МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по компьютерной графике

Учебно-методическое пособие

Часть 4. Визуализация геометрических моделей

2018 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по компьютерной графике. Учебно-методическое пособие. Часть 4. Визуализация геометрических моделей. Озерск: ОТИ НИЯУ МИФИ, 2018. — 43 с.

В пособии подробно излагается, как выполнять практические работы по дисциплине «Инженерная и компьютерная графика». Работы четвертой части изучения дисциплины включают в себя методы визуализации геометрических моделей: методы Гуро и Фонга, метод обратной трассировки лучей.

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Общие цели занятий

В ходе практических работ изучаются основы визуализации геометрических моделей. В этой части работ рассматриваются следующие темы:

1) Вывод каркасных моделей.

2) Закраска методом Гуро.

3) Закраска методом Фонга.

4) Метод обратной трассировки лучей.

Для выполнения работ преподаватель предоставляет шаблоны проектов, реализующие оконные приложения.

1. Работа КГ-401. Каркасная модель

Цели:
- изучение представления полигональных моделей.
Задачи:
- определение графических классов;
- вывод каркасной модели;
- вывод видимых граней модели;

1.1. Рабочее пространство

Скачаем архив проекта render. Извлечем из архива каталог render в корневой каталог диска C:. Откроем проект.

Убедимся, что целевой платформой является x86 или Win32.

Следующие модули проекта используются для программирования.

В модулях box.h, cyl.h и user.h описываются геометрические модели параллелепипеда, цилиндра и объекта по заданию.

В модуле point.h описывается класс точки.

В модуле triang.h описывается класс треугольной грани. Часть методов этого класса вынесена в отдельные модули. В модуле gouraud.h описываются методы для закраски по методу Гуро, а в модуле phong.h описывается методы для закраски по методу Фонга.

В модуле trans.h описываются функция проецирования.

В модуле draw.h описывается рисование объекта.

В функции Initials этого модуля задаются начальные установки, используемые при запуске программы. Функция GetViewPoint вычисляет координаты точки зрения в мировой системе координат. Функция DrawView выполняет необходимые преобразования и выводит грани полигонального объекта. Эта функция вызывается автоматически при изменении точки зрения, положения источника света или параметров объекта.

Основной программный код должен быть написан в методах графических классов. Основной графический класс — triang, описывающий одну треугольную грань полигонального объекта.

1.2. Классы и объекты проекта

1.2.1. Класс point

Класс point описывает пространственные точку или вектор.

Элементы X, Y, Z представляют координаты.

В классе определены конструктор по умолчанию и конструктор преобразования. Метод set задает координаты, метод length вычисляет длину вектора, метод normalize нормализует вектор, метод scalar вычисляет скалярное произведение. В классе реализован набор операций, который может быть расширен при необходимости.

1.2.2. Типы matrix и triangs

Для описания матриц в шаблоне используется тип matrix, определенный как указатель на точку. Для описания множества треугольных граней используется тип triangs (модуль point.h):

```
// тип ссылки на матрицу
typedef point * matrix;

// предварительное объявление класса треугольной грани
struct triang;

// тип ссылки на треугольные грани
typedef triang * triangs;
```

При описании объектов данные заносятся в массивы, определенные для каждого объекта отдельно (модуль render.h):

```
// матрицы объектов
point box_matrix_W[BOX_POINTS]; // мировые координаты коробки
point box_matrix_E[BOX_POINTS]; // видовые координаты коробки
point box_matrix_S[BOX_POINTS]; // экранные координаты коробки
point cyl_matrix_W[CYL_POINTS]; // мировые координаты цилиндра
point cyl_matrix_E[CYL_POINTS]; // видовые координаты цилиндра
point cyl_matrix_S[CYL_POINTS]; // экранные координаты цилиндра
point usr_matrix_W[USR_POINTS]; // мировые координаты объекта
point usr_matrix_E[USR_POINTS]; // видовые координаты объекта
point usr_matrix_S[USR_POINTS]; // экранные координаты объекта

// треугольные грани объектов
triang box_triangs[BOX_triang]; // треугольники коробки
triang cyl_triangs[CYL_triang]; // треугольники цилиндра
triang usr_triangs[USR_triang]; // треугольники объекта
```

При выборе для отображения конкретного объекта, на который указывает переменная object, в функции формирования сцены устанавливаются указатели на матрицы координат и множества треугольников, а также значения количеств вершин и треугольных граней:

```
// формирование сцены
void Scene() {
    switch (object) {
    case BOX:
        WCO = box_matrix_W;
        ECO = box_matrix_E;
        SCO = box_matrix_S;
        TRIS = box_triangs;
        PointNumber = BOX_POINTS;
        TriasNumber = BOX_triang;
        break;
    case CYL:
        . . .
        break;
    default:
        . . .
    }
}
```

Программист использует переменные WCO, ECO, SCO и TRIS как синонимы соответствующих массивов координат и треугольников, а переменные PointNumber и TriasNumber задают размерности массивов.

Точка зрения на объект задается в сферических координатах при помощи переменных XAngle, ZAngle и Plane (модуль render.h).

1.2.3. Класс triang

Основные действия происходят в этом классе.

Треугольная грань задается тремя вершинами. Следующие константы используются для символического определения вершин идентификаторами A, B и C:

```
// индексы вершин грани
#define PA 0
#define PB 1
#define PC 2
```

В классе определены следующие элементы данных.

V[] — номера вершин в матрице координат.

VI[] — освещенности вершин.

VN[] — нормали граней.

VL[] —векторы на источник света в вершинах.

ldist[] —расстояния от вершин до источника света.

VS[] —векторы на точку зрения в вершинах.

ADJ[] — грани, сопряженные с данной в вершинах.

В классе определены следующие методы.

set — задает вершины.

adj — задает сопряженную грань в вершине.

normalZ — возвращает координату Z вектора нормали.

normal — вычисляет вектор нормали грани.

draw_edges — рисует ребра.

vectors — вычисляет векторы в вершинах граней.

gouraud — выполняет закраску грани по методу Гуро.

phong — выполняет закраску грани по методу Фонга.

vertexN — вычисляет нормаль в вершине.

Вспомогательная функция intens вычисляет освещенность точки.

Вспомогательная функция gouraud_line рисует линию растра для закраски по методу Гуро, а вспомогательная функция phong_line рисует линию растра для закраски по методу Фонга.

Переменные для расчета освещенности:

Intens — сила источника света,

LX, LY, LZ — положение источника света.

Свойства поверхности объекта заданы в модуле render.h переменными KA, KD, KS и PH.

1.3. Описание объекта «Параллелепипед»

Описание параллелепипеда выполняется в модуле box.h.
Лучше всего иметь рисунок объекта (рисунок 1).

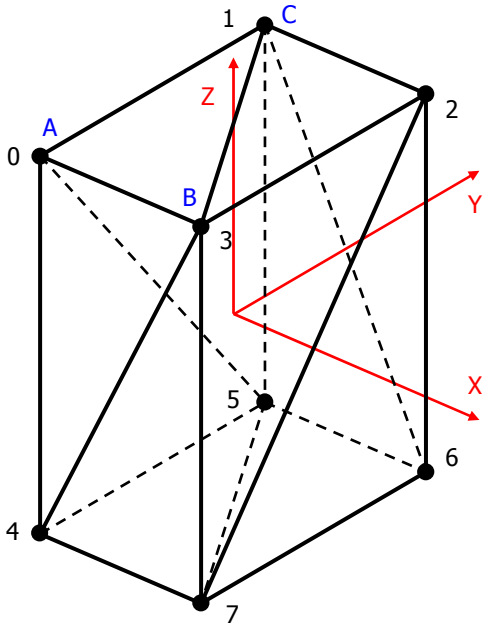

Рисунок 1 — Расчетная схема параллелепипеда

Каждой вершине присваивается номер. Центр системы координат находится в центре объекта. Половины сторон объекта задаем константами BX, BY, BZ. Нумерация вершин производится от нуля.

Вершина 0, например, задается следующим программным кодом:

```
WCO[0].set(-BX, -BY,  BZ);
```

Аналогичным образом задаются другие вершины. Не рекомендуется изменять порядок вершин, так как задание точки зрения сверху отображает в этом случае первые две грани, что требуется для отладки.

После задания всех вершин нужно задать треугольные грани.

Грань задается номерами вершин, указываемыми в порядке против часовой стрелки, если смотреть на лицо грани. Например, верхняя грань, вершины которой пронумерованы 0, 1 и 3, задается следующим образом:

```
TRIS[0].set(0, 3, 1);
```

Вторая верхняя треугольная грань при этом следующим образом:

```
TRIS[1].set(1, 3, 2);
```

Обратим внимание: две последних точки первой грани повторяются в обратном порядке во второй грани: 0–3–1 — 1–3–2. Так легче и задавать, и контролировать грани. Для этого для первой грани первую точку нужно выбирать ту, которая не лежит на общем ребре.

1.4. Рисование граней

Чтобы проверить правильность задания граней, их нужно нарисовать.

Рисование выполняет метод draw_edges, параметрами которого являются контекст устройства hdc и матрица экранных координат M.

Для рисования используется функция DrawObjectEdge.

Параметрами функции являются:

- контекст устройства hdc,
- координата X первой точки ребра,
- координата Y первой точки ребра,
- координата X второй точки ребра,
- координата Y второй точки ребра.

У грани три ребра, следовательно, эта функция вызывается три раза, при этом в первом вызове используются вершины грани A и B, при втором вызове — вершины B и C, при третьем вызове — вершины C и A. Координаты выбираются из матрицы M, номера вершин в этой матрице заданы в массиве V[] грани.

После этого переходим в функцию DrawView модуля draw.h, и описываем действия, необходимые для построения изображения.

Сначала в функции выполняются видовое и финальное перспективное преобразования. При этом координаты вершин пересчитываются сначала в матрицу ECO, затем в матрицу SCO.

Формируем цикл по всем граням массива TRIS, вызываем для каждой грани метод draw_edges, используя матрицу экранных координат SCO.

Если объект описан правильно, при его вращении должны наблюдаться все грани, видимые и невидимые.

1.5. Удаление невидимых граней

Для удаления ребер невидимых граней воспользуемся тем обстоятельством, что координата Z вектора нормали невидимой грани имеет отрицательное значение. Так как для последующих расчетов требуется знать вектор нормали, то нужно определить метод normal в классе грани, после чего определить метод normalZ.

Нормаль плоской грани рассчитывается по координатам любых трех ее вершин, заданных против часовой стрелки по следующим формулам:

```
X = (B.Y - A.Y) * (C.Z - A.Z) - (B.Z - A.Z) * (C.Y - A.Y)
Y = (B.Z - A.Z) * (C.X - A.X) - (B.X - A.X) * (C.Z - A.Z)
Z = (B.X - A.X) * (C.Y - A.Y) - (B.Y - A.Y) * (C.X - A.X)
```

Здесь обозначение вида A.X указывает на координату X вершины A.

Рассчитываем вектор нормали в методе normal, после чего расчет координаты Z копируем в метод normalZ. Переходим в функцию Initials модуля draw.h и устанавливаем признак рисования ребер drawedges.

Отлаживаем эту часть работы.

2. Работа КГ-402. Расчет освещенности в вершинах граней

Цели:
- вычисление освещенности в точке;
Задачи:
- определение функции для вычисления освещенности в точке;
- вычисление освещенности вершин граней;

2.1. Составляющие освещенности

В целях упрощения задачи будем полагать, что есть только один точечный источник света, имеющий приведенную яркость I_l.

Освещенность в точке для закраски по методам Гуро и Фонга складывается из трех составляющих:

$I = I_a + I_d + I_s$.

$I_a = I_l\, k_l\, k_a$ — рассеянная в пространстве освещенность, отраженная от других объектов, и составляющая постоянный фон (*ambient*).

$I_d = I_l\, k_d\, cos\,\theta / d$ — диффузное рассеяние от падающих на поверхность лучей от источника света; угол θ — это угол между вектором нормали N и вектором источника света L. Здесь d — расстояние до источника света.

$I_s = I_l\, k_s\, cos^P\,\alpha / d$ — зеркальное отражение от поверхности лучей от источника света; угол α — это угол между вектором отражения R и вектором наблюдения S.

Используемые коэффициенты:

k_l — учитывает долю рассеяния света источника;

k_a — учитывает свойство поверхности отражать рассеянный свет;

k_d — учитывает свойство поверхности рассеивать падающий свет;

k_s — учитывает свойство поверхности отражать падающий свет;

P — учитывает свойство поверхности рассеивать отражаемый свет.

Последний коэффициент называется коэффициентом Фонга, он формирует так называемое пятно Фонга, которое в методе Гуро выглядит не как круглое пятно, а как многоугольник, что связано с особенностью интерполяции по методу Гуро.

В программе яркость источника света задана переменной Intens.

Ориентировочные значения Intens — 30000…80000.

Освещенность I_a задана переменной IntensA, коэффициент k_l принят равным 0,01. Ориентировочное значение IntensA равно 100.

Коэффициенты, учитывающие свойства поверхности, задаются как глобальные переменные KA, KD, KS, PH.

Ориентировочные значения равны:

KA = 0,2

KS = 0,5

KD = 0,3

PH = 2

2.2. Вычисление вектора нормали в вершине

Сначала нужно определить метод vertexN, вычисляющий нормаль в вершине треугольной грани. Параметрами метода являются матрица мировых координат W, номер вершины *i*, массив треугольных граней t.

На данном этапе достаточно просто вычислить нормаль грани N методом normal и вернуть ее.

2.3. Вычисление векторов в вершинах граней

Далее переходим в метод vectors и вычисляем векторы в вершинах.

Вычисления ведутся в мировой системе координат.

Параметры метода: матрица мировых координат W, вектор источника света *l*, вектор точки зрения s, интенсивность источника света Io, приведенная окружающая освещенность Ia, массив треугольных граней t.

Вычисления ведутся в цикле по трем вершинам, номер текущей вершины обозначен переменной цикла *i*.

Сначала вычисляем вектор нормали VN[i] при помощи метода vertexN.

Далее определяем точку вершины, для чего объявляем вектор p типа point, и присваиваем ему значение номер V[i] из матрицы координат W.

Рассчитываем вектор источника света VL[i] как разность векторов *l* и p. Рассчитываем расстояние до источника света ldist[i], используя метод length, применяемый к вектору VL[i].

Рассчитываем вектор точки зрения VS[i] как разность векторов s и p.

Рассчитываем освещенность в вершинах:

```
// освещенность в вершине i
VI[i] = intens(VN[i], VL[i], VS[i], ldist[i], Io, Ia);
```

Конец итерации.

Отлаживаем эту часть кода.

Для этого нужно установить определенные параметры в функции Initials модуля draw.h.

XAngle = 0

ZAngle = 0

Plane = 800

LX = -BX

LY = -BY

LZ = 180

Далее нужно ограничить количество граней одной.

Для этого в описании объекта нужно закомментировать строки, описывающие все треугольные грани, кроме первой (нулевой). При этом предполагается, что грань задана точками 0–3–1 (рисунок 1). Любое отклонение от этого условия не даст возможности быстро проконтролировать вычисляемые значения.

Устанавливаем точку остановки в конце цикла и сверяем вычисляемые значения с приведенными в следующей таблице.

i	0	1	2
VN[i]	(0, 0, 1)	(0, 0, 1)	(0, 0, 1)
p	(-50, -60, 70)	(50, -60, 70)	(-50, 60, 70)
VL[i]	(0, 0, 110)	(-100, 0, 110)	(0, -120, 110)
ldist[i]	110	148	162
VS[i]	(50, 60, 730)	(-50, 60, 730)	(50, -60, 730)

2.4. Вычисление освещенности точки

Переходим во вспомогательную функцию intens.

В соответствии с приведенными выше формулами рассчитываем освещенность точки, используя параметры функции. Параметрами являются: вектор нормали N, вектор источника света L, вектор точки зрения S, расстояние до источника света dist, интенсивность источника света Io, приведенная окружающая освещенность Ia.

Сначала нормализуем все векторы методом normalize.

Далее делим интенсивность источника света Io на расстояние до источника света dist.

Вычисляем косинус θ cosT как скалярное произведение N и L.

Проверяем значение cosT. Если оно больше нуля, то рассчитываем диффузное рассеяние и зеркальное отражение следующим образом:

- перемножаем Io, KD и cosT, результат записываем в переменную id;
- вычисляем вектор отражения R по формуле $N\cdot(2\cdot\cos\theta) - L$.
- вычисляем косинус α cosA как скалярное произведение R и S.
- если значение cosA больше нуля, рассчитываем зеркальное отражение, перемножая Io, KS и $\cos A^{PH}$. Результат записываем в переменную is.

В конце функции вычисляем I как сумму Ia, id и is.

Для отладки перейдем в функцию Initials модуля draw.h.

Используем одну грань объекта, как и в предыдущем случае, с теми же параметрами точки зрения.

Координаты источника света должны быть следующими:

LX = -BX

LY = -BY

LZ = 180

Значения коэффициентов свойств поверхности должны быть равны:

KA = 0,2

KD = 0,5

KS = 0,3

PH = 2

Значение переменной Intens должно быть равно 25000, а значение переменной IntensA должно получиться равным 50 (25000×0,01×0,2).

Устанавливаем точку остановки в конце функции и сверяем вычисляемые значения с приведенными в следующей таблице.

точка	0	3	1
N	(0, 0, 1)	(0, 0, 1)	(0, 0, 1)
L	(0, 0, 1)	(-0.672, 0, 0.739)	(0, -0.737, 0.675)
S	(0.068, 0.081, 0.994)	(-0.068, 0.081, 0.994)	(0.068, -0.081, 0.994)
Io	227	168	153
cosT	1	0.739	0.675
id	113	62	51
R	(0, 0, 1)	(-0.672, 0, 0.739)	(0, 0.737, 0.675)
cosA	0.994	0.689	0.611
is	67	24	17
I	231	136	119

Если все верно, эта часть завершена.

3. Работа КГ-403. Закраска методом Гуро

Цели:
- закрашивание треугольных граней методом Гуро;
Задачи:
- интерполяция освещенности вдоль линии растра.
- вращение вершин;
- интерполяция освещенности вдоль ребер грани;
Вычисления производятся в видовой системе координат.

Закраска методом Гуро выполняется интерполированием освещенности вершин по ребрам и по линиям растра (рисунок 2).

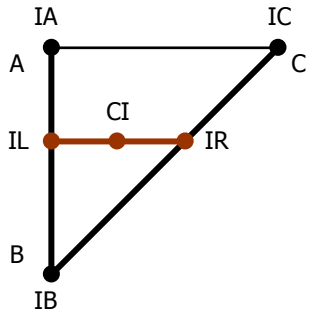

Рисунок 2 — Интерполяция освещенности точки

Чтобы вычислить освещенность крайних точек линии растра IL и IR, освещенность в вершинах интерполируется вдоль ребер AB и CB.

Цвет точки вычисляется как функция освещенности, значения которой лежат в диапазоне 0…255. В простейшем случае цвет точки серый, составляющие цвета RGB одинаковы и равны значению освещенности. Если освещенность точки равна CI, точка рисуется примерно так:

```
// вычисляемая освещенность
int c = (int)CI;
// приведение к 0...255
if (c > 255) {
    // яркость максимальная
    c = 255;
//} else if (c < 0) {
//  // яркость минимальная
//  c = 0;
}
// вычисляем цвет
COLORREF color = RGB(c, c, c);
// выводим точку
DrawPixel(hdc, x, y, color);
```

При этом мы движемся вдоль линии растра (по координате x) от точки с освещенностью IL до точки с освещенностью IR, с каждым приращением x на единицу изменяя текущую освещенность CI на величину DI, приращение освещенности на один пиксель.

3.1. Интерполирование вдоль линии растра

Интерполяция вдоль линии растра выполняется во вспомогательной функции gouraud_line. Параметрами функции являются:

- контекст устройства hdc;
- координата y линии растра;
- координата x левой границы линии растра XL;
- координата x правой границы линии растра XR;
- освещенность левой точки линии растра IL;
- освещенность правой точки линии растра IR.

Количество точек в линии растра DX равно XR – XL.

Рисуем точки слева направо, поэтому текущая освещенность CI равна освещенности левой точки IL. Если освещенность правой точки равна IR, а освещенность левой точки IL, диапазон освещенности равен IR – IL.

Поделив диапазон освещенности на длину линии растра в точках, получим значение DI, соответствующее приращению освещенности при переходе от одной точки к другой. То есть, освещенность точки равна освещенности точки, расположенной левее, плюс DI.

Вычисления выполняются в цикле по координате x, значения которой изменяются от XL до XR включительно. В цикле на основании текущей освещенности CI вычисляется цвет и рисуется точка, как показано выше.

После рисования очередной точки текущая освещенность CI увеличивается на значение DI.

3.2. Вращение вершин

Интерполяция вдоль ребер грани выполняется отдельно для каждого возможного случая ориентации грани отдельно. Рассмотрим одну из возможных ориентаций (рисунок 3, ориентация ABCB).

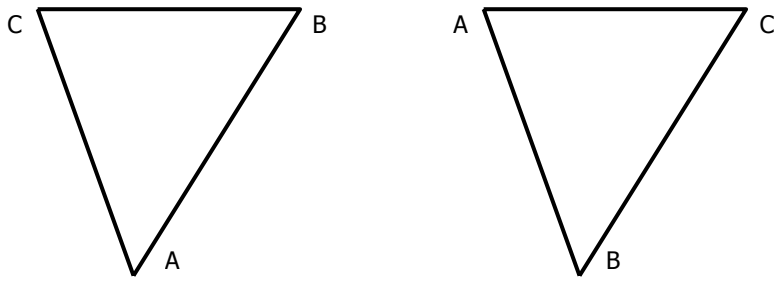

Рисунок 3 — Вращение вершин

Слева на рисунке 3 показано положение вершин, при котором вверху слева находится вершина C. На самом деле вверху слева может оказаться любая из вершин A, B или C. Чтобы не программировать каждый возможный случай расположения вершин отдельно, условимся, что программный код исходит из предположения, что вверху слева (или просто вверху) находится вершина A, как на рисунке 3 справа.

Для расчета будем использовать переменные XA, YA, XB, YB, XC и YC, которые обозначают координаты вершин точек A, B и C после вращения.

Если на самом деле вверху слева находится вершина C, то расчетным координатам XA и YA присваиваем координаты вершины C. Соответственно, XB и YB присваиваем координаты вершины A, а XC и YC присваиваем координаты вершины B.

Таким образом, мы как бы повернем вершины по часовой стрелке.

Никакого вращения на самом деле нет. Вращение вершин, — это просто удобный термин для обозначения действия.

Вращение вершин выполняется в функции get_orient. Параметрами функции являются три абсциссы вершин и три возвращаемых индекса.

Чтобы выполнить вращение, нужно выяснить, где находятся вершины после преобразования координат следующим образом:

```
void get_orient(int ya, int yb, int yc, int & A, int & B, int & C) {
    if (yc > ya && yc >= yb) {
        // до поворота
        //   C   | C   B | C     |    C |
        //     |   |   B | A     |
        // A   B |   A   | A     |    B |
        //---- вращаем вершины вправо ---
        // после поворота
        //   A   | A   C | A     |    A |
        //     |   |   C | B     |
        // B   C |   B   | B     |    C |
        return;
    }
    if (yb > yc && yb >= ya) {
        // до поворота
        //   B   | B   A | B     |    B |
        //     |   |   A | C     |
        // C   A |   C   | C     |    A |
        //---- вращаем вершины влево ----
        // после поворота
        //   A   | A   C | A     |    A |
        //     |   |   C | B     |
        // B   C |   B   | B     |    C |
        return;
    }
    //------ не вращаем вершины -----
    // до и после поворота
    //   A   | A   C | A     |    A |
    //     |   | C | B |
    // B   C |   B   | B     |    C |
    A = PA;
    B = PB;
    C = PC;
}
```

В каждом из трех случаев нужно присвоить параметрам A, B и C значения индексов вершин PA, PB, PC. Например, в случае, когда вращения нет, A = PA, B = PB, C = PC. После того, как вращение вершин запрограммировано, нужно определить координаты и освещенности в методе gouraud.

В методе gouraud сначала определяем расчетные координаты:

```
// вращение вершин
get_orient((int)M[V[PA]].Y, (int)M[V[PB]].Y, (int)M[V[PC]].Y, VA…
// координаты после поворота
int XA = (int)M[V[VA]].X;
int YA = (int)M[V[VA]].Y;
int XB = (int)M[V[VB]].X;
int YB = (int)M[V[VB]].Y;
int XC = (int)M[V[VC]].X;
int YC = (int)M[V[VC]].Y;
```

Далее определяем фактические освещенности вершин:

```
// освещенность вершин
double IA = VI[VA];
double IB = VI[VB];
double IC = VI[VC];
```

Остается только выполнить интерполяцию вдоль границ растра.

3.3. Интерполирование вдоль ребер грани

После вращения вершин и определения координат и освещенностей в вершинах нужно выяснить действительную ориентацию грани, используя координаты вершин после вращения:

```
if (YA == YC) {
    // A   C
    //
    //   B
    // ориентация ABCB
} else if (YB == YC) {
    //   A
    //
    // B   C
    // ориентация ABAC
} else if (YB > YC) {
    //   A
    // B
    //   C
    // ориентация ACAC
} else {
    // A
    //   C
    // B
    // ориентация ABAB
}
```

При параметрах точки зрения, которые сейчас установлены, текущей ориентацией является ABCB.

Поэтому сначала программируем этот случай.

Интерполяция вдоль ребер выполняется примерно так же, как и интерполяция вдоль линии растра. Однако в этом случае необходимо также интерполировать не только освещенность, но и координаты X левой и правой точек линии растра.

Для рассматриваемого случая сначала определяется количество линий растра DY, вычисляемое как разность координат Y вершин A и B (или C и B). Далее, если значение DY более нуля, вычисляются приращения DXL и DXR координат X по мере перемещения по линиям растра (по координате y):

```
DXL = (double)(XB - XA) / DY;
DXR = (double)(XB - XC) / DY;
```

Затем нужно вычислить приращение освещенности для левого и правого ребер DIL и DIR. Принцип вычисления аналогичный, например:

```
DIL = (IB - IA) / DY;
DIR = (IB - IC) / DY;
```

Далее вычисляются координаты XL и XR левой и правой точек первой линии растра. Они совпадают с координатами точек A и C, находящихся вверху. Точно также вычисляются освещенности этих точек IL и IR, равные освещенностям точек A и C.

После вычисления этих значений формируем цикл по координате y, которая принимает значения от YA до YB включительно, причем координата y уменьшается в цикле, а не увеличивается.

В рамках одной итерации рисуется одна линия растра, соответствующая текущей координате y. Для рисования вызываем вспомогательную функцию gouraud_line, определенную ранее. Заметим, что при вызове этой функции координаты XL и XL границ линии растра нужно приводить к целому типу.

После этого нужно изменить значения текущих значений границ линии растра XL, YL, IL, IR, прибавляя к ним соответствующие приращения, вычисленные перед началом цикла.

Результат закраски расчетной схемы приведен на рисунке 4.

Рисунок 4 — Закрашенный треугольник, ориентация ABCB

Другие ориентации на расчетной схеме можно получить, если вращать нулевую грань вокруг оси Z, изменяя угол XAngle в функции Initials модуля draw.h на 40°, 333° и 150°.

При этом нужно учитывать, что в ориентациях ACAC и ABAB левая или правая ветвь, образующая границу линий растра, состоит из двух ребер, поэтому процесс интерполяции разбивается на два участка, до точки соединения ребер, и после этой точки (рисунок 5, точка соединения B).

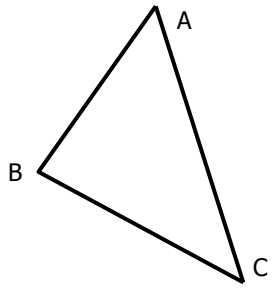

Рисунок 5 — Ориентация ACAC

Правая ветвь на рисунке 5 является общей для двух участков интерполяции, поэтому значения DXR и DIL рассчитываются один раз в начале, а значения DXL и DIL рассчитываются перед началом каждого участка.

Следует также учесть, что цикл первого участка должен рисовать линии растра ДО точки B, а цикл второго участка начинать рисование с точки B, так, чтобы ни одна линия растра не рисовалась дважды, и не была бы пропущена.

На рисунке 6 показан результат закраски грани при значениях XAngle, равных 40° (слева), 333° (в центре) и 150° (справа).

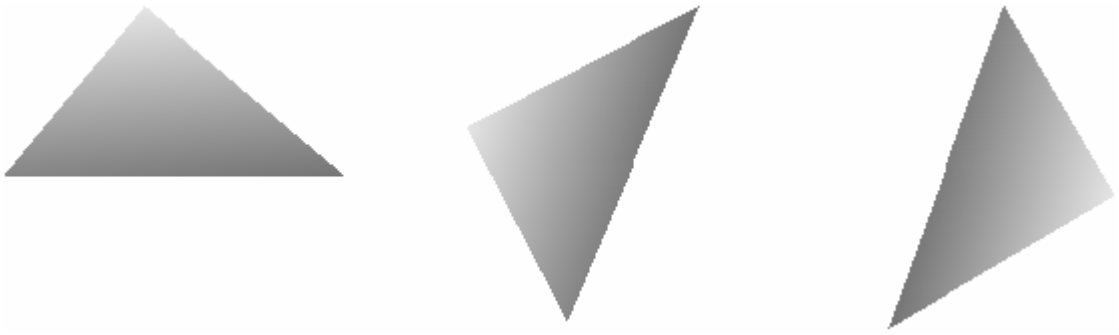

Рисунок 6 — Закраска Гуро в разной ориентации грани

После того, как все ориентации будут запрограммированы, нужно включить все грани объекта и установить следующие начальные параметры в функции Initials модуля draw.h:

XAngle = 30, ZAngle = 60

LX = 160, LY = 180, LZ = 180

Intens = 50000

Далее, интерактивно изменяя значения коэффициентов поверхности и положения источника света, можно оценить влияние параметров на результат закраски.

4. Работа КГ-404. Закраска гладкого объекта

Цели:

- уточнение вектора нормали в вершине;

Задачи:

- построение полигональной модели гладкого объекта;
- вычисление усредненного вектора нормали в вершинах граней;

В предыдущих работах в качестве объекта использовался параллелепипед, ребра граней которого ярко выражены. Для объекта, треугольные грани которого описывают гладкие поверхности, например, боковые поверхности цилиндра, ребра этих граней должны отсутствовать при их визуализации.

4.1. Построение полигональной модели цилиндра

Полигональная модель цилиндра строится в модуле cyl.h. В этом модуле определены следующие константы, задающие размеры объекта:

CR — радиус основания;

CH — половина высоты;

CN — количество боковых граней.

Фактически должна получиться призма с количеством граней CN.

На рисунке 7 показана примерная расчетная схема для нумерации вершин полигональной модели цилиндра при значении CN, равном 5.

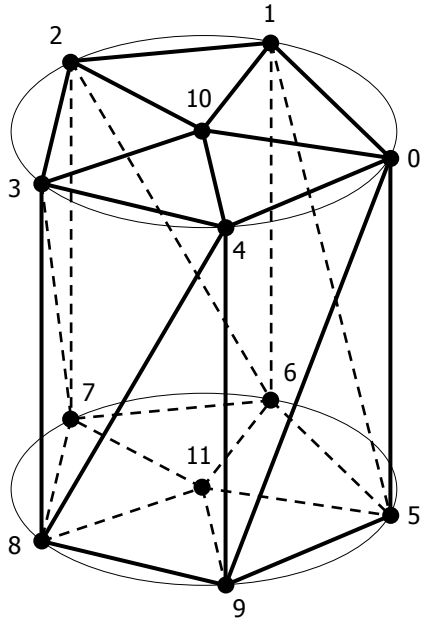

Рисунок 7 — Расчетная схема нумерации вершин цилиндра

Количество вершин цилиндра равно $2 \cdot CN + 2$, а количество треугольных граней равно $4 \cdot CN$.

Сначала нумеруем вершины верхнего круга, затем нижнего круга, затем центры кругов. Координаты нулевой вершины (CR, 0, –CH).

Проблема заключается в том, что далее при создании граней нужно указывать вершины граней в цикле, поэтому нужна формула для вычисления номеров вершин, зависящая от номера итерации *i* и константы CN.

Чтобы осилить эту сложность, нужна развертка модели (рисунок 8).

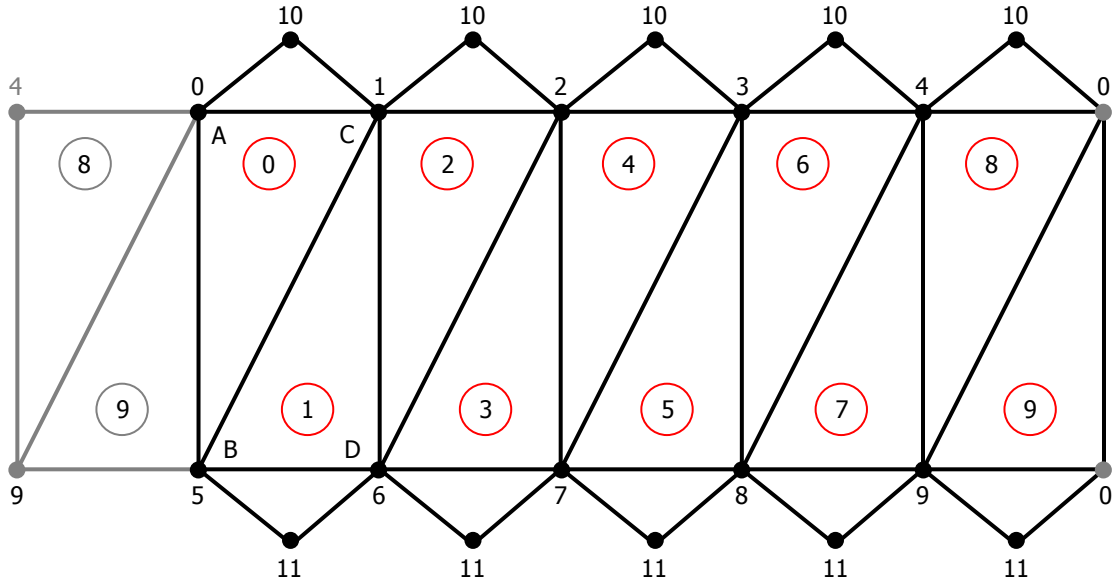

Рисунок 8 — Развертка модели цилиндра

Цифрами в кружках на рисунке 8 обозначены номера граней. Последовательность нумерации граней не обязательно такая, как указана на рисунке, в принципе может быть любая другая.

Процесс формирования цилиндра состоит из трех частей:

- расчет координат вершин;
- задание треугольных граней;
- задание сопряженных граней боковых граней.

4.1.1. Расчет координат

Для расчета требуется текущий угол поворота CA и приращение угла поворота DA. Изначально угол CA равен нулю, нулевая точка находится на направлении оси X на высоте CH. Приращение угла DA равно 2π/CN. Число, равное 2π, задано в проекте константой PI2.

Формируем цикл по переменной *i*, изменяющейся от 0 до CN исключительно, при этом *i* соответствует номеру вершины верхнего круга. Пусть переменная m соответствует номеру вершины нижнего круга. Тогда начальное значение m равно CN, и в каждой итерации переменная m увеличивается на единицу. Такое изменение переменных описывает следующая форма параметрического цикла:

```
for (int i = 0, m = CN; i < Cn; i++, m++, CA += DA) {
    // точки по окружности
}
```

В цикле вычисляем координаты x и y по формулам:

$x = r \cdot \cos\alpha$

$y = r \cdot \sin\alpha$

где r — радиус окружности, α — угол CA.

Рассчитав координаты, можем в одной итерации задать вершины i и m матрицы WCO при помощи метода set. Координата Z для вершины i равна CH, а для вершины m — минус CH.

Далее нужно задать вершины, соответствующие верхней и нижней центральным точкам. Удобно обозначить эти вершины переменными, например, переменная pu для верхнего круга и переменная pd для нижнего.

Для дальнейших расчетов потребуется также значение, равное 2·CN.

Поэтому удобно будет ввести переменную N2, равную 2·CN.

Тогда верхняя центральная точка pu будет иметь номер, равный N2, а нижняя центральная точка pd, — номер, равный N2 + 1.

Соответственно, задаем координаты точек pu и pd.

На этом все вершины заданы.

4.1.2. Расчет треугольных граней

В одной итерации цикла будем задавать одновременно две боковые грани, соответствующие одной плоскости (например, грани 0 и 1), а также верхнюю и нижнюю грани, примыкающие к этим боковым.

Обозначим переменной M боковую четную, переменной N — боковую нечетную, переменной u — верхнюю, а переменной d — нижнюю грани.

Тогда после каждой итерации M и N изменяются на 2 единицы, а переменные u и d, — на единицу. Начальные значения:

M = 0, N = 1, u = N2, d = N2 + CN.

Введем также номера вершин граней в виде переменных A, B, C и D, в соответствии с рисунком 8. Тогда четная грань задается точками (A, B, C), а нечетная грань, — точками (C, B, D). Верхняя грань при этом задается точками (pu, A, C), нижняя грань, — точками (pd, D, B). Для расчета переменных A, B, C и D потребуется также переменная j, начальное значение которой равно 1, и в каждой итерации увеличивающаяся на единицу.

Для нулевой и средних граней A, B, C и D вычисляются так:

A = i

B = i + CN

C = j

D = j + CN

Для последней грани (i = CN – 1) эти формулы не подходят, для нее:

A = i

B = N2 – 1

C = 0

D = CN

Программируем цикл и проверяем, как рисуются ребра.

4.1.3. Расчет сопряженных граней

Остается вычислить номера сопряженных граней боковых граней.

Для цилиндра у любой боковой грани сопряженной является только одна грань, поэтому класс triang определяет только по одной сопряженной грани для каждой вершины в массиве ADJ[]. Для других объектов, например, для сферы, количество сопряженных граней может быть больше.

Обратимся к рисунку 8.

Обозначим грань слева переменной L, грань справа, — переменной R.

Тогда для четной грани сопряженными в вершинах A и B являются левые грани, в вершине C — правая грань. Для нечетной грани в вершинах C и D сопряженными являются правые грани, в вершине B — левая грань.

Программируется это так:

```
TRIS[M].adj(PA, L);
TRIS[M].adj(PB, L);
TRIS[M].adj(PC, R);
TRIS[N].adj(PA, R);
TRIS[N].adj(PB, L);
TRIS[N].adj(PC, R);
```

Для нулевой и первой граней левая грань N2 – 1, правая — вторая.

Для последних граней (8 и 9) левая M – 1, правая — нулевая.

Для средних граней левая M – 1, правая — M + 2.

Программирование ведется в цикле определения граней, так как используются переменные, обозначающие грани M и N.

4.2. Вектор нормали в вершине

После того, как будет получена правильная модель цилиндра, нужно отключить вывод ребер. В результате мы должны увидеть правильно закрашенную призму, а не цилиндр. Чтобы получить закраску, соответствующую цилиндру, нужно вычислить нормаль в вершинах граней.

Метод vertexN.

Сейчас он вычисляет вектор нормали грани N.

Теперь нужно проверить, чему равно значение ADJ[*i*]. Для удобства это значение запишем в переменную n. Если значение n не равно –1, значит задана сопряженная грань, и нужно вычислить ее вектор нормали N1. Обе нормали нужно нормализовать и сложить (прибавить N1 к N).

Если все сделано верно, цилиндр должен выглядеть как цилиндр, а не как призма.

5. Работа КГ-406. Метод обратной трассировки лучей

Цели:
- изучение основ обратной трассировки лучей;

Задачи:
- формирование геометрических моделей;
- трассировка первичным лучом;
- трассировка отраженным лучом;
- трассировка преломленным лучом;

Опорные документы:
[1]

5.1. Шаблон проекта tracing

Для выполнения работы требуется шаблон проекта tracing.

Шаблон максимально подготовлен для того, чтобы заниматься только созданием графических объектов и методов трассировки.

Модули проекта:
tracing.h — константы и переменные;
tracing.cpp — оконная часть проекта;
util.h — разные классы;
point.h — класс точки;
scene.h — класс сцены;
render.h — методы трассировки;
object.h — графические объекты;
draw.h — построение сцен.

5.1.1. Классы модуля util

Класс Color описывает цвет и операции с цветом.

Класс Medium описывает среду распространения луча. Задает такие параметры среды, как затухание atten и преломление refract.

Класс Surface описывает свойства поверхности.

Содержит коэффициенты фоновой освещенности Ka, диффузного рассеяния Kd, зеркального отражения Ks, рассеяния отраженного света (коэффициент Фонга) Ph, отраженной освещенности Kr, преломленной освещенности Kt.

Свойство color задает цвет поверхности, свойство medium — характеристики внутренней среды объекта.

Класс Ray описывает луч.

Свойства луча:
org — точка начала луча;
dir — вектор направления луча;
medium —среда распространения;
weight — вес луча;

Метод point_at вычисляет точку на расстоянии t от начала луча.

Класс Hit описывает точку пересечения луча с объектом.

Свойства пересечения:

intersect — признак наличия пересечения;

id — идентификатор пересеченного объекта;

cosVN — косинус угла между лучом и нормалью;

dist — расстояние до точки пересечения;

normal — нормаль поверхности в точке пересечения;

point — координаты точки пересечения;

entering — признак входа в тело объекта;

surface — поверхность в точке пересечения.

Класс Light описывает источник света. Свойства:

point — координаты источника света;

color — цвет источника света;

scale — коэффициент ослабления светового потока в зависимости от расстояния.

Класс Object — это интерфейс, который должны поддерживать все объекты сцены. Важным здесь является метод intersect, который вычисляет точку пересечения поверхности объекта с лучом, и который переопределяется для каждого объекта в соответствии с его геометрией.

Функция next_id этого модуля вычисляет очередной идентификатор для нумерации объектов.

5.1.2. Класс точки

Класс Point описывает точку или вектор. В классе определены все необходимые операции. Скалярное произведение векторов определено как операция *, а векторное произведение — как операция ^.

5.1.3. Класс сцены

В модуле scene.h определен класс сцены Scene. Основное программирование происходит в модуле render.h, в котором находятся методы класса сцены. Свойства сцены:

LASTX, LASTY — размеры экрана проецирования;

HOMEX, HOMEY — смещение к центру экрана;

distance — расстояние до экрана;

anglex — начальный угол точки зрения в плоскости XY;

anglez — начальный угол отклонения точки зрения от оси Z;

plane — начальное расстояние до точки зрения;

level — текущая глубина трассировки;

count_light — количество источников света;

count_object — количество объектов на сцене;

background — цвет фона (неба);

ambient — цвет рассеянного света.

Методы сцены *вспомогательные*:

set_max_level — задает максимальную глубину трассировки.

light_add — добавляет источник света;

object_add — добавляет объект.

Методы сцены *основные*:

render — вычисляет первичные лучи;

trace — трассирует луч;

shade — вычисляет цвет точки пересечения луча и объекта;

shadow — вычисляет затенение источника света.

5.1.4. Модули object и draw

В модуле object.h определяются геометрические модели объектов в виде классов, производных от класса Object.

В модуле draw.h описываются сцены, и задается начальная сцена. Здесь же описана функция DrawView, которая рисует изображение.

5.2. Построение геометрической модели прямоугольника

Геометрические модели объектов описываются в модуле object.h.

Описываем геометрический объект «прямоугольник»:

```
class Rect : public Object {
    Point Loc, N, KU, KV; // точки объекта, нормаль, базис
    double U0, V0;        // начальная точка базиса
public:
    // строит объект, вычисляет параметры
    Rect(Point A, Point B, Point C) { }
    // вычисляет пересечение объекта с заданным лучом
    int intersect(Ray ray, Hit & hit) {
        double cosVN = 0, U = 0, V = 0;
        return 0;
    }
};
```

Точка *A* задает начальную точку объекта, в точки *B* и *C* задают точки, расположенные на концах направлений *X* и *Y* (рисунок 9).

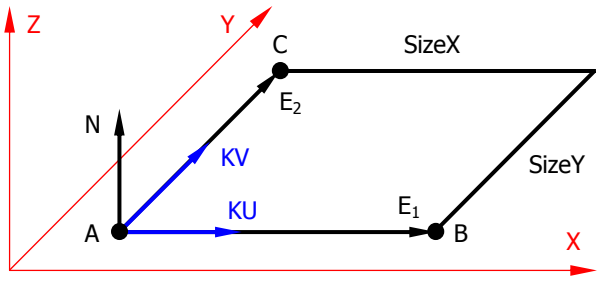

Рисунок 9 — Базис прямоугольника

По точкам A, B и C вычисляются векторы E_1 и E_2 направлений *X* и *Y*, и рассчитывается вектор нормали N.

5.2.1. Расчет параметров прямоугольника

Параметрами является базис K_U, K_V, и начальная точка базиса U_0, V_0.

Параметры конструктора — это точки A, B и C.

Сначала в конструкторе задается идентификатор объекта посредством вызова функции next_id.

Точка A запоминается как начальная точка объекта Loc.

Затем рассчитываются векторы E_1 и E_2 по формулам:

$E_1 = B – Loc$

$E_2 = C – Loc$

Нормаль вычисляется как векторное произведение векторов E_1 и E_2.

Нормаль нормализуется методом normalize.

Далее вычисляются коэффициенты матрицы по скалярным произведениям векторов E_1 и E_2:

$S_{11} = E_1 \cdot E_1$

$S_{12} = E_1 \cdot E_2$

$S_{22} = E_2 \cdot E_2$

Вычисляется детерминант матрицы

S_{11} S_{12}

S_{12} S_{22}

по формуле $D = S_{11} \cdot S_{22} – S_{12} \cdot S_{12}$

Наконец, вычисляется базис по формулам

$K_U = (E_1 \cdot S_{22} – E_2 \cdot S_{12}) / D$

$K_V = (E_2 \cdot S_{11} – E_1 \cdot S_{12}) / D$

и начальная точка базиса по формулам

$U_0 = Loc \cdot K_U$

$V_0 = Loc \cdot K_V$

5.2.2. Расчет точки пересечения

Пересечение произвольного луча с прямоугольником определяется с помощью метода intersect. Этот метод должен возвращать 1 в случае, если луч пересекает прямоугольник, и 0, если не пересекает.

Параметрами метода являются луч ray и объект hit, описывающий точку пересечения (если есть). Этот объект возвращается вызывающей процедуре, поэтому описан как ссылка.

Сначала нужно определить, не является ли луч параллельным плоскости объекта, вычислив косинус угла между нормалью объекта и лучом. Если косинус с точностью EPS равен нулю, луч не пересекает плоскость объекта, метод возвращает 0. Объект hit при этом должен быть очищен.

Если же косинус отличен от нуля, нужно вычислить точку пересечения луча с плоскостью объекта, определить координаты U и V точки в базисе K_U, K_V, и, если эти координаты находятся в интервале [0…1], принять, что точка пересечения находится внутри прямоугольника.

Порядок вычислений в методе intersect следующий.

1. Очистить объект hit.

2. Вычислить косинус угла между лучом и нормалью cosVN.

3. Если абсолютная величина cosVN меньше EPS, вернуть 0.

4. Вычислить расстояние до точки пересечения, записать в hit.dist.

Расстояние вычисляется по формуле

```
hit.dist = ((Loc - ray.org) * N) / cosVN;
```

5. Если расстояние hit.dist меньше EPS, вернуть 0.

6. Вычислить точку пересечения и записать ее в свойство hit.point.

Точку пересечения возвращает метод point_at объекта ray, параметром является расстояние hit.dist.

7. Вычислить координату U точки пересечения в базисе UV:

```
U = (hit.point * KU) - U0;
```

8. Если U < 0 или U > 1,000001, вернуть 0.

9. Вычислить координату V точки пересечения в базисе UV:

```
V = (hit.point * KV) - V0;
```

10. Если V < 0 или V > 1,000001, вернуть 0.

11. Установить свойство intersect объекта hit равным 1.

12. Записать идентификатор объекта в hit.id.

13. Записать surface объекта в hit.surface.

14. Записать cosVN в hit.cosVN,

15. Если cosVN меньше нуля, то

- записать N в hit.normal,
- установить свойство entering объекта hit равным 1,

иначе

- записать –N в hit.normal,
- установить свойство entering объекта hit равным 0.

15. Вернуть 1.

5.3. Построение сцены

В функции Initials модуля draw.h должна быть выбрана первая сцена:

```
void Initials() {
    current_scene = 0;
}
```

Переходим в функцию построения первой сцены, состоящей на первом этапе из одного только зеленого прямоугольника Rect2 (рисунок 10).

Порядок создания сцены.

1. Установить размеры экрана равными 80 и 80.

2. Установить центр экрана 0 и 0.

3. Углы anglex и anglez установить в 0.

4. Расстояния plane и distance установить равными 100.

5. Установить максимальную глубину трассировки равную 1.

6. Установить цвет фона background равным RGB(128, 207, 255).

7. Установить цвет ambient равным RGB(255, 255, 255).
8. Объявить источник света L.
9. Установить цвет источника света равным RGB(255, 255, 255).
10. Задать координаты источника света равными (0, 0, 60).
11. Добавить источник света L в сцену.
12. Объявить новый объект Rect:

```
// зеленый
Rect * rg = new Rect(Point(0,0,20), Point(50,0,20), Point(0,50,20));
```

13. Задать цвет поверхности объекта равным RGB(21, 234, 21).
14. Задать параметры поверхности объекта равными:
Ka = 0,5; Kd = 0,5; Ks = 0,5; Ph = 3; Kr = 0; Kt = 0.
14. Добавить объект rg в объекты сцены.

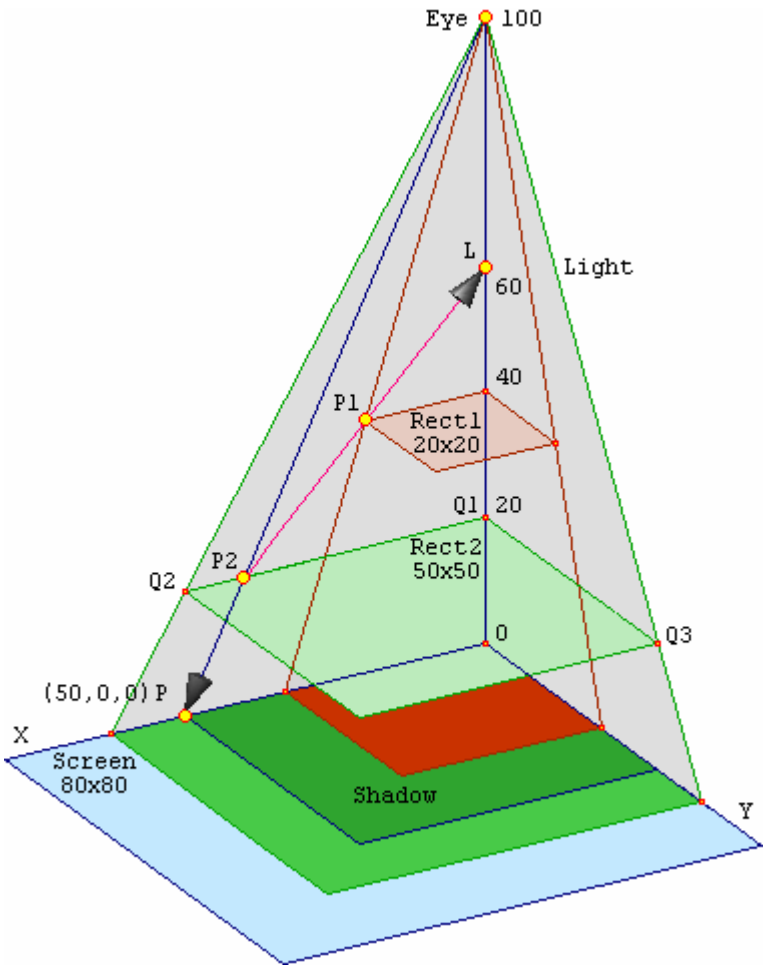

Рисунок 10 — Сцена 1

В дальнейшем в сцену будет добавлен еще один прямоугольник.

5.3.1. Контроль параметров прямоугольника

Устанавливаем точку остановки в конструкторе прямоугольника и проверяем правильность вычислений:

id	1
A	(0, 0, 20)
B	(50, 0, 20)
C	(0, 50, 20)
E1	(50, 0, 0)
E2	(0, 50, 0)
N	(0, 0, 1) после нормализации
S11	2500
S12	0
S22	2500
D	6250000
KU	(0.01, 0, 0)
KV	(0, 0.02, 0)
U0	0
V0	0

5.4. Расчет первичных лучей

Первичные лучи выпускаются из точки зрения в пиксели экрана.

Метод Scene::render модуля render.h.

```
// трассировка сцены
void Scene::render(HDC hdc, double XAngle, double ZAngle, . . .) {
    // трассирующий луч
    Ray ray;
    // цвет пикселя
    Color color;
}
```

Сначала переводим углы точки зрения в радианы, и вычисляем синусы и косинусы углов:

```
    // углы
    double A = XAngle * PI180;
    double B = ZAngle * PI180;
    // синусы и косинусы
    double SX = sin(A);
    double CX = cos(A);
    double SZ = sin(B);
    double CZ = cos(B);
```

Вычисляем положение точки зрения в мировой системе координат:

```
    // точка зрения в МСК
    double X0 = Plane * SZ * CX;
    double Y0 = Plane * SZ * SX;
    double Z0 = Plane * CZ;
```

Задаем точку начала луча и его начальный вес:

```
    // начало луча и начальный вес
    ray.org.values(X0, Y0, Z0);
    ray.weight = 1;
```

Описываем экранные координаты:

```
    // экранные координаты
    int YY, XX;
    double YS, XS, Y;
    double Z = Plane - distance;
```

Далее нужно построить цикл по линиям растра и пикселям линий, и преобразовать экранные координаты в мировые. Определяем направление луча, нормализуем его и трассируем при помощи метода trace. Этот метод возвращает цвет точки экрана, которую мы рисуем в устройство вывода:

```
    // пересчитываем в мировые
    for (YY = 0; YY <= LASTY; YY++) {
        YS = (double)(HOMEY - YY);
        Y = -YS * CZ - Z * SZ;
        for (XX = 0; XX <= LASTX; XX++) {
            XS = (double)(XX - HOMEX);
            // направление трассирующего луча
            ray.dir.X = Y * CX - XS * SX - X0;
            ray.dir.Y = XS * CX + Y * SX - Y0;
            ray.dir.Z = YS * SZ - Z * CZ - Z0;
            // нормализуем направление
            ray.dir.normalize();
            // трассируем луч и получаем цвет пикселя
            color = trace(ray);
            // выводим пиксель на экран
            SetPixel(hdc, XX, YY, color.getRGB());
        }
    }
```

Метод trace на данном этапе возвращает цвет фона:

```
Color Scene::trace(Ray ray) {
    // вычисляемый цвет
    Color color = background;
    return color;
}
```

Программа должна выводить прямоугольник цвета фона (неба).

5.4.1. Контрольный луч

Сейчас нужно выпустить один контрольный луч в точку (50, 0, 0), рисунок 10. Для этого нужно изменить код, выпускающий лучи:

```
void Scene::render(HDC hdc, double XAngle, double ZAngle, . . .) {
    Ray ray;
    Color color;
    // контрольный луч
    ray.org.values(0, 0, 100);
    ray.dir.X = 50;
    ray.dir.Y = 0;
    ray.dir.Z = -100;
    ray.dir.normalize();
    ray.weight = 1;
    color = trace(ray);
    return;
    . . .
```

Теперь в методе trace нужно вычислить точку P2 (на рисунке 10) пересечения луча с плоскостью прямоугольника:

```
Color Scene::trace(Ray ray) {
    // вычисляемый цвет
    Color color = background;
    // точка пересечения
    Hit hit;
    objects[0]->intersect(ray, hit);
    return color;
}
```

Ставим точку остановки в функции intersect класса Rect и проверяем контрольные значения.

cosVN	–0.894
hit.dist	89.44
hit.point	(40, 0, 20)
U	0.8
V	0

Луч пересекает прямоугольник в точке (40, 0, 20) с косинусом между лучом и нормалью –0,894. Это соответствует расчетной схеме.

Теперь нужно вычислить освещенность в точке P2.

Будем вычислять первичную освещенность в соответствии с начальной частью формулы Уиттеда следующим образом:

$B = B_A + B_D + B_S$

Здесь:

B — цвет точки, возвращаемый методом trace;

B_A = ambient · C · Ka — фоновая освещенность (цвет);

B_D = LC · C · (Kd · SH · LN) — диффузная освещенность (цвет);

B_S = LC · (Ks · SH · pow(HN, Ph)) — зеркальная освещенность (цвет).

В формулах:

C — цвет поверхности (берется из hit);

LC — цвет источника света;

SH — затенение источника света;

LN — косинус угла между вектором на источник света и нормалью;

HN — косинус угла между нормалью микрограни и нормалью.

Затенение для случая, когда источник света виден из точки пересечения напрямую, вычисляется по формуле:

```
double SH = lights[0].scale / dist;
```

Здесь scale — это свойство объекта light, dist — это расстояние от точки до источника света.

Нормаль микрограни вычисляется по формуле:

```
H = L - ray.dir;
```

где L — это нормализованный вектор из точки P2 на источник света.

Записываем все эти формулы непосредственно в методе trace после определения наличия пересечения:

```
if (hit.intersect) {
    // вычисление освещенности
}
return color;
```

Для контрольного луча вычисления должны давать примерно следующие значения:

B_A	(10, 117, 10)
L	(–40, 0, 40)
dist	56.57
L	(–0.707, 0, 0.707) — после нормализации
SH	30 / 56.57 = 0.53
LN	0.707
B_D	(4, 44, 4)
H	(–1.154, 0, 1.601)
H	(–0.584, 0, 0.811) — после нормализации
HN	0.811
B_S	(36, 36, 36)
B	(50, 197, 50)

Значения могут немного отличаться от приведенных.

Заметим, что сначала переменной B присваивается значение фоновой освещенности, а вычисляемые значения B_S и B_D прибавляются при помощи операции += (в классе Color нет другой подходящей операции).

Если теперь закомментировать код, формирующий контрольный луч, то получим примерно следующую картинку трассировку зеленого прямоугольника на фоне неба (рисунок 11).

Рисунок 11 — Трассировка зеленого прямоугольника

5.4.2. Методы shade и shadow

На самом деле все несколько сложнее. Во-первых, источников света может быть несколько. Это значит, что нужно конструировать цикл по всем источникам света, и вычислять привнесенную ими освещенность, которую добавлять к освещенности точки.

Во-вторых, объектов также может быть несколько. Это значит, что объекты могут заслонять собой источники света, но при этом могут пропускать часть света через себя.

Эти соображения усложняют вычисление освещенности, вносимой первичным лучом. Поэтому в классе Scene есть еще два метода.

Метод shade вычисляет освещенность по приведенным выше формулам, с учетом множества источников света.

Метод shadow вычисляет затенение источника света SH, с учетом множества объектов сцены.

Сначала рассмотрим схему метода shadow.

Параметры метода: луч на источник света, расстояние до источника света, начальное затенение источника света, идентификатор объекта, которому принадлежит точка пересечения.

Переменные метода: Hit hit — объект пересечения.

Функция формирует цикл по объектам сцены.

Если идентификатор объекта совпадает с идентификатором, передаваемым в качестве параметра, объект не рассматривается, иначе вычисляется пересечение.

Если расстояние до точки пересечения больше EPS и это расстояние меньше расстояния до источника света, то начальное затенение источника света atten умножается на коэффициент Kt поверхности пересечения. Если результат умножения менее EPS, затенение atten принимается равным нулю, и, в принципе, цикл можно завершить. Метод возвращает atten.

Перейдем к методу shade.

Именно в ней вычисляется освещенность в точке пересечения луча с объектом по формуле Уиттеда. Параметры этой функции — точка пересечения hit и вектор наблюдения view.

Для удобства все используемые значения записываются в локальные переменные, и их насчитывается достаточно много:

```
// вычисляемый цвет
Color B;
// параметры поверхности
double Ka, Ks, Kd, Ph, Kr, Kt;
double ETA; // отношение коэффициентов преломления
double CV;  // косинус угла между нормалью и вектором наблюдения
double SH;  // затенение источника света
double LN;  // косинус угла между нормалью и вектором источника
double HN;  // косинус угла между нормалью и нормалью микрограни
double VN;  // косинус угла между нормалью и вектором наблюдения
Medium M;   // среда объекта
Point N;// нормаль
Point H;// нормаль микрограни
Color C;// цвет поверхности
Color LC;   // цвет источника света
Color BA;   // цвет рассеянного света
Color BD;   // цвет диффузного отражения
Color BS;   // цвет зеркального отражения
Color BR;   // цвет отраженного луча
Color BT;   // цвет преломленного луча
Point L;// направление на источник
Ray ray;// новый трассирующий луч
```

Сначала нужно получить значения переменных Ka, Kd, Ks, Kr, Kt, Ph, M, C, N и VN, которые записаны в объекте hit.

Далее может быть вычислена доля освещенности от окружающего цвета, обозначаемая в формуле как B_A, значение которой сразу может быть присвоено результирующему цвету B.

Затем в методе на данном этапе только цикл по источникам света, в рамках которого вычисляется освещенность, привносимая первичным лучом, первичная освещенность.

Сначала нужно определить вектор на источник света L, вычислить расстояние до него dist, и записать в объект ray точку пересечения hit.point как начало луча, а нормализованный вектор L как направление.

Вычисляем начальное затенение atten по формуле scale / dist.

Вычисленные значения требуются для вызова метода shadow, который теперь нужно вызвать, а результат присвоить переменной SH.

Если полученное значение SH больше EPS, то можно вычислять значения освещенностей B_D и B_S. Заметим, что при вычислении составляющей B_D нужно контролировать значение косинуса LN. Если косинус имеет значение большее EPS, то составляющая B_D присутствует, иначе диффузного рассеяния нет. Аналогично, при вычислении составляющей B_S нужно контролировать значение косинуса HN. Если косинус больше EPS, составляющая B_S есть, иначе зеркальное отражение отсутствует.

Все составляющие добавляются к переменной B, которая возвращается методом shade.

5.4.3. Метод trace

Что же остается в методе trace.

В нем нужно найти ближайший объект, который пересекается трассирующим лучом, и ближайшее пересечение nearest, вычислить освещенность, которую несет луч, при помощи метода shade, и уточнить цвет, умножая его на экспоненту затухания.

Сначала нужно найти ближайший пересекаемый объект. Предположим, что расстояние до объекта чрезвычайно велико, например, такое:

```
// расстояние до объекта
double dist = 1e+20;
```

Формируем цикл по объектам, и вычисляем пересечение с каждым из них. Если точка пересечения есть, и расстояние до точки пересечения больше значения EPS, то сравниваем это расстояние с dist.

Если расстояние меньше dist, значит, этот объект ближе, и тогда запоминаем dist, как новое ближайшее расстояние, а точку пересечения hit запоминаем как новую ближайшую точку пересечения nearest.

После проверки всех объектов либо есть точка пересечения nearest, либо нет. Если точка пересечения есть (nearest.intersect истинно), вычисляем цвет color при помощи метода shade, объекта nearest и луча ray.

Полученный цвет нужно скорректировать. Луч распространяется в среде Medium, которая записана в луче как свойство medium. У среды есть затухание atten. Это учитывает следующая формула:

$$color \mathrel{*}= e^{-atten \cdot dist}$$

Если значение ray.medium.atten больше EPS, то уточняем значение возвращаемого цвета по данной формуле, иначе цвет остается таким, какой был получен из метода shade.

Поскольку метод trace вызывается рекурсивно, для ограничения количества вторичных лучей нужно увеличивать текущую глубину level при входе в метод, и уменьшать при выходе из него.

Если все расчеты правильно описаны, то результат трассирования сцены должен остаться прежним. В этом случае мы добавляем в сцену еще один прямоугольник размером 50×50, начальная точка (0, 0, 40), прямоугольник расположен на высоте 40 (рисунок 10), цвет RGB(234, 21, 21), название переменной для объекта rr, добавляем его перед прямоугольником rg. Результат трассирования сцены показан на рисунке 12.

Рисунок 12 — Трассировка сцены 1

5.5. Трассировка отраженным лучом

Для трассировки отраженного луча используем сцену 2. Она состоит из зеленого прямоугольника, который расположен так же, как в сцене 1, и красного прямоугольника, который расположен перпендикулярно зеленому прямоугольнику в плоскости, параллельной плоскости *ZY* (рисунок 13).

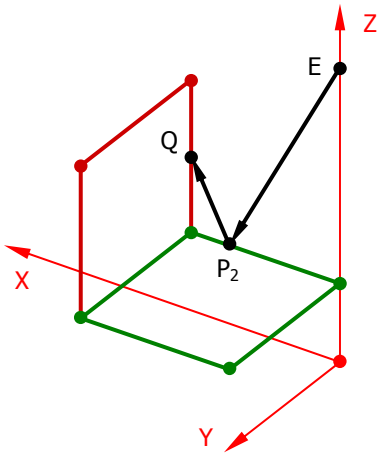

Рисунок 13 — Сцена 2

Копируем код scene1 в scene2 и изменяем следующие значения: LASTX и LASTY равны 120, plane равно 180, максимальная глубина трассировки 3. Красный треугольник задается теперь точками (50, 0, 20), (50, 0, 70) и (50, 50, 20). Коэффициент Kr зеленого прямоугольника равен 0,5.

Расчетная схема для контрольного луча показана на рисунке 14.

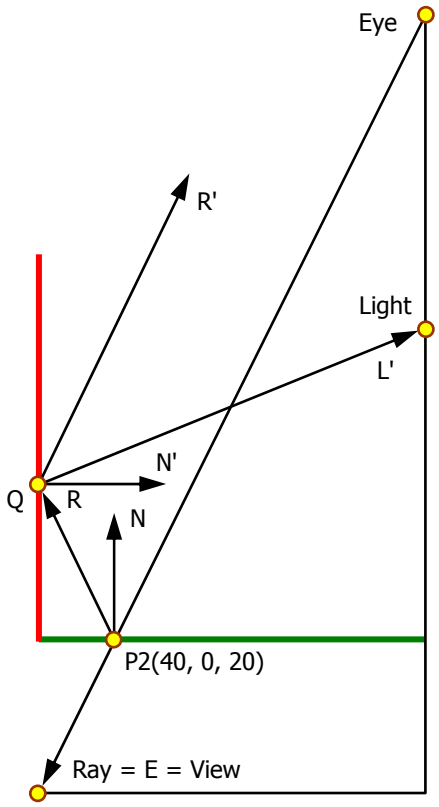

Рисунок 14 — Расчет отраженного луча

Контрольный первичный луч отражается в точке P2 зеленого прямоугольника, и попадает в точку Q красного прямоугольника (луч R), и в этой точке отражается в пространство (луч R').

Расчет вторичных лучей (отраженного и преломленного) выполняется в методе shade после расчета первичного луча. Сначала нужно убедиться, что глубина трассировки не превышает заданное значение:

```
// ПЕРВИЧНАЯ ОСВЕЩЕННОСТЬ
. . .
// ВТОРИЧНАЯ ОСВЕЩЕННОСТЬ
if (level < max_level) {
    // расчет вторичной освещенности
    ray.org = hit.point;
    // расчет отраженного луча
    // расчет преломленного луча
}
```

Далее вычисляется вес отраженного луча ray. Для этого значение веса из луча view копируется в свойство weight луча ray и умножается на Kr. Если полученный вес превышает EPS, то расчет производится, иначе нет:

```
if (level < max_level) {
    . . .
    // расчет отраженного луча
    ray.weight = view.weight * Kr;
    if (ray.weight > 0.001) {
    }
}
```

Точка начала луча совпадает с точкой пересечения объекта hit.

Направление луча вычисляется по формуле

ray.dir = view.dir – N·(VN·VN).

Полученное направление требуется нормализовать.

Свойство medium копируется в луч ray из луча view.

После этого сцена трассируется лучом ray, результат умножается на Kr и добавляется к рассчитываемому значению B.

Восстанавливаем в методе render контрольный луч, и проверяем вычисляемые значения отраженного луча в методе shade:

ray.weight	0,5
ray.org	(40, 0, 20)
ray.dir	(0.447, 0, 0.984)
B_R	(236, 73, 73)

После умножения на Kr получим B_R = (118, 36, 36).

Если получен не тот цвет, следует проверить, как вторичный луч трассирует сцену и вычисляет освещенность точки Q. Более детально этот процесс описан в документе Tracing.pdf.

Если все выполнено верно, то трассировка всего экрана дает примерно следующую картинку (ось X направлена вниз, ось Y вправо):

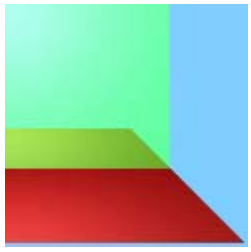

Рисунок 15 — Трассировка сцены 2

5.6. Трассировка преломленным лучом

Для проверки преломленного луча требуется прозрачный объект.

В качестве такого объекта будем использовать стеклянный кубик.

Поэтому нужно описать формирование параметров параллелепипеда и вычисление точки пересечения трассирующего луча с одной из его граней.

Модуль object.h.

Описываем конструктор объекта Box:

```
// строит объект, вычисляет параметры
Box(Point A, Point X, Point Y, Point Z) {
    Loc = A;
    Point E1 = X - Loc;
    Point E2 = Y - Loc;
    Point E3 = Z - Loc;
    // центральная точка объекта
    C = Loc + (E1 + E2 + E3) * 0.5;
    // нормали граней
    N[0] = (E1 ^ E2).normalize();
    N[1] = (E1 ^ E3).normalize();
    N[2] = (E2 ^ E3).normalize();
    D1[0] = -(N[0] * Loc);
    D2[0] = -(N[0] * (Loc + E3));
    D1[1] = -(N[1] * Loc);
    D2[1] = -(N[1] * (Loc + E2));
    D1[2] = -(N[2] * Loc);
    D2[2] = -(N[2] * (Loc + E1));
    for (int i = 0; i < 3; i++) {
        if (D1[i] > D2[i]) {
            D1[i] = -D1[i];
            D2[i] = -D2[i];
            N[i].negate();
        }
    }
}
```

Далее описываем, как объект вычисляет точку пересечения, более подробно см. [1].

```
// вычисляет пересечение объекта с заданным лучом
int intersect(Ray ray, Hit & hit) {
    double VD = 0, VO = 0, T1 = 0, T2 = 0;
    int index = 0;
    double TNear = -1e+20, TFar = 1e+20;
    hit.clear();
    for (int i = 0; i < 3; i++) {
        VD = ray.dir * N[i];
        VO = ray.org * N[i];
        if (VD > 0.001) {
            T1 = -(VO + D2[i]) / VD;
            T2 = -(VO + D1[i]) / VD;
        } else if (VD < -0.001) {
            T1 = -(VO + D1[i]) / VD;
            T2 = -(VO + D2[i]) / VD;
        } else if (D1[i] > VO || VO > D2[i]) {
            return 0;
        } else {
            continue;
        }
        if (T1 > TNear) {
            TNear = T1;
            index = i;
        }
        if (TFar > T2) TFar = T2;
        if (TFar < 0.00001) return 0;
        if (TNear > TFar) return 0;
    }
```

```
        // КОРРЕКЦИЯ
        if (TNear < 0.00001) {
            hit.dist = TFar;
        } else {
            hit.dist = TNear;
        }
        if (hit.dist > 0.00001) {
            hit.intersect = 1;
            hit.id = id;
            hit.surface = surface;
            hit.point = ray.point_at(hit.dist);
            Point n = N[index];
            if ((hit.point - C) * n < 0) {
                n.negate();
            }
            double cosVN = ray.dir * n;
            if (cosVN < 0) {
                hit.cosVN = cosVN;
                hit.normal = n;
                hit.entering = 1;
            } else {
                hit.cosVN = -cosVN;
                hit.normal = -n;
                hit.entering = 0;
            }
            return 1;
        }
        return 0;
    }
```

Теперь нужно сформировать сцену 3.

Она состоит из оранжевого прямоугольника размером 90×90, расположенного в плоскости $Y = -30$, и голубого куба размером 40×40×40, средняя точка которого расположена в центре системы координат.

Параметры сцены 3 такие:

LASTX = LASTY = GDE_AREA

HOMEX = GDEC

HOMEY = GDEC – 20

anglex = 30

anglez = 45

plane = 360

distance = 100

максимальная глубина трассировки = 5

фон (background) = (32, 64, 140)

рассеянный свет (ambient) = (255, 255, 255)

Один источник света, цвет белый, точка источника света (–45, –45, 0), параметр scale = 70.

Далее описываем объект Box, переменная bx.

Box * bb = new Box(...)

Точки, задаваемые в конструкторе, следующие:

A = (–20, –20, –20)

X = (20, –20, –20)

Y = (–20, 20, –20)
Z = (–20, –20, 20)
Цвет куба равен (64, 128, 255)
Коэффициенты:
Ka = 0,3, Kd = 0,3, Ks = 0,3, Ph = 30, Kr = 0,5, Kt = 0,5.
Среда medium = GlassMedium.
Добавляем объект в сцену методом object_add.
Далее описываем прямоугольник, переменная rr.
Rect * rr = new Rect(…)
Точки, задаваемые в конструкторе, следующие:
A = (–45, –45, –30)
B = (45, –45, –30)
C = (–45, 45, –30)
Цвет прямоугольника равен (255, 128, 64)
Коэффициенты:
Ka = 0,3, Kd = 0,3, Ks = 0,3, Ph = 30, Kr = 0,5, Kt = 0.
Добавляем объект в сцену методом object_add.

В методе shade после расчета отраженного луча рассчитываем преломленный луч. Сначала проверяем вес луча, так же, как и в случае с отраженным лучом:

```
// расчет преломленного луча
ray.weight = view.weight * Kt;
if (ray.weight > 0.001) {
}
```

Далее проверяем, входит луч в объект, или выходит из него.

Если входит, среда medium луча равна среде трассирующего луча view, иначе среда принимается равной AirMedium. Кроме того, рассчитываем отношение коэффициентов преломления:

```
// отношение коэффициентов преломления
if (hit.entering) {
    ETA = view.medium.refract / M.refract;
    ray.medium = M;
} else {
    ETA = M.refract / view.medium.refract;
    ray.medium = AirMedium;
}
```

Далее вычисляем косинус угла между лучом и нормалью и вычисляем дискриминант формулы Снеллиуса:

```
// -cos(V,N)
CV = -VN;
// дискриминант формулы Снеллиуса
double D = 1 + ETA * ETA * (CV * CV - 1);
```

Если значение D превышает значение EPS, то преломление есть и тогда рассчитываем нормаль и направление преломленного луча:

```
if (D > 0.001) {
    N *= (ETA * CV - sqrt(D));
    // направление преломленного луча
    ray.dir = ((view.dir * ETA) + N).normalize();
```

Чтобы трассирующий луч ray находился не на поверхности объекта, немного смещаем его в объект (или из объекта):

```
// КОРРЕКЦИЯ
// луч входит в тело (или выходит из него)
// чтобы получить начало луча в теле (вне тела)
ray.org += (ray.dir / 2);
```

Наконец, трассируем сцену преломленным лучом, результат умножаем на Kt, и добавляем к рассчитываемому цвету:

```
// трассирование сцены преломленным лучом
BT = trace(ray) * Kt;
// операции разделены для контроля
// добавляем пришедший по преломленному лучу цвет
B += BT;
```

Расчетной схемы нет, поэтому полагаемся на то, что все рассчитывается правильно. Примерный вид результата трассировки сцены 3 показан на рисунке 16:

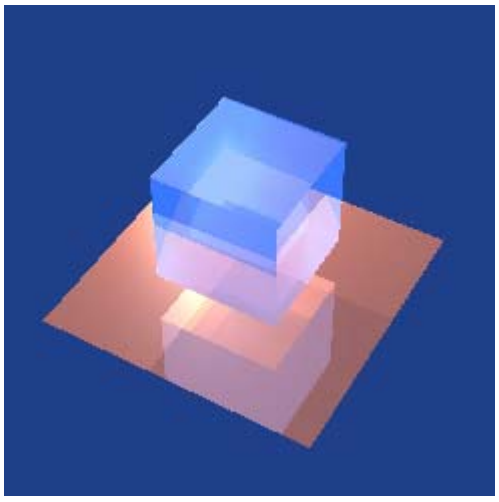

Рисунок 16

Литература

1. Е.В. Шикин, А.В. Боресков. Компьютерная графика. Динамика, реалистические изображения. М.: "Диалог-МИФИ", 1995. — 288 с.

Владимир Вадимович Пономарев
Практикум по компьютерной графике
Учебно-методическое пособие

Отпечатано с готового оригинал-макета
Издательство ОТИ НИЯУ МИФИ, 2018
Тираж 11 экз.