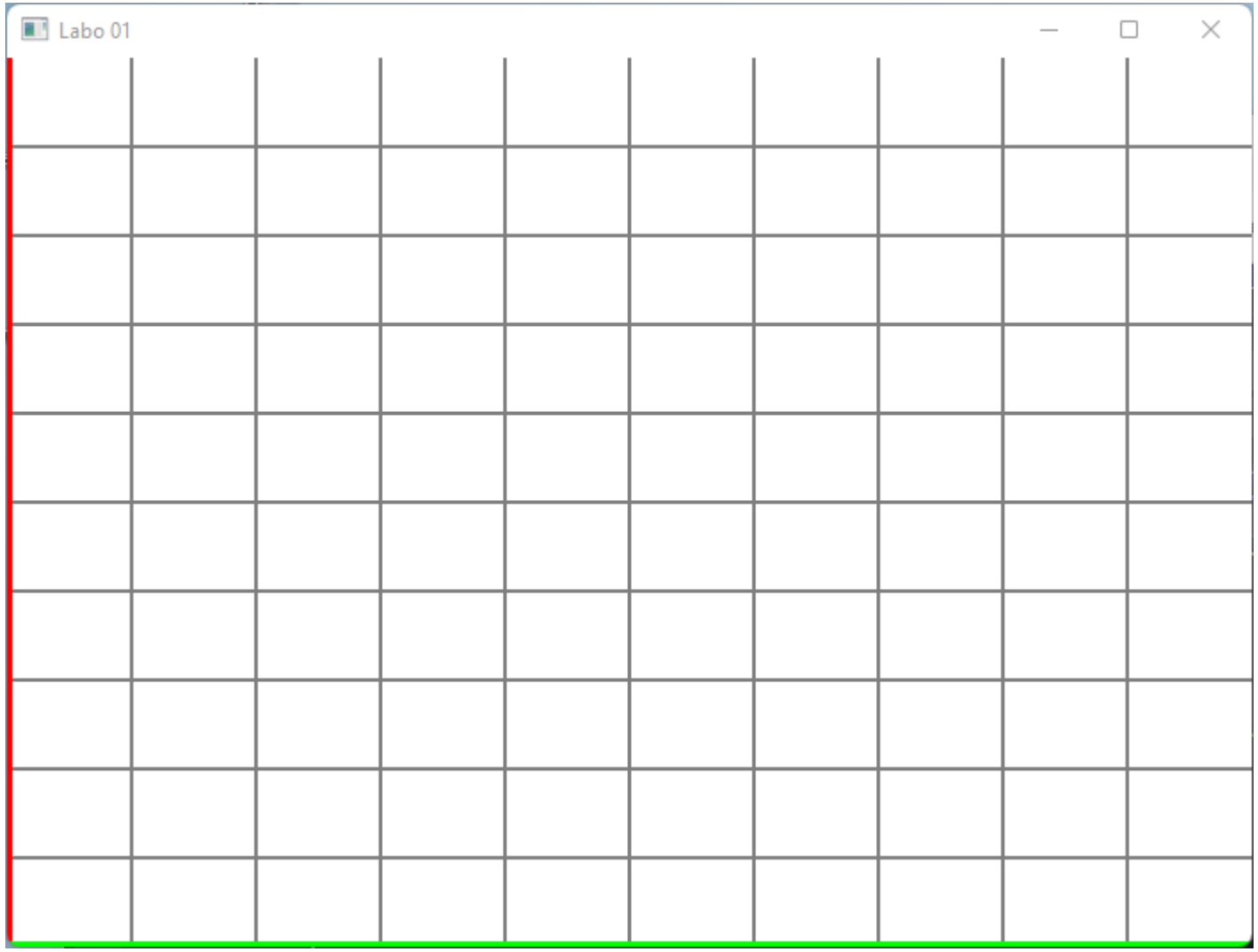
- 5. Become Mondrian



Hint: think carefully on which line is over which other

EXERCISES

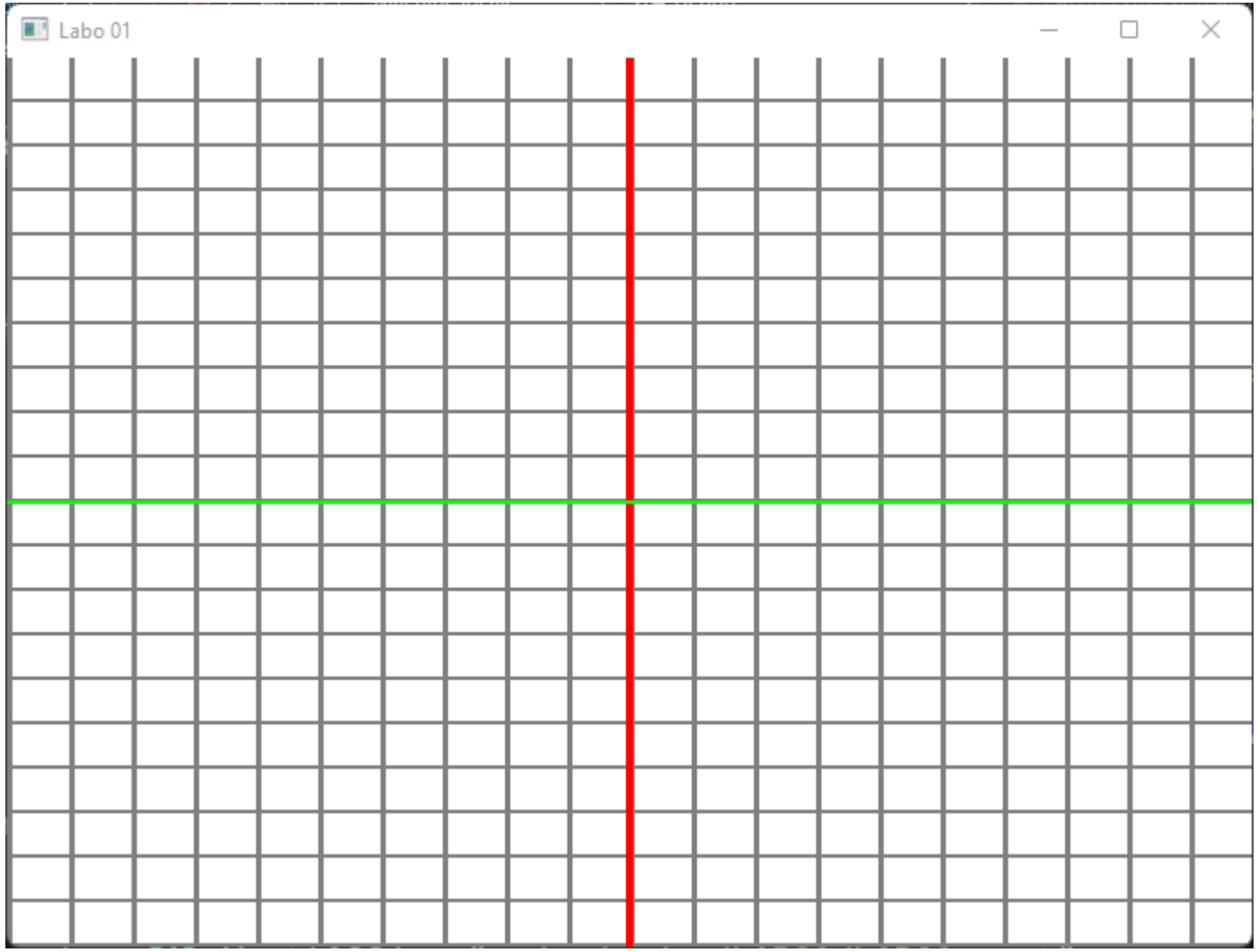
- 6. Draw axis and grid



Question: What is the meaning of the axis colors here?

EXERCISES

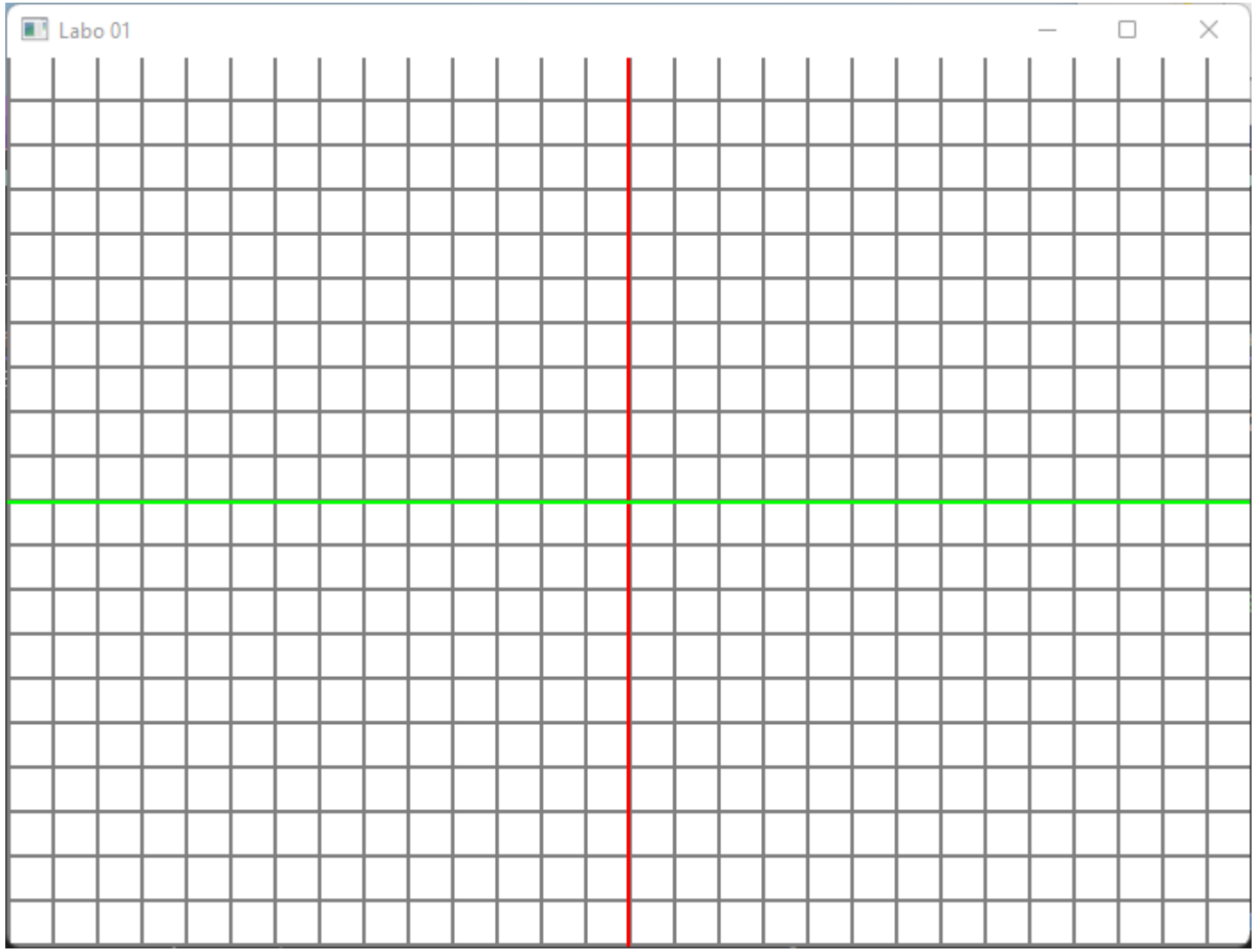
- 7. Center the axis and grid



Hint: You need to define transformations f: $[0, \text{iRes.x}] \times [0, \text{iRes.}y] \rightarrow [0,1] \times [0,1]$, g: $[0,1] \times [0,1] \rightarrow [-1,1] \times [-1,1]$

EXERCISES

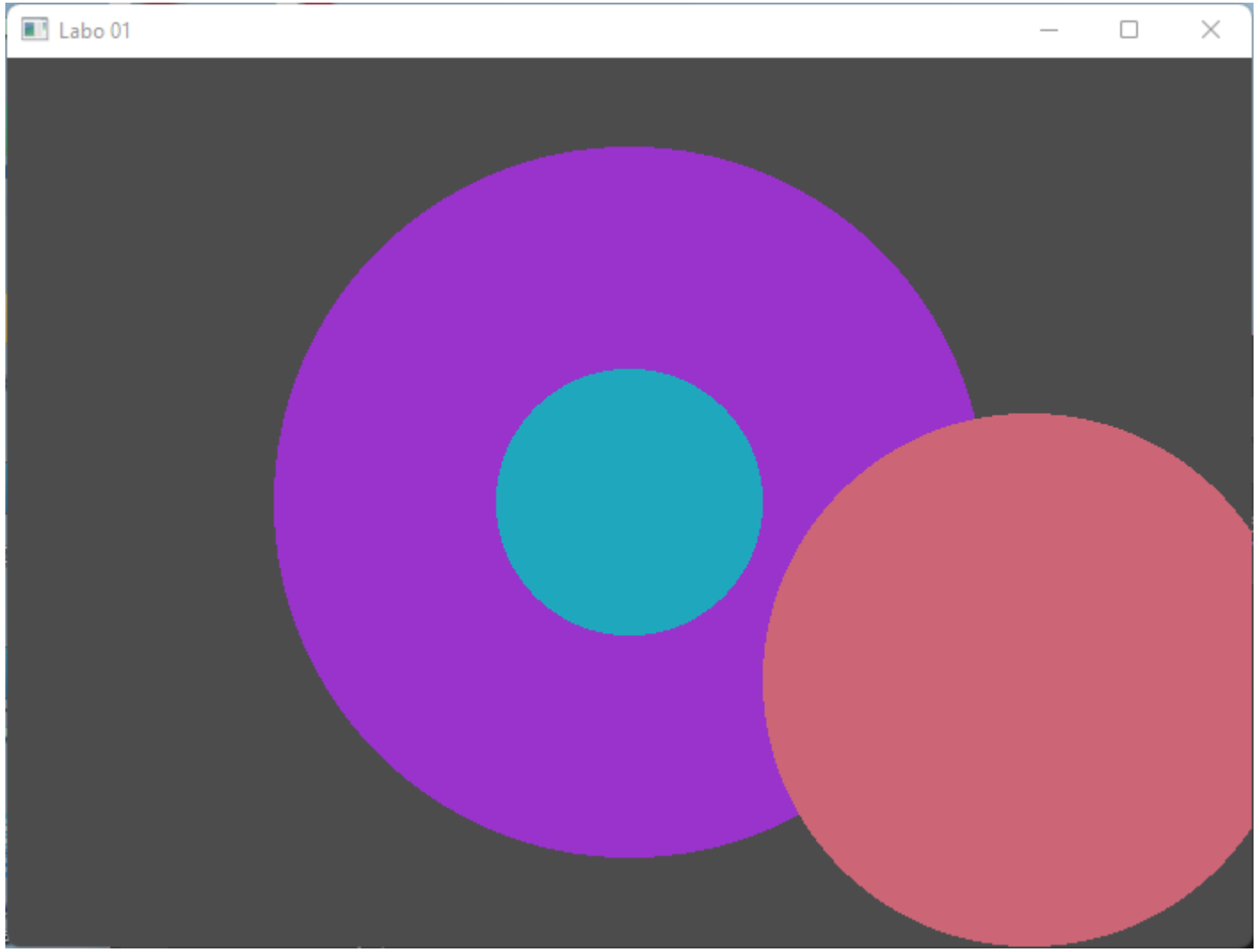
- 8. Center the axis and grid
- With the same spacial resolution!



Hint: Why the squares were not squares before ?

EXERCISES

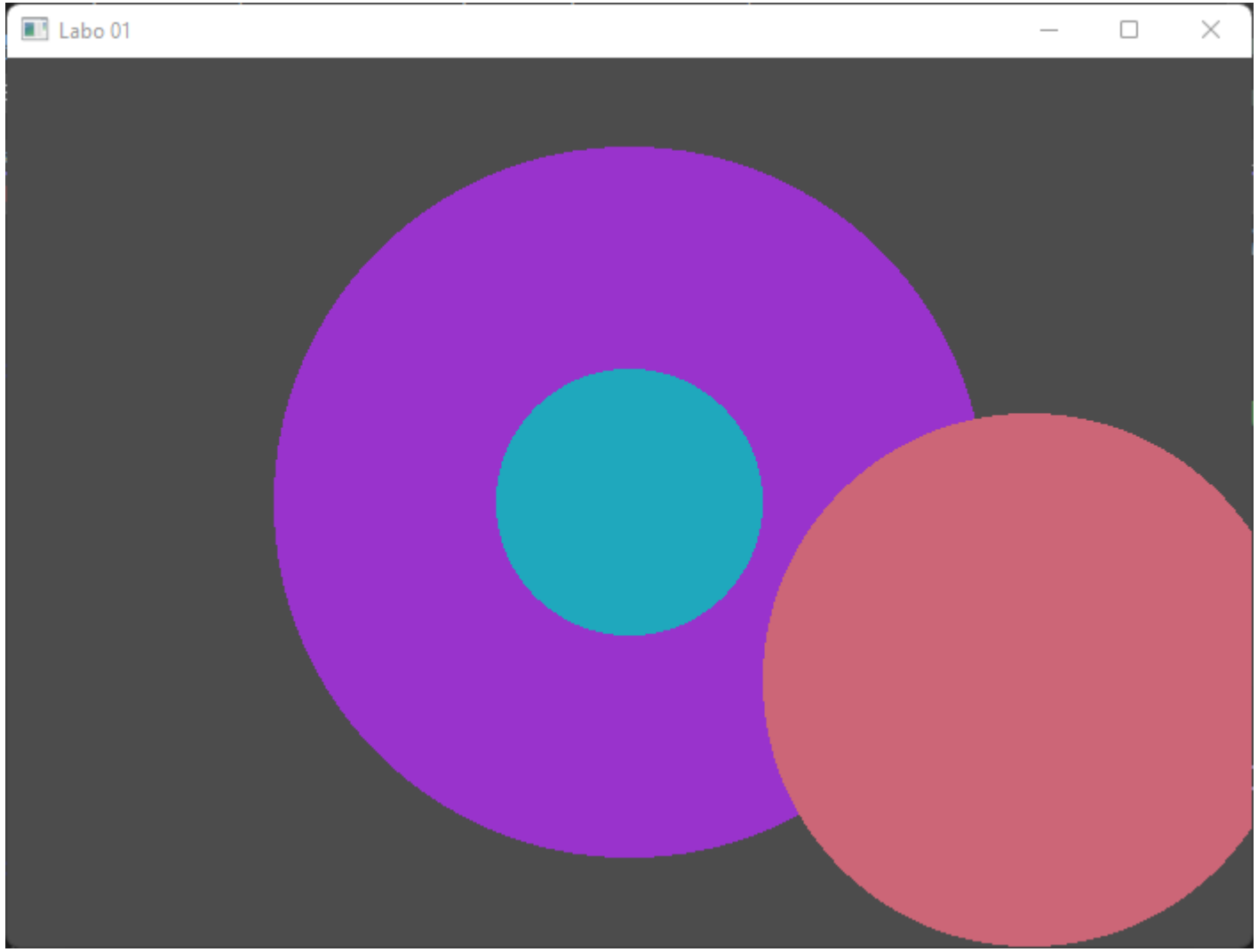
- 9. Draw disks
- What does the « length » function do?



Hint: Look at the length function online : « GLSL length »

EXERCISES

- 10. Draw disks
- Same as before
  BUT
  write a new FUNCTION
- Look the « inout »
  type of variable



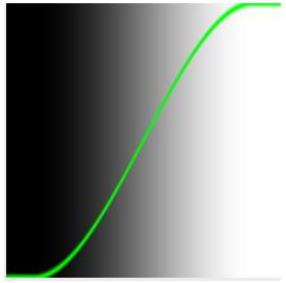
INTERLUDE ON FUNCTION

```
1  float pow1(in float x) { return x*x; }
2  void pow2(out float o, in float x) { o = x*x; }
3  void pow3(inout float io) { float tmp = io*io; io = tmp; }
4  // Not at the same time reading and writing
5  // for inout variables!
6
7  void mainImage( out vec4 fragColor, in vec2 fragCoord )
8  {
9      float c1 = pow1(0.7);
10
11     float c2;
12     pow2(c2, 0.1);
13
14     float c3 = 0.3;
15     pow3(c3);
16     vec3 col = vec3(c1, c2, c3);
17
18     fragColor = vec4(col,1.0);
19 }
```

Three ways to return values!

- in
- out
- inout

EXERCISES



- 11. Look at the « smoothstep » function
- Draw this:

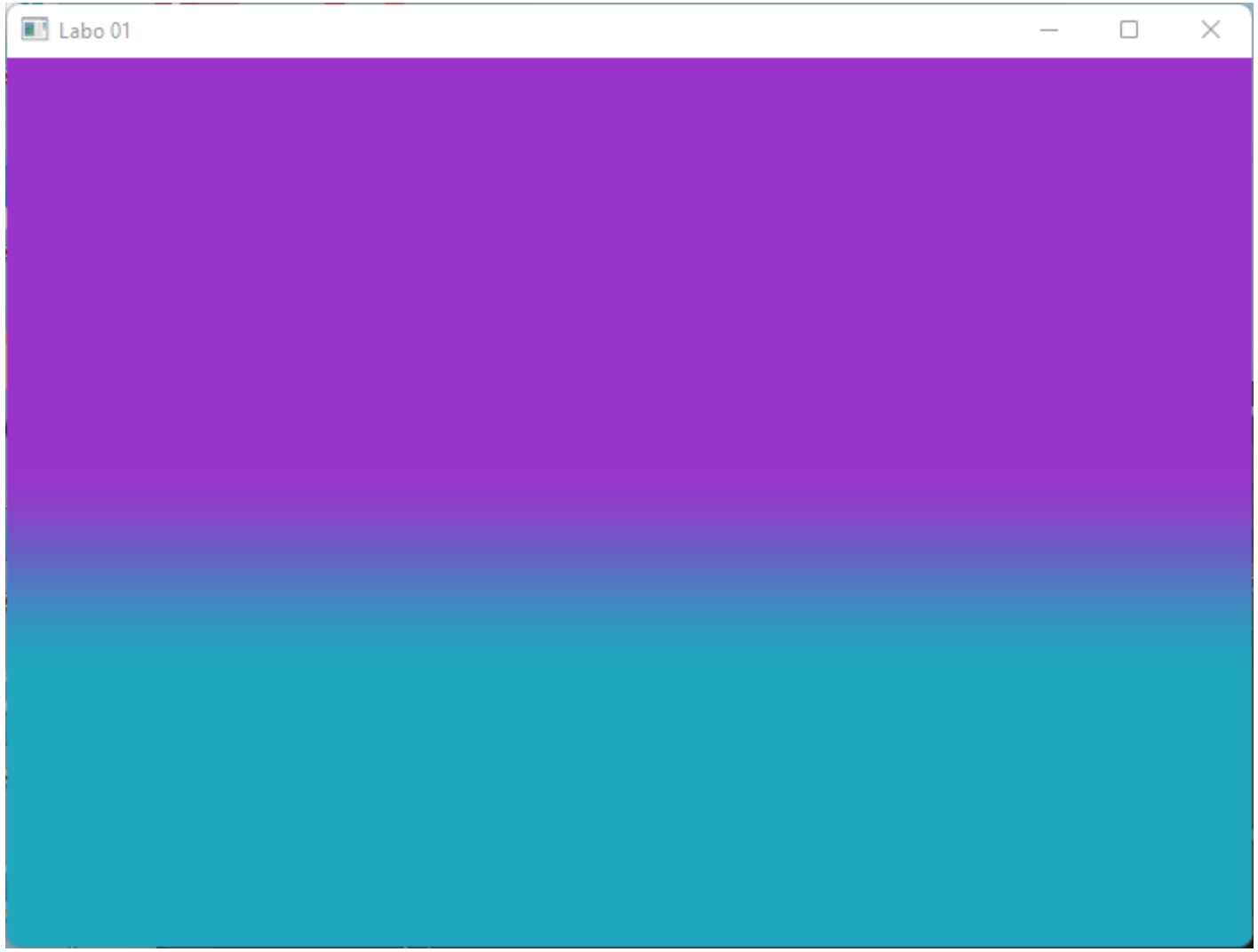
smoothstep is almost <u>linear</u> interpolation between colors smoothed with a 3rd degree polynomial !

```
t = clamp((x - edge0) / (edge1 - edge0), 0.0, 1.0);
return t * t * (3.0 - 2.0 * t);
```

https://thebookofshaders.com/glossary/?search=smoothstep

EXERCISES

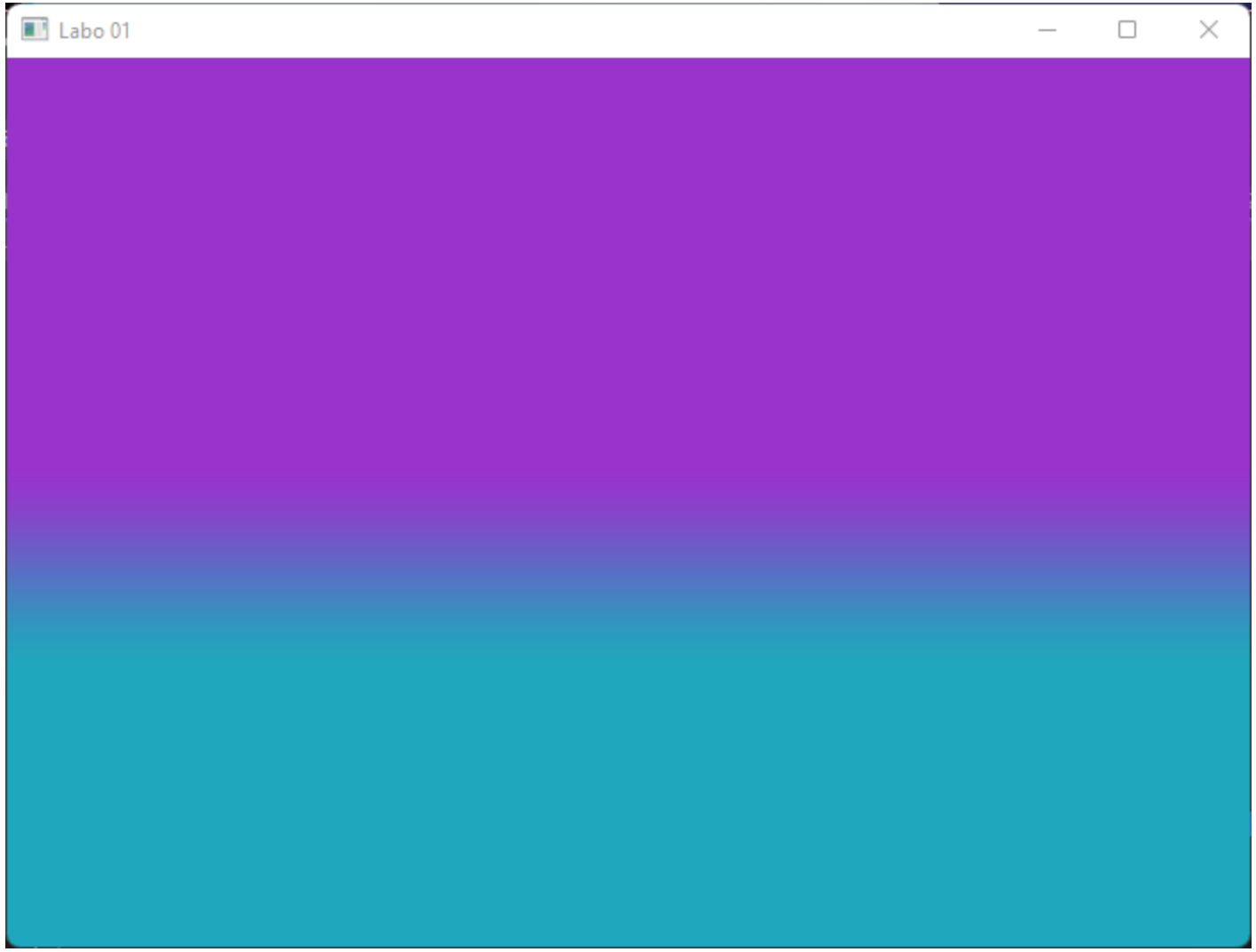
- 12. Replace with colors
- Draw this:



Hint: in mathematics, how do we interpolate between two values? $f(0) = a, f(1) = b$, f = ?

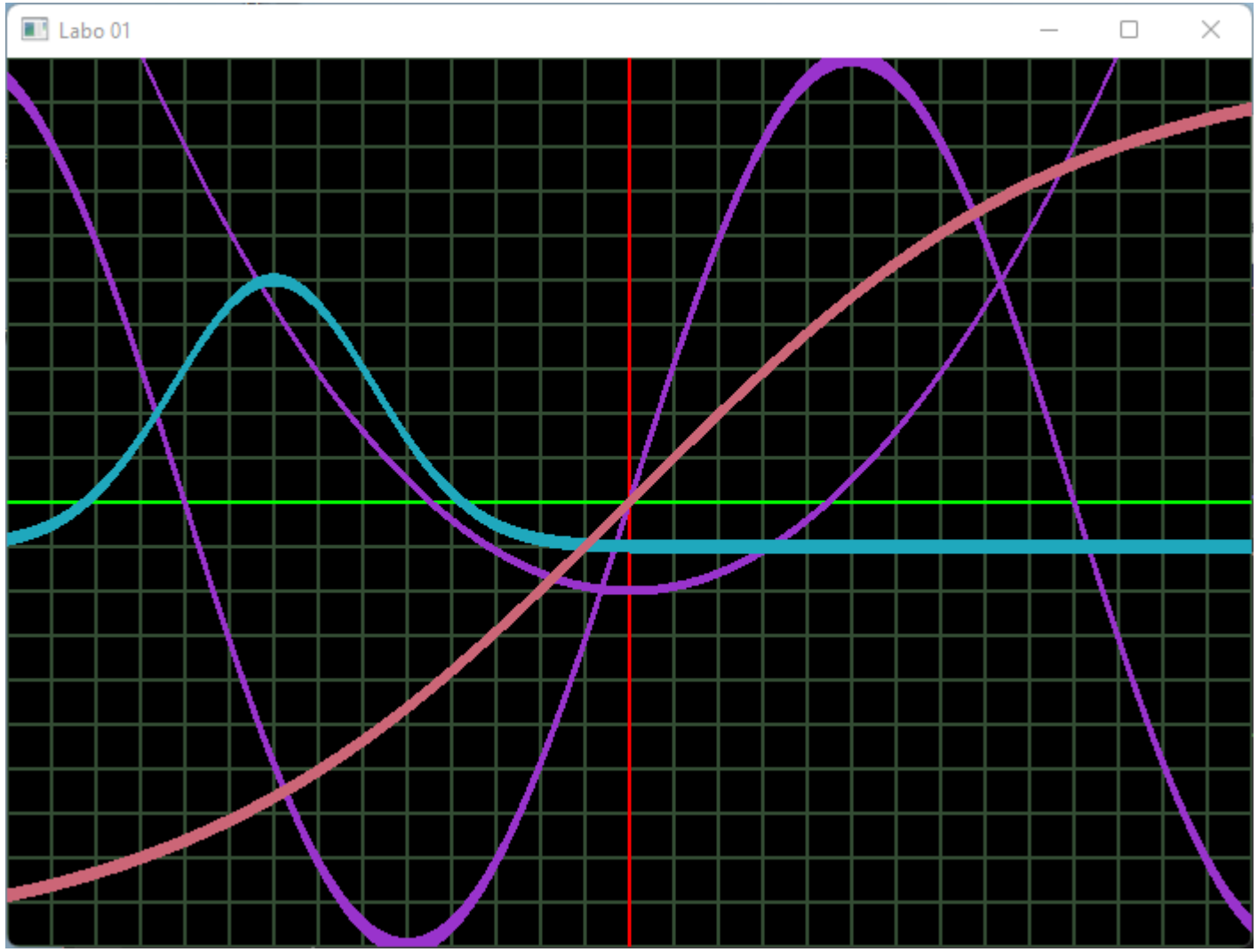
EXERCISES

- 13. SAME
- BUT with « mix » function



EXERCISES

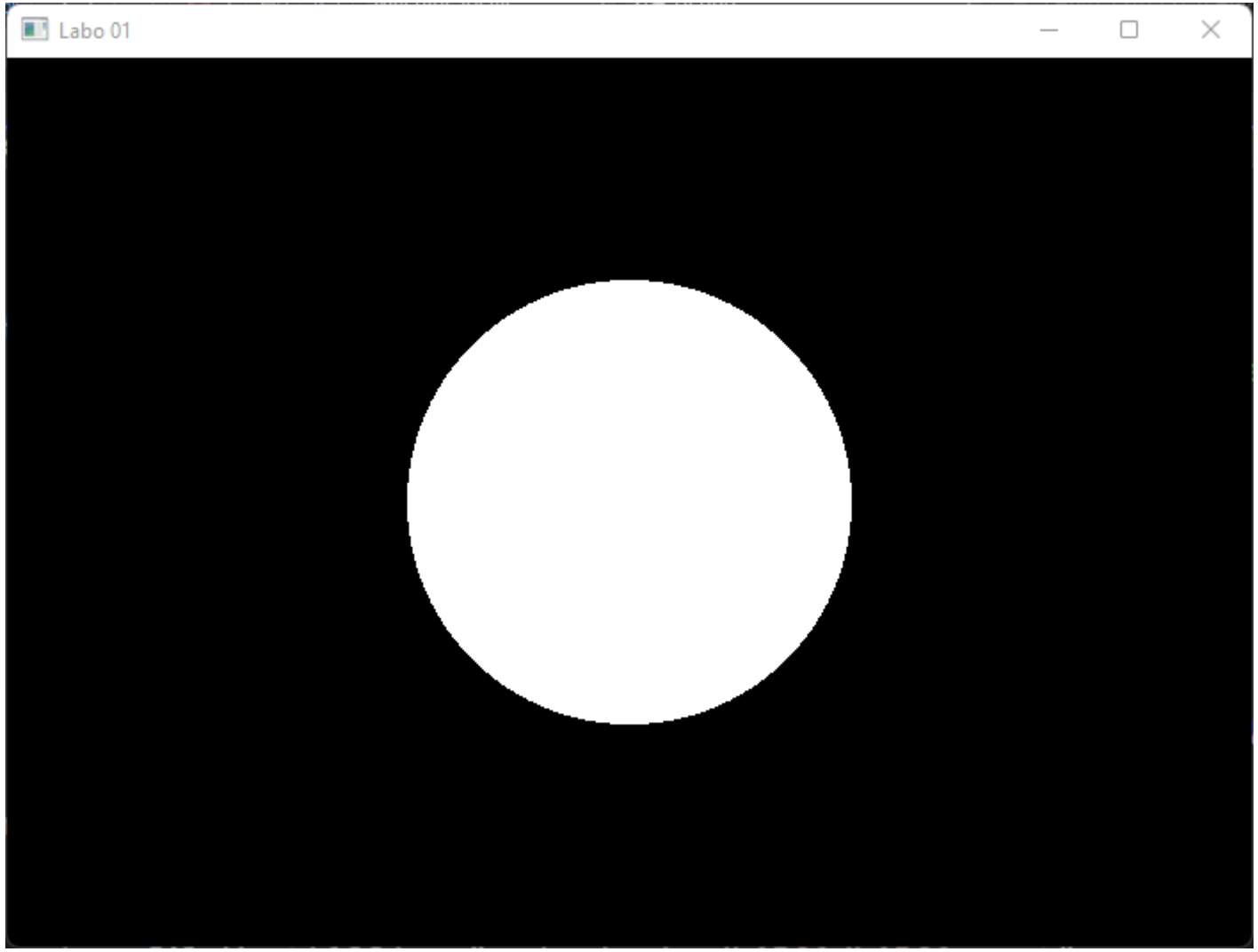
- 14. Draw Functions!
- A function IS a set of points in space
- Here:
  $x^2 - 0.2$
  $\sin(x\pi)$
  $bell\ curve$
  $\tanh(x)$



Hint: When you compare floating points values, you need to use a ε threshold due to floating point arithmetics

EXERCISES

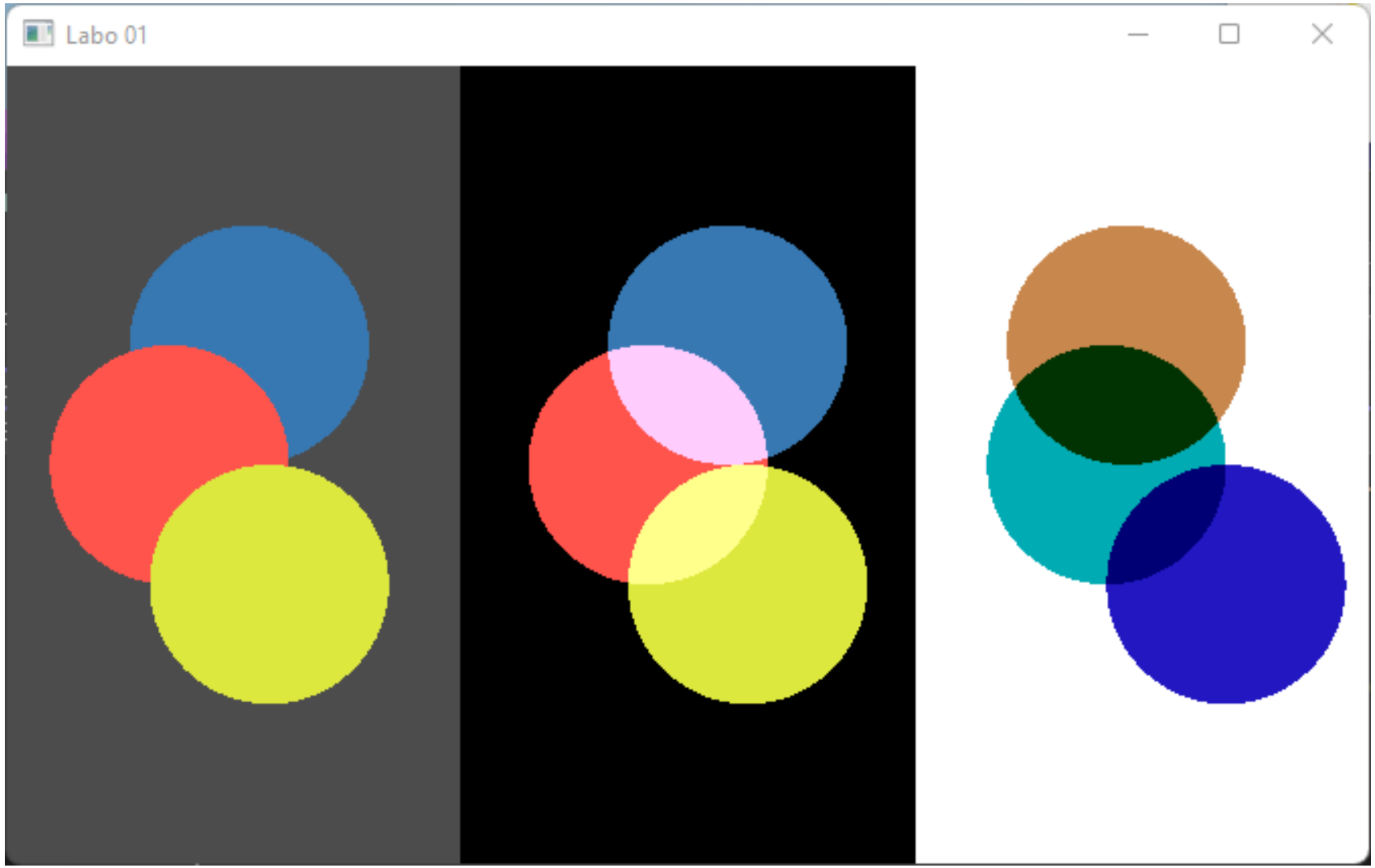
- 15. make a function
- That says if we are inside OR outside of a disk
- Returns
  - 0.0 if outside
  - 1.0 if inside



This is very similar to ex10, but we return a « true » or « false » value in terms of colors!

EXERCISES

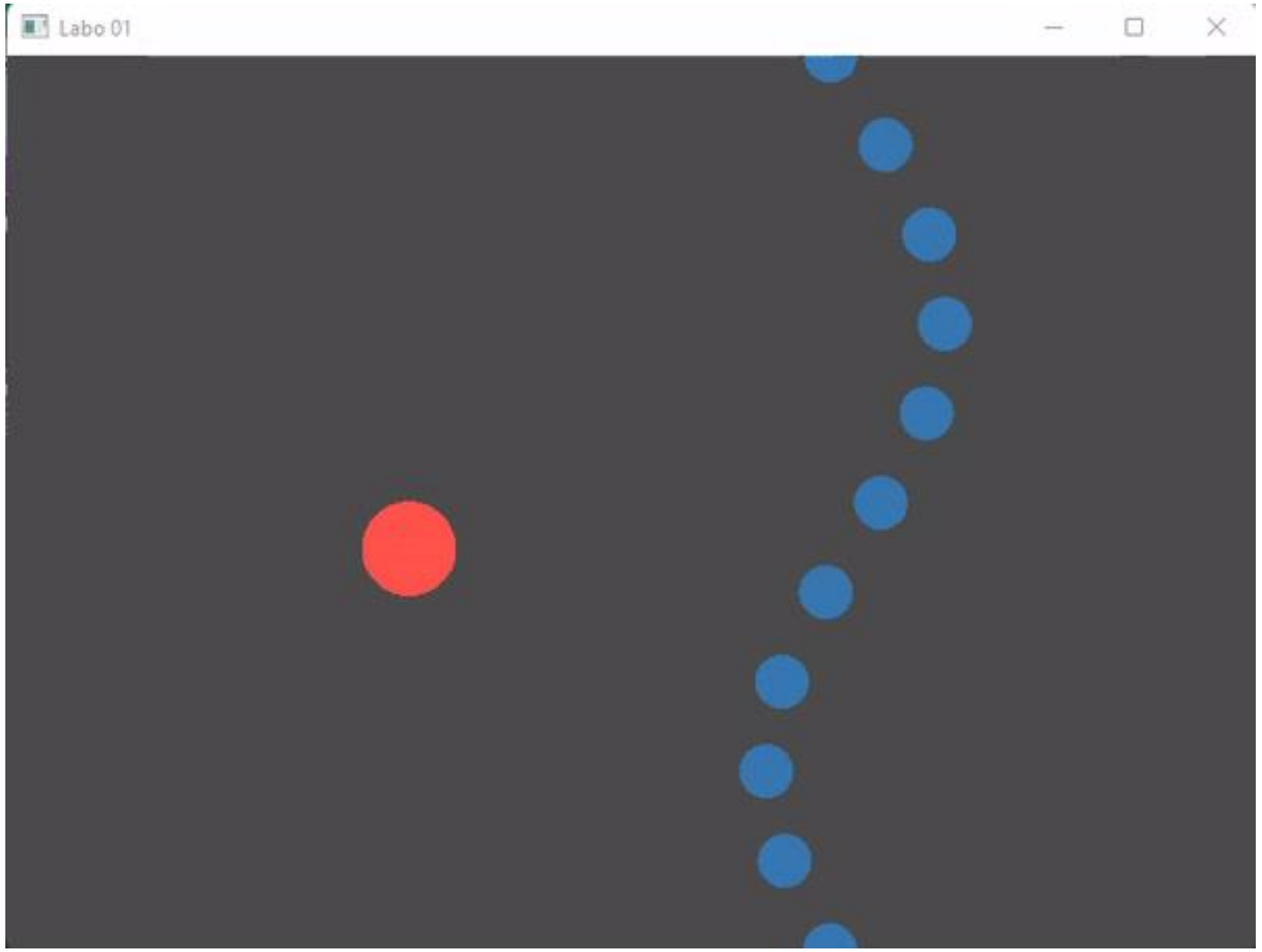
- 16. Colors
- Using the previous function, make:
  - Opaque colors
    - Replacing the previous
  - Additive colors
    - Mixing the colors
  - Substractive colors



Hint: See wikipedia for Addittive and Substractive colors!

EXERCISES

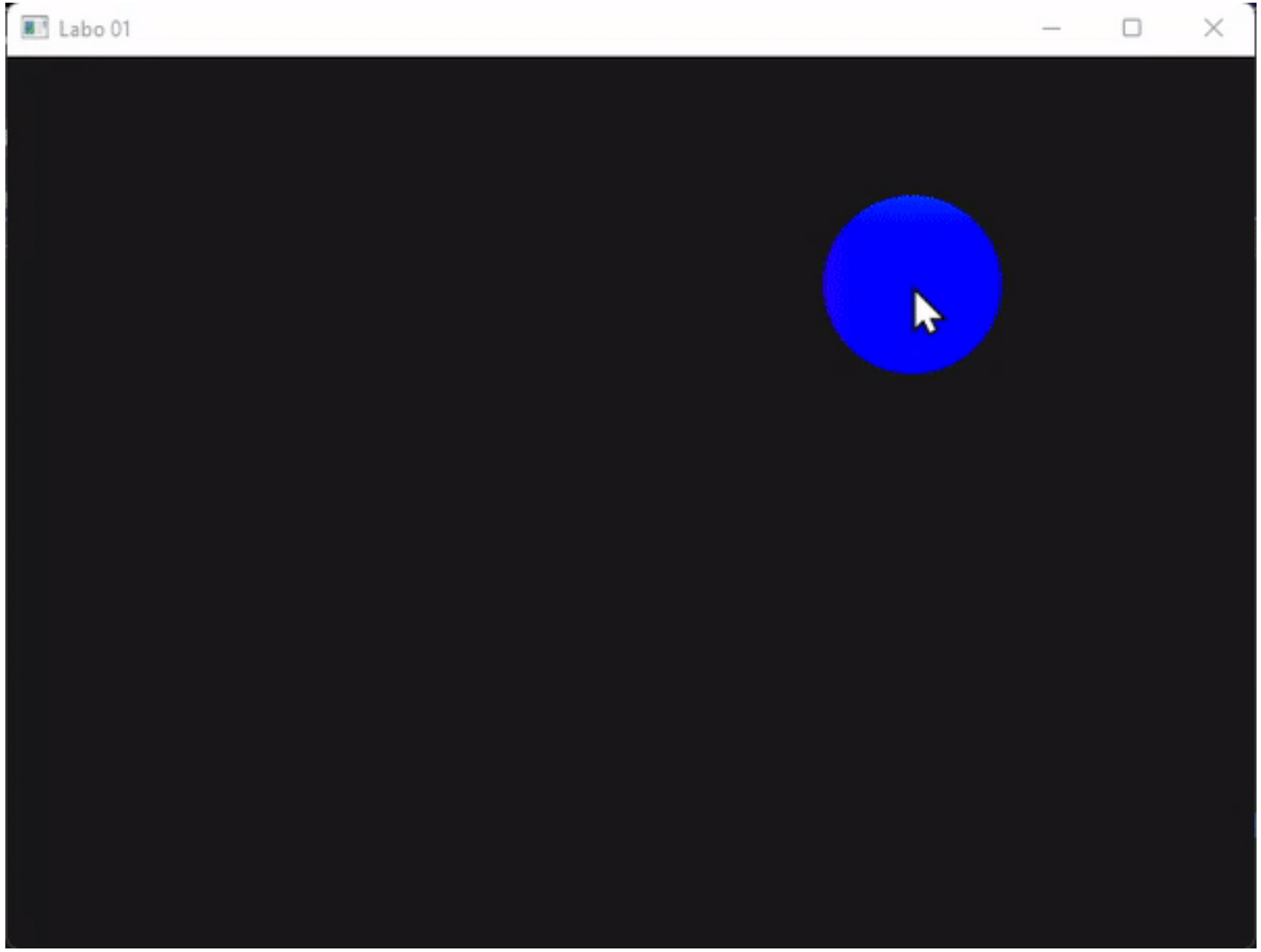
- 17. Animations
- Using « iTime »
- Set the x and y position using sinus functions!



Question: Why the red seems to bounce toward you while the other seems to be more sneaky?

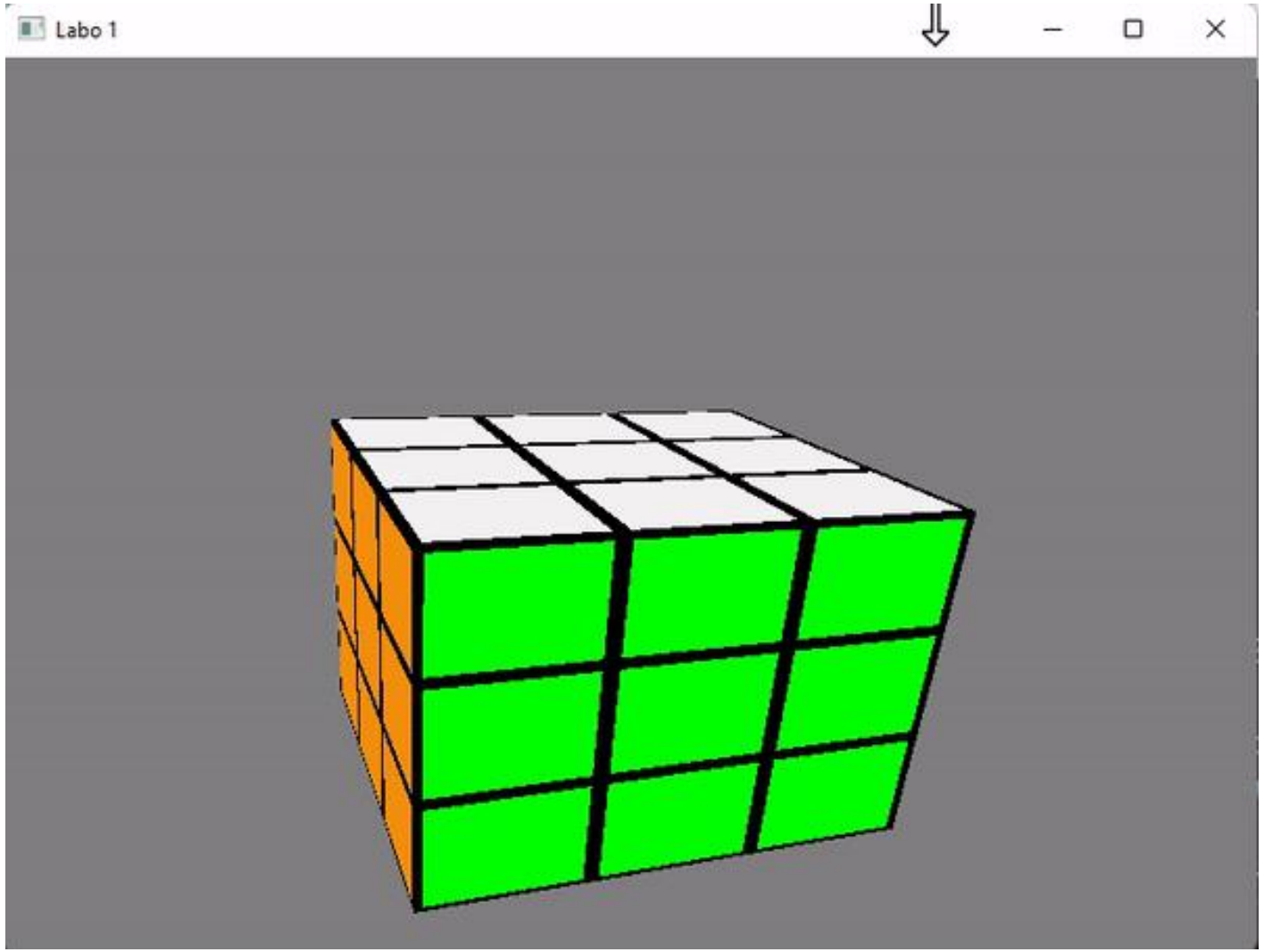
EXERCISES

- 18. Mouse control
- Using « iMouse »



EXERCISES

- 19. Rubik's Cube
- This time the fragement shader is applied to an object!
  - Use the template LAB01_ex19.frag to solve this exercise.



WE HAVE CIRCLES! WHAT ABOUT OTHER FORMS?

- Difficult! We are in the fragment shader
- We need to define figures implicitly
- Next time with vertex shader!

APPENDIX A: IMPLICIT EUCLIDEAN 2D AND 3D FIGURES

Circle: https://www.shadertoy.com/view/3ltSW2
Segment: https://www.shadertoy.com/view/3tdSDj
Triangle: https://www.shadertoy.com/view/XsXSz4
Isosceles Triangle: https://www.shadertoy.com/view/MldcD7
Regular Triangle: https://www.shadertoy.com/view/Xl2yDW
Regular Pentagon: https://www.shadertoy.com/view/llVyWW
Regular Octogon: https://www.shadertoy.com/view/llGfDG
Rounded Rectangle: https://www.shadertoy.com/view/4llXD7
Rhombus: https://www.shadertoy.com/view/XdXcRB
Trapezoid: https://www.shadertoy.com/view/MlycD3
Polygon: https://www.shadertoy.com/view/wdBXRW
Hexagram: https://www.shadertoy.com/view/tt23RR
Regular Star: https://www.shadertoy.com/view/3tSGDy
Star5: https://www.shadertoy.com/view/wlcGzB
Ellipse 1: https://www.shadertoy.com/view/4sS3zz
Ellipse 2: https://www.shadertoy.com/view/4lsXDN
Quadratic Bezier: https://www.shadertoy.com/view/MlKcDD
Uneven Capsule: https://www.shadertoy.com/view/4lcBWn
Vesica: https://www.shadertoy.com/view/XtVfRW
Cross: https://www.shadertoy.com/view/XtGfzw
Pie: https://www.shadertoy.com/view/3l23RK
Arc: https://www.shadertoy.com/view/wl23RK
Horseshoe: https://www.shadertoy.com/view/WlSGW1
Parabola: https://www.shadertoy.com/view/ws3GD7
Parabola Segment: https://www.shadertoy.com/view/3lSczz
Rounded X: https://www.shadertoy.com/view/3dKSDc
Joint: https://www.shadertoy.com/view/WldGWM
Simple Egg: https://www.shadertoy.com/view/Wdjfz3

2D

https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

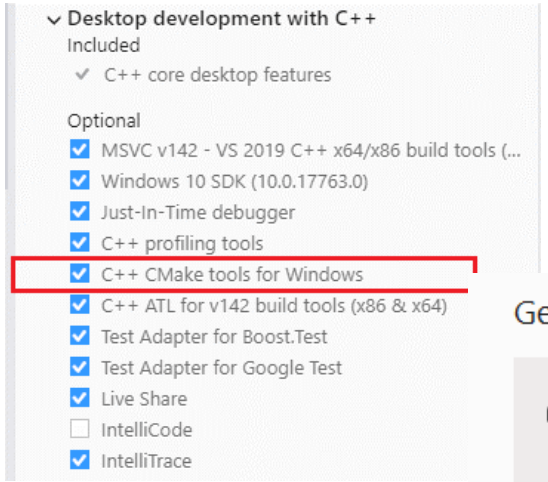
3D

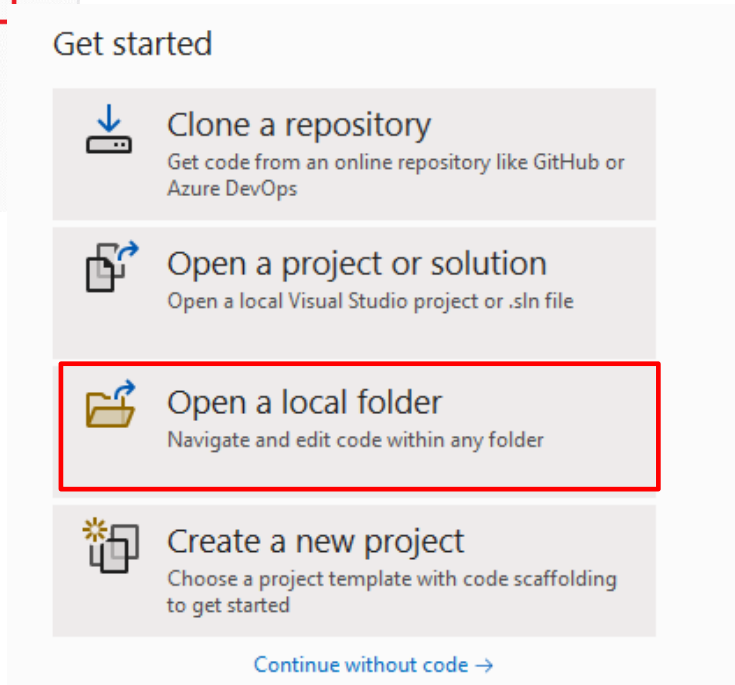
APPENDIX A: IMPLICIT EUCLIDEAN 3D FIGURES – LEVEL MADNESS

- Video from an expert in implicit geometry functions rendering:
  - https://www.youtube.com/watch?v=8--5LwHRhjk
- Explains how to make a complex scene **without any geometry** using only the **fragment shader**!
- Every pixel's color you see are computed using only the spatial fragment coordinates!!!
- Code: https://www.shadertoy.com/view/WsSBzh

APPENDIX B : RUNNING THE CODE (VISUAL STUDIO (WINDOWS ONLY)

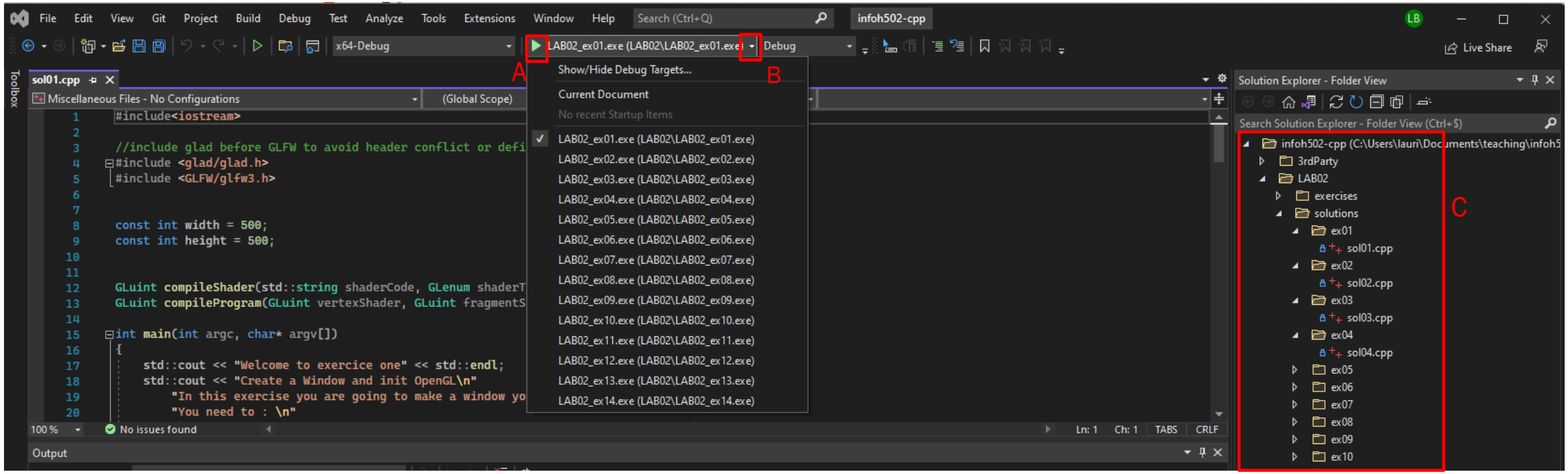
- Install Visual Studio (Community) if not already done :
  - https://visualstudio.microsoft.com/downloads/
  - Install the component « Desktop development with C++ » during the installation process
    Be sure that « C++ Cmake tools for windows » is installed
- Install git if not done: https://git-scm.com/downloads
- Using the terminal, clone the repository in the folder you want: git clone --recursive https://gitlab.com/lisa-vr-course/info-h502_202324
- Open Visual Studio and select « Open a local folder » then select the folder that you just clone





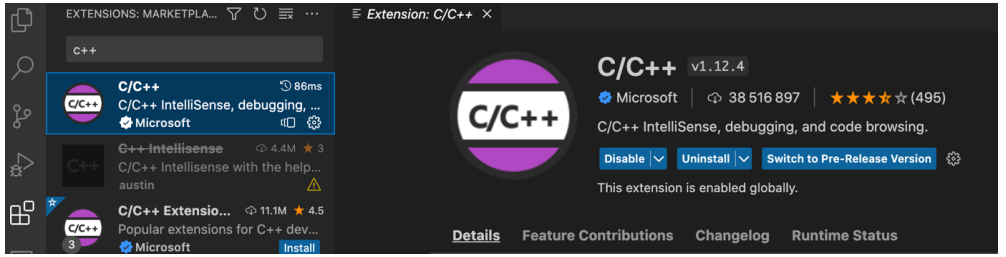
→ More info on visual studio and cmake are available here: https://learn.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio?view=msvc-170

APPENDIX B : RUNNING THE CODE (VISUAL STUDIO (WINDOWS ONLY)

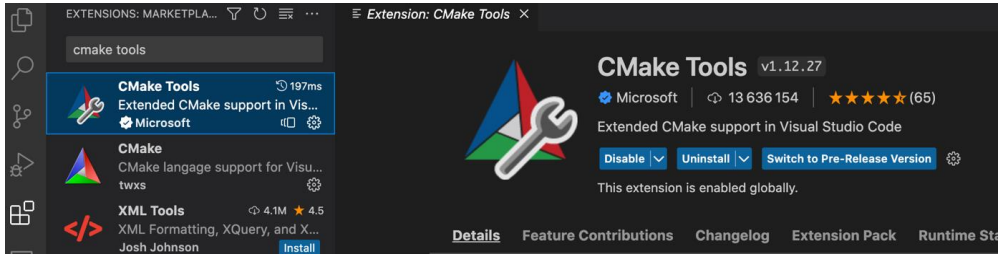


- A. Press the green play button to run the executable
- B. Select the executable you want to run
- C. Navigate into the files and edit them to do the exercises

APPENDIX C : RUNNING THE CODE (VISUAL STUDIO CODE)



- Install Visual Studio Code : https://code.visualstudio.com/download
- C++ extension for VSC : go to the extension page and search « c++ », install it
- CMake tools extension for VSC : search « cmake tools », install it
- If you don’t have CMake, install it : https://cmake.org/download/
  - Check your CMake version to see if it’s installed : « cmake –version »



APPENDIX C : RUNNING THE CODE (VISUAL STUDIO CODE - MACOS)

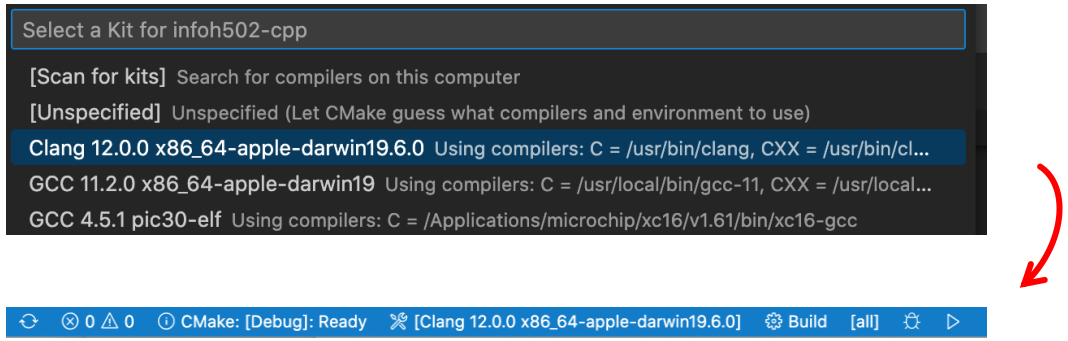
- You need a C++ compiler, you can install gcc by installing Xcode (heavy), or the *Command Line Tools for Xcode* (light) by running « xcode-select –install » in the terminal
  - Check you c++ compiler with « clang –v »
  - If you want to use VSC on another operating system, this step is the only one that change

```
(base) MacBook-Pro-8:LAB02 elinesoetens$ clang -v
Apple clang version 12.0.0 (clang-1200.0.32.21)
Target: x86_64-apple-darwin19.6.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
```

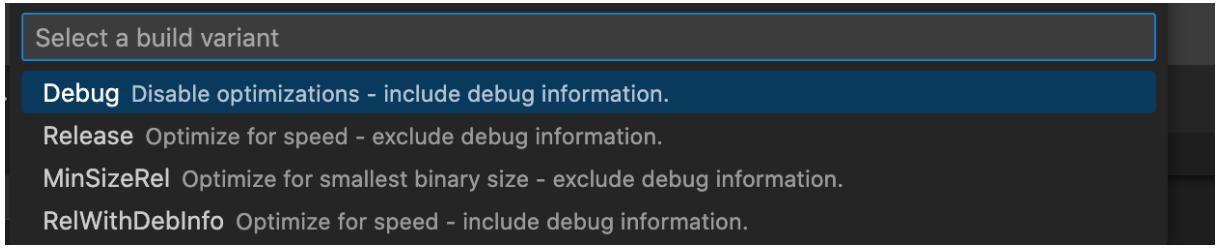
- Clone the git repository : « git clone --recursive https://gitlab.com/lisa-vr-course/info-h502_202324.git»

APPENDIX C : RUNNING THE CODE (VISUAL STUDIO CODE)

- Open the cloned repository in VSC



- Select a Kit (C++ compiler) : Open the Command Palette (MacOs : ⇧⌘P, Linux : Ctrl+Shift+P) and run « CMake : Select a Kit », select the compiler you want
  - You should see the selected kit in the status bar at the bottom
- Choose the mode (Debug, Release, ...) by running « Cmake : Select Variant » in the Command Palette or select the mode in the status bar at the bottom. Use Debug to have access to the debug info

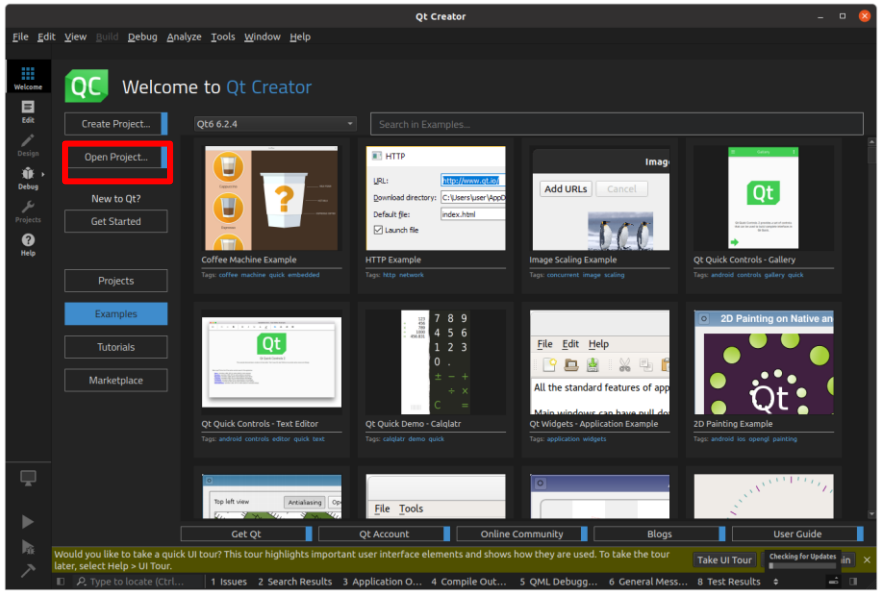


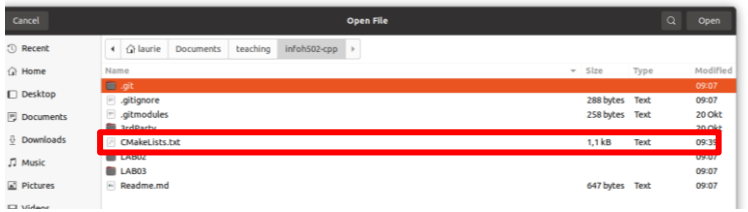
APPENDIX C : RUNNING THE CODE (VISUAL STUDIO CODE)

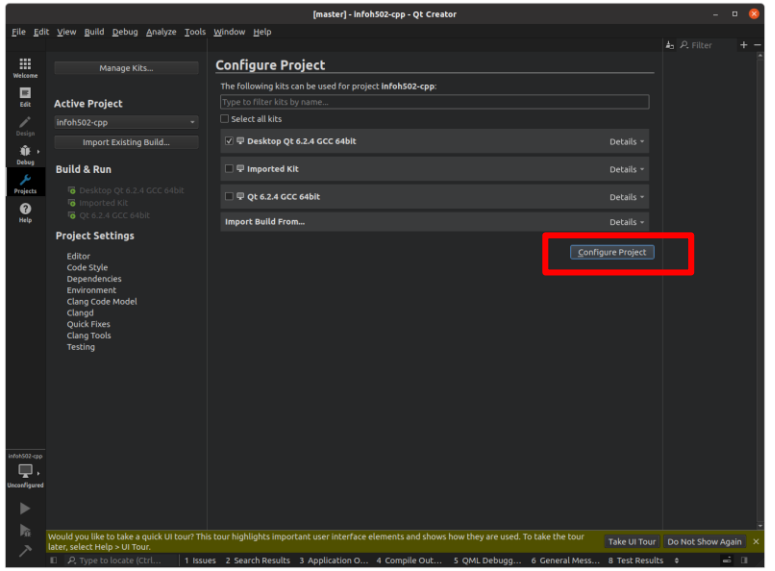
- Run « Cmake : Configure » in the Command Palette to generate a build folder
- Build you code : choose your target in the status bar (default : all the target)
- Run your code : select a target then launch it or click the executable produced in the build folder

APPENDIX D : RUNNING THE CODE WITH QT CREATOR (CROSS-PLATFORM)

- Install cmake if not already done: https://cmake.org/download/ or sudo apt-get install cmake
- Install git: https://git-scm.com/downloads or sudo apt-get install git
- Using the terminal, clone the repository in the folder you want: git clone –recursive https://gitlab.com/lisa-vr-course/info-h502_202324
- Install Qt creator (open-source): https://www.qt.io/download-open-source
- Open Qt Creator and select Open project
- Chose the default configuration and press configure project

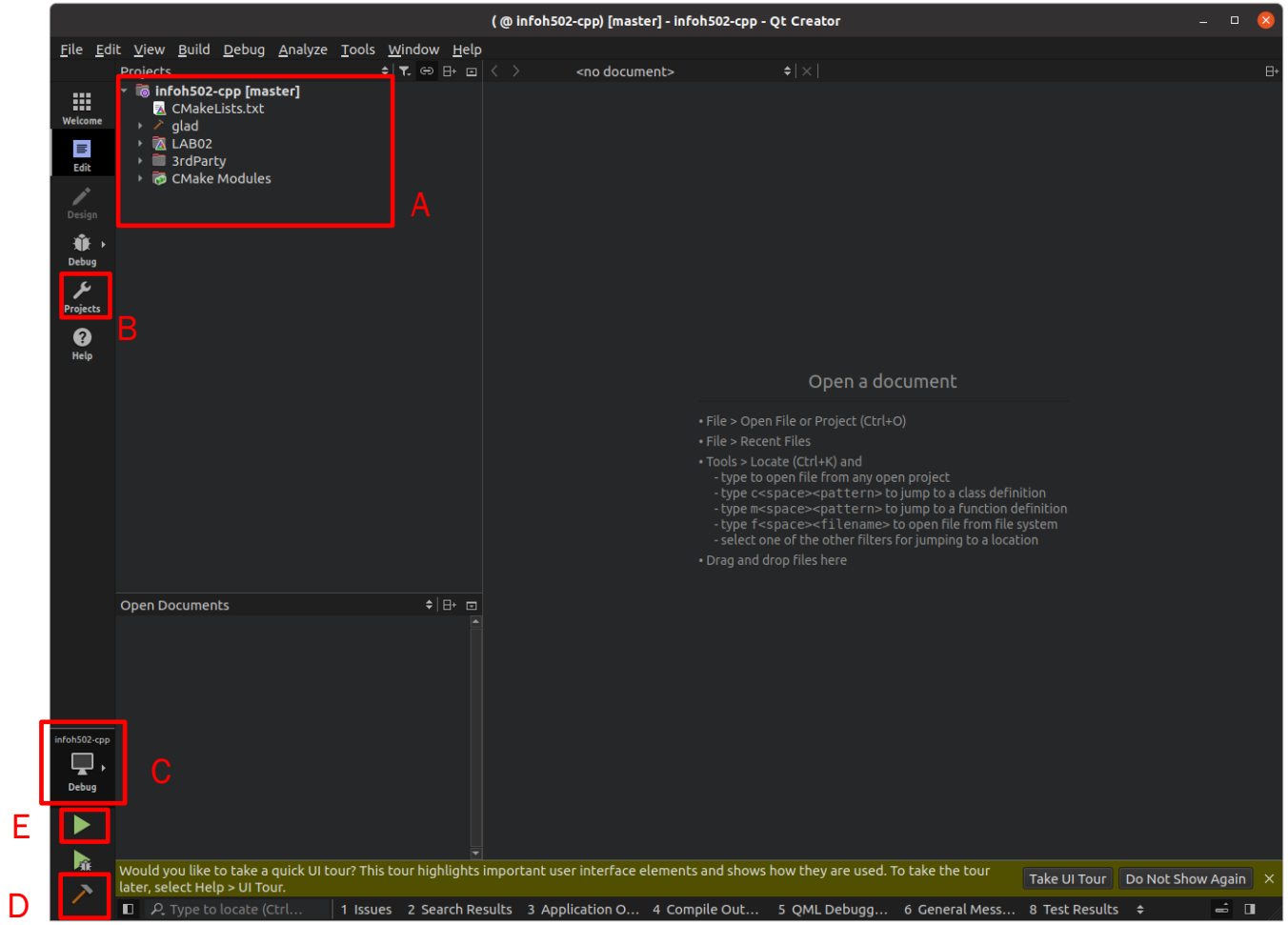






APPENDIX D : RUNNING THE CODE WITH QT CREATOR (CROSS-PLATFORM)

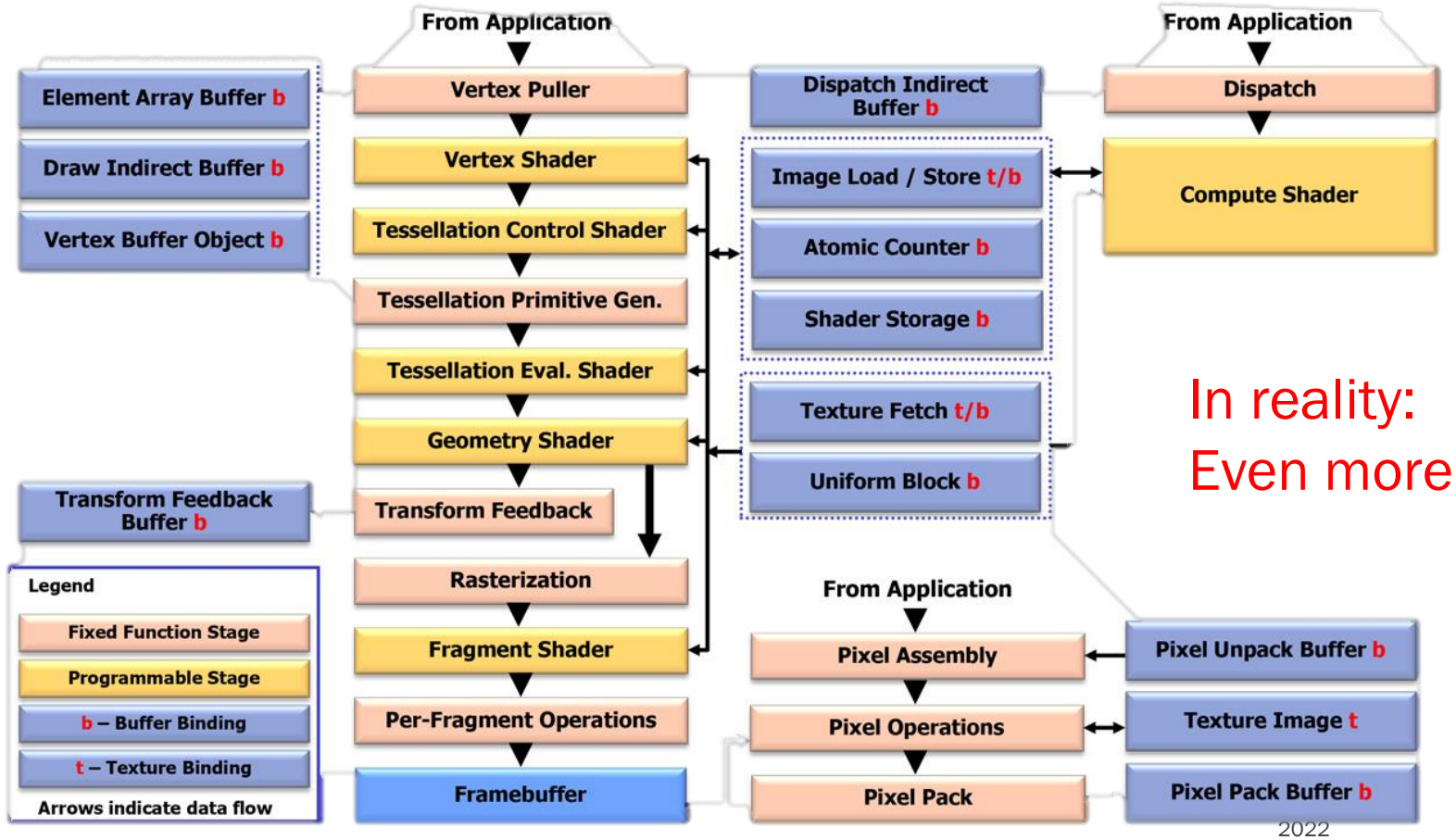
- A. Navigate the project
- B. Change Project settings
- C. Change the selected executable
- D. Build the executable
- D. Run the selected executable



APPENDIX E: INSIGHTS ON THE PIPELINE

Transform feedback

- Tissue simulations
- particle systems
- Physical simulations



In reality:
Even more stuff!