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

Отчет
по лабораторной работе №5
по дисциплине «КОМПЬЮТЕРНАЯ 3D-ГРАФИКА»
Тема: Карты теней

Студентка гр. 5381 _____ Буздина М.А.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург
2019

Цель работы.

Применить методы расчёта теней к вашей 4 лабораторной работе.

Задание.

В итоге выполнения пяти лабораторных работ полученный результата должен максимально приближаться к изображению, полученному в начале семестра рис. 1.

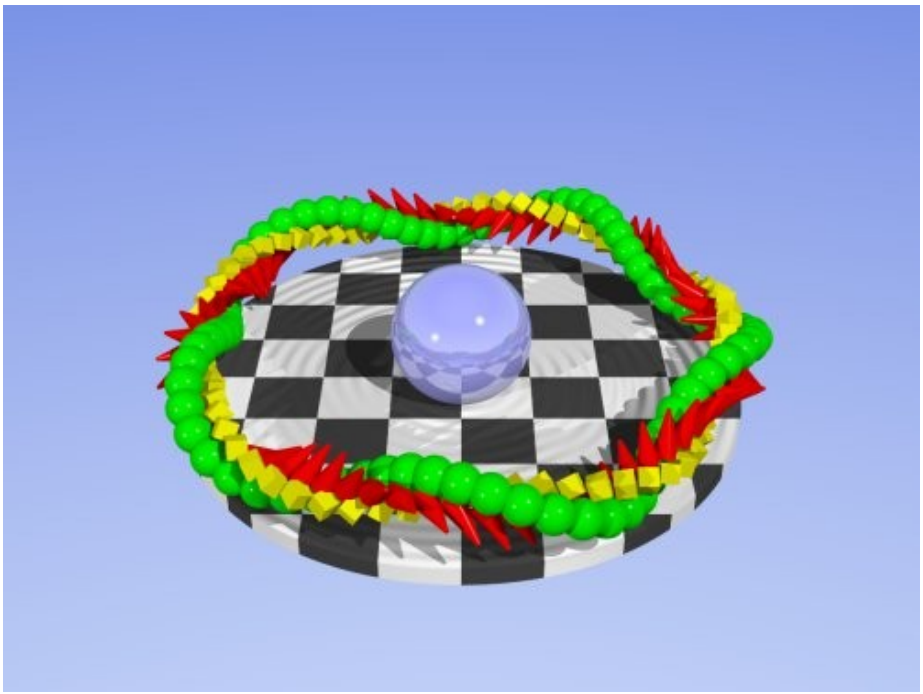

Рисунок 1 — Исходная сцена

Основные теоретические сведения.

Традиционный алгоритм «карты теней» состоит из следующих шагов. Обратите внимание, что мы визуализируем сцену дважды, как мы делали для выбора объекта. Первый рендеринг определяет, какие поверхности получают прямой свет. Второй рендеринг создает визуальное изображение, видимое пользователем.

1. Установите цель рендеринга для созданного программистом буфера кадров, состоящего из объектов текстуры.

2. Поместите «камеру» в источник света и визуализируйте сцену. Это помещает значение z ближайшей поверхности к источнику в буфер глубины (то есть, буфер z).
3. Измените цель рендеринга на обычный буфер кадров (который называется буфером рисования).
4. Рендеринг сцены из положения камеры и ориентации.
    1) Вершинный шейдер вычисляет расположение поверхности по отношению к как «источник света» и «камера». Оба местоположения помещаются в arying переменные и интерполируются по всей поверхности. Поэтому для каждого фрагмента мы знаем его местоположение как в «световом пространстве», так и в «пространстве камеры».
    2) Пиксельный шейдер использует карту текстуры от «источника света рендеринга» из шагов 1 и 2 ,чтобы определить, если пиксель находится в полном свете или тени.

Ход работы.

Отобразим сцену с точки зрения света, а затем с текстурой глубины.

Код текстуры глубины представлен ниже:

```
const depthTexture = gl.createTexture();
const depthTextureSize = 512;
gl.bindTexture(gl.TEXTURE_2D, depthTexture);
gl.texImage2D(
    gl.TEXTURE_2D,      // target
    0,                  // mip level
    gl.DEPTH_COMPONENT, // internal format
    depthTextureSize,   // width
    depthTextureSize,   // height
    0,                  // border
    gl.DEPTH_COMPONENT, // format
    gl.UNSIGNED_INT,    // type
    null);              // data
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
```

Рендеринг сцены с точки зрения света осуществляется с использованием шейдеров представленных ниже:

```
<script  id="color-vertex-shader" type="x-shader/x-vertex">
attribute vec4 a_position;
uniform mat4 u_projection;
uniform mat4 u_view;
uniform mat4 u_world;
void main() {
  // Multiply the position by the matrices.
  gl_Position = u_projection * u_view * u_world * a_position;
}
</script>

<script  id="color-fragment-shader" type="x-shader/x-fragment">
precision mediump float;
uniform vec4 u_color;
void main() {
  gl_FragColor = u_color;
}
</script>
```

Спроецируем текстуру карты теней обратно на сцену, чтобы получить глубину любых объектов, отбрасывающих тени, и сравнить ее с положением света на пиксель в пиксельном шейдере. Если мы находим свет ближе к камере, то освещаем пиксель. Если мы находим объект ближе к камере, мы затеняем пиксель. Шейдеры представленные ниже реализуют отображение фигур и карты теней:

```
<script  id="3d-vertex-shader" type="x-shader/x-vertex">
attribute vec4 a_position;
attribute vec2 a_texcoord;
attribute vec3 a_normal;
uniform vec3 u_lightWorldPosition;
uniform vec3 u_viewWorldPosition;
uniform mat4 u_projection;
uniform mat4 u_view;
uniform mat4 u_world;
uniform mat4 u_textureMatrix;
varying vec2 v_texcoord;
varying vec4 v_projectedTexcoord;
varying vec3 v_normal;
varying vec3 v_surfaceToLight;
varying vec3 v_surfaceToView;
void main() {
  // Multiply the position by the matrix.
  vec4 worldPosition = u_world * a_position;
  gl_Position = u_projection * u_view * worldPosition;
  // Pass the texture coord to the fragment shader.
  v_texcoord = a_texcoord;
  v_projectedTexcoord = u_textureMatrix * worldPosition;
  // orient the normals and pass to the fragment shader
```

```
          v_normal = mat3(u_world) * a_normal;
          // compute the world position of the surface
          vec3 surfaceWorldPosition = (u_world * a_position).xyz;
          // compute the vector of the surface to the light
          // and pass it to the fragment shader
          v_surfaceToLight = u_lightWorldPosition - surfaceWorldPosition;
          // compute the vector of the surface to the view/camera
          // and pass it to the fragment shader
          v_surfaceToView = u_viewWorldPosition - surfaceWorldPosition;
        }
        </script>
        <!-- fragment shader -->
        <script  id="3d-fragment-shader" type="x-shader/x-fragment">
        precision mediump float;
        // Passed in from the vertex shader.
        varying vec2 v_texcoord;
        varying vec4 v_projectedTexcoord;
        varying vec3 v_normal;
        varying vec3 v_surfaceToLight;
        varying vec3 v_surfaceToView;
        uniform vec4 u_colorMult;
        uniform sampler2D u_texture;
        uniform sampler2D u_projectedTexture;
        uniform float u_bias;
        uniform float u_shininess;
        uniform vec3 u_lightDirection;
        uniform float u_innerLimit;          // in dot space
        uniform float u_outerLimit;          // in dot space
        void main() {
          // because v_normal is a varying it's interpolated
          // so it will not be a unit vector. Normalizing it
          // will make it a unit vector again
          vec3 normal = normalize(v_normal);
          vec3 surfaceToLightDirection = normalize(v_surfaceToLight);
          vec3 surfaceToViewDirection = normalize(v_surfaceToView);
          vec3 halfVector = normalize(surfaceToLightDirection +
surfaceToViewDirection);
          float dotFromDirection = dot(surfaceToLightDirection,
                                       -u_lightDirection);
          float limitRange = u_innerLimit - u_outerLimit;
          float inLight = clamp((dotFromDirection - u_outerLimit) / limitRange,
0.0, 1.0);
          float light = inLight * dot(normal, surfaceToLightDirection);
          float specular = inLight * pow(dot(normal, halfVector), u_shininess);
          vec3 projectedTexcoord = v_projectedTexcoord.xyz /
v_projectedTexcoord.w;
          float currentDepth = projectedTexcoord.z + u_bias;
          bool inRange =
              projectedTexcoord.x >= 0.0 &&
              projectedTexcoord.x <= 1.0 &&
              projectedTexcoord.y >= 0.0 &&
              projectedTexcoord.y <= 1.0;
          // the 'r' channel has the depth values
          float projectedDepth = texture2D(u_projectedTexture,
projectedTexcoord.xy).r;
          float shadowLight = (inRange && projectedDepth <= currentDepth) ? 0.4
: 1.0;
          vec4 texColor = texture2D(u_texture, v_texcoord) * u_colorMult;
          gl_FragColor = vec4(
              texColor.rgb * light * shadowLight +
              specular * shadowLight,
```

```
            texColor.a);
    }
    </script>
```

Результат работы программы представлен на рис. 2 — 3.

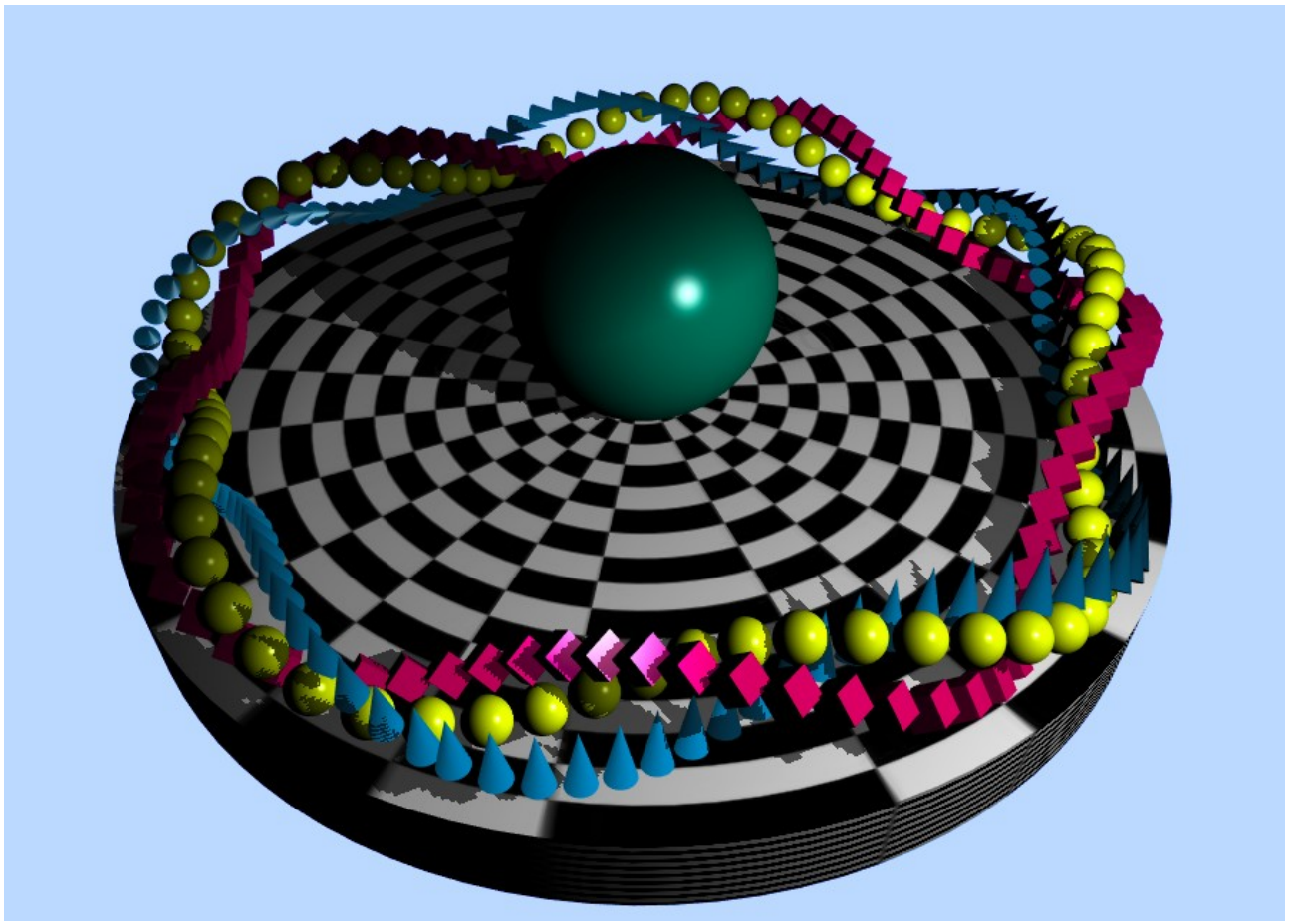

Рисунок 2 - Итоговая сцена

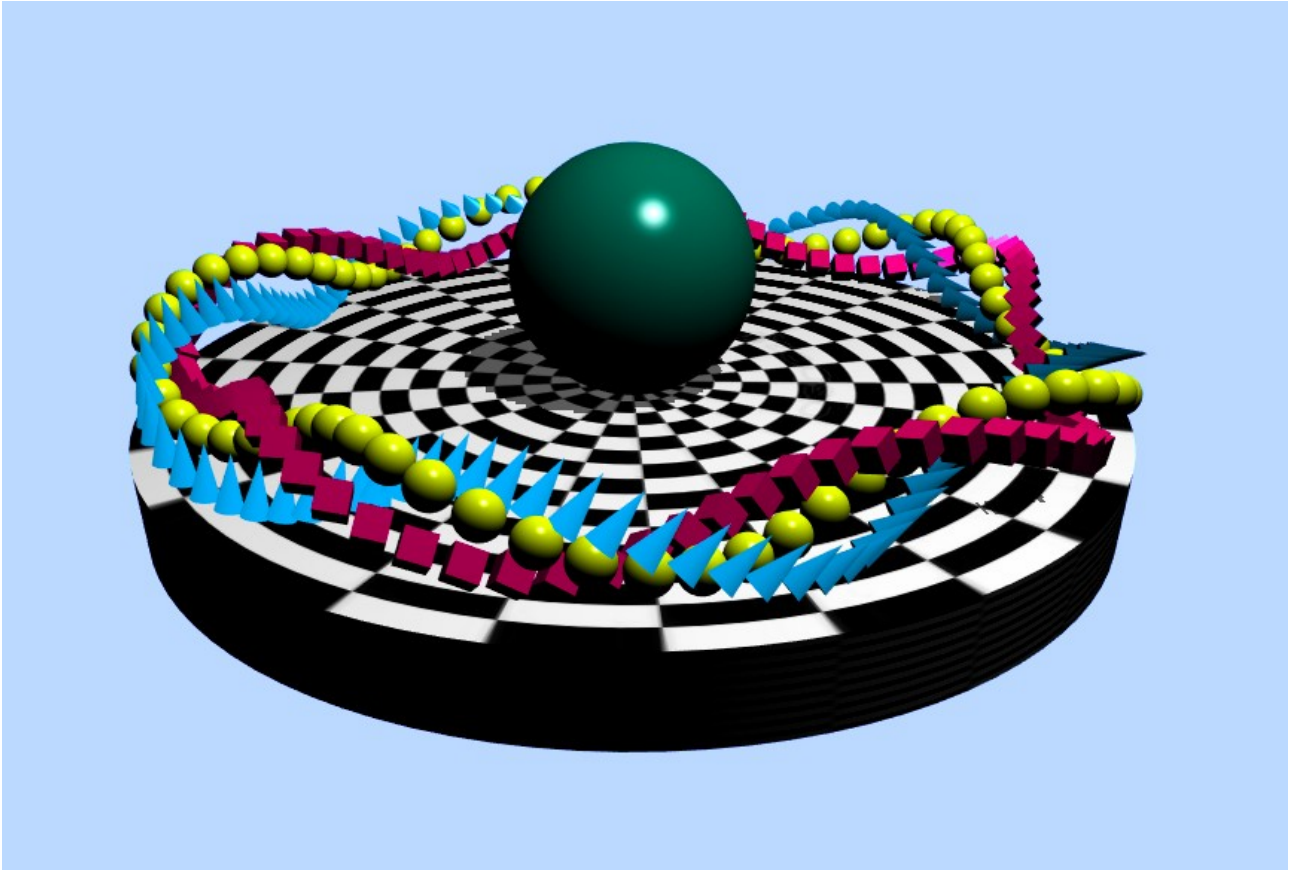

Рисунок 3 — Итоговая сцена

Выводы.

В ходе данной работы была отредактирована сцена и добавлены тени. Была получена финальная сцена.