МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра вычислительной математики  
и прикладных информационных технологий

**Технологии программирования компьютерной графики**

Отчёт по Лабораторной работе №3

Направление 01.04.02 Прикладная математика и информатика

Профиль Математические основы  
и программирование компьютерной графики

|  |  |  |  |
| --- | --- | --- | --- |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_ |  | И. Б. Рахимов |

Воронеж 2023

# Содержание

Содержание 2

Постановка задачи 3

Результаты 4

Часть 1 4

Часть 2 14

Список используемых источников 15

Приложения 16

Приложение 1 Код шейдера 16

# Постановка задачи

Написать шейдер со skybox и несколькими цветными сферами.

# Результаты

## Часть 1

Для выполнения лабораторной работы буду использоваться язык GLSL и сайт <https://www.shadertoy.com>, можно написать небольшую программу на любом языке, для которого есть OpenGL биндинги, но так будет проще, так же можно использовать программы для написания шейдеров, например KodeLife.

Стандартная программа с сайта shadertoy:

void mainImage( out vec4 fragColor, in vec2 fragCoord )

{

// Normalized pixel coordinates (from 0 to 1)

vec2 uv = fragCoord/iResolution.xy;

// Time varying pixel color

vec3 col = 0.5 + 0.5\*cos(iTime+uv.xyx+vec3(0,2,4));

// Output to screen

fragColor = vec4(col,1.0);

}

Изменим немного шейдер:

void mainImage( out vec4 fragColor, in vec2 fragCoord )

{

vec2 uv = fragCoord/iResolution.xy;

fragColor = vec4(uv, 0.0f, 1.0f);

}

Результат шейдера можно увидеть на рисунке 1. Переменная fragCoord – координаты текущего пикселя, iResolution – разрешение текущего кадра, fragColor – результирующий цвет текущего пикселя. По рисунку 1 видно, что ось Y направлена вверх, а X вправо.



Рис. 1. Стандартные координаты пикселей

Для простоты перейдем из координат в (clip координаты) (рис. 2):

void mainImage( out vec4 fragColor, in vec2 fragCoord )

{

vec2 uv = fragCoord/iResolution.xy;

vec2 clip = uv \* 2.0f - 1.0f;

fragColor = vec4(clip, 0.0f, 1.0f);

}

Так как при таком переходе появляются отрицательные значения, а в RGBA числа должны быть положительными, они обрисовываются черным цветом.

Будем считать, что камера находится в начале координат и луч испускается в каждый пиксель на экране (рис. 3). Код для этого:

vec3 rayBegin = vec3(0.0f, 0.0f, 0.0f);

vec3 rayTarget = vec3(clip, 1.0f);

vec3 rayDir = normalize(rayTarget - rayBegin);



Рис. 2. Переход к координатам

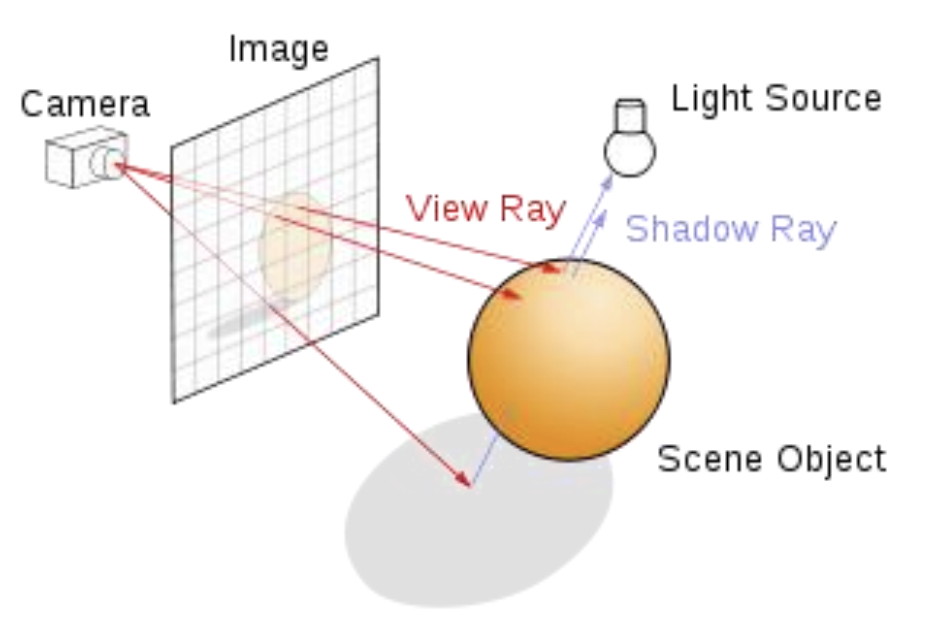


Рис. 3. Трассировка лучей

Далее необходимо написать функцию, которая будет возвращать информацию о визуализируемом мире. Для представления объектов в мире будут использоваться Функции расстояния со знаком (англ. signed distance function, SDF) при передаче координат точки в пространстве возвращают кратчайшее расстояние между этой точкой и некоторой поверхностью (пример рис. 4). Знак возвращаемого значения указывает, находится ли точка внутри этой поверхности или снаружи.

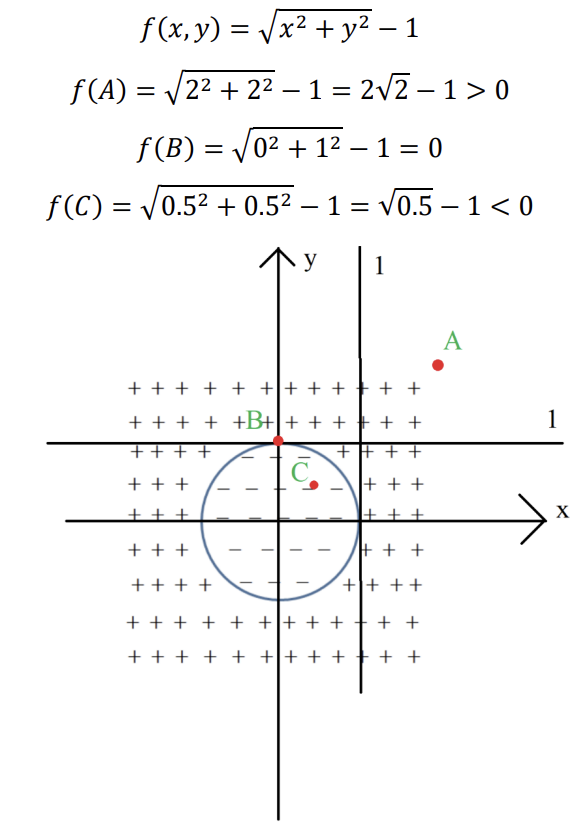


Рис. 4. Функция расстояния со знаком для двухмерной сферы

Пример функции мира с тремя сферами:

const float MAX\_DIST = 100.0f;

float sphereSdf(vec3 p, vec3 c, float r) {

return length(p - c) - r;

}

struct HitInfo {

float hit;

vec3 color;

};

HitInfo scene(vec3 p) {

HitInfo result;

result.hit = MAX\_DIST;

float hit;

hit = sphereSdf(p, vec3(0.0f, 2.0f, 5.0f), 1.0f);

if(hit < result.hit)

{

result.hit = hit;

result.color = vec3(1.0f, 0.0f, 0.0f);

}

hit = sphereSdf(p, vec3(0.0f, 0.0f, 5.0f), 1.0f);

if(hit < result.hit)

{

result.hit = hit;

result.color = vec3(0.0f, 1.0f, 0.0f);

}

hit = sphereSdf(p, vec3(0.0f, -2.0f, 5.0f), 1.0f);

if(hit < result.hit)

{

result.hit = hit;

result.color = vec3(0.0f, 0.0f, 1.0f);

}

return result;

}

Дополнительные функции для функции сцены (scene) была добавлена функция sphereSdf – функция расстояния со знаком для трёхмерной сферы. Функция мира возвращает структуру HitInfo которая хранит в себе два значения hit – расстояние до ближайшего объекта, color – цвет этого объекта. Дополнительно была введена переменная MAX\_DIST для хранения максимального расстояния визуализируемого мира.

Далее необходимо реализовать функцию испускания луча. В трассировке лучей, мы выбираем позицию для камеры, ставим перед ней сетку, отправляем лучи от камеры через каждую точку сетки, причем каждая точка сетки соответствует пикселю на выходном изображении (рис. 3).

Такой алгоритм имеет следующий недостаток, отправляя луч из камеры по направлению точки сетки возникает вопрос с каким шагом шагать по выбранному направлению. Выбрав слишком маленькое значение, можно потерять в эффективности алгоритма, выбрав слишком большое есть шанс перескочить через объект, который пересекает луч. Для устранения этого недостатка существует алгоритм сферической трассировки лучей, который делает максимально возможный шаг на каждой итерации алгоритма (рис. 5).

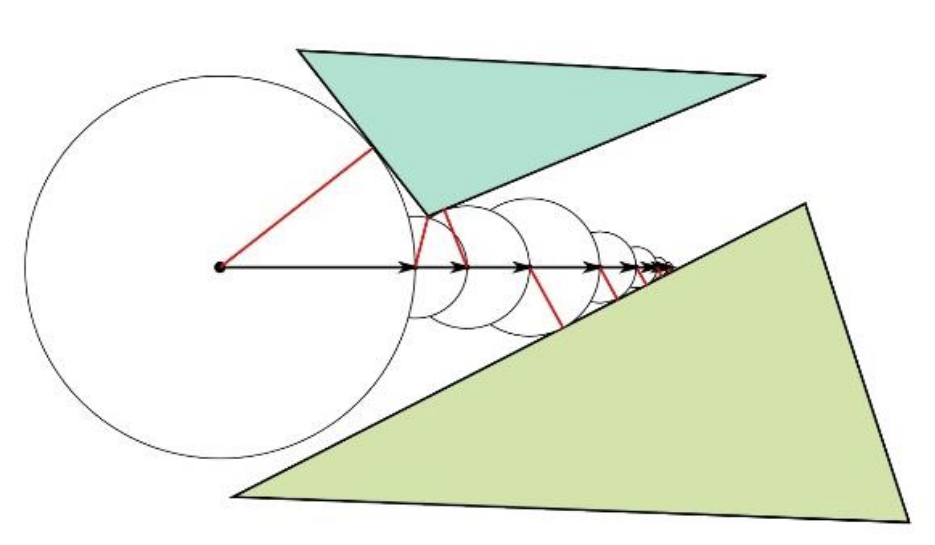


Рис. 5. Сферическая трассировка лучей

Реализация этого алгоритма представлена ниже:

const int MAX\_STEPS = 255;

const float EPSILON = 0.0001f;

HitInfo raycast(vec3 rayBegin, vec3 rayDir) {

HitInfo result;

float depth = 0.0f;

vec3 hitPos = vec3(0.0f);

for (int i = 0; i < MAX\_STEPS; ++i) {

result = scene(hitPos = rayBegin + depth \* rayDir);

if (result.hit < EPSILON) {

return result;

}

depth += result.hit;

if (depth > MAX\_DIST) {

return result;

}

}

result.hit = MAX\_DIST;

return result;

}

Основная функция raycast, дополнительных функций не требуется, дополнительные переменны MAX\_STEPS – максимальное количество шагов трассировки, EPSILON – точность при трассировке. Результирующие изображение представлено на шестом рисунке.

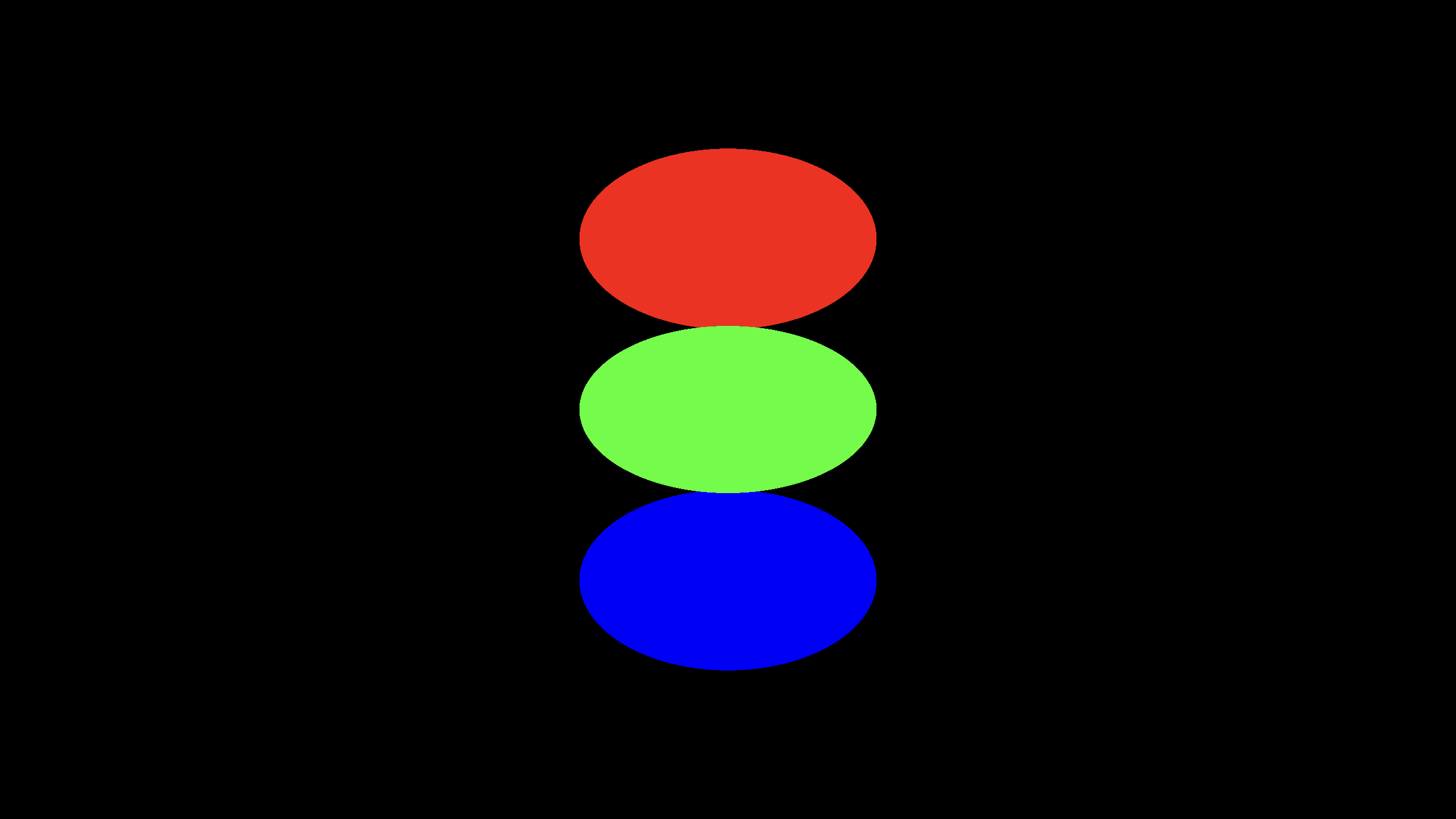


Рис. 6. Результат простейшей трассировки лучей

Исправим соотношение сторон, функция mainImage после использования новых функций и с исправленным соотношением сторон (рис. 7):

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {

vec2 uv = fragCoord/iResolution.xy;

vec2 clip = uv \* 2.0f - 1.0f;

float aspectRatio = iResolution.y / iResolution.x;

clip.y \*= aspectRatio;

vec3 rayBegin = vec3(0.0f, 0.0f, 0.0f);

vec3 rayTarget = vec3(clip, 1.0f);

vec3 rayDir = normalize(rayTarget - rayBegin);

HitInfo info = raycast(rayBegin, rayDir);

if(info.hit < EPSILON)

{

fragColor = vec4(info.color, 1.0);

}

else

{

fragColor = vec4(0.0);

}

}

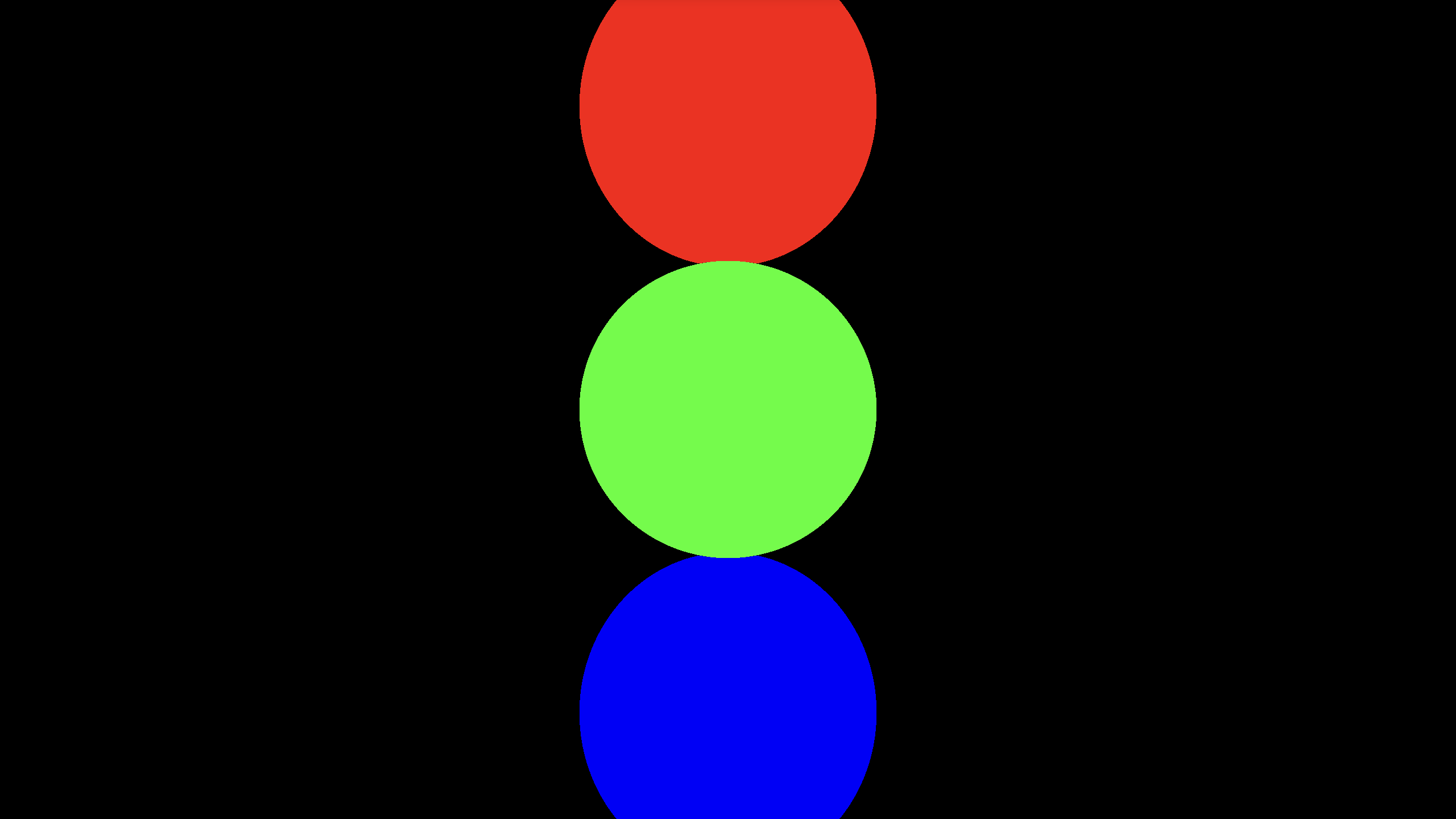


Рис. 7. Трассировка лучей с исправленным соотношением сторон

Добавим skybox (рис. 8), для проверки skybox добавил вращение камеры, но оно не обязательно для решения лабораторной работы:

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {

vec2 uv = fragCoord/iResolution.xy;

vec2 clip = uv \* 2.0f - 1.0f;

float aspectRatio = iResolution.y / iResolution.x;

clip.y \*= aspectRatio;

float s = sin(iTime);

float c = cos(iTime);

mat3 rotation = mat3(

c, 0.0, s,

0.0, 1.0, 0.0,

-s, 0.0, c

);

vec3 rayBegin = vec3(0.0f, 0.0f, 0.0f);

vec3 rayTarget = rotation \* vec3(clip, 1.0f);

vec3 rayDir = normalize(rayTarget - rayBegin);

HitInfo info = raycast(rayBegin, rayDir);

if(info.hit < EPSILON)

{

fragColor = vec4(info.color, 1.0);

}

else

{

fragColor = vec4(texture(iChannel0, rayDir).xyz, 1.0);

}

}

Для того чтобы на сайте <https://www.shadertoy.com> добавить skybox, необходимо добавить кубическую текстуру, это можно сделать через интерфейс добавления каналов для того, чтобы его открыть необходимо нажать на канал в нижней части интерфейса (рис. 8). Далее в открывшемся меню (рис. 9) выбрать либо Misc -> Cubemap, либо в разделе Cubemaps выбрать одну из существубщих на сайте кубических карт. Полученный результат представлен на рисунке 10.

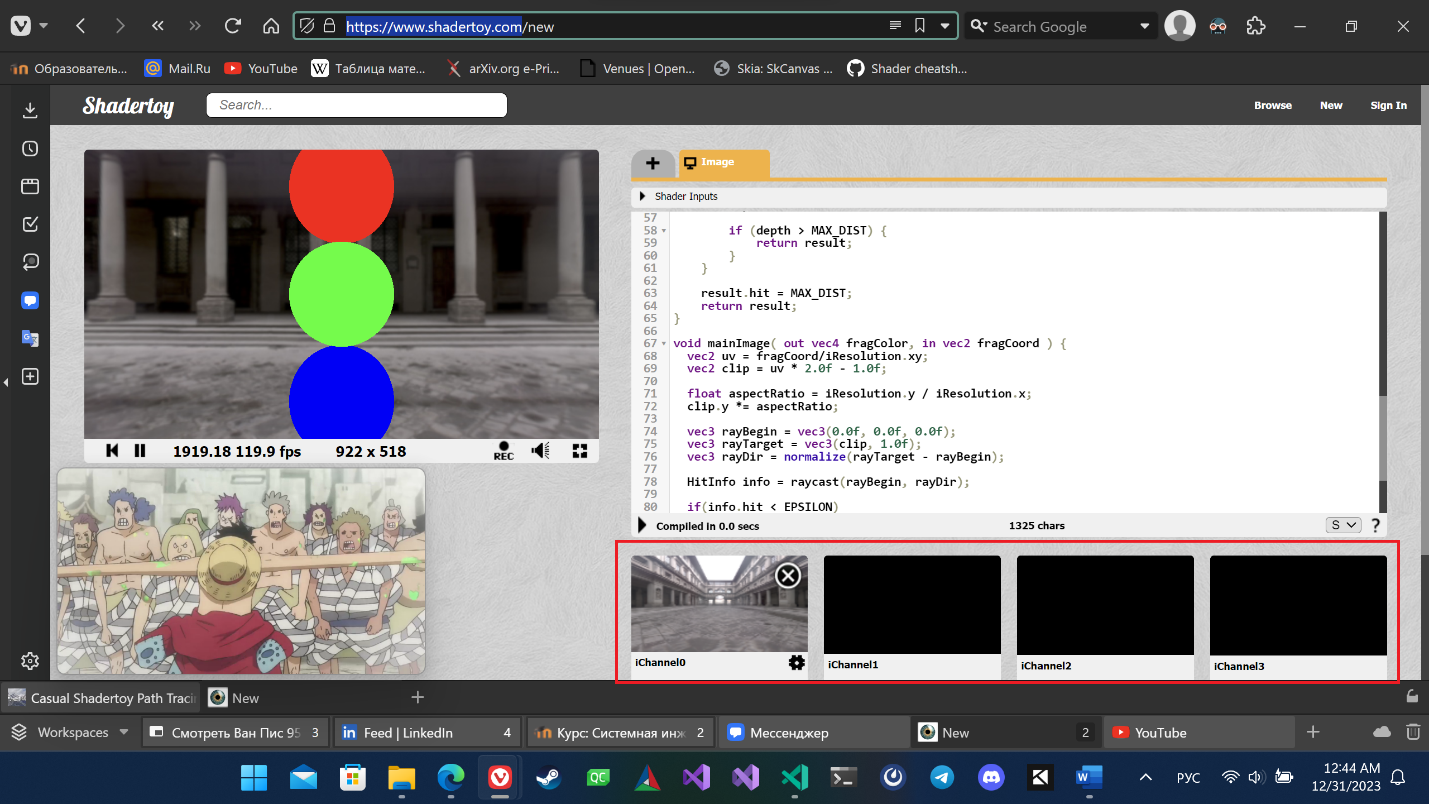


Рис. 8. Интерфейс выбора канала (выделено красным)

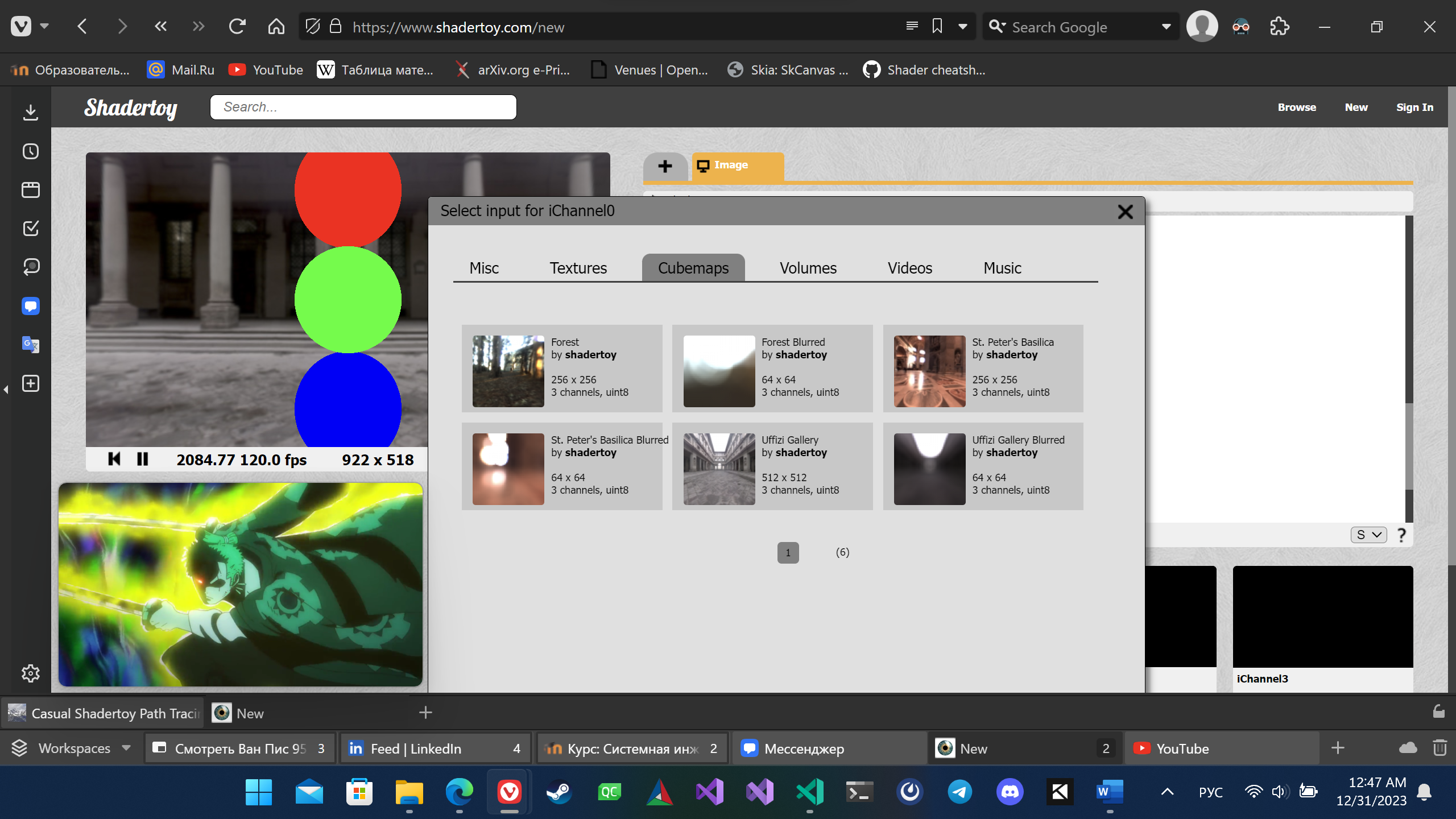


Рис. 9. Интерфейс выбора кубической карты

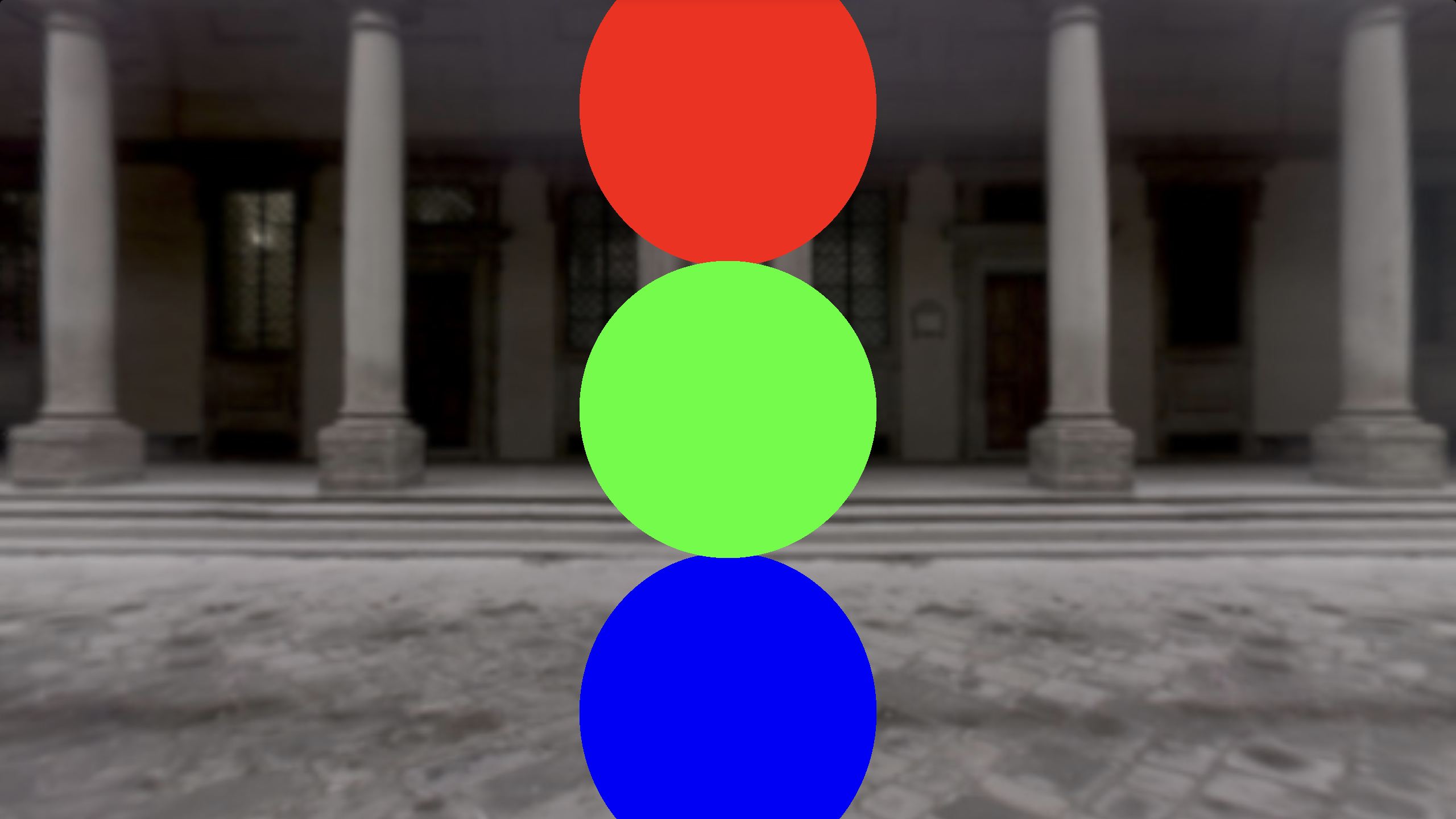


Рис. 10. Рендер после добавления кубической карты

## Часть 2

Добавим освещение, для этого необходимо знать нормаль объекта, на который попал луч. Для этого будет аппроксимировать нормаль как градиент функции scene:

vec3 normal(vec3 p) {

vec2 e = vec2(EPSILON, 0.);

return normalize(vec3(

scene(p+e.xyy).hit - scene(p-e.xyy).hit,

scene(p+e.yxy).hit - scene(p-e.xyy).hit,

scene(p+e.yyx).hit - scene(p-e.xyy).hit));

}

ыфафы

# Список используемых источников

1. <https://inspirnathan.com/posts/63-shadertoy-tutorial-part-16/>
2. <https://www.shadertoy.com/view/DltGWX>

# Приложения

## Приложение 1 Код шейдера

const float MAX\_DIST = 100.0f;

const int MAX\_STEPS = 255;

const float EPSILON = 0.0001f;

float sphereSdf(vec3 p, vec3 c, float r) {

return length(p - c) - r;

}

struct HitInfo {

float hit;

vec3 color;

};

HitInfo scene(vec3 p) {

HitInfo result;

result.hit = MAX\_DIST;

float hit;

hit = sphereSdf(p, vec3(0.0f, 2.0f, 5.0f), 1.0f);

if(hit < result.hit)

{

result.hit = hit;

result.color = vec3(1.0f, 0.0f, 0.0f);

}

hit = sphereSdf(p, vec3(0.0f, 0.0f, 5.0f), 1.0f);

if(hit < result.hit)

{

result.hit = hit;

result.color = vec3(0.0f, 1.0f, 0.0f);

}

hit = sphereSdf(p, vec3(0.0f, -2.0f, 5.0f), 1.0f);

if(hit < result.hit)

{

result.hit = hit;

result.color = vec3(0.0f, 0.0f, 1.0f);

}

return result;

}

HitInfo raycast(vec3 rayBegin, vec3 rayDir) {

HitInfo result;

float depth = 0.0f;

vec3 hitPos = vec3(0.0f);

for (int i = 0; i < MAX\_STEPS; ++i) {

result = scene(hitPos = rayBegin + depth \* rayDir);

if (result.hit < EPSILON) {

return result;

}

depth += result.hit;

if (depth > MAX\_DIST) {

return result;

}

}

result.hit = MAX\_DIST;

return result;

}

vec3 normal(vec3 p) {

vec2 e = vec2(EPSILON, 0.);

return normalize(vec3(

scene(p+e.xyy).hit - scene(p-e.xyy).hit,

scene(p+e.yxy).hit - scene(p-e.xyy).hit,

scene(p+e.yyx).hit - scene(p-e.xyy).hit));

}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {

vec2 uv = fragCoord/iResolution.xy;

vec2 clip = uv \* 2.0f - 1.0f;

float aspectRatio = iResolution.y / iResolution.x;

clip.y \*= aspectRatio;

vec3 rayBegin = vec3(0.0f, 0.0f, 0.0f);

vec3 rayTarget = vec3(clip, 1.0f);

vec3 rayDir = normalize(rayTarget - rayBegin);

HitInfo info = raycast(rayBegin, rayDir);

if(info.hit < EPSILON)

{

fragColor = vec4(info.color, 1.0);

}

else

{

fragColor = vec4(texture(iChannel0, rayDir).xyz, 1.0);

}

}