Lahav Lipson, Yuxuan Mei, Crystal Ren

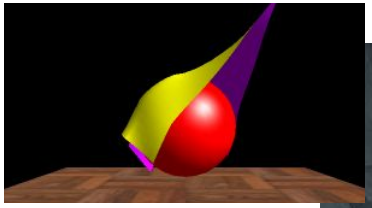
Project Link: https://github.com/Lahav174/SimpleGL

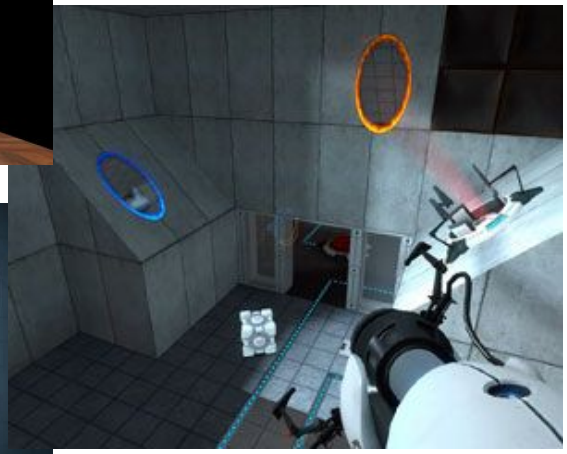
Abstract

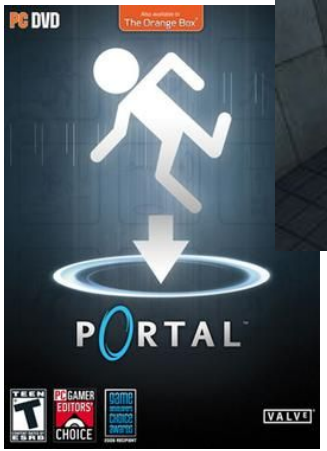
OpenGL is complex due to its high level of customizability. However, the majority of users and use-cases do not require extensive customizability yet are still burdened with OpenGL's complexity. Our goal is to eliminate the extensive boilerplate code and streamline and/or automate a lot of setup and 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)



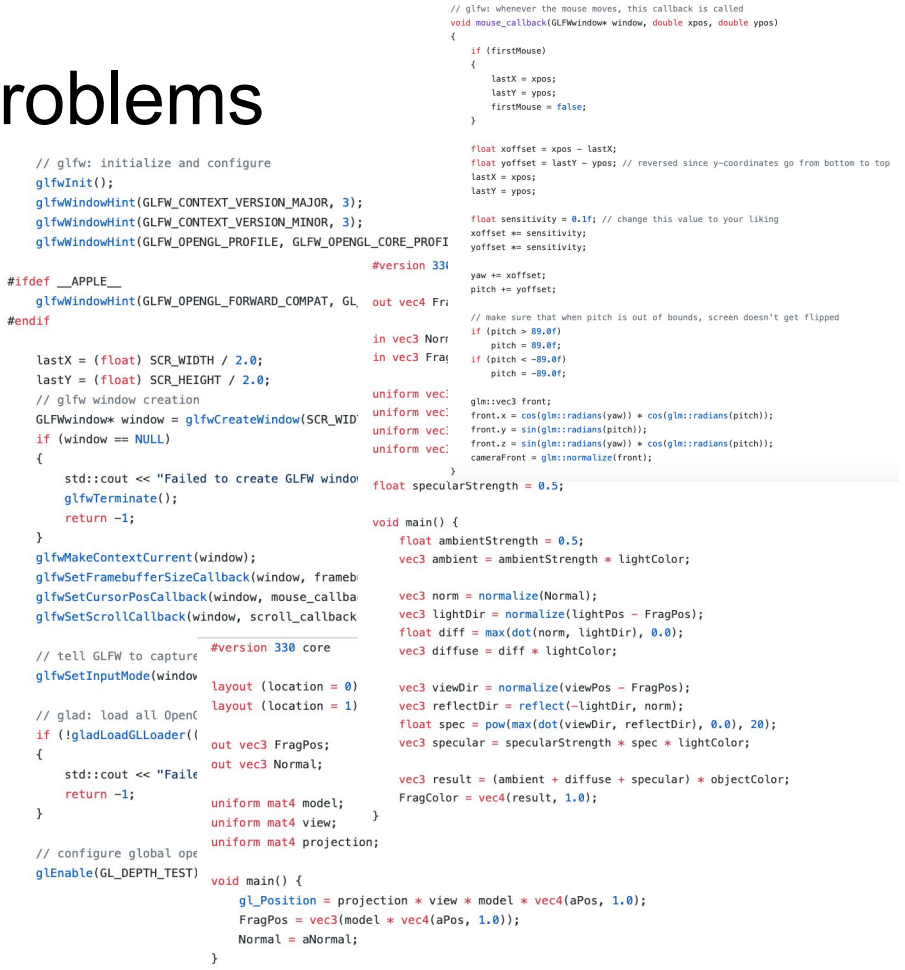




https://en.wikipedia.org/wiki/Portal_(video_game)
http://www.paulsprojects.net/opengl/cloth/cloth.html

OpenGL Problems

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



SimpleGL

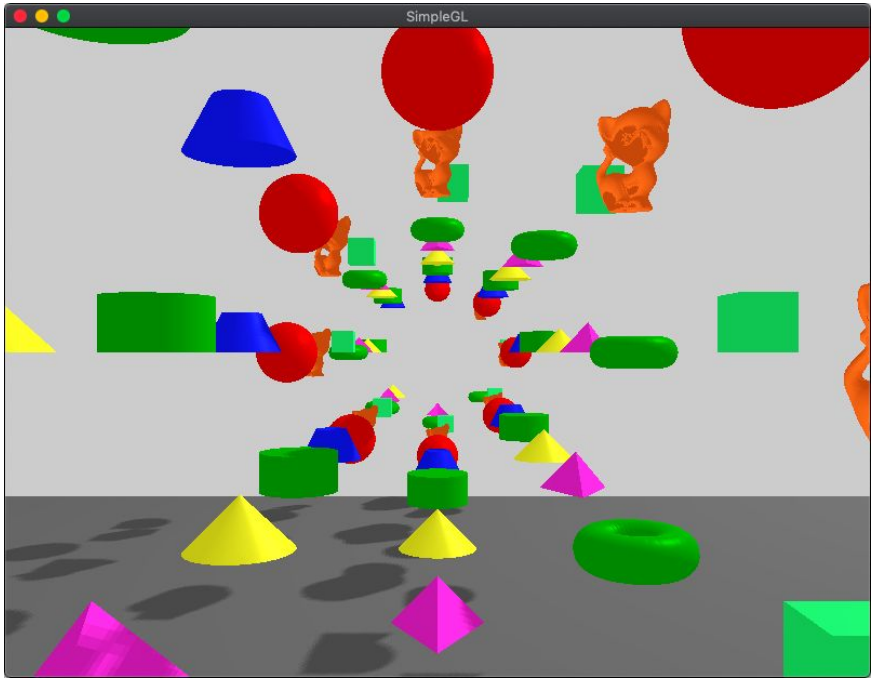
Pros

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

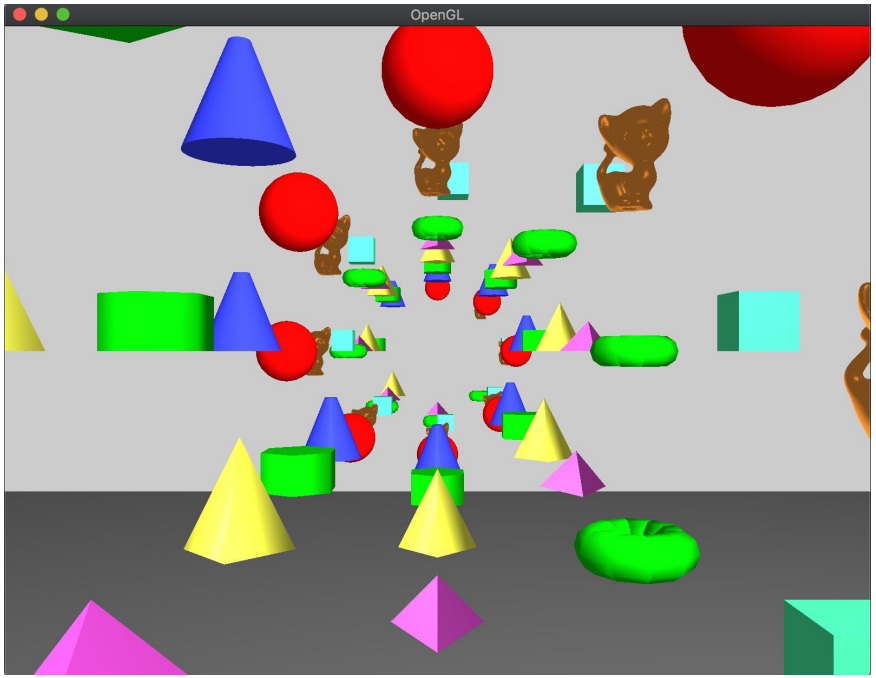
Cons

- 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:

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 cylinders
auto verticesCylinder = glp::cylinder(6, 1, 1);
glGenVertexArrays(1, &vao_cylinder);
glBindVertexArray(vao_cylinder);
glGenBuffers(1, &vbo_cylinder);
glBindBuffer(GL_ARRAY_BUFFER, vbo_cylinder);
glBufferData(GL_ARRAY_BUFFER, verticesCylinder.size()*sizeof(double), verticesCylinder.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 pyramid
auto verticesPyramid = glp::pyramid(3, 1, 1);
glGenVertexArrays(1, &vao_p);
glBindVertexArray(vao_p);
glGenBuffers(1, &vbo_p);
glBindBuffer(GL_ARRAY_BUFFER, vbo_p);
glBufferData(GL_ARRAY_BUFFER, verticesPyramid.size()*sizeof(double), verticesPyramid.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 obj
std::vector<double> verticesObj;
if (argc > 3) {
    obj::Loader loader;
    loader.LoadFile(std::string(argv[3]));
    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);
    }
    assert (verticesObj.size() == 6 * loader.LoadedVertices.size());
    glGenVertexArrays(1, &vao_obj);
    glBindVertexArray(vao_obj);
    glGenBuffers(1, &vbo_obj);
    glBindBuffer(GL_ARRAY_BUFFER, vbo_obj);
    glBufferData(GL_ARRAY_BUFFER, verticesObj.size()*sizeof(double), verticesObj.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 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);

// render truncated cones
auto verticesTCone = glp::truncatedCone(32, 2, 1, 0.2);
glGenVertexArrays(1, &vao_tcone);
glBindVertexArray(vao_tcone);
glGenBuffers(1, &vbo_tcone);
glBindBuffer(GL_ARRAY_BUFFER, vbo_tcone);
glBufferData(GL_ARRAY_BUFFER, verticesTCone.size()*sizeof(double), verticesTCone.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

- 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:

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, {-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));
```

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

SimpleGL

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

OpenGL

88% LoC reduction
40% frame rate improvement

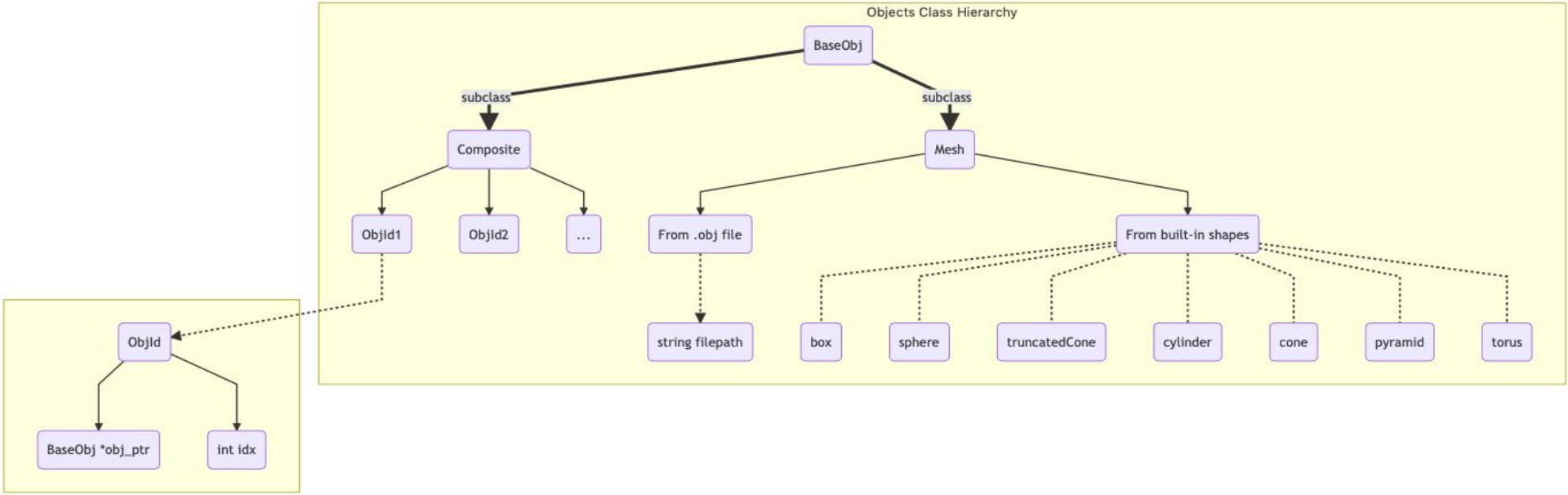
SimpleGL Implementation: Scene

- 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?