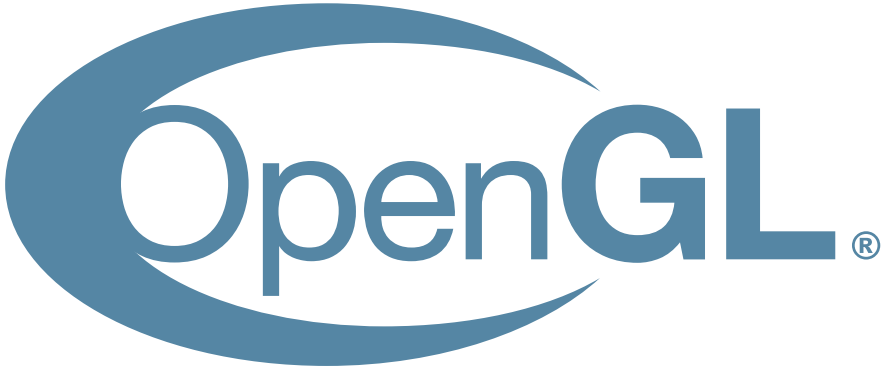
A Short Introduction to OpenGL

Jan Wedekind

Thursday, May 30th 2024

Motivation



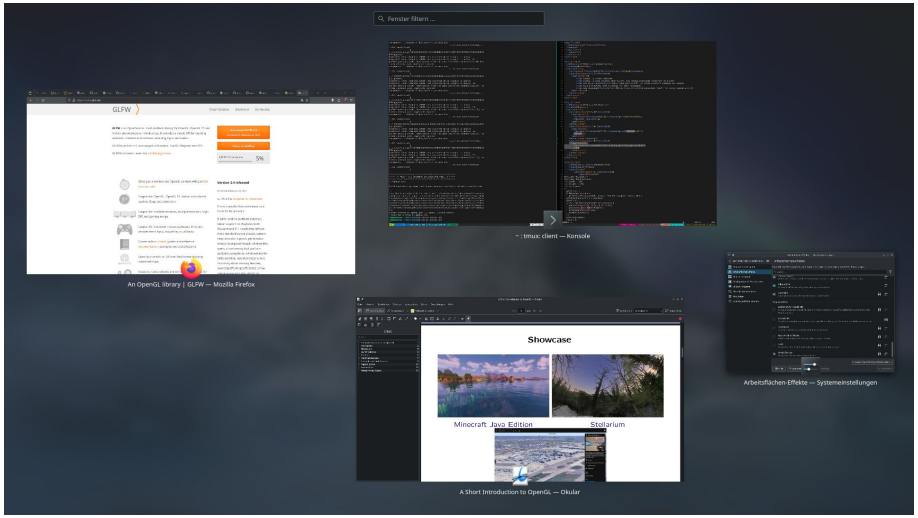
- real-time visualisation or parallel computation
- cross-platform
- less verbose than Vulkan, but knowledge transferable to Vulkan
- bindings for many programming languages

Showcase

Minecraft Java Edition

Stellarium

WebGL (e.g. Google Earth)



KDE Plasma desktop

Getting Started: GLEW and GLFW

OpenGL interface and window management

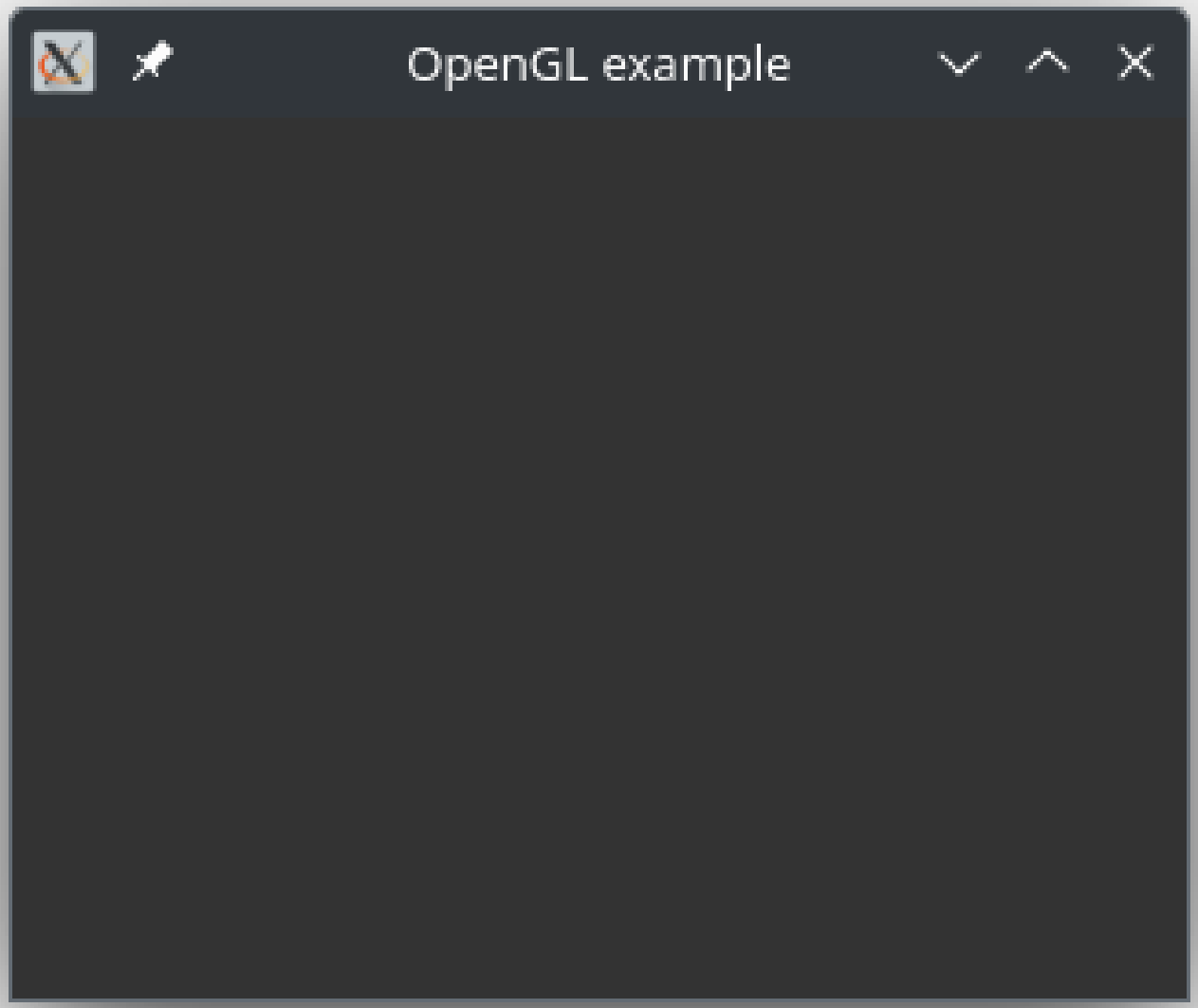
- OpenGL Extension Wrangler Library (GLEW[a]) to access graphics driver
- Graphics Library Framework (GLFW[b]) for windowing and input

[a] https://glew.sourceforge.net/
[b] https://www.glfw.org/

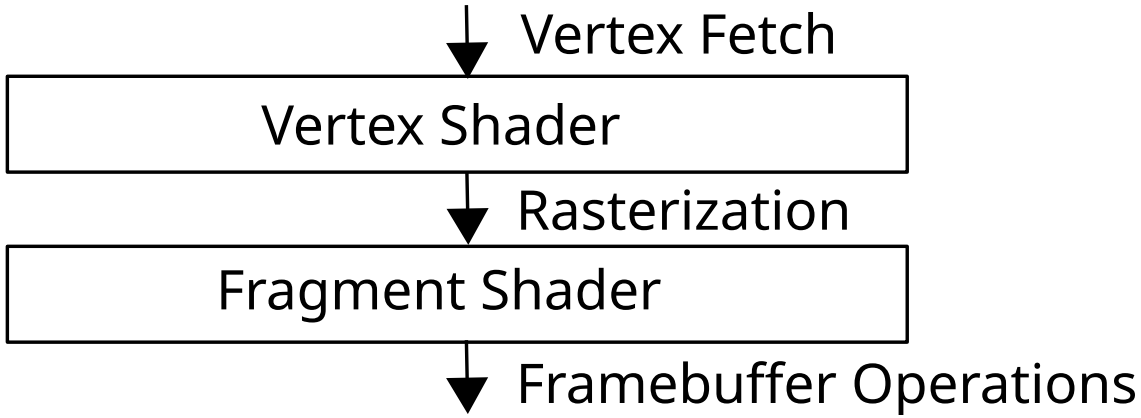
Getting Started: GLFW Window

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
int width = 320; int height = 240;
void main(void)
{
  glfwInit();
  GLFWwindow *window =
    glfwCreateWindow(width, height, "OpenGL example", NULL, NULL);
  glfwMakeContextCurrent(window);
  glewInit();
  glClearColor(0.2f, 0.2f, 0.2f, 0.0f);
  glViewport(0, 0, width, height);
  while (!glfwWindowShouldClose(window)) {
    glClear(GL_COLOR_BUFFER_BIT);
    glfwSwapBuffers(window); glfwPollEvents();
  };
  glfwTerminate();
}
```

Getting Started: Result



Minimal Pipeline: Overview



Minimal Pipeline: Shader Code (GLSL)

```glsl
#version 410 core
in vec3 point;
void main()
{
  gl_Position = vec4(point, 1);
}
```

```glsl
#version 410 core
out vec3 fragColor;
void main()
{
  fragColor = vec3(1, 0, 0);
}
```

Minimal Pipeline: C Strings

```
const char *vertexSource = "#version 410 core\n\
in vec3 point;\n\
void main()\n\
{\n\
  gl_Position = vec4(point, 1);\n\
}";

const char *fragmentSource = "#version 410 core\n\
out vec3 fragColor;\n\
void main()\n\
{\n\
  fragColor = vec3(1, 0, 0);\n\
}";
```

Minimal Pipeline: Compile & Link Shaders

```
// ...
GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexSource, NULL);
glCompileShader(vertexShader);
handleCompileError("Vertex shader", vertexShader);

GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentSource, NULL);
glCompileShader(fragmentShader);
handleCompileError("Fragment shader", fragmentShader);

GLuint program = glCreateProgram();
glAttachShader(program, vertexShader);
glAttachShader(program, fragmentShader);
glLinkProgram(program);
handleLinkError("Shader program", program);
// ...
```

Minimal Pipeline: Compile Errors

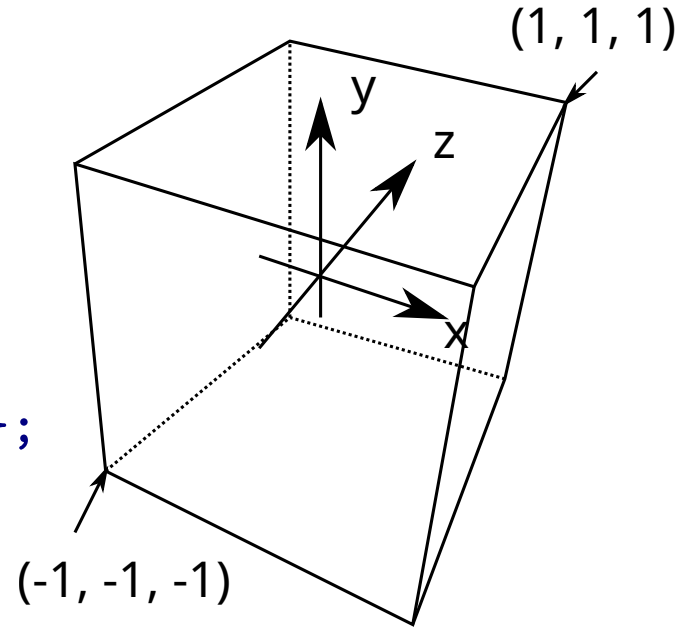
```
#include <stdio.h>
// ...
void handleCompileError(const char *step, GLuint shader)
{
  GLint result = GL_FALSE;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &result);
  if (result == GL_FALSE) {
    char buffer[1024];
    glGetShaderInfoLog(shader, 1024, NULL, buffer);
    if (buffer[0]) fprintf(stderr, "%s: %s\n", step, buffer);
  };
}
// ...
```

Minimal Pipeline: Link Errors

```
#include <stdio.h>
// ...
void handleLinkError(const char *step, GLuint program)
{
  GLint result = GL_FALSE;
  glGetProgramiv(program, GL_LINK_STATUS, &result);
  if (result == GL_FALSE) {
    char buffer[1024];
    glGetProgramInfoLog(program, 1024, NULL, buffer);
    if (buffer[0]) fprintf(stderr, "%s: %s\n", step, buffer);
  };
}
// ...
```

Pipeline Input: Vertex and Index Data

```
// ...
GLfloat vertices[] = {
  //  x,     y,     z
  -0.5f, -0.5f,  0.0f,
   0.5f, -0.5f,  0.0f,
  -0.5f,  0.5f,  0.0f,
   0.5f,  0.5f,  0.0f
};

unsigned int indices[] = {0, 1, 3, 2};

GLuint vao;
GLuint vbo;
GLuint idx;
// ...
```



normalised device coordinates (NDC)

Pipeline Input: Vertex Array Object

```
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);

glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
             GL_STATIC_DRAW);
glGenBuffers(1, &idx);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, idx);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
             GL_STATIC_DRAW);

glVertexAttribPointer(glGetAttribLocation(program, "point"),
                      3, GL_FLOAT, GL_FALSE,
                      3 * sizeof(float), (void *)0);

glUseProgram(program);
glEnableVertexAttribArray(0);
```

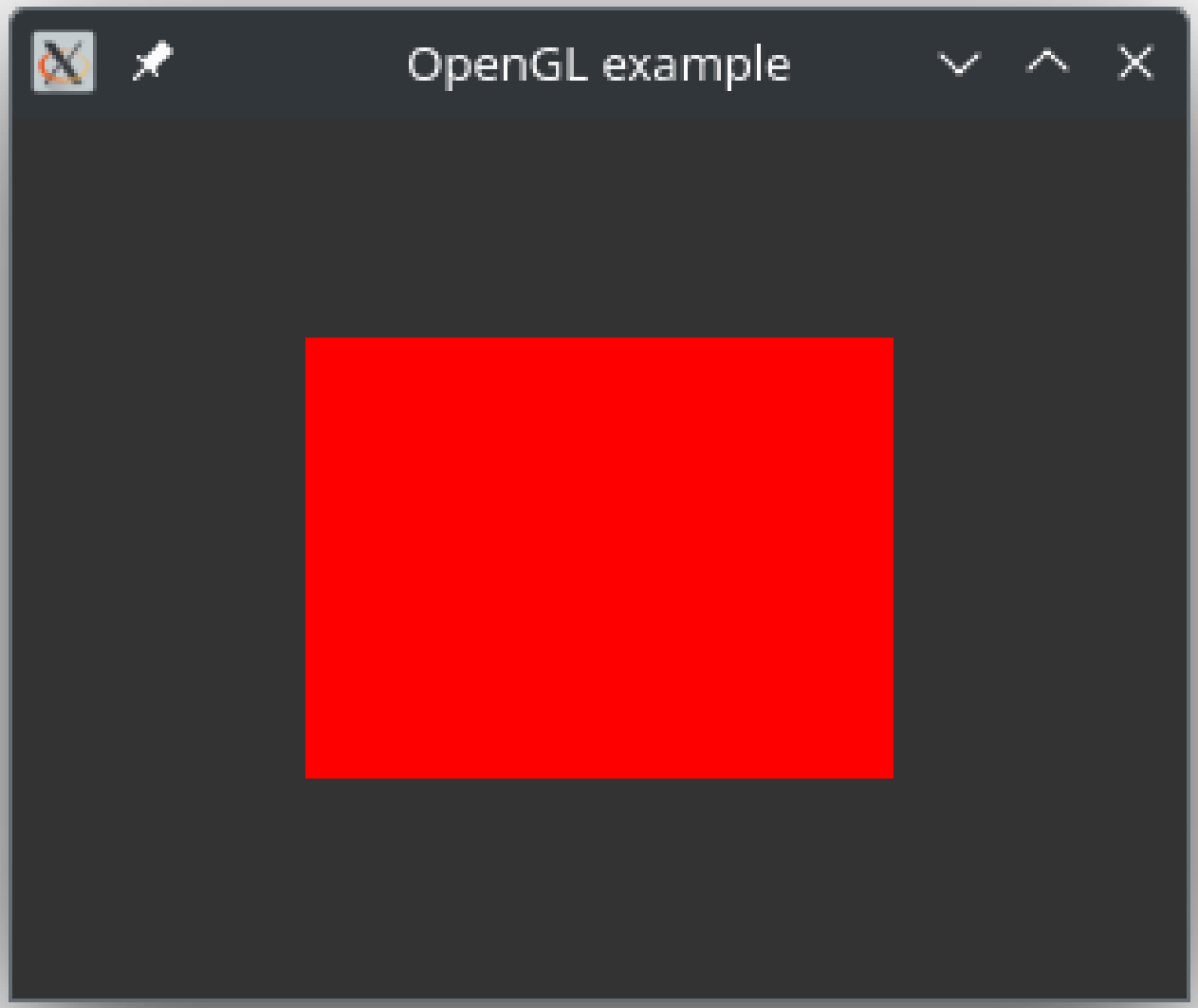
Pipeline Input: Render Quads

```
// ...
while (!glfwWindowShouldClose(window)) {
  glClear(GL_COLOR_BUFFER_BIT);
  glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, (void *)0);
  glfwSwapBuffers(window);
  glfwPollEvents();
};
// ...
```

Pipeline Input: Cleanup

```
// ...
glDisableVertexAttribArray(0);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
glDeleteBuffers(1, &idx);

glBindBuffer(GL_ARRAY_BUFFER, 0);
glDeleteBuffers(1, &vbo);

glBindVertexArray(0);
glDeleteVertexArrays(1, &vao);

glDeleteProgram(program);
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);

glfwTerminate();
// ...
```

Pipeline Input: Result



Textures: Coordinates and Pixels

```
// ...
GLfloat vertices[] = {
  //  x,     y,     z,    u,    v
  -0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
   0.5f, -0.5f,  0.0f, 6.0f, 0.0f,
  -0.5f,  0.5f,  0.0f, 0.0f, 6.0f,
   0.5f,  0.5f,  0.0f, 6.0f, 6.0f
};

float chequer[] = {
  // r,    g,    b
  0.4f, 0.4f, 0.4f,
  1.0f, 1.0f, 1.0f,
  1.0f, 1.0f, 1.0f,
  0.4f, 0.4f, 0.4f
};
// ...
```

Textures: Shaders

```
#version 410 core
in vec3 point;
in vec2 texcoord;
out vec2 UV;
void main()
{
  gl_Position = vec4(point, 1);
  UV = texcoord;
}
```

```
#version 410 core
uniform sampler2D tex;
in vec2 UV;
out vec3 fragColor;
void main()
{
  fragColor = texture(tex, UV).rgb;
}
```

Textures: Multiple Vertex Attributes

```
// ...
glVertexAttribPointer(glGetAttribLocation(program, "point"),
                      3, GL_FLOAT, GL_FALSE,
                      5 * sizeof(float),
                      (void *)0);
glVertexAttribPointer(glGetAttribLocation(program, "texcoord"),
                      2, GL_FLOAT, GL_FALSE,
                      5 * sizeof(float),
                      (void *)(3 * sizeof(float)));
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
// ...
```

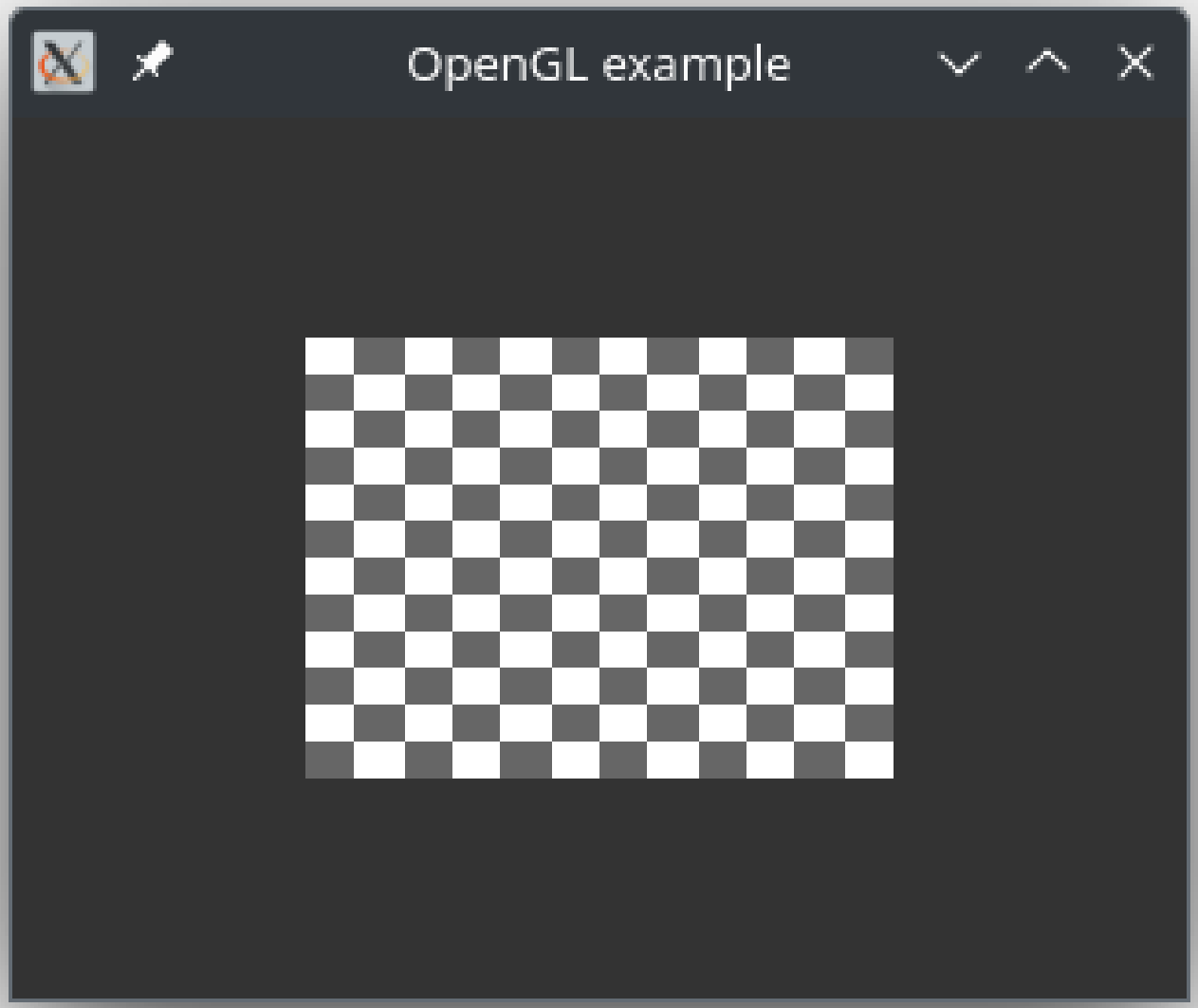
Textures: Setup

```
// ...
GLuint tex;
glGenTextures(1, &tex);
glActiveTexture(GL_TEXTURE0 + 0);
glBindTexture(GL_TEXTURE_2D, tex);
glUniform1i(glGetUniformLocation(program, "tex"), 0);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 2, 2, 0, GL_BGR,
             GL_FLOAT, chequer);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
// ...
```

Textures: Cleanup

```
// ...
glBindTexture(GL_TEXTURE_2D, 0);
glDeleteTextures(1, &tex);
// ...
```

Textures: Result



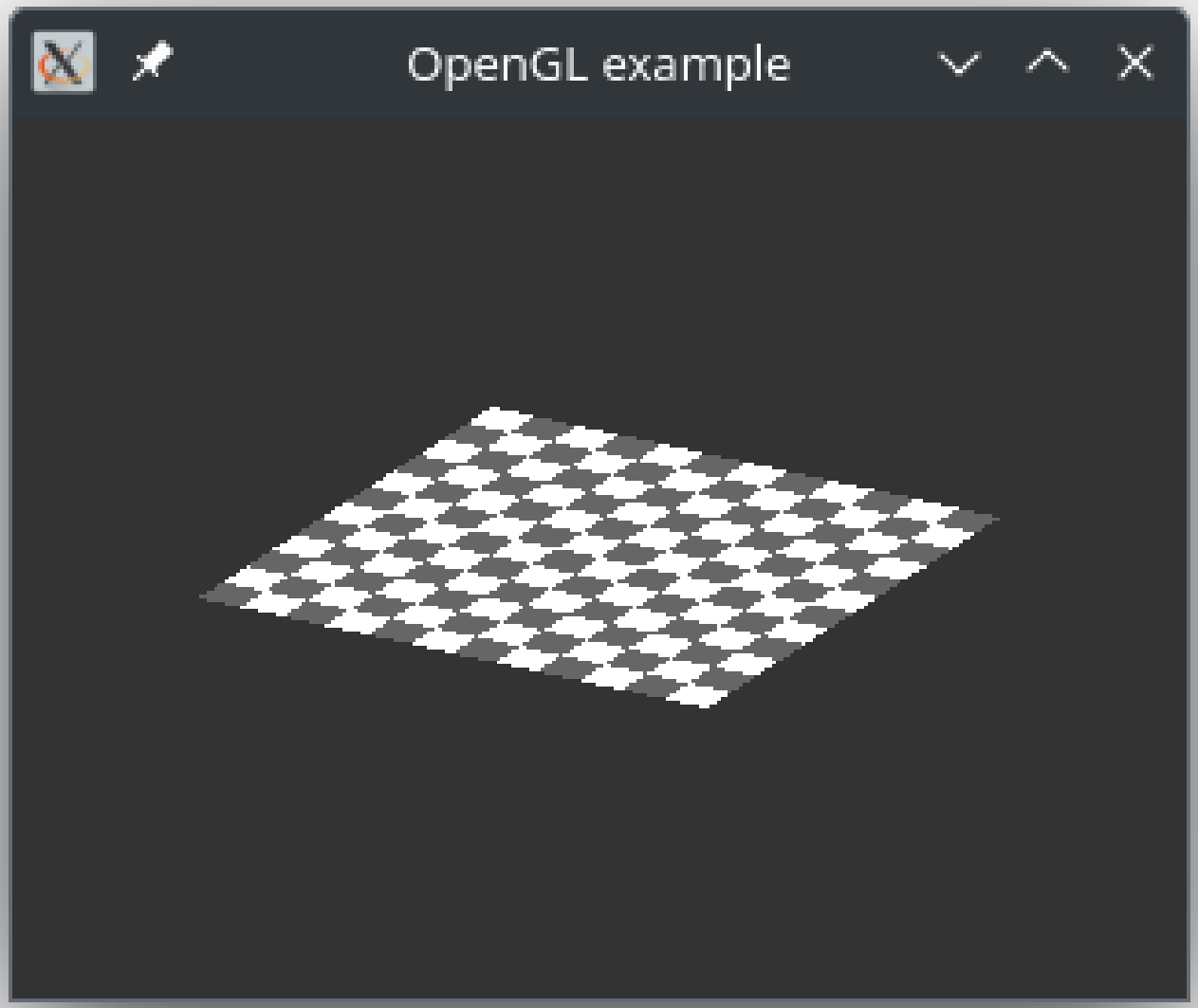
3D: Rotations

```
#version 410 core
uniform mat3 rotz;
uniform mat3 rotx;
in vec3 point;
in vec2 texcoord;
out vec2 UV;
void main()
{
  vec3 pos = rotx * rotz * point;
  gl_Position = vec4(pos, 1);
  UV = texcoord;
}
```

3D: Uniform Rotation Matrices

```
#include <math.h>
// ...
  float alpha = 30 * M_PI / 180;
  float ca = cos(alpha);
  float sa = sin(alpha);
  float rotz[9] = {ca, sa, 0, -sa, ca, 0, 0, 0, 1};
  glUniformMatrix3fv(glGetUniformLocation(program, "rotz"),
                     1, GL_TRUE, rotz);

  float beta = 60 * M_PI / 180;
  float cb = cos(beta);
  float sb = sin(beta);
  float rotx[9] = {1, 0, 0, 0, cb, sb, 0, -sb, cb};
  glUniformMatrix3fv(glGetUniformLocation(program, "rotx"),
                     1, GL_TRUE, rotx);
  // ...
```

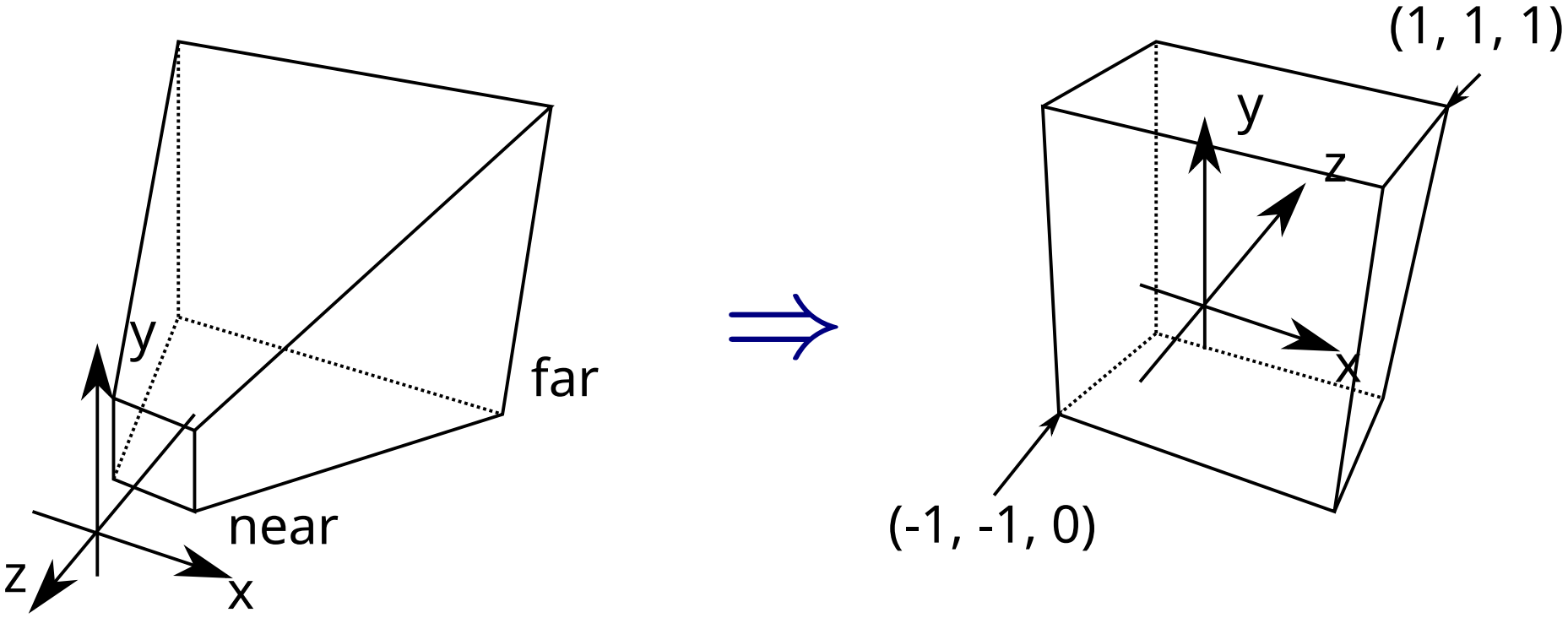
3D: Rotated Quad



3D: Enable Depth Testing

```
glDepthFunc(GL_GEQUAL);
glClipControl(GL_LOWER_LEFT, GL_ZERO_TO_ONE);
glClearDepth(0.0);
glEnable(GL_DEPTH_TEST);
// ...
while (!glfwWindowShouldClose(window)) {
  // ...
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  // ...
```

3D: Projection Matrix



$$\mathcal{P} = \begin{pmatrix} dx & 0 & 0 & 0 \\ 0 & dy & 0 & 0 \\ 0 & 0 & b & a \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

where $dx = \frac{1}{\tan(\frac{1}{2}fov)}$, $dy = \frac{width}{height}dx$, $a = \frac{far \cdot near}{far - near}$, $b = \frac{near}{far - near}$

https://www.wedesoft.de/software/2021/09/20/reversed-z-rendering/

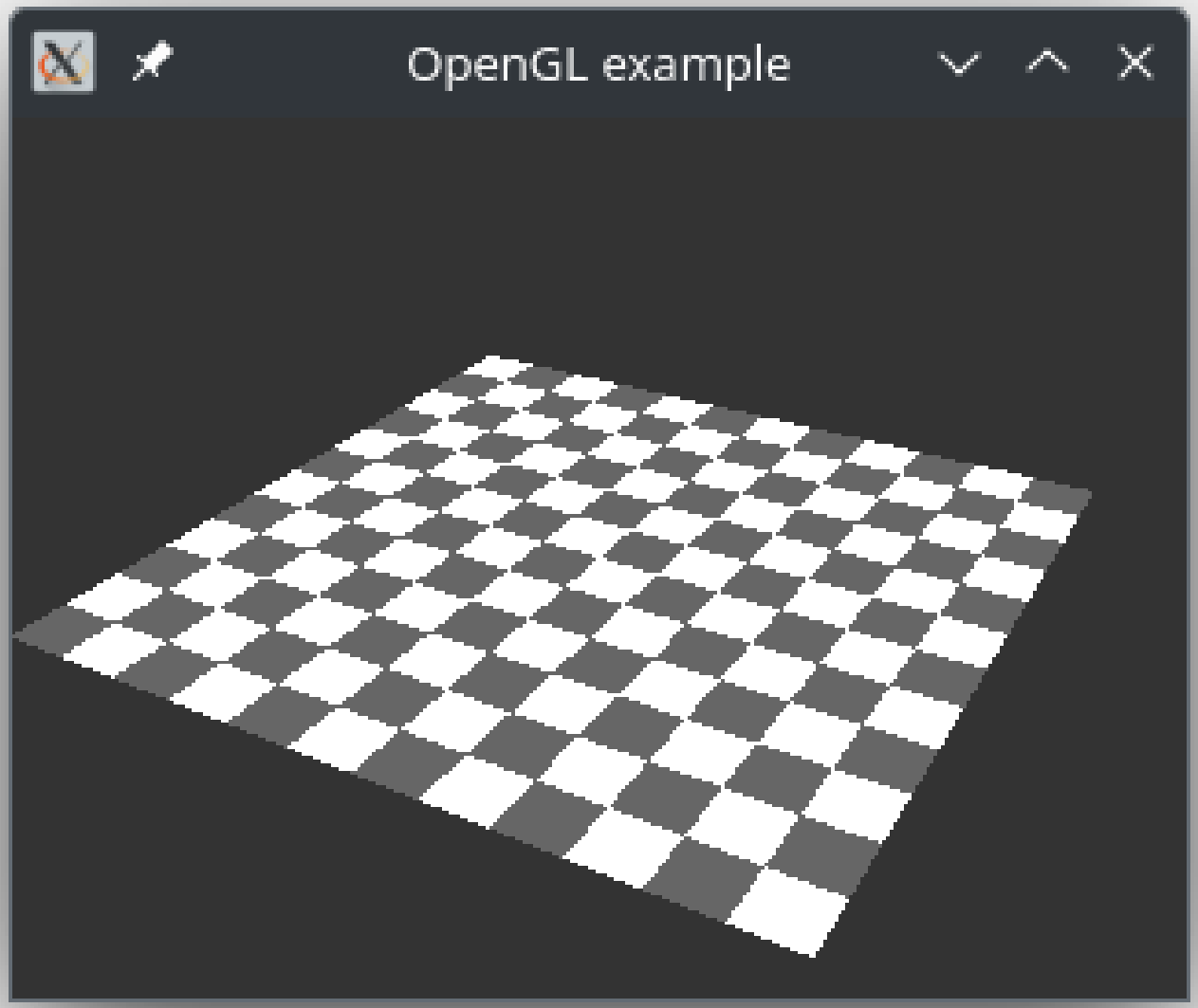
3D: Shader with Translation and Projection

```
#version 410 core
uniform mat3 rotz;
uniform mat3 rotx;
uniform mat4 projection;
uniform float distance;
in vec3 point;
in vec2 texcoord;
out vec2 UV;
void main()
{
  vec3 translation = vec3(0, 0, -distance);
  vec3 pos = rotx * rotz * point + translation;
  gl_Position = projection * vec4(pos, 1);
  UV = texcoord;
}
```

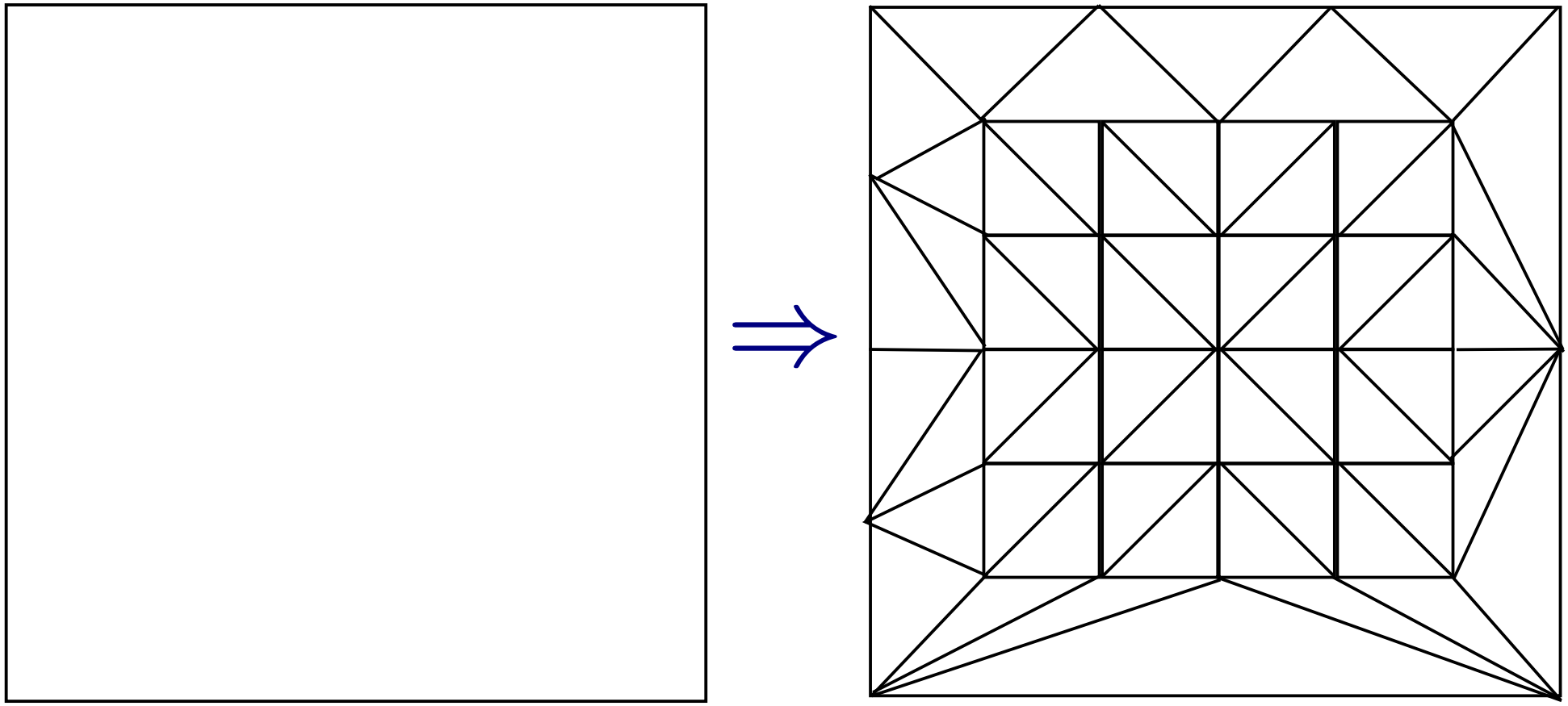
3D: Uniform Distance and Projection Matrix

```
// ...
glUniform1f(glGetUniformLocation(program, "distance"), 1.8);

float fov = 45.0 * M_PI / 180;
float near = 0.1;
float far = 10.0;
float dx = 1.0 / tan(0.5 * fov);
float dy = dx * width / height;
float a = far * near / (far - near);
float b = near / (far - near);
float projection[16] = {dx, 0, 0, 0, 0, dy, 0, 0,
                        0, 0, b, a, 0, 0, -1, 0};
glUniformMatrix4fv(glGetUniformLocation(program, "projection"),
                   1, GL_TRUE, projection);
// ...
```

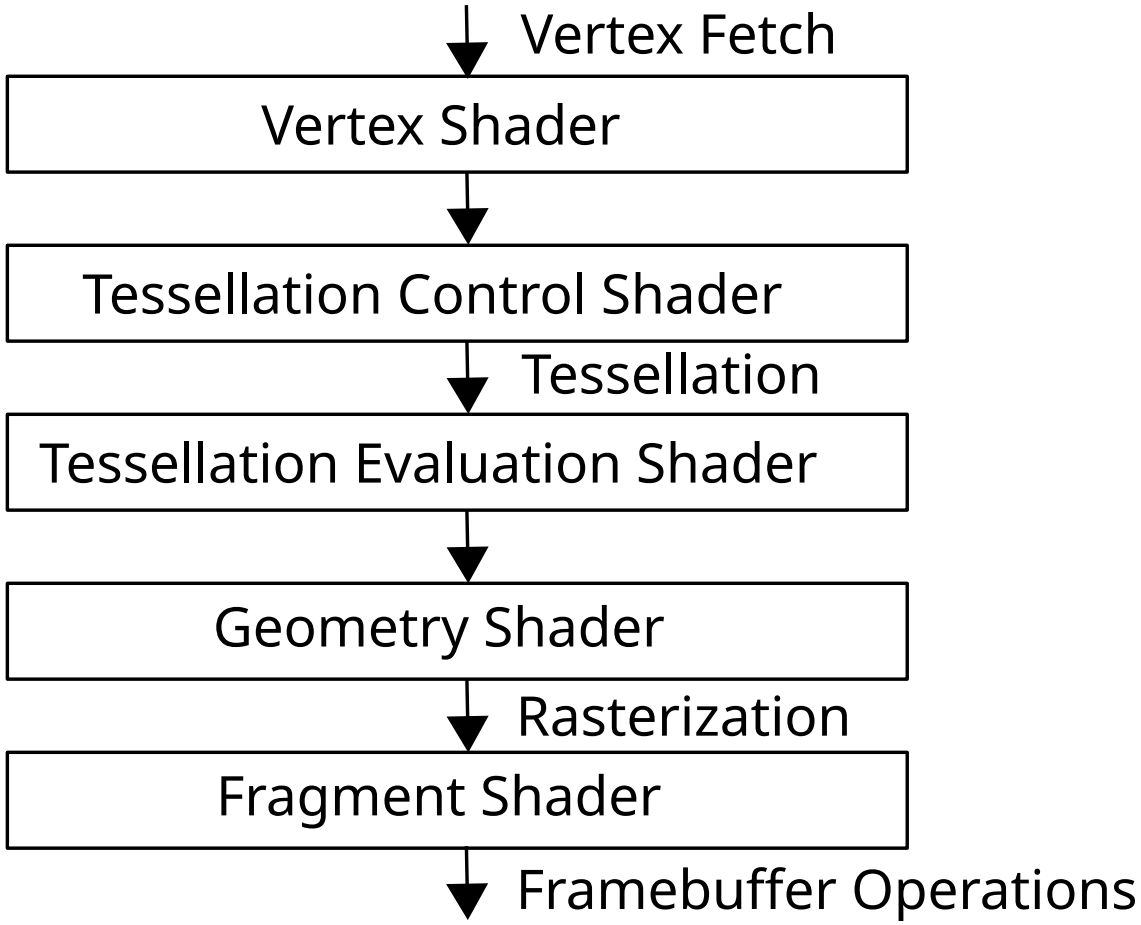
3D: Projected Quad



Tessellation: Quad



Tessellation: Full Pipeline



Tessellation: Vertex Shader

```
#version 410 core
in vec3 point;
in vec2 texcoord;
out vec2 uv_vert;
void main()
{
  gl_Position = vec4(point, 1);
  uv_vert = texcoord;
}
```

Tessellation: Tessellation Control Shader

```
#version 410 core
layout(vertices = 4) out;
in vec2 uv_vert[];
out vec2 uv_contr[];
void main()
{
  if (gl_InvocationID == 0) {
    gl_TessLevelOuter[0] = 25;
    gl_TessLevelOuter[1] = 25;
    gl_TessLevelOuter[2] = 25;
    gl_TessLevelOuter[3] = 25;
    gl_TessLevelInner[0] = 25;
    gl_TessLevelInner[1] = 25;
  };
  gl_out[gl_InvocationID].gl_Position =
    gl_in[gl_InvocationID].gl_Position;
  uv_contr[gl_InvocationID] = uv_vert[gl_InvocationID];
}
```

Tessellation: Tessellation Evaluation Shader

```
#version 410 core
layout(quads, equal_spacing, ccw) in;
uniform mat3 rotz;
uniform mat3 rotx;
uniform mat4 projection;
uniform float distance;
in vec2 uv_contr[];
out vec2 uv_eval;
float amplitude = 0.4;
float scale = 30;
float sinc(float x)
{
  return x > 0 ? sin(x) / x : 1.0;
}
float f(vec2 v)
{
  return amplitude * sinc(scale * length(v));
}
```

Tessellation: Tessellation Evaluation Shader

```
// ...
void main()
{
  vec4 pos = mix(mix(gl_in[0].gl_Position, gl_in[1].gl_Position,
                     gl_TessCoord.x),
                 mix(gl_in[3].gl_Position, gl_in[2].gl_Position,
                     gl_TessCoord.x),
                 gl_TessCoord.y);
  pos.z = f(pos.xy);
  vec3 translation = vec3(0, 0, -distance);
  gl_Position = projection *
                vec4(rotx * rotz * pos.xyz + translation, 1);
  uv_eval = mix(mix(uv_contr[0], uv_contr[1], gl_TessCoord.x),
                mix(uv_contr[3], uv_contr[2], gl_TessCoord.x),
                gl_TessCoord.y);
}
```

Tessellation: Geometry Shader

```
#version 410 core
layout(triangles) in;
in vec2 uv_eval[3];
layout(triangle_strip, max_vertices = 3) out;
out vec2 UV;
void main(void)
{
  gl_Position = gl_in[0].gl_Position;
  UV = uv_eval[0];
  EmitVertex();
  gl_Position = gl_in[1].gl_Position;
  UV = uv_eval[1];
  EmitVertex();
  gl_Position = gl_in[2].gl_Position;
  UV = uv_eval[2];
  EmitVertex();
  EndPrimitive();
}
```

Tessellation: Fragment Shader

```
#version 410 core
uniform sampler2D tex;
in vec2 UV;
out vec3 fragColor;
void main()
{
  fragColor = texture(tex, UV).rgb;
}
```

Tessellation: Compile & Link Shaders

```
// ...
GLuint tessControlShader = glCreateShader(GL_TESS_CONTROL_SHADER);
glShaderSource(tessControlShader, 1, &tessControlSource, NULL);
glCompileShader(tessControlShader);
handleCompileError("Tess. Control shader", tessControlShader);

GLuint tessEvalShader = glCreateShader(GL_TESS_EVALUATION_SHADER);
glShaderSource(tessEvalShader, 1, &tessEvalSource, NULL);
glCompileShader(tessEvalShader);
handleCompileError("Tess. Evaluation shader", tessEvalShader);

GLuint geometryShader = glCreateShader(GL_GEOMETRY_SHADER);
glShaderSource(geometryShader, 1, &geometrySource, NULL);
glCompileShader(geometryShader);
handleCompileError("Geometry shader", geometryShader);
// ...
```

Tessellation: Compile & Link Shaders

```
// ...
GLuint program = glCreateProgram();
glAttachShader(program, vertexShader);
glAttachShader(program, tessControlShader);
glAttachShader(program, tessEvalShader);
glAttachShader(program, geometryShader);
glAttachShader(program, fragmentShader);
glLinkProgram(program);
handleLinkError("Shader program", program);
// ...
```

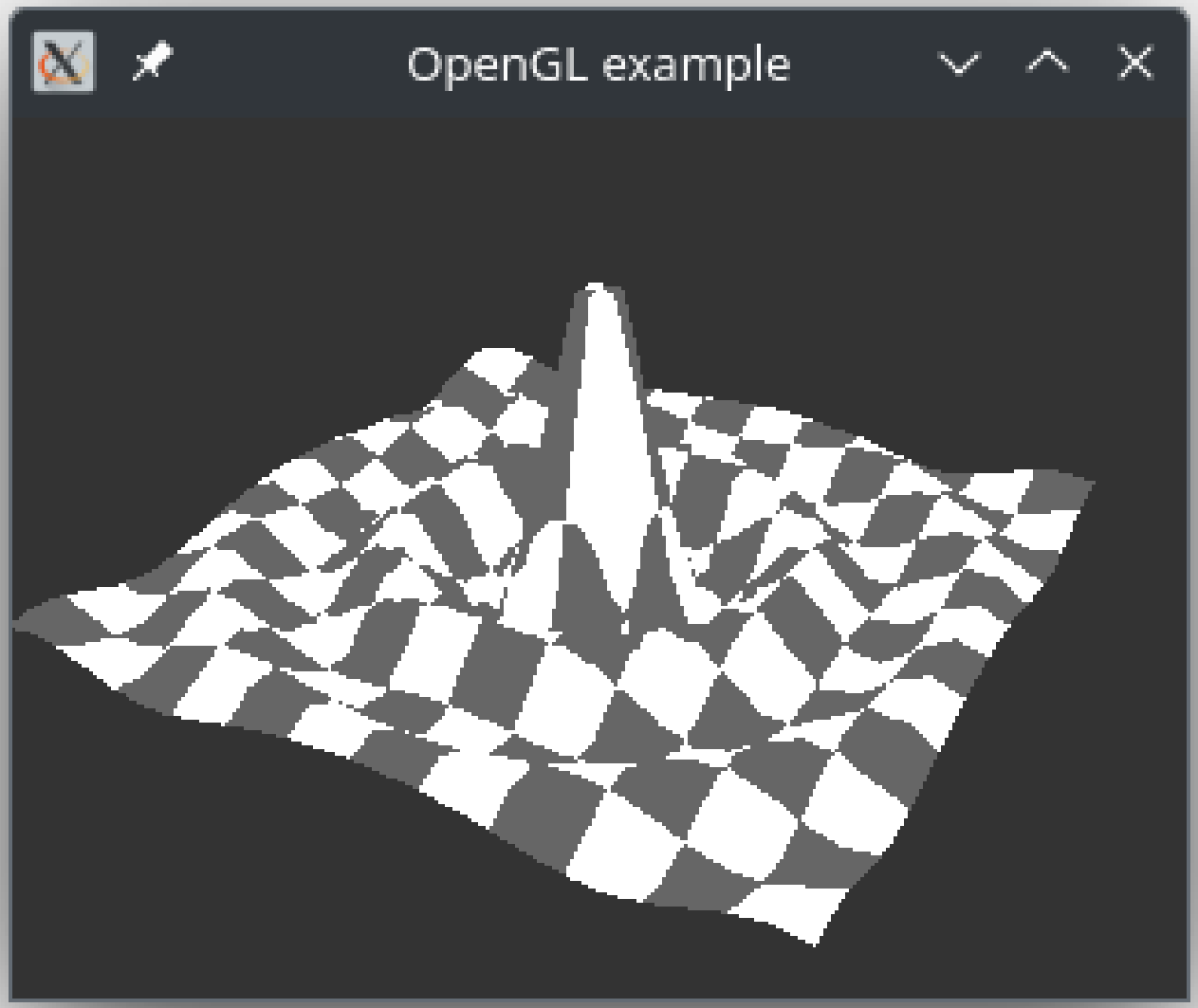
Tessellation: Render Patches

```
// ...
while (!glfwWindowShouldClose(window)) {
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glPatchParameteri(GL_PATCH_VERTICES, 4);
  glDrawElements(GL_PATCHES, 4, GL_UNSIGNED_INT, (void *)0);
  glfwSwapBuffers(window);
  glfwPollEvents();
};
// ...
```

Tessellation: Cleanup

```
// ...
glDeleteProgram(program);
glDeleteShader(vertexShader);
glDeleteShader(tessControlShader);
glDeleteShader(tessEvalShader);
glDeleteShader(geometryShader);
glDeleteShader(fragmentShader);
// ...
```

Tessellation: Result



Diffuse Lighting: Tessellation Evaluation Shader

```
out vec3 normal_eval;
// ...
vec2 fdv(vec2 v)
{
  float l = length(v);
  if (l > 0) {
    float radial = (cos(scale * l) / (l * l) -
                    sin(scale * l) / (scale * (l * l * l)));
    return amplitude * v * radial;
  } else
    return vec2(0, 0);
}
void main()
{
  // ...
  normal_eval = rotx * rotz * vec3(-fdv(pos.xy), 1);
  // ...
}
```

Diffuse Lighting: Geometry Shader

```
// ...
in vec3 normal_eval[3];
// ...
out vec3 normal;
void main(void)
{
  // ...
  normal = normal_eval[0];
  EmitVertex();
  // ...
  normal = normal_eval[1];
  EmitVertex();
  // ...
  normal = normal_eval[2];
  EmitVertex();
  EndPrimitive();
}
```

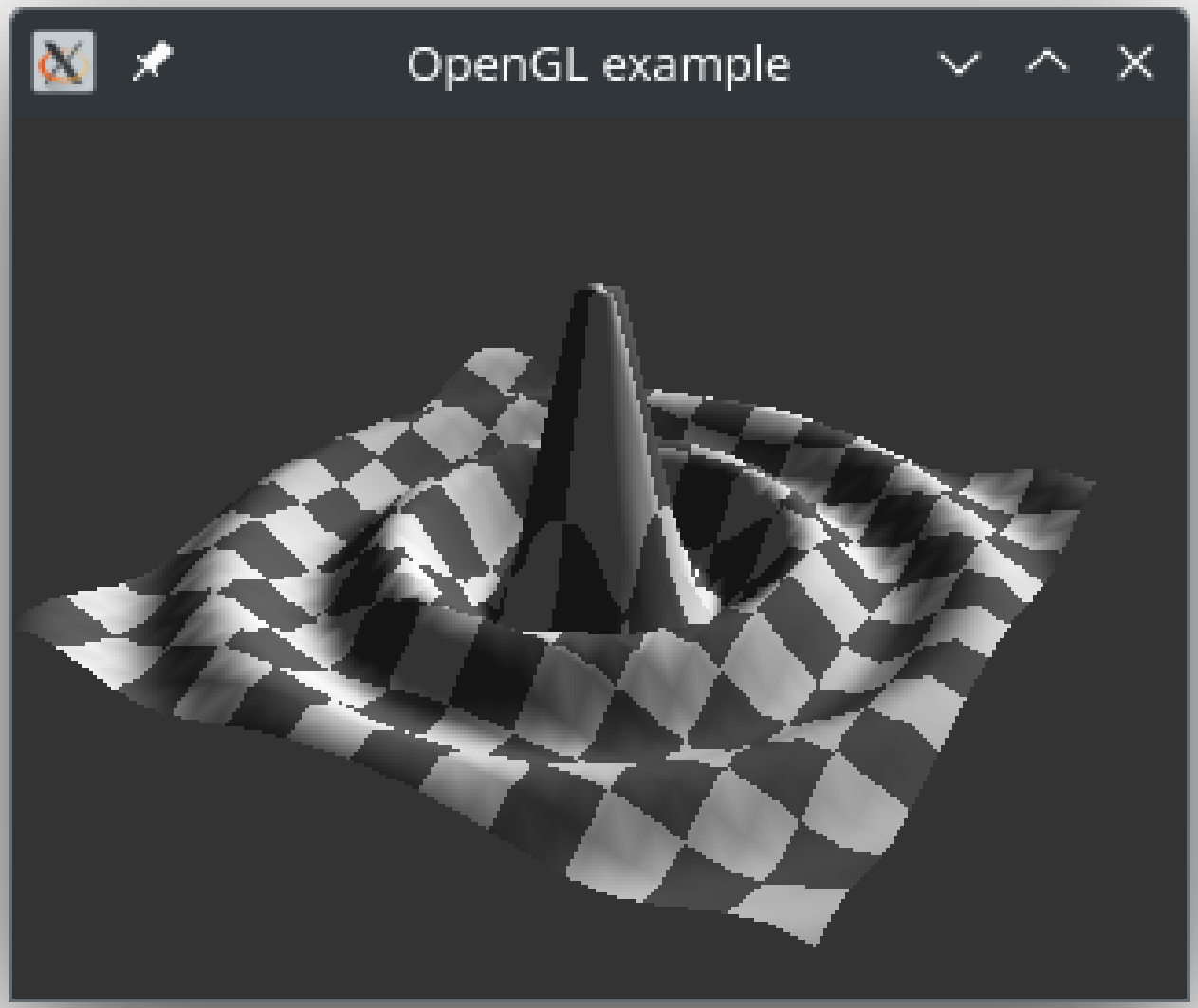
Diffuse Lighting: Fragment Shader

```
#version 410 core
uniform sampler2D tex;
uniform vec3 light;
in vec2 UV;
in vec3 normal;
out vec3 fragColor;
void main()
{
  vec3 n = normalize(normal);
  float ambient = 0.2;
  float diffuse = 0.8 * max(dot(light, n), 0);
  fragColor = (ambient + diffuse) * texture(tex, UV).rgb;
}
```

Diffuse Lightning: Initialise Light Vector

```
// ...
float light[3] = {sqrt(0.5), sqrt(0.5), 0.0};
glUniform3fv(glGetUniformLocation(program, "light"), 1, light);
// ...
```

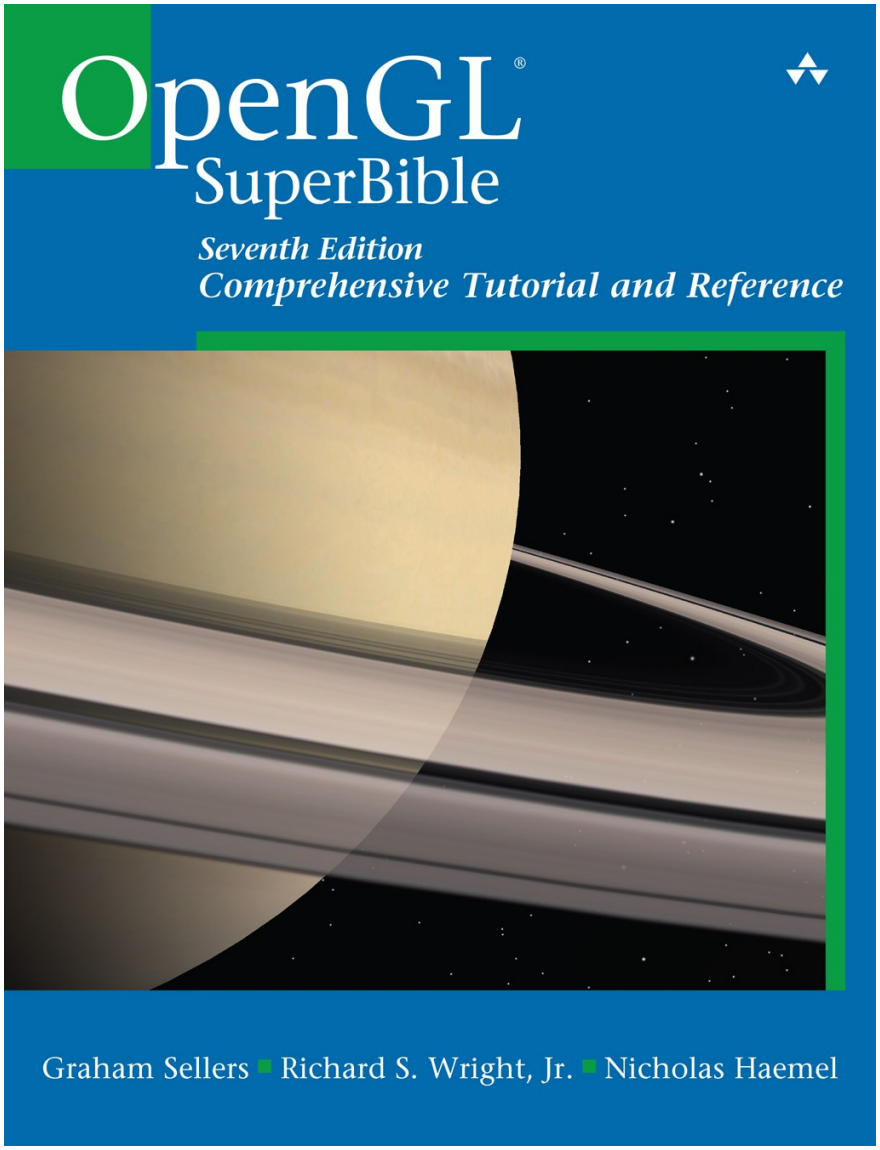
Diffuse Lighting: Result



Further Topics

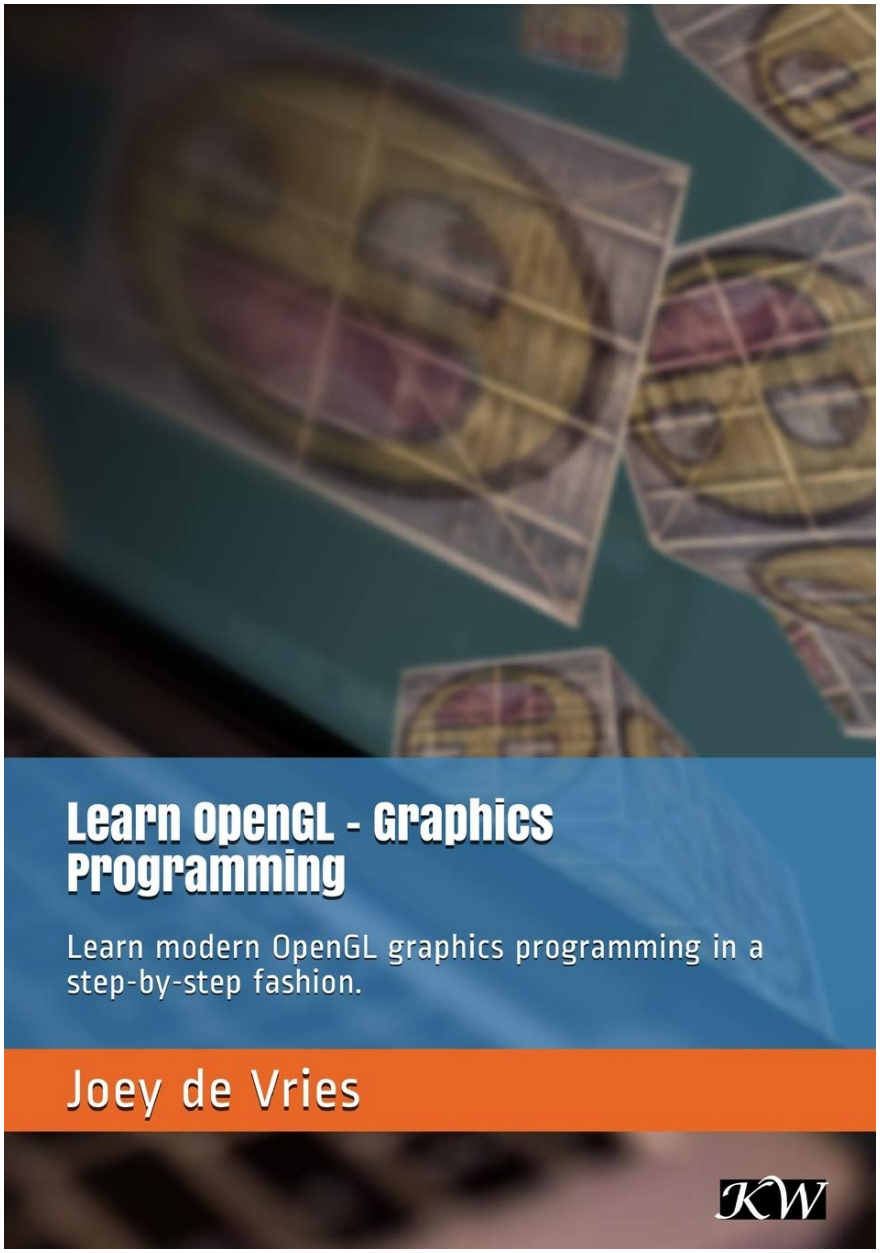
- Face culling
- Phong shading
- Normal maps
- Fog
- Shadow mapping
- Volumetric rendering
- Physically Based Rendering (PBR)
- glTF asset import (e.g. using Assimp)
- Approaching Zero Driver Overhead (AZDO) features
- Compute Shaders

References: OpenGL Superbible



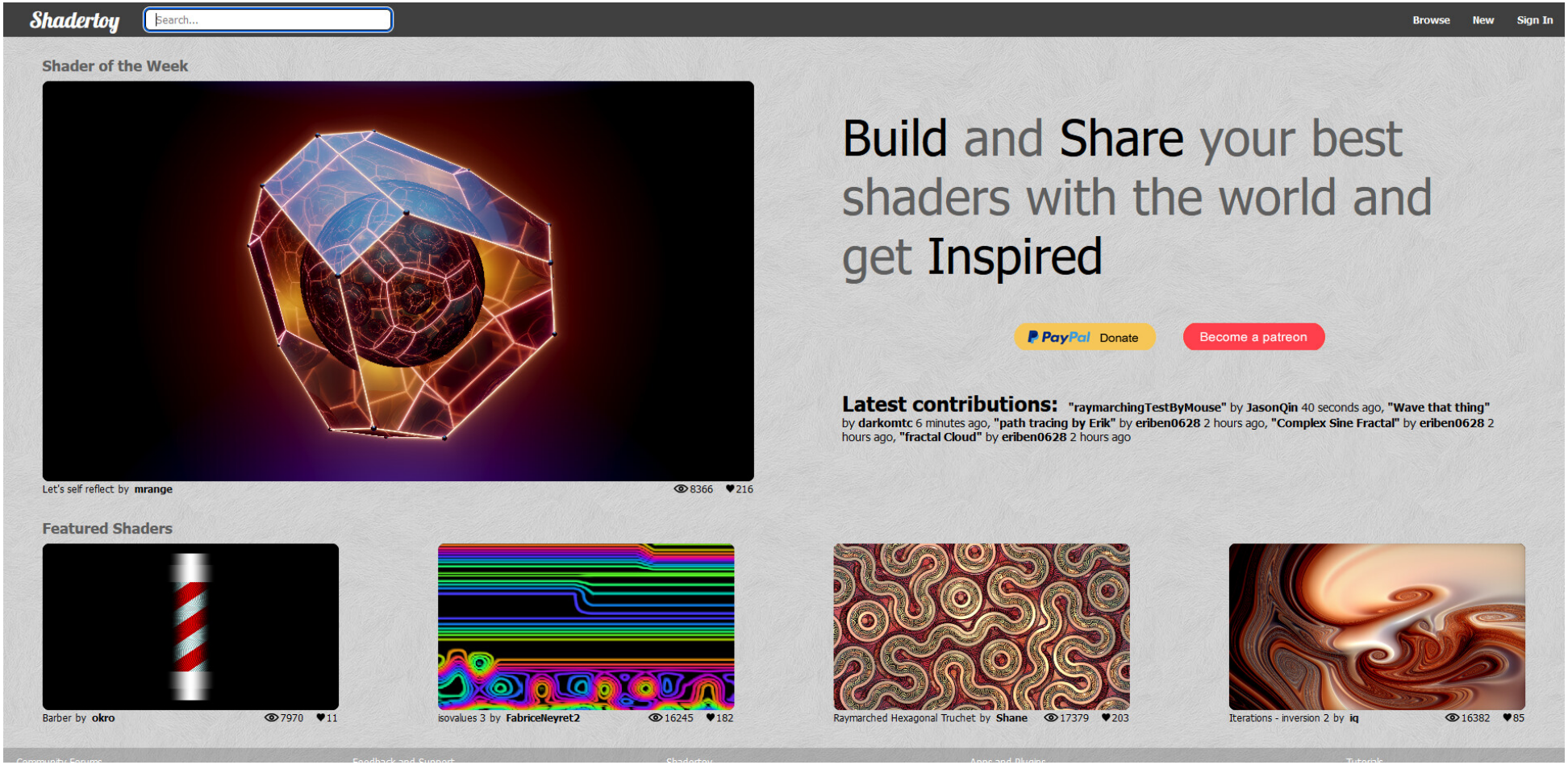
https://www.informit.com/store/opengl-superbible-comprehensive-tutorial-and-reference-9780134193137

References: Learn OpenGL



https://learnopengl.com/

References: Shadertoy



https://www.shadertoy.com/

Questions?