Lab 24.01.2020

- today we will discuss what is happening in the source we were given
- he showed how to use xcode with our development environment
- if you use an IDE we can also do debugging of the code which is really useful

Code Description `lab_02.cpp`

- initialize OpenGL
- create a window
- call glew to find out which functions our GPU supports
- set the refresh rate
- `vertex_shader_source`, `fragment_shader_source` are strings written in GLSL
- `vertex_shader_source` is used for geometry of the point and also for the color, *vertex shader* is meant to process this data. Here we don't do anything and just let it stay like it is
- `fragment_shader_source` is processing pixel data and it will generate the color of the individual pixels, this will be called for each pixel that makes up our triangle, it is interpolated
- the fragment shader is generally run in super parallel
- after that the shaders need to be compiled in order to run
- we link the shaders together, delete the now obsolete shaders and then get the location of the shaders in memory to be able to pass values to it
- we create an array of points that make up our triangle
- it is possible to pass most things to the GPU, we only need a few basic things
- then we will create all the buffers and pass the buffer data, giving it all the data and how often the data will change, here not very often
- we then call some functions that tell the CPU how to send the data to the GPU, like where is the position, where is the data, etc
- this means that we can use whatever layout that you want
- `stride` specifies how much data there is per point
- `position_attribute_location` starts at 0 because our points start right at the beginning
- `color_attribute_location` tells the GPU how far to jump to reach the first color values in a point
- `glClearColor`, `glViewport` tells OpenGl how to reset the screen in between frames and where to draw the 2D screen representation
- in the infinite loop:
  - `glClear` actually clears the screen, we just clear the color here
  - `glUseProgram` runs the shaders we programmed earlier
  - `glBindVertexArray` uses the specified vertex array
  - `glDrawArrays` finally draws the stuff, we specify the render type, how many vertices and when they start in the array
  - `SDL_GL_SwapWindow` *bufferswapping*, makes one buffer visible, the invisible one is then rendered to while the other frame is being displayed
- finally we do the cleanup just like in lab 1

Changing stuff

fragment shader – Line 46

- `gl_FragColor = vec4(1.0, 0, 0, 1.0);` to make it red
- `gl_FragColor = fragment_color + 0.3;` to everything brighter or whiter
- `gl_FragColor = fragment_color * 0.7;` to everything darker or increase the contrast

vertex shader – Line 37

- `gl_Position = position + vec4(0.5, 0.5, 0.0, 0.0);` moves triangle to top right

- `gl_Position = position + vec4(0.5, 0.5, 10.0, 0.0);` doesn't work because it leaves the drawing window and this is currently configured to not even have perspective