Lahav Lipson, Yuxuan Mei, Crystal Ren

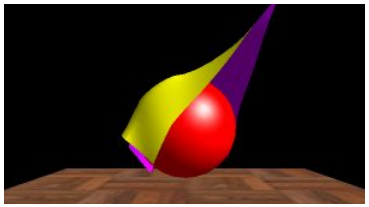
Project Link: https://github.com/Lahav174/SimpleGL

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)
http://www.paulsprojects.net/opengl/cloth/cloth.html

OpenGL Problems

- Difficult to learn and use due to extensive customizability and functionality
  - OpenGL almost entirely implemented in C
  - Lots of boilerplate and repeated code for most users
  - 3D Rendering/shading math must be implemented manually
    - Users must write their own shader programs in OpenGL
  - Non-intuitive relation between code and end render result
  - Users must track complex relations (binding vertex buffers, vertex array objects, frame buffers, sending variables to the shader)

SimpleGL

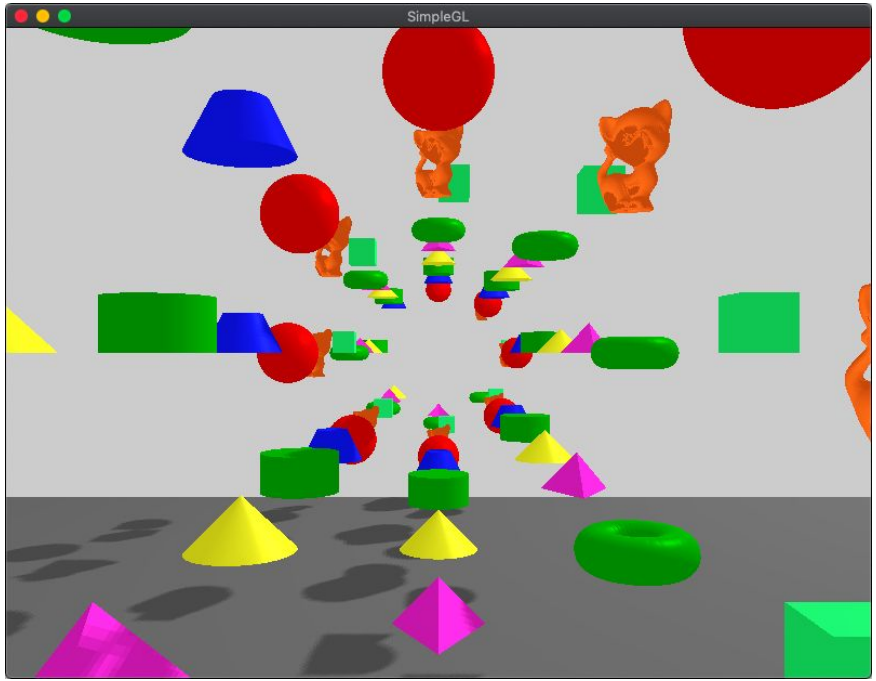
Pros

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

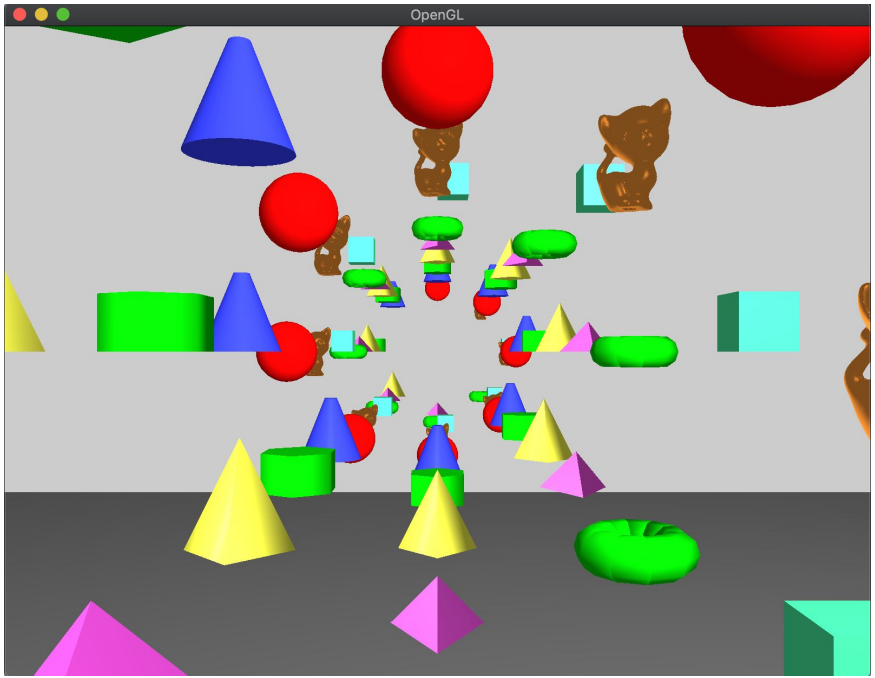
Cons

- Less flexibility and customizability than OpenGL
- Hides fundamental graphics concepts (ie. shader/rendering math, buffers, transformation matrices, etc.)

SimpleGL vs. OpenGL Example



SimpleGL



OpenGL

Example: Setup/Creating the Scene

- Create a Scene

```
Scene s; // initialize the scene.
s.set_callback(print_frame_rate); // optional: set callback
s.set_shadow(true); // optional: enable shadows
s.set_light_pos({30, 30, -30}); // optional: set the light position
```

SimpleGL

- Initialize UI, init shaders, data/frame buffers:

```
int main(int argc, char *argv[]) {

    if (argc < 3) {
        std::cout << "Usage: " << *argv << " <vshader_fpath> <fshader_fpath> [optional_.obj_path]\n";
        return 0;
    }
    // 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); // uncomment this statement to fix compilation on OS X
#endif

    lastX = (float) SCR_WIDTH / 2.0;
    lastY = (float) SCR_HEIGHT / 2.0;
    // glfw window creation
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "OpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    glfwSetCursorPosCallback(window, mouse_callback);
    glfwSetScrollCallback(window, scroll_callback);

    // tell GLFW to capture our mouse
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

    // glad: load all OpenGL function pointers
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // configure global opengl state
    glEnable(GL_DEPTH_TEST);

    // build and compile our shader program
    // 1. retrieve the vertex/fragment source code from filePath
    unsigned int ID;
    std::string vertexCode;
```

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"});
```

SimpleGL

- Add objects:

```
// render boxes
glGenVertexArrays(1, &vao_box);
glBindVertexArray(vao_box);
glGenBuffers(1, &vbo_box);
glBindBuffer(GL_ARRAY_BUFFER, vbo_box);
glBufferData(GL_ARRAY_BUFFER, vertices.size()*sizeof(double), vertices.data(), GL_STATIC_DRAW);
// position attribute
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 6*sizeof(double), (void*)0);
glEnableVertexAttribArray(0);
// texture coord attribute
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 6*sizeof(double), (void*)(3 * sizeof(double)));
glEnableVertexAttribArray(1);

// render spheres
auto verticesSphere = glp::sphere(3, 1);
glGenVertexArrays(1, &vao_sphere);
glBindVertexArray(vao_sphere);
glGenBuffers(1, &vbo_sphere);
glBindBuffer(GL_ARRAY_BUFFER, vbo_sphere);
glBufferData(GL_ARRAY_BUFFER, verticesSphere.size()*sizeof(double), verticesSphere.data(), GL_STATIC_DRAW);
// position attribute
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 6*sizeof(double), (void*)0);
glEnableVertexAttribArray(0);
// normal attribute
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE,6*sizeof(double), (void*)(3*sizeof(double)));
glEnableVertexAttribArray(1);

// render toruses
auto verticesTorus = glp::torus(10, 10, 0.7, 0.4);
glGenVertexArrays(1, &vao_torus);
glBindVertexArray(vao_torus);
glGenBuffers(1, &vbo_torus);
glBindBuffer(GL_ARRAY_BUFFER, vbo_torus);
glBufferData(GL_ARRAY_BUFFER, verticesTorus.size()*sizeof(double), verticesTorus.data(), GL_STATIC_DRAW);
// position attribute
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 6*sizeof(double), (void*)0);
glEnableVertexAttribArray(0);
// normal attribute
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE,6*sizeof(double), (void*)(3*sizeof(double)));
glEnableVertexAttribArray(1);

// render cones
auto verticesCone = glp::cone(4, 2, 1);
glGenVertexArrays(1, &vao_cone);
glBindVertexArray(vao_cone);
glGenBuffers(1, &vbo_cone);
glBindBuffer(GL_ARRAY_BUFFER, vbo_cone);
glBufferData(GL_ARRAY_BUFFER, verticesCone.size()*sizeof(double), verticesCone.data(), GL_STATIC_DRAW);
// position attribute
glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 6*sizeof(double), (void*)0);
glEnableVertexAttribArray(0);
// normal attribute
glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE,6*sizeof(double), (void*)(3*sizeof(double)));
glEnableVertexAttribArray(1);
```

OpenGL

Example: Transforming Objects

```
double angle = (M_PI/4)*(1+i);
sphere.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
sphere.rotate(angle * radToDeg, {0,0,1});
sphere.scale(1.5 + sin(angle*3));
angle = (M_PI/4)*(2+i);
truncated_cone.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
truncated_cone.rotate(angle * radToDeg, {0,0,1});
truncated_cone.scale(1.5 + sin(angle*3));
angle = (M_PI/4)*(3+i);
cylinder.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
cylinder.rotate(angle * radToDeg, {0,0,1});
cylinder.scale(1.5 + sin(angle*3));
angle = (M_PI/4)*(4+i);
cone.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
cone.rotate(angle * radToDeg, {0,0,1});
cone.scale(1.5 + sin(angle*3));
angle = (M_PI/4)*(5+i);
pyramid.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
pyramid.rotate(angle * radToDeg, {0,0,1});
pyramid.scale(1.5 + sin(angle*3));
angle = (M_PI/4)*(6+i);
torus.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
torus.rotate(angle * radToDeg, {0,0,1});
torus.scale(1.5 + sin(angle*3));
angle = (M_PI/4)*(7+i);
box.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
box.rotate(angle * radToDeg, {0,0,1});
box.scale(1.5 + sin(angle*3));
angle = (M_PI/4)*(8+i);
kitten.translate({radius*cos(angle),radius*sin(angle), -5 - i*6});
box.rotate(angle * radToDeg, {0,0,1});
box.scale(1.5 + sin(angle*3));
```

SimpleGL

- Transform objects:

OpenGL

Example: Rendering the Scene

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

SimpleGL

```
// 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, {-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() / 6));

    double angle;
    const double radius = 5;

    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)*(7+i);
        model = glm::translate(model, {radius*cos(angle),radius*sin(angle), -5 - i*6});
        glUniformMatrix4fv(glGetUniformLocation(ID,"model"), 1, GL_FALSE, &model[0][0]);
        glUniform3fv(glGetUniformLocation(ID, "objectColor"), 1, &TEAL[0]);
        glDrawArrays(GL_TRIANGLES, 0, (int) (vertices.size() / 6));

        glBindVertexArray(vao_sphere);
        model = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
        angle = (M_PI/4)*(1+i);
        model = glm::translate(model, {radius*cos(angle),radius*sin(angle), -5 - i*6});
```

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
- Averaged over 479 frames: 63 fps

SimpleGL

- Lines of Code: 521
- Averaged over 363 frames: 63 fps

OpenGL

88% LoC reduction
Same speed as OpenGL

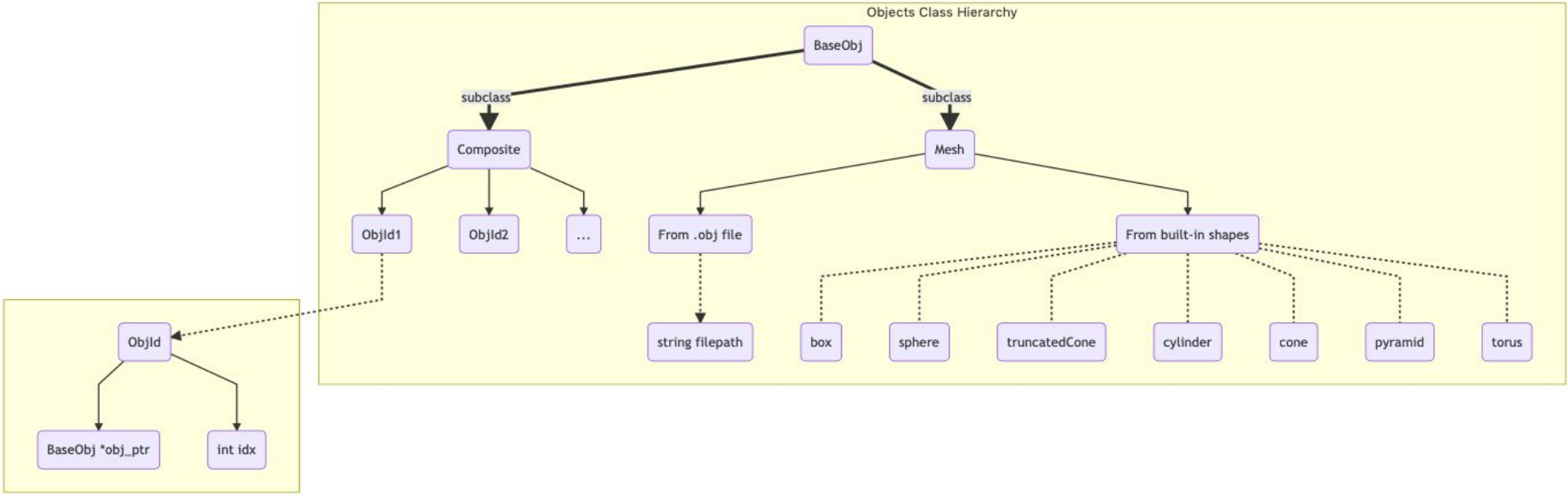
SimpleGL Implementation: Scene

- Scene
  - Scene class handles boilerplate code for window setup, default shaders, UI-callbacks, rendering logic/math, etc.
  - User allowed to provide custom shaders, change colors, modify lighting, custom callback, transform objects

```
Scene::Scene(bool use_full_ctrl, char *vs, char *fs, int width, int height) {
    // glfw: initialize and configure
    glfwSetErrorCallback(error_callback);
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    // uncomment this statement to fix compilation on OS X
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    scr_width = width;
    scr_height = height;
    window = glfwCreateWindow(scr_width, scr_height, "SimpleGL", nullptr, nullptr);
    glfwGetFramebufferSize(window, &scr_width, &scr_height);
    last_x = (float) scr_width / 2.0;
    last_y = (float) scr_height / 2.0;
    if (window == nullptr) {
        std::cout << "Failed to create GLFW window\n";
        glfwTerminate();
        abort();
    }
    glfwMakeContextCurrent(window);
    if (use_full_ctrl) {
        glfwSetKeyCallback(window, key_callback);
    }
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    glfwSetCursorPosCallback(window, mouse_callback);
    glfwSetScrollCallback(window, scroll_callback);

    // glad: load all OpenGL function pointers
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress)) {
        std::cout << "Failed to initialize GLAD" << std::endl;
        glfwTerminate();
        abort();
    }

    // configure global opengl state
    glEnable(GL_DEPTH_TEST);

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

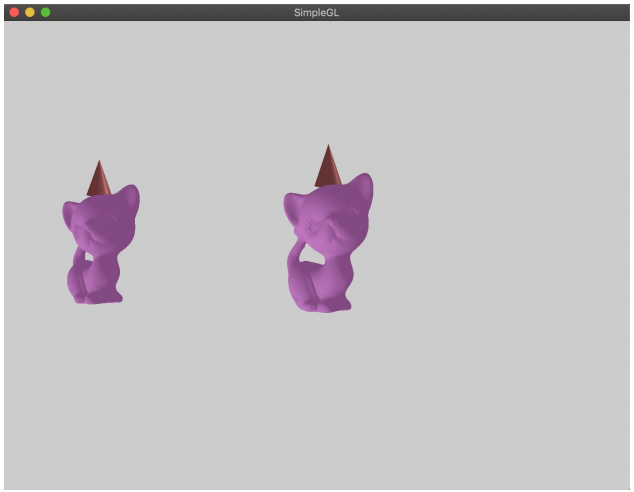
Boilerplate Code

SimpleGL Implementation: Objects



Composite Objects: example

```
// add a composite object of a kitten mesh and a default cone.
ObjId cone = s.add_obj(Shape::cone, pink);
Color peach(0.8, 0.6, 0.8);
ObjId obj = s.add_obj("kitten", peach, oparams);
oparams.comp = {cone, obj};
ObjId c1 = s.add_obj(Shape::composite, peach, oparams);
```



SimpleGL Features: Misc

- Shadows
- Blinn-Phong Shading
- 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
  - Mesh (group of objects w/ same type)
  - ObjId (single object instance)
  - Color
  - << overloading

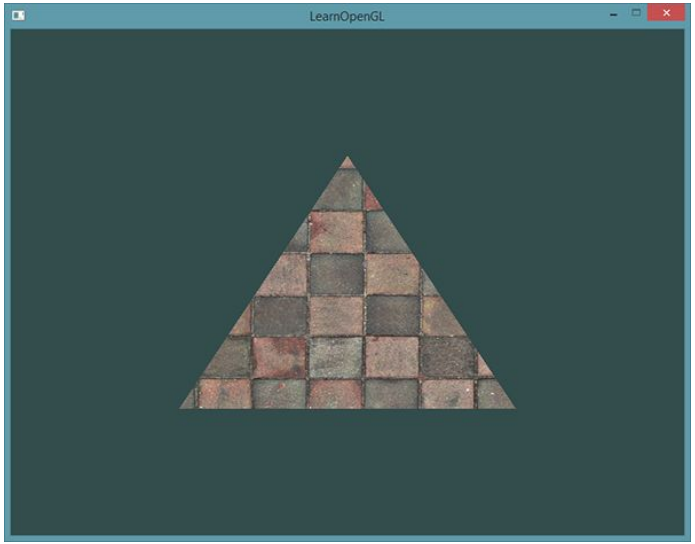
Demos

- https://www.youtube.com/watch?v=jURae_25UHM
- https://www.youtube.com/watch?v=8WRg9SgDZhA

And live demo…!

Future Work

- Subtractive compositions
- More variety in the built-in shapes
- Allow texture
- Scene configuration in JSON
- Multi-threaded support
- Expose render loop to allow user to implement custom rendering logic (like reflections)



Acknowledgements

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

Questions?