COBURG UNIVERSITY
of applied sciences and arts

Introduction to WebGL

Computer Graphics 1

Prof. Dr.-Ing. Quirin Meyer

WebGL – Infrastructure

- **Web-Browser**
  - Mozilla Firefox
    https://www.mozilla.org/de/firefox/new/
  - Google Chrome
  - …
- **Editor**
  - Visual Studio Code
    https://code.visualstudio.com/
  - Emacs
  - Vim
  - …
- **Web-Server**
  - Visual Studio Code Plugin *Live Server*
  - node.js
    https://nodejs.org
  - Python
  - …
- I use/recommend
  - Firefox, Visual Studio Code, Live-Server Plugin

Overview – Computer System

Lenovo CS/IdeaCentre T540/R53600/16GB/512/W10H

Hersteller- / Modellnummer: 90L5002JGE

AMD Ryzen™ 5 3600 (6C / 12T, 3.6 / 4.2GHz, 3MB L2 / 32MB L3) / NVIDIA GeForce GTX 1650 SUPER 4GB GDDR6 / 2x 8GB DIMM DDR4 / 512GB SSD M.2 2280 NVMe / Windows® 10 Home 64

★★★★★ (4)

✓ Mit hoher Leistung ganz weit vorn
✓ Mit leistungsstarker Grafik ganz weit vorn

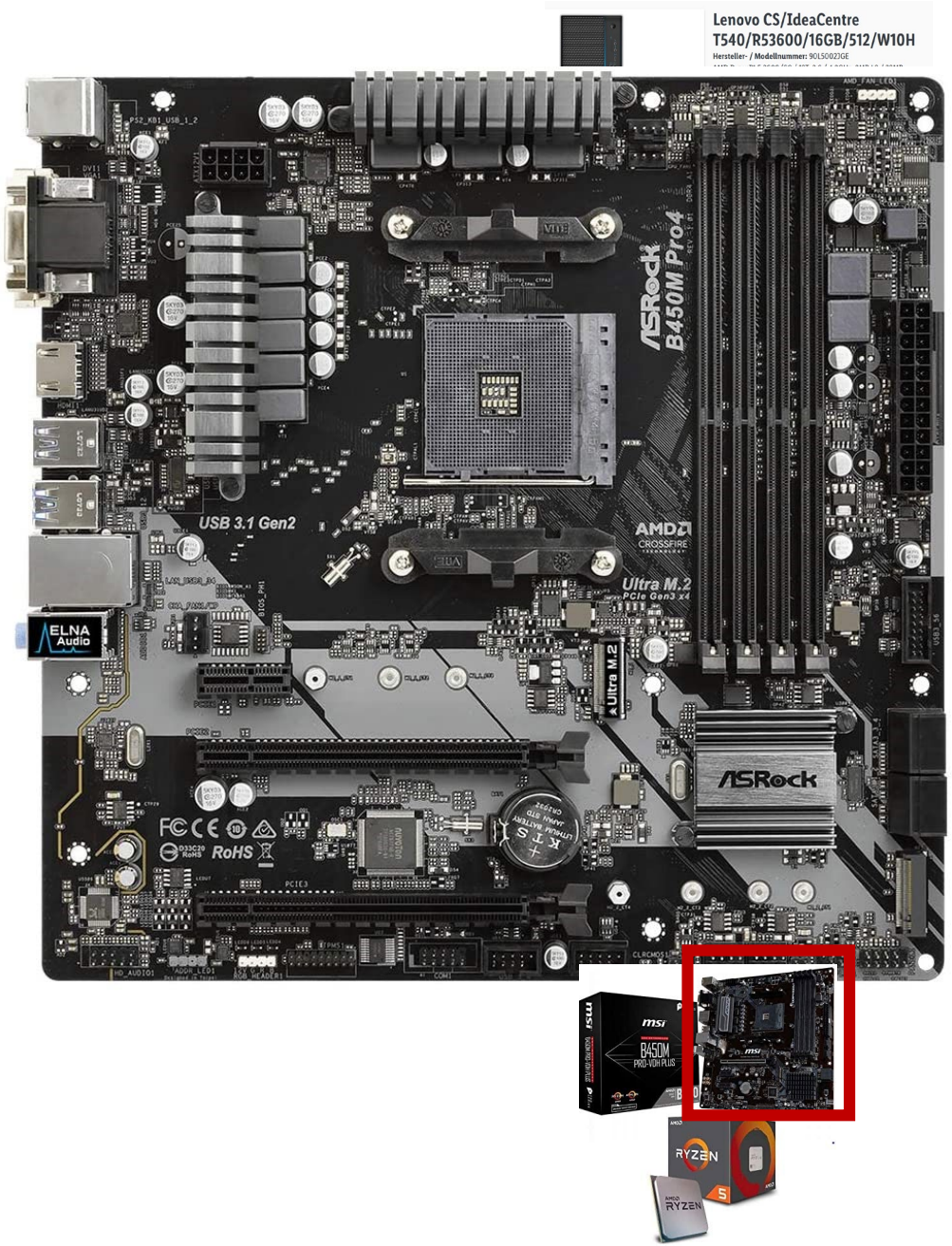
Weitere Produktdetails

Overview – Computer System

Mainboard

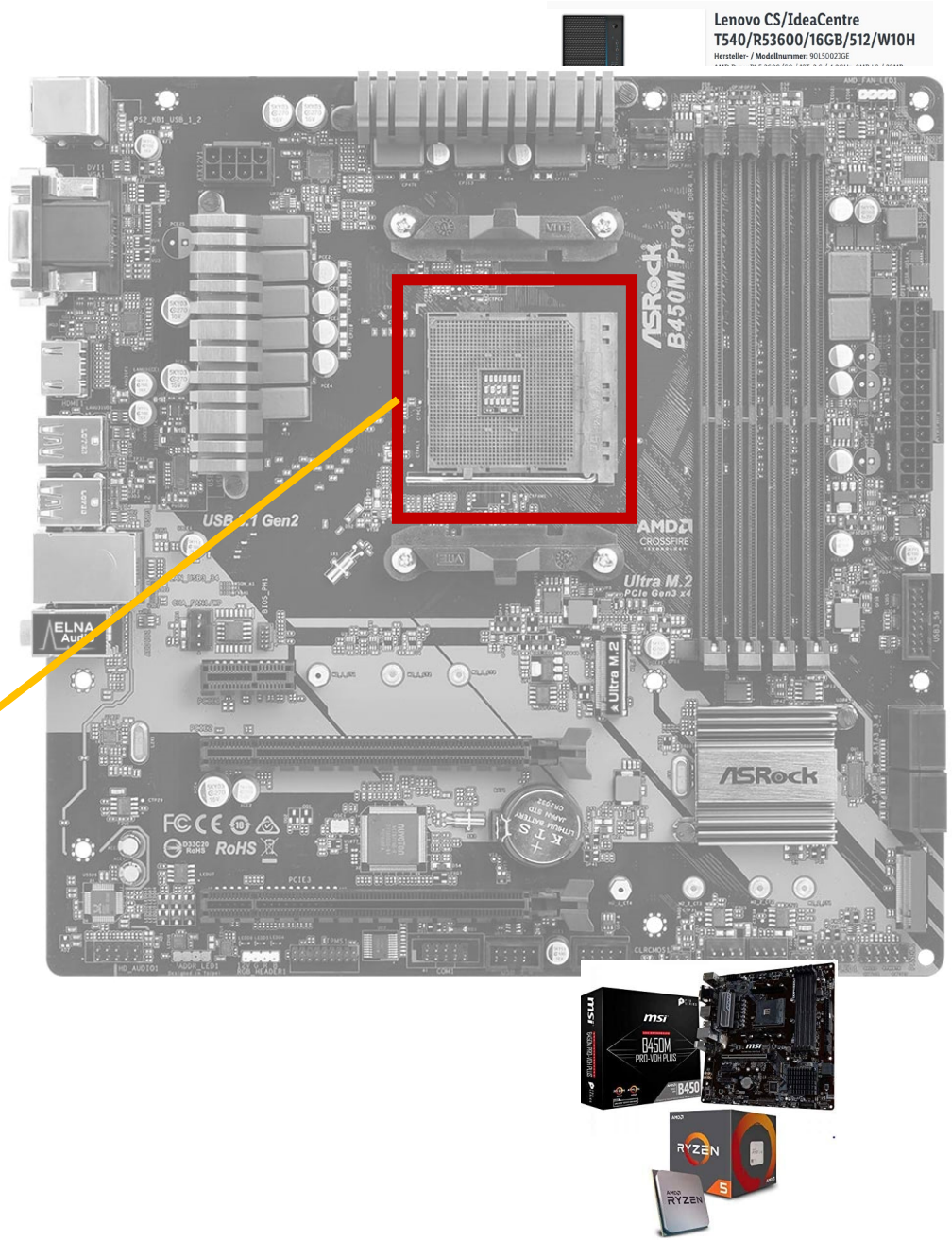
Overview – Computer System

Mainboard



Overview – Computer System

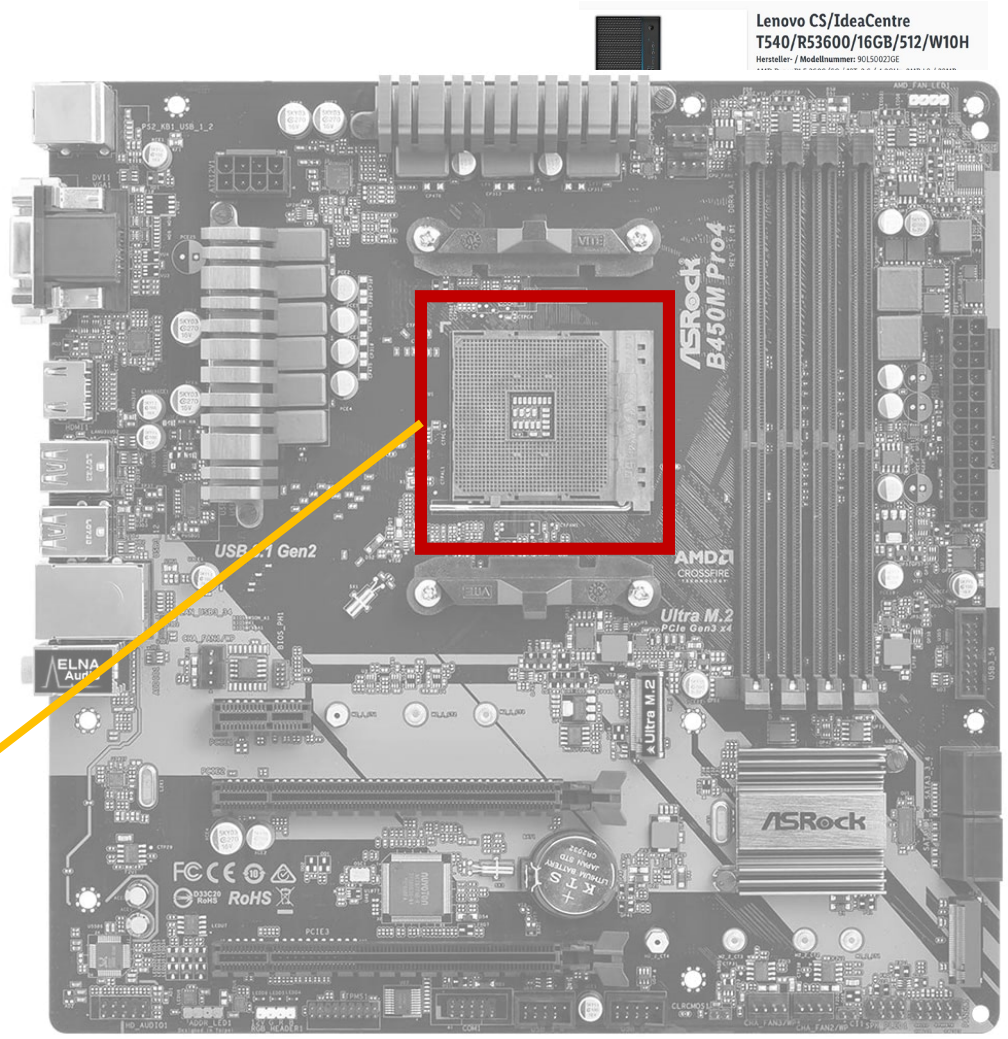
Mainboard



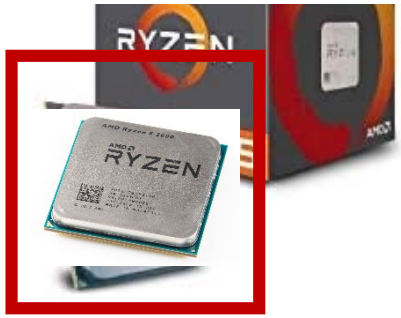
Overview – Computer System

Mainboard

Central Processing Unit
(CPU)



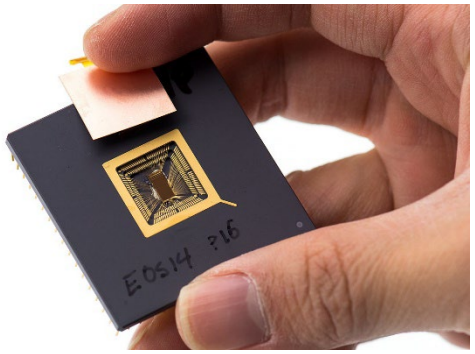
CPU Socket



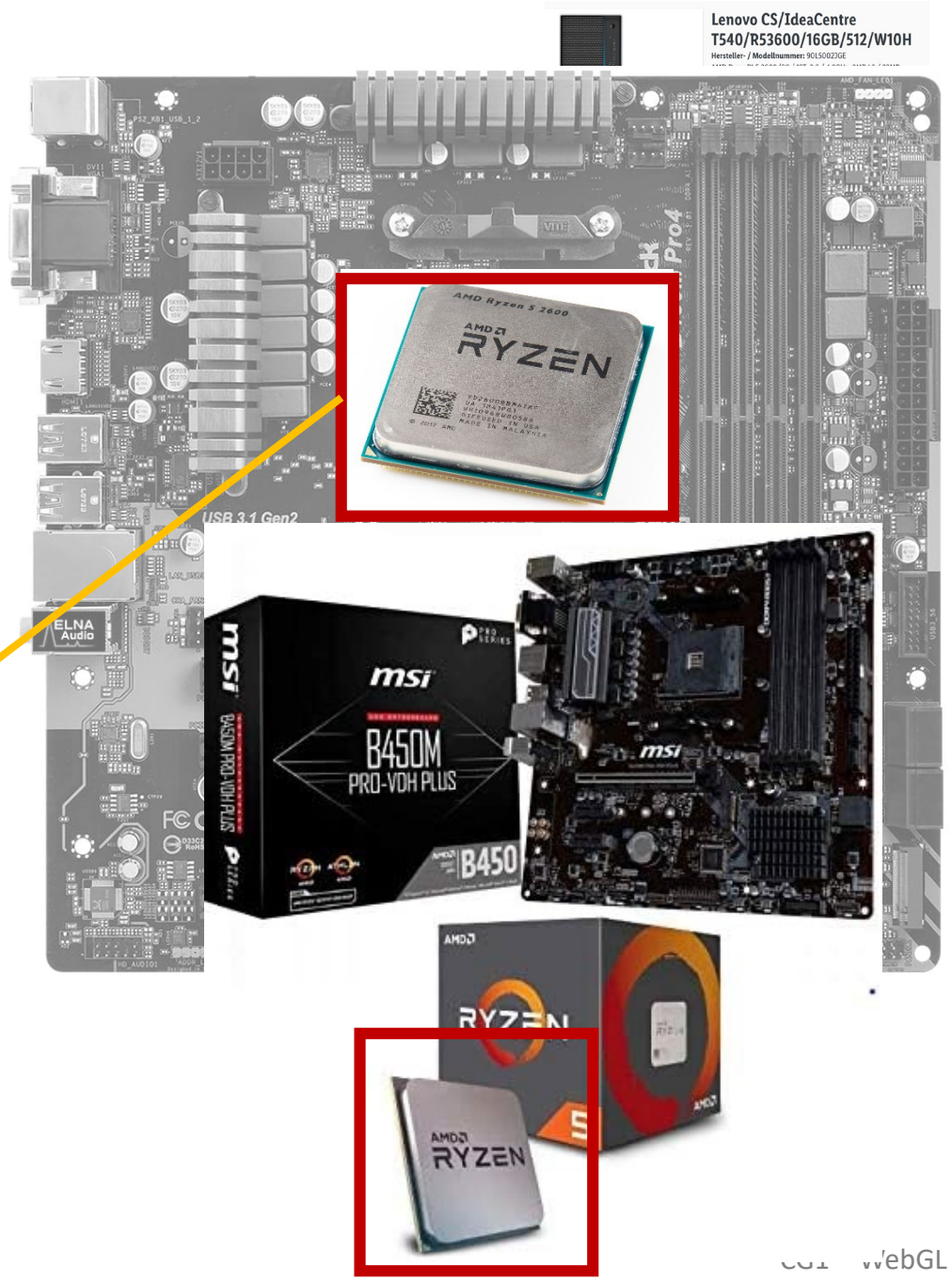
Overview – Computer System

Mainboard

Central Processing Unit
(CPU)



CPU



Overview – Computer System

Mainboard

Central Processing Unit (CPU)

Lenovo CS/IdeaCentre T540/R53600/16GB/512/W10H

Hersteller- / Modellnummer: 90L5002JGE

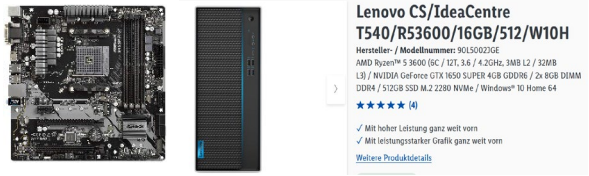
AMD Ryzen™ 5 3600 (6C / 12T, 3.6 / 4.2GHz, 3MB L2 / 32MB L3) / NVIDIA GeForce GTX 1650 SUPER 4GB GDDR6 / 2x 8GB DIMM DDR4 / 512GB SSD M.2 2280 NVMe / Windows® 10 Home 64

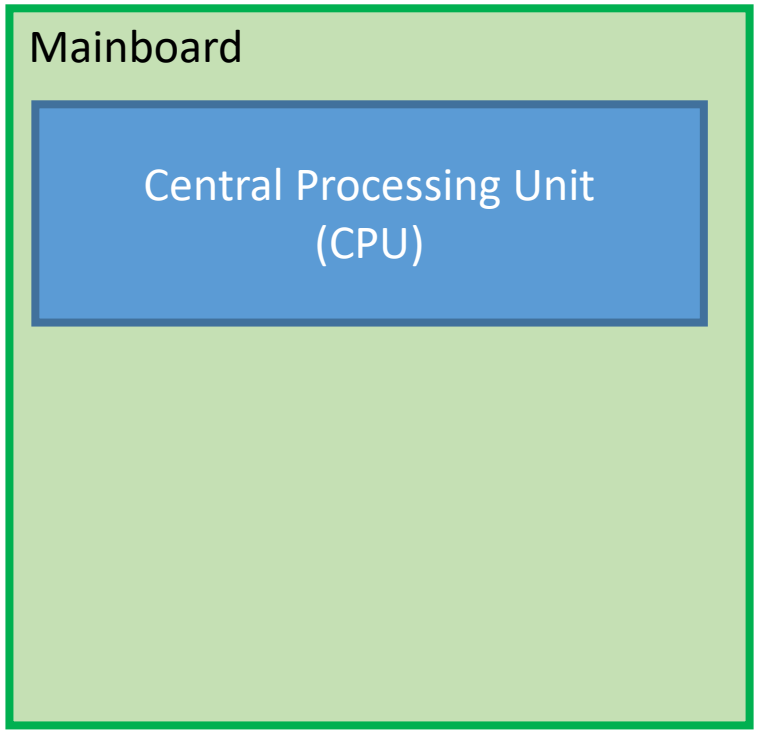
★★★★★ (4)

✓ Mit hoher Leistung ganz weit vorn
✓ Mit leistungsstarker Grafik ganz weit vorn

Weitere Produktdetails

Overview – Computer System

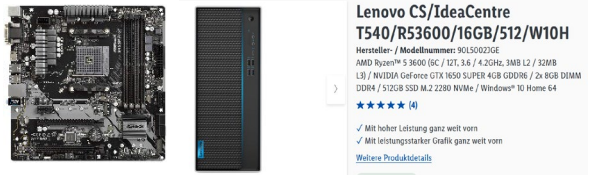


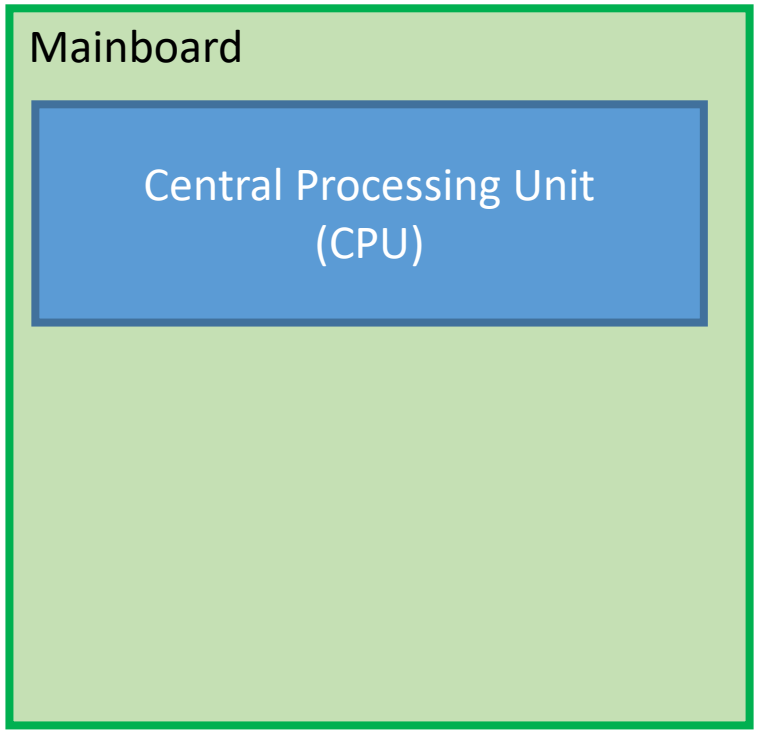


Companies that design CPUs

- Intel, AMD
  - Mostly Desktop, Workstation, Servers
- ARM
  - Mostly Embedded
    - Smartphones
    - Cars
    - Washing machines
- …

Overview – Computer System





CPU

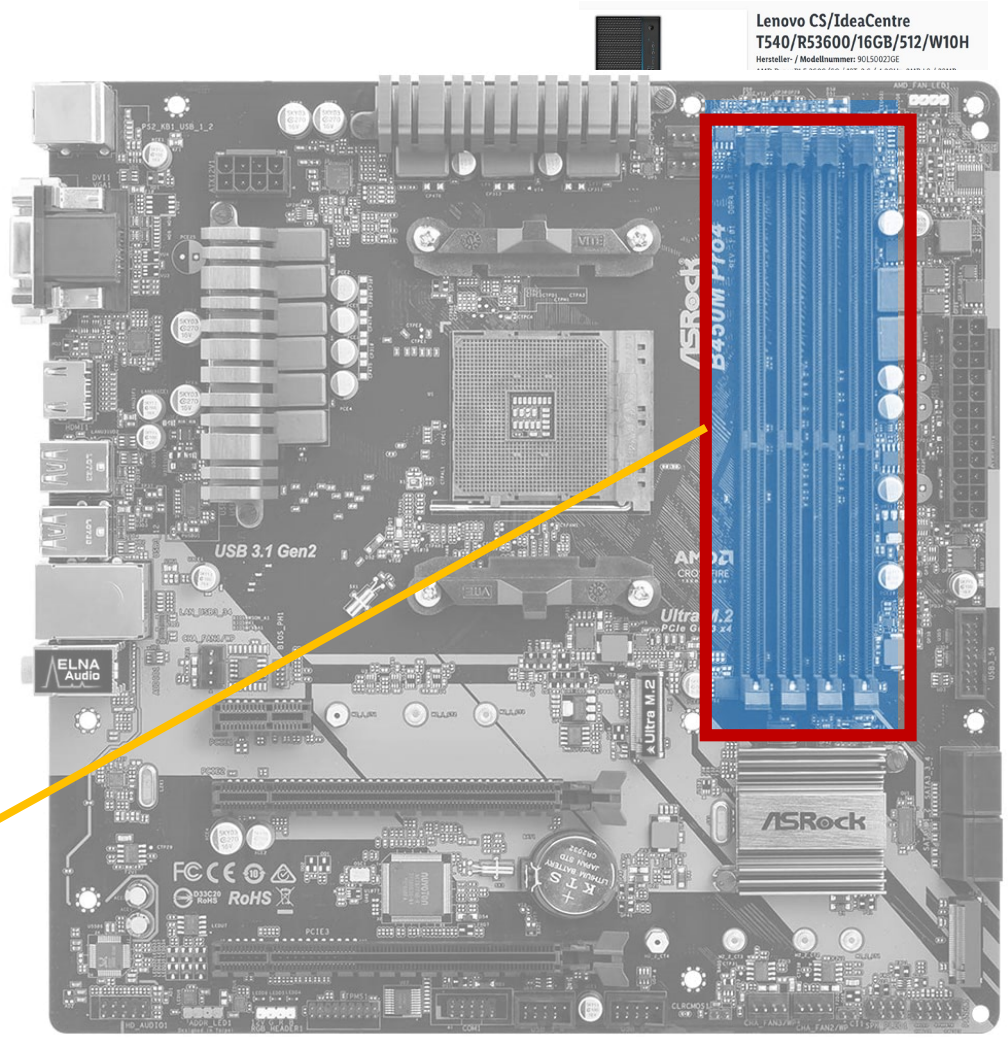
- Executes the code that you have written so far

  (in Java, Processing, C++, etc.)

Overview – Computer System

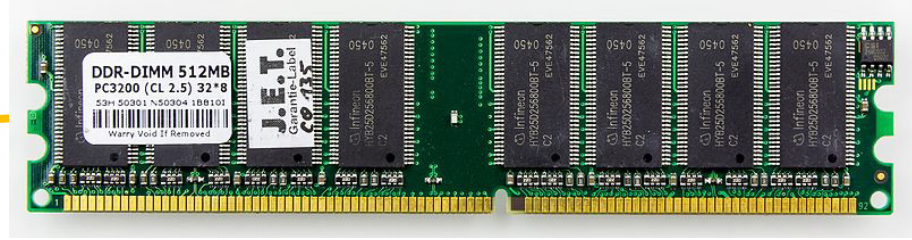
Mainboard

Central Processing Unit
(CPU)

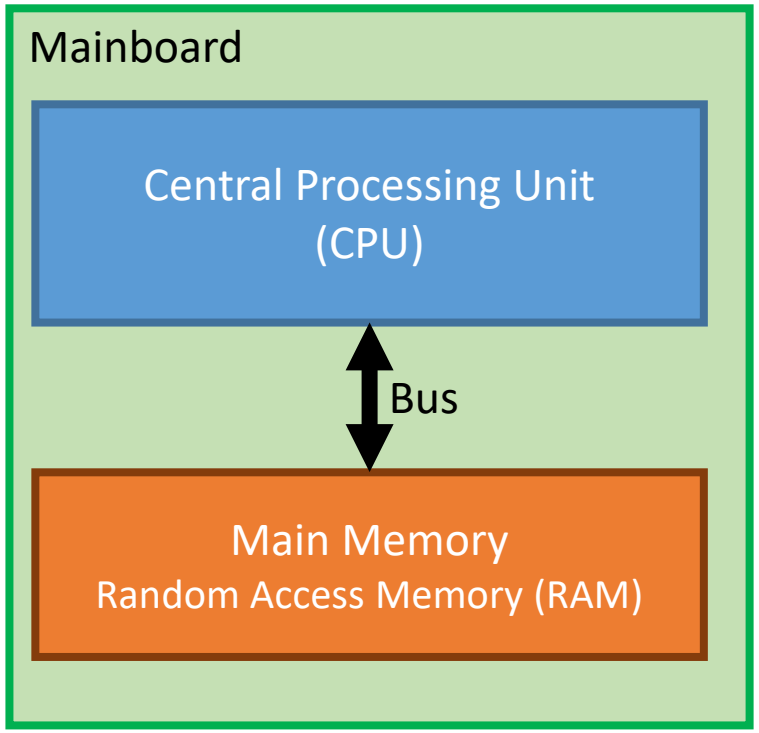


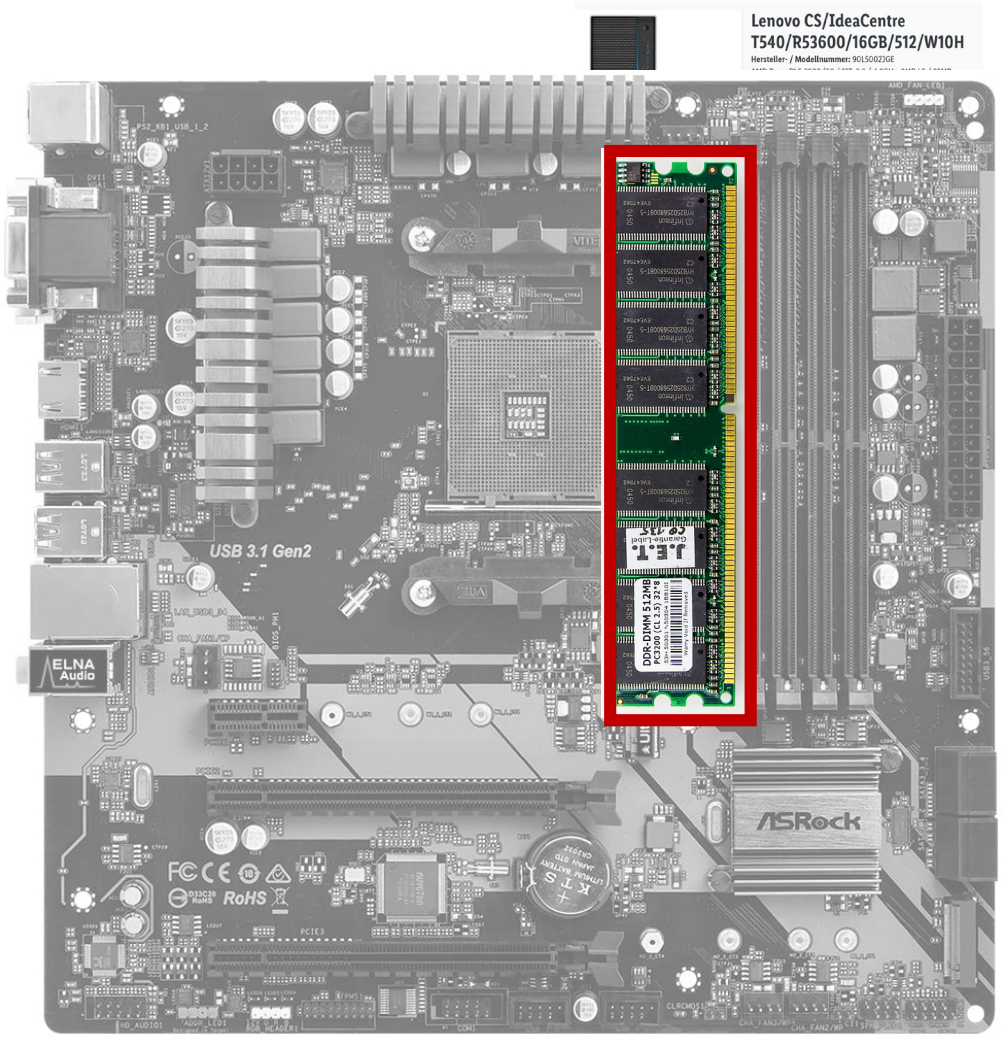
RAM Slots

RAM Module

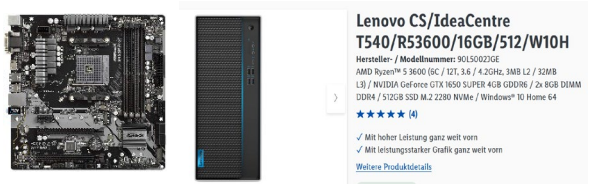


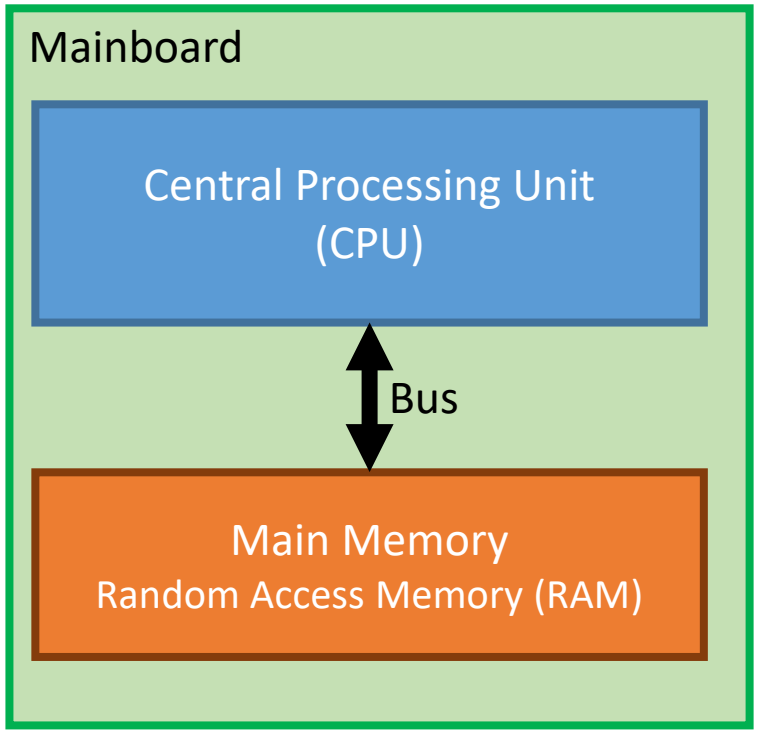
Overview – Computer System





Overview – Computer System





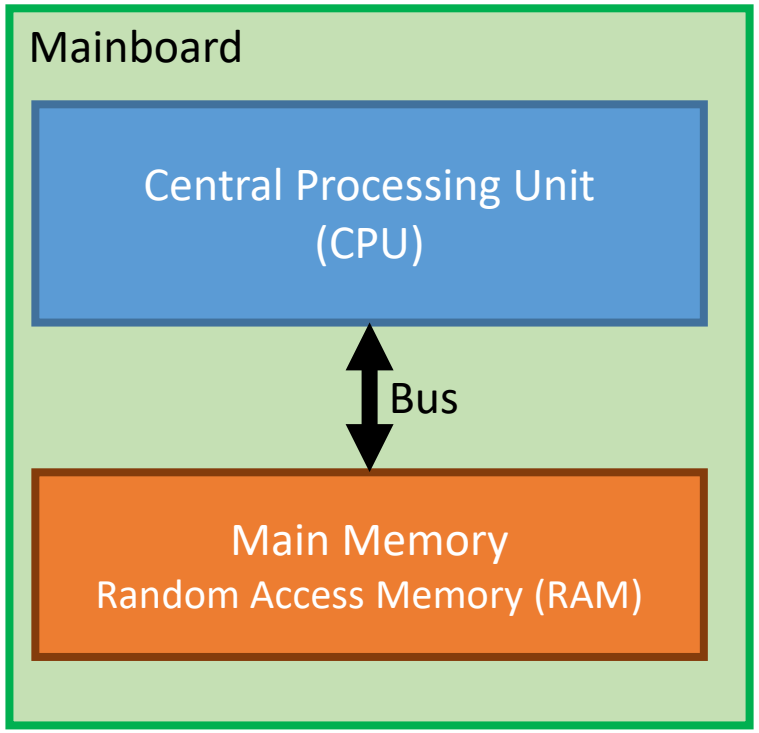
Main Memory (Primary Memory)

- Random Access Memory (RAM)
  - Big Array of bytes
  - Read, Write, at any location
- Connect over CPU over bus
- Java: This where the variables, objects are located
- Data is lost upon power-off

Bus

- Data Lines to transfer data from CPU to RAM

Overview – Computer System



Lenovo CS/IdeaCentre T540/R53600/16GB/512/W10H

Hersteller- / Modellnummer: 90L5002JGE

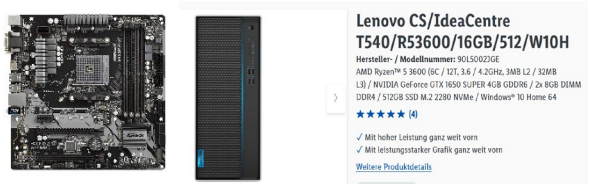
AMD Ryzen™ 5 3600 (6C / 12T, 3.6 / 4.2GHz, 3MB L2 / 32MB L3) / NVIDIA GeForce GTX 1650 SUPER 4GB GDDR6 / 2x 8GB DIMM DDR4 / 512GB SSD M.2 2280 NVMe / Windows® 10 Home 64

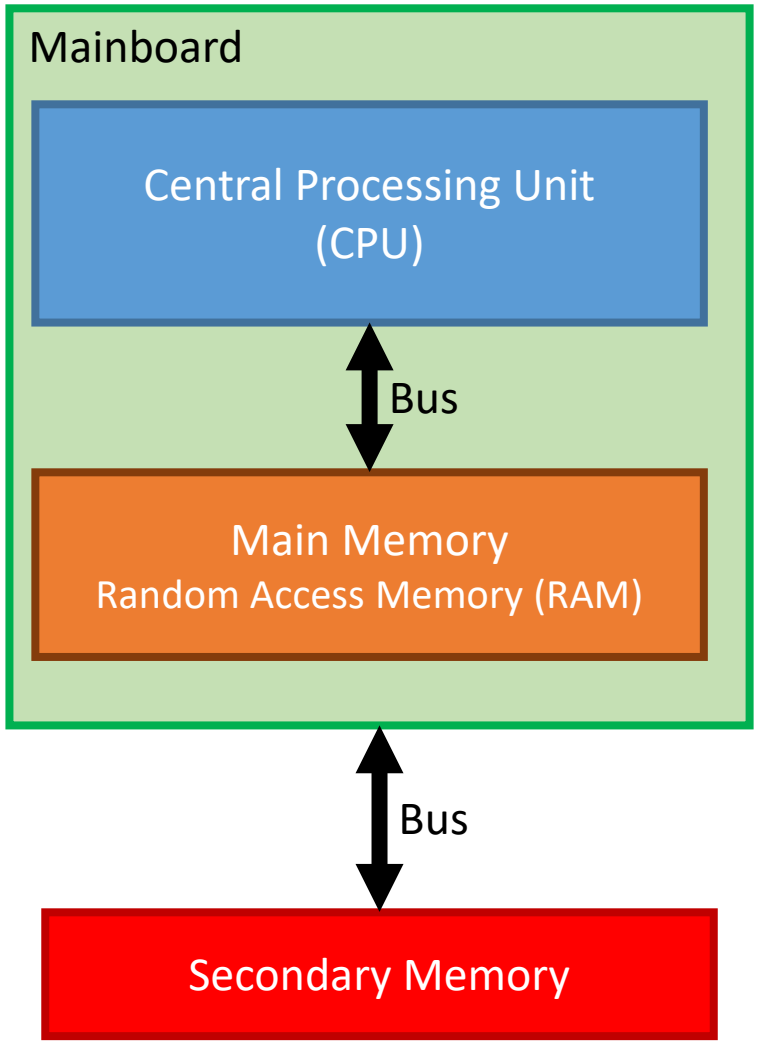
★★★★★ (4)

✓ Mit hoher Leistung ganz weit vorn
✓ Mit leistungsstarker Grafik ganz weit vorn

Weitere Produktdetails

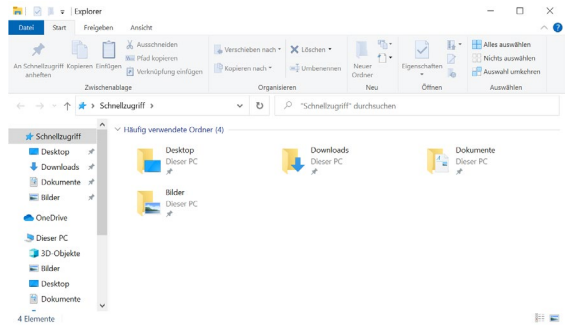
Overview – Computer System





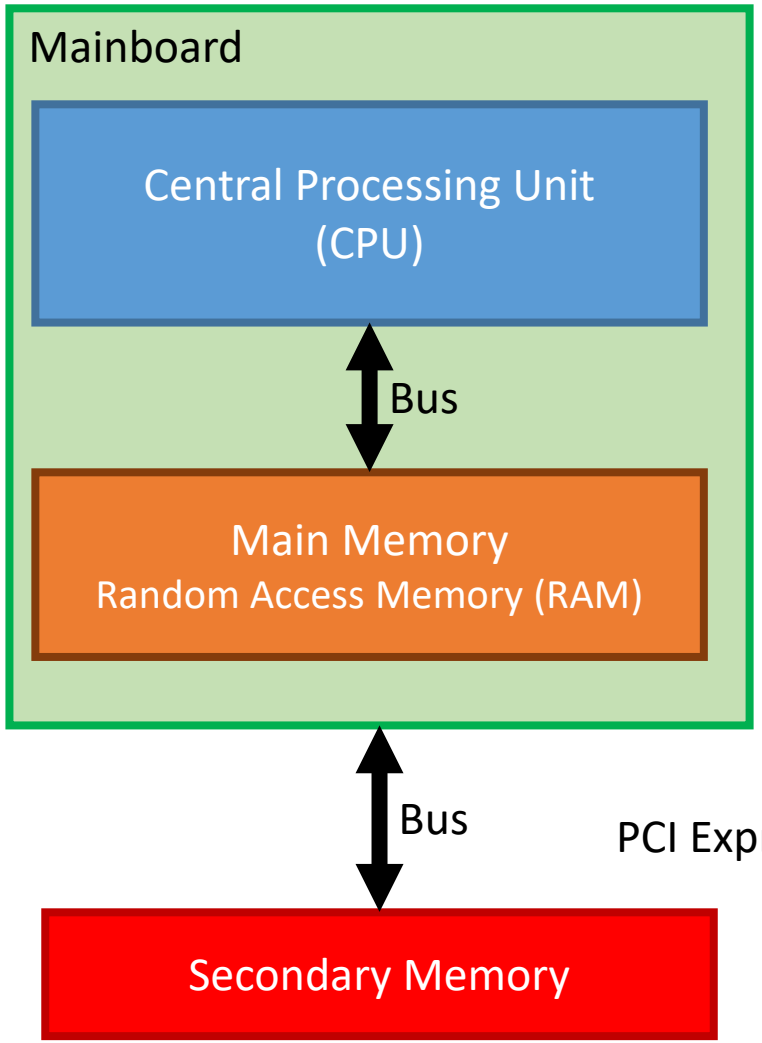
Secondary Memory

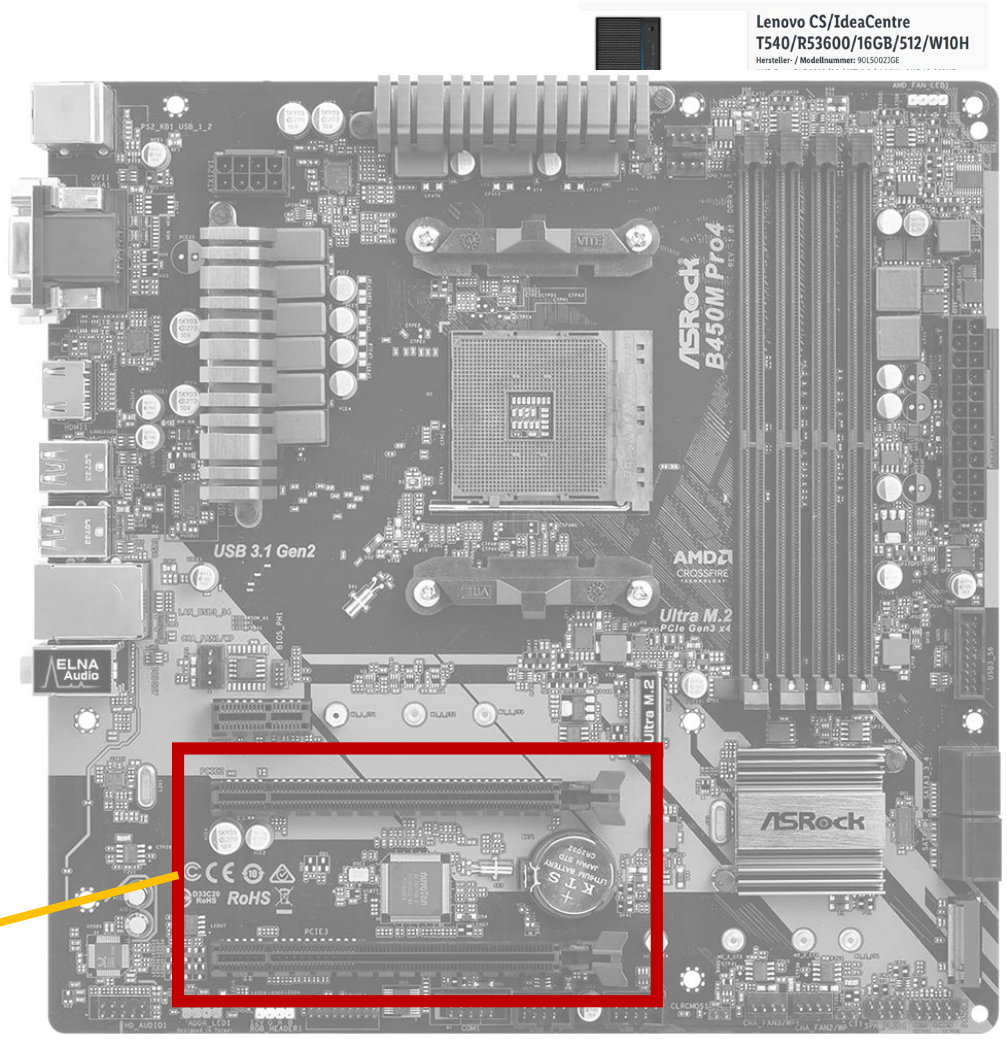
- Stores Files Persistently
  - Java files, Exe Files, JPEG Files, …
  - Browse in a File Browser
- Technologies
  - Hard Disk Drives (HDD)
  - Solid State Drives (SSD)



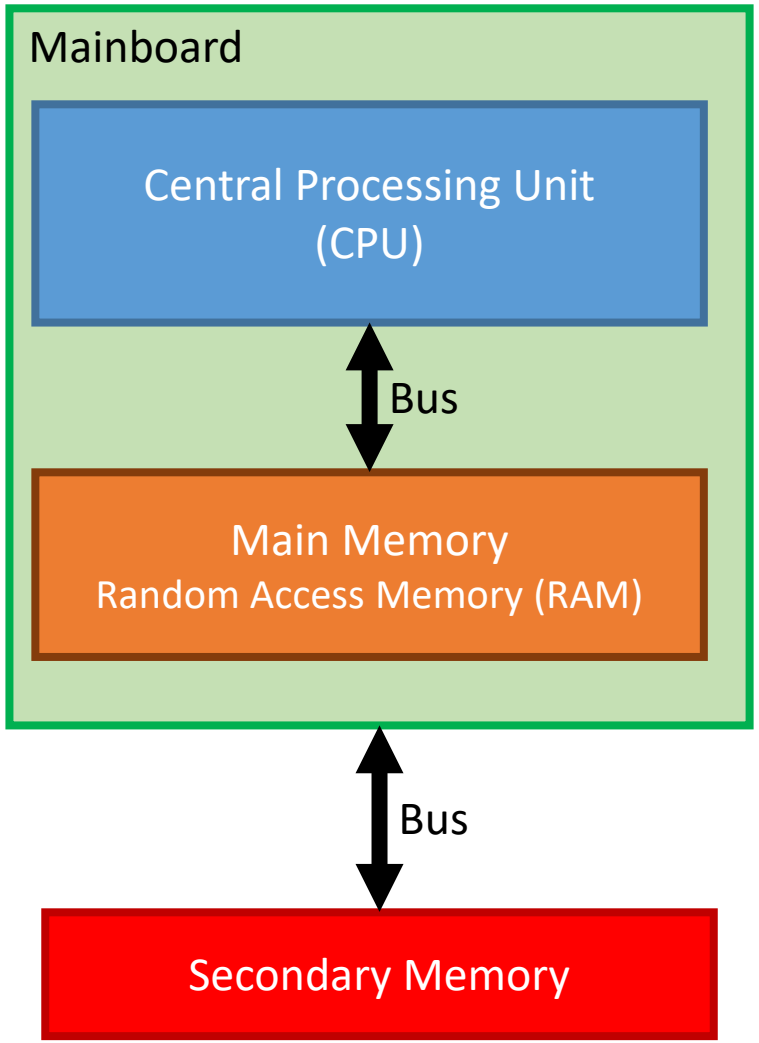
By Evan-Amos - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=27940250

Overview – Computer System

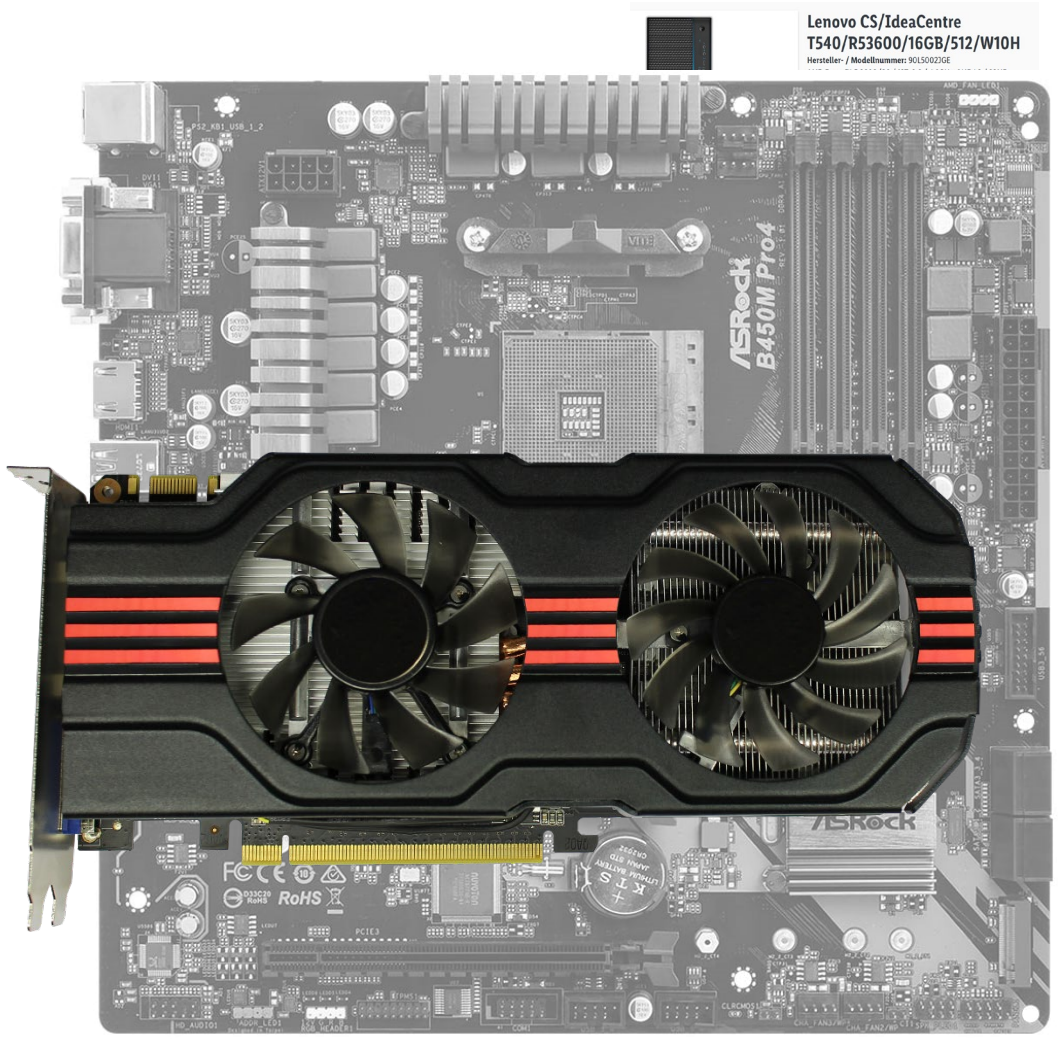




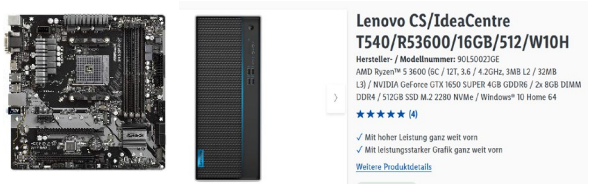
Overview – Computer System

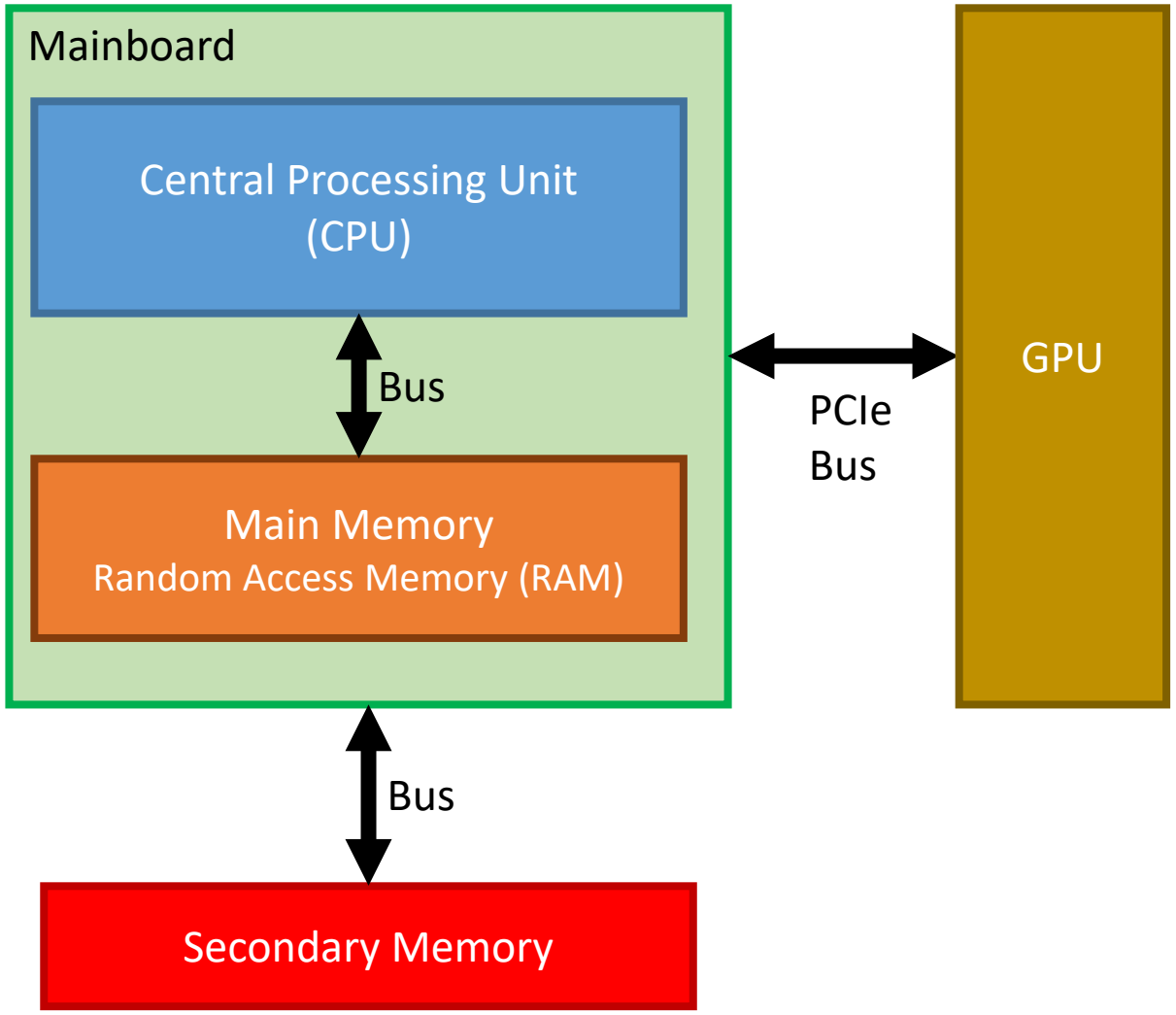


GPU



Overview – Computer System

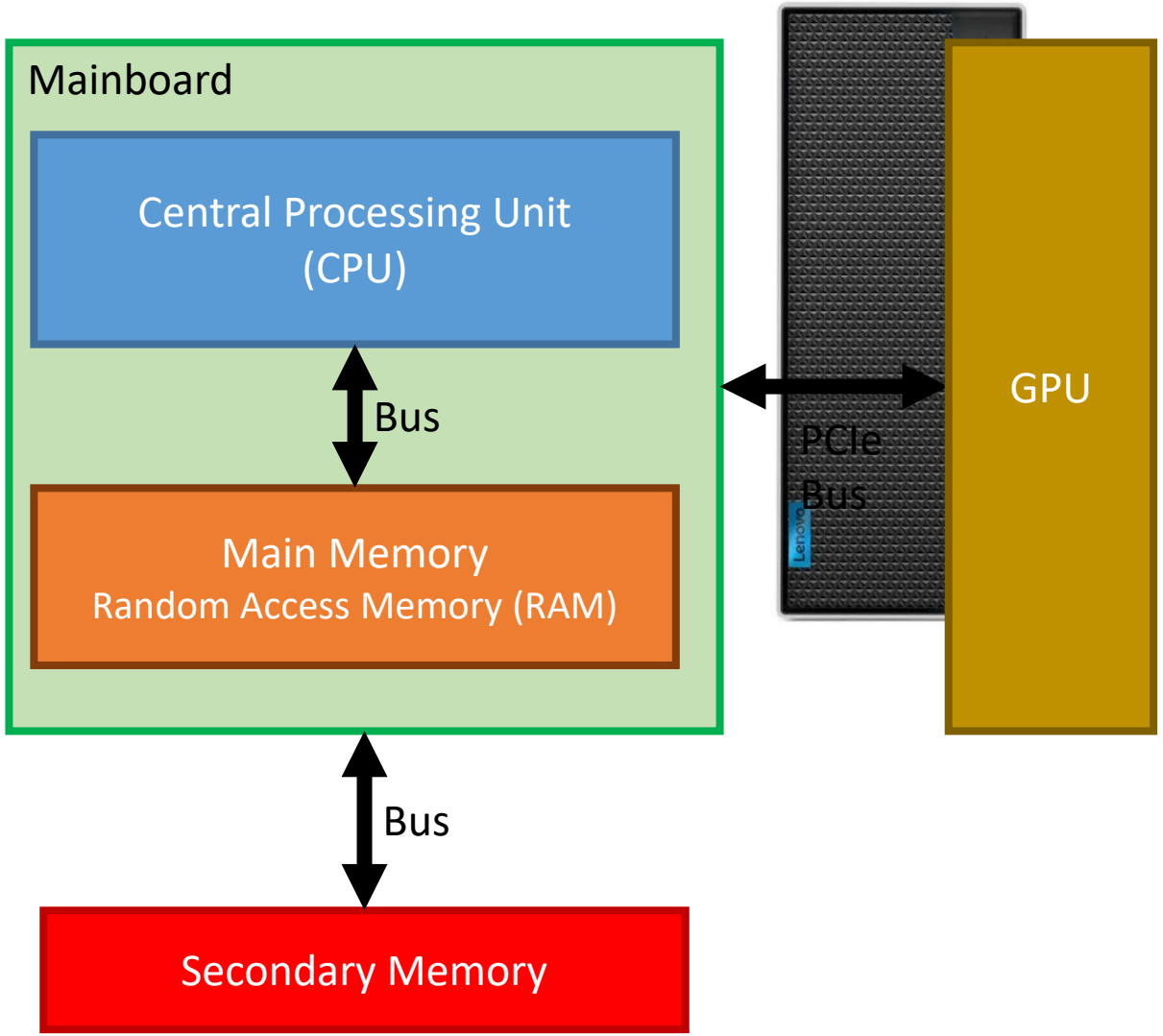




Graphics Processing Unit (GPU)

- Connected over PCI Express Bus with Mainboard
- PCI: Peripheral Component Interconnect Express
- GPU: Graphics Output
- Co-Processor

Overview – Computer System



Lenovo CS/IdeaCentre T540/R53600/16GB/512/W10H

Hersteller- / Modellnummer: 90L5002JGE

AMD Ryzen™ 5 3600 (6C / 12T, 3.6 / 4.2GHz, 3MB L2 / 32MB L3) / NVIDIA GeForce GTX 1650 SUPER 4GB GDDR6 / 2x 8GB DIMM DDR4 / 512GB SSD M.2 2280 NVMe / Windows® 10 Home 64

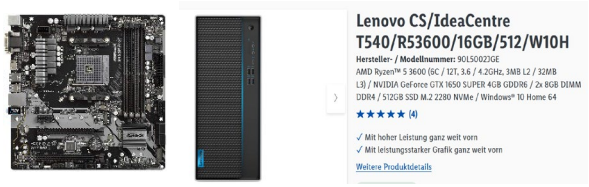
★★★★★ (4)

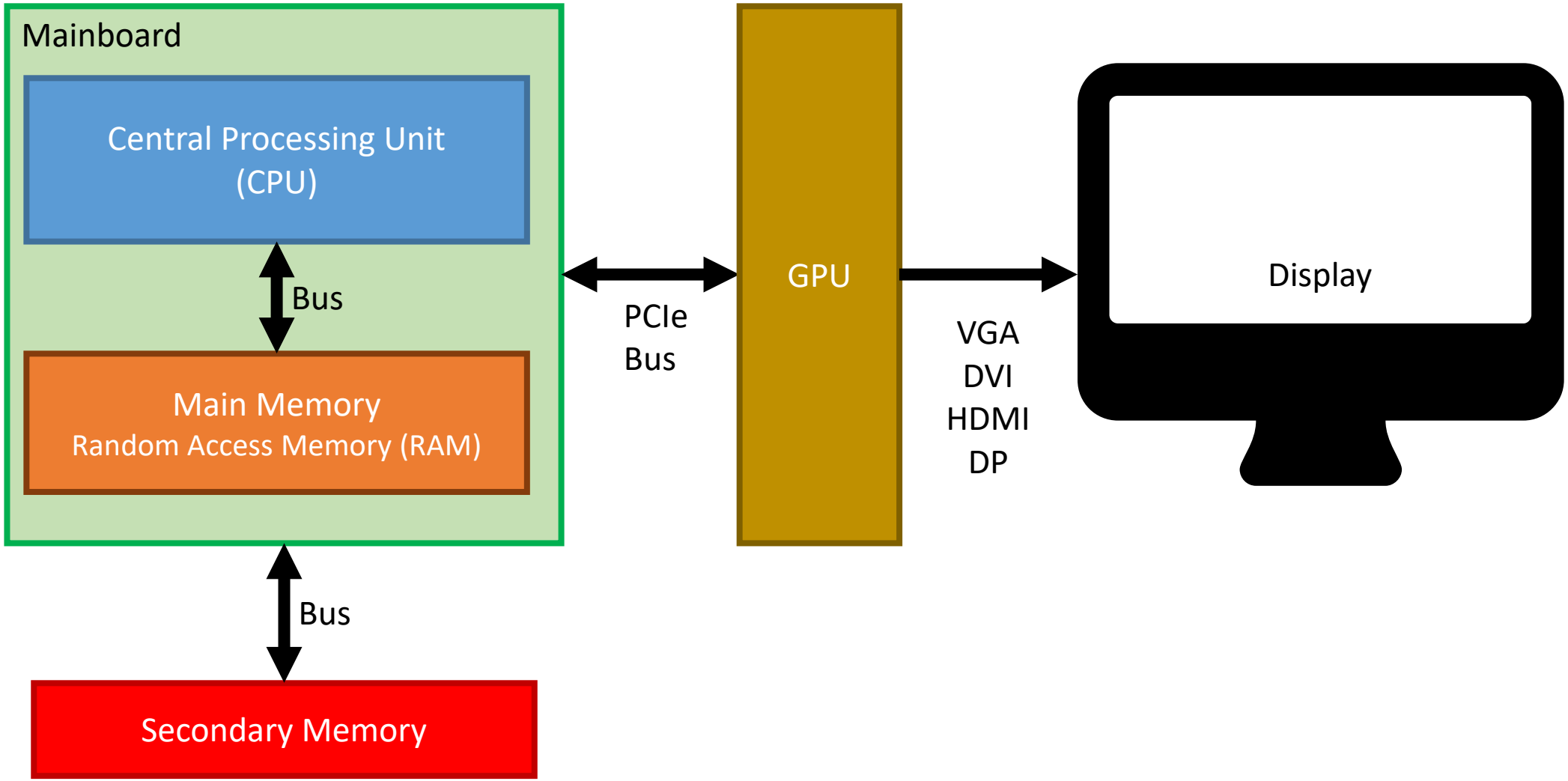
✓ Mit hoher Leistung ganz weit vorn

✓ Mit leistungsstarker Grafik ganz weit vorn

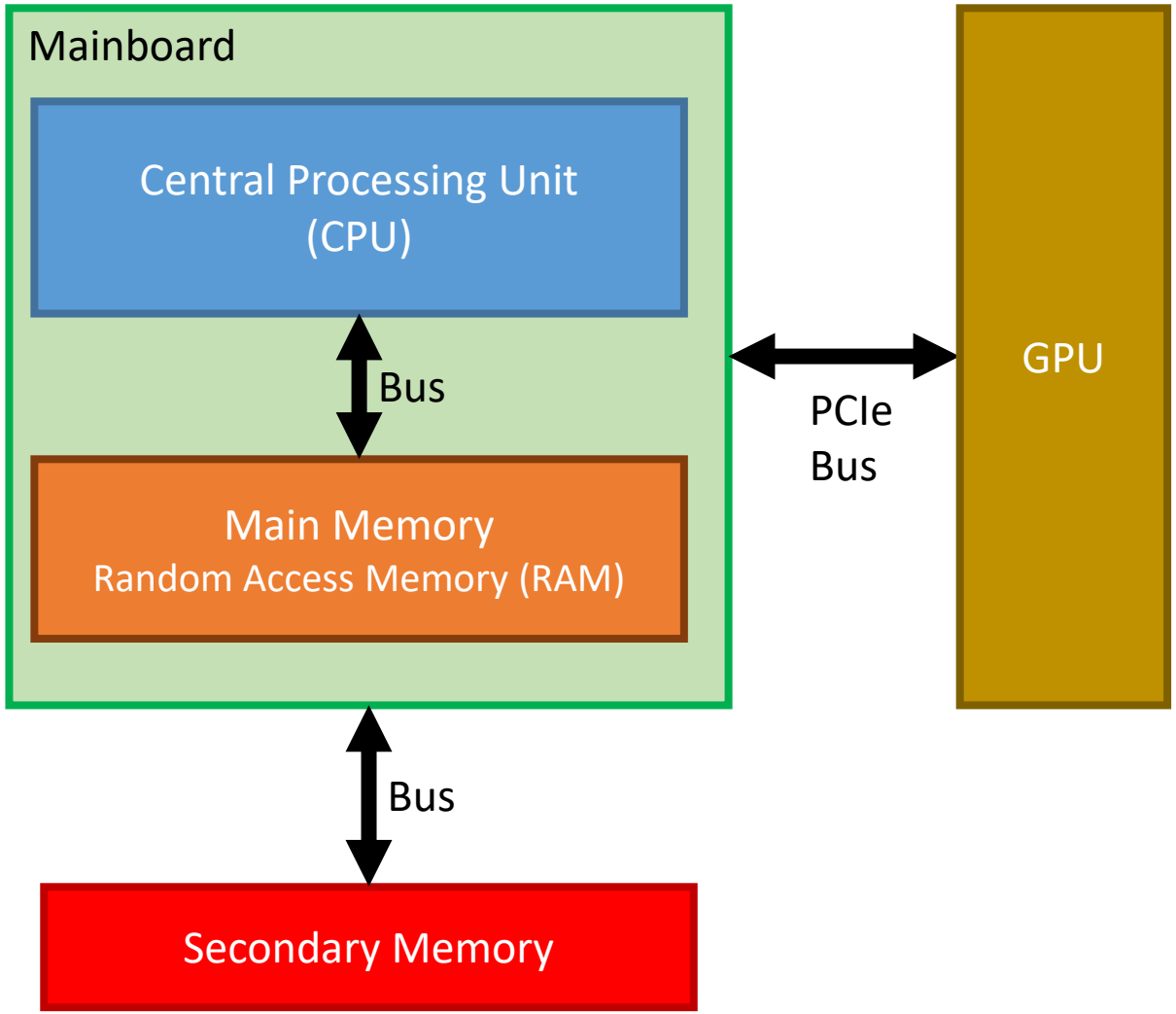
Weitere Produktdetails

Overview – Computer System





Overview – Computer System



GPU Programming

- API

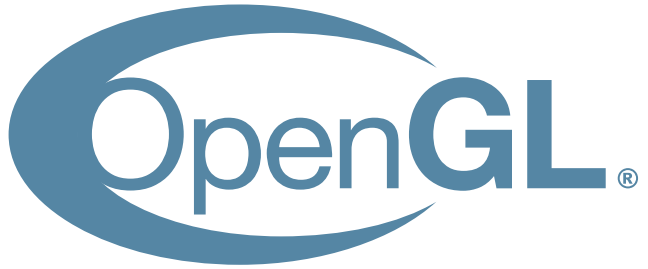
  (Application Programmers Interface)

- **In our course: WebGL**

WebGL

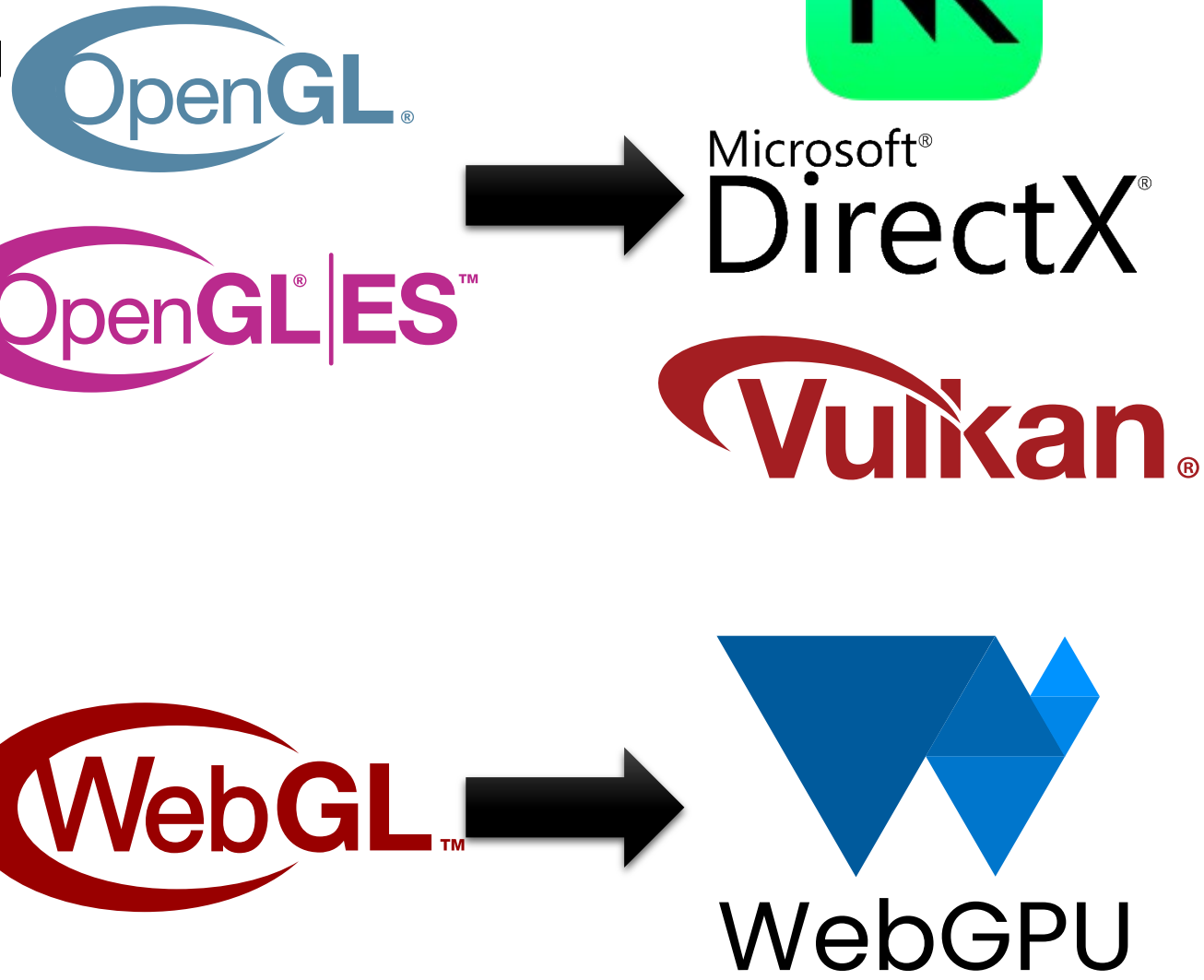
WebGL: Standard by Khronos Group

- Open Standard
- Based on OpenGL Family
  - OpenGL 1992 – 2017
    - Desktop
    - Current Version OpenGL 4.6 (2017)
  - Mobile: OpenGL ES: 2003 – 2015
    - Mobile
    - Current Version: 3.0
- OpenGL, OpenGL|ES: C Bindings
- WebGL: Java Script API for Web



WebGL

- OpenGL and OpenGL|ES: become less used
- Is progressively replaced by verbose APIs
  - Vulkan, DirectX 12 Ultimate, Metal
  - More verbose
  - Less abstract, closer to hardware
  - Harder to learn, harder to program
  - But faster
- WebGL might be replaced by WebGPU
  - Still under development
  - Similar to Vulkan, DirectX



WebGL

- WebGL: JavaScript Interface
- Documentation:
  - Learn JavaScript: https://developer.mozilla.org/en-US/docs/Learn/JavaScript
  - JavaScript Guide: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
  - WebGL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
  - WebGL 2: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API#webgl_2

- JavaScript is pre-requisit.
- Caution: This is no JavaScript course

WebGL – Introductory Project: A Triangle

- Goal for today: Let‘s draw a triangle

WebGL – Introductory Project: A Triangle

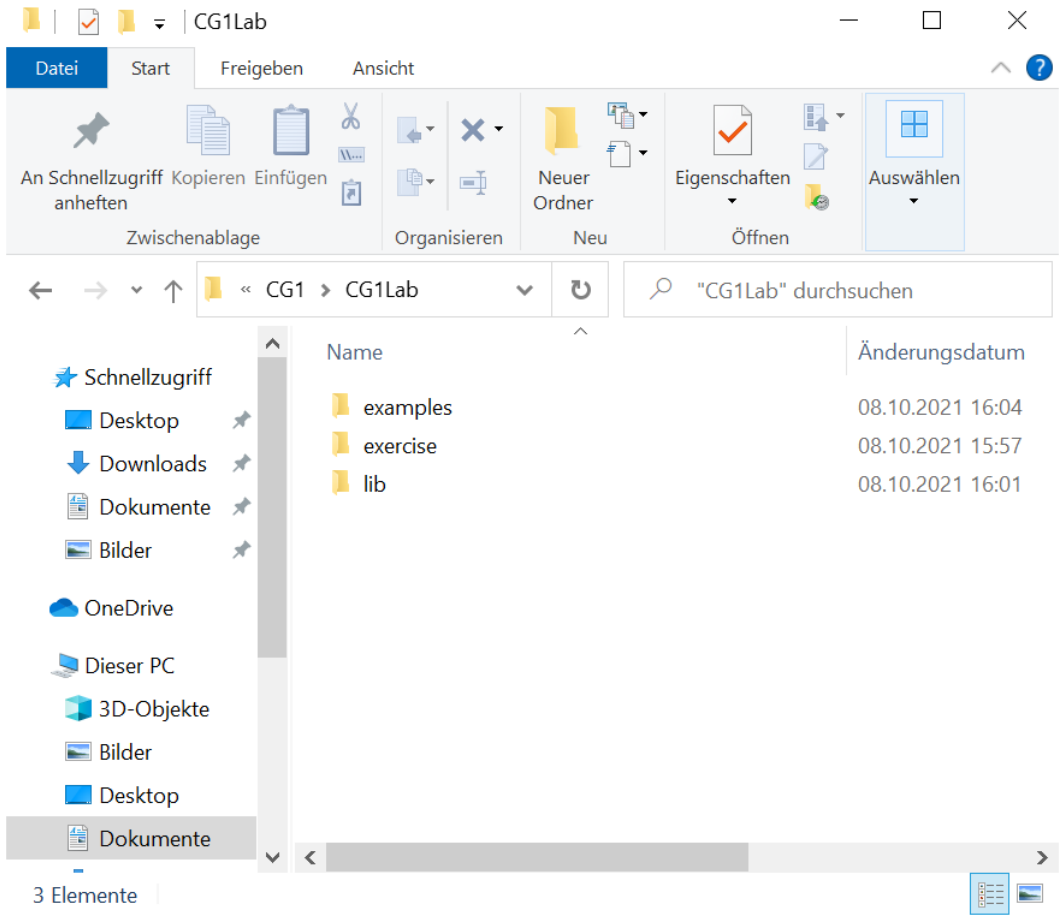
1. Extra CG1Lab.zip to File-System

`<path to lib>`

is the path to the extracted CG1Lab.zip

E.g.

D:\CG1\CG1Lab



WebGL – Introductory Project: A Tria

2. Start Web-Server
    1. Start Node.js command prompt

    2. Install web-server (first time only)

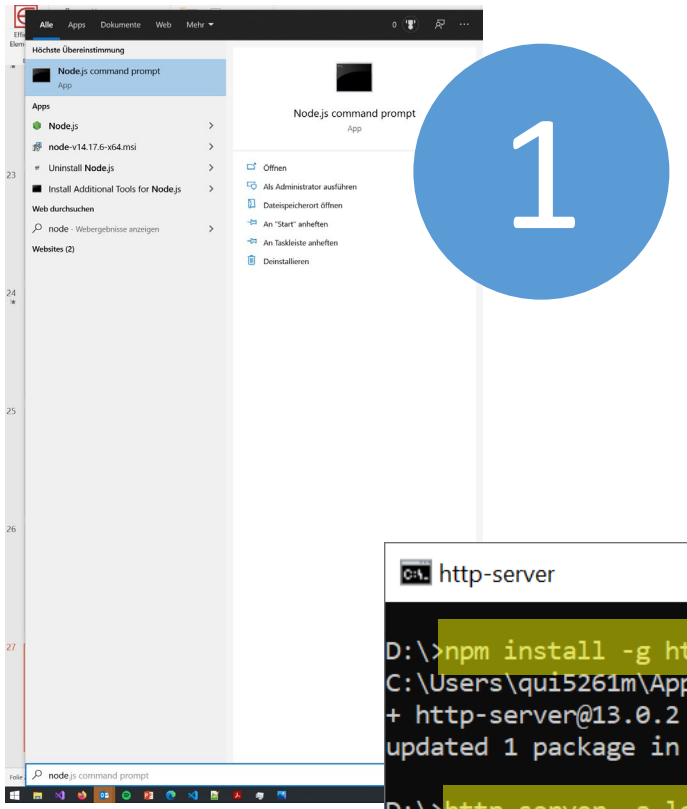
        ```
        npm install –g http-server
        ```

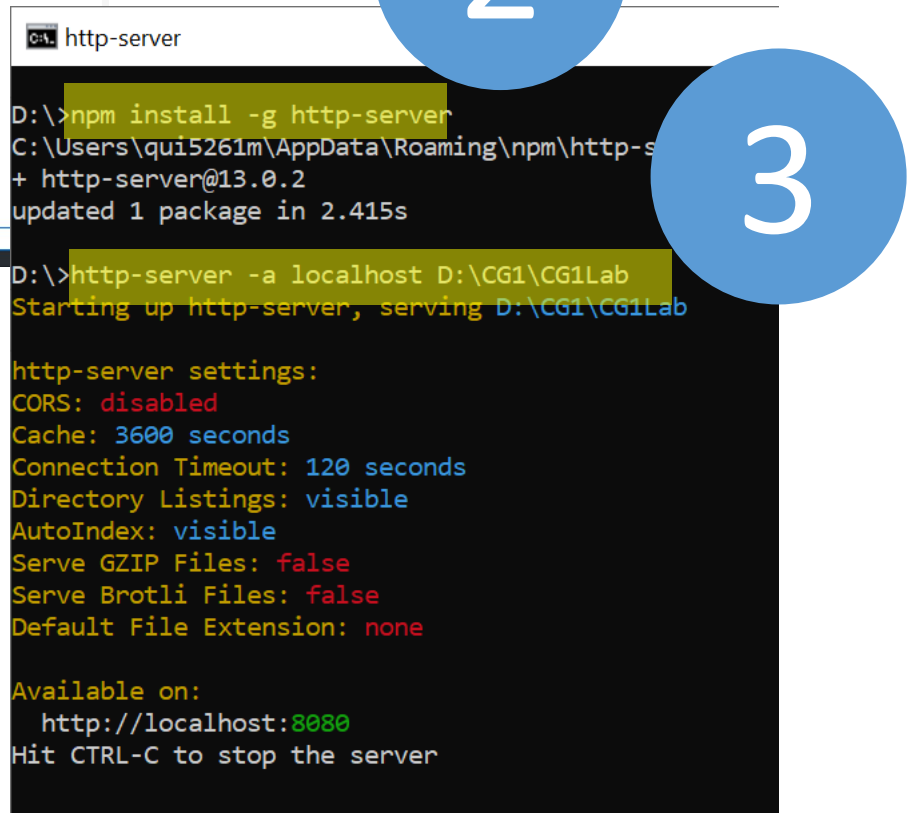
    3. Start Web-Server

        ```
        http-server –c-1 –a localhost <path to lib>
        ```

        e.g.

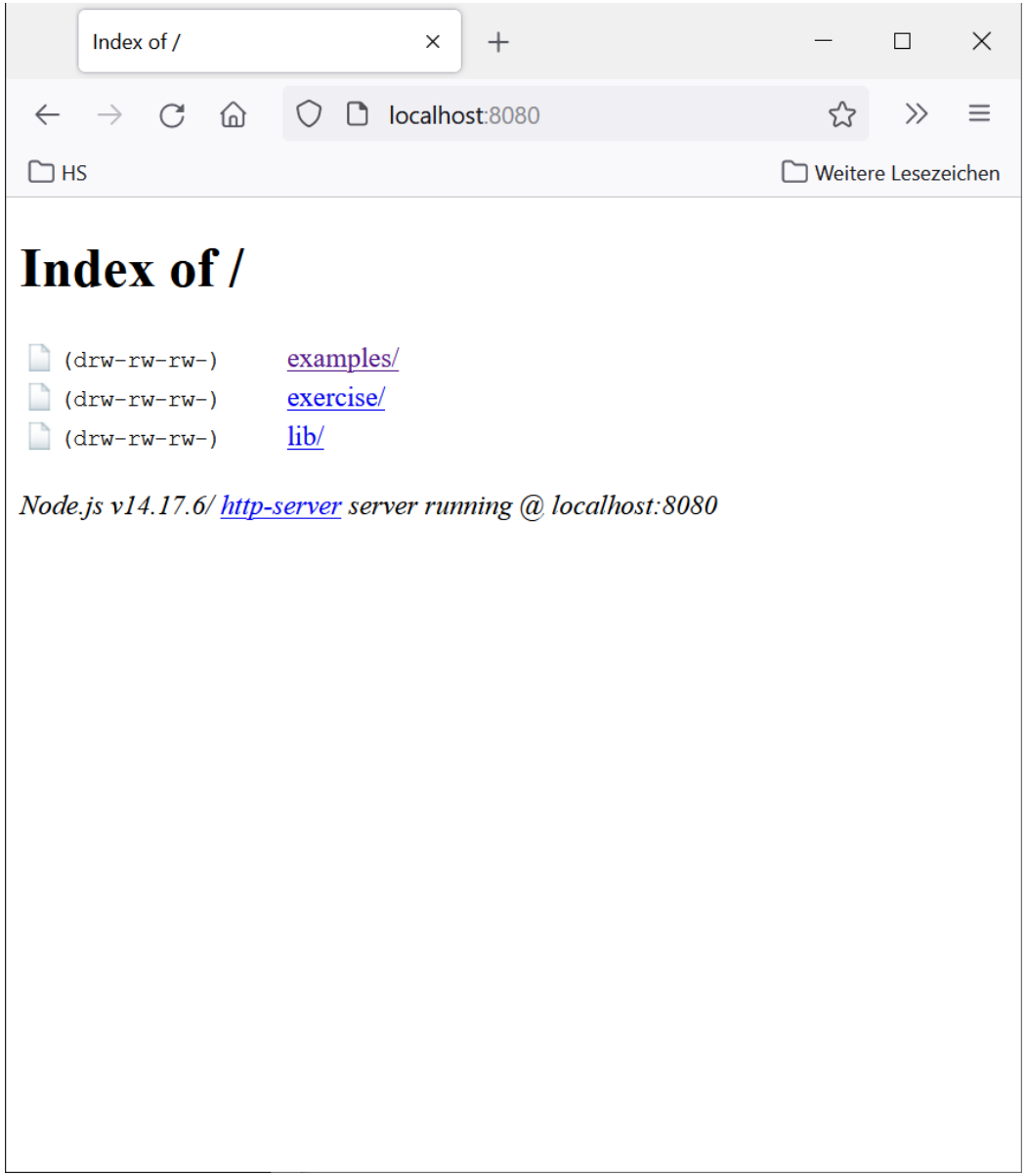
        ```
        http-server –c-1 –a localhost D:\CG1\CG1Lab
        ```





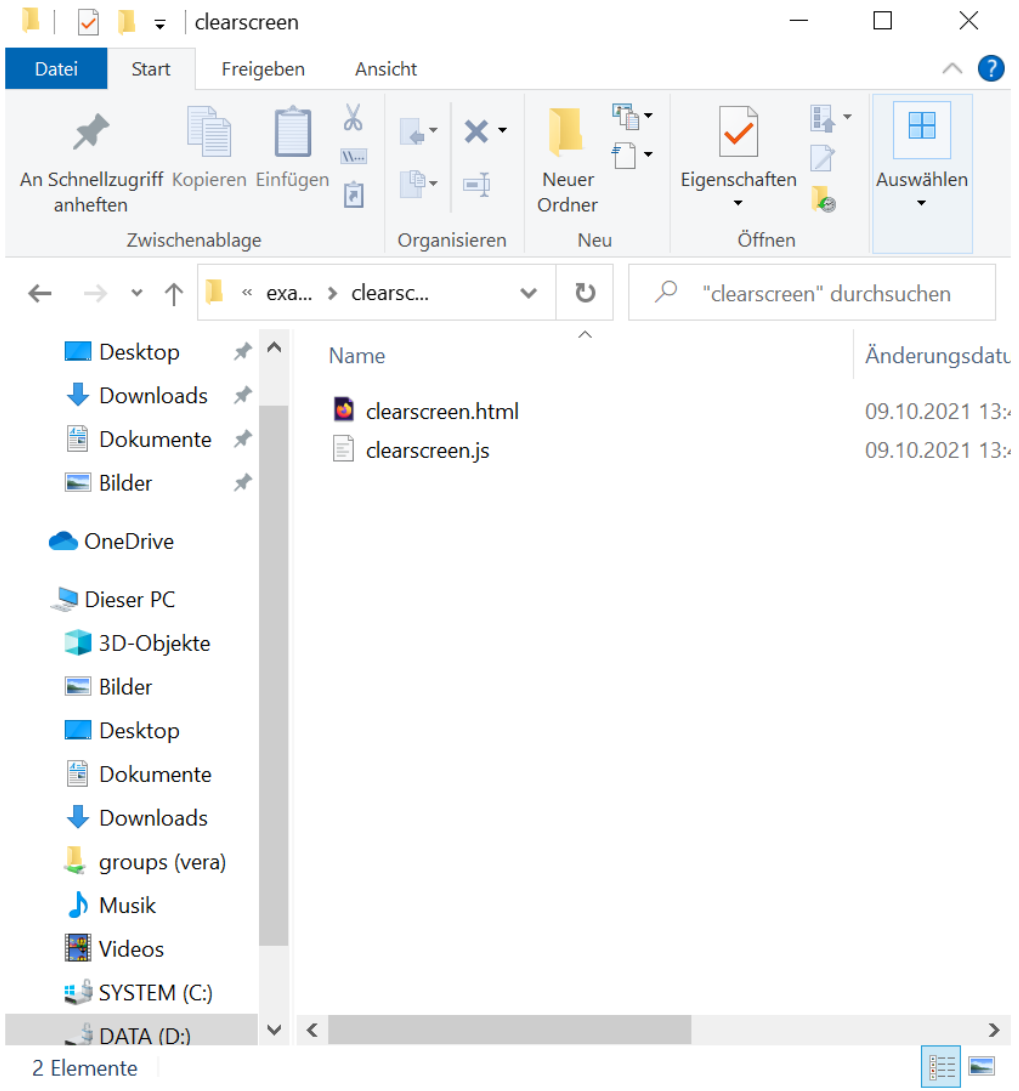
WebGL – Introductory Project: A Triangle

1. Start Webbrowser
2. Browse to http://localhost:8080/



WebGL – Project Structure

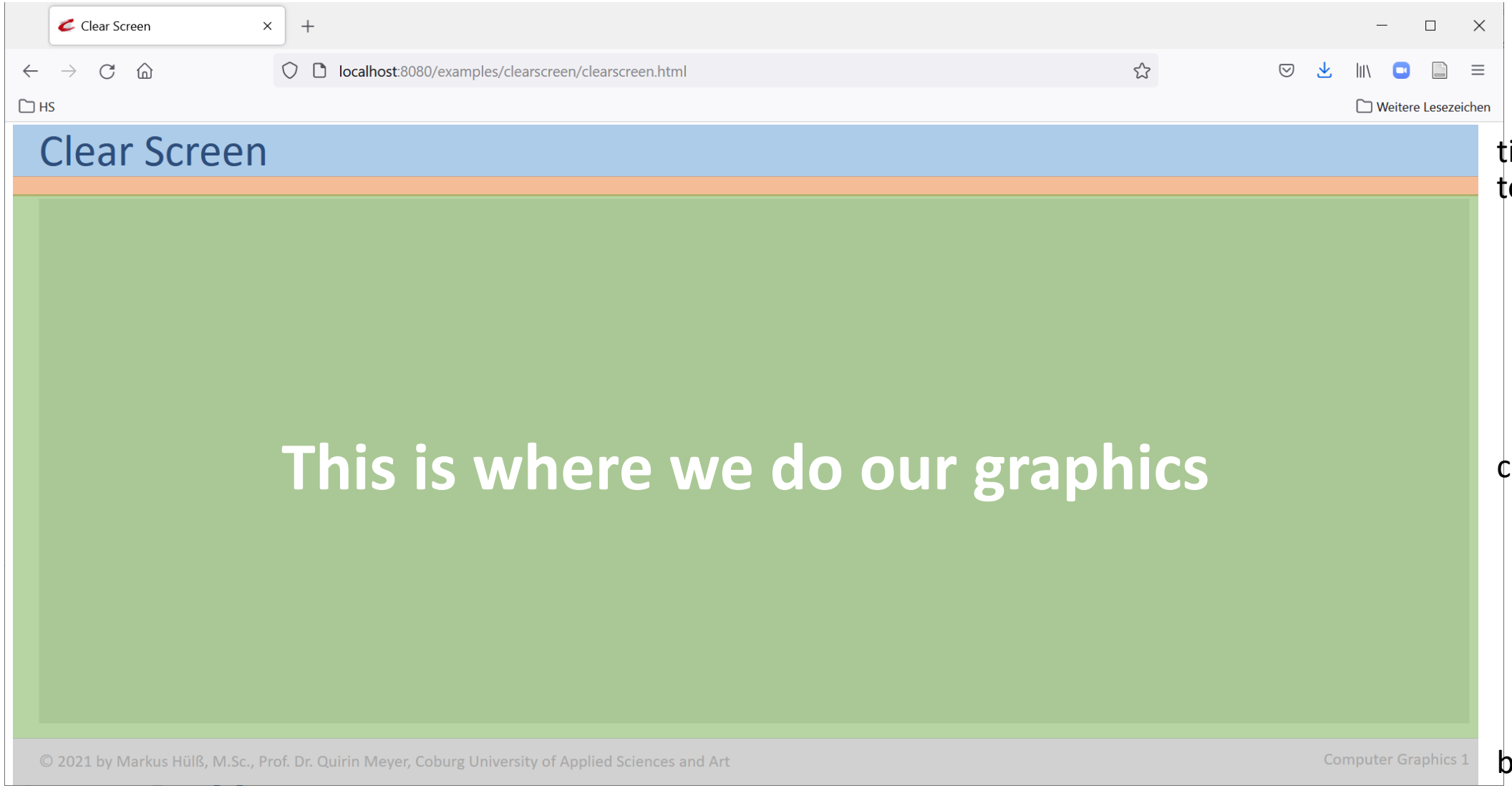
- Two Files
  - HTML File
  - JavaScript File
- Folders with Resources
  - Shaders
  - Images
  - 3D Models



WebGL – Project Structure – HTML File

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Clear Screen</title>
  <link rel="stylesheet" type="text/css" href="../../lib/style/cogra.css">
  <script src="./clearscreen.js" type="module" defer></script>
</head>

<title>Triangle</title>

<body>
    <link rel="icon" href="../../lib/style/favicon.ico">
    <div class="titlerow">Clear Screen</div>
    <div class="topbar"></div>
    <canvas id="canvas" class="output" ></canvas>
    <div class="bottombarleft">&#x24B8; 2021 by Markus Hülß, M.Sc., Prof. Dr. Quirin Meyer,
                               Coburg University of Applied Sciences and Art</div>
    <div class="bottombarright">Computer Graphics 1</div>
</body>

</html>
```

WebGL – Project Structure – HTML File



WebGL – Project Structure – HTML File

- Simple boiler-plate code: Won’t change too much during this coure
- Loads CSS
- Loads Java Script File (Does the actual work!)

WebGL – Project Structure – JavaScript File

```
function ClearScreen() {
  const mCanvas = document.querySelector("#canvas");
  const gl = mCanvas.getContext("webgl2");

  async function setup() {
    requestAnimationFrame(draw);
  }

  function draw() {
    // Draw the frame

    // First Task

    // Tell the browser to draw a frame
    requestAnimationFrame(draw);
  }
  setup();
}

var t = new ClearScreen();
```

- Create a “Class” for the application

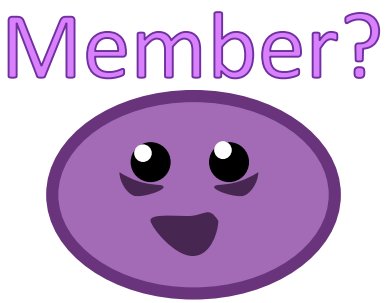
WebGL – Project Structure – JavaScript File

```
function ClearScreen() {
  const mCanvas = document.querySelector("#canvas");
  const gl = mCanvas.getContext("webgl2");

  async function setup() {
    requestAnimationFrame(draw);
  }

  function draw() {
    // Draw the frame

    // First Task

    // Tell the browser to draw a frame
    requestAnimationFrame(draw);
  }
  setup();
}

var t = new ClearScreen();
```

Two Methods

- setup
  - Load Resources
  - Start Animation
- draw
  - Draws a frame

- Processing?

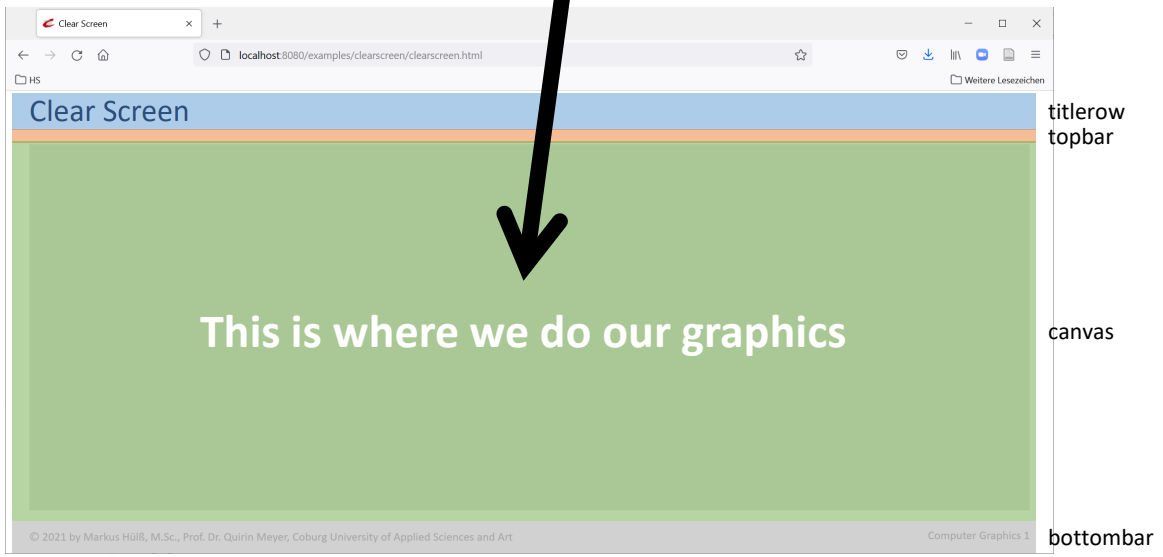


WebGL – Project Structure – JavaScript File

```
function ClearScreen() {
  const mCanvas = document.querySelector("#canvas");
  const gl = mCanvas.getContext("webgl2");

  async function setup() {
    requestAnimationFrame(draw);
  }

  function draw() {
    // Draw the frame

    // First Task

    // Tell the browser to draw a frame
    requestAnimationFrame(draw);
  }
  setup();
}

var t = new ClearScreen();
```

Get the canvas

And it's webgl2 context



WebGL – Project Structure – JavaScript File

```
function ClearScreen() {
  const mCanvas = document.querySelector("#canvas");
  const gl = mCanvas.getContext("webgl2");

  async function setup() {
    requestAnimationFrame(draw);
  }

  function draw() {
    // Draw the frame

    // First Task

    // Tell the browser to draw a frame
    requestAnimationFrame(draw);
  }
  setup();
}

var t = new ClearScreen();
```

- We‘ll use the webgl2 context‘s methods
- Prefer regular webgl (more compaitble)
- https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

WebGL – Project Structure – JavaScript File

```
function ClearScreen() {
  const mCanvas = document.querySelector("#canvas");
  const gl = mCanvas.getContext("webgl2");

  async function setup() {
    requestAnimationFrame(draw);
  }

  function draw() {
    // Draw the frame

    // First Task

    // Tell the browser to draw a frame
    requestAnimationFrame(draw);
  }
  setup();
}

var t = new ClearScreen();
```

- Draw a frame as soon as setup is done
- Draw a frame after each frame
  - Allows for animations

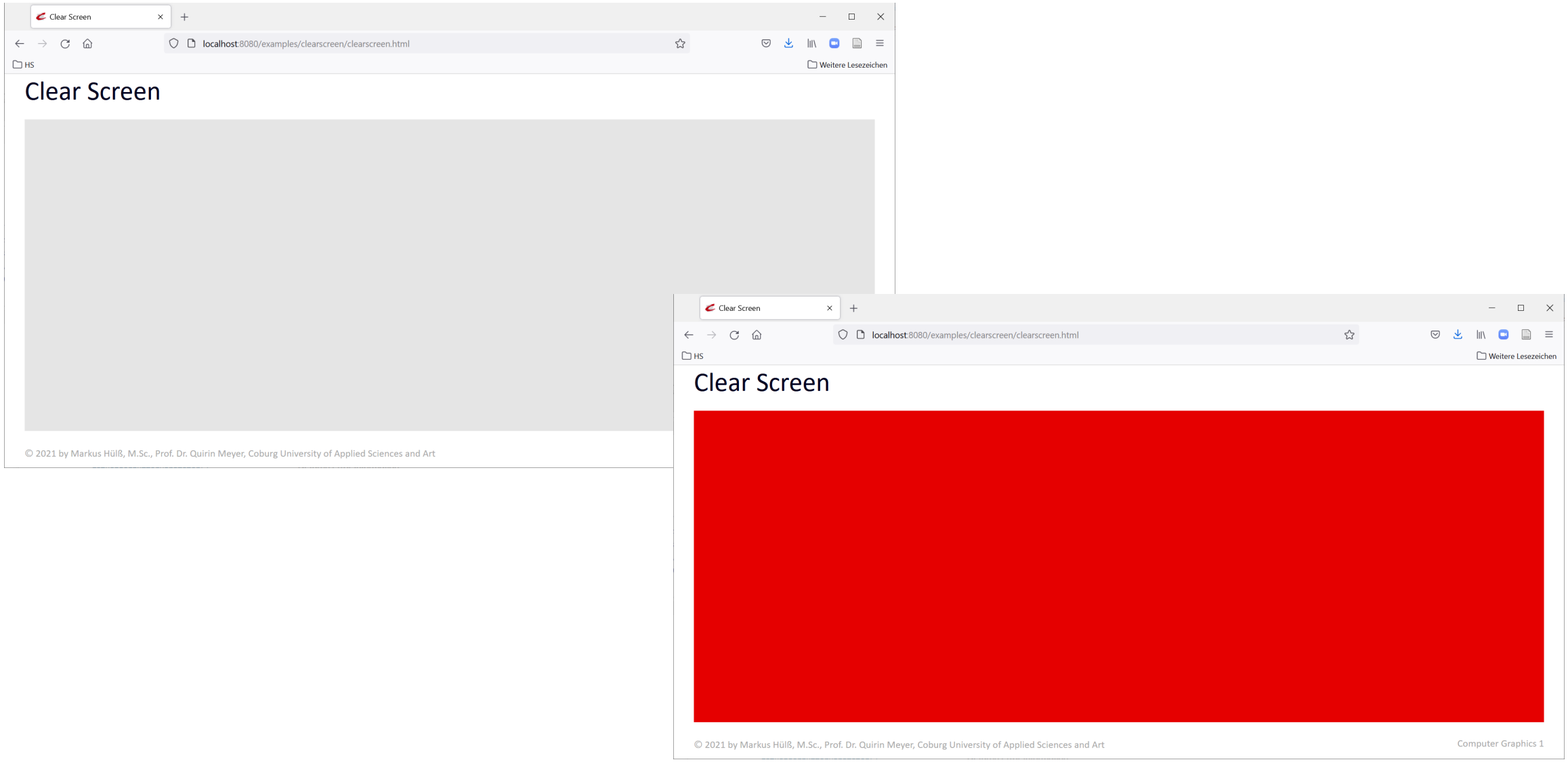
WebGL – First Task

```
function ClearScreen() {
  const mCanvas = document.querySelector("#canvas");
  const gl = mCanvas.getContext("webgl2");

  async function setup() {
    requestAnimationFrame(draw);
  }

  function draw() {
    // Draw the frame

    // First Task

    // Tell the browser to draw a frame
    requestAnimationFrame(draw);
  }
  setup();
}

var t = new ClearScreen();
```

- Task: Clear the screen with a beautiful color!
- Search on

https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext

WebGL – First Task



WebGL – First Task

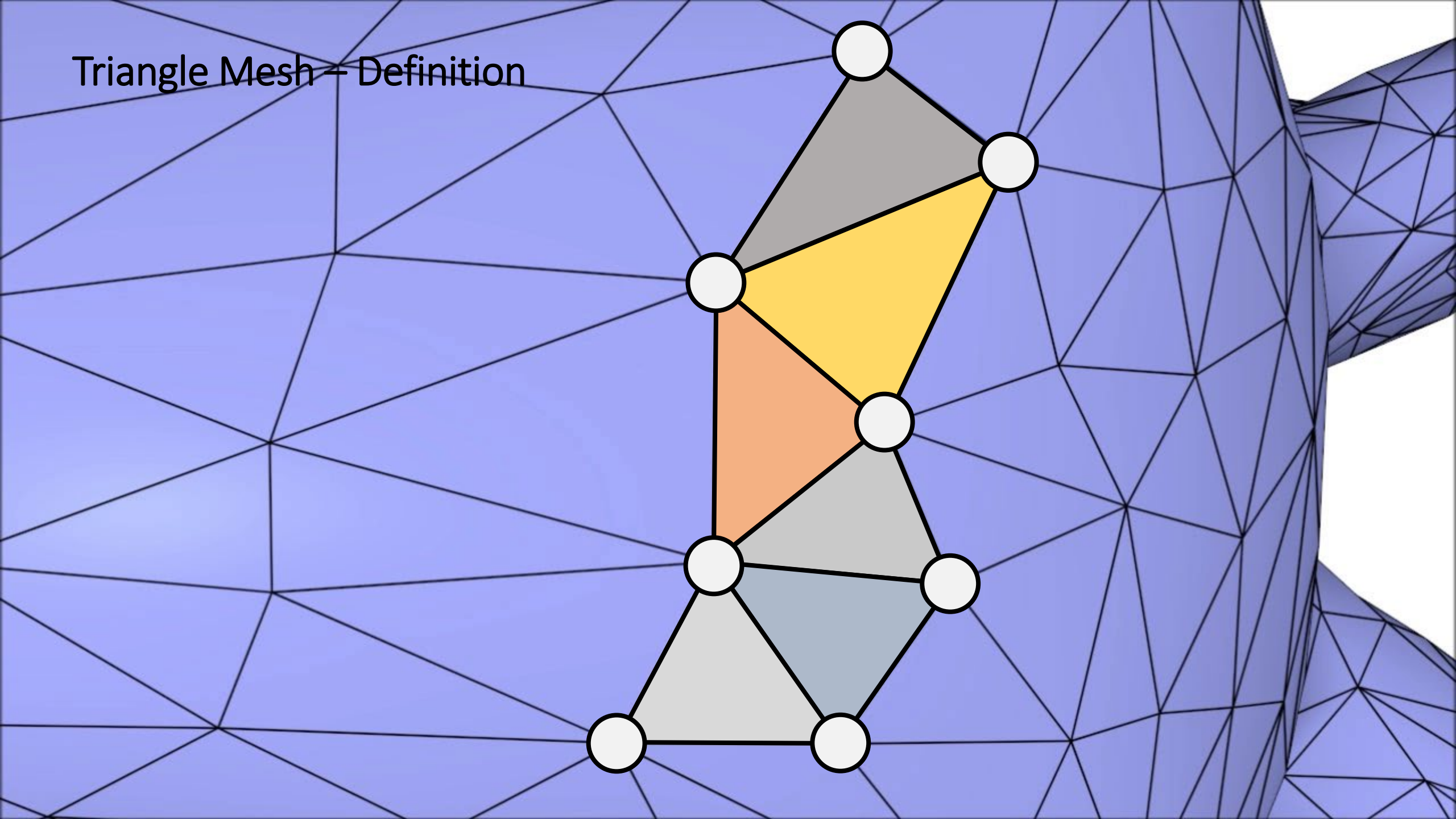
Solution – Wait for it

WebGL – First Task – Solution

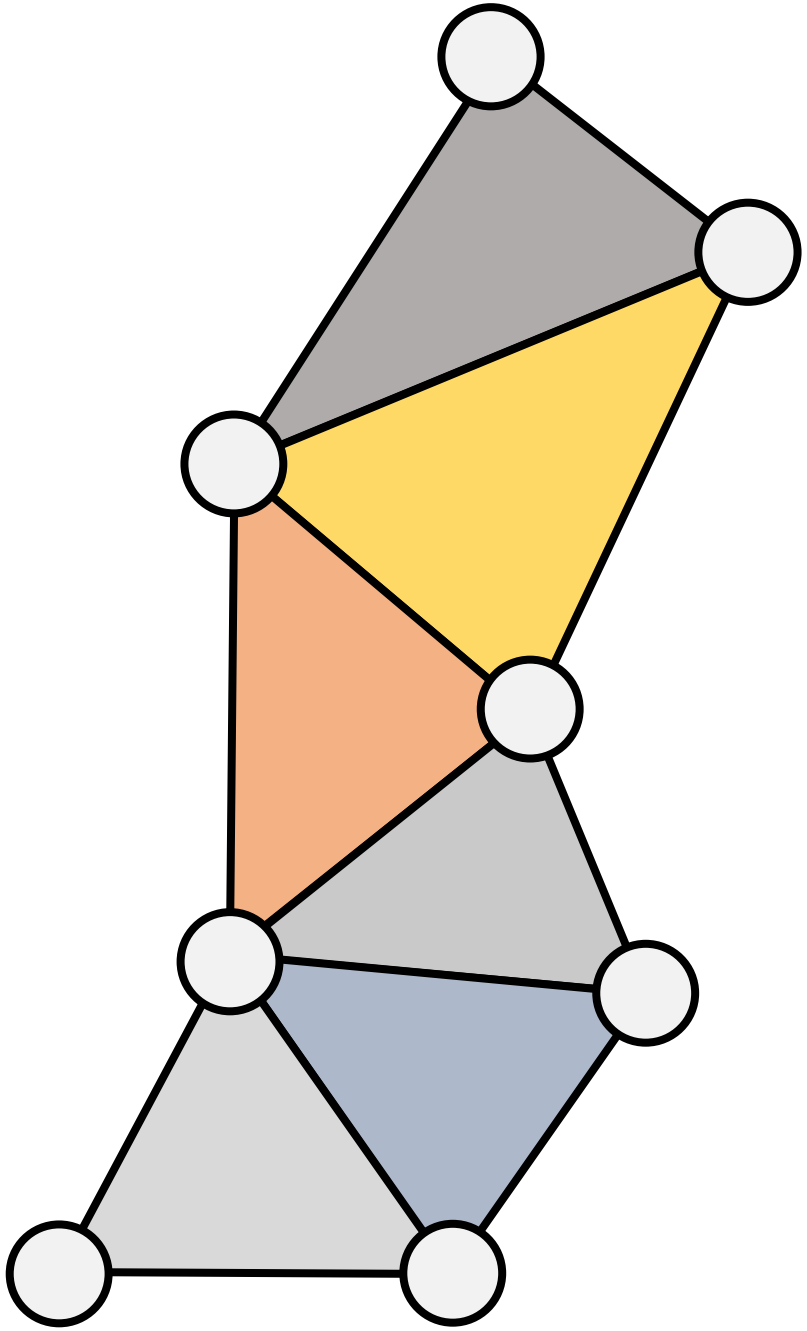
```
function ClearScreen() {
  const mCanvas = document.querySelector("#canvas");
  const gl = mCanvas.getContext("webgl2");

  async function setup() {
    requestAnimationFrame(draw);
  }

  function draw() {
    // Draw the frame
    gl.clearColor(0.9, 0.9, 0.9, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    // draw next
    requestAnimationFrame(draw);
  }
  setup();
}


var t = new ClearScreen();
```

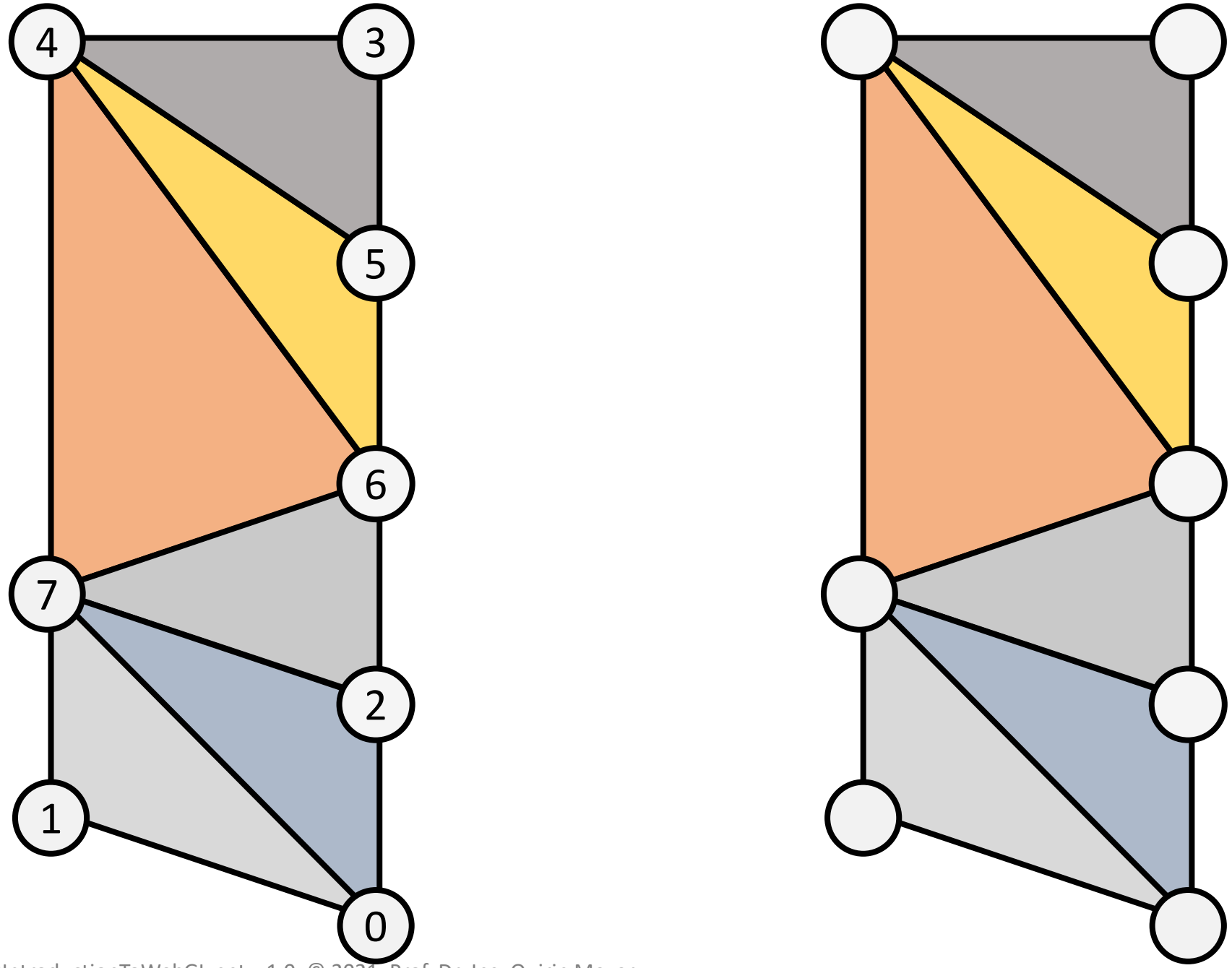
Triangle Mesh – Definition

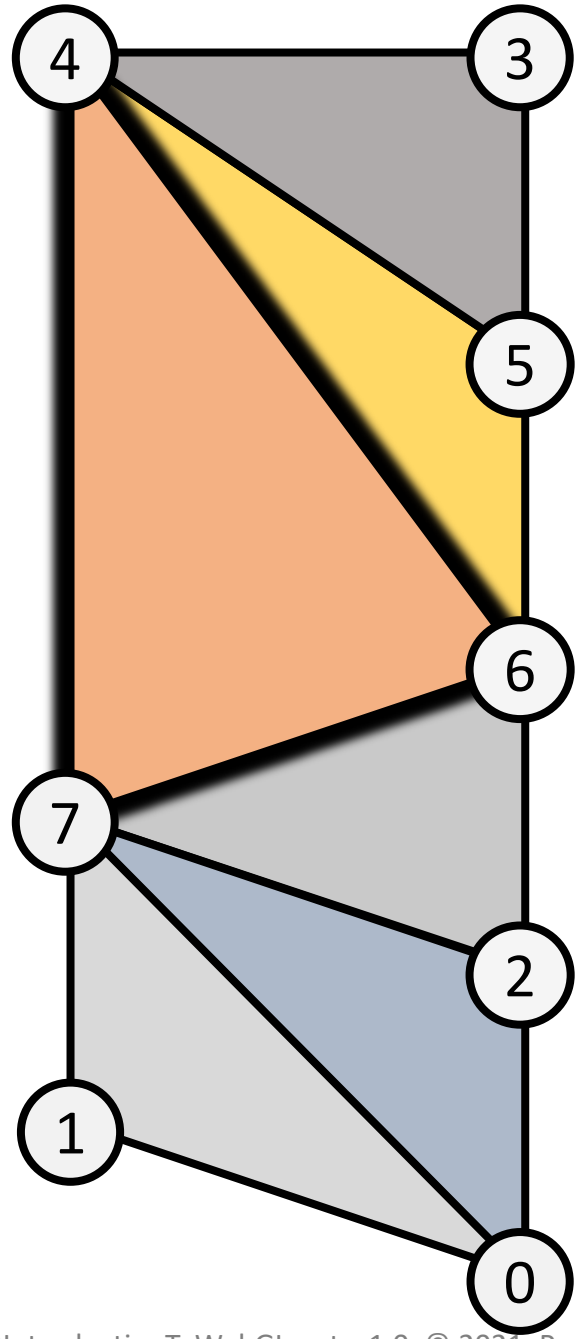
Triangle Mesh – Definition



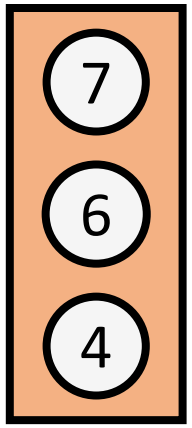
Triangle Mesh – Definition

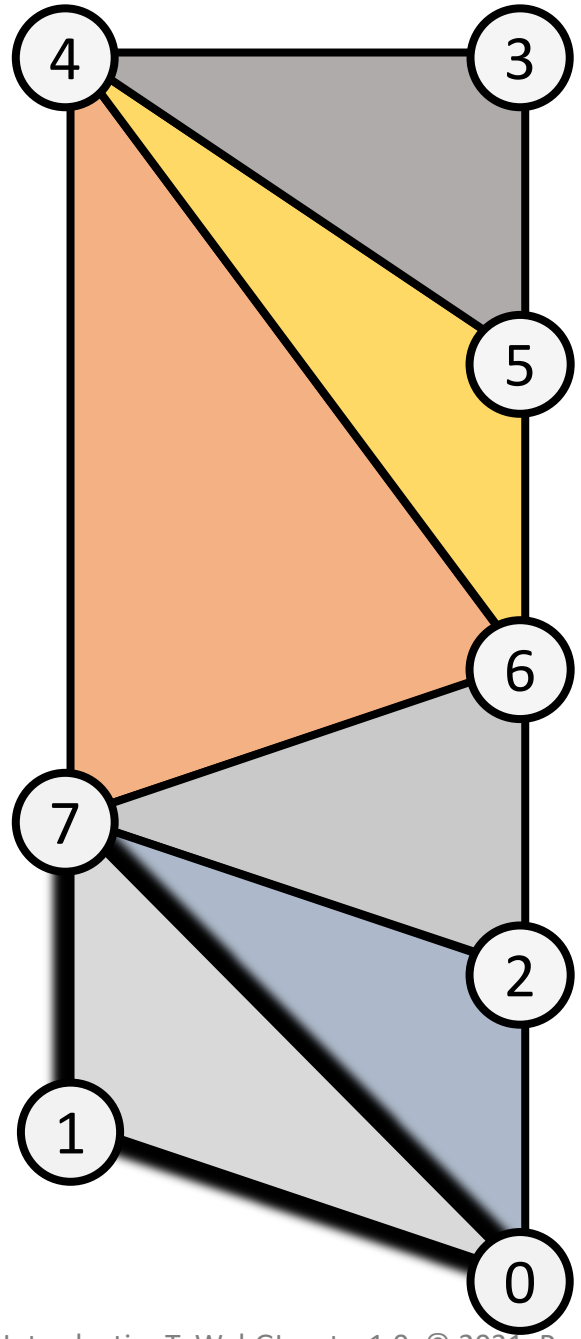



4
3
5
6
7
2
1
0



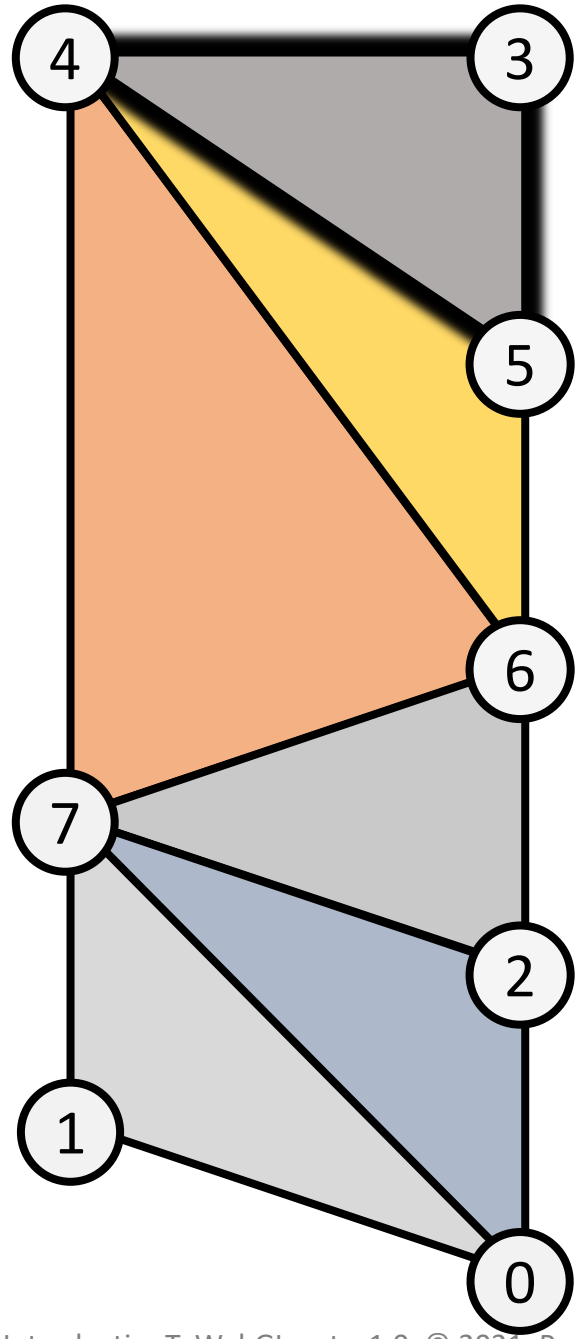
I=





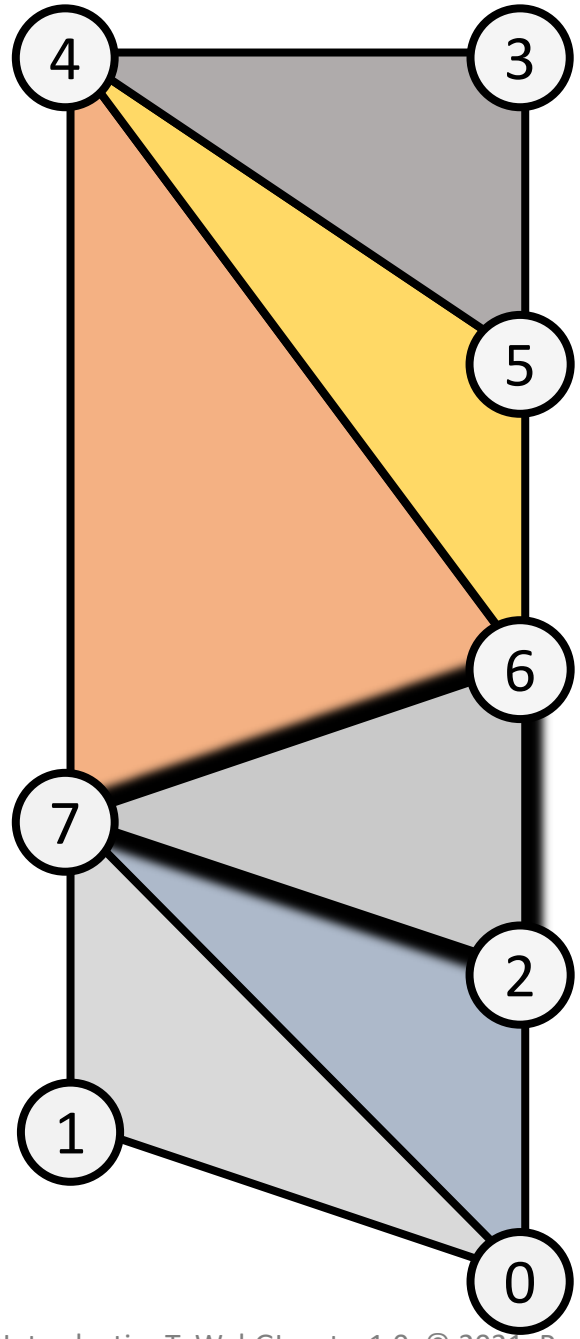
I=

7	1
6	0
4	7



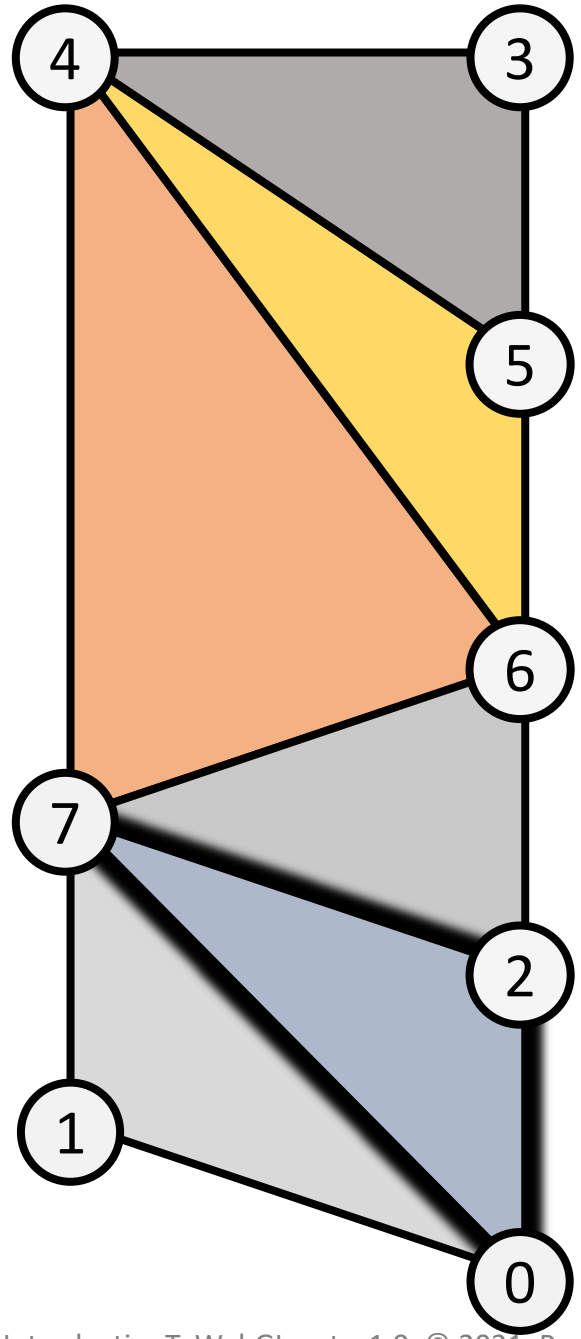
I=

7	1	4
6	0	5
4	7	3



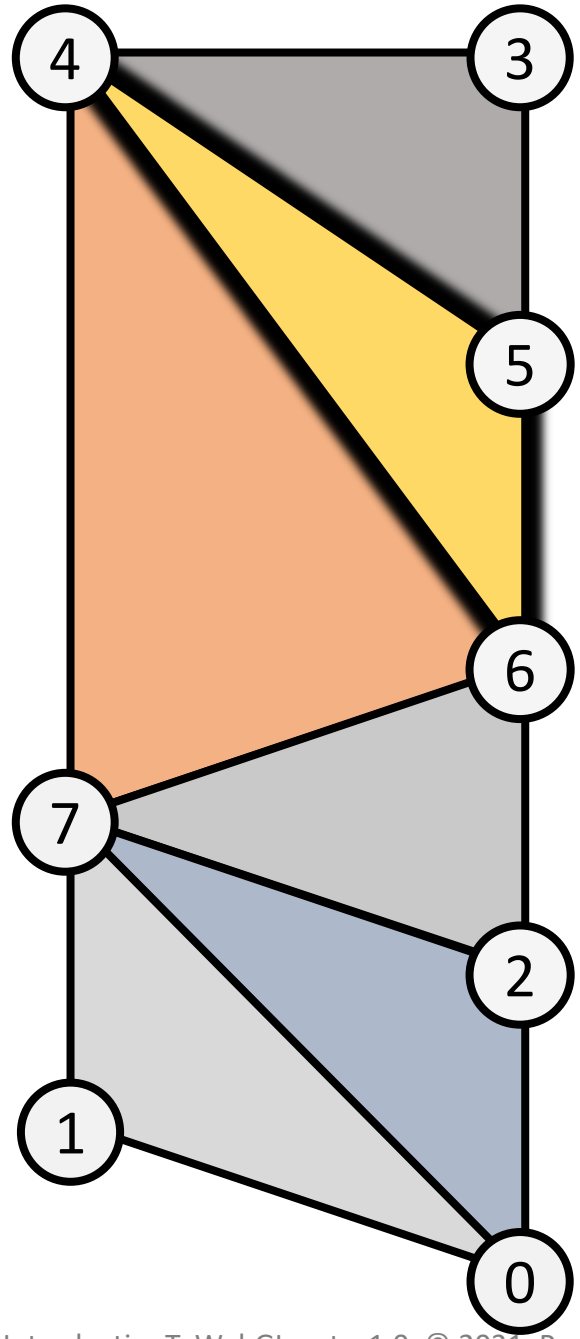
I=

7	1	4	7
6	0	5	2
4	7	3	6



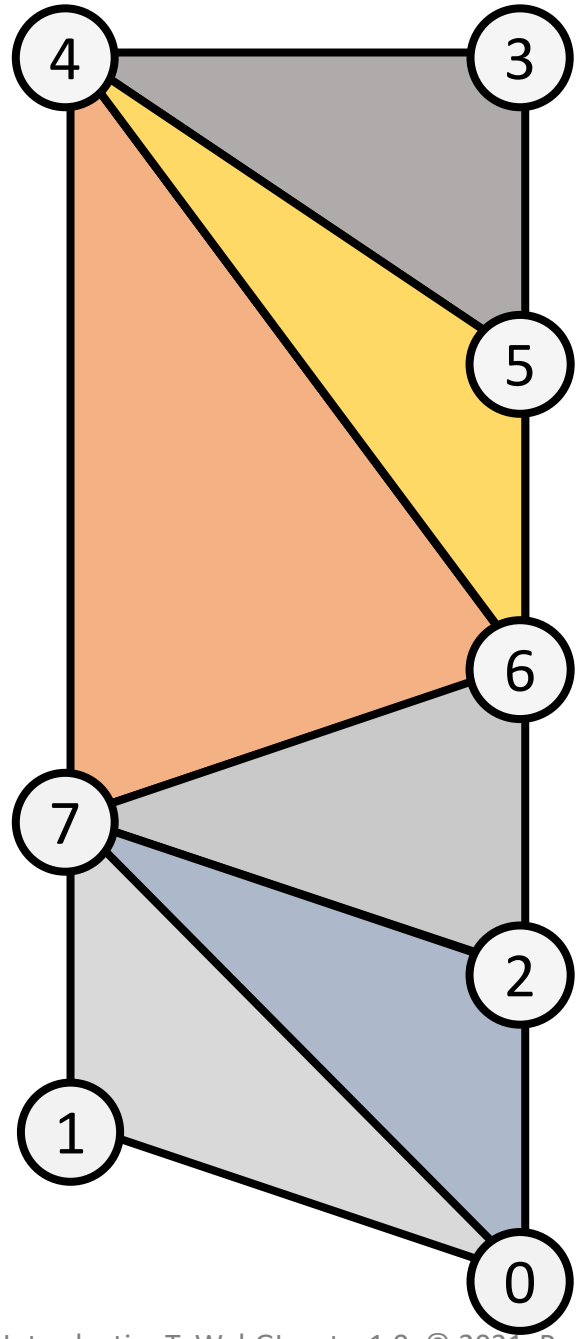
I=

7	1	4	7	7
6	0	5	2	0
4	7	3	6	2



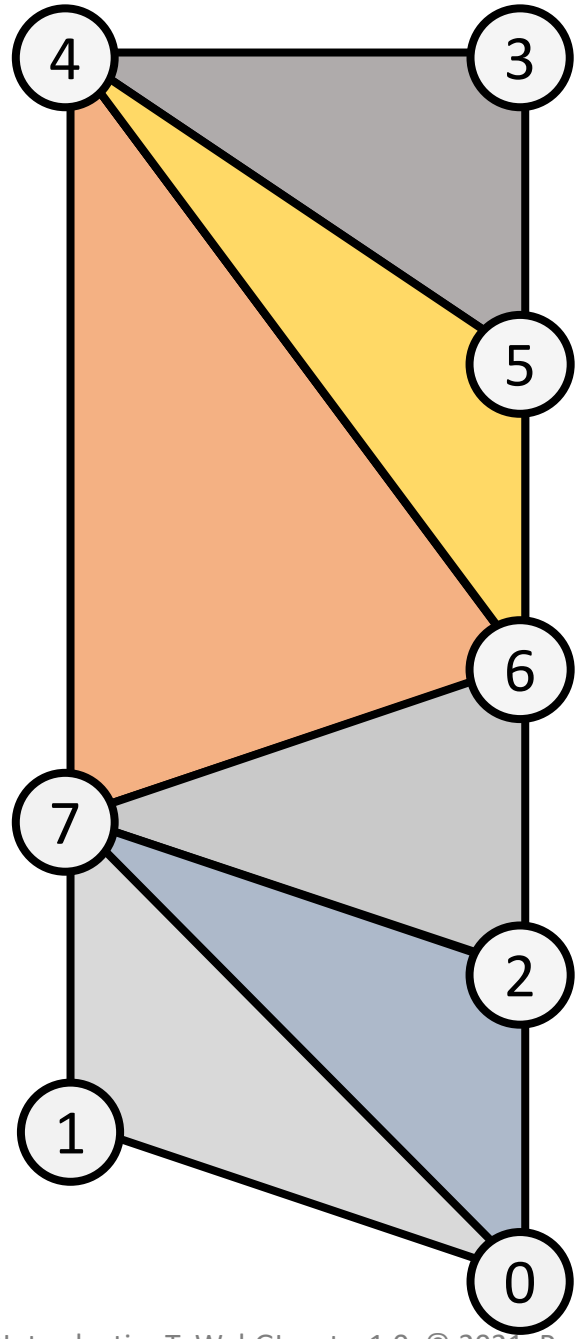
I=

7	1	4	7	7	4
6	0	5	2	0	6
4	7	3	6	2	5



I=

7	1	4	7	7	4
6	0	5	2	0	6
4	7	3	6	2	5



I=

7	1	4	7	7	4
6	0	5	2	0	6
4	7	3	6	2	5

Triangle Mesh – Definition

- **Triangle Mesh** is a *graph*
  - **I:** Set of Triangles
    - Array of indices
    - Indices point to vertices
    - Common Names:
      Index Buffer, Element Array Buffer
  - **V:** Set of Vertices with Attributes
    - Positions
    - Texture Coordinates
    - Normal Vectors
    - …

I=

7	1	4	7	7	4
6	0	5	2	0	6
4	7	3	6	2	5

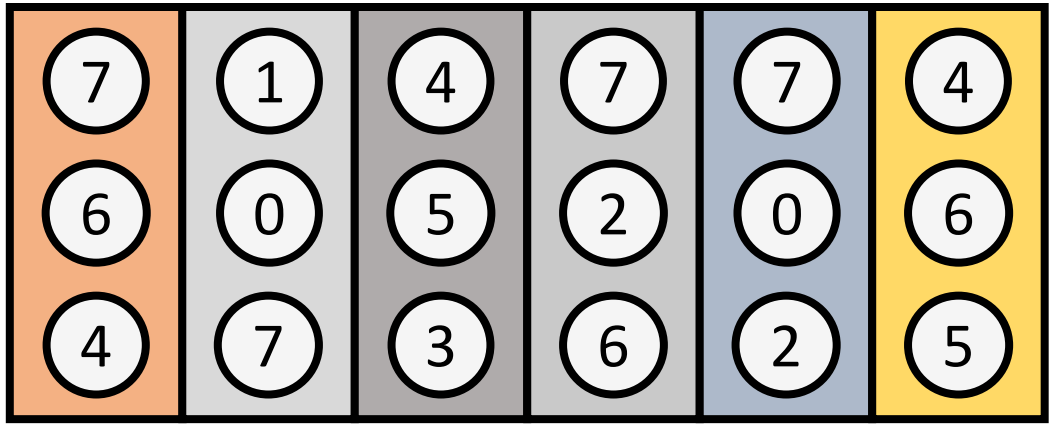
V=

Vertex Index	Position	Colors
0	[1.0, 0.0]	[1.0, 1.0, 0.0]
1	[0,0.5]	[1.0, 0.0, 0.0]
2	[1.0,3.0]	[1.0, 1.0, 1.0]
3	[0.0,10.0]	[0.0, 1.0, 0.0]

Triangle Mesh – Specification

- Initialize the buffers are in CPU memory
- Upload CPU buffers to GPU memory

I=

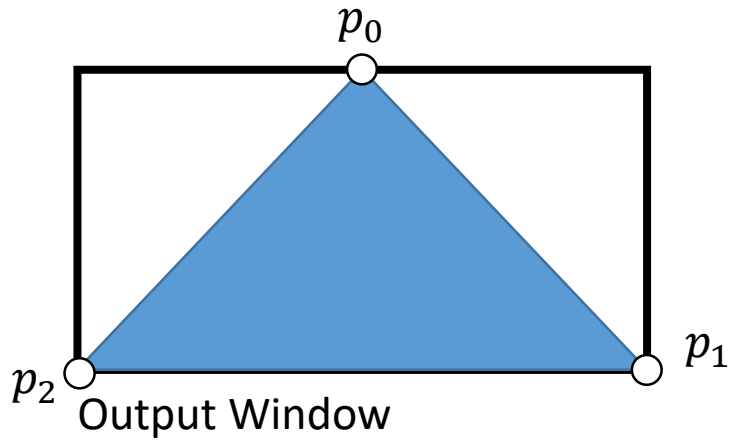


V=

Vertex Index	Position	Texture Coordinates
0	[1.0, 0.0]	[1.0, 1.0, 0.0]
1	[0,0.5]	[1.0, 0.0, 0.0]
2	[1.0,3.0]	[1.0, 1.0, 1.0]
3	[0.0,10.0]	[0.0, 1.0, 0.0]

Triangle Mesh – CPU – Specification

- **Example:** Specify a simple Triangle mesh that consists of a single triangle



Triangle Mesh – CPU – Specify a Simple Mesh

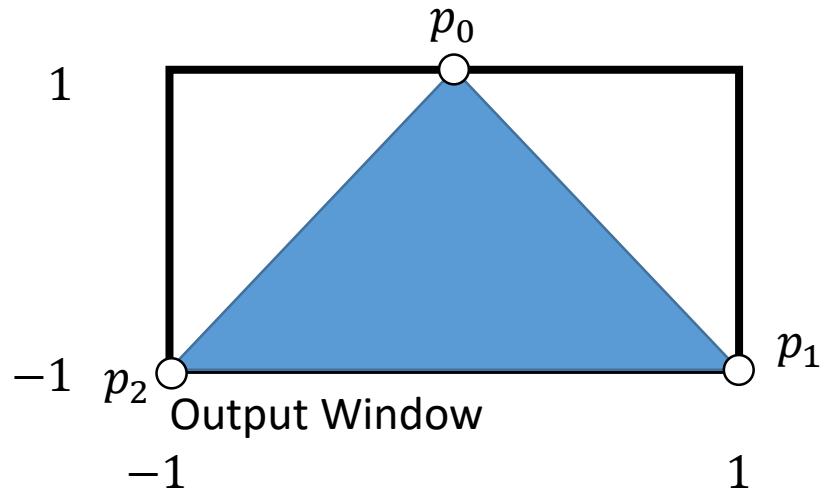
- Specify Positions array on the CPU

```
const positions = [
 0.0,  1.0, 0.0, 1.0,
 1.0, -1.0, 0.0, 1.0,
-1.0, -1.0, 0.0, 1.0];
```

$z \in [-1..1]$
Ignored for now

x, y coordinate

$w = 1$
for now



- Array of floats:

  Four floats make a 4D point

- OpenGL Window Coordinates :

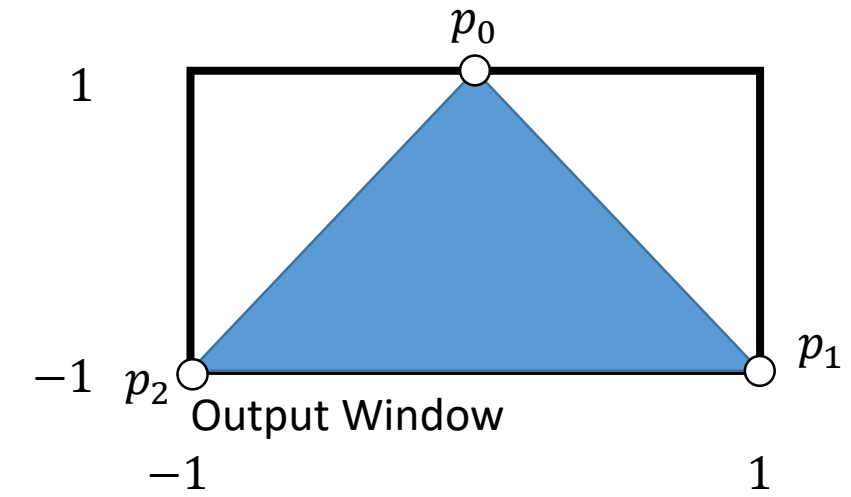
  [−1..1]×[−1..1]

- Positions can be 2D, 3D, 4D

Triangle Mesh – CPU – Specify a Simple Mesh

- Specify Indices array on the CPU
- `const indices = [0, 1, 2];`
- Array of integers
- Three integers

  make a triangle




```
positions[2]: (-1.0f, -1.0f, 0.0f, 1.0f); // p2
positions[1]: ( 1.0f, -1.0f, 0.0f, 1.0f); // p1
positions[0]: ( 0.0f,  1.0f, 0.0f, 1.0f); // p0
```

Triangle Mesh – GPU – Geometry Upload

- **Vertex-Arrays:** Object that gathers
  - Element Array Buffer (Index Buffer)
  - Vertex Attribute Buffers
  - Information that is required to render the mesh

Triangle Mesh – GPU – Geometry Upload

Create **Vertex-Arrays**

- `vao = gl.createVertexArray();`

Triangle Mesh – GPU – Geometry Upload

Bind **Vertex-Arrays**

- `gl.bindVertexArray(vao);`

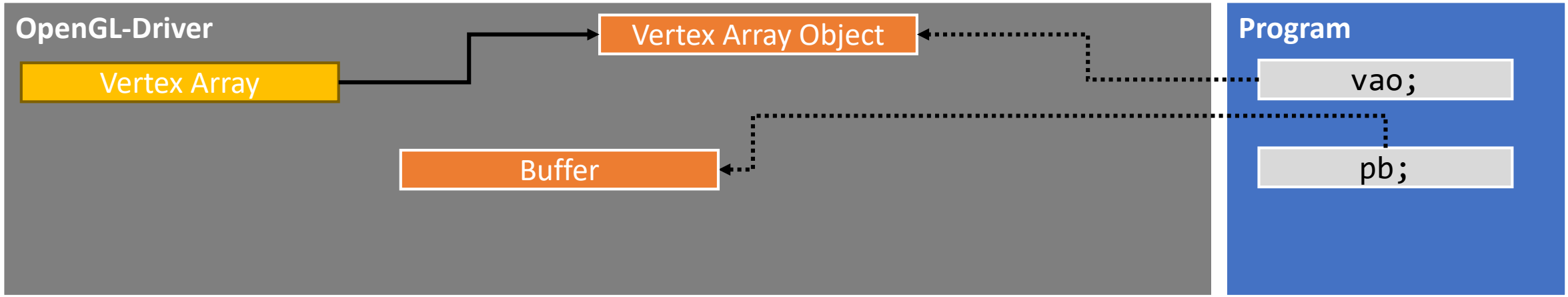
Triangle Mesh – GPU – Geometry Upload

- **Vertex Buffer:** GPU Array with Vertex Information
  - Attributes (positions, normal vectors, texture coordinates)
  - Element Array Buffer

Triangle Mesh – GPU – Geometry Upload

Create **Vertex Buffer**

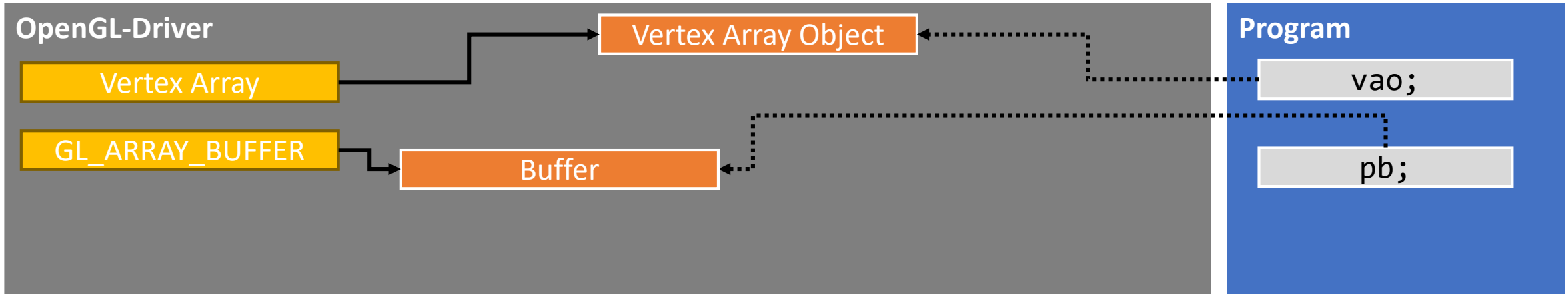
- `const pb = gl.createBuffer();`



Triangle Mesh – GPU – Geometry Upload

Bind **Vertex Buffer**

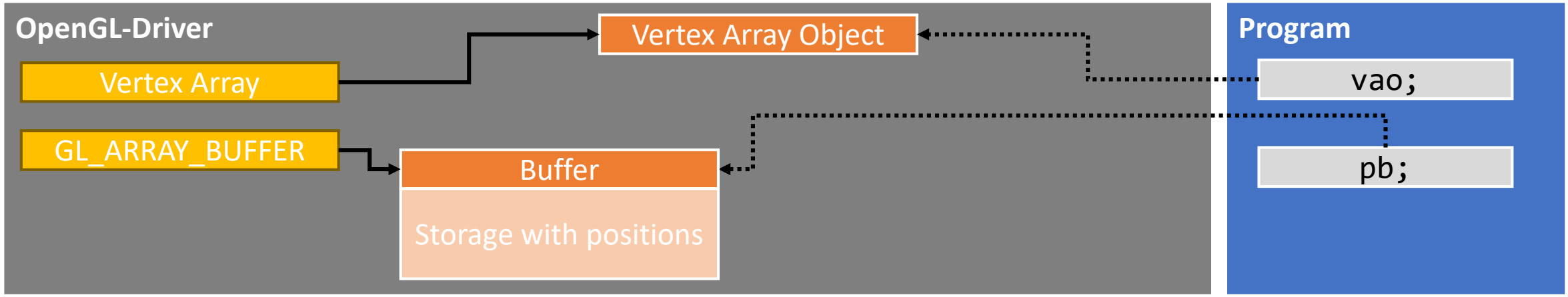
- `gl.bindBuffer(gl.ARRAY_BUFFER, pb);`



Triangle Mesh – GPU – Geometry Upload

Add **Storage** to Vertex Buffer

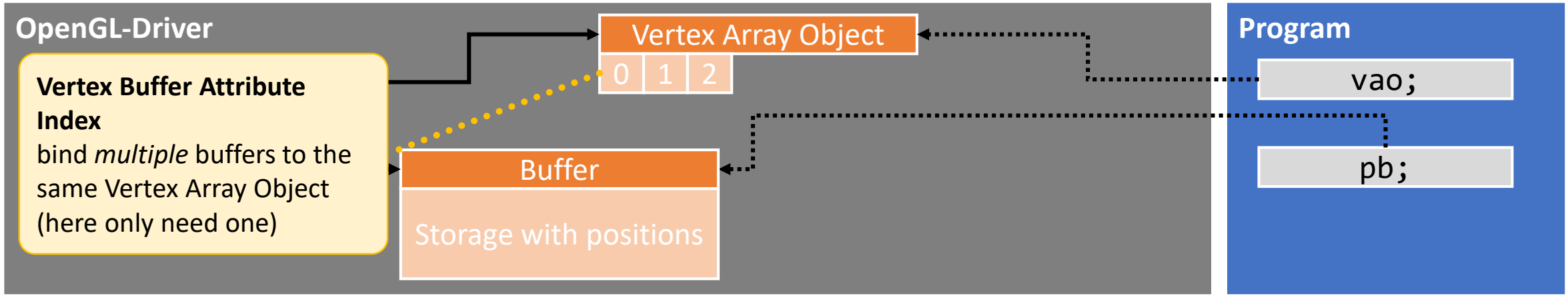
- `gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions),`
  `gl.STATIC_DRAW);`



Triangle Mesh – GPU – Geometry Upload

Connect **Vertex Buffer** with **Vertex Array Object**

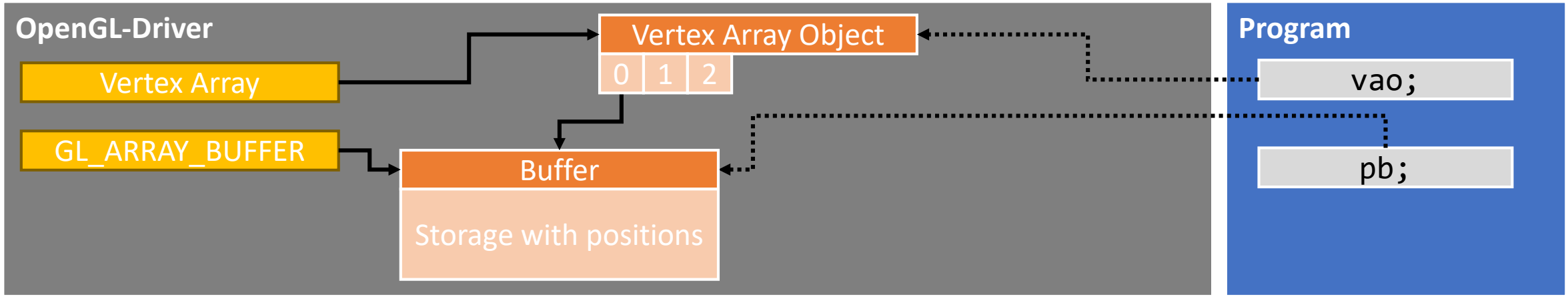
```
const positionAttributeLocation = 0;


gl.vertexAttribPointer(positionAttributeLocation, 4, gl.FLOAT,
                       false, 0, 0);
```



Triangle Mesh – GPU – Geometry Upload

Connect **Vertex Buffer** with **Vertex Array Object**

```
const positionAttributeLocation = 0;


gl.vertexAttribPointer(positionAttributeLocation, 4, gl.FLOAT,
                       false, 0, 0);
```

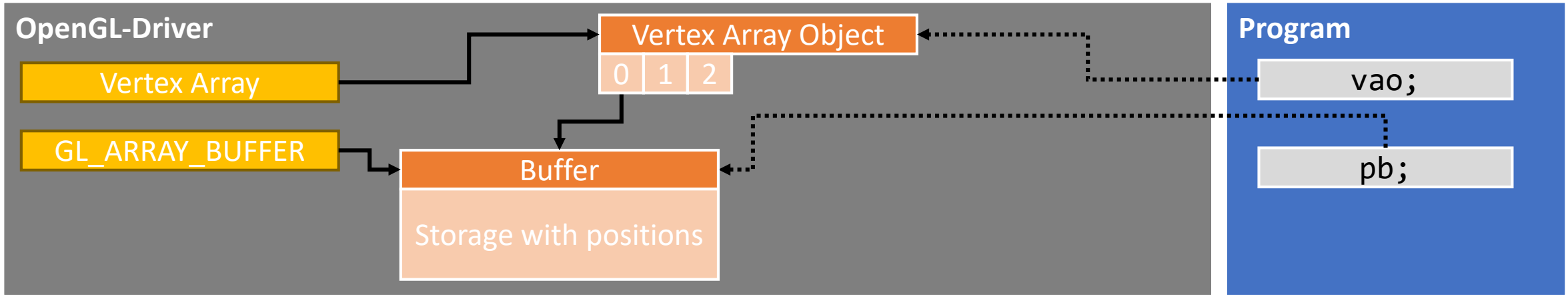


Triangle Mesh – GPU – Geometry Upload

Enable Vertex Attribute for Buffer

We need to adjust this later

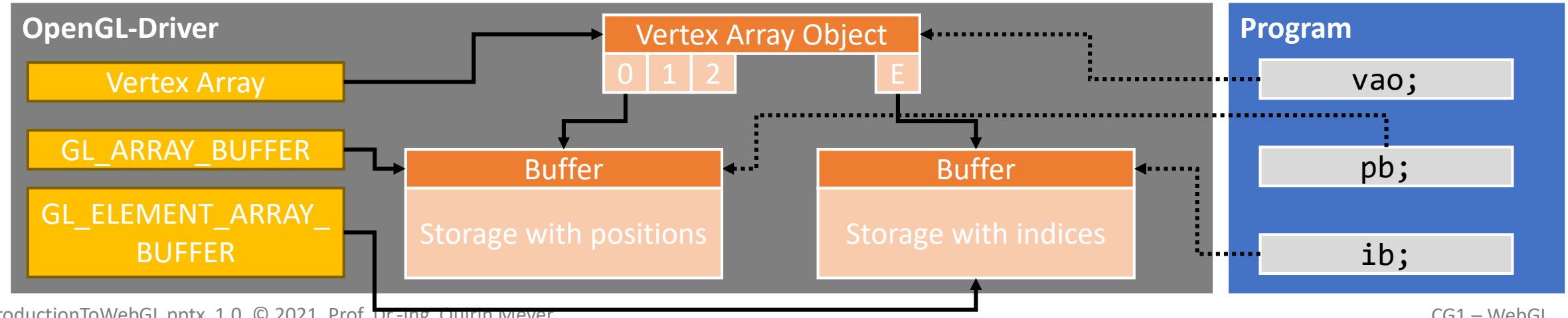
```
const positionAttributeLocation = 0;
gl.enableVertexAttribArray(positionAttributeLocation);
gl.vertexAttribPointer(positionAttributeLocation, 4, gl.FLOAT,
                       false, 0, 0);
```



Triangle Mesh – GPU – Geometry Upload

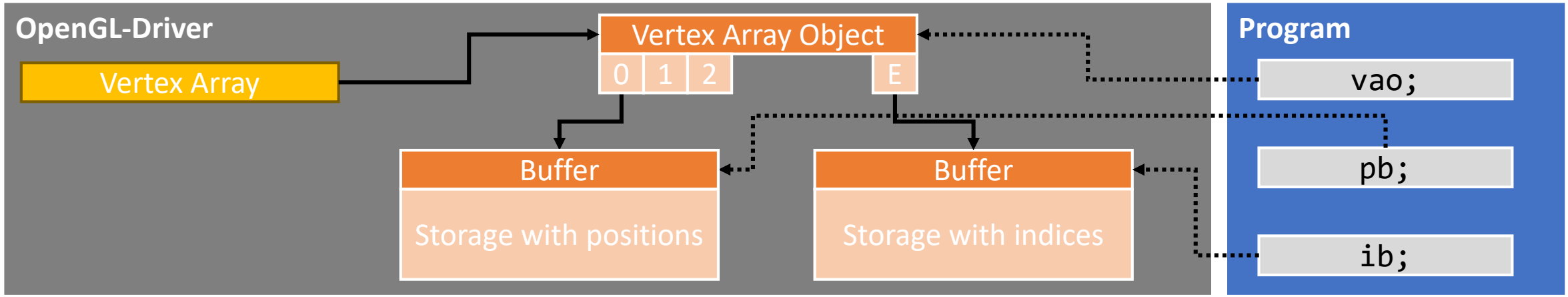
Treat `indices` similar to positions

```
const ib = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, ib);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint32Array(indices),
gl.STATIC_DRAW);
```



Triangle Mesh – GPU – Drawing

```
function draw() {
  gl.clearColor(0.9, 0.9, 0.9, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);
  gl.bindVertexArray(vao);
  gl.drawElements(gl.TRIANGLES, 3, gl.UNSIGNED_INT, 0);
  requestAnimationFrame(draw);
}
```



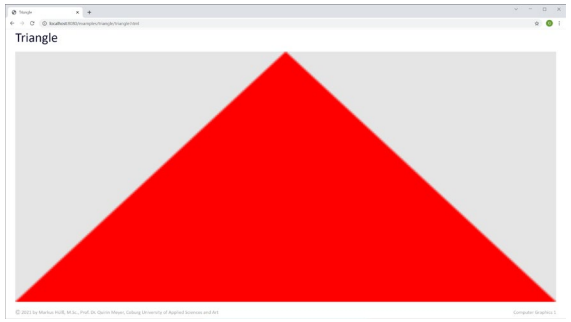
Shading – Simplified GPU Pipeline

Position buffer on the GPU

pb	[0, 1, 0, 1] [1, -1, 0, 1] [-1, -1, 0, 1] [1, 1, 0, 1]

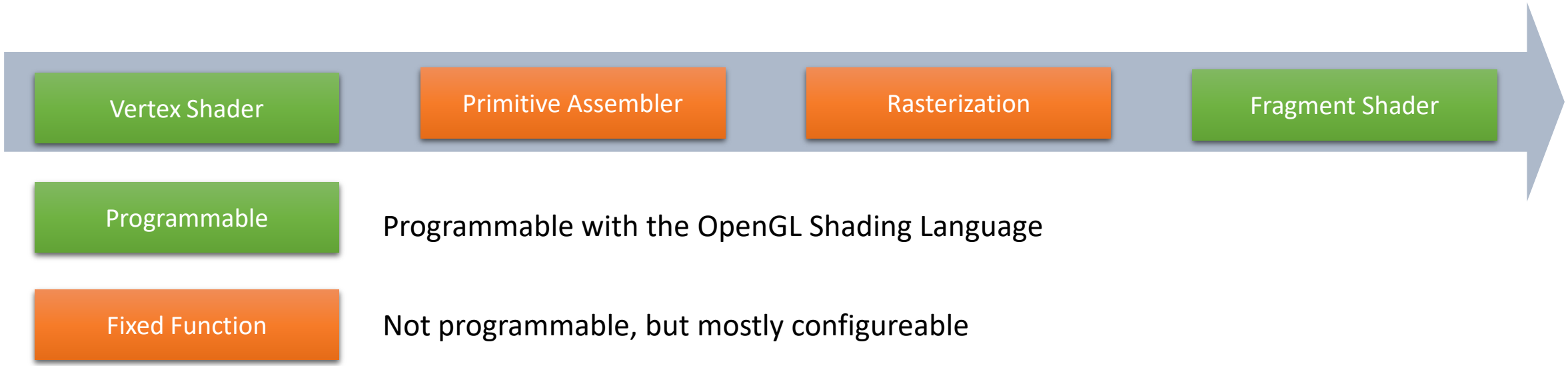
Element Array Buffer on the GPU

ib	[0, 1, 2] [0, 1, 3]



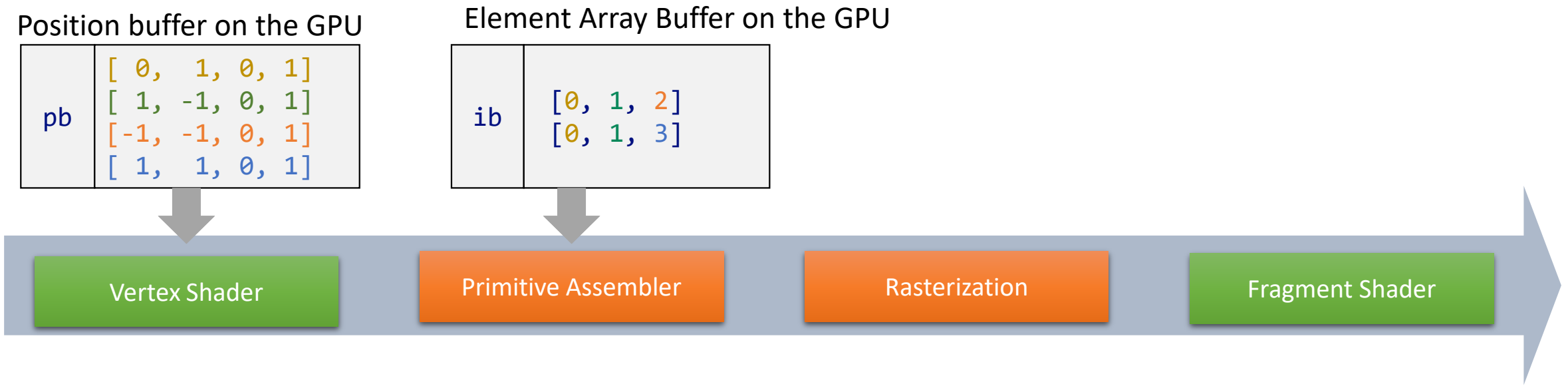
- Input: Geometry
  - as specified by vertex array object
- Output: Rendered Image
  - Dimensions specified by glViewport

Shading – Simplified GPU Pipeline



- Pipeline has several stages
- Here: Simplified to four stages

Shading – Simplified GPU Pipeline



Shading

Position buffer on the GPU

pb	[0, 1, 0, 1] [1, -1, 0, 1] [-1, -1, 0, 1] [1, 1, 0, 1]

Element Array Buffer on the GPU

ib	[0, 1, 2] [0, 1, 3]

Vertex Shader

gl_Position

```
[ 0,  1, 0, 1]
[ 1, -1, 0, 1]
[-1, -1, 0, 1]
[-1, -1, 0, 1]
```

- Transform vertices individually
- Output gl_Position

Primitive Assembler

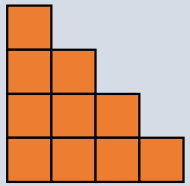
Triangles

```
[ 0,  1, 0, 1]
[ 1, -1, 0, 1]
[-1, -1, 0, 1]

[ 0,  1, 0, 1]
[ 1, -1, 0, 1]
[-1, -1, 0, 1]
```

- Create Triangles out of Element Array Buffer and gl_Positions

Rasterization

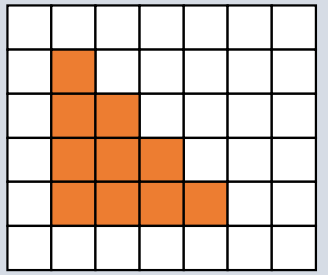
Fragments



- Rasterize: Find all fragments that cover the triangle

Fragment Shader

Shaded Fragments



- Determines Color of the Fragments

Shading

Vertex Shader

```
layout (location=0)
in vec4 inVertex;

void main()
{
    gl_Position = inVertex;
}
```

Fragment Shader

```
out vec4 fragColor;


void main()
{
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

- Let's Program
  - Vertex Shader Stages
  - Fragment Shader Stage
- C-Like Language called **GLSL (OpenGL Shading Language)**

Shading – Vertex Shader

Vertex Shader

```
layout (location=0)
in vec4 inVertex;

void main()
{
    gl_Position = inVertex;
}
```

Attribute location

Specify the input data from Vertex Array Buffers

Here it is just the positions of triangle vertices

Main function of the vertex shader

Here we simply copy the input vertex to the window position

Mandatory output: Tells the pipeline the position Of the vertex on the window

Shading – Vertex Shader

Vertex Shader

```
layout (location=0)
in vec4 inVertex;

void main()
{
    gl_Position = inVertex;
}
```

- Vertex Shader is invoked once for every vertex in the vertex buffers

pb	[-1, 0, 0, 1] [1, -1, 0, 1] [-1, -1, 0, 1] [1, 1, 0, 1]

- Independently
- In Parallel

Shading – Fragment Shader

Out defines the name of the pixel and type that we wish to write

Main function of the fragment shader

Set the color to read of the pixel to read

Fragment Shader

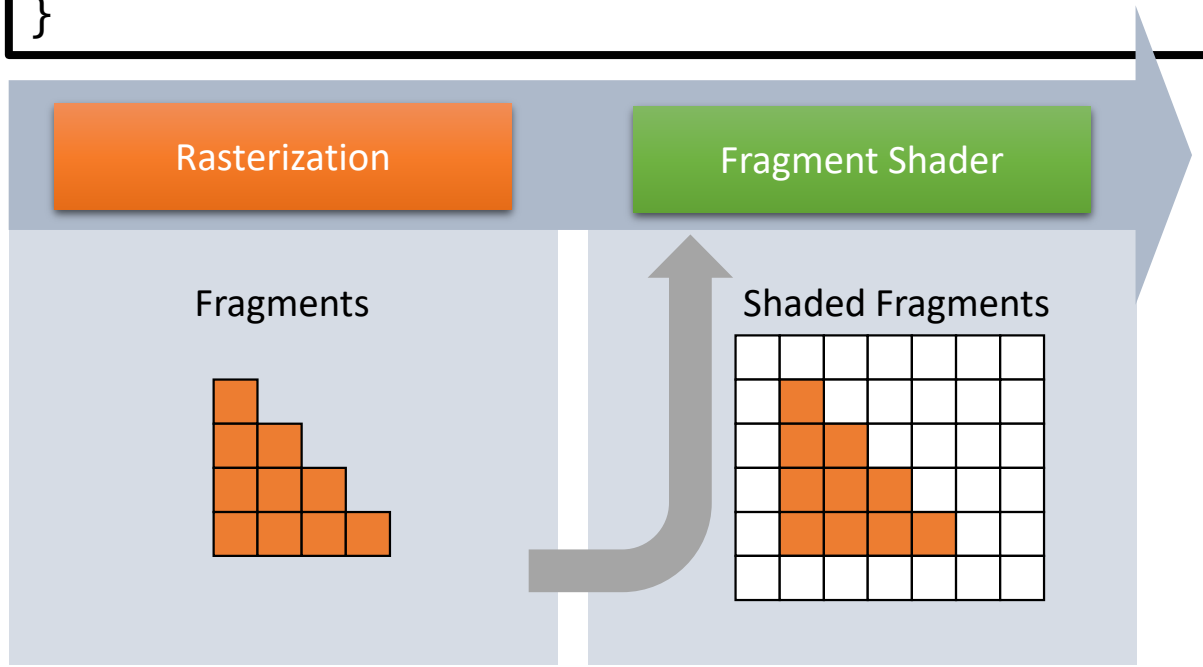
```
out vec4 fragColor;


void main()
{
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Shading – Fragment Shader

- The fragment shader is invoked for every fragment produced by the rasterizer
- Independently
- In Parallel

Fragment Shader

```
out vec4 fragColor;


void main()
{
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

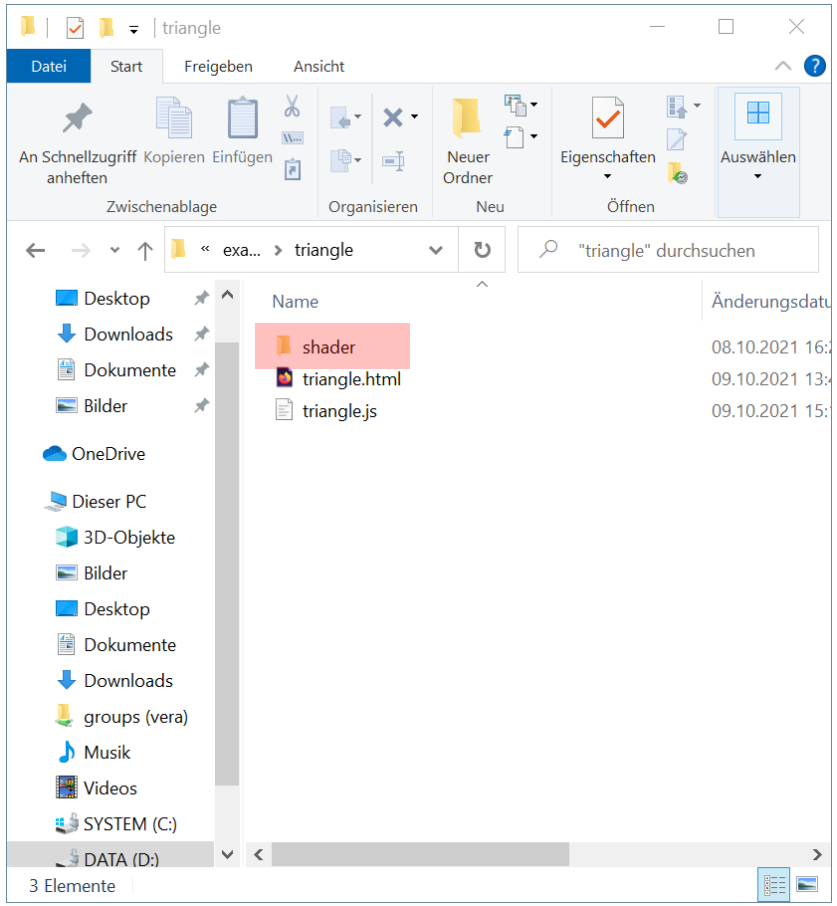


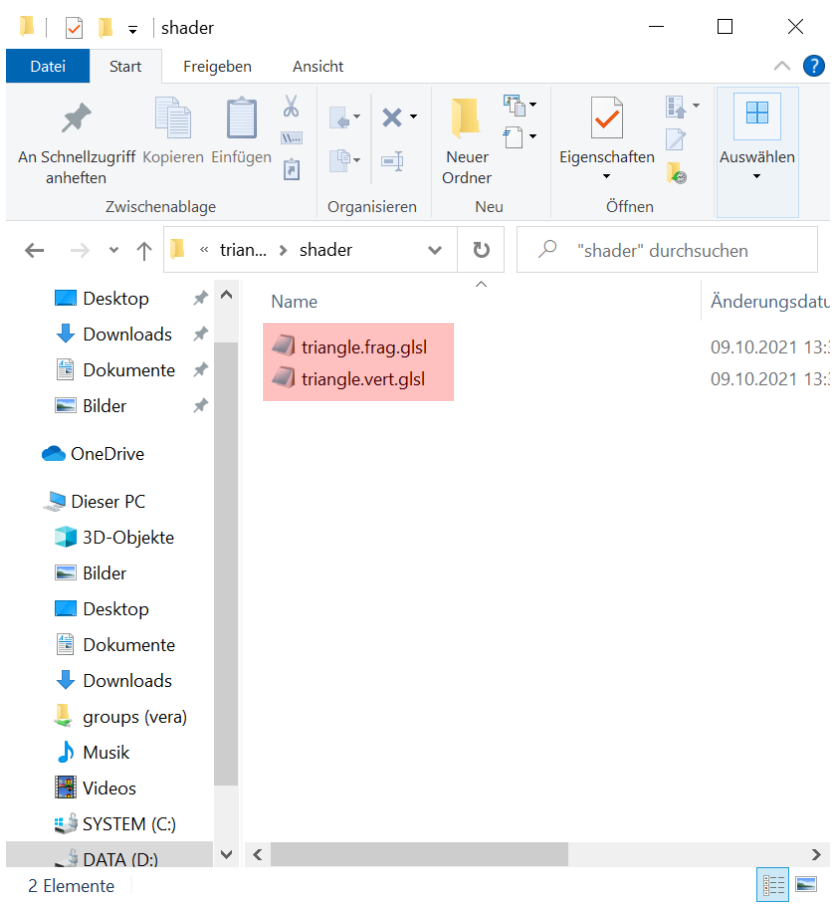
Shading – Specify Shaders

- Text Files
- Path specified in html file

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Triangle</title>
  <script id="vertexShader" src="./shader/triangle.vert.glsl" type="x-shader/x-vertex" defer></script>
  <script id="fragmentShader" src="./shader/triangle.frag.glsl" type="x-shader/x-fragment" defer></script>
  <link rel="stylesheet" type="text/css" href="../../lib/style/cogra.css">
  <script src="./triangle.js" type="module" defer></script>
</head>

<title>Triangle</title>

<body>
    <link rel="icon" href="../../lib/style/favicon.ico">
    <div class="titlerow">Triangle</div>
    <div class="topbar"></div>
    <canvas id="canvas" class="output" ></canvas>
    <div class="bottombarleft">&#x24B8; 2021 by Markus Hülß, M.Sc., Prof. Dr. Quirin Meyer, Coburg University of Appli
Sciences and Art</div>
    <div class="bottombarright">Computer Graphics 1</div>
</body>
```

Shading – Specify Shaders



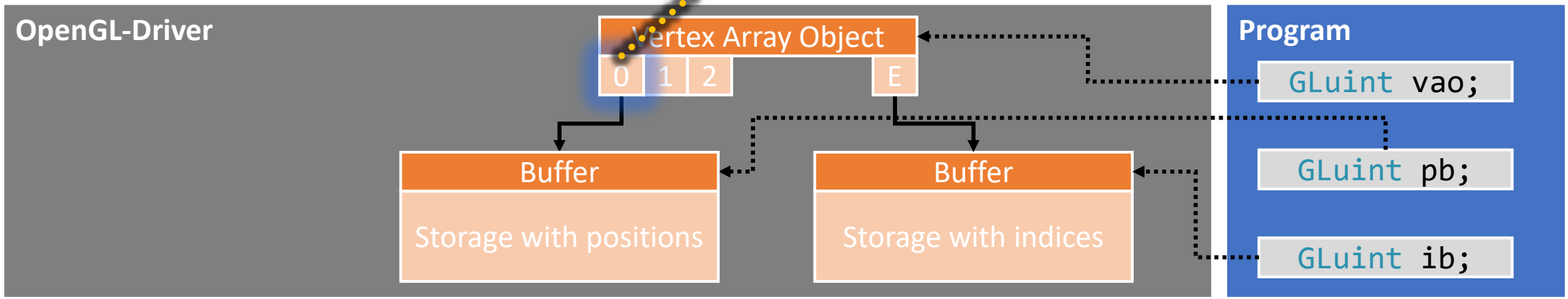


Shading – Specify Shaders

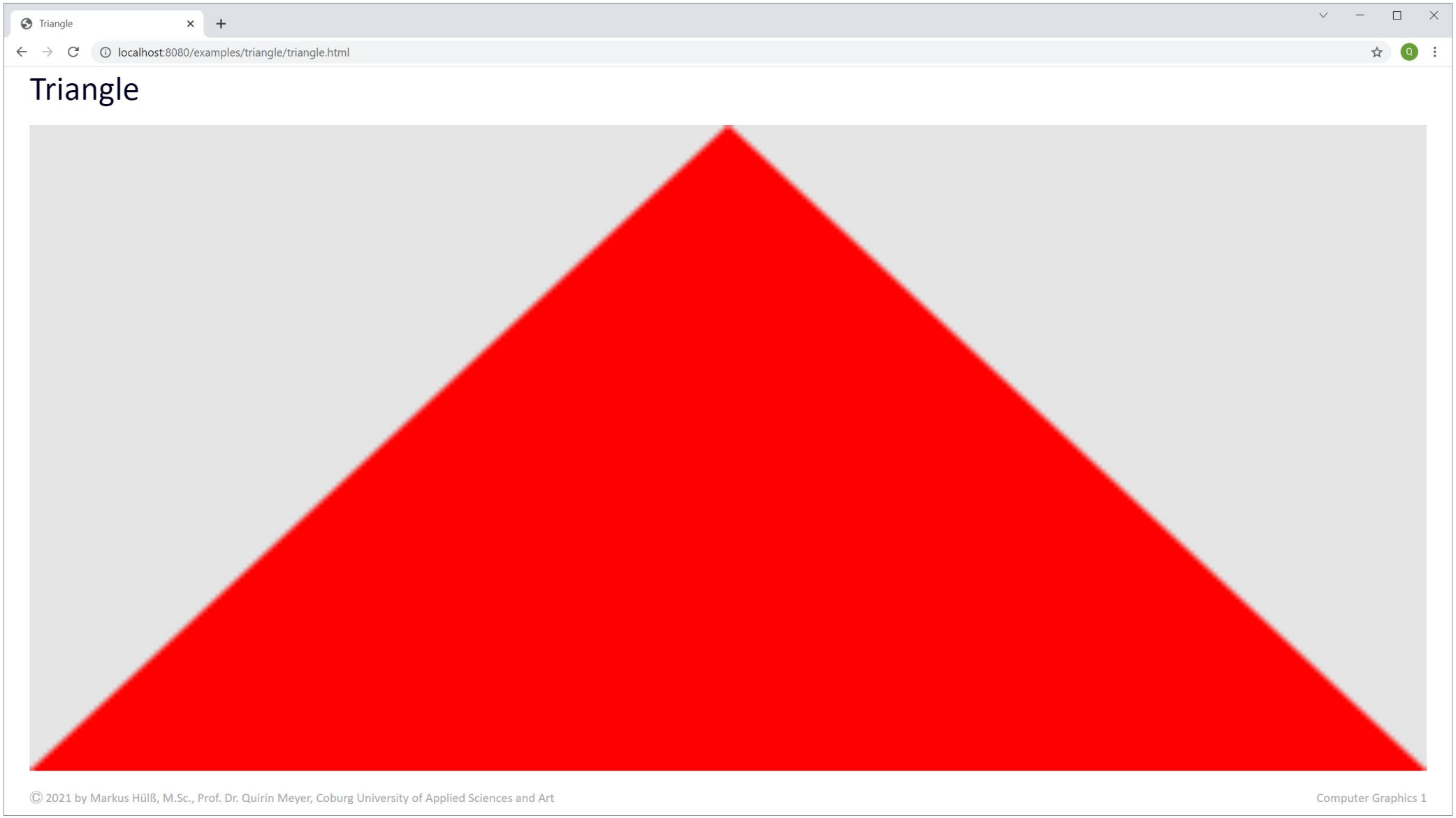
```
import GLSLProgram from "./../../lib/helper/glsl-program.js";
import { loadDataFromURL } from "./../../lib/helper/http.js";
function SimpleTriangle() {
   / …
  var mGlslProgram = null;
  async function setup() {
    // …
    const vertexShaderUrl = document.querySelector("#vertexShader").src;
    const fragmentShaderUrl = document.querySelector("#fragmentShader").src;
    mGlslProgram = new GLSLProgram(
      mCanvas, await loadDataFromURL(vertexShaderUrl), await loadDataFromURL(fragmentShaderUrl)
    );
// …
  }
  function draw() {
// …
    mGlslProgram.use();
    gl.bindVertexArray(vao);
    gl.drawElements(gl.TRIANGLES, 3, gl.UNSIGNED_INT, 0);
    requestAnimationFrame(draw);
// …
  }
```

Shading – Connect Vertex Shading Input Attributes with Vertex Array Object

```
const positionAttributeLocation = mGlslProgram.getAttributeLocati
on("inVertex");
gl.enableVertexAttribArray(positionAttributeLocation);
gl.vertexAttribPointer(positionAttributeLocation, 4, gl.FLOAT, fa
lse, 0, 0);
```



Shading – Result



Shader Input – More Input Attributes

Vertex Shader

```
layout (location=0)
in vec4 inVertex;

layout (location=1)
in vec3 inColor;

void main()
{
    gl_Position = inVertex;
    // Do something with inColor

}
```

- Add more input put attributes
- To do so, follow the slides

"Triangle Mesh – GPU"

"Shading – Connect Shader with Vertex Array Object"

"Shading – Tell Vertex Shader Format of Input Attribute"

"Shading – Enable Vertex Attribute"

Shader Input – Varying Variables

Vertex Shader

```
layout (location=0)
in vec4 inVertex;
out vec3 v_color;
void main()
{
    gl_Position = inVertex;
    v_color = abs(inVertex.xyz);
}
```

Fragment Shader

```
out vec4 fragColor;
in vec3 v_color;

void main()
{
    fragColor = vec4(v_color, 1.0);
}
```

- **Input** to a fragment shader:prefixed with `in` qualifier
- **Output** of vertex shader: prefixed with `out` qualifier

  (in fact, shader stage, that is active before fragment shader)

Shader Input – Varying Variables

Vertex Shader

```
layout (location=0)
in vec4 inVertex;
out vec3 v_color;
void main()
{
    gl_Position = inVertex;
    v_color = abs(inVertex.xyz);
}
```

Fragment Shader

```
out vec4 fragColor;
in vec3 v_color;

void main()
{
    fragColor = vec4(v_color, 1.0);
}
```

- **Name** and **type** must match between successive shader stages
- Otherwise linker cannot connect them

Shader Input – Varying Variables

Vertex Shader

```
layout (location=0)
in vec4 inVertex;
out vec3 v_color;
void main()
{
    gl_Position = inVertex;
    v_color = abs(inVertex.xyz);
}
```

Fragment Shader

```
out vec4 fragColor;
in vec3 v_color;

void main()
{
    fragColor = vec4(v_color, 1.0);
}
```

- Must be written in vertex shader
- Must be read in fragement shader
- Otherwise, compiler/linker optimizes it away.

Shader Input – Varying Variables – Example – Vertex Shader

Position buffer on the GPU

pb	[0, 1, 0, 1] [1, -1, 0, 1] [-1, -1, 0, 1] [1, 1, 0, 1]

Vertex Shader

gl_Position

[0, 1, 0, 1]
[1, -1, 0, 1]
[-1, -1, 0, 1]
[-1, -1, 0, 1]

v_color

[0, 1, 0]
[1, 1, 0]
[1, 1, 0]
[1, 1, 0]

Primitive Assembler

Vertex Shader

```
layout (location=0)
in vec4 inVertex;
out vec3 v_color;
void main()
{
    gl_Position = inVertex;
    v_color = abs(inVertex.xyz);
}
```

p_0

Vertex Shader

- Vertex shader operators on **per-vertex attributes**
  - `inVertex`
  - `v_color`
  - `gl_Position`
- Writes positions to `gl_Position` for every vertex
- Writes to a output vertex attribute `v_color` for every vertex

Shader Input – Varying Variables – Example – Primitive Assembler

Position buffer on the GPU

pb	[0, 1, 0, 1] [1, -1, 0, 1] [-1, -1, 0, 1] [1, 1, 0, 1]

Element Array Buffer on the GPU

ib	[0, 1, 2] [0, 1, 3]

Vertex Shader

gl_Position

```
[ 0,  1, 0, 1]
[ 1, -1, 0, 1]
[-1, -1, 0, 1]
[-1, -1, 0, 1]
```

v_color

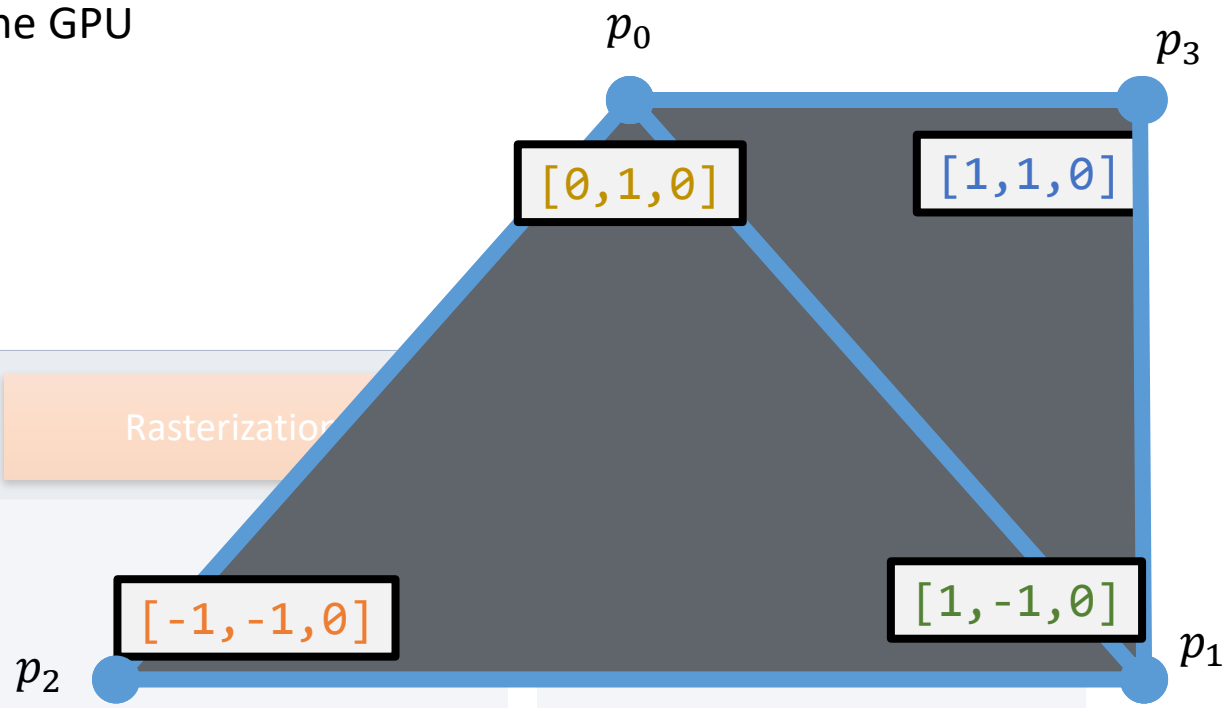
```
[ 0,  1, 0]
[ 1,  1, 0]
[ 1,  1, 0]
[ 1,  1, 0]
```

Primitive Assembler

Triangles

gl_Position	v_color
[0, 1, 0, 1]; [1, -1, 0, 1]; [-1, -1, 0, 1];	[0,1,0] [1,1,0] [1,1,0]
[0, 1, 0, 1]; [1, -1, 0, 1]; [-1, -1, 0, 1];	[0,1,0] [1,1,0] [1,1,0]

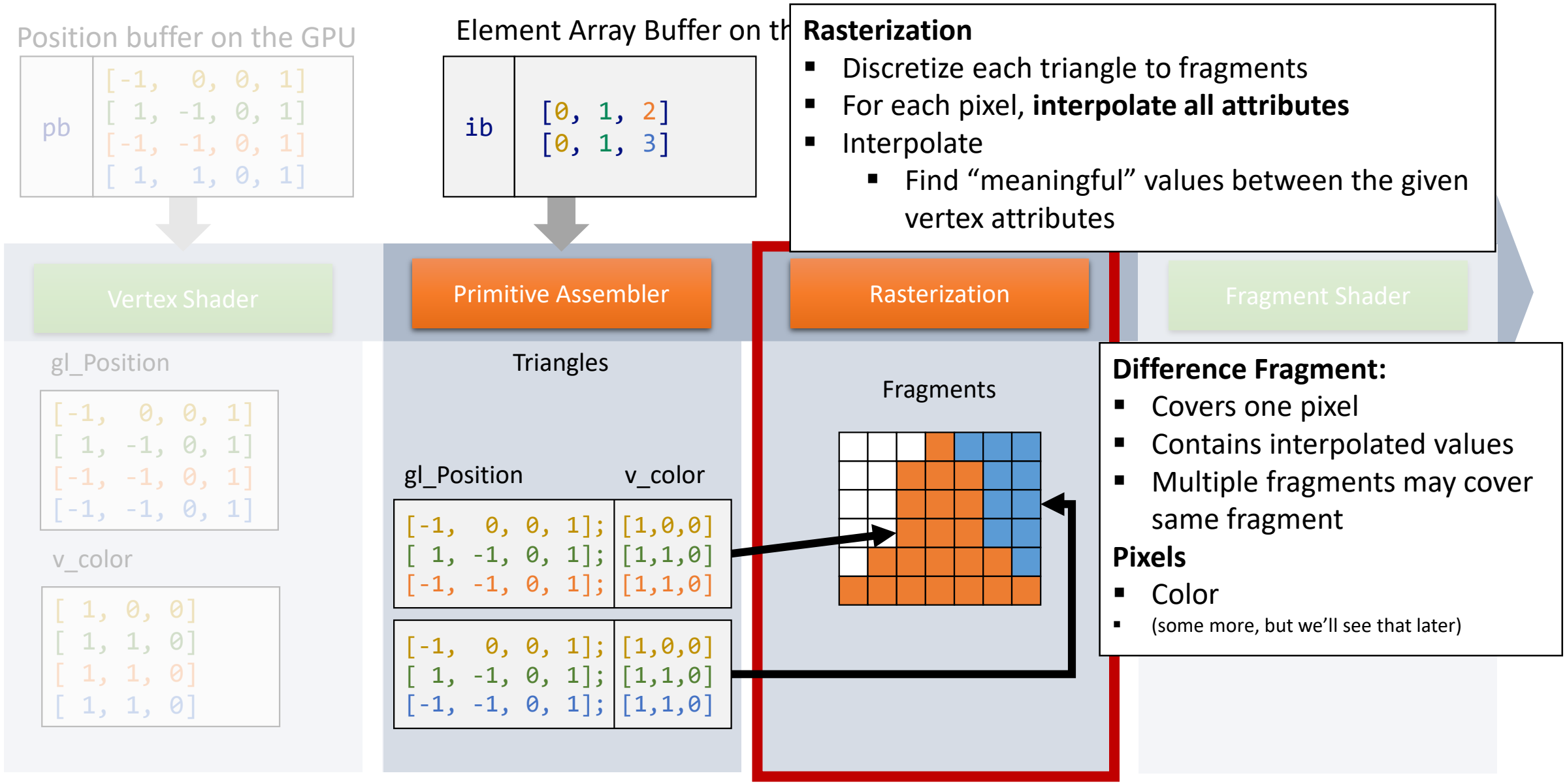
Rasterization



Primitive Assembler

- Creates individual triangles with output per-vertex attribute from vertex shader
  - `gl_Position` and
  - Other attributes (here `v_color`)

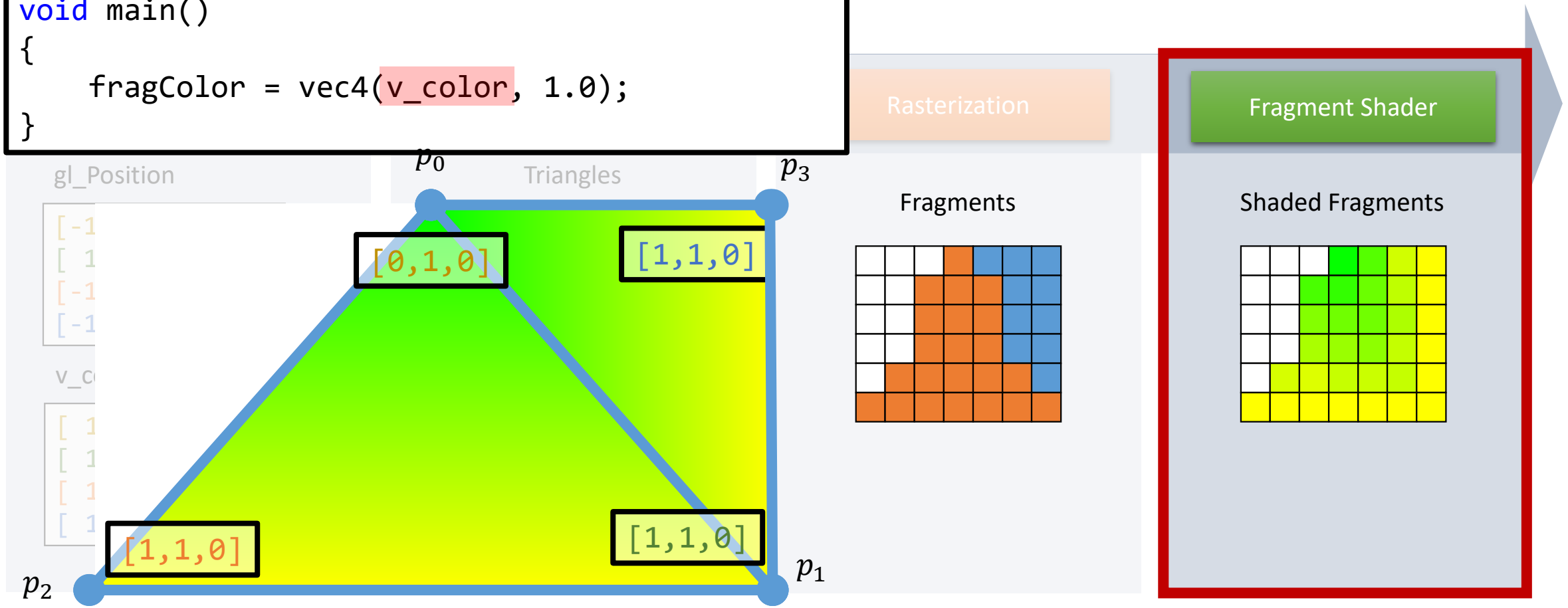
Shader Input – Varying Variables – Example – Rasterization



Shader Input – Varying Variables – Example – Fragment Shader

Fragment Shader

```
out vec4 fragColor;
in vec3 v_color;

void main()
{
    fragColor = vec4(v_color, 1.0);
}
```



Shader Input – Uniforms

Vertex Shader

```
layout (location=0)
in vec4 inVertex;
uniform float u_scale;
void main()
{
    gl_Position = vec4(u_scale *
                       inVertex.xyz, 1);
}
```

Fragment Shader

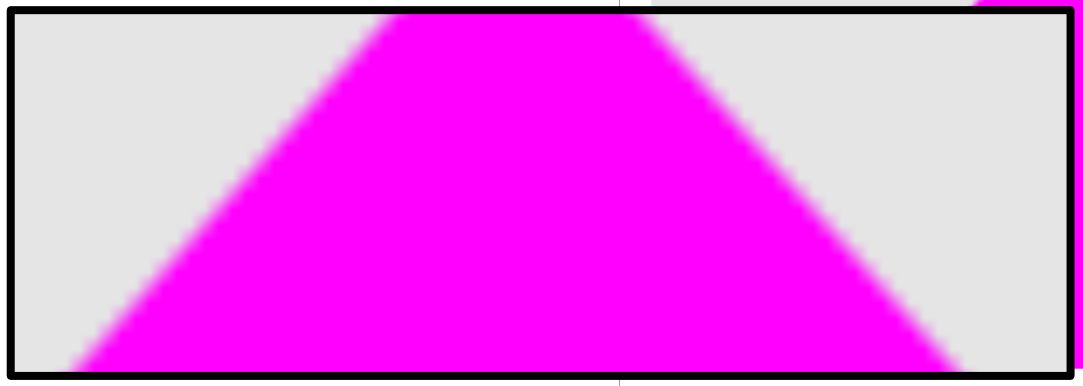
```
out vec4 fragColor;
uniform vec3 u_color;

void main()
{
    fragColor = vec4(u_color, 1.0);
}
```

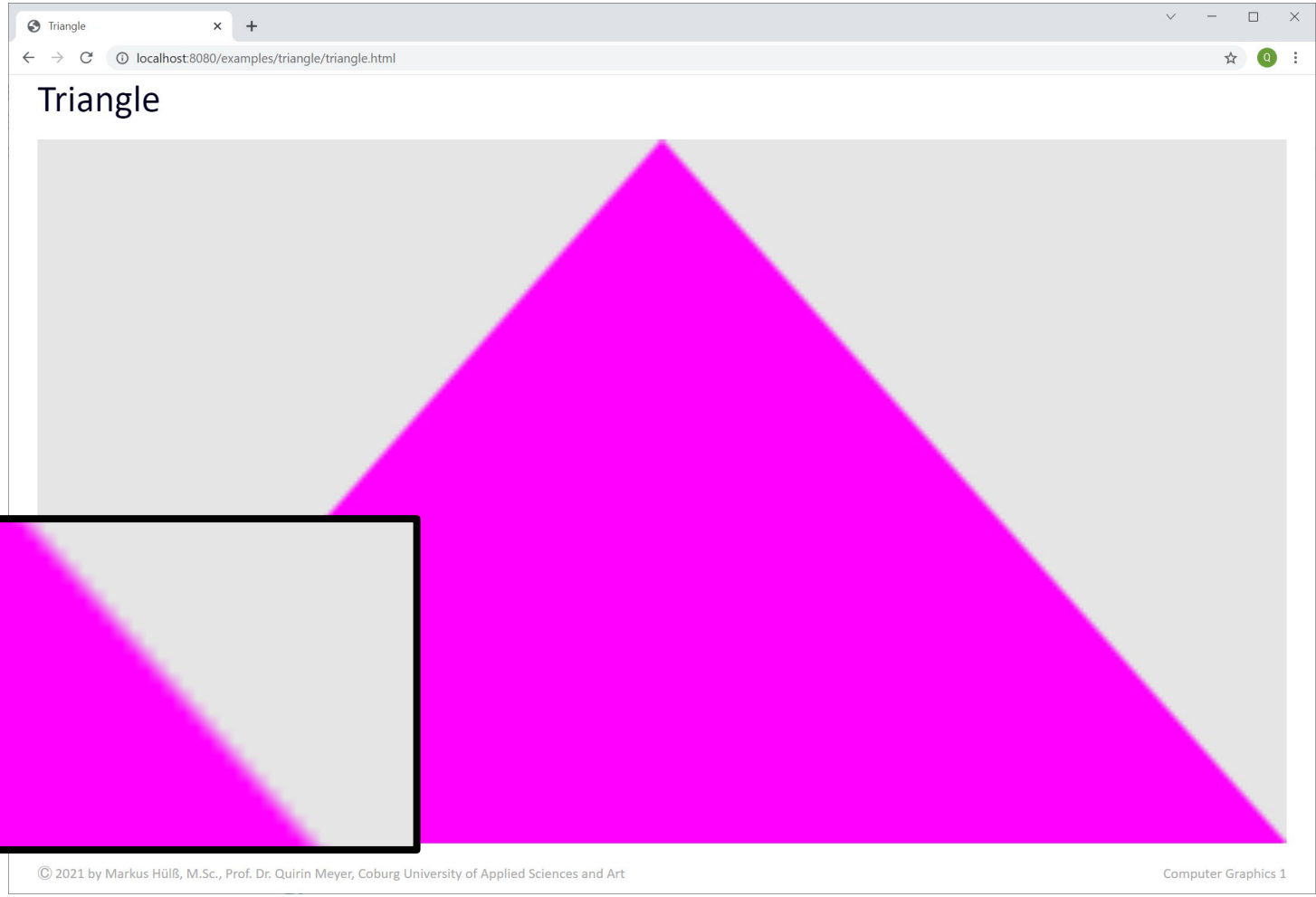
- Uniforms are constants provided by the CPU-side

```
const u_color = mGlslProgram.getUniformLocation("u_color");

mGlslProgram.use();

gl.uniform3f(u_color, 1.0, 0.0, 1.0);
```

Resize

- Artefact: Jaggies
- Reason: OpenGL Context size does not match HTML Canvas-size





Resize

- Solution: Resize before drawing

```
function draw() {
    resize();
    …
  }

  function resize() {
    var w = mCanvas.clientWidth;
    var h = mCanvas.clientHeight;

    if (mCanvas.width != w || mCanvas.height != h) {
      mCanvas.width = w;
      mCanvas.height = h;
      gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);
    }
  }
```