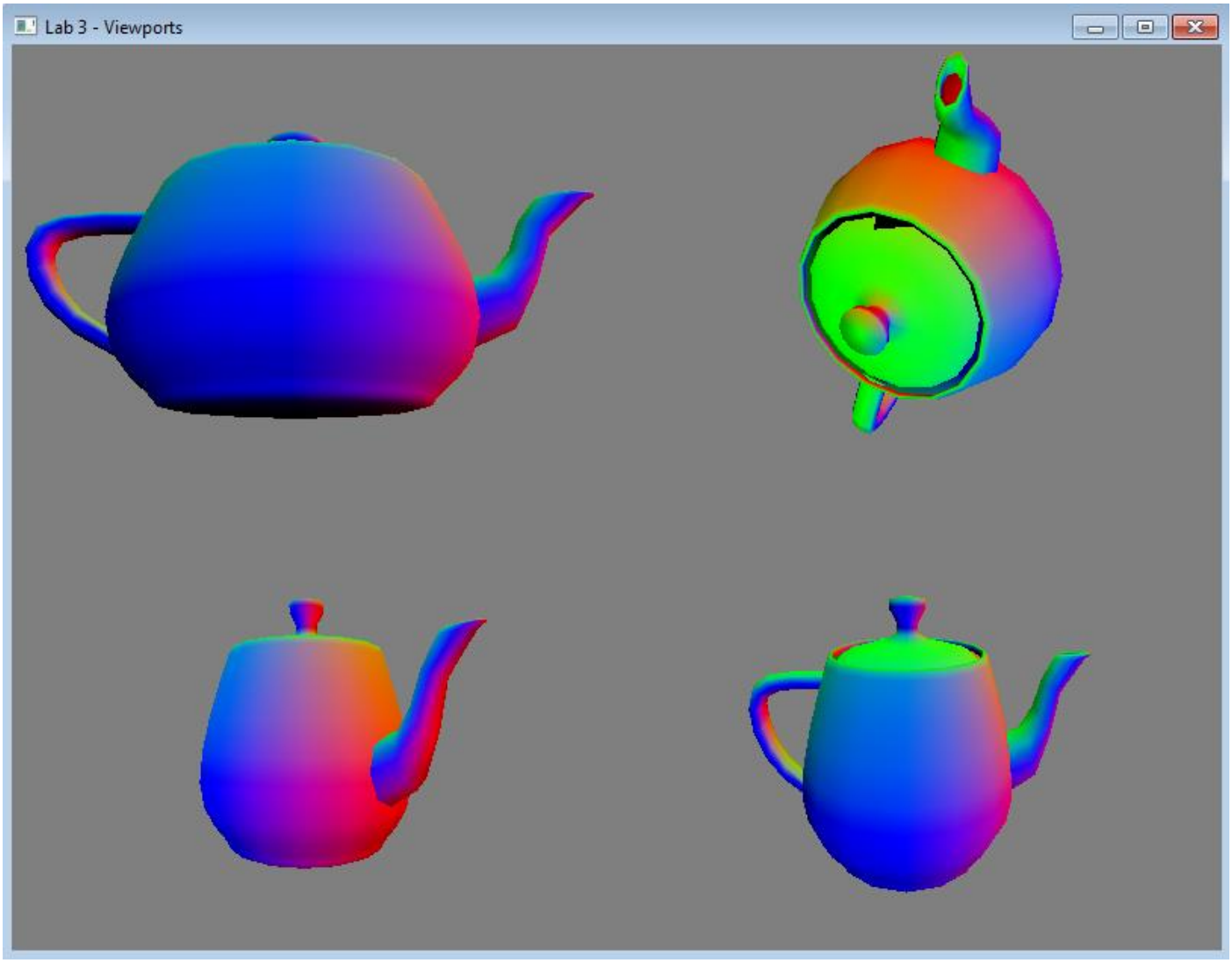
Lab 3 - Viewing
Computer Graphics CS4052

Edmond O’ Flynn
12304742

“Each viewport should show a different view of the teapot”



I modified the sample code given to have a class for viewports. In this class, I moved all repeatable code associated with displaying in order to reduce duplication of code, and allow for quicker modifications through refactoring.

“The teapot should appear moving in at least one viewport”

The top right and bottom left viewports move independently of one another through accessing independent viewport objects in the vector object.

```
for (Perspective perspective : perspectives) {
    perspective.draw(proj_mat_location, view_mat_location, matrix_location, teapot_vertex_count);
}
```

```
void draw(int proj_mat_location, int view_mat_location, int matrix_location, int teapot_vertex_count) {
    switch (position) {
    case tl:
        glViewport(0, height / 2, width / 2, height / 2);
        break;
    case tr:
        glViewport(width / 2, height / 2, width / 2, height / 2);
        break;
    case bl:
        glViewport(0, 0, width / 2, height / 2);
        break;
    case br:
        glViewport(width / 2, 0, width / 2, height / 2);
        break;
    }

    glUniformMatrix4fv(proj_mat_location, 1, GL_FALSE, getProj().m);
    glUniformMatrix4fv(view_mat_location, 1, GL_FALSE, getView().m);
    glUniformMatrix4fv(matrix_location, 1, GL_FALSE, getModel().m);
    glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);
}

float getAngle() { return this->angle; }
void rotateObject(float angle) {
    this->angle += angle;
    model = rotate_x_deg(identity_mat4(), this->angle) *
        rotate_y_deg(identity_mat4(), this->angle) *
        rotate_z_deg(identity_mat4(), this->angle);
}

void spin(float angle) {
    this->angle += angle;
    model = rotate_y_deg(identity_mat4(), this->angle);
}
```

“The teapot should appear static in at least one viewport”

Two of the viewports have static objects that don’t undergo any transformations during update.

```
void updateScene() {
    // Wait until at least 16ms passed since start
    static double last_time = 0;
    double curr_time = timeGetTime();
    float delta = (curr_time - last_time) * 0.001f;
    if (delta > 0.03f)
        delta = 0.03f;
    last_time = curr_time;

    //update objects in scene
    perspectives.at(1).rotateObject(0.01f);
    perspectives.at(2).spin(-0.01f);

    // Draw the next frame
    glutPostRedisplay();
}
```

“At least one viewport should depict an orthographic projection, and you must do this using an orthographic projection matrix as presented in the notes”

Two constructors are provided, one for perspective viewports, and another for orthographic viewports. I used the given perspective function from the sample code, but I created my own mat4 viewport using some linear algebra.

```
Perspective(Position position, float fov, vec3 pos, Rotation rotation = Rotation::none, float angle = 0) {
    this->position = position;
    this->angle = angle;
    this->view = translate(identity_mat4(), pos);
    this->proj = perspective(fov, (float)width / (float)height, 0.1, 100);
    switch (rotation) {
    case Perspective::x:
        model = rotate_x_deg(identity_mat4(), angle);
        break;
    case Perspective::y:
        model = rotate_y_deg(identity_mat4(), angle);
        break;
    case Perspective::z:
        model = rotate_z_deg(identity_mat4(), angle);
        break;
    default:
        model = identity_mat4();
        break;
    }
}
```

```
Perspective(Position position, mat4 ortho, float fov, Rotation rotation = Rotation::none, float angle = 0) {
    this->position = position;
    this->angle = angle;
    this->view = look_at(vec3(0.0f, 0.0f, 35.0f), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
    this->proj = ortho; // perspective(fov, (float)width / (float)height, 0.1, 100);
    switch (rotation) {
    case Perspective::x:
        model = rotate_x_deg(identity_mat4(), angle);
        break;
    case Perspective::y:
        model = rotate_y_deg(identity_mat4(), angle);
        break;
    case Perspective::z:
        model = rotate_z_deg(identity_mat4(), angle);
        break;
    default:
        model = identity_mat4();
        break;
    }
}
```

```
static mat4 ortho(float b, float t, float l, float r, float n, float f) {
    mat4 m = zero_mat4();
    m.m[0] = (2 / (r - l));
    m.m[3] = -((r + l) / (r - l));
    m.m[5] = (2 / (t - b));
    m.m[7] = -((t + b) / (t - b));
    m.m[10] = -(2 / (f - n));
    m.m[11] = -((f + n) / (f - n));
    m.m[15] = 1.0f;
    return m;
}
```

“At least two viewports should depict a perspective projection, each with different properties”

Both of the perspective viewports have different properties, one is static, the other is dynamic. The FOVs are different, as well as the positions of the objects in each respective viewport.

```
for (auto i = 0; i < sizeof(teapot_vertex_points) / sizeof(teapot_vertex_points[0]); i += 3) {
    min_x = min_x < teapot_vertex_points[i] ? min_x : teapot_vertex_points[i];
    max_x = max_x > teapot_vertex_points[i] ? max_x : teapot_vertex_points[i];

    min_y = min_y < teapot_vertex_points[i + 1] ? min_y : teapot_vertex_points[i + 1];
    max_y = max_y > teapot_vertex_points[i + 1] ? max_y : teapot_vertex_points[i + 1];

    min_z = min_z < teapot_vertex_points[i + 2] ? min_z : teapot_vertex_points[i + 2];
    max_z = max_z > teapot_vertex_points[i + 2] ? max_z : teapot_vertex_points[i + 2];
}

// perspective + static
perspectives.push_back(Perspective(Perspective::tl, 35, vec3(0.0f, 0.0f, -40.0f),
    Perspective::Rotation::x, -15));
// perspective + moving
perspectives.push_back(Perspective(Perspective::tr, 45, vec3(0.0f, 0.0f, -40.0f),
    Perspective::Rotation::x, 15));
// orthographic + look at + moving
perspectives.push_back(Perspective(Perspective::bl,
    Perspective::ortho(min_y / 20, max_y / 20, min_x / 20, max_x / 20, 0.1f, 100.0f),
    90.0f));
// orthographic + look at + static
perspectives.push_back(Perspective(Perspective::br,
    Perspective::ortho(min_y / 20, max_y / 20, min_x / 20, max_x / 20, 0.1f, 100.0f),
    60.0f, Perspective::Rotation::x, 15.0f));
```

“At least one viewport should use the look-at function for the view matrix”

Both of the orthographic projections use the provided look_at() function.