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

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по компьютерной графике

Учебно-методическое пособие
по дисциплине «Инженерная и компьютерная графика»

Часть 4. Визуализация геометрических моделей

2017 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по компьютерной графике. Учебно-методическое пособие по дисциплине «Инженерная и компьютерная графика». Часть 4. Визуализация геометрических моделей. Озерск: ОТИ НИЯУ МИФИ, 2017. — 42 с.

В пособии излагается, как выполнять практические работы по дисциплине «Инженерная и компьютерная графика». Работы четвертой части изучения дисциплины включают в себя методы визуализации геометрических моделей: закраску граней методами 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 4
1. Каркасная модель 5
1.1. Шаблон проекта render 5
1.2. Классы и объекты проекта 5
1.2.1. Класс point 5
1.2.2. Класс imatrix 5
1.2.3. Класс matrix 5
1.2.4. Класс triang 6
1.2.5. Класс triangs 6
1.3. Описание объекта параллелепипед 6
1.4. Рисование граней 7
1.5. Удаление невидимых граней 8
2. Расчет освещенности в вершинах граней 9
2.1. Составляющие освещенности 9
2.2. Алгоритм вычисления освещенности 10
2.3. Освещенность в вершинах 11
2.4. Расчетная схема 11
3. Закраска методом Гуро 14
3.1. Интерполирование вдоль линии растра 14
3.2. Вращение вершин 15
3.3. Интерполяция освещенности вдоль ребер грани 16
4. Закраска гладкого объекта 19
4.1. Построение полигональной модели цилиндра 19
4.2. Вектор нормали в вершине 21
5. Метод обратной трассировки лучей 22
5.1. Шаблон проекта tracing 22
5.1.1. Классы модуля surf 22
5.1.2. Классы модуля gras 22
5.1.3. Классы модуля scene 23
5.1.4. Модуль object 24
5.2. Построение геометрической модели прямоугольника 24
5.2.1. Расчет параметров прямоугольника 25
5.2.2. Расчет точки пересечения 25
5.3. Построение сцены 26
5.3.1. Контроль параметров прямоугольника 28
5.4. Расчет первичных лучей 28
5.4.1. Контрольный луч 30
5.4.2. Функции shade и shadow 32
5.4.3. Функция trace 34
5.5. Трассировка отраженным лучом 35
5.6. Трассировка преломленным лучом 37

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) Метод обратной трассировки лучей.

1. Каркасная модель

Цели:
- изучение представления полигональных моделей.

Задачи:
- определение графических классов;
- вывод каркасной модели;
- вывод видимых граней модели;

1.1. Шаблон проекта render

Для выполнения работы требуется шаблон проекта render.

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

Модуль grasses.h предназначен для описания графических классов.

Модуль grobject.h предназначен для описания графических объектов.

В модуле render.h описываются константы и переменные.

В начале модуля render.cpp находится функция draw_view, которая вызывается автоматически при изменении точки зрения на сцену, после чего происходит автоматическое обновление изображения.

Сцена описывается в функции scene, определенной в модуле grobject.h.

Конкретные графические объекты, параллелепипед и цилиндр, описываются в функциях box и cylinder модуля grobject.h. Для контроля расчетной схемы здесь же описывается сцена из одного треугольника.

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.

Метод values задает координаты.

Метод length вычисляет длину вектора.

В классе заданы также несколько операций.

1.2.2. Класс imatrix

Класс imatrix является интерфейсом матрицы координат.

1.2.3. Класс matrix

Шаблон класса matrix предназначен для создания матриц координат вершин полигонального объекта с заданным количеством точек. Количество точек должно быть задано константой POINTS модуля grobject.h.

Класс matrix определяет видовое преобразование, совмещенное с простой перспективной проекцией.

Точка зрения на объект задается в сферических координатах при помощи переменных XYAngle, ZAngle и Plane, описанных в модуле render.h.

Перевод сферических координат точки зрения в декартовы координаты выполняет функция GetViewPoint. Вектор точки зрения view определен в модуле grasses.h. Вектор источника света light определен в модуле grasses.h, а задается в функции scene модуля grobject.h.

1.2.4. Класс triang

Основные действия происходят в этом классе.

В ходе выполнения практических работ в этот класс должны быть добавлены необходимые элементы данных и методы.

Треугольная грань задается своими вершинами.

Массив вершин V[3] хранит номера вершин в матрице координат.

Метод draw_edges предназначен для рисования ребер граней.

1.2.5. Класс triangs

Этот класс описывает все треугольные грани сцены. Максимальное количество граней ограничено константой MAX_TRIS, значение которой должно уточняться при задании конкретного объекта.

Ниже описаний классов в модуле grasses.h расположены объекты:

light, view — положение источника света и точки зрения,

tris — треугольники сцены,

WCO, ECO — матрицы мировых и экранных координат.

1.3. Описание объекта параллелепипед

Описание объекта параллелепипед выполняется в функции box модуля grobject.h. В функции scene эта функция должна быть вызвана.

Лучше всего иметь рисунок объекта (рисунок 1).

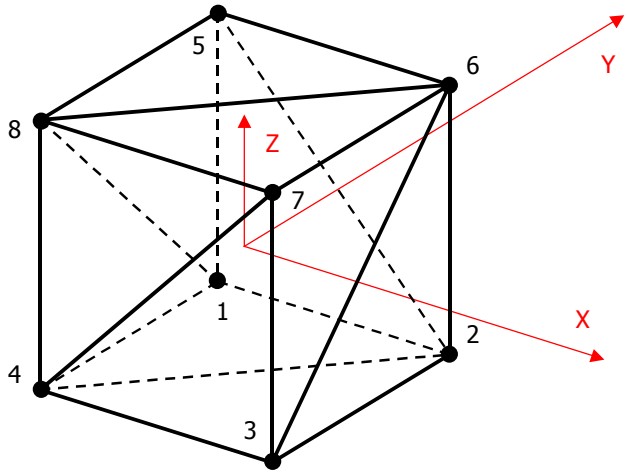


Рисунок 1 — Расчетная схема параллелепипеда

Каждой вершине присваивается номер. Центр системы координат находится в центре объекта. Половины сторон объекта задаем константами BOX_X, BOX_Y, BOX_Z.

Вершина 1, например, задается следующим программным кодом:

```
(*WCO)[1].values(-BOX_X,  BOX_Y, -BOX_Z);
```

Аналогичным образом задаются другие вершины.

Какая вершина задается первой, значения не имеет.

После задания всех вершин нужно задать треугольные грани.

Грань задается номерами вершин, указываемыми в порядке против часовой стрелки, если смотреть на лицо грани. Например, нижняя грань, вершины которой пронумерованы 1, 2 и 4, задается следующим образом:

```
tris.add().values(1, 2, 4);
```

Какая вершина грани задается первой, значения не имеет.

Какая грань задается первой, значения также не имеет.

1.4. Рисование граней

Чтобы проверить правильность задания граней, их нужно нарисовать.

Рисование выполняет функция draw_edges класса triang, параметры которой

- контекст устройства hdc,
- матрица видовых координат M,
- смещение изображения к центру устройства вывода CX и CY.

Для рисования используются функции MoveToEx и LineTo. Например, чтобы нарисовать грань от первой вершины до второй, можно использовать следующий код:

```
void draw_edges(HDC hdc, imatrix * M, int CX, int CY) {
    int x1, y1, x2, y2;
    x1 = (int)(*M)[V[0]].X + CX;
    y1 = (int)(*M)[V[0]].Y + CY;
    x2 = (int)(*M)[V[1]].X + CX;
    y2 = (int)(*M)[V[1]].Y + CY;
    MoveToEx(hdc, x1, y1, 0);
    LineTo(hdc, x2, y2);
}
```

После этого переходим в функцию draw_view модуля render.cpp, и описываем действия, необходимые для построения изображения.

Сначала нужно выполнить преобразование координат из матрицы мировых координат WCO в матрицу видовых координат ECO при помощи метода perspective. Метод вызывается с матрицей ECO, а матрица WCO подставляется как параметр. Другие параметры — сферические координаты точки зрения.

Затем нужно сформировать цикл по всем граням объекта tris, и вызывать для каждой грани метод draw_edges.

В метод подставляются контекст устройства hdc, матрица видовых координат, и два раза константа GDEC, равная половине ширины и высоты устройства вывода.

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 положительное значение в мировой системе координат, и отрицательное значение в видовой системе координат. 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. Вершины, таким образом, обозначены здесь буквами A, B и C.

Рассчитываем вектор нормали в методе normal, после чего расчет координаты Z копируем в метод normalZ.

Затем переходим в функцию draw_view и добавляем в нее условие, при выполнении которого ребра грани рисуются: считаем, что грань видна, если координата Z вектора нормали строго меньше нуля. Заметим, что условие «строго больше» в данный момент также применимо.

Если при вращении объекта наблюдаются не все грани, или видны невидимые грани, то, скорее всего, задан неправильный порядок вершин.

2. 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.

$I_s = I_l\, k_s\, cos^{\mathrm{P}}\,\alpha / d$ — 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.

В программе яркость источника света задана переменной Intens.

Ориентировочные значения Intens — 30000…90000.

Произведение $I_l\, k_l$ задано переменной IntensA.

Ориентировочные значения IntensA — 100…400.

Коэффициенты, учитывающие свойства поверхности, задаются как свойства объекта triang под именами Ka, Kd, Ks, Ph. Под этими же именами в программе определены переменные, задающие следующие ориентировочные значения:

Ka ≈ 0,3

Ks ≈ 0,4

Kd ≈ 0,4

Ph ≈ 100

Эти значения должны передаваться объектам triang в функции scene модуля grobject.h.

2.2. Алгоритм вычисления освещенности

При разработке алгоритма вычисления освещенности в точке следует учитывать:

- если косинус угла θ меньше или равен нулю, свет от источника света вообще не падает на грань, следовательно, освещенность точки состоит только из рассеянной в пространстве освещенности I_a;
- если косинус угла θ больше нуля, то следует учитывать диффузное рассеяние I_d;
- если косинус угла θ больше нуля, и косинус угла α больше нуля, то следует учитывать отраженный свет I_s.

Вычисление освещенности точки описываем в функции gouraud_intens.

Вычисления производятся в мировой системе координат.

Параметры функции:

- точка p;
- вектор нормали N;
- положение источника света l;
- положение точки зрения s;
- яркость источника света Io;
- рассеянная освещенность Ia.

Для расчета векторов в классе point нужно определить метод, вычисляющий скалярное произведение. Название метода scalar, метод возвращает сумму произведений координат векторов. Для нормализации векторов требуется метод normalize, который делит координаты вектора на его длину.

Для расчета освещенности используются следующие переменные типа double:

I — рассчитываемая освещенность;

ia — составляющая I_a;

id — составляющая I_d;

is — составляющая I_s;

dist — расстояние до источника света;

cost — косинус угла θ;

cosa — косинус угла α;

Следующие переменные имеют тип point:

L — вектор на источник света;

R — вектор отражения;

S — вектор на точку зрения;

Вектор на источник света рассчитывается как разность l и p.

Вектор на точку зрения рассчитывается как разность s и p.

Вектор отражения рассчитывается так:

- принять вектор R равным вектору N;
- умножить R на 2 косинуса угла θ;
- вычесть из R вектор L.

Следует также понимать, что косинус угла между векторами равен скалярному произведению нормализованных векторов, поэтому векторы должны нормализоваться по мере необходимости.

Порядок вычислений примерно следующий:

- ia;
- L;
- dist;
- cost;
- R;
- S;
- cosa;
- is;
- I.

Рассчитанное значение освещенности I должно быть приведено к диапазону 0…255. Если, например, значение I превышает 255, оно должно быть принято равным 255 (цвет точки не может быть белее белого).

2.3. Освещенность в вершинах

Освещенность в вершинах грани следует описать в методе intens класса triang. Параметрами метода являются:

- матрица мировых координат W;
- положение источника света *l*;
- положение точки зрения s;
- яркость источника света Io;
- рассеянная освещенность Ia;
- коллекция треугольных граней t.

На данном этапе разработки коллекция треугольных граней не требуется, однако позднее она нам понадобится.

Сначала вычисляется нормаль грани TN, которая нормализуется методом normalize.

Затем формируется цикл по вершинам грани.

В цикле нормаль в вершине VN[i] принимается равной TN, после чего освещенность в вершине VI[i] вычисляется при помощи вспомогательной функции gouraud_intens, определенной в предыдущем разделе.

Метод получается таким вот простым.

В последующих работах он будет уточнен.

2.4. Расчетная схема

Проверку правильности вычисления освещенности нужно выполнять при помощи расчетной схемы, в которой все вычисления выполняются вручную с целью получения контрольных значений всех элементов расчетов (рисунок 2).

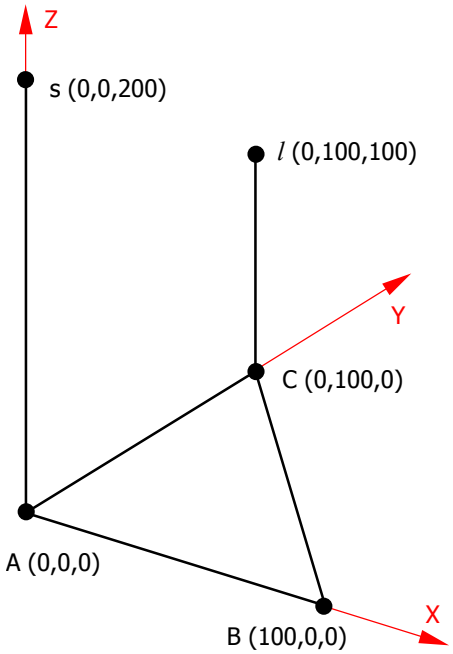


Рисунок 2 — Расчетная схема вычисления освещенности

Эта схема заложена в функции triangle модуля grobject.h, и сейчас вместо функции box в функции scene нужно вызвать функцию triangle.

Расчетные параметры:

Intens = 30000

IntensA = 240

XYAngle = 0

ZAngle = 0

Plane = 200

Ka = 0.3, Kd = 0.4, Ks = 0.4

Ph = 10

Вычисления для точки A (рисунок 3), плоскость YZ.

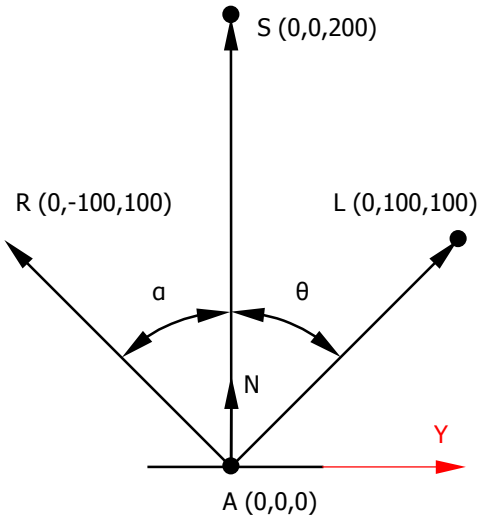


Рисунок 3 — Освещенность в точке A

Расчетные значения:
- *ambient* освещенность ia = 72
- вектор L = (0, 100, 100)
- расстояние до источника света dist = 141.42
- нормализованный вектор L = (0, 0.707, 0.707)
- косинус угла θ cost = 0.707 (45°)
- вектор отражения (нормализованный) R = (0, –0.707, 0.707)
- вектор точки зрения S = (0, 0, 200)
- вектор точки зрения нормализованный S = (0, 0, 1)
- косинус угла α cosa = 0.707 (45°)
- доля зеркального отражения is = 0.0125
- доля диффузного рассеяния Kd * cost = 0.28
- освещенность I = 134

Расчетные значения для точки B:
- *ambient* освещенность ia = 72
- вектор L = (–100, 100, 100)
- расстояние до источника света dist = 173.2
- нормализованный вектор L = (–0.577, 0.577, 0.577)
- косинус угла θ cost = 0.577 (54.7°)
- вектор отражения (нормализованный) R = (0.577, –0.577, 0.577)
- вектор точки зрения S = (–100, 0, 200)
- вектор точки зрения нормализованный S = (–0.447, 0, 0.894)
- косинус угла α cosa = 0.258 (75°)
- доля зеркального отражения is = 5.26e-7
- доля диффузного рассеяния Kd * cost = 0.23
- освещенность I = 112

Расчетные значения для точки C:
- *ambient* освещенность ia = 72
- вектор L = (0, 0, 100)
- расстояние до источника света dist = 100
- нормализованный вектор L = (0, 0, 1)
- косинус угла θ cost = 1 (0°)
- вектор отражения (нормализованный) R = (0, 0, 1)
- вектор точки зрения S = (0, –100, 200)
- вектор точки зрения нормализованный S = (0, –0.447, 0.894)
- косинус угла α cosa = 0.894 (26.6°)
- доля зеркального отражения is = 0.131
- доля диффузного рассеяния Kd * cost = 0.4
- освещенность I = 231

3. 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 4).

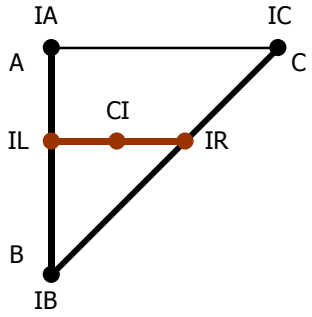


Рисунок 4 — Интерполяция освещенности точки

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:

```
// вычисляем цвет
int c = (int)CI;
COLORREF color = RGB(c, c, c);
// выводим точку
SetPixel(hdc, x, y, color);
```

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 (пикселей) в линии растра 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 равна 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. Заметим, что значение DI может быть положительным (левая точка темнее), отрицательным (левая точка светлее) и нулевым (точки имеют одинаковую освещенность).

3.2. Вращение вершин

Интерполяция вдоль ребер грани выполняется отдельно для каждого возможного случая ориентации грани отдельно. Рассмотрим одну из возможных ориентаций (рисунок 5, ориентация ABCB).

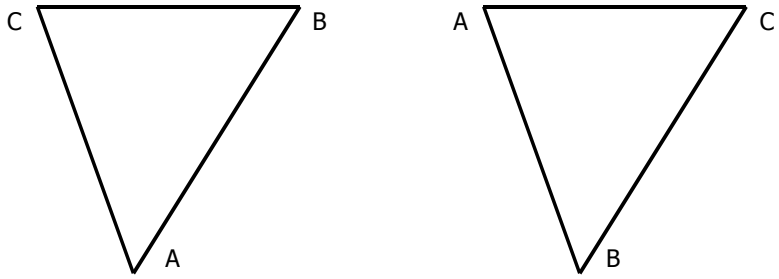


Рисунок 5 — Вращение вершин

Слева на рисунке 5 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 5 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 xc и yc вершины C. Соответственно, XB и YB присваиваем координаты xa и ya вершины A, а XC и YC присваиваем координаты xb и yb вершины B.

Таким образом мы как бы повернули вершины по часовой стрелке.

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

Чтобы выполнить вращение, нужно выяснить, какая вершина где находится после преобразования координат следующим образом:

```
if ((yc < ya) && (yc <= yb)) {
    //---- вращаем вершины вправо ---
} else if ((yb < yc) && (yb <= ya)) {
    //---- вращаем вершины влево ----
} else {
    //------ не вращаем вершины -----
}
```

Здесь xa, ya, xb, yb, xc и yc — это координаты вершин до поворота.

Если происходит вращение вправо, как на рисунке 5, координаты вершин после поворота будут следующими:

```
XA = CX + xc;
YA = CY + yc;
XB = CX + xa;
YB = CY + ya;
XC = CX + xb;
YC = CY + yb;
```

Заметим, что одновременно с вращением к координатам прибавляется смещение к центру экрана CX и CY.

Одновременно с вращением координат нужно также вращать освещенности в вершинах. Например, при повороте вправо, нужно записать в переменные IA, IB и IC следующие значения:

```
IA = VI[POINT_C];
IB = VI[POINT_A];
IC = VI[POINT_B];
```

Вращение выполняется в функции gouraud класса triang, в самом начале функции.

После вращения переменные XA и YA равны координаты верхней левой вершины, переменная IA — освещенности верхней левой вершины.

Соответственно, переменные XB и YB равны координатам вершины следующей по порядку против часовой стрелки, а IB равно освещенности в этой вершине.

И, наконец, переменные XC и YC равны координатам вершины, далее следующей по порядку против часовой стрелки, а IC равно освещенности в этой вершине.

3.3. Интерполяция освещенности вдоль ребер грани

После вращения вершин нужно выяснить действительную ориентацию грани, используя координаты вершин после вращения:

```
if (YA == YC) {
    // A   C
    //
    //   B
} else {
    // A - верхняя вершина
    if (YB == YC) {
        //   A
        //
        // B   C
    } else if (YB < YC) {
        //     A
        // B
        //     C
    } else {
        // A
        //     C
        // B
    }
}
```

Расчетной схеме соответствует ориентация, когда вверху находятся вершины A и C. Поэтому программируем этот случай.

Интерполяция вдоль ребер выполняется примерно также, 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 B и A (или B и C).

Далее вычисляются приращения DXL и DXR координат X по мере перемещения по линиям растра (по координате y).

Для рисунка 5 справа приращения могут быть вычислены следующим образом:

```
DXL = (double)((XB - XA) / (YB - YA));
DXR = (double)((XB - XC) / (YB - YC));
```

Заметим, что частное (YB - YA) или (YB - YC) может совпадать с количеством линий растра в определенной ориентации.

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

```
DIL = (IB - IA) / (YB - YA);
DIR = (IB - IC) / (YB - YC);
```

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.

В рамках одной итерации рисуется одна линия растра, соответствующая текущей координате y. Для рисования вызываем вспомогательную функцию gouraud_line, определенную ранее.

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 6.

Рисунок 6 — Закрашенный треугольник, ориентация ABCB

Далее переходим к программированию других ориентаций.

Другие ориентации на расчетной схеме можно получить, если вращать треугольник вокруг оси Z, изменяя угол XYAngle в функции triangle модуля grobject.h на 45°, 300° 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 7, точка соединения B).

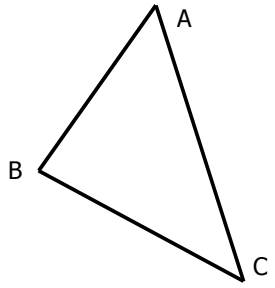


Рисунок 7 — Ориентация ACAC

Правая ветвь на рисунке 7 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.

4. 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

Модуль grobject.h. В функции scene вызываем функцию cylinder, переходим в эту функцию и выстраиваем модель. Для построения полигональной модели цилиндра используются константы модуля:

CYL_R — радиус основания;

CYL_H — половина высоты;

CYL_N — количество боковых граней.

Фактически должна получиться призма с количеством граней CYL_N.

На рисунке 8 показана примерная расчетная схема для нумерации вершин полигональной модели цилиндра при значении CYL_N, равном 5.

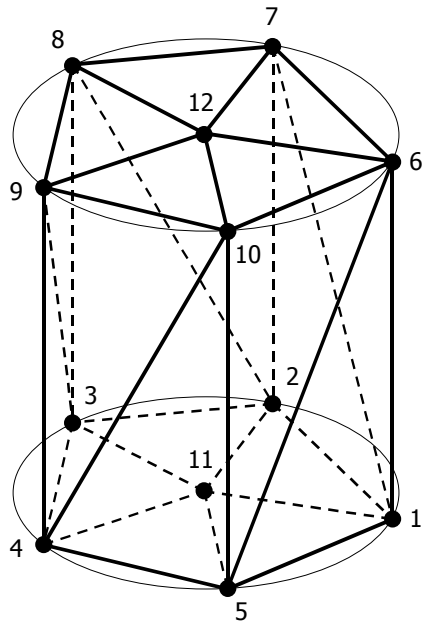


Рисунок 8 — Расчетная схема нумерации вершин цилиндра

В этой расчетной схеме первая вершина лежит на направлении оси X, и ее координаты равны (CYL_R, 0, –CYL_H). Сначала нумеруются вершины нижнего круга, затем верхнего круга, затем центры кругов.

Прежде, чем формировать вершины, нужно задать значение константы POINTS, которая определяет размер матрицы. Это значение является функцией от константы CYL_N. Уточните также константу MAX_TRIS.

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 и константы CYL_N.

Чтобы осилить эту сложность, нужна развертка модели (рисунок 9).

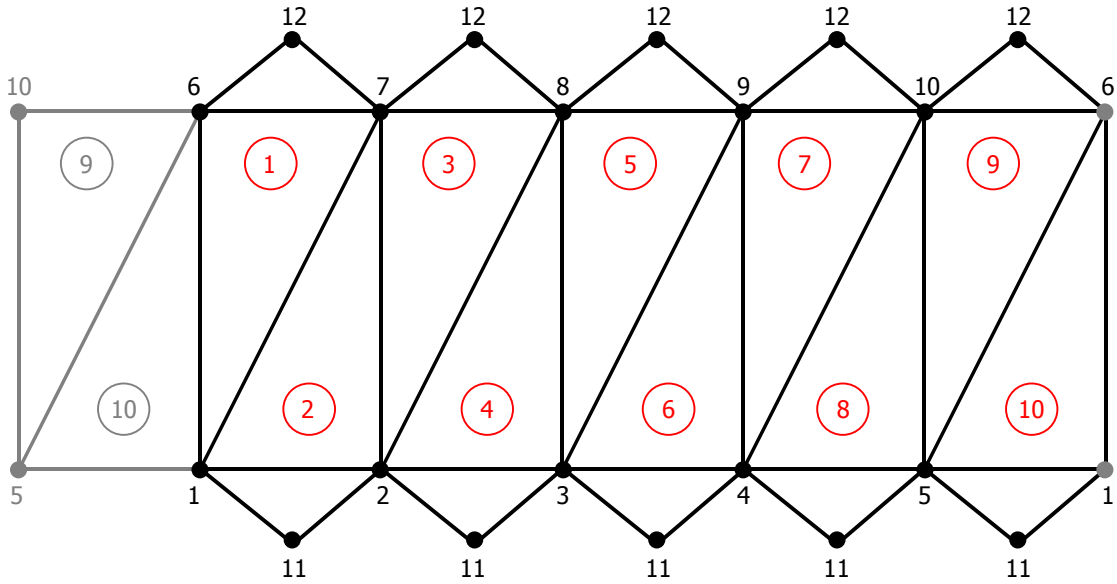


Рисунок 9 — Развертка модели цилиндра

Цифрами в кружках на рисунке 9 обозначены номера граней. Последовательность нумерации граней не обязательно такая, как указана на рисунке, может быть любая другая.

В процессе добавления граней нужно также указывать сопряженные грани вершин при помощи метода грани adjacent. Первым параметром метода является номер вершины POINT_A, POINT_B или POINT_C, вторым параметром является номер грани, которая является сопряженной для данной вершины. Если в вершине не одна, а больше сопряженных граней, метод adjacent вызывается столько раз, сколько нужно. В результате добавление грани в итерации цикла имеет примерно следующий вид:

```
tris.add().values(a, b, c).adjacent(POINT_A, d)
                          .adjacent(POINT_B, d)
                          .adjacent(POINT_B, e);
```

Задача заключается в том, чтобы вычислить переменные a, b и c, задающие номера вершин грани в порядке против часовой стрелки, а также переменные d и e, задающие номера сопряженных граней, так, чтобы они зависели от переменной цикла (номера итерации) и константы CYL_N.

После построения модели нужно убедиться в ее правильности, отключив закраску грани, и включив вывод ребер в функции draw_view.

4.2. Вектор нормали в вершине

После того, как будет получена правильная модель цилиндра, нужно включить закраску и отключить вывод ребер. В результате мы должны увидеть правильно закрашенную призму, а не цилиндр.

Чтобы получить закраску, соответствующую цилиндру, нужно перейти в метод triang::intens и вычислить нормаль в вершинах граней.

Сейчас метод имеет примерно следующий вид:

```
void triang::intens(. . .) {
    // МИРОВАЯ СИСТЕМА КООРДИНАТ
    point TN;
    triang p;
    // нормаль грани
    TN = normal(W);
    TN.normalize();
    // для каждой вершины из трех
    for (int i = 0; i < 3; i++) {
        // нормаль в вершине i
        VN[i] = TN;
        // освещенность в вершине i
        VI[i] = gouraud_intens((*W)[V[i]], VN[i], l, s, Io, Ia);
    }
}
```

Нормаль в вершине изначально принимается равной нормали самой грани. Теперь же нужно добавить цикл, в котором к этой нормали прибавляются нормали сопряженных граней. Количество сопряженных граней для вершины POINT_A определяется свойством грани adjc[POINT_A]. Номер сопряженной грани возвращает свойство adja[POINT_A][n], где n — порядковый номер сопряженной грани от нуля. Нормаль грани возвращает метод normal. Векторы нормалей должны нормализоваться до и после сложения.

5. Метод обратной трассировки лучей

Цели:
- изучение основ обратной трассировки лучей;

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

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 оконной части проекта;

tracing.cpp — оконная часть проекта, не требует редактирования;

surf.h — классы цвета, среды и поверхности;

gras.h — графические классы;

scene.h — класс сцены и методов трассировки;

object.h — классы графических объектов, построение сцен.

5.1.1. Классы модуля surf

Класс Color описывает цвет и операции с цветом.

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

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.

5.1.2. Классы модуля gras

Класс Point описывает точку или вектор.

В классе определены практически все необходимые операции. Следует обратить внимание, что скалярное произведение векторов определено как операция умножения *, а векторное произведение — как операция ^.

Класс Ray описывает луч. Свойства:

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:

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 Hits описывает множество точек пересечения. Отличается тем, что при добавлении новой точки пересечения (объекта Hit) все точки пересечения сортируются по расстоянию (свойству dist) так, что первой является точка пересечения с наименьшим расстоянием.

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.

5.1.3. Классы модуля scene

В этом модуле происходит основное программирование.

В модуле определен только один класс — класс сцены Scene.

Свойства сцены:

LASTX, LASTY — размеры экрана проецирования;

HOMEX, HOMEY — смещение к центру экрана;

distance — расстояние до экрана;

anglexy — 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. Модуль object

В этом модуле находится функция build_scene, в которой описывается построение сцены, и функция draw_view, которая рисует изображение.

В этом модуле должны определяться геометрические модели объектов в виде классов, производных от класса Object.

5.2. Построение геометрической модели прямоугольника

Модуль object.h.

Описываем геометрический объект «прямоугольник»:

```
// объект прямоугольная поверхность
class Rect : public Object {
    // начало объекта, нормаль, базис
    Point Loc, N, KU, KV;
    // начальная точка базиса
    double U0, V0;
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, U, V;
        return 0;
    }
};
```

Точка A задает начальную точку объекта, в точки B и C задают точки, расположенные на концах направлений X и Y (рисунок 10).

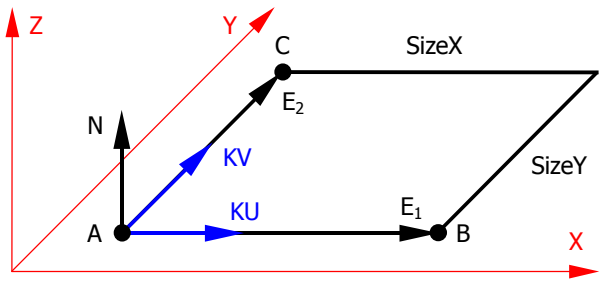


Рисунок 10 — Базис прямоугольника

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 E1 и E2 по формулам:

$E_1 = B - Loc$,

$E_2 = C - Loc$.

Нормаль вычисляется как векторное произведение векторов E1 и E2. Напоминаю, что векторное произведение определено как операция ^, а скалярное произведение — как операция *.

Нормаль нормализуется методом normalize.

Далее вычисляются коэффициенты матрицы по скалярным произведениям векторов E1 и E2:

$S_{11} = E_1 \cdot E_1$,

$S_{12} = E_1 \cdot E_2$,

$S_{22} = E_2 \cdot E_2$.

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 ε 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 0.001, вернуть 0.
4. Вычислить расстояние до точки пересечения и записать его в hit.dist.

Расстояние вычисляется по формуле

hit.dist = ((Loc – ray.org) · N) / cosVN .

5. Если расстояние hit.dist меньше 0.001, 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 · K_U) – U_0

8. Если U < 0 или U > 1.0000001, вернуть 0.
9. Вычислить координату V точки пересечения в базисе UV:

V = (hit.point · K_V) – V_0

10. Если V < 0 или V > 1.0000001, 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. Если cosVN меньше нуля, то

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

иначе

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

15. Вернуть 1.

5.3. Построение сцены

Сейчас в функции build_scene должна быть выбрана первая сцена:

```
void build_scene() {
    scene1();
    //scene2();
    //scene3();
}
```

Переходим в функцию построения первой сцены, состоящей на первом этапе из одного только зеленого прямоугольника Rect2 (рисунок 11).

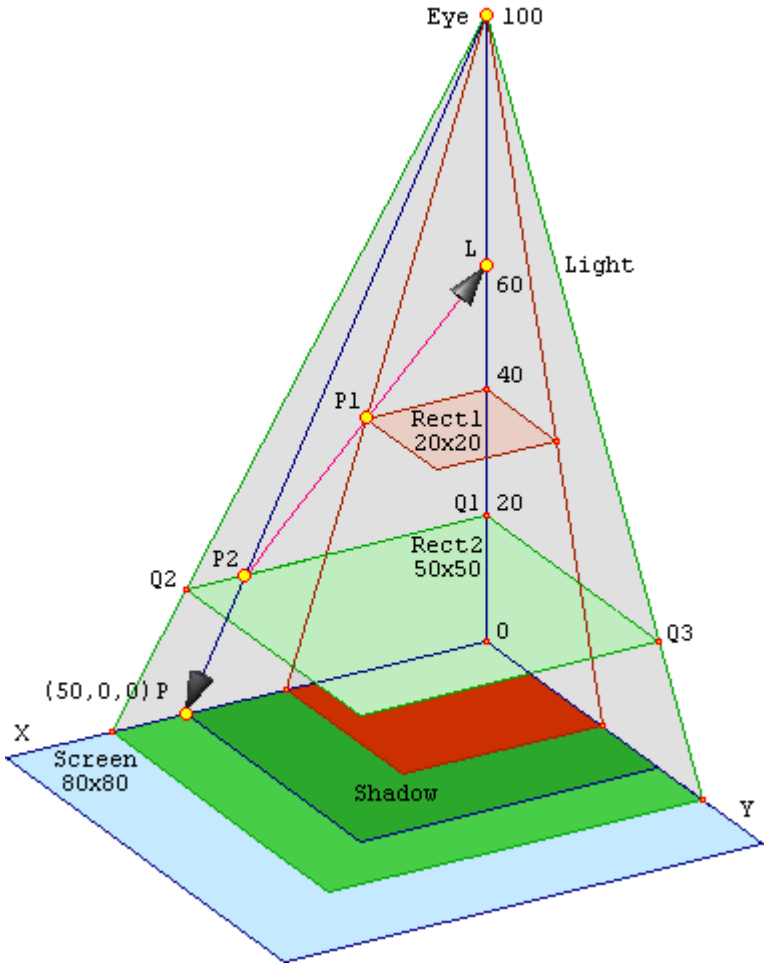


Рисунок 11 — Сцена 1

Порядок создания сцены.

1. Установить размеры экрана равными 80 и 80.
2. Установить центр экрана 0 и 0.
2. Углы anglexy и anglez установить в 0.
3. Расстояния plane и distance установить равными 100.
4. Установить максимальную глубину трассировки равную 1.
5. Установить цвет фона background равным RGB(128, 207, 255).
6. Установить цвет ambient равным RGB(255, 255, 255).
7. Объявить источник света

```
Light L;
```

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

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

12. Задать цвет поверхности объекта равным RGB(21, 234, 21).
13. 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.
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.
Функция Scene::render модуля scene.h.

```
// трассировка сцены
void Scene::render(HDC hdc, double XYAngle, 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 = XYAngle * 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:

```
// положение точки 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 выполнить преобразование экранных координат в мировые, противоположное видовому преобразованию.

Получив мировые координаты, 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);
        // вычисляем 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:

```
// процедура трассировки луча
Color Scene::trace(Ray ray) {
    // вычисляемый и возвращаемый цвет, по умолчанию - цвет фона
    Color color = background;
}
```

Если процедура render описана верно, то программа должна выводить прямоугольник цвета фона (неба).

5.4.1. Контрольный луч

Сейчас нужно выпустить один контрольный луч в точку (50, 0, 0), см. рисунок 11. Для этого нужно изменить код, выпускающий лучи:

```
void Scene::render(HDC hdc, double XYAngle, double ZAngle, . . .) {
    // трассирующий луч
    Ray ray;
    // цвет пикселя
    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;
    // углы
    double A = XYAngle * PI180;
```

Теперь в функции trace нужно вычислить точку P2 (на рисунке 11) пересечения луча с плоскостью прямоугольника:

```
Color Scene::trace(Ray ray) {
    // вычисляемый и возвращаемый цвет, по умолчанию - цвет фона
    Color color = background;
    // точка пересечения
    Hit hit;
    objects[i]->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 = BA + BD + BS

Здесь:

B — цвет точки, возвращаемый функцией trace;

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

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

BS = 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 вектором нормали микрограни и нормалью.

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

SH = light[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 функции trace после определения наличия пересечения:

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

Для контрольного луча вычисления должны давать примерно следующие значения:

BA = (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

BD = (4, 44, 4)

H = (–1.154, 0, 1.601)

H = (–0.584, 0, 0.811) — после нормализации

HN = 0.811

BS = (36, 36, 36)

B = (50, 197, 50)

Значения могут немного отличаться от приведенных, что в конечном итоге скажется на цвете пикселя, но это отличие не должно превышать 2 единицы для каждой из составляющих цвета R, G и B.

Заметим, что сначала переменной B присваивается значение фоновой освещенности, а вычисляемые значения BS и BD 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:

5.4.2. Функции 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. Однако далее появятся еще вторичные лучи, которые рассчитываются по тем же формулам, при этом функция trace может неоднократно вызываться рекурсивно.

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 функции shadow.

Параметры функции:

- луч на источник света;
- расстояние до источника света;
- начальное затенение источника света;
- идентификатор объекта, которому принадлежит точка пересечения.

Переменные функции:

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 0.001 и это расстояние меньше расстояния до источника света, то начальное затенение источника света atten умножается на коэффициент Kt поверхности пересечения. Если результат умножения менее 0.001, затенение atten принимается равным нулю, и, в принципе, цикл можно завершить. Функция возвращает atten.

Перейдем к функции 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, коих насчитывается достаточно много:

```
Color B;     // вычисляемый и возвращаемый цвет
double Ka, Ks, Kd, Ph, Kr, Kt; // параметры поверхности
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 цвета
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
double dist;    // расстояне до источника света
double atten;   // начальное затенение
```

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 Ba, значение которой сразу может быть присвоено результирующему цвету B.

Затем в функции на данном 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 источника света dist, и записать в объект ray точку пересечения hit.point как начало луча, и нормализованный вектор L как направление.

Вычисляем начальное затенение atten по формуле scale / dist.

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

Если полученное значение SH больше 0.001, то можно вычислять значения освещенностей BD и BS.

Заметим, что при вычислении составляющей BD нужно контролировать значение косинуса LN. Если косинус имеет значение большее 0.001, то составляющая BD присутствует, иначе диффузного рассеяния нет.

Аналогично, при вычислении составляющей BS нужно контролировать значение косинуса HN. Если косинус имеет значение, большее 0.001, составляющая BS есть, иначе зеркальное отражение отсутствует.

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

5.4.3. Функция trace

Что же остается в функции trace.

В ней нужно найти ближайший объект, который пересекается трассирующим лучом, получить ближайшее пересечение nearest, вычислить освещенность, которую несет луч, при помощи функции 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 0.001, 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.

Тогда после проверки всех объектов либо есть точка пересечения nearest, либо нет (тогда нет никакого пересечения).

Если точка пересечения есть (nearest.intersect истинно), вычисляем цвет color при помощи функции 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 значение следует учесть и скорректировать цвет при помощи формулы

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

Если значение ray.medium.atten больше 0.001, то уточняем значение возвращаемого цвета по данной формуле, иначе цвет остается таким, какой был получен из функции shade.

Заметим также, что, поскольку функция trace вызывается рекурсивно, для ограничения количества вторичных лучей нужно увеличивать текущую глубину level при входе в функцию, и уменьшать при выходе из нее. В дальнейшем значение глубины рекурсии будет учтено при расчете вторичных лучей.

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 с координатами (0, 0, 40), прямоугольник расположен на высоте 40 в соответствии с рисунком 11, цвет RGB(234, 21, 21), название переменной для объекта rr, добавляем его перед прямоугольником rg. Результат трассирования сцены примерно следующий:

Если картинка получена, можно переходить к расчету отраженного луча и привносимой им освещенности.

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 12).

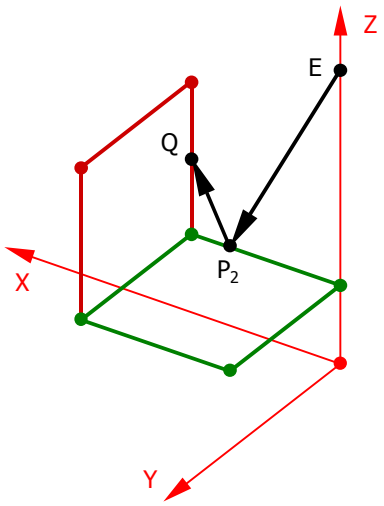


Рисунок 12 — Сцена 2

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 13.

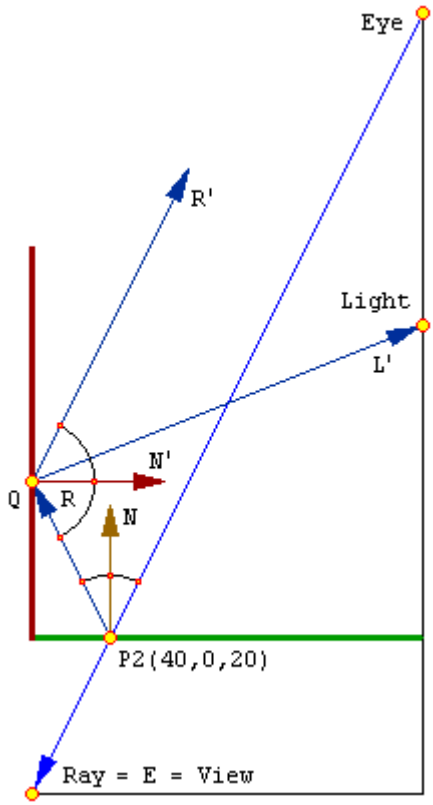


Рисунок 13 — Расчет отраженного луча

Контрольный первичный луч отражается в точке P2 зеленого прямоугольника и попадает в точку Q красного прямоугольника (луч R), в которой отражается в пространство (луч R').
Расчет вторичных лучей (отраженного и преломленного) выполняется в функции shade после расчета первичного луча. Сначала нужно убедиться, что глубина трассировки не превышает заданное значение:

```
// ПЕРВИЧНАЯ ОСВЕЩЕННОСТЬ
. . .
// ВТОРИЧНАЯ ОСВЕЩЕННОСТЬ
if (level < max_level) {
    // 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 0.001, 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 функции render контрольный луч, и проверяем вычисляемые значения отраженного луча в функции shade:

ray.weight = 0.5

ray.org = (40, 0, 20)

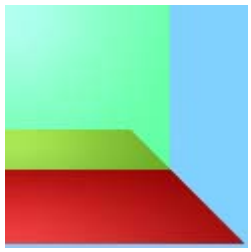
ray.dir = (0.447, 0, 0.984)

BR = (236, 73, 73)

После умножения на Kr получим BR = (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):



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. [].

```
// 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 < -EPS001) {
            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 < 0.001) return 0;
            if (TNear > TFar) return 0;
        }
        if (TNear < 0.001) {
            hit.dist = TFar;
        } else {
            hit.dist = TNear;
        }
        if (hit.dist > 0.001) {
            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

anglexy = 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 равна (–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 функции 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 0.001, 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;
    ray.dir.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)
// а не на грани
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;
// 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: