CS 405 Computer Graphics 3D Project Part 1

Sadi Çelik 25542

First Scene:

In this scene I drew four basic shapes (a sphere, a torus and two random parametric shapes) in wireframe mode using provided parametric functions. First two shapes (sphere and torus) were created.

```
GenerateParametricShapeFrom2D(positions, normals, indices, ParametricHalfCircle, 16,
16);
VAO sphereVAO(positions, normals, indices);

GenerateParametricShapeFrom2D(positions, normals, indices, ParametricCircle, 16, 16);
VAO torusVAO(positions, normals, indices);

GenerateParametricShapeFrom2D(positions, normals, indices, ParametricSpikes, 64, 32);
VAO parametric_one_VAO(positions, normals, indices);

GenerateParametricShapeFrom2Dv2(positions, normals, indices, ParametricSpikes, 1024,
1024);VAO parametric_two_VAO(positions, normals, indices);
```

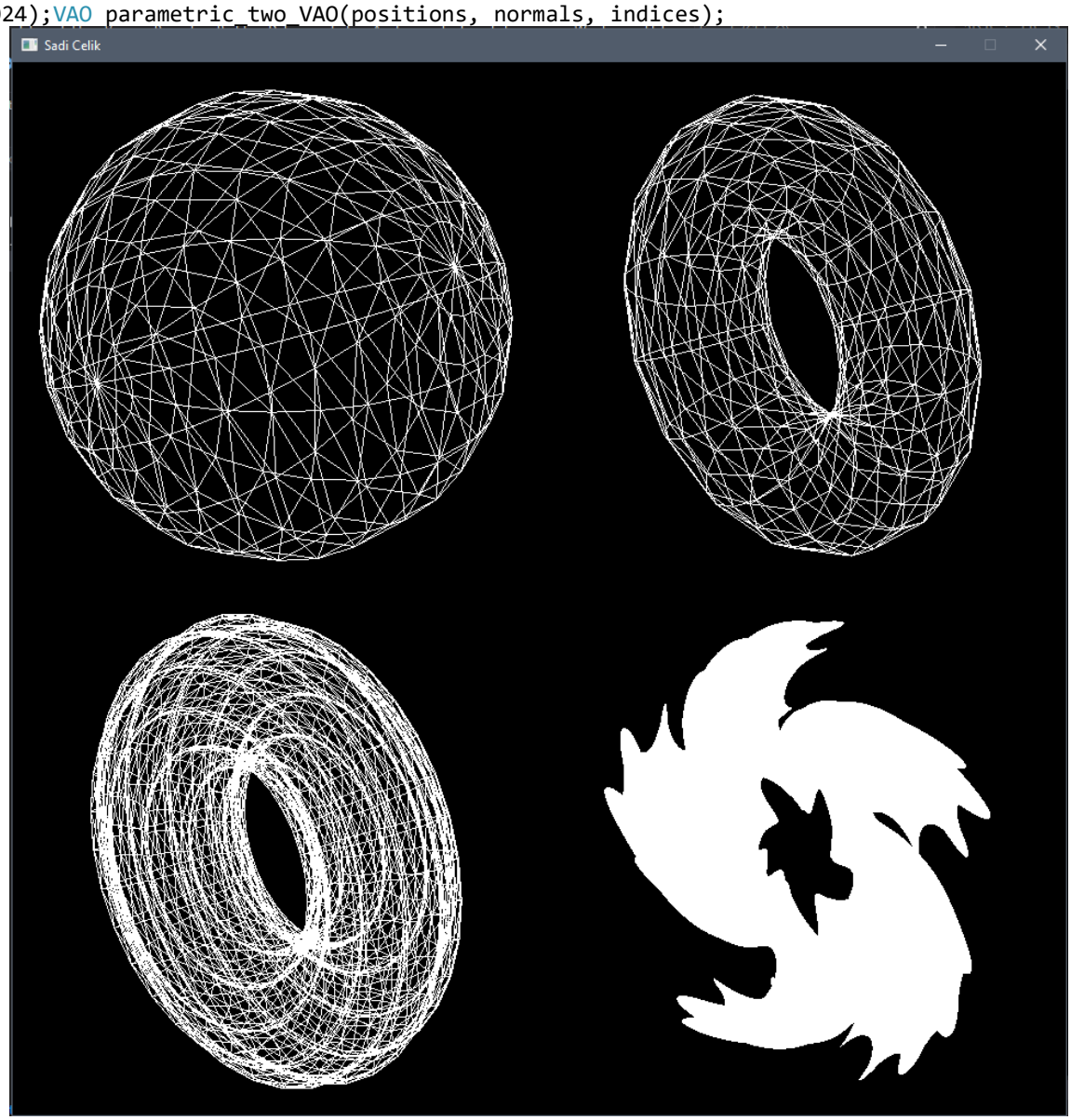

Figure 1: Scene 1

Then, I have created `GenerateParametricShapeFrom2Dv2` from `GenerateParametricShapeFrom2D` to create the 4th shape. I used `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);` to enable the wireframe mode. I closed this mode in other scenes using `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);`

My `GenerateParametricShapeFrom2Dv2` function's drawing part looks like:

```
auto parametric_surface = [parametric_line](double t, double r)
{
	auto p = glm::dvec3(parametric_line(t), 0);

	p *= (sin(r * 5 * glm::two_pi<double>()) + 3) / 4.;
	p.y *= (pow(sin((r + 0.5) * 5 * glm::two_pi<double>()), 6) + 3) / 3;
	auto xy_len = glm::length(glm::vec2(p));
	p.y *= pow(xy_len, 1.3);
	auto a = sin(xy_len * 1.2 * glm::two_pi<double>() * 0.4);

	return glm::rotateY(p, a + r * glm::two_pi<double>());
};
```

Except 5th scene I used below rotation the rotate the shapes around x and y axis.

```
transform = glm::rotate(transform, glm::radians(float(glfwGetTime() * 10)),
glm::vec3(1, 1, 0));
```

My program for the first scene:

```
GLuint scene_one = CreateProgramFromSources(
	R"VERTEX(
	#version 330 core

	layout(location = 0) in vec3 a_position;
	layout(location = 1) in vec3 a_normal;

	uniform mat4 u_transform;

	out vec3 vertex_position;
	out vec3 vertex_normal;

	void main()
	{
		gl_Position = u_transform * vec4(a_position, 1);
		vertex_normal = (u_transform * vec4(a_normal, 0)).xyz;
		vertex_position = gl_Position.xyz;
	}
	)VERTEX",

	R"FRAGMENT(
	#version 330 core

	out vec4 out_color;

	void main()
	{
		out_color = vec4(1, 1, 1, 1);
	}
	)FRAGMENT");
```

For the first four scenes with four objects, I translated and scaled each object.

```
transform = glm::scale(transform, glm::vec3(0.45f)); // Scaled by 0.45
transform = glm::translate(glm::vec3(-0.5, 0.5, 0)); // Sphere translation
transform = glm::translate(glm::vec3(0.5, 0.5, 0));  // Torus translation
transform = glm::translate(glm::vec3(-0.5, -0.5, 0));// Parametric one translation
transform = glm::translate(glm::vec3(0.5, -0.5, 0)); // Parametric two translation
```

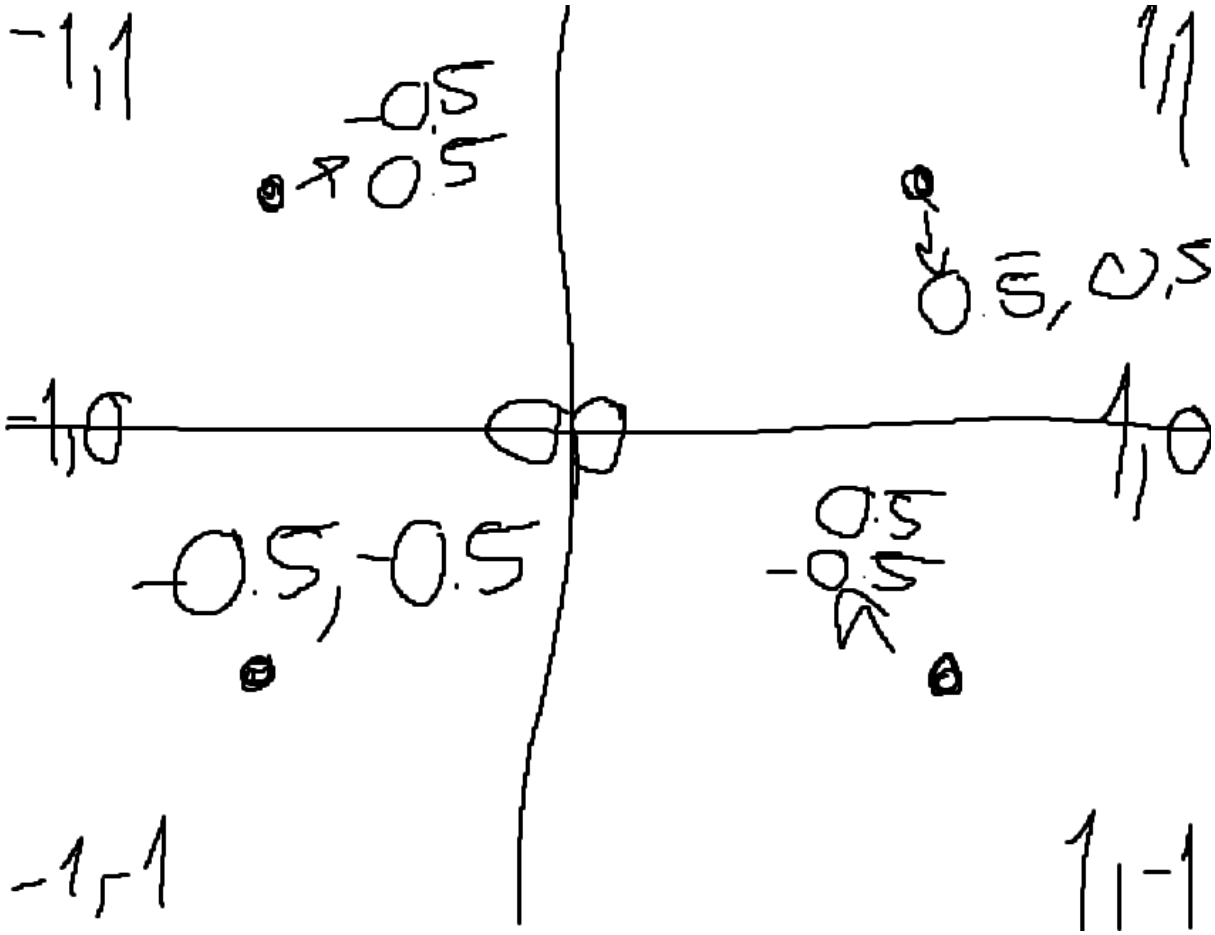

Figure 2: Coordinate plane for objects

Second Scene:

Similarly, to first scene I drew 4 objects at specific locations. But this time in the fragment shader of my program, I normalized and used vertex normal as colors. Vertex shader stayed as the same.

My program for the second scene (Only fragment shader):

```
R"FRAGMENT(
#version 330 core

in vec3 vertex_normal;
out vec4 out_color;

void main()
{
	vec3 color = normalize(vertex_normal);
	out_color = vec4(color, 1);
}
)FRAGMENT"
```

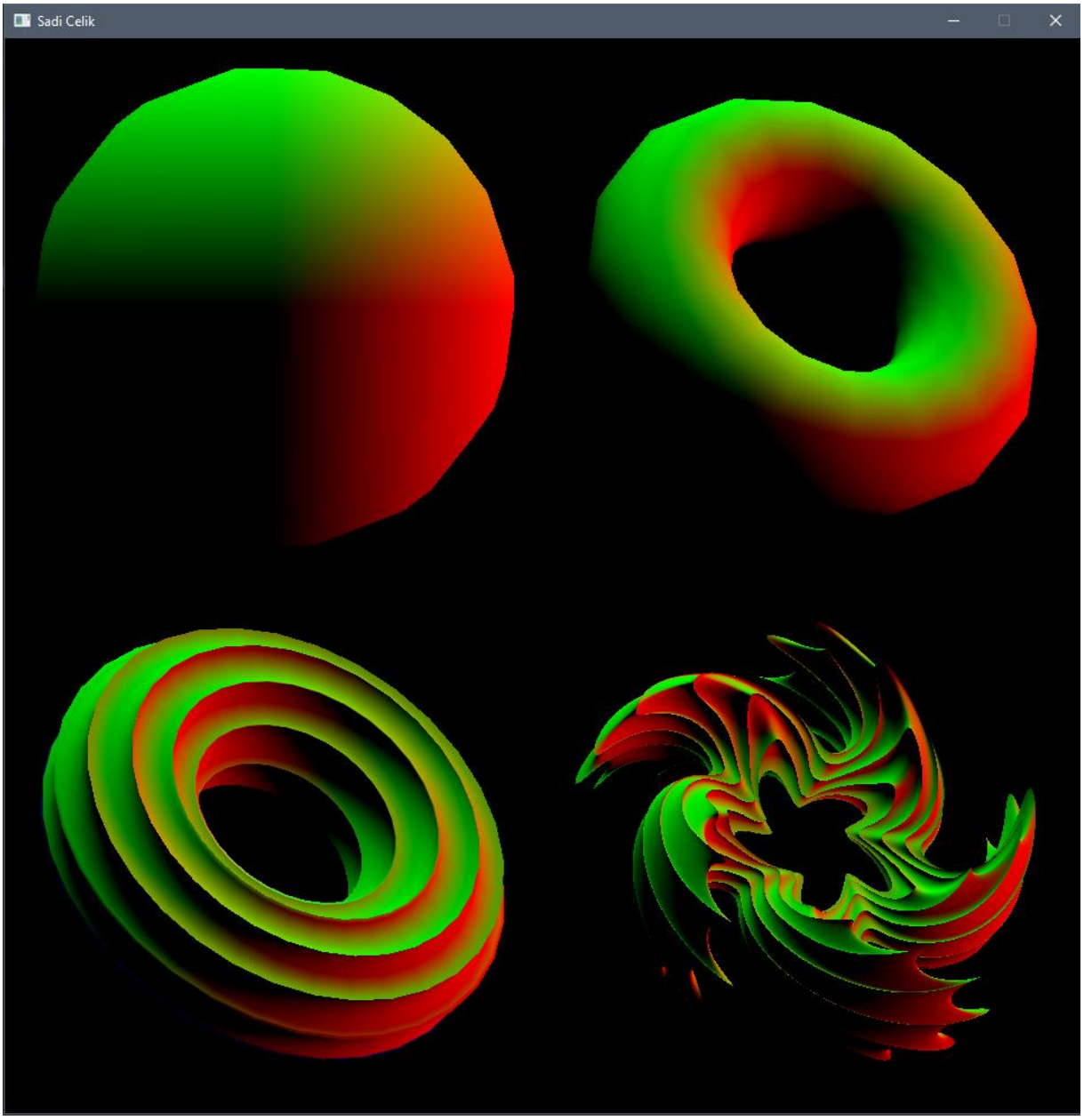

Figure 3: Scene 2

Third Scene:

Similarly, to other scenes I drew 4 objects at specific locations. But this time in the fragment shader of my program, I implemented Blinn-Phong reflection model to shade. The colors of the objects were gray. There was a single major directional light with three components. Ambient light, diffuse light, and specular light. I used the given values to get the desired result. Vertex shader stayed as the same.

My program for the third scene (Only fragment shader):

```
R"FRAGMENT(
#version 330 core

in vec3 vertex_position;
in vec3 vertex_normal;

out vec4 out_color;

void main()
{
	vec3 color = vec3(0);

	vec3 surface_color = vec3(0.5, 0.5, 0.5);
	vec3 surface_position = vertex_position;
	vec3 surface_normal = normalize(vertex_normal);

	// Ambient light
	float ambient_k = 1;
	vec3 ambient_color = vec3(0.5, 0.5, 0.5);
	color += ambient_k * ambient_color * surface_color;

	vec3 light_direction = normalize(vec3(-1, -1, 1));
	vec3 to_light = -light_direction;
	vec3 light_color = vec3(0.4, 0.4, 0.4);

	// Diffuse light
	float diffuse_k = 1;
	float diffuse_intensity = max(0, dot(to_light, surface_normal));
	color += diffuse_k * diffuse_intensity * light_color * surface_color;

	// Specular Lighting
	vec3 view_dir = vec3(0, 0, -1);
	vec3 halfway_dir = normalize(view_dir + to_light);

	float specular_k = 1;
	float shininess = 64;
	float specular_intensity = max(0, dot(halfway_dir, surface_normal));
	color += specular_k * pow(specular_intensity, shininess) * light_color;

	out_color = vec4(color, 1);
}
)FRAGMENT"
```

We have a gray surface color `vec3 surface_color = vec3(0.5, 0.5, 0.5);` with a single directional light coming from `vec3 light_direction = normalize(vec3(-1, -1, 1));` with color `vec3 light_color = vec3(0.4, 0.4, 0.4);`. And the ambient light's color is `vec3 ambient_color = vec3(0.5, 0.5, 0.5);` .

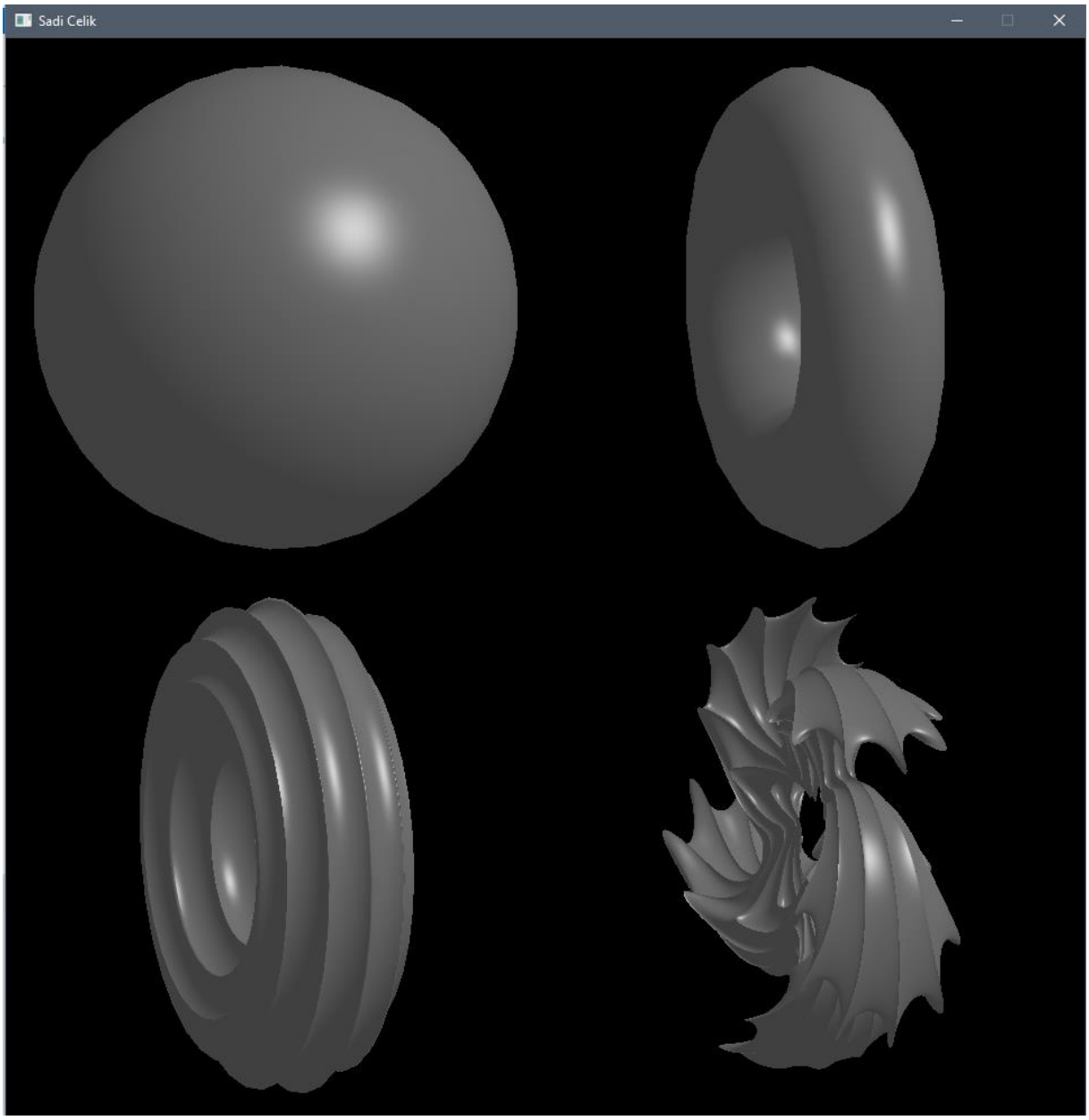

Figure 4: Scene 3

Fourth Scene:

Similarly, to other scenes I drew 4 objects at specific locations. But this time I used four different fragment shaders and four different programs to achieve different surface colors. I implemented Blinn-Phong reflection model to shade. The only major difference in those fragment shaders was the color. And had different shininess values. Vertex shader stayed as the same.

My program for the fourth scene (Gray fragment shader (Other shaders are similar)):

```
const GLchar* fragment_shader_gray = R”FRAGMENT(
	#version 330 core

	uniform vec2 u_mouse_position;

	in vec3 vertex_position;
	in vec3 vertex_normal;

	out vec4 out_color;

	void main()
	{
		vec3 color = vec3(0);

		vec3 surface_color = vec3(0.5, 0.5, 0.5);
		vec3 surface_position = vertex_position;
		vec3 surface_normal = normalize(vertex_normal);

		// Ambient light
		float ambient_k = 1;
		vec3 ambient_color = vec3(0.5, 0.5, 0.5);
		color += ambient_k * ambient_color * surface_color;

		vec3 light_direction = normalize(vec3(-1, -1, 1));
		vec3 to_light = -light_direction;
		vec3 light_color =  vec3(0.4, 0.4, 0.4);

		// Diffuse light
		float diffuse_k = 1;
		float diffuse_intensity = max(0, dot(to_light, surface_normal));
		color += diffuse_k * diffuse_intensity * light_color * surface_color;

		// Specular Lighting
		vec3 view_dir = vec3(0, 0, -1);
		vec3 halfway_dir = normalize(view_dir + to_light);

		float specular_k = 1;
		float shininess = 128;
		float specular_intensity = max(0, dot(halfway_dir, surface_normal));
		color += specular_k * pow(specular_intensity, shininess) * light_color;

		// Light 2
		vec3 point_light_position = vec3(u_mouse_position, -1);
		vec3 point_light_color =  vec3(0.5, 0.5, 0.5);
		vec3 to_point_light = normalize(point_light_position – surface_position);

		// Diffuse light
		diffuse_k = 1;
		diffuse_intensity = max(0, dot(to_point_light, surface_normal));
```

```
        color += diffuse_k * diffuse_intensity * point_light_color *
        surface_color;

        // Specular Lighting
        view_dir = vec3(0, 0, -1);
        halfway_dir = normalize(view_dir + to_point_light);

        specular_k = 1;
        shininess = 128;
        specular_intensity = max(0, dot(halfway_dir, surface_normal));
        color += specular_k * pow(specular_intensity, shininess) *
        point_light_color;

        out_color = vec4(color, 1);
    }
    )FRAGMENT
```

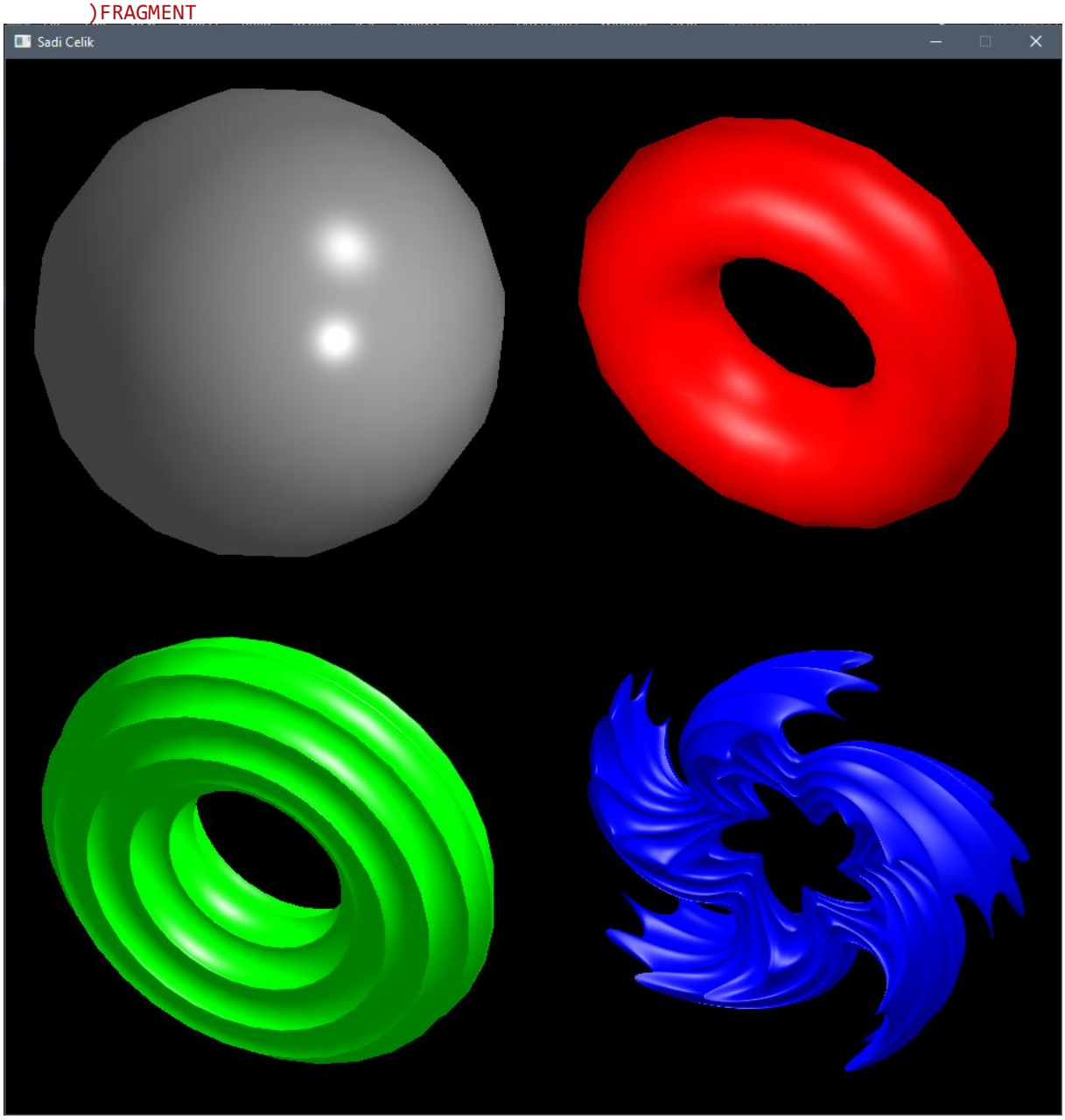

Figure 5: Scene 4

In this scene there are two major lights. One of them is a directional light same from the third scene and one point light at the mouse position with color `vec3 point_light_color = vec3(0.5, 0.5, 0.5);`. (You can change the point's light position with your cursor.)

```
vec3 surface_color = vec3(0.5, 0.5, 0.5); // Sphere color
float shininess = 128;                    // Sphere shininess

vec3 surface_color = vec3(1, 0, 0);       // Torus color
float shininess = 32;                     // Torus shininess

vec3 surface_color = vec3(0, 1, 0);       // Parametric one color
float shininess = 32;                     // Parametric one shininess

vec3 surface_color = vec3(0, 0, 1);       // Parametric two color
float shininess = 32;                     // Parametric two shininess
```

Fifth Scene:

In this scene, I implemented the game using the red, gray, and green fragment shaders which I used in fourth scene. I translated my colorful sphere at my mouse position. The other chasing sphere was created with `glm::dvec2 chasing_pos = glm::mix(mouse_position, chasing_pos, 0.99f);` I changed the shaders to red from green when the condition below is satisfied.

```
double distance_bet = abs(glm::distance(mouse_position, chasing_pos));

if (distance_bet >= 0.3 * 2)
{
	u_transform_location = glGetUniformLocation(scene_four_obj3, "u_transform");
	glUseProgram(scene_four_obj3);
}
else
{
	u_transform_location = glGetUniformLocation(scene_four_obj2, "u_transform");
	glUseProgram(scene_four_obj2);
}
```

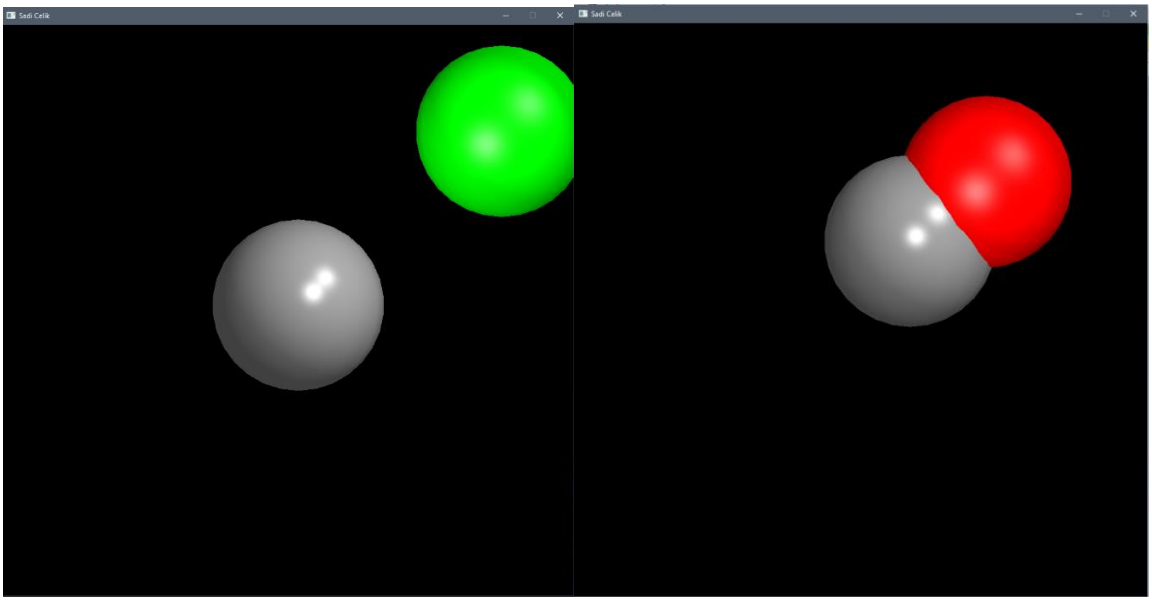

Figure 6: Scene 5

Sixth Scene:

In this scene, I tried to mimic the object in the demo. I created a new function as I mentioned in the first scene. (**`GenerateParametricShapeFrom2Dv2).`** I enabled transparent window setting with `glfwWindowHint(GLFW_TRANSPARENT_FRAMEBUFFER, GLFW_TRUE);` and set `glClearColor(0, 0, 0, 0);` to get the desired result.

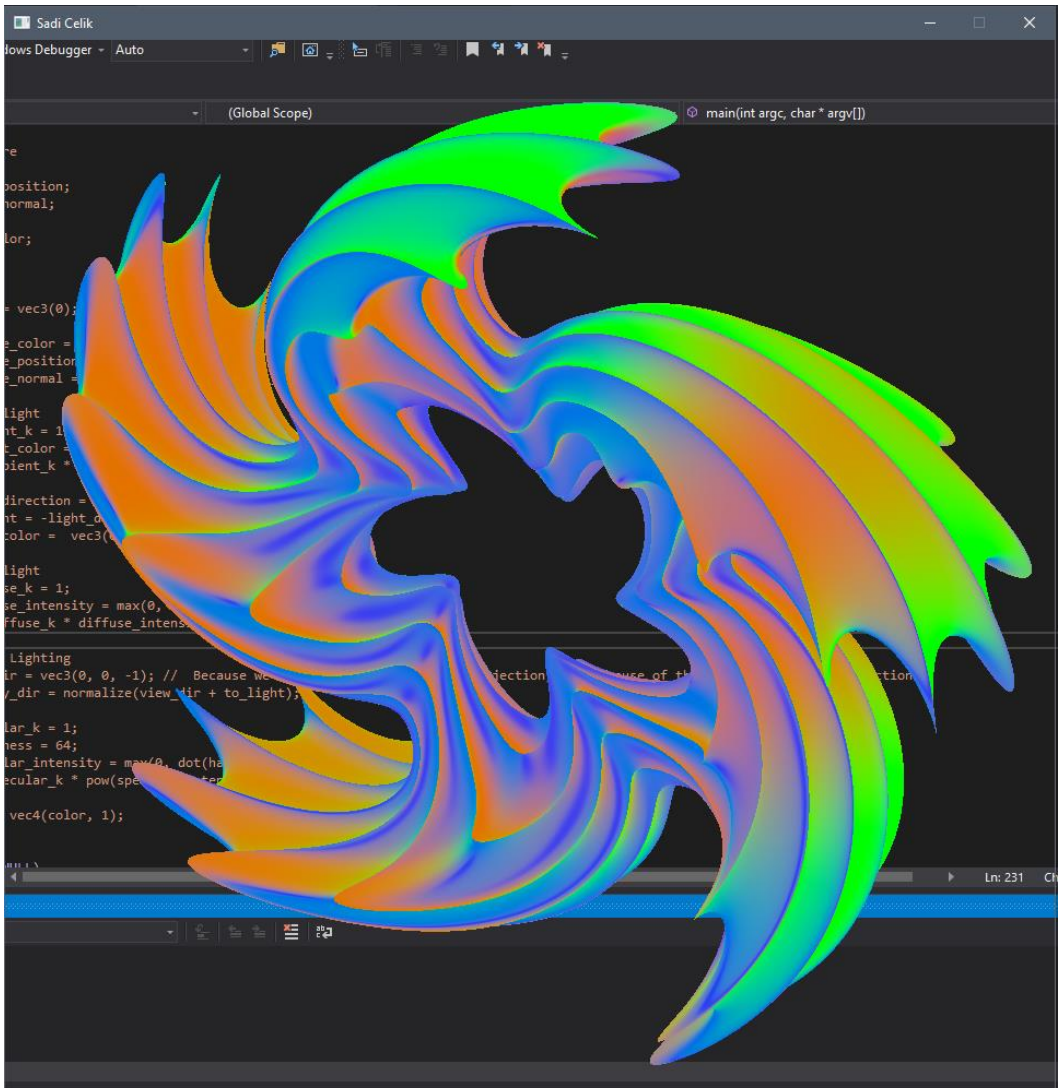

Figure 7: Scene 6

My program for the sixth scene (Only fragment shader):

```
GLuint scene_six = CreateProgramFromSources(vertex_shader_scene_ottffs,
	R"FRAGMENT(
	#version 330 core

	uniform vec2 u_mouse_position;

	in vec3 vertex_position;
	in vec3 vertex_normal;

	out vec4 out_color;

	void main()
	{
		vec3 color = vec3(0);

		vec3 surface_color = vec3(1, 1, 1);
		vec3 surface_position = vertex_position;
		vec3 surface_normal = normalize(vertex_normal);

		// Ambient light
		float ambient_k = 0.5;
		vec3 ambient_color = vec3(0, 1, 0);
		color += ambient_k * ambient_color * surface_color;

		vec3 light_direction = normalize(vec3(1, 1, 1));
		vec3 to_light = -light_direction;
		vec3 light_color =  vec3(0, 0, 1);

		// Diffuse light
		float diffuse_k = 1;
		float diffuse_intensity = max(0, dot(to_light, surface_normal));
		color += diffuse_k * diffuse_intensity * light_color * surface_color;

		// Specular Lighting
		vec3 view_dir = vec3(0, 0, -1);
		vec3 halfway_dir = normalize(view_dir + to_light);

		float specular_k = 1;
		float shininess = 64;
		float specular_intensity = max(0, dot(halfway_dir, surface_normal));
		color += specular_k * pow(specular_intensity, shininess) * light_color;

		// Light 2
		vec3 point_light_position = vec3(u_mouse_position, -1);
		vec3 point_light_color =  vec3(1, 0, 0);
		vec3 to_point_light = normalize(point_light_position - surface_position);

		// Diffuse light
		diffuse_k = 1;
		diffuse_intensity = max(0, dot(to_point_light, surface_normal));
		color += diffuse_k * diffuse_intensity * point_light_color *
surface_color;

		out_color = vec4(normalize(color), 1);
		}
		)FRAGMENT");
```