Evolution Engineering

WebGL

Ivan Kaliada

What could be done

https://tympanus.net/Tutorials/apple-fifth-avenue/

https://lusionreplica.netlify.app/

https://bruno-simon.com/

https://tympanus.net/Tutorials/codrops-kinetic-typo/

https://threejs.org/examples/?q=physics#physics_ammo_break

What is WebGL

WebGL runs on the GPU on your computer. You provide that code in the form of pairs of functions. Those 2 functions are called a vertex shader and a fragment shader and they are each written in a very strictly typed C/C++ like language called GLSL. (GL Shader Language). Paired together they are called a program.

Connecting CPU to GPU

1. Buffers and attributes (positions or other arrays of data)
2. Uniforms (constants)
3. Textures (images)
4. Varying (vertex to fragment)

GLSL Data types

GLSL allows for three basic types of data:

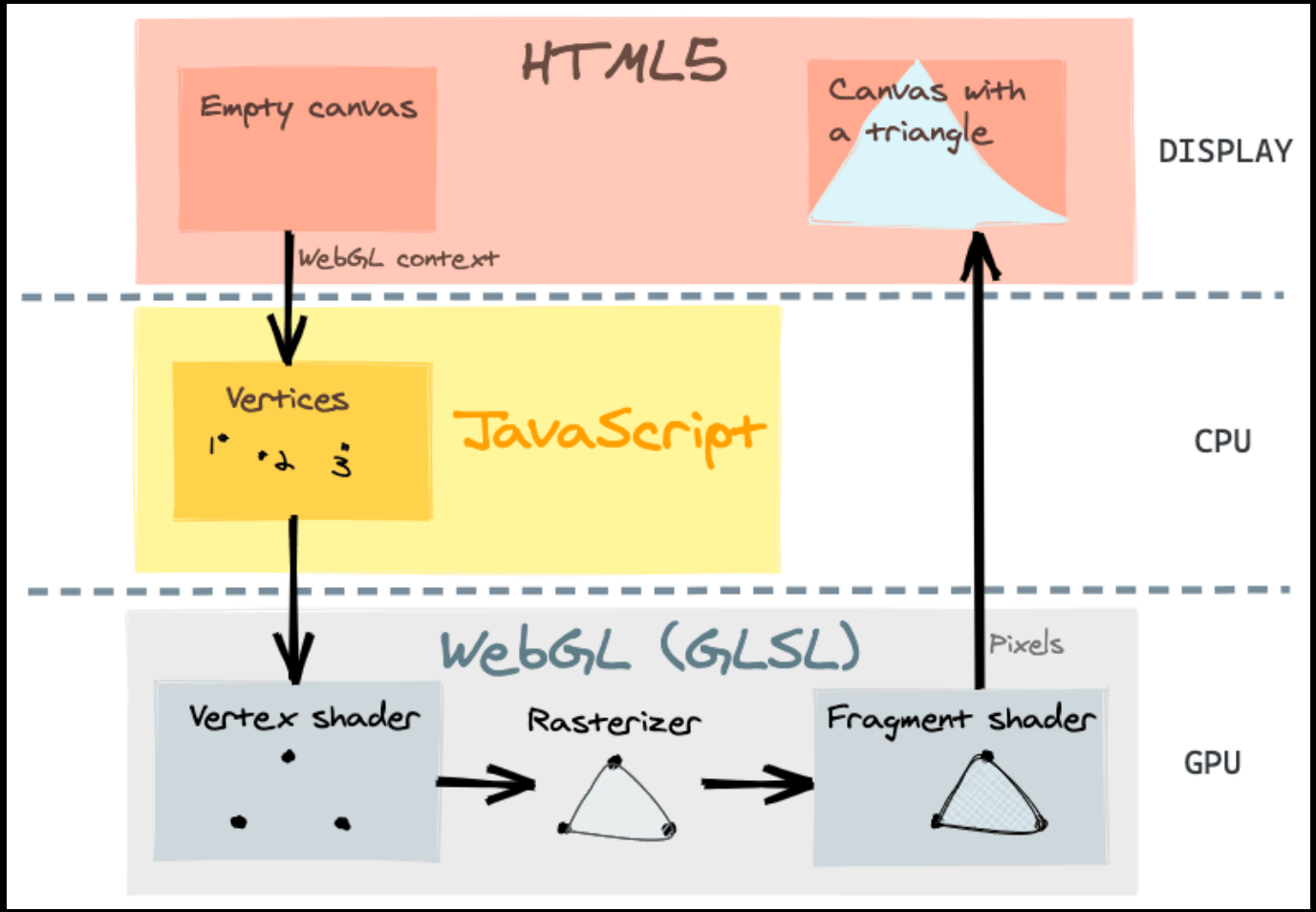
- `bool` : Boolean values; `true` or `false`
- `int` : integer values; whole numbers in a certain range, -n..n
- `float` : floating point values; numbers with a fractional component

GLSL facilitates the manipulation of vectors and matrices. A vector or matrix is always composed of values of the same basic data type. If a data type starts with a ‘b’ it contains Boolean values; if it starts with an ‘i’ it contains integer values; if it starts with anything else it contains floating point values. The vector and matrix data types are:

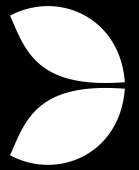
- `bvec2` , `bvec3` , `bvec4` : 2, 3, and 4-component Boolean vectors
- `ivec2` , `ivec3` , `ivec4` : 2, 3, and 4-component integer vectors
- `vec2` , `vec3` , `vec4` : 2, 3, and 4-component floating point vectors
- `mat2` , `mat3` , `mat4` : 2x2, 3x3, and 4x4 floating point matrices

There are three other specialized data types:

- `sampler2D` : a reference to a TEXTURE_2D *texture unit* (which has an attached *texture object*)
- `samplerCube` : a reference to a SAMPLER_CUBE *texture unit*
- `void` : used to identify functions that do not return a value or parameter lists to a function that are empty.


HTML5
Empty canvas
Canvas with a triangle
DISPLAY
WebGL context
Vertices
1
2
3
JavaScript
CPU
WebGL (GLSL)
Pixels
Vertex shader
Rasterizer
Fragment shader
GPU

Vertex Shader

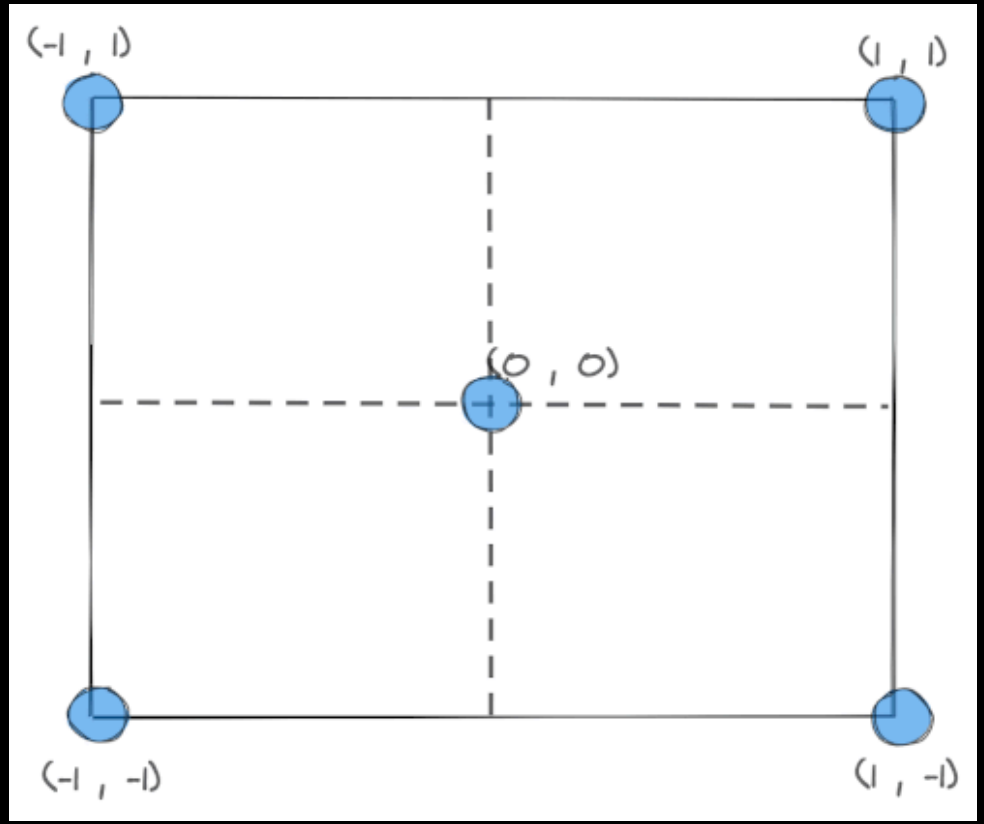


A vertex shader's job is to compute vertex positions

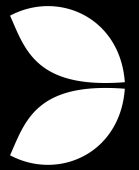
```
attribute vec4 a_position;

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

Clip space (пространство отсечения)



Fragment Shader



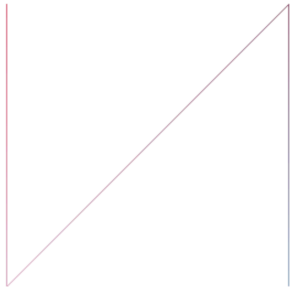
A fragment shader's job is to compute a color for each pixel of the primitive currently being drawn.

```
void main() {
  gl_FragColor = vec4(1, 0, 0.5, 1);
}
```

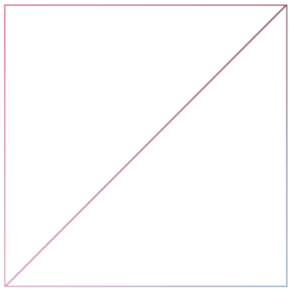
Setting up basic scene with WebGL (code)

Passing coordinates buffer to GPU(code)

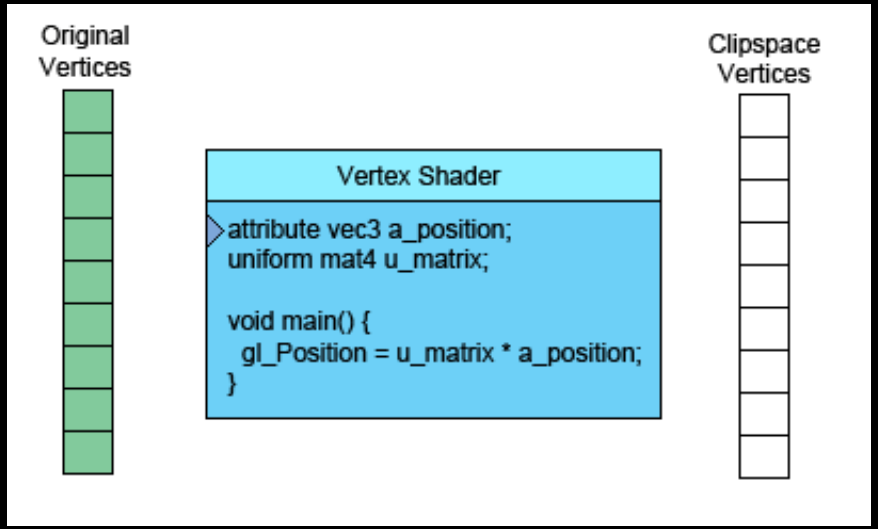
Draw modes



gl.POINTS
gl.LINES
gl.LINE_LOOP
gl.TRIANGLES



Passing buffers visualized



https://webglfundamentals.org/webgl/lessons/resources/vertex-shader-anim.gif

Passing color uniform to GPU(code)

Varying visualized (code)

https://webglfundamentals.org/webgl/lessons/resources/fragment-shader-anim.html

Adding textures (code)

Adding glitch effect (code)

Practice

Sandbox: https://codesandbox.io/s/webgl-cube-task-from9

Goal: https://2viyo.csb.app/

2D vectors math

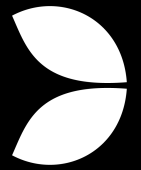
1. Translation
2. Rotation
3. Scale

2D Translation

vec2 position = a_position + u_translation;

https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-2d-geometry-translate-better.html

2D Rotation



```
float angleInRadians = 3.14;
vec2 rotation = vec2(sin(angleInRadians), cos(angleInRadians));

vec2 rotatedPosition = vec2(
   a_position.x * u_rotation.y + a_position.y * u_rotation.x,
   a_position.y * u_rotation.y - a_position.x * u_rotation.x);
```

https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-2d-geometry-rotation-angle.html

2D Scale

```
vec2 scaledPosition = a_position * u_scale;
```

https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-2d-geometry-scale.html

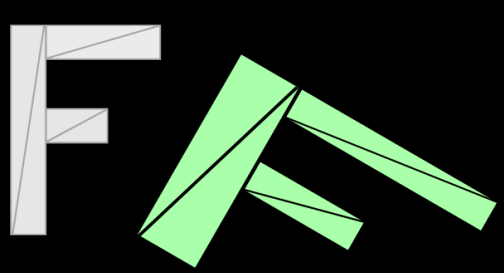
2D Rotation

vec2 scaledPosition = a_position * u_scale;

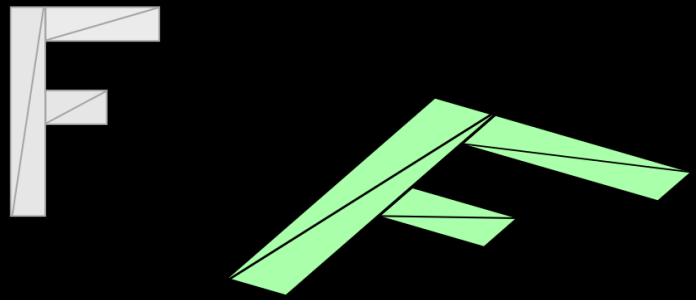
https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-2d-geometry-scale.html

Applying transformations in different order

Scale, Rotate, Translate

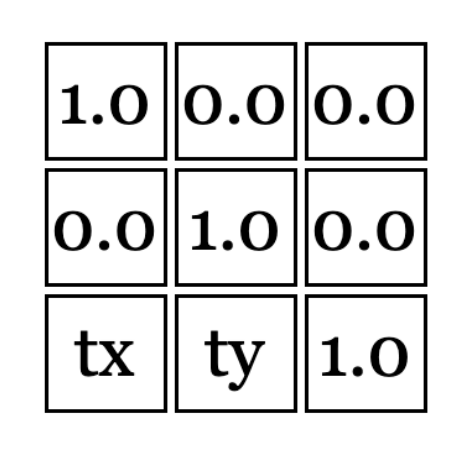


Translate, Rotate, Scale

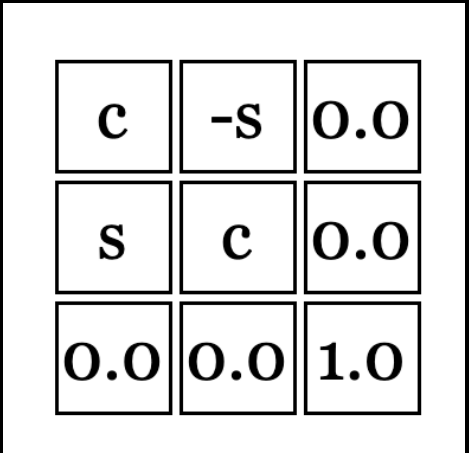


Matrices

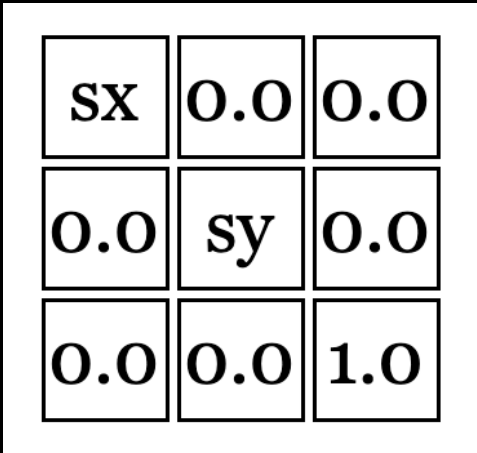
Translate



Rotate

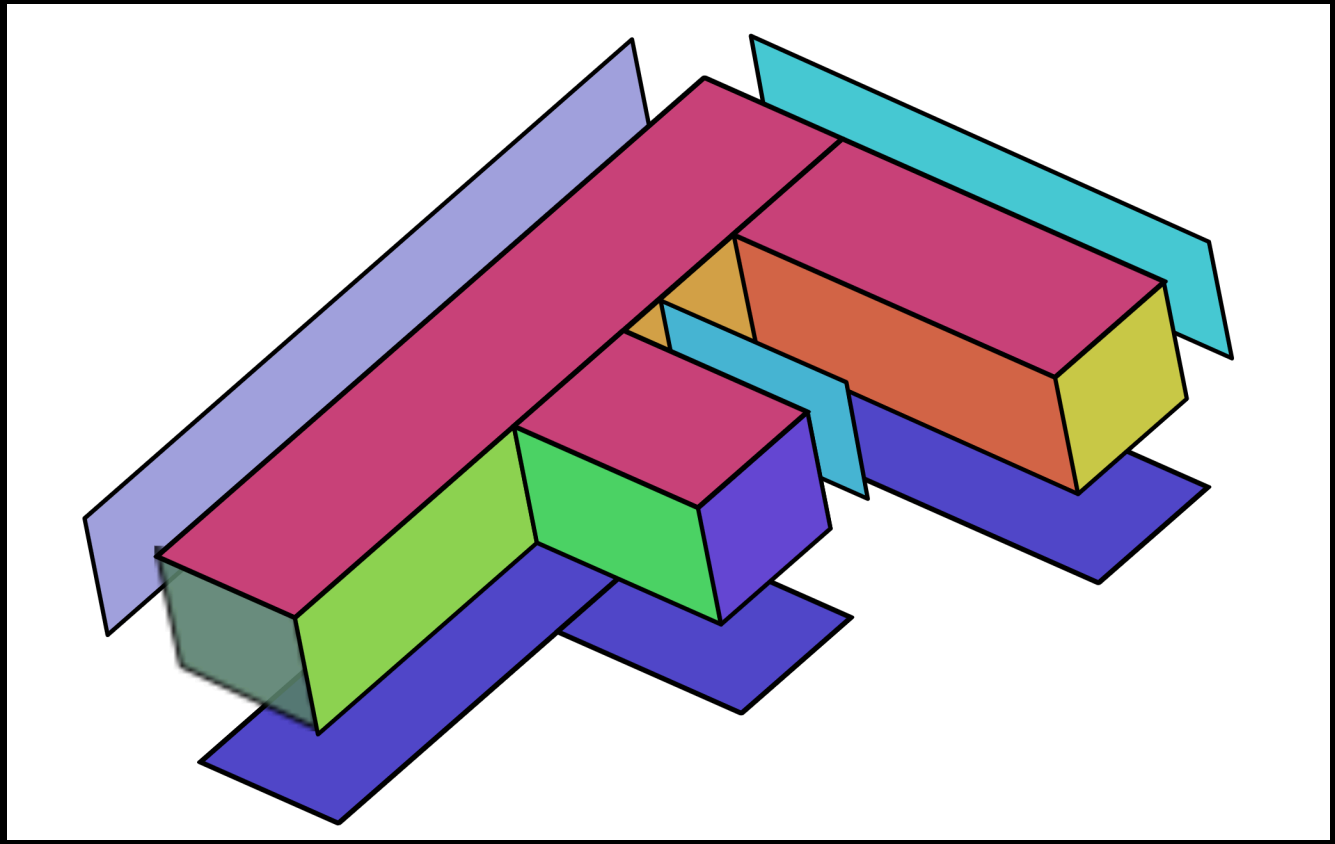


Scale



https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-2d-geometry-matrix-transform.html

Moving to 3D world



Passing position buffers

https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-3d-step2.html

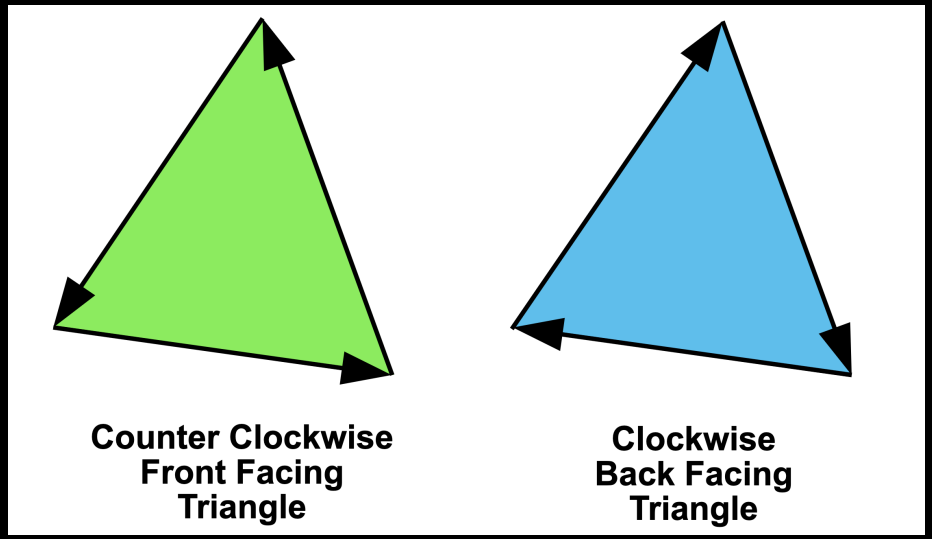
Adding color buffers

https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-3d-step3.html

Why we see back faces, order of drawing:

https://webglfundamentals.org/webgl/lessons/resources/polygon-drawing-order.gif

Disable back facing triangles



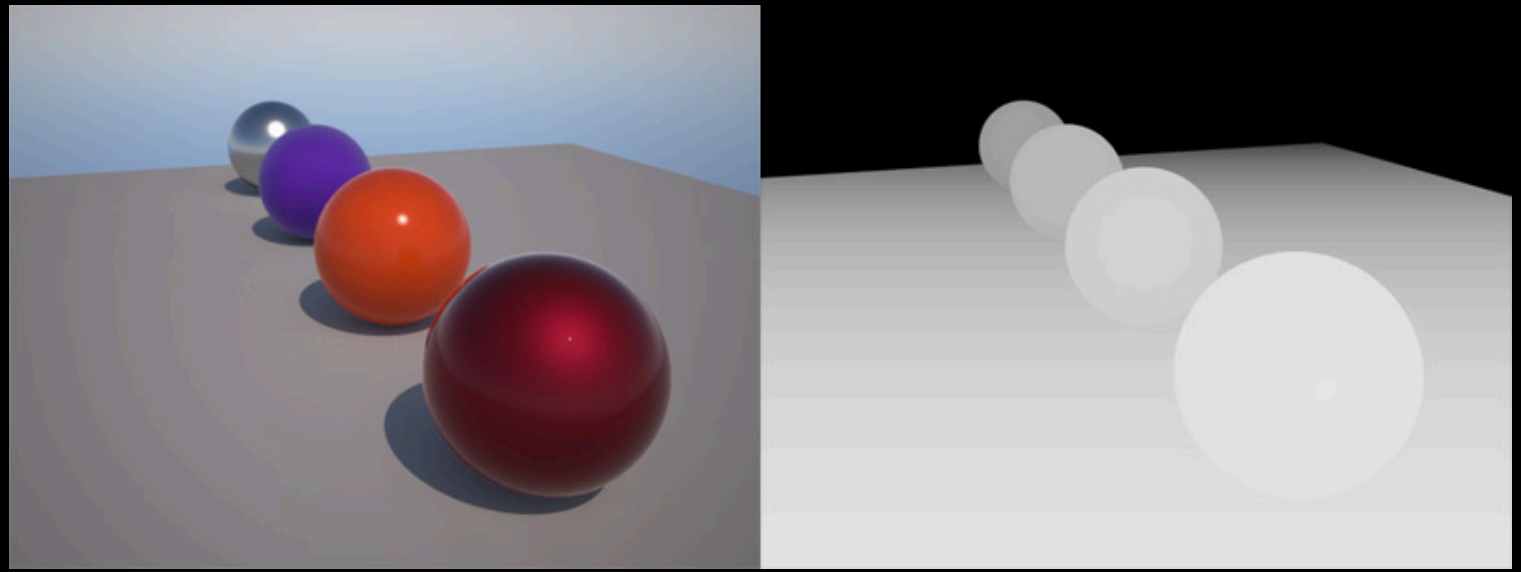
gl.enable(gl.CULL_FACE);

https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-3d-step5.html

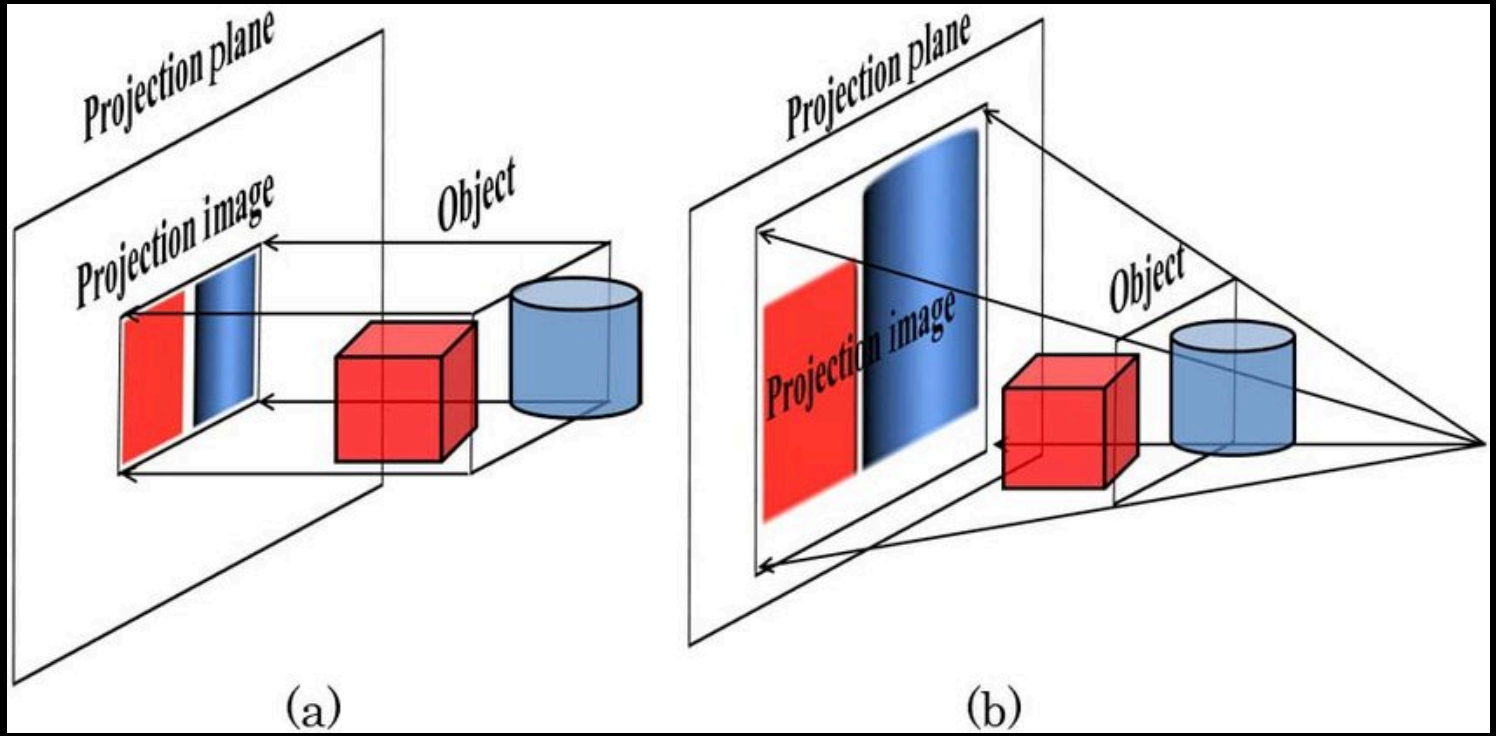
Enabling depth buffer

gl.enable(gl.DEPTH_TEST);

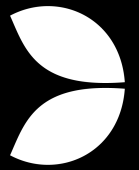
https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-3d-step6.html



Projection


Projection plane
Projection image
Object
(a)
Projection plane
Projection image
Object
(b)

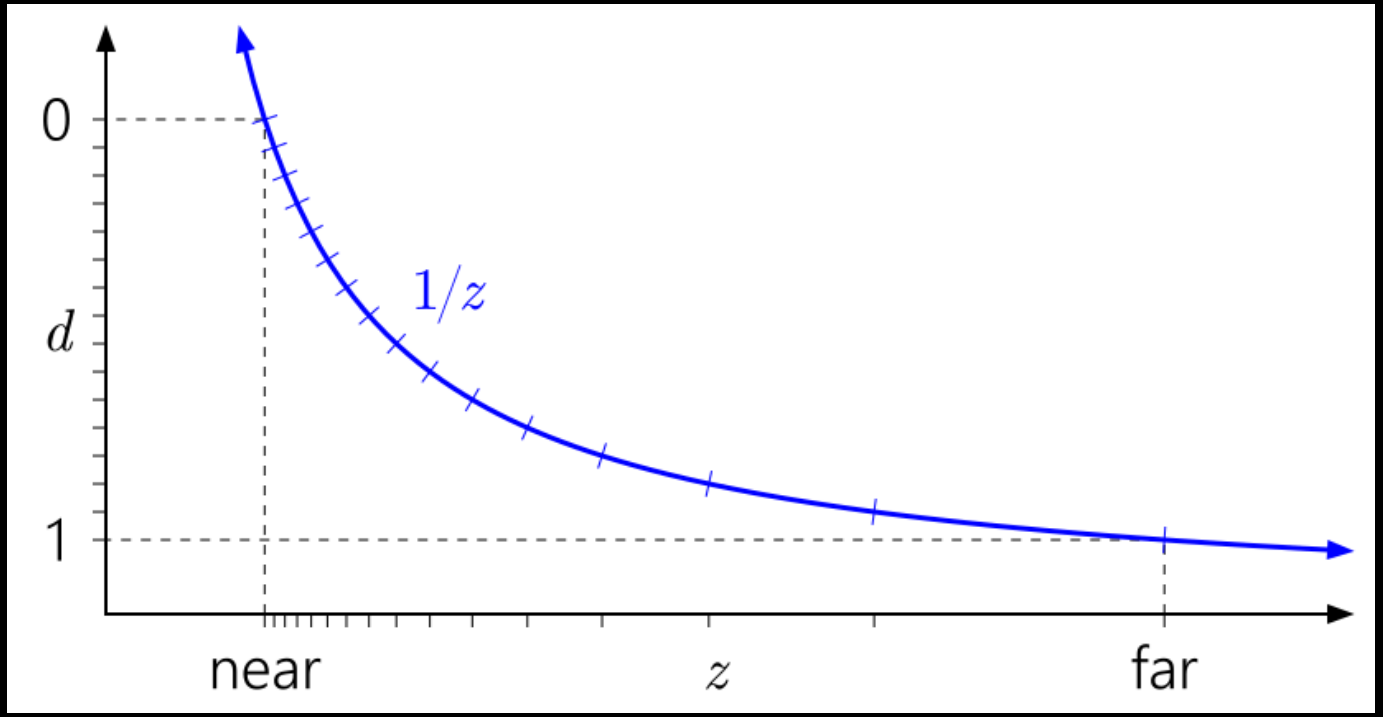
Frustum (Усеченная пирамида)



https://webglfundamentals.org/webgl/frustum-diagram.html

```
const f = Math.tan(Math.PI * 0.5 - 0.5 * fieldOfViewInRadians);
const rangeInv = 1.0 / (near - far);
const perspectiveMatrix = [
  f / aspect, 0, 0, 0,
  0, f, 0, 0,
  0, 0, (near + far) * rangeInv, -1,
  0, 0, near * far * rangeInv * 2, 0
]
```

Why near cannot be 0



Perspective camera

https://webglfundamentals.org/webgl/resources/editor.html?url=/webgl/lessons/..%2Fwebgl-3d-perspective-matrix.html

FOV: 90

FOV: 110

WebGPU

https://www.youtube.com/watch?v=eYgkDymaNr8

Links

https://webglfundamentals.org/webgl/lessons/ru/
https://thebookofshaders.com/