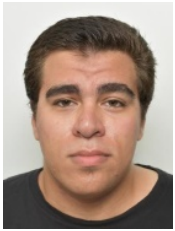

Graphics-programming with WebGPU

Bachelor Mobile Computing, Summer Semester 2022

Shehata Abd El Rahaman

A picture of me

Agenda

Agenda

- Computer Graphics

Agenda

- Computer Graphics
- Tech Stack

Agenda

- Computer Graphics
- Tech Stack
- Final Result

Agenda

- Computer Graphics
- Tech Stack
- Final Result
- How to draw a Triangle

Agenda

- Computer Graphics
- Tech Stack
- Final Result
- How to draw a Triangle
  - Shaders
  - Walkthrough
  - Result

Agenda

- Computer Graphics
- Tech Stack
- Final Result
- How to draw a Triangle
  - Shaders
  - Walkthrough
  - Result
- Learnings

Computer Graphics

Intro

Computer Graphics

The creation of, manipulation of, analysis of, and interaction with pictorial representations of objects and data using computers.

- Dictionary of Computing

Intro

Computer Graphics

The creation of, manipulation of, analysis of, and interaction with pictorial representations of objects and data using computers.

- Dictionary of Computing

Applications:

- Entertainment: Movies, Video games
- Graphical user interface (GUI)
- Computer aided design and manufacturing (CAD/CAM)
- Medical application
- Scientific visualization / simulation
- etc..

Tech Stack

Language and Tech

Figure: Rust logo

Figure: WebGPU, logo

Figure: glTF

Figure: Rust mascot (Ferris)

Figure: Wgpu logo

Final Result

3D Models

Figure: Stanford Dragon

3D Models

Figure: Stanford Dragon

Figure: Armadillo

How to draw a Triangle

Modern Graphics Pipeline

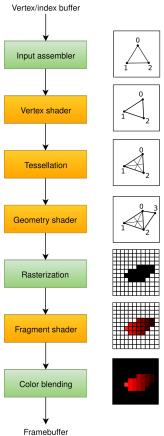

Figure: Modern Graphics Pipeline

Shaders

- Vertex Shader
  - Must be always provided
- Fragment Shader
  - Also known as Pixel Shader
- Compute Shader
  - Used for general purpose computation that can be parallized
- Geomery Shader
  - Allows to manipulate primitives (Point/Line/Triangle)
- Raytracing has 5 shaders and is an extenion
  - Ray Generation Shader
  - Closest-Hit Shader
  - Miss Shader
  - Intersection Shader
  - Any-Hit Shader

Shaders

- ✔ Vertex Shader
  - Must be always provided
- ✔ Fragment Shader
  - Also known as Pixel Shader
- Compute Shader
  - Used for general purpose computation that can be parallized
- Geomery Shader
  - Allows to manipulate primitives (Point/Line/Triangle)
- Raytracing has 5 shaders and is an extenion
  - Ray Generation Shader
  - Closest-Hit Shader
  - Miss Shader
  - Intersection Shader
  - Any-Hit Shader

Shader Code

```
[[stage(vertex)]]
fn main(
    [[location(0)]] aPos: vec3<f32>,
) -> [[builtin(position)]] vec4<f32> {
    return vec4<f32>(aPos, 1.0);
}
```

Figure: Vertex Code

```
[[stage(fragment)]]
fn main() -> [[location(0)]] vec4<f32> {
    return vec4<f32>(1.0, 0.0, 0.0, 1.0);
}
```

Figure: Fragment Code

Rust code

1. Create a Window and a GPU-Instance
2. Create a Surface from the Window
3. Request for an Adapter (Your GPU)
4. Request for a Queue and a Device from the Adapter.
5. Create Vertices

```rust
let vertices = [
    [0.0, 0.5, 0.0],
    [-0.5, -0.5, 0.0],
    [0.5, -0.5, 0.0],
];
```

6. Compile the Shader

```
device.create_shader_module(&wgpu::ShaderModuleDescriptor {
    label: Some("Triangle <<Stage>> Shader"),
    source: wgpu::ShaderSource::Wgsl(include_str!("path/to/shader").into()),
});
```

7. Create GPU Buffer and Description

```
let vertex_buffer = device
    .create_buffer_init(&wgpu::util::BufferInitDescriptor {
        label: Some("Vertex Buffer"),
        contents: bytemuck::cast_slice(&vertices),
        usage: wgpu::BufferUsages::VERTEX,
    });
```

```
let desc = wgpu::VertexBufferLayout {
    array_stride: std::mem::size_of::<Vertex>() as wgpu::BufferAddress,
    step_mode: wgpu::VertexStepMode::Vertex,
    attributes: &wgpu::vertex_attr_array![0 => Float32x3],
};
```

8. Create Render Pipeline

```
let render_pipeline = device.create_render_pipeline(&wgpu::RenderPipelineDescriptor {
    label: Some("Render Pipeline"),
    primitive: wgpu::PrimitiveState {
        topology: wgpu::PrimitiveTopology::TriangleList,
        front_face: wgpu::FrontFace::Ccw,
        cull_mode: Some(wgpu::Face::Back),
        polygon_mode: PolygonMode::Fill,
        ..Default::default()
    },
    vertex: wgpu::VertexState { .. },
    fragment: Some(wgpu::FragmentState { .. }),
    ..Default::default()
});
```

9. Draw and submit to Queue

```
let mut render_pass = encoder.begin_render_pass(&wgpu::RenderPassDescriptor { .. });
render_pass.set_vertex_buffer(0, vertex_buffer.slice(..));
render_pass.set_pipeline(render_pipeline);
render_pass.draw(0..state.vertices.len() as u32, 0..1);
queue.submit(std::iter::once(encoder.finish()));
```

10. And now to the Result

Result

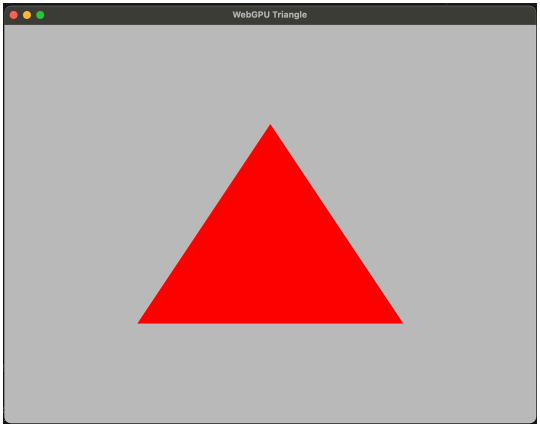

Figure: Red Triangle

Learnings

- Rust is AWESOME!!

Learnings

- Rust is AWESOME!!
- Graphics promgramming involves a lot of Math

Learnings

- Rust is AWESOME!!
- Graphics promgramming involves a lot of Math
- In Math there is something called a Quaternion.

Learnings

- Rust is AWESOME!!
- Graphics promgramming involves a lot of Math
- In Math there is something called a Quaternion. For further questions please ask Prof. Ostermayer

Learnings

- Rust is AWESOME!!
- Graphics promgramming involves a lot of Math
- In Math there is something called a Quaternion. For further questions please ask Prof. Ostermayer
- Drawing Trivial shapes is harder than in OpenGL.
  But with great power comes great responsibility

Thank You !!