**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №6**

**по дисциплине «Компьютерная графика»**

Тема: **Формирования реалистических изображений с использованием простых моделей освещения одним или двумя точечными источниками**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7307 |  | Торопов В.А. |
| Преподаватель |  | Матвеева И.В. |

**Цель работы**

Реализовать алгоритм трассировки лучей при формировании сложной сцены.

**Основные теоретические положения**

Трассировка лучей — один из методов геометрической оптики — исследование оптических систем путём отслеживания взаимодействия отдельных лучей с поверхностями.

Наилучшим способом представления лучей для нашей цели будет использование параметрического уравнения. Мы можем выразить любую точку P луча как

где t — произвольное действительное число > 1, О – положение камеры, V – точка просмотра.  
  
Обозначим (V−O), то есть направление луча, как  ; тогда уравнение примет простой вид

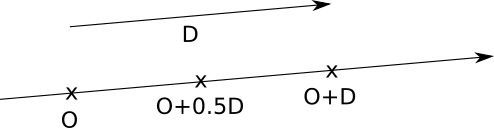
Если мы начнём из начальной точки и продвинемся на какое-нибудь кратное направления луча, то всегда будем двигаться вдоль луча:

Рисунок 1 – Движение луча

Сферу можно представить следующим уравнением:

Точки пересечения со сферой можно найти следующим образом:

, где

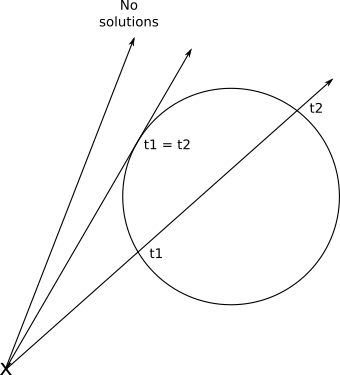


Рисунок 2 – Пересечение луча со сферой

Сформулируем уравнение диффузного отражения. Оно используется для вычисления полного количества света, полученного точкой P с нормалью в сцене с окружающим освещением яркостью и n точечных или направленных источников света с яркостью и световыми векторами или известными (для направленных источников), или вычисленными для P (для точечных источников):

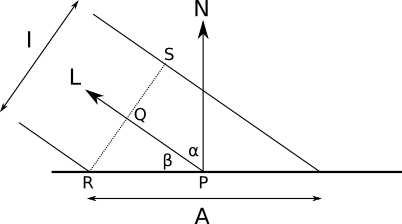
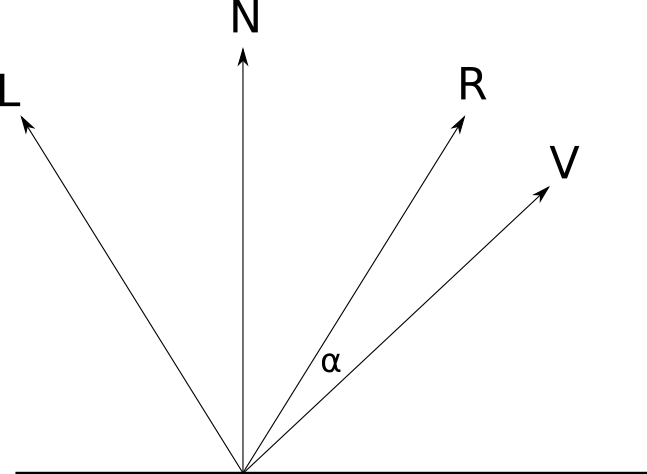
Исключением является < 0, так как освещённость не может быть отрицательной

Рисунок 3 – Моделирование диффузного отражения

Нормаль сферы можно найти с помощью уравнения:

Уравнение зеркального отражения:

, где



– коэффициент падающего луча, – вектор обзора, – нормаль, – падающий луч света, – отражённый свет, – коэффициент отражённого света, – показатель отражения.

Рисунок 4 – Моделирование зеркального отражения

Проверка теней осуществляется следующим способом: «Если между источником света и точкой есть другой объект, то освещение от этого источника учитывать не надо». Проверка осуществляется на подобии испускания луча света из источника к сфере, только в другом направлении. Следовательно, можно представить луч следующим уравнением:

Нас беспокоит только один случай, когда t = 0. При данном параметре получается так, что объект отбрасывает тень на самого себя. Чтобы этого избежать примем t > ε = 0,001.

Отражение работает также как тени, с одним исключением, что направление луча, которое было = становится = . Для того, чтобы не произошёл эффект «бесконечного коридора» нужно ввести предел рекурсии r.

**Математическое обоснование (Python)**

Функция нахождения пересечения луча со сферой:

def ClosestIntersection(O, D, t\_min, t\_max):  
 *"""  
 Finding intersections of rays with a sphere* ***:param*** *O: Starting point of the beam* ***:param*** *D: Ray coordinates* ***:param*** *t\_min: minimum parameter* ***:param*** *t\_max: maximum parameter* ***:return****: closest\_sphere and closest t parameter  
 """* closest\_t = np.inf  
 closest\_sphere = None  
 global spheres  
  
 DD = np.dot(D, D)  
  
 for sphere in spheres:  
 t1, t2 = IntersectRaySphere(O, D, DD, sphere)  
 if t1 > t\_min and t1 < t\_max and t1 < closest\_t:  
 closest\_t = t1  
 closest\_sphere = sphere  
 if t2 > t\_min and t2 < t\_max and t2 < closest\_t:  
 closest\_t = t2  
 closest\_sphere = sphere  
  
 return closest\_sphere, closest\_t

Функция нахождения отражённого луча:

def ReflectRay(R, N):  
 *"""  
 counting the reflection beam* ***:param*** *R: ray* ***:param*** *N: normal line* ***:return****: reflection beam  
 """* return 2\*N\*np.dot(N, R) – R

Функция решения квадратного уравнения:

def IntersectRaySphere(O, D, DD, sphere):  
 *"""  
 Solves the square equation* ***:param*** *O: Starting point of the beam* ***:param*** *D: Ray coordinates* ***:param*** *DD: dot(D, D)* ***:param*** *sphere: sphere* ***:return****: Intersection  
 """* global center\_window  
  
 rr = sphere.get\_rr()  
 OC = sphere.get\_oc()  
 k3 = sphere.get\_ococ() - rr  
  
  
 k1 = DD  
 k2 = 2\*np.dot(OC, D)  
  
 discriminant = k2\*k2 - 4\*k1\*k3  
 if discriminant < 0:  
 return np.inf, np.inf  
  
 t1 = (-k2 + np.sqrt(discriminant)) / (2\*k1)  
 t2 = (-k2 - np.sqrt(discriminant)) / (2\*k1)  
 return t1, t2

Функция подсчёта освещённости:

def ComputeLighing(P, N, V, s):  
 *"""  
 Calculation of lighting* ***:param*** *P: to calculate the intersection* ***:param*** *N: calculating the sphere normal at the intersection point* ***:param*** *V: overview vector* ***:param*** *s: the reflection rate* ***:return****: the lighting coefficient  
 """* global lights  
 i = 0.0  
 for light in lights:  
 if light.get\_elements()['type'] == 'ambient':  
 i += light.get\_elements()['intensity']  
 else:  
 L = []  
 if light.get\_elements()['type'] == 'point':  
 L = (light.get\_elements()['position'] - P).copy()  
 t\_max = 1  
 else:  
 L = light.get\_elements()['direction'].copy()  
 t\_max = np.inf  
  
 #shadow  
 shadow\_sphere, shadow\_t = ClosestIntersection\_P(P, L, 0.001, t\_max)  
 if shadow\_sphere != None:  
 continue  
  
 #diffuse  
 NL = np.dot(N, L)  
 if NL > 0:  
 i += light.get\_elements()['intensity']\*NL/(np.linalg.norm(N)\*np.linalg.norm(L))  
  
 #mirror  
 if s != -1:  
 R = 2\*N\*np.dot(N, L) - L  
 RV = np.dot(R, V)  
 if RV > 0:  
 i += light.get\_elements()['intensity']\*np.power(RV/(np.linalg.norm(R)\*np.linalg.norm(V)), s)  
  
 return i

Функция трассировки:

def TraceRay(O, D, t\_min, t\_max, depth):  
 *"""  
 Counts the color of the sphere at the nearest point t in the interval* ***:param*** *O: Starting point of the beam* ***:param*** *D: Ray coordinates* ***:param*** *t\_min: minimum parameter* ***:param*** *t\_max: maximum parameter* ***:param*** *depth: parameter of the mirror index* ***:return****: color of point  
 """* global BACKGROUND\_COLOR  
  
 closest\_sphere, closest\_t = ClosestIntersection(O, D, t\_min, t\_max)  
  
 if closest\_sphere is None:  
 return BACKGROUND\_COLOR  
  
 P = np.array(O + closest\_t\*D)  
 N = P - closest\_sphere.get\_elements()['center']  
 N = N / np.linalg.norm(N)  
 local\_color = np.array(closest\_sphere.get\_elements()['color']) \* ComputeLighing(P, N, -D, closest\_sphere.get\_elements()['specular'])  
  
 reflective = closest\_sphere.get\_elements()['reflective']  
 if depth <= 0 or reflective <= 0:  
 return local\_color  
  
 R = ReflectRay(-D, N)  
 reflected\_color = TraceRay\_P(P, R, 0.001, np.inf, depth - 1)  
  
 return np.array(np.dot(local\_color, (1 - reflective)) + np.dot(reflected\_color, reflective))

**Экспериментальные результаты.**

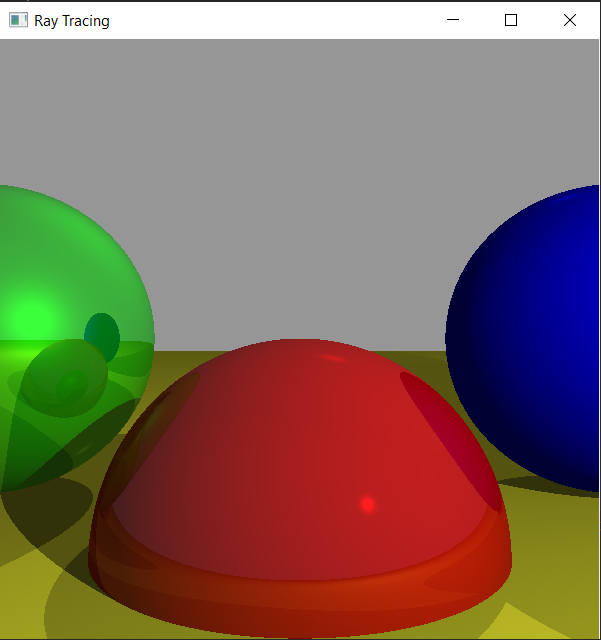
На рисунке 1 представлена сформированная сложная сцена.

Рисунок 1 – сложная сцена

**Вывод**

В данной лабораторной работе был реализован алгоритм трассировки лучей при формировании сложной сцены.