Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и механики

Кафедра «Теоретическая механика»

КУРСОВОЙ ПРОЕКТ

Визуализация 3d сцены с помощью метода трассировки лучей

по дисциплине «Основы алгоритмизации и программирования»

Выполнил

студент гр.3630103/90003 Кедров А.А.

«___» _____ 2020 г.

Санкт-Петербург

2020

СОДЕРЖАНИЕ

Введение

Данная работа посвящена визуализации трехмерных объектов и световых явлений. Актуальность работы связана с необходимостью получения правдопобного изображения используя ограниченные численные данные об объекте. Цель курсовой работы заключается в исследовании математических методах описания световых явлений. Объект исследования – процесс использования языков программирования, а также математическая теория в области интегрирования и аналитической геометрии. Предмет исследования – программные инструменты языков JavaScript и HTML.

Постановка задачи

Реализовать следующие компоненты построения изображения с помощью метода трассировки лучей:

1. Создание проекции трехмерного объекта на плоскость
2. Решение уравнения рендеринга

Реализация

Описание объекта:

Объект задается набором граней. Этот способ описания позволяет использовать объекты различной сложности, но требует иногда больше вычислительных ресурсов в сравнении с точным математическим описанием некоторых объектов, например сфер.

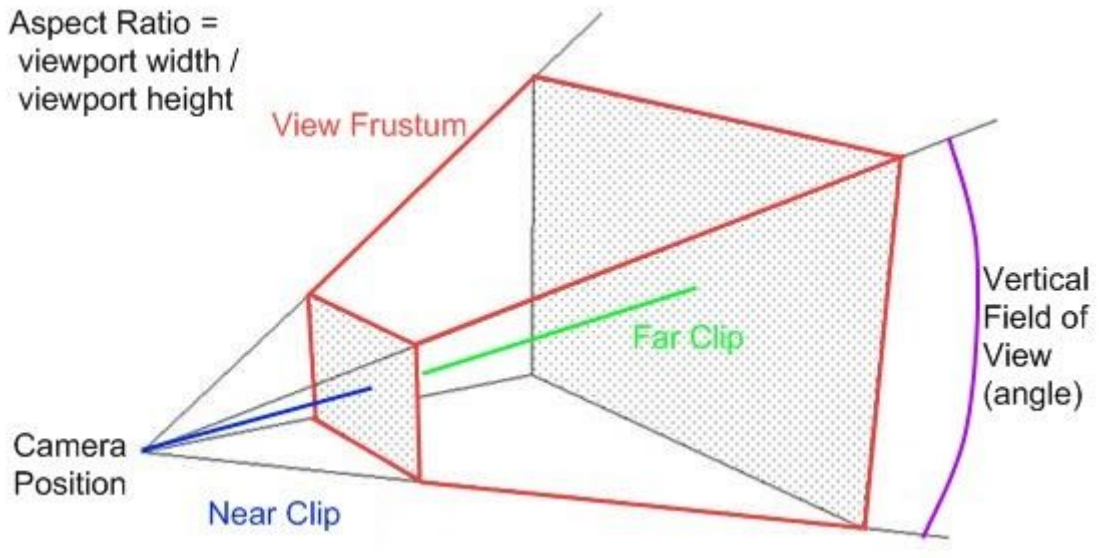
Проекция на плоскость:

Для создания проекции, для начала, необходимо выбрать плоскость (near clip). Для удобства плоскость задавалась положением камеры и расстоянием от камеры до этой плоскости. Затем на самой плоскости необходимо выбрать прямоугольник, который будет представлять конечное изображение. Для вычисления его границ используется отношение длины к ширине изображения (aspect ratio) и горизонтальный угол обзора (FOV). Итого, размеры прямоугольника в пространстве получаются по формулам:

$$width = 2 * distance * tg(\frac{FOV}{2})$$

$$height = \frac{width}{aspect}$$

Данные величины позволяют легко вычислить вершины прямоугольника в пространстве. Потом прямоугольник разбивается на равные части, соответствующих пикселям конечного изображения. Из каждой части берется произвольная точка (в данном случае левый верхний угол), и вычисляется направляющий вектор от камеры до точки. Направляющий вектор может быть использован для определения пересечения луча с гранью объекта и последующего вычисления цвета пикселя.



Поскольку каждая грань задаётся набором из трёх точек не составит труда, используя свойства смешанного произведения, определить расстояние от камеры до точки пересечения луча с плоскостью грани. Затем, воспользовавшись свойством о сумме углов треугольника, проверить факт попадания луча непосредственно во внутреннюю часть треугольника.

Свет:

Вычисление освещенности объекта и других световых явлений заключается в вычислении излучения, приходящегося на направление заданного луча, с помощью уравнения рендеринга

$$L_o = L_e + \int_{\Omega} BDRF(\omega_o, \omega_i) \cdot L_i \cdot (\omega_i \bullet n) d\omega_i$$

ω_o -- вектор “уходящего” луча света т.е. направленного к камере. Называется “уходящим”, потому что сначала свет испускается из источника, а затем, в результате нескольких отражений, попадает в камеру или на сетчатку глаза.

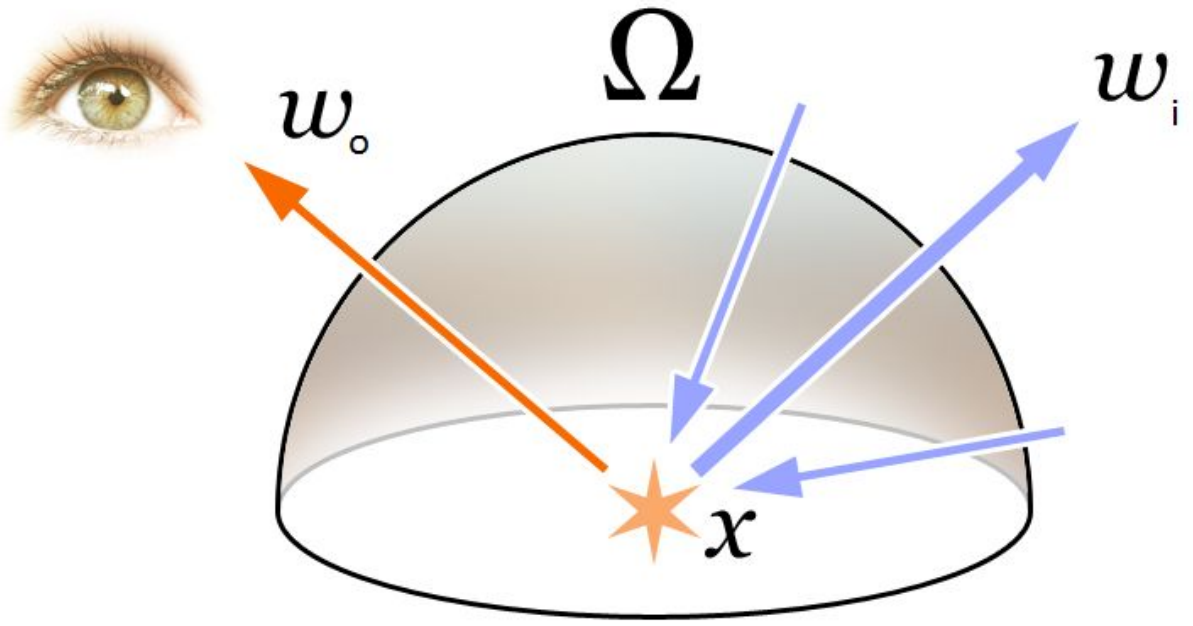
ω_i -- вектор, противоположно направленный “падающему” лучу т.е. лучу света попавший из источника на объект. Поскольку все объекты пространства могут излучать свет, это объясняет необходимость интегрирования по полусфере.

n -- вектор нормали к поверхности.

L_e -- излучение самого объекта, в простейшем случае свет, передающий цвет поверхности.

L_i -- излучение, падающее на объект.

$BDRF(\omega_o, \omega_i)$ -- bidirectional reflectance distribution function, функция, задающая вероятность отражения луча ω_i по направлению ω_o.



В данной работе формула была представлена в виде суммы нескольких компонент излучения:

$$L_o = L_{specular} + L_{diffuse} = L_{specular} + L_{ambient} + L_{direct}$$

$L_{diffuse}$ -- рассеянный свет, т.е. свет, образующий привычный для человека цвет поверхности.

$L_{specular}$ -- отраженный свет, задающий зеркальность/металличность поверхности.

$L_{ambient}$ -- свет окружения, предполагаемый свет, полученный от источников, о которых почти ничего не известно, кроме как то, что он приблизительно одинаков во всех точках рассматриваемого пространства, например Солнце.

L_{direct} -- прямое освещение или свет, полученный от источников.

Использование физической величины излучения неудобно, поэтому его можно заменить на значение цвета в RGB палитре.

Рассеянный свет:
Практически зависит только от освещенности поверхности, поэтому выражается, как

$$L_{diffuse} = L_{ambient} + L_{direct}$$

Практически применялась линейная формула зависимости:

$$diffuse_i = color_i \cdot (l_{ambient,i} + l_{direct,i})(1 - gloss)$$

$l_{ambient}$ -- контрибуция света окружения в освещённость.

l_{direct} -- контрибуция света от источников в освещённость.

$gloss$ -- глянцевость поверхности или отношение кол-ва отраженного света к преломленному

Отражённый свет:
Отраженный свет рассчитывается с помощью уравнения рендеринга, за тем исключением, что не учитывается излучаемый свет (его роль играет рассеянный свет) и не берутся в расчет строго определенные источники света т.е. только другие объекты.

В качестве BDRF была использована Cook–Torrance модель.

$$BDRF = \frac{DFG}{\pi(\omega_i \bullet n)(\omega_o \bullet n)}$$

Где D, F, G - функции определяющие следующие величины:

D -- вероятность, что нормаль к микронеровностям на поверхности в данной точке будет направлена таким образом, что ω_i отразится по ω_o.

F -- “эффект” Френеля, описывающий абсолютное отражение луча по поверхности.

G -- вероятность, что падающий луч не отразится в сам объект.

Интегрирование производилось с помощью метода Монте-Карло по формуле

$$I = \frac{1}{a} \sum_{k=1}^{a} 2\pi(\omega_{ok} \bullet n) \cdot BDRF \cdot L_k = \frac{2}{a} \sum_{k=1}^{a} \frac{DFG}{\omega_{ik} \bullet n} \cdot L_k$$

Свет окружения:

Свет окружения можно задать в виде константы, которая будет прибавляться к итоговому результату, но есть и более правдоподобная техника - Ambient Occlusion. Она основывается на наблюдательном факте, что поверхности, которые оставляют маленькое пространство между собой (углубления, углы комнаты) кажутся более затемненными. Этот факт позволяет получить более точную модель:

$$L_a = \int_{\Omega} hit(\omega_i)\ (\omega_i \bullet n) \cdot L_{a0} d\omega_i$$

$hit(\omega_i)$ -- функция, которая равна 0, если луч пересекает другие объекты, и 1, если не встречает никаких препятствий на своем пути.

Прямой свет:

Для прямого освещения использовался метод Photon mapping. Его идея заключается в том, что из каждого источника выпускается большое число частиц в различных направлениях. Ударяясь об объекты, частицы случайным образом могут отразиться или поглотиться (эта разновидность метода называется Russian Roulette). Положения всех частиц можно использовать, как направления ω_i для уравнения рендеринга и применять в методе Монте-Карло.

Примеры кода JavaScript

Функция, которая по направлению и положения камеры определяет итоговый цвет - рекурсивна т.к. используется при расчёте отраженного света.

```
trace_recursive(tracer, bounce_num) {
    var color = new Vector(0, 0, 0);

    // Search for intersections with objects
    tracer.trace_scene(this);

    if (tracer.trace !== undefined && tracer.trace !== null) {
      // Ambient
      var ambient_eval = new AmbientEval(this, tracer);
      var ambient =
                      this.is_ao_enabled  ?  ambient_eval.eval()  :
this.ambient_color;
      var material = tracer.object.material;

      // Shadows
      if (this.is_shadows_enabled && this.is_shadowed(tracer)) {
        return material.color.mul(ambient);
      }

      // Direct light
      var radience = this.is_lightning_enabled ?
          this.photonmap.calcualte_light(tracer) :
          Vector.zero();

                  if   (tracer.object.material.gloss   ==   0   ||
!this.is_reflections_enabled) {
        // Diffuse light
                                                        color       =
color.add(material.color.mul(radience.add(ambient)));
      } else {
        // Diffuse light
        color = color.add(material.color.mul(radience.add(ambient))
                          .scale(1 - material.gloss))
        // Specular light
        if (bounce_num < this.max_bounces) {
```

```
          var reflection = new ReflectionEval(this, tracer);
          var reflection_color = reflection.eval(bounce_num);

                                                              color      =
color.add(reflection_color.scale(material.gloss));
        }
      }
    } else {
      // Sky
           if (this.sky_texture !== null && this.sky_texture !==
undefined) {
        var sphere_hit =
                          tracer.trace_to_sphere(this.sky_sphere_pos,
this.sky_sphere_radius);

        if (sphere_hit) {
          var n = sphere_hit.sub(this.sky_sphere_pos).normalize();
          var u = Math.atan2(n.x, n.y) / (2 * Math.PI) + 0.5;
          var v = 0.5 - n.z * 0.5;

          color = color.add(this.sky_texture.get_color(u, v));
        }
      } else {
        color = color.add(new Vector(0.5, 0.5, 0.8));
      }
    }

    return color;
  }
```

Функция, рассчитывающая расстояние, которое должен пройти луч до плоскости, а затем проверяющая факт пересечения с гранью.

```
trace_to_face(v1, v2, v3, b_force) {
    var vv1 = v1.sub(this.start);
    var vv2 = v2.sub(this.start);
    var vv3 = v3.sub(this.start);

    var d1 =
           (vv2.y - vv1.y) * (vv3.z - vv1.z) - (vv2.z - vv1.z) *
(vv3.y - vv1.y);
    var d2 =
```

```
            (vv2.x - vv1.x) * (vv3.z - vv1.z) - (vv2.z - vv1.z) *
(vv3.x - vv1.x);
    var d3 =
            (vv2.x - vv1.x) * (vv3.y - vv1.y) - (vv2.y - vv1.y) *
(vv3.x - vv1.x);

    var t =
             d1 * this.direction.x - d2 * this.direction.y + d3 *
this.direction.z;

    if (t == NaN || Math.abs(t) < 0.00001) {
      return false;
    }

    var scale = (d1 * vv1.x - d2 * vv1.y + d3 * vv1.z) / t;

    var trace_vector = this.direction.scale(scale);

    if (b_force) {
      return trace_vector;
    }

    if (scale < 0.00001) {
      return false;
    }

            if   (!Vector.is_vector_inside_triangle(vv1,   vv2,   vv3,
trace_vector)) {
      return false;
    }

    return trace_vector.add(this.start);
  }
```

В ней применяется функция, которая проверяет, находится ли точка внутри треугольника

```
  static is_vector_inside_triangle(vec1, vec2, vec3, target) {
    // Observing vectors representing triangle sides
    var ab = vec2.sub(vec1);
    var ac = vec3.sub(vec1);
```

```
    var bc = vec3.sub(vec2);

    // M is the end of the target
    var am = target.sub(vec1);
    var bm = target.sub(vec2);

    var ab_length = ab.length();
    var ac_length = ac.length();
    var bc_length = bc.length();
    var am_length = am.length();
    var bm_length = bm.length();

    var angle_bac_cos = ab.dot(ac) / (ab_length * ac_length);
    var angle_abc_cos = -ab.dot(bc) / (ab_length * bc_length);

    // Compare angles
    if (ab.dot(am) / (ab_length * am_length) < angle_bac_cos) {
      return false;
    }

    if (ac.dot(am) / (ac_length * am_length) < angle_bac_cos) {
      return false;
    }

    if (-ab.dot(bm) / (ab_length * bm_length) < angle_abc_cos) {
      return false;
    }

    if (bc.dot(bm) / (bc_length * bm_length) < angle_abc_cos) {
      return false;
    }

    return true;
  }
```

Нахождение интеграла на примере отраженного света:

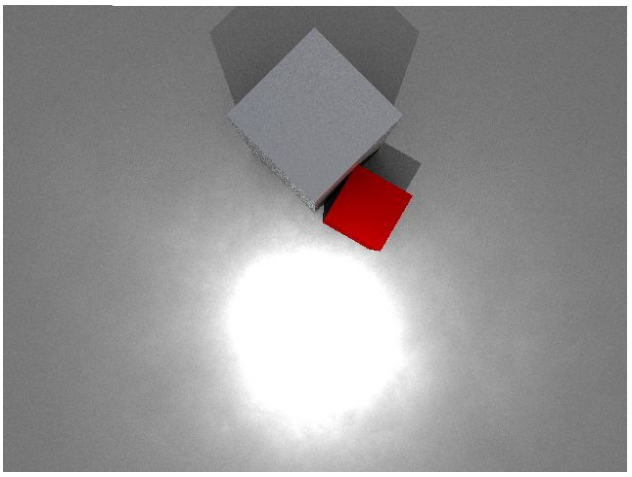
```
  eval(bounce) {
    var ndotwo = this.normal.dot(this.wo);
    var color = Vector.zero();

    for (var i = 0; i < this.scene.max_samples; i++) {
```

```
      var ndotwi = (i + 1) / (this.scene.max_samples + 1);

                  var   sample   =   Sampler.sample_hemisphere(ndotwi,
Math.random());
      this.wi = this.quat.rotate_vector(sample);
      this.half_dir = this.wi.add(this.wo).normalize();
      var ndotwi = this.normal.dot(this.wi);

      var pdf = ndotwi +
                     this.fresnel_schlick()  *  this.d_beckmann()  *
this.g_smith_beckmann() /
                ndotwo;

      var new_tracer = new Tracer(this.tracer.trace, this.wi);
      new_tracer.ignore_obj = this.tracer.object;
      new_tracer.ignore_index = this.tracer.start_index;

      color = color.add(
                    this.scene.trace_recursive(new_tracer,  bounce  +
1).scale(pdf));
    }

    color = color.scale(1 / this.scene.max_samples);
    return color;
  }
```

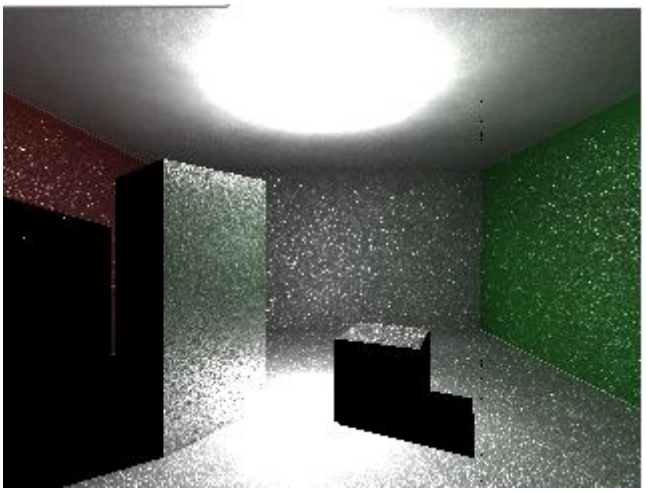
Результаты

При помощи скриптового языка программирования был реализован инструмент для создания изображения трехмерного пространства, а также пользовательский интерфейс, осуществляющий ввод данных.

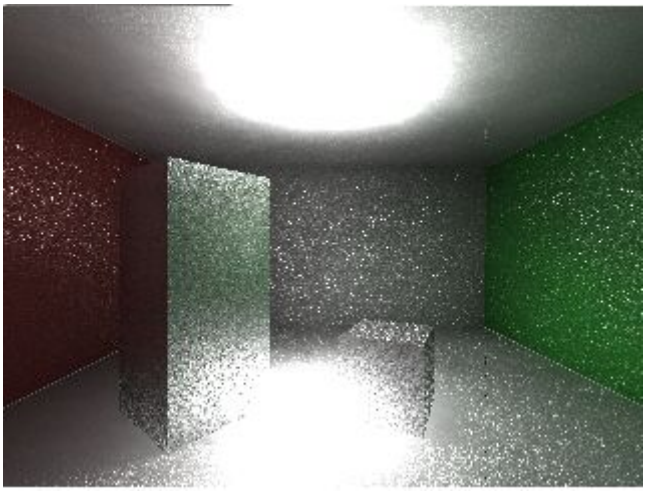
Ниже представлены примеры результата работы программы.



Сцена с двумя кубами - стальным и обычным.



Cornell Box. Закрытая сцена состоящая из отражающих объектов.



Cornell Box с отключенными сценами, чтобы продемонстрировать рассеивание “фотонов”

Модель monkey из Blender, чтобы продемонстрировать возможность работы с более сложными моделями

Вывод

В ходе работы была выполнена поставленная задача. Были реализованы основные возможности физически-корректного рендера. Также был получен интерфейс, который позволяет быстро добавлять новый функционал.

Заключение

Немаловажно отметить, что программу можно улучшить, добавив прозрачные/полупрозрачные поверхности, а также больше видов источников освещения.