МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Компьютерная графика»

Тема: Реализация трехмерного объекта с использованием библиотеки OpenGL

Студент гр. 7383 _____ Бергалиев М.

Студент гр. 7383 _____ Власов Р.А.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург

2020

Цель работы

Разработать программу, реализующую представление разработанной трехмерной сцены с добавлением возможности использования различных видов источников света, используя предложенные функции библиотеки OpenGL (матрицы видового преобразования, проецирование) и язык GLSL.

Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя, замена типа проекции, управление преобразованиями, как с помощью мыши, так и с помощью диалоговых элементов.

Постановка задачи

Требования:

- Д.б. установлено изменение свойств источника света (интенсивность).
- Д.б. добавлены различные типы источников света.

Задание: Тостер

Ход работы

Для создания и использования шейдерной программы создадим класс Shader. `В конструкторе` происходит считывание исходных кодов шейдоров из файлов, компиляция вершинного и фрагментного шейдеров и связывание их в шейдерную программу. Чтобы использовать шейдерную программу, нужно использовать метод класса `Use`.

Для удобства определен класс камеры Camera. Этот класс содержит в себе информацию об ориентации в пространстве наблюдателя и угол обзора. Методы `get_view` и `get_projection` вычисляют матрицы вида и проекции соответственно.

Определены функции создания буфера графического конвейера, заполнение его данными и привязки атрибутов шейдеров к буферам:

```
void initBuffer(GLuint* buff, void* data, size_t size, GLuint type)
void initVAO(GLuint* VAO, void* vertices, int vsize, void* indices, int esize, std::vector<int> asize, int step, std::vector<int> offset)
```

Определена процедура загрузки текстуры:

```
void loadTexture(const char* file, GLuint* buff)
```

Вершинный шейдер:

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 normal;
layout (location = 2) in vec2 texcoords;
uniform mat4 rotation;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

out vec3 FragPos;
out vec3 Normal;
out vec2 TexCoords;
void main()
{
     gl_Position = projection * view * model * vec4(position, 1.0f);
    FragPos = vec3(model * vec4(position, 1.0f));
    Normal = vec3(rotation * vec4(normal, 1.0f));
    TexCoords = texcoords;
}
```

Фрагментный шейдер:

```
#version 330 core
in vec3 FragPos;
in vec3 Normal;
in vec2 TexCoords;
out vec4 color;
```

```
uniform bool inv_draw;
uniform vec3 viewPos;
struct Material {
    vec3 diffuse;
    vec3 specular;
    float shininess;
};
uniform Material material;
struct Light {
    vec3 position;
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
uniform Light light[3];
vec3 calculateLight(Light light, vec3 viewDir){
    // ambient
    vec3 ambient = light.ambient * material.diffuse;
    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(light.position - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = light.diffuse * diff * material.diffuse;
    // specular
    vec3 halfwayDir = normalize(lightDir + viewDir);
         float  spec  =  pow(max(dot(halfwayDir,  norm),  0.0),
material.shininess);
    vec3 specular = light.specular * (spec * material.specular);
    return ambient + diffuse + specular;
}
void main()
{
    vec3 viewDir = normalize(viewPos - FragPos);
    if(!inv_draw && dot(Normal, viewDir) < 0)
        discard;
```

```
    vec3 result = vec3(0.0f, 0.0f, 0.0f);
    for(int i=0; i<3; ++i)
        result += calculateLight(light[i], viewDir);
    color = vec4(result, 1.0f);
}
```

Фрагментный шейдер освещения:

```
#version 330 core
out vec4 color;
struct Material{
    vec3 diffuse;
};
uniform Material material;
void main()
{
    color = vec4(material.diffuse, 1.0f);
}
```

Изменение положения точки наблюдения осуществляется при помощи клавиш W, A, S, D. Включение (выключение) режима свободного обзора осуществляется при помощи клавиши C. При включенном режиме свободного обзора направление обзора регулируется при помощи мыши.

Тестирование

Тестирование проводилось в ОС Ubuntu 19.04.

На сцену добавлен источник направленного цвета, для которого можно изменять цвет, навпреление и интенсивность. На сцену добавлены 3 точечных источника света, для которых можно изменять их положение, цвет, интенсивность. Также на сцену добавлен прожектор, для которого можно изменять положение, навправление, цвет, интенсивность, угол концентрации света и угол рассеивания. Работа программы продемонстрирована на рис. 1-10.

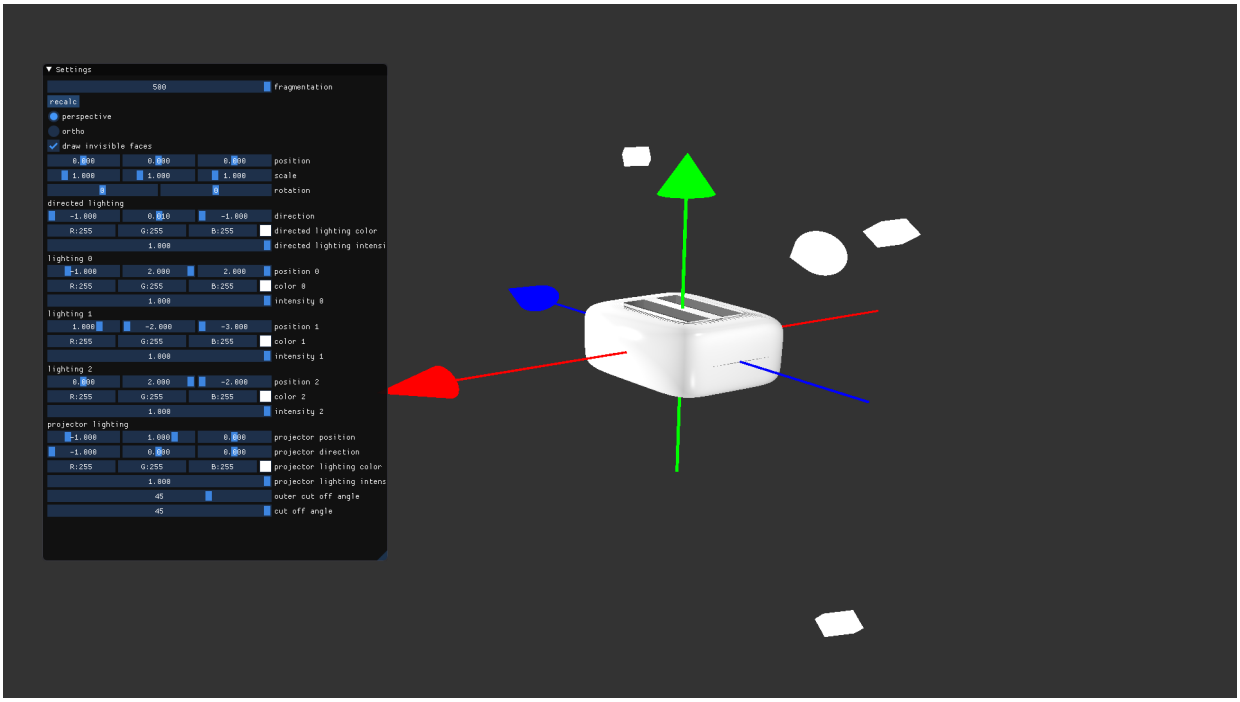

Рисунок 1 — Параметры источников света после запуска программы

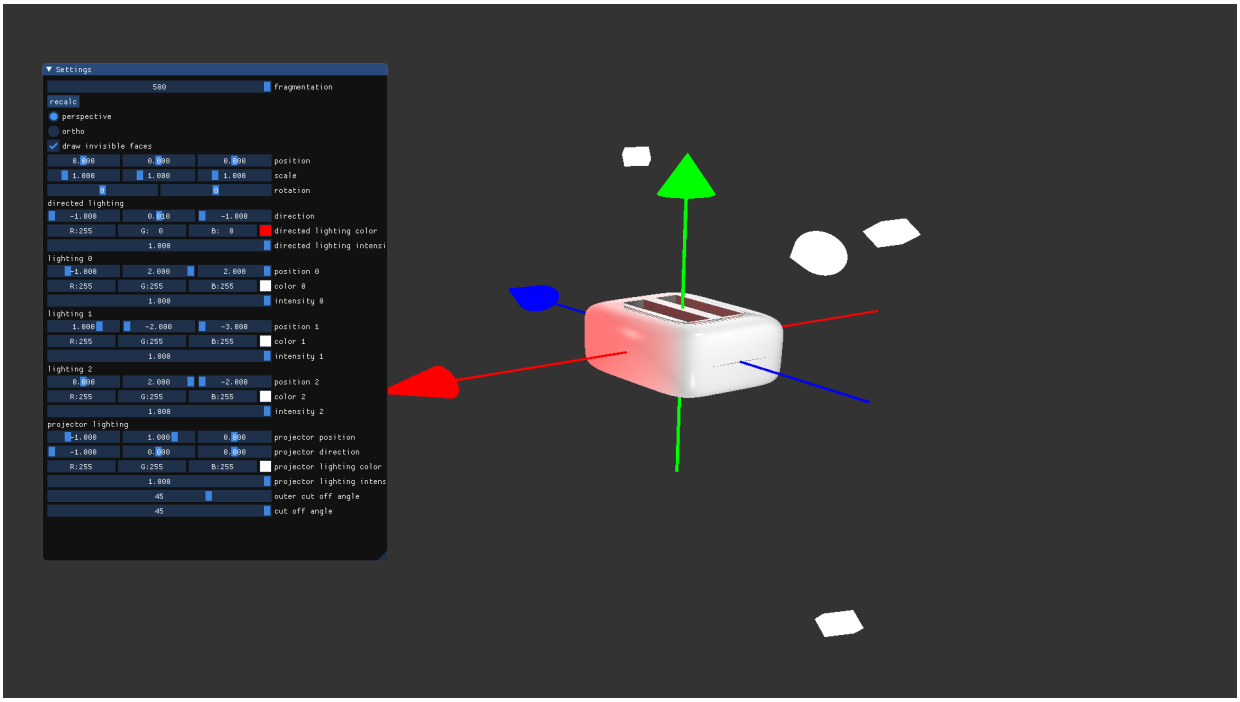

Рисунок 2 — Цвет направленного источника цвета изменен на красный

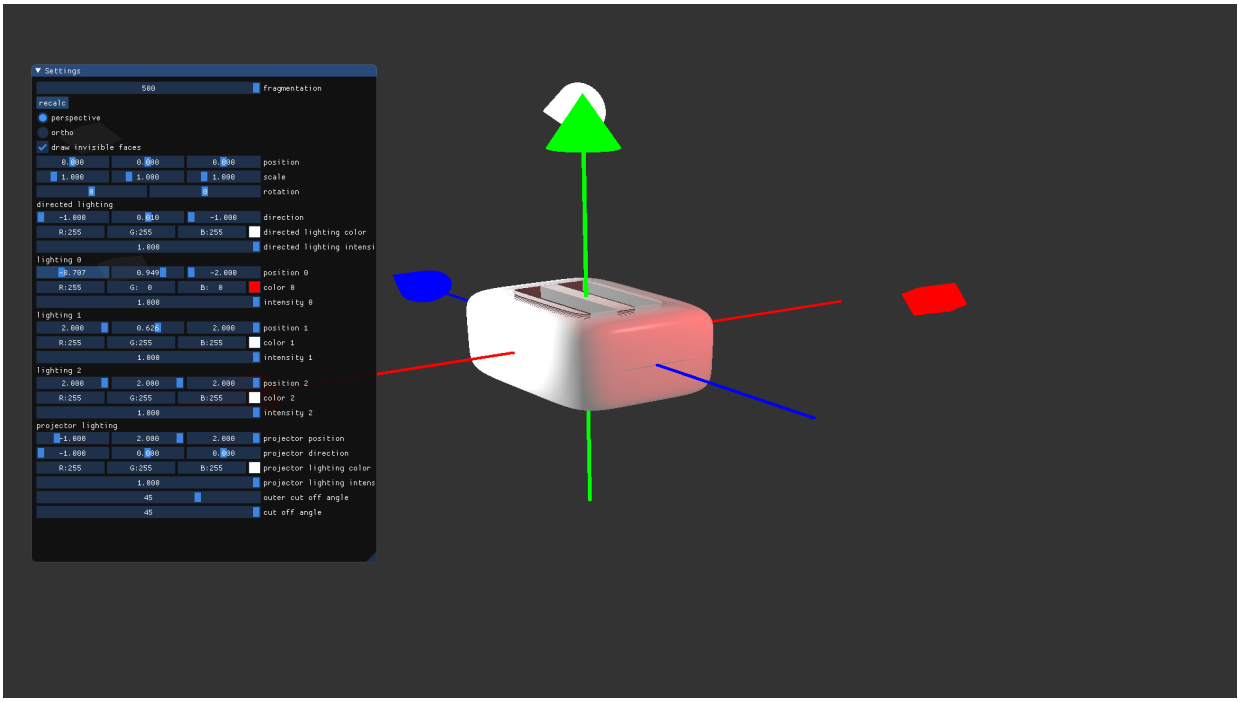

Рисунок 3 — Цвет одного из точечных источников установлен на красный

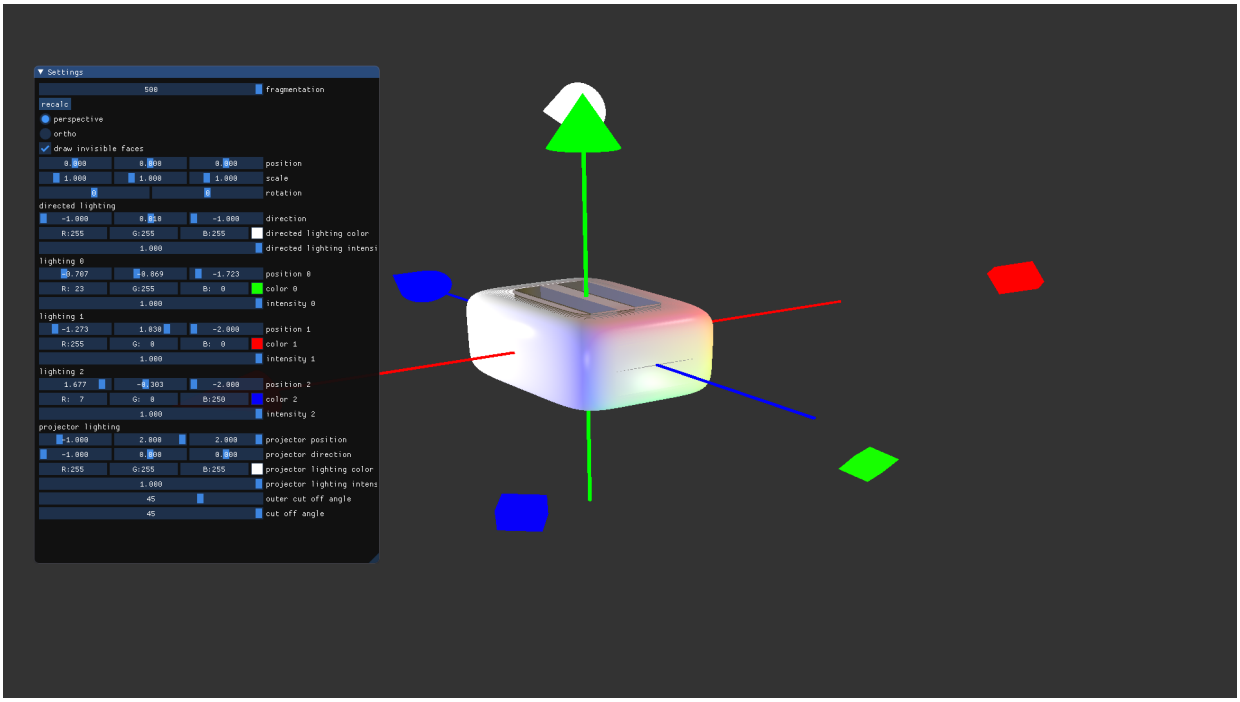

Рисунок 4 — Три точечных источника освещают грань разными цветами

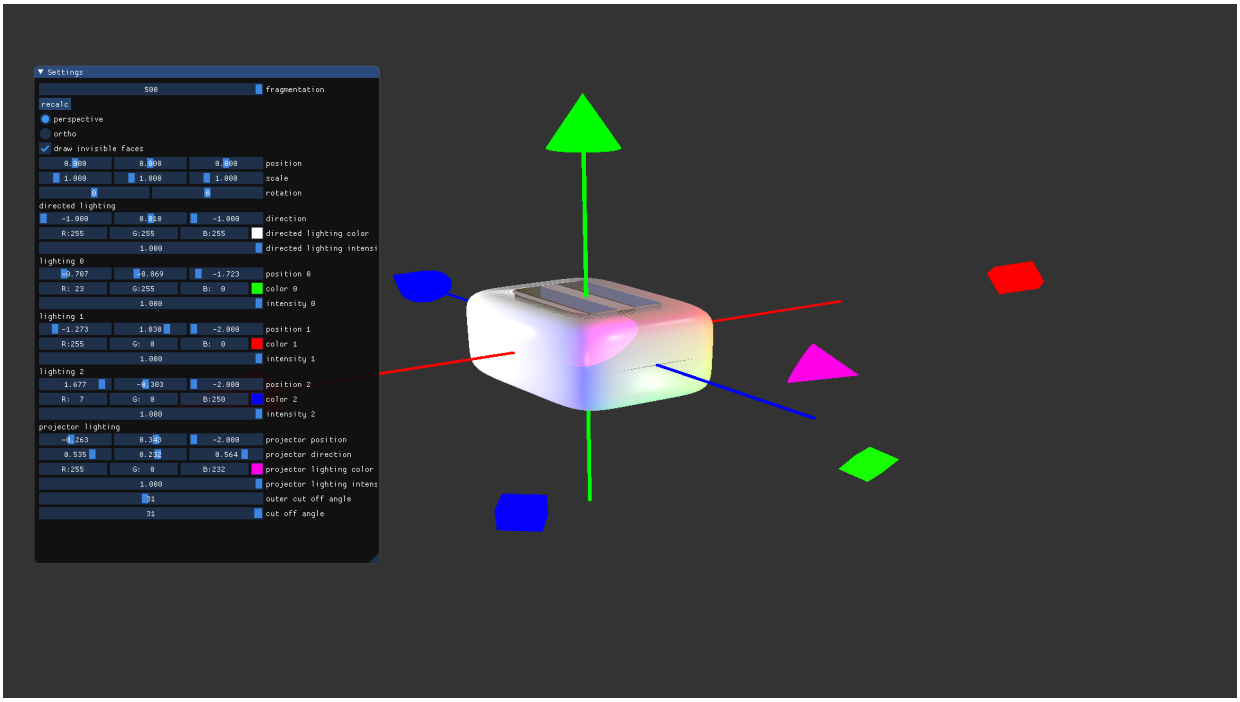

Рисунок 5 — Прожектор направлен на видимую грань

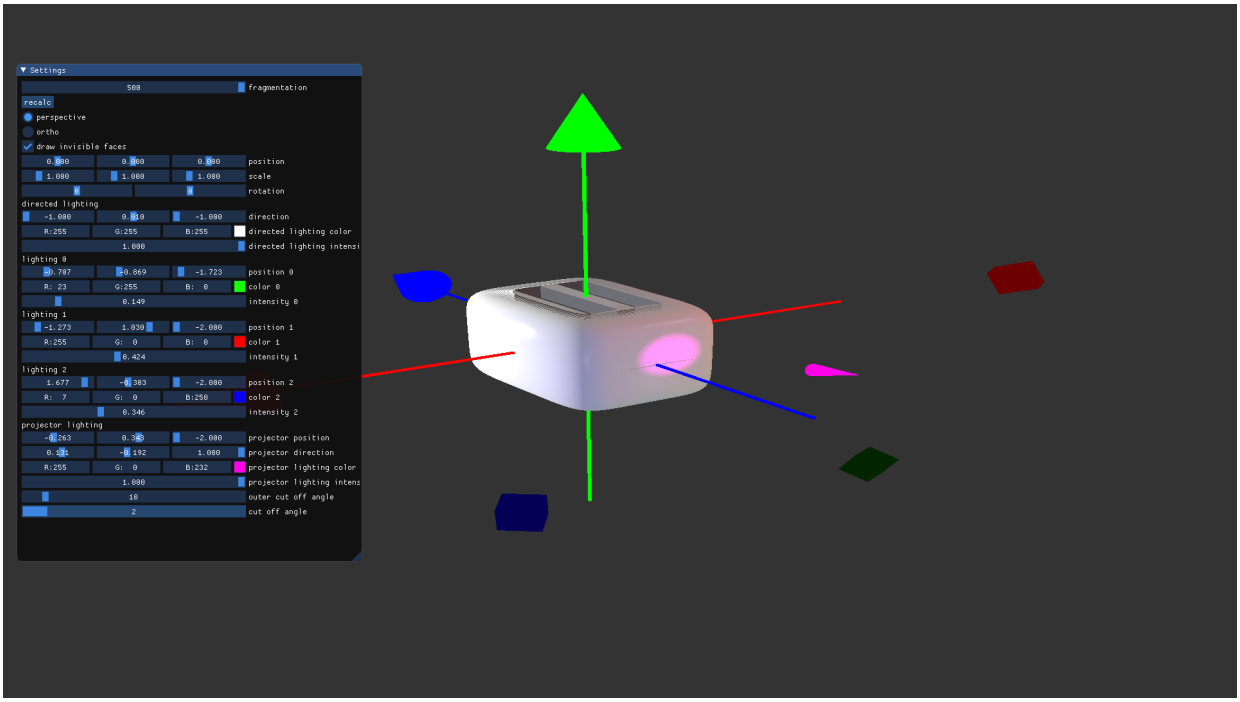

Рисунок 6 — Активировано рассеивание света от прожектора

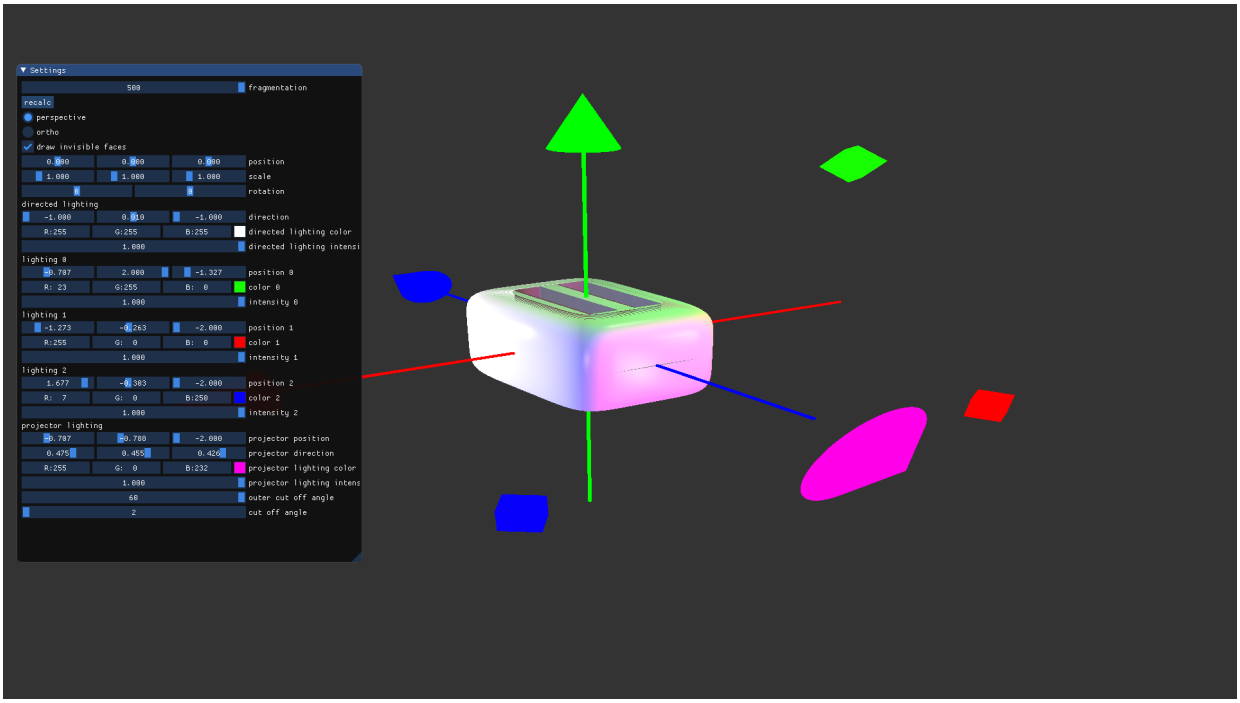

Рисунок 7 — Увеличен угол концентрации света от прожектора

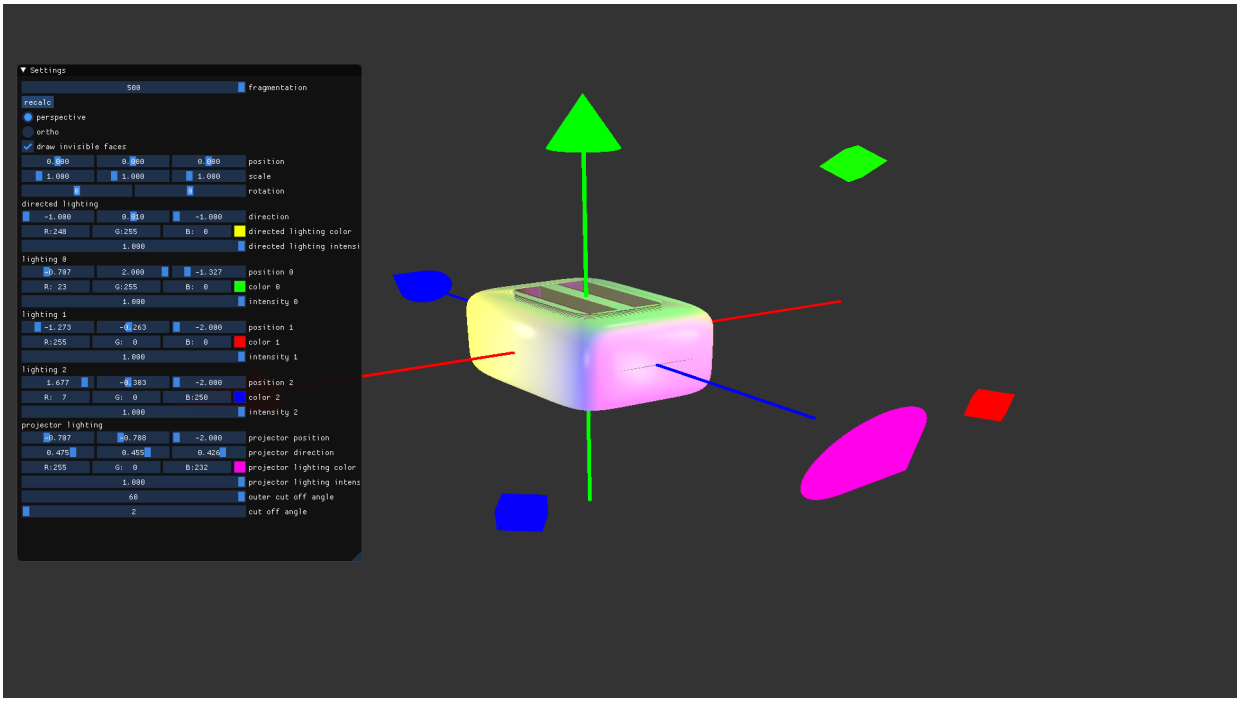

Рисунок 8 — Цвет направленного источника цвета изменен на желтый

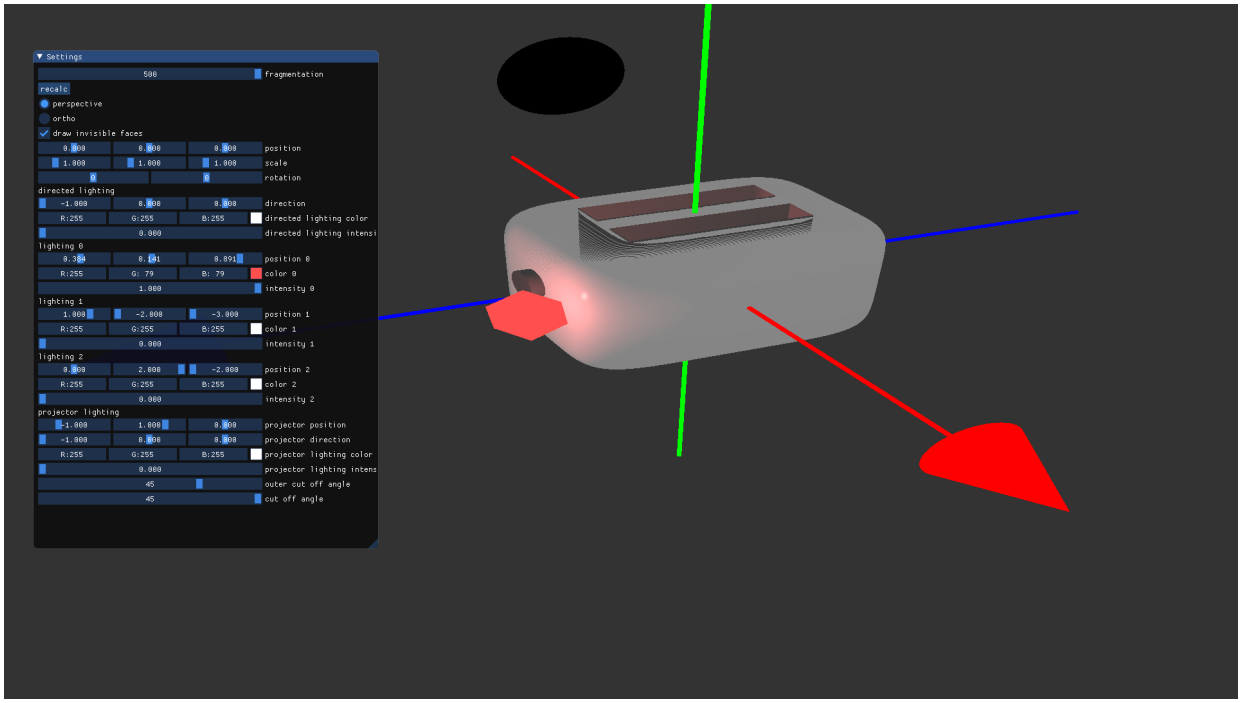

Рисунок 9 — Точечный источник света вблизи объекта

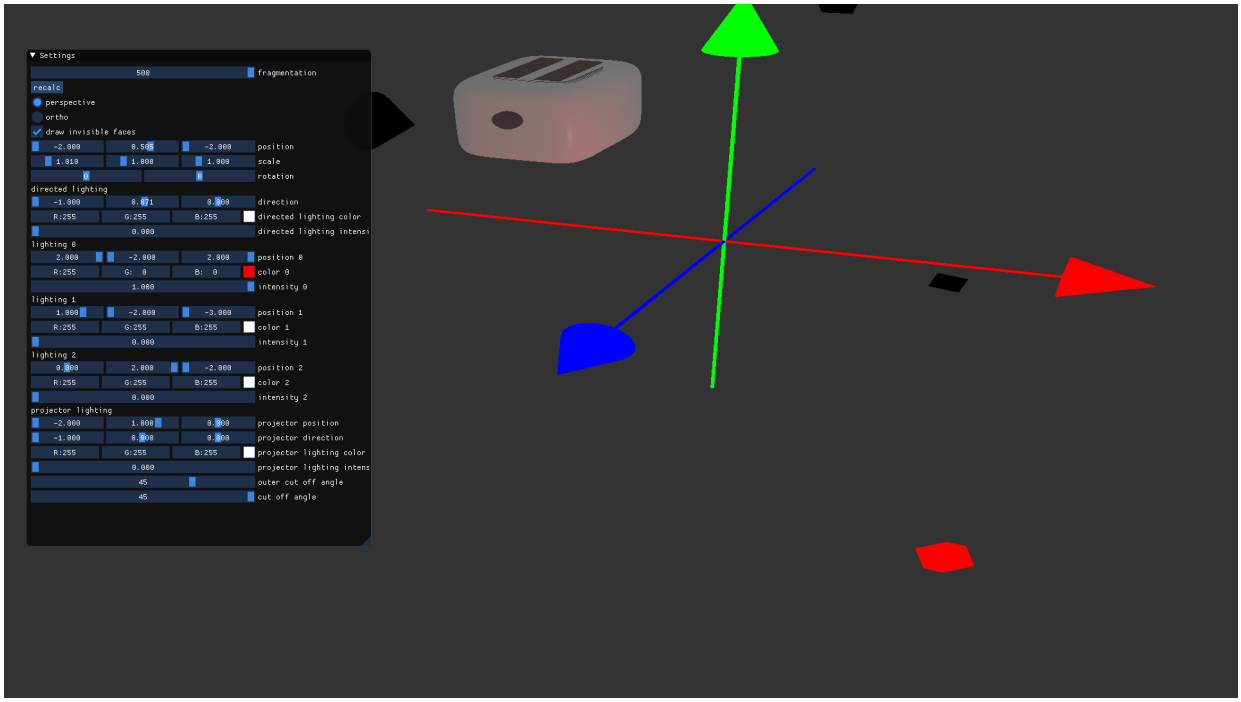

Рисунок 10 — Точечный источник света вдали от объекта

Выводы

Была разработана программа, реализующая представление трехмерного тостера с использованием функций библиотеки OpenGL и языка GLSL. При выполнении работы были приобретены навыки работы с графической библиотекой OpenGL, а также навыки использования языка шейдеров GLSL.