



Lahav Lipson, Yuxuan Mei, Crystal Ren

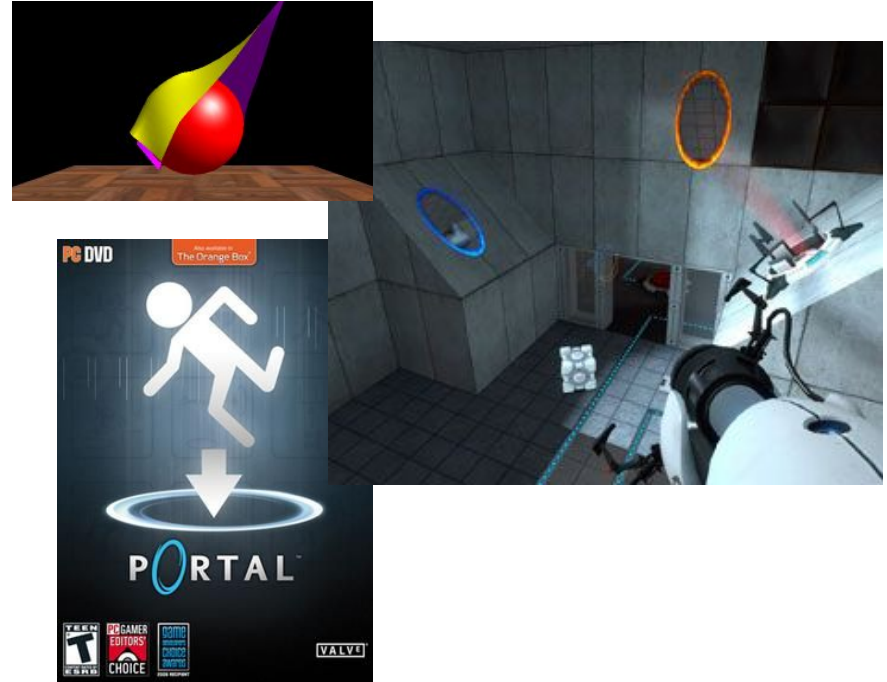
Project Link: <https://github.com/Lahav174/SimpleGL>

# Abstract

OpenGL is complex due to its high level of customizability. However, the majority of users and use-cases do not require extensive customizability yet are still burdened with OpenGL's complexity. Our goal is to eliminate the extensive boilerplate code and streamline and/or automate a lot of setup and teardown that have been thus far left to the user. SimpleGL makes it simple for beginners to get started with OpenGL.

# Graphics Programming and OpenGL

- Graphics Programming:
  - animations
  - visualizations of physics simulations
  - video games
  - modeling
- OpenGL: industry standard for graphics specification (2D and 3D scenes)



[https://en.wikipedia.org/wiki/Portal\\_\(video\\_game\)](https://en.wikipedia.org/wiki/Portal_(video_game))  
<http://www.paulsprojects.net/opengl/cloth/cloth.html>

# OpenGL Problems

- Difficult to learn and use due to extensive customizability and functionality

- Lots of boilerplate and repeat code for most users
- Hard for beginners
- Non-intuitive relation between code and end render result: ie. OpenGL uses vertex arrays to define objects

```
// glfw: initialize and configure
glfwInit();
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

// glfw window creation
GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "OpenGL Window", NULL, NULL);
if (window == NULL)
{
    std::cout << "Failed to create GLFW window\n";
    glfwTerminate();
    return -1;
}
glfwMakeContextCurrent(window);
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
glfwSetCursorPosCallback(window, mouse_callback);
glfwSetScrollCallback(window, scroll_callback);

// tell GLFW to capture input
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);

// glad: load all OpenGL extensions
if (!gladLoadGLLoader((GLAD_CALLBACK)glfwGetProcAddress))
{
    std::cout << "Failed to load OpenGL extensions\n";
    return -1;
}

// configure global opengl settings
glEnable(GL_DEPTH_TEST);

// glfw: whenever the mouse moves, this callback is called
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to top
    lastX = xpos;
    lastY = ypos;

    float sensitivity = 0.1f; // change this value to your liking
    xoffset *= sensitivity;
    yoffset *= sensitivity;

    yaw += xoffset;
    pitch += yoffset;

    // make sure that when pitch is out of bounds, screen doesn't get flipped
    if (pitch > 89.0f)
        pitch = 89.0f;
    if (pitch < -89.0f)
        pitch = -89.0f;

    glm::vec3 front;
    front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    front.y = sin(glm::radians(pitch));
    front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
    cameraFront = glm::normalize(front);

    float specularStrength = 0.5;

    void main() {
        float ambientStrength = 0.5;
        vec3 ambient = ambientStrength * lightColor;

        vec3 norm = normalize(Normal);
        vec3 lightDir = normalize(lightPos - FragPos);
        float diff = max(dot(norm, lightDir), 0.0);
        vec3 diffuse = diff * lightColor;

        vec3 viewDir = normalize(viewPos - FragPos);
        vec3 reflectDir = reflect(-viewDir, norm);
        float spec = pow(max(dot(viewDir, reflectDir), 0.0), 20);
        vec3 specular = specularStrength * spec * lightColor;

        vec3 result = (ambient + diffuse + specular) * objectColor;
        FragColor = vec4(result, 1.0);
    }

    void main() {
        glm::mat4 projection = projection * view * model * vec4(aPos, 1.0);
        FragPos = vec3(model * vec4(aPos, 1.0));
        Normal = aNormal;
    }
}
```

# SimpleGL

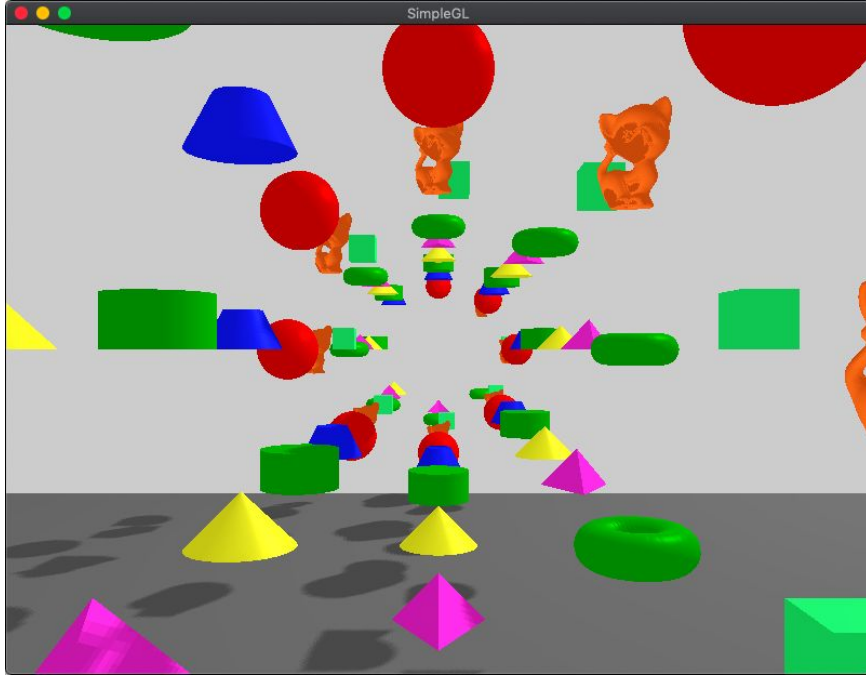
## Pros

- Beginner-friendly
- Eliminates boilerplate code
  - Decreases lines of code
- More intuitive
  - Object-oriented structure mirrors 3D object result
  - Single render call
  - RAll deals with setup and cleanup

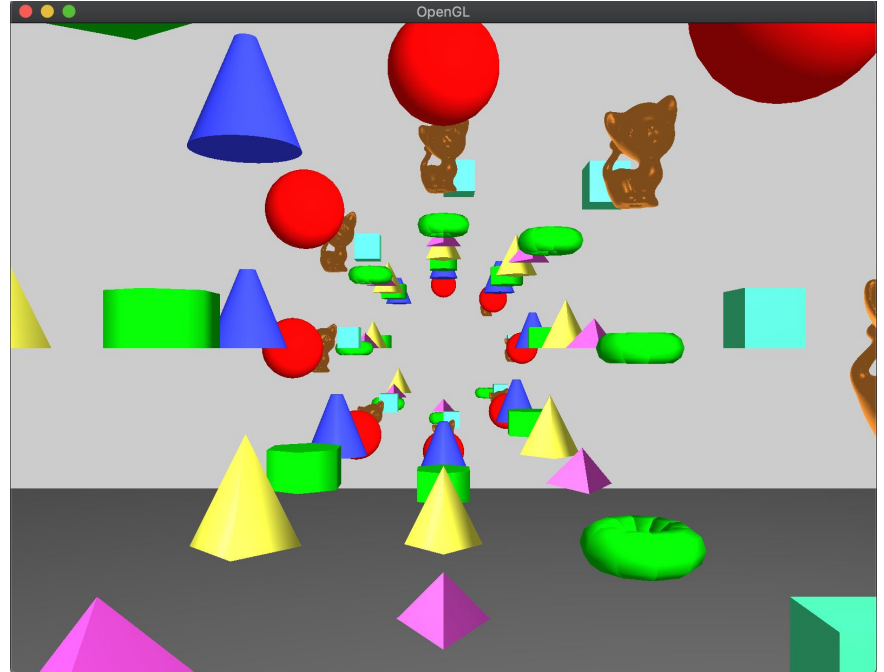
## Cons

- Less flexibility and customizability than OpenGL
- Hides fundamental graphics concepts (ie. vertex array, transformation matrices)

# SimpleGL vs. OpenGL Example



SimpleGL



OpenGL

# Example: Setup/Creating the Scene

- Create a Scene

```
// initialize the scene.  
Scene s(true);  
s.set_callback(print_frame_rate);  
s.set_shadow(true);  
s.set_light_pos({30,30,-30});
```

SimpleGL

- Scene, window, GLFW, user input setup:

```
// glfw initialize and configure  
glfwInit();  
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);  
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);  
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);  
  
#if defined __APPLE__  
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); // uncomment this statement to fix compilation on OS X  
#endif  
  
int width = (float) SCR_WIDTH / 2.0;  
int height = (float) SCR_HEIGHT / 2.0;  
// glfw window creation  
GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "OpenGL", NULL, NULL);  
if (!window) {  
    std::cout << "Failed to create GLFW window" << std::endl;  
    return -1;  
}  
glfwMakeContextCurrent(window);  
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);  
glfwSetCursorPosCallback(window, mouse_callback);  
glfwSetScrollCallback(window, scroll_callback);  
  
// set GLFW to capture our mouse  
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);  
  
// start load all shader function pointers  
if (!glGetShaderSourceARB || !glGetShaderSourceARB || !glGetShaderSourceARB) {  
    std::cout << "Failed to initialize GLAD" << std::endl;  
    return -1;  
}  
  
// configure glad, compile state  
gladLoadGLLoader(&glLoadGL);  
  
// build and compile our shader program  
// 1. retrieve the vertex/fragment source code from filePath  
unsigned int vshID = 0;  
std::string vertexCode;  
std::string fragmentCode;  
std::ifstream vShaderFile;  
std::ifstream fShaderFile;  
  
// check if files exist  
if (!vShaderFile.is_open() || !fShaderFile.is_open()) {  
    std::cout << "ERROR: SHADER FILES NOT SUCCESSFULLY READ" << std::endl;  
    return -1;  
}  
  
// read files  
vShaderFile.open("vshader.glsl");  
fShaderFile.open("fshader.glsl");  
std::stringstream vShaderStream, fShaderStream;  
vShaderStream << vShaderFile.rdbuf();  
fShaderStream << fShaderFile.rdbuf();  
vShaderFile.close();  
fShaderFile.close();  
  
// convert stream into string  
vertexCode = vShaderStream.str();  
fragmentCode = fShaderStream.str();  
  
// create shader objects  
GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);  
glShaderSource(vertexShader, 1, &vertexCode, NULL);  
glCompileShader(vertexShader);  
checkCompileErrors(vertexShader, "VERTEX");  
  
GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);  
glShaderSource(fragmentShader, 1, &fragmentCode, NULL);  
glCompileShader(fragmentShader);  
checkCompileErrors(fragmentShader, "FRAGMENT");  
  
// create program  
GLuint program = glCreateProgram();  
glAttachShader(program, vertexShader);  
glAttachShader(program, fragmentShader);  
glLinkProgram(program);  
checkCompileErrors(program, "PROGRAM");  
  
// delete the shaders as they're linked into our program now and no longer necessary  
glDeleteShader(vertexShader);  
glDeleteShader(fragmentShader);
```

OpenGL

# Example: Adding Objects to the Scene

## ● Add objects:

```
ObjId floor = s.add_obj(Shape::box, {0.6, 0.6, 0.6});
const double radius = 5;
for (int i=0; i<10; i++){
    ObjId sphere = s.add_obj(Shape::sphere, RED);
    ObjId truncated_cone = s.add_obj(Shape::truncatedCone, BLUE);
    ObjId cylinder = s.add_obj(Shape::cylinder, GREEN);
    ObjId cone = s.add_obj(Shape::cone, YELLOW);
    ObjId pyramid = s.add_obj(Shape::pyramid, PINK);
    ObjId torus = s.add_obj(Shape::torus, GREEN, {.r1 = 0.7, .r2 = 0.4, .accuracy = 5});
    ObjId box = s.add_obj(Shape::box, TEAL);
    ObjId kitten = s.add_obj("kitten", ORANGE, {.filepath = "./test/obj_files/kitten.obj"});
}
```

## Add objects:

```
// render cylinder
auto verticesCylinder = glp::cylinder(6, 5, 1);
glBufferData(GL_ARRAY_BUFFER, verticesCylinder.size()*sizeof(double), verticesCylinder.data(), GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesCylinder.data());
glEnableVertexAttribArray(0);

// normal attributes
glVertexAttribPointer(0, 3, GL_DOUBLE, 0, verticesCylinder.data());
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesCylinder.data());
glEnableVertexAttribArray(1);

// render pyramid
auto verticesPyramid = glp::pyramid(4, 1);
glBufferData(GL_ARRAY_BUFFER, verticesPyramid.size()*sizeof(double), verticesPyramid.data(), GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesPyramid.data());
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesPyramid.data());
glEnableVertexAttribArray(1);

// normal attributes
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesPyramid.data());
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesPyramid.data());
glEnableVertexAttribArray(1);

//render obj
std::vector<double> verticesObj;
if (argc > 3) {
    obj::Loader loader;
    for (int i = 0; i < loader.LoadedVertices.size(); i++){
        verticesObj.push_back(loader.LoadedVertices[i].Position.X);
        verticesObj.push_back(loader.LoadedVertices[i].Position.Y);
        verticesObj.push_back(loader.LoadedVertices[i].Position.Z);
        verticesObj.push_back(loader.LoadedVertices[i].Normal.X);
        verticesObj.push_back(loader.LoadedVertices[i].Normal.Y);
        verticesObj.push_back(loader.LoadedVertices[i].Normal.Z);
    }
    auto verticesObjSize = 6 * loader.LoadedVertices.size();
    glBufferData(GL_ARRAY_BUFFER, verticesObjSize*sizeof(double), verticesObj.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesObj.data());
    glEnableVertexAttribArray(0);
    // normal attributes
    glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesObj.data());
    glEnableVertexAttribArray(1);
}

// render cone
auto verticesCone = glp::cone(6, 2, 1);
glBufferData(GL_ARRAY_BUFFER, verticesCone.size()*sizeof(double), verticesCone.data(), GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesCone.data());
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesCone.data());
glEnableVertexAttribArray(1);

// normal attributes
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesCone.data());
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesCone.data());
glEnableVertexAttribArray(1);

// render truncated cone
auto verticesTruncatedCone = glp::truncatedCone(10, 2, 1, 6, 2);
glBufferData(GL_ARRAY_BUFFER, verticesTruncatedCone.size()*sizeof(double), verticesTruncatedCone.data(), GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesTruncatedCone.data());
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesTruncatedCone.data());
glEnableVertexAttribArray(1);

// normal attributes
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 0, verticesTruncatedCone.data());
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 0, verticesTruncatedCone.data());
glEnableVertexAttribArray(1);
}
```

SimpleGL

OpenGL



# Example: Transforming Objects

- Transform objects:

```
double angle = (M_PI/4)*(1+i);
sphere.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
angle = (M_PI/4)*(2+i);
truncated_cone.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
angle = (M_PI/4)*(3+i);
cylinder.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
angle = (M_PI/4)*(4+i);
cone.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
angle = (M_PI/4)*(5+i);
pyramid.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
angle = (M_PI/4)*(6+i);
torus.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
angle = (M_PI/4)*(7+i);
box.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
angle = (M_PI/4)*(8+i);
kitten.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
kitten.scale(2);
}
floor.translate(glm::vec3(-35, -7, -35));
floor.scale({70, 0.01, 70});
```

SimpleGL

- Transform objects:

```
for (int i=0; i<10; i++){
    glBindVertexArray(vao_box);
    // calculate the model matrix for each object and pass it to shader before drawing
    model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    angle = (M_PI/4)*(i+1);
    model = glm::translate(model, (radius*cos(angle),radius*sin(angle), -5 - i*6));
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices_box.size() / 3));

    glBindVertexArray(vao_sphere);
    model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    angle = (M_PI/4)*(i+1);
    model = glm::translate(model, (radius*cos(angle),radius*sin(angle), -5 - i*6));
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices_sphere.size() / 3));

    glBindVertexArray(vao_cone);
    model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    angle = (M_PI/4)*(i+1);
    model = glm::translate(model, (radius*cos(angle),radius*sin(angle), -5 - i*6));
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices_cone.size() / 3));

    glBindVertexArray(vao_torus);
    model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    angle = (M_PI/4)*(i+1);
    model = glm::translate(model, (radius*cos(angle),radius*sin(angle), -5 - i*6));
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices_torus.size() / 3));

    glBindVertexArray(vao_cylinder);
    model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    angle = (M_PI/4)*(i+1);
    model = glm::translate(model, (radius*cos(angle),radius*sin(angle), -5 - i*6));
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices_cylinder.size() / 3));

    glBindVertexArray(vao_pyramid);
    model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    angle = (M_PI/4)*(i+1);
    model = glm::translate(model, (radius*cos(angle),radius*sin(angle), -5 - i*6));
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices_pyramid.size() / 3));

    glBindVertexArray(vao_box);
    model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    angle = (M_PI/4)*(i+1);
    model = glm::translate(model, (radius*cos(angle),radius*sin(angle), -5 - i*6));
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glm::mat4x4 transformFromRotationID = glm::mat4(1.0f); // GL_FALSE, model[i][0]);
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices_box.size() / 3));
}
```

OpenGL

# Example: Rendering the Scene

- Rendering the scene:

```
// render the scene.  
s.render();
```

SimpleGL

- Rendering the scene:

```
// render loop  
while (!glfwWindowShouldClose(window))  
{  
    // start measuring frame time  
    auto t0 = std::chrono::high_resolution_clock::now();  
  
    // input  
    processInput(window);  
  
    // render  
    glClearColor(0.8f, 0.8f, 0.8f, 1.0f);  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
  
    // activate shader  
    glUseProgram(ID);  
  
    // pass projection matrix to shader (note that in this case it could change every frame)  
    glm::mat4 projection = glm::perspective(glm::radians(fov), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);  
    glUniformMatrix4fv(glGetUniformLocation(ID, "projection"), 1, GL_FALSE, &projection[0][0]);  
  
    // camera/view transformation  
    glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);  
  
    // pass camera/view transformation to shade  
    glUniformMatrix4fv(glGetUniformLocation(ID, "view"), 1, GL_FALSE, &view[0][0]);  
    glUniform3f(glGetUniformLocation(ID, "objectColor"), 1.0f, 0.5f, 0.71f);  
  
    // floor  
    glBindVertexArray(vao_box);  
    // calculate the model matrix for each object and pass it to shader before drawing  
    glm::mat4 model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first  
    model = glm::translate(model, glm::vec3(-35, -7, -35));  
    model = glm::scale(model, glm::vec3(70, 0.2, 70));  
    glUniformMatrix4fv(glGetUniformLocation(ID, "model"), 1, GL_FALSE, &model[0][0]);  
    glUniform3fv(glGetUniformLocation(ID, "objectColor"), 1, &GREY[0]);  
    glDrawArrays(GL_TRIANGLES, 0, (int)(vertices.size() / 0));  
}
```

OpenGL

# Example: Cleanup

- No additional code

```
Bluefish_@dyn-160-39-242-93:~/Desktop/4995/SimpleGL$ leaks 88877
Process:      demo [88877]
Path:         /Users/USER/Desktop/*/demo
Load Address: 0x102ef4000
Identifier:    demo
Version:      ???
Code Type:    X86-64
Parent Process: bash [85596]
[
Date/Time:     2019-05-03 00:10:41.296 -0400
Launch Time:   2019-05-03 00:10:33.414 -0400
OS Version:    Mac OS X 10.14.4 (18E226)
Report Version: 7
Analysis Tool:  /Applications/Xcode.app/Contents/Developer/usr/bin/leaks
Analysis Tool Version: Xcode 10.2.1 (10E1001)

Physical footprint:      51.0M
Physical footprint (peak): 51.2M
----

leaks Report Version: 4.0
Process 88877: 21438 nodes malloced for 5506 KB
Process 88877: 0 leaks for 0 total leaked bytes.
```

SimpleGL

- Explicit deletes

```
// optional: de-allocate all resources once they've outlived their purpose:
glDeleteVertexArrays(1, &vao_box);
glDeleteBuffers(1, &vbo_box);
glDeleteVertexArrays(1, &vao_obj);
glDeleteBuffers(1, &vbo_obj);
glDeleteVertexArrays(1, &vao_sphere);
glDeleteBuffers(1, &vbo_sphere);
glDeleteVertexArrays(1, &vao_p);
glDeleteBuffers(1, &vbo_p);
glDeleteVertexArrays(1, &vao_cylinder);
glDeleteBuffers(1, &vbo_cylinder);
glDeleteVertexArrays(1, &vao_tcone);
glDeleteBuffers(1, &vbo_tcone);
glDeleteVertexArrays(1, &vao_cone);
glDeleteBuffers(1, &vbo_cone);
glDeleteVertexArrays(1, &vao_torus);
glDeleteBuffers(1, &vbo_torus);

// glfw: terminate, clearing all previously allocated GLFW resources.
glfwTerminate();
```

OpenGL

## Example: Measurements

- Lines of Code: 62
- Average frame rate over 446 frames: 90 fps
- Lines of Code: 521
- Average frame rate over 467 frames: 64 fps

88% LoC reduction  
40% frame rate improvement

SimpleGL

OpenGL

# SimpleGL Implementation: Scene

- Scene

- Scene class handles boilerplate related to setting up a window, shaders (and shader classes), callbacks etc.
- the window setup portion is pretty mature, but the shading and callback portion can still be greatly customized
- Ref: [LearnOpenGL](#) tutorials

```
// build and compile our shader program
if (vs && fs) {
    light_shader = new Shader(vs, fs);
} else {
    light_shader = new Shader(ShaderType::light);
}
depth_shader = new Shader(ShaderType::depth);

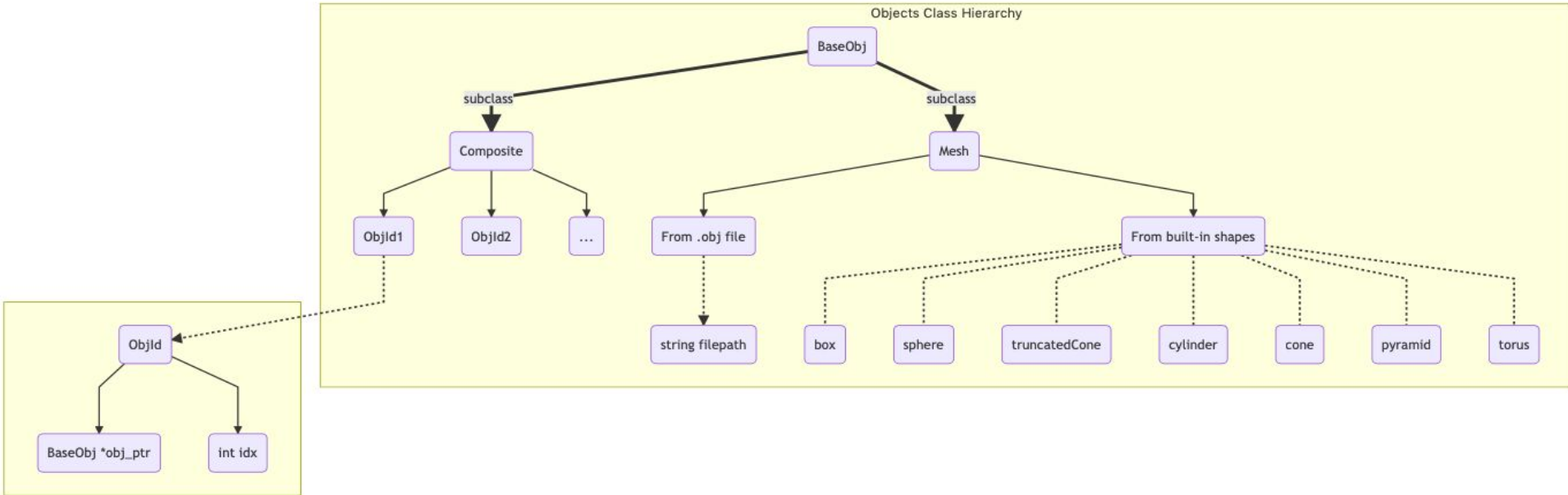
// configure depth map FBO
glGenFramebuffers(1, &depth_map_fbo);
// create depth texture
glGenTextures(1, &depth_map);
glBindTexture(GL_TEXTURE_2D, depth_map);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, shadow_params.depth_map_resolution, shadow_par
ams.depth_map_resolution, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
// attach depth texture as FBO's depth buffer
glBindFramebuffer(GL_FRAMEBUFFER, depth_map_fbo);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depth_map, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
}

Scene::~Scene() {
    // Clean up obj pointers.
    for (auto& entry : obj_map) {
        delete entry.second;
    }
    // Delete shaders.
    delete light_shader;
    delete depth_shader;
    // glfw: terminate, clearing all previously allocated GLFW resources.
    glfwTerminate();
}

ObjId Scene::add_obj(ObjType t, const Color c, ObjParams params) {
    int id = 0;
    BaseObj *obj_ptr;
    if (obj_map.find(t) != obj_map.end()) { // contains(s) is c++20
        // adding an instance of this shape to the scene
```

Boilerplate Code

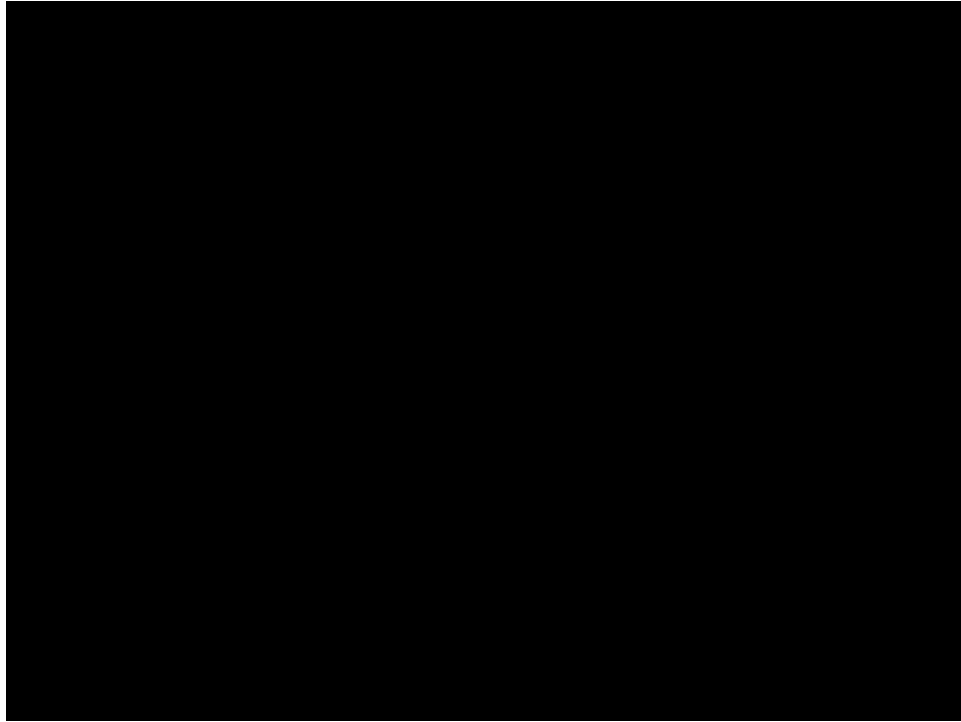
# SimpleGL Implementation: Objects



# SimpleGL Implementation: Misc

- Shadows
- Additional Key controls
  - Select shape types and instances of a shape type and manipulate them with key controls (see demo)
- Errors
  - Custom error conditions
- Utility methods/classes
  - GLP wrapper
  - Color
  - << overloading

# Demo





And live demo...!

# Future Work

- Subtractive compositions
- More variety in the built-in shapes
- Allow texture
- Scene configuration in JSON
- Multi-threaded support

# Acknowledgements

- We used some third party libraries to handle things like:  
loading .obj files, creating vertex array buffers for the various shapes etc.
  - glp.h: <https://www.ethanlipson.com/glp>
  - OBJ\_Loader.h: <https://github.com/Bly7/OBJ-Loader>
- Thanks to Professor Bjarne Stroustrup, TAs, and everyone else for feedback!

Thanks for listening!  
Questions?