Open Graphics Library

- Application Programming Interface (API) for rendering graphics

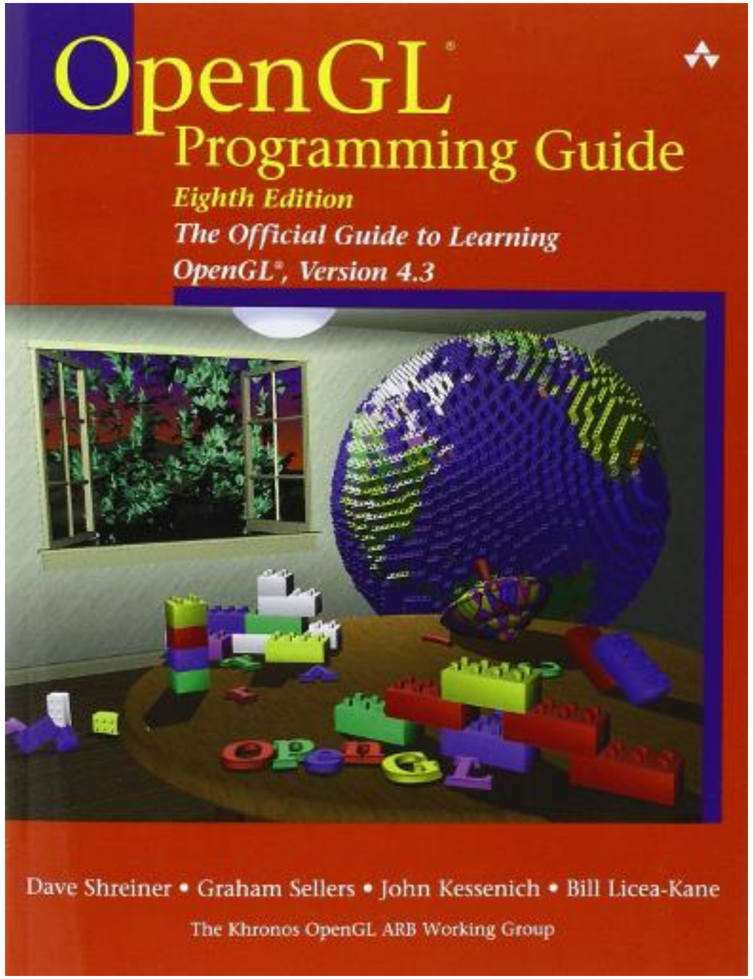
- Hardware acceleration through the GPU

- Language Independent

WebGL™ PyOpenGL The Python OpenGL® Binding JOGL Perl OpenGL

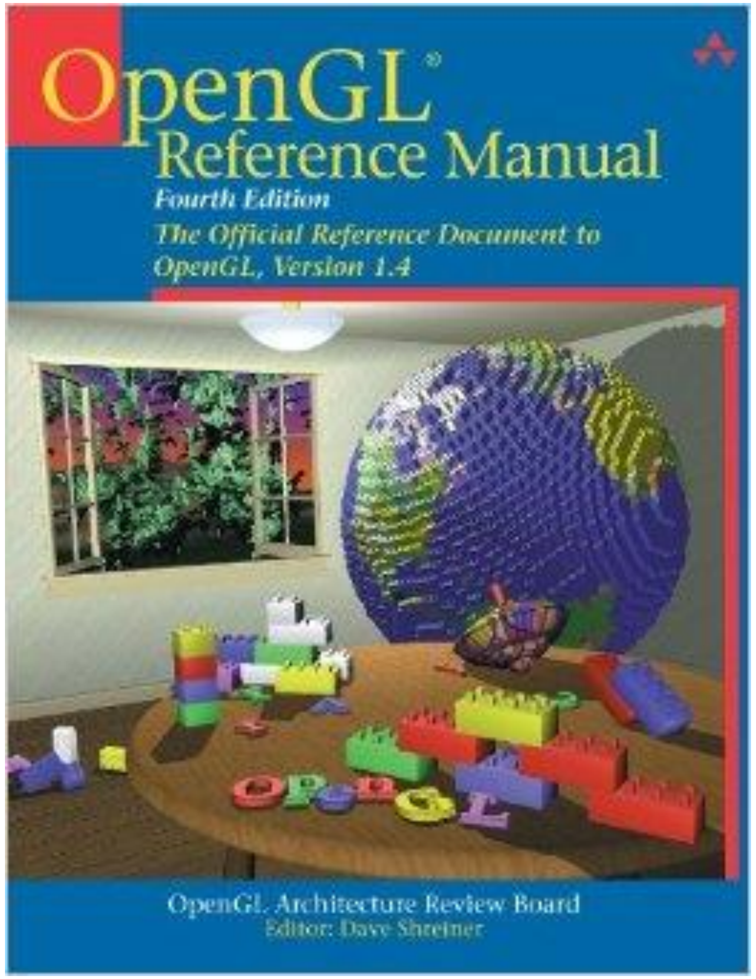
- Platform Independent

Windows® 7 Linux Mac OS X android iOS



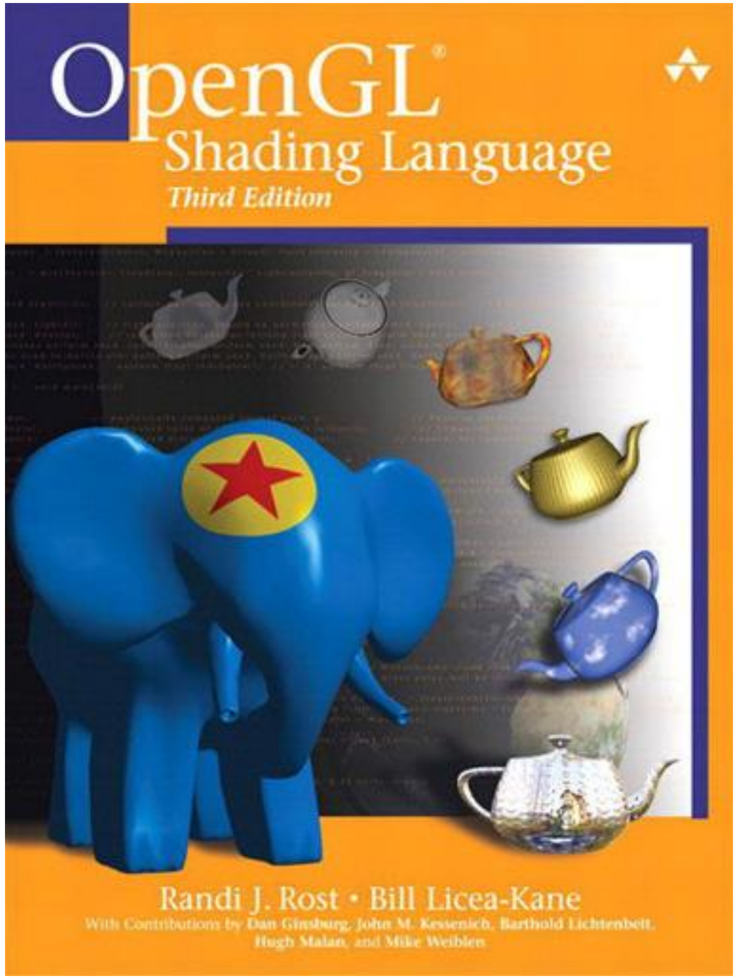
OpenGL®
Programming Guide
Eighth Edition
The Official Guide to Learning OpenGL®, Version 4.3

Dave Shreiner • Graham Sellers • John Kessenich • Bill Licea-Kane
The Khronos OpenGL ARB Working Group



OpenGL®
Reference Manual
Fourth Edition
The Official Reference Document to OpenGL, Version 1.4

OpenGL Architecture Review Board
Editor: Dave Shreiner



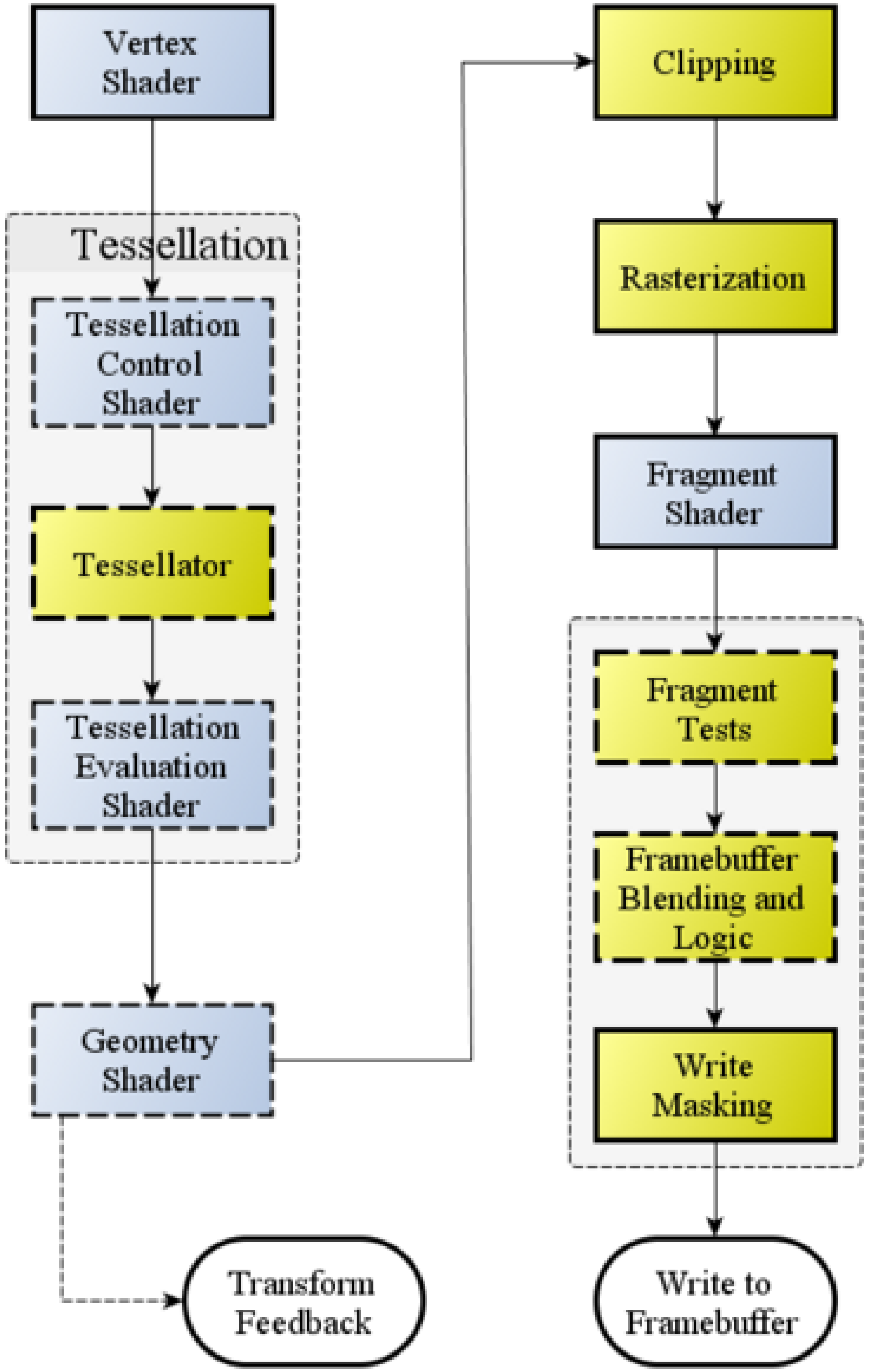
OpenGL®
Shading Language
Third Edition

Randi J. Rost • Bill Licea-Kane
With Contributions by Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen

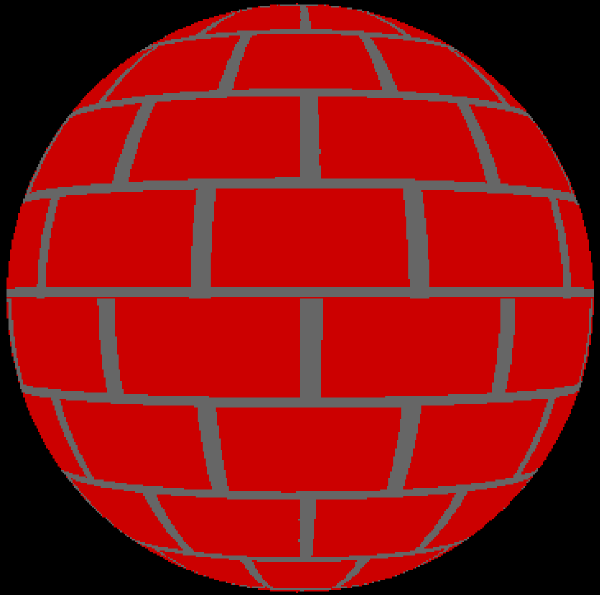
Versions

GLSL Version	OpenGL Version	Date	Shader Preprocessor
1.10.59	2.0	April 2004	#version 110
1.20.8	**2.1**	**September 2006**	**#version 120**
1.30.10	3.0	August 2008	#version 130
1.40.08	**3.1**	**March 2009**	**#version 140**
1.50.11	3.2	August 2009	#version 150
3.30.6	3.3	February 2010	#version 330
4.00.9	4.0	March 2010	#version 400
4.10.6	4.1	July 2010	#version 410
4.20.11	4.2	August 2011	#version 420
4.30.8	4.3	August 2012	#version 430
4.40	4.4	July 2013	#version 440
4.50	4.5	August 2014	#version 450

https://en.wikipedia.org/wiki/OpenGL_Shading_Language



Vertex Shader
Tessellation
Tessellation Control Shader
Tessellator
Tessellation Evaluation Shader
Geometry Shader
Transform Feedback
Clipping
Rasterization
Fragment Shader
Fragment Tests
Framebuffer Blending and Logic
Write Masking
Write to Framebuffer



OpenGL 2.1 and GLSL 1.2

```
vector<vec3f> vertices;
vector<vec3f> normals;
vector<vec2f> texcoords;
vector<unsigned int> indices;

GLuint program;
```

Vertex Shader

- Run independently on each vertex
- Can’t pass data between vertices
- Pass interpolated data to fragment shaders

```
#version 120

varying vec2 brick_coord;

void main()
{
	brick_coord = gl_MultiTexCoord0.st;
	gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

Fragment Shader

- Run independently on each pixel
- Can’t pass data between pixels

```
#version 120

uniform float brick_size;
uniform vec3 brick_color;
uniform vec3 mortar_color;
uniform vec2 brick_pct;

varying vec2 brick_coord;

void main()
{
        vec2 interp = brick_coord/brick_size;
        if (fract(interp.y * 0.5) > 0.5)
                interp.x += 0.5;

        interp = step(fract(interp), brick_pct);

        gl_FragColor = vec4(mix(mortar_color, brick_color, interp.x*interp.y), 1.0);
}
```

Loading the Shaders

```
// Load the shaders
GLuint vertex = load_shader_file("res/example.vx", GL_VERTEX_SHADER);
GLuint fragment = load_shader_file("res/example.ft", GL_FRAGMENT_SHADER);

program = glCreateProgram();
glAttachShader(program, vertex);
glAttachShader(program, fragment);
glLinkProgram(program);
```

Rendering Geometry

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);

glVertexPointer(3, GL_FLOAT, sizeof(GLfloat)*3, vertices.data());
glNormalPointer(GL_FLOAT, sizeof(GLfloat)*3, normals.data());
glTexCoordPointer(2, GL_FLOAT, sizeof(GLfloat)*2, texcoords.data());

// Draw the triangles
glDrawElements(GL_TRIANGLES, (int)indices.size(), GL_UNSIGNED_INT, indices.data());

// Clean up
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);

glUseProgram(0);
```

Transformations

```
glLoadIdentity();
glTranslatef(0.0, 0.0, -5.0);
glRotatef(radtodeg(angle[0]), 1.0, 0.0, 0.0);
glRotatef(radtodeg(angle[1]), 0.0, 1.0, 0.0);
glRotatef(radtodeg(angle[2]), 0.0, 0.0, 1.0);
glScalef(scalex, scaley, scalez);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.6, 1.6, -1.0, 1.0, 2.0, 100.0);
glMatrixMode(GL_MODELVIEW);
```

OpenGL 3.1 and GLSL 1.4

```
vector<vec3f> vertices;
vector<vec3f> normals;
vector<vec2f> texcoords;
vector<unsigned int> indices;

mat4f modelview_matrix = identity<float, 4, 4>();
mat4f projection_matrix = identity<float, 4, 4>();
mat3f normal_matrix = identity<float, 3, 3>();

GLuint program;
```

Vertex Shader

```
#version 140

in vec3 vertex;
in vec3 normal;
in vec2 texcoord;

uniform mat4 modelview_projection_matrix;
uniform mat3 normal_matrix;

out vec2 brick_coord;

void main()
{
        brick_coord = texcoord;
        gl_Position = modelview_projection_matrix*vec4(vertex, 1.0);
}
```

Fragment Shader

```
#version 140

uniform float brick_size;
uniform vec3 brick_color;
uniform vec3 mortar_color;
uniform vec2 brick_pct;

in vec2 brick_coord;
out vec4 frag_color;

void main()
{
        vec2 interp = brick_coord/brick_size;
        if (fract(interp.y * 0.5) > 0.5)
                interp.x += 0.5;

        interp = step(fract(interp), brick_pct);

        frag_color = vec4(mix(mortar_color, brick_color, interp.x*interp.y), 1.0);
}
```

Loading the Shaders

```
// Load the shaders
GLuint vertex = load_shader_file("res/example.vx", GL_VERTEX_SHADER);
GLuint fragment = load_shader_file("res/example.ft", GL_FRAGMENT_SHADER);

program = glCreateProgram();
glAttachShader(program, vertex);
glAttachShader(program, fragment);
glBindFragDataLocation(program, 0, "frag_color");
glLinkProgram(program);
```

Rendering Geometry

```
// specify the shader to use
glUseProgram(program);

// Find the locations of the vertex, normal, and texcoord variables in the shader
int vertex_location        = glGetAttribLocation(program, "vertex");
int normal_location        = glGetAttribLocation(program, "normal");
int texcoord_location      = glGetAttribLocation(program, "texcoord");

int mvp_matrix_location    = glGetUniformLocation(program, "modelview_projection_matrix");
int normal_matrix_location = glGetUniformLocation(program, "normal_matrix");

int brick_size_location    = glGetUniformLocation(program, "brick_size");
int brick_color_location   = glGetUniformLocation(program, "brick_color");
int mortar_color_location  = glGetUniformLocation(program, "mortar_color");
int brick_pct_location     = glGetUniformLocation(program, "brick_pct");

// Pass in the necessary transformation matrices
mat4f mpv_matrix = projection_matrix*modelview_matrix;
glUniformMatrix4fv(mvp_matrix_location,    1, true, (GLfloat*)&mpv_matrix);
glUniformMatrix3fv(normal_matrix_location, 1, true, (GLfloat*)&normal_matrix);

// Pass in the parameters for the brick shader
glUniform1f(brick_size_location,   0.1);
glUniform3f(brick_color_location,  0.8, 0.0, 0.0);
glUniform3f(mortar_color_location, 0.4, 0.4, 0.4);
glUniform2f(brick_pct_location,    0.9, 0.9);
```

```
glEnableVertexAttribArray(vertex_location);
glEnableVertexAttribArray(normal_location);
glEnableVertexAttribArray(texcoord_location);

glVertexAttribPointer(vertex_location,   3, GL_FLOAT, false, sizeof(GLfloat)*3,
vertices.data());
glVertexAttribPointer(normal_location,   3, GL_FLOAT, false, sizeof(GLfloat)*3,
normals.data());
glVertexAttribPointer(texcoord_location, 2, GL_FLOAT, false, sizeof(GLfloat)*2,
texcoords.data());

// Draw the triangles
glDrawElements(GL_TRIANGLES, (int)indices.size(), GL_UNSIGNED_INT, indices.data());

// Clean up
glDisableVertexAttribArray(vertex_location);
glDisableVertexAttribArray(normal_location);
glDisableVertexAttribArray(texcoord_location);

glUseProgram(0);
```