ЯРОШЕВИЧ В.А.

КОМПЬЮТЕРНАЯ ГРАФИКА

Практические занятия, лабораторный практикум

Версия 1.16 (03.04.2015)

3-ий курс

МИЭТ — 2015

Оглавление

Глава 1

Практические занятия

1.1 Занятие 1

1.1.1 Аффинные преобразования на плоскости

См. [2, стр. 76–114].

Аффи́нное преобразование (от лат. affinis — соприкасающийся, близкий, смежный) — отображение плоскости или пространства в себя, при котором прямые переходят в прямые.

Аффинное преобразование $f : \mathbb{R}^n \to \mathbb{R}^n$ представимо в виде

$$f(\mathbf{q}) = M\mathbf{q} + \mathbf{v} \tag{1.1}$$

Элементарные преобразования

- *Параллельный перенос (translation)* на вектор $\mathbf{v} = (a, b)$

$$\begin{cases} x' = x + a, \\ y' = y + b. \end{cases}$$

  Нельзя осуществить параллельный перенос, умножая матрицу 2×2 на вектор, необходимо прибавлять вектор сдвига.

- *Гомотетия (преобразование подобия, homothetic transformation)* с коэффициентами α вдоль оси Ox и β вдоль оси Oy.

$$\begin{cases} x' = \alpha x, \\ y' = \beta y. \end{cases}$$

  Матрица

$$H_{\alpha,\beta} = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}$$

- *Отражение (reflection, mirroring)* относительно оси Oy.

$$\begin{cases} x' = -x, \\ y' = y. \end{cases}$$

Матрица

$$M_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

- *Поворот (rotation)* против часовой стрелки на угол φ с центром в начале координат.

$$\begin{cases} x' = x\cos\varphi - y\sin\varphi, \\ y' = x\sin\varphi + y\cos\varphi. \end{cases}$$

Матрица

$$R_\varphi = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}$$

После перехода к однородным координатам $(x, y) \to (xw, yw, w)$ любое аффинное преобразование вида (1.1) на плоскости можно представить в виде произведения матрицы размера 3×3 на вектор с однородными координатами

$$\begin{pmatrix} f(\mathbf{q}) \\ 1 \end{pmatrix} = \begin{pmatrix} M & \mathbf{v} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ 1 \end{pmatrix}$$

1.1.2 Аффинные преобразования в пространстве

См. [2, стр. 115–129]

- *Параллельный перенос (translation)* на вектор $\mathbf{v} = (a, b, c)$

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- *Гомотетия (преобразование подобия, homothetic transformation)* с коэффициентами α вдоль оси Ox, β вдоль оси Oy и γ вдоль оси Oz.

$$\begin{pmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Если коэффициент растяжения одинаков для всех направлений, то есть $\alpha = \beta = \gamma$, то допустимо использовать матрицу

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/\alpha \end{pmatrix}.$$

- *Поворот* против часовой стрелки на угол φ, ось направлена на наблюдателя.

Вокруг оси Ox	Вокруг оси Oy	Вокруг оси Oz
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & & 1 & 0 \\ 0 & & 0 & 1 \end{pmatrix}$

- *Отражение.*

Относительно пл-ти Oxy	Относительно пл-ти Oxz	Относительно пл-ти Oyz
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Чтобы получить матрицу A некоторого аффинного преобразования A, требуется перемножить несколько матриц элементарных преобразований. Но имея в распоряжении такую матрицу A, можно для любой точки v плоскости (или пространства) найти её образ $\mathrm{v}' = A\mathrm{v}$. Если нужно найти образы сразу нескольких точек $\mathrm{v}_1, \mathrm{v}_2, \dots \mathrm{v}_k$, то с точки зрения вычислительных затрат выгоднее объединить координаты точек в общую матрицу и затем умножить её на матрицу преобразований A:

$$\begin{cases} \mathrm{v}'_1 = A\mathrm{v}_1, \\ \mathrm{v}'_2 = A\mathrm{v}_2, \\ \dots \\ \mathrm{v}'_k = A\mathrm{v}_k; \end{cases} \quad \rightarrow \quad \begin{array}{c} \begin{pmatrix} \mathrm{v}'_1 & \mathrm{v}'_2 & \dots & \mathrm{v}'_k \\ | & | & & | \end{pmatrix} = A \begin{pmatrix} \mathrm{v}_1 & \mathrm{v}_2 & \dots & \mathrm{v}_k \\ | & | & & | \end{pmatrix} \\ \begin{pmatrix} x_1 & x_2 & \dots & x_k \\ y_1 & y_2 & \dots & y_k \\ z_1 & z_2 & \dots & z_k \\ w_1 & w_2 & \dots & w_k \end{pmatrix} = A \begin{pmatrix} x_1 & x_2 & \dots & x_k \\ y_1 & y_2 & \dots & y_k \\ z_1 & z_2 & \dots & z_k \\ w_1 & w_2 & \dots & w_k \end{pmatrix} \end{array}$$

1.1 На плоскости задана прямая $3x - 2y + 1 = 0$. Получить матрицу $\mathcal{M}$, осуществляющую зеркальное отражение плоскости относительно прямой.

Возможны два решения:

Решение. (1) Получим искомую матрицу $\mathcal{M}$ в виде произведения матриц элементарных преобразований. С помощью параллельного переноса и поворота совместим прямую в осью Ox. Далее осуществим зеркальное отражение относительно Ox. Теперь выполним поворот в обратную сторону и параллельный перенос, чтобы перевести Ox в исходную прямую.

$$T_{(0,-\frac{1}{2})} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}, \quad R_{-\varphi} = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad M_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$R_{\varphi} = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad T_{(0,\frac{1}{2})} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}, \text{ где } \varphi = \arctan\frac{3}{2}.$$

В итоге

$$\mathcal{M} = T_{(0,\frac{1}{2})} R_\varphi M_x R_{-\varphi} T_{(0,-\frac{1}{2})}$$

□

Решение. (2) Рассмотрим прямую на плоскости в общем виде:

$$Ax + By + C = 0.$$

Она имеет нормаль $\mathbf{n} = (A, B)$.

Пусть P — некоторая точка и Q её отражение относительно прямой. Очевидно, $\overline{PQ} = \alpha\mathbf{n}$ при некотором $\alpha \in \mathbb{R}$. Середина отрезка — точка T принадлежит прямой:

$$T = P + \frac{1}{2}\overline{PQ} = P + \frac{\alpha}{2}\mathbf{n}, \quad AT_x + BT_y + C = 0.$$

Получаем отсюда

$$A(P_x + \frac{\alpha}{2}A) + B(P_y + \frac{\alpha}{2}B) + C = 0,$$

или

$$(\mathbf{n}, P) + \frac{\alpha}{2}(\mathbf{n}, \mathbf{n}) + C = 0.$$

Следовательно,

$$\alpha = -2\frac{(\mathbf{n}, P) + C}{(\mathbf{n}, \mathbf{n})} \quad \text{и} \quad Q = P + \alpha\mathbf{n} = P - 2\frac{(\mathbf{n}, P) + C}{(\mathbf{n}, \mathbf{n})}\mathbf{n}.$$

В покоординатном виде

$$Q = \begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} P_x - \frac{2(AP_x+BP_y+C)}{A^2+B^2}A \\ P_y - \frac{2(AP_x+BP_y+C)}{A^2+B^2}B \\ 1 \end{pmatrix} = \begin{pmatrix} P_x[1 - \frac{2A^2}{A^2+B^2}] - P_y[\frac{2AB}{A^2+B^2}] - \frac{2AC}{A^2+B^2} \\ -P_x[\frac{2AB}{A^2+B^2}] + P_y[1 - \frac{2B^2}{A^2+B^2}] - \frac{2BC}{A^2+B^2} \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 - \frac{2A^2}{A^2+B^2} & -\frac{2AB}{A^2+B^2} & -\frac{2AC}{A^2+B^2} \\ -\frac{2AB}{A^2+B^2} & 1 - \frac{2B^2}{A^2+B^2} & -\frac{2BC}{A^2+B^2} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = \mathcal{M}P.$$

□

1.2 Задан бесконечно удалённый источник света, лучи которого направлены параллельно вектору ℓ, и плоскость $\gamma : Ax + By + Cz + D = 0$. Некоторый объект Δ расположен между источником света и плоскостью. Найти матрицу, позволяющую построить тень объекта на плоскости.

Решение. Всякой точке $P \in \Delta$ соответствует некоторая точка $Q \in \gamma$ (рис. 1.1). Наша цель — найти такую матрицу $\mathcal{M}$, что $Q = \mathcal{M}P$.

Обозначим $\mathbf{n} = (A, B, C)$ — нормаль плоскости. Заметим, что $\overline{PQ} || \bar{\ell}$. Отсюда $\overline{PQ} = \alpha\ell$ для некоторого числа $\alpha \in \mathbb{R}$ и

$$Q = P + \alpha\ell. \tag{1.2}$$

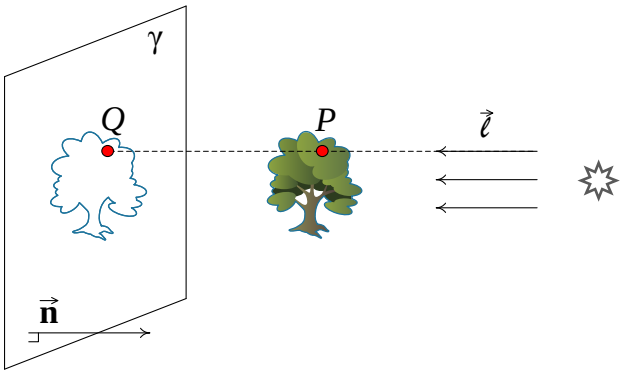


Рис. 1.1: Тень при бесконечно удалённом источнике света.

Точка Q лежит в плоскости γ, поэтому

$$AQ_x + BQ_y + CQ_z + D = 0. \tag{1.3}$$

Подставляя (1.2) в (1.3) найдём α:

$$A(P_x + \alpha\ell_x) + B(P_y + \alpha\ell_y) + C(P_z + \alpha\ell_z) + D = 0,$$

$$(\mathbf{n}, P) + \alpha(\mathbf{n}, \ell) + D = 0,$$

$$\alpha = -\frac{D + (\mathbf{n}, P)}{(\mathbf{n}, \ell)}.$$

Вернёмся обратно к (1.2):

$$Q = P - \frac{D + (\mathbf{n}, P)}{(\mathbf{n}, \ell)}\ell.$$

Перепишем покоординатно:

$$Q = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} P_x - \frac{D+(AP_x+BP_y+CP_z)}{(\mathbf{n},\ell)}\ell_x \\ P_y - \frac{D+(AP_x+BP_y+CP_z)}{(\mathbf{n},\ell)}\ell_y \\ P_z - \frac{D+(AP_x+BP_y+CP_z)}{(\mathbf{n},\ell)}\ell_z \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} P_x[1 - \frac{A}{(\mathbf{n},\ell)}\ell_x] - P_y[\frac{B}{(\mathbf{n},\ell)}\ell_x] - P_z[\frac{C}{(\mathbf{n},\ell)}\ell_x] - \frac{D}{(\mathbf{n},\ell)}\ell_x \\ -P_x[\frac{A}{(\mathbf{n},\ell)}\ell_y] + P_y[1 - \frac{B}{(\mathbf{n},\ell)}\ell_y] - P_z[\frac{C}{(\mathbf{n},\ell)}\ell_y] - \frac{D}{(\mathbf{n},\ell)}\ell_y \\ -P_x[\frac{A}{(\mathbf{n},\ell)}\ell_z] - P_y[\frac{B}{(\mathbf{n},\ell)}\ell_z] + P_z[1 - \frac{C}{(\mathbf{n},\ell)}\ell_z] - \frac{D}{(\mathbf{n},\ell)}\ell_z \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 - \frac{A}{(\mathbf{n},\ell)}\ell_x & -\frac{B}{(\mathbf{n},\ell)}\ell_x & -\frac{C}{(\mathbf{n},\ell)}\ell_x & -\frac{D}{(\mathbf{n},\ell)}\ell_x \\ -\frac{A}{(\mathbf{n},\ell)}\ell_y & 1 - \frac{B}{(\mathbf{n},\ell)}\ell_y & -\frac{C}{(\mathbf{n},\ell)}\ell_y & -\frac{D}{(\mathbf{n},\ell)}\ell_y \\ -\frac{A}{(\mathbf{n},\ell)}\ell_z & -\frac{B}{(\mathbf{n},\ell)}\ell_z & 1 - \frac{C}{(\mathbf{n},\ell)}\ell_z & -\frac{D}{(\mathbf{n},\ell)}\ell_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} = \mathcal{M}P.$$

□

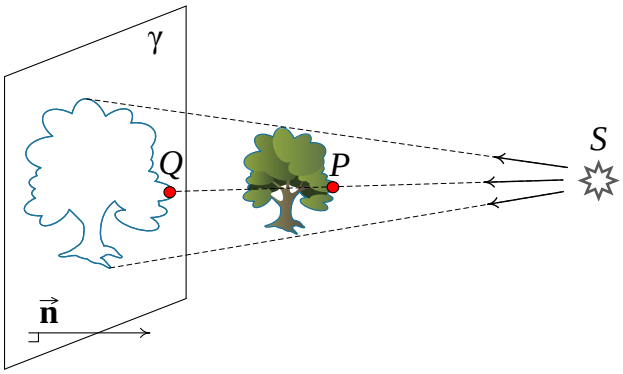


Рис. 1.2: Тень при источнике света, расположенном в конечной точке.

1.3 В некоторой конечной точке S расположен источник света. Задана плоскость $\gamma : Ax + By + Cz + D = 0$. Некоторый объект Δ расположен между источником света и плоскостью. Найти матрицу, позволяющую построить тень объекта на плоскости.

Решение. Всякой точке $P \in \Delta$ соответствует некоторая точка $Q \in \gamma$ (рис. 1.2). Наша цель — найти такую матрицу $\mathcal{M}$, что $Q = \mathcal{M}P$.

Обозначим $\mathbf{n} = (A, B, C)$ — нормаль плоскости. Заметим, что $\overline{SQ} || \overline{SP}$. Отсюда $\overline{SQ} = \alpha \overline{SP}$ для некоторого числа $\alpha \in \mathbb{R}$ и

$$Q = S + \alpha \overline{SP}. \tag{1.4}$$

Точка Q лежит в плоскости γ, поэтому

$$AQ_x + BQ_y + CQ_z + D = 0. \tag{1.5}$$

Подставляя (1.4) в (1.5) найдём α:

$$A(S_x + \alpha \overline{SP}_x) + B(S_y + \alpha \overline{SP}_y) + C(S_z + \alpha \overline{SP}_z) + D = 0,$$

$$(\mathbf{n}, S) + \alpha(\mathbf{n}, \overline{SP}) + D = 0,$$

$$\alpha = -\frac{D + (\mathbf{n}, S)}{(\mathbf{n}, \overline{SP})}.$$

Вернёмся обратно к (1.4):

$$Q = S - \frac{D + (\mathbf{n}, S)}{(\mathbf{n}, \overline{SP})} \overline{SP}.$$

Перепишем покоординатно:

$$Q = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} Q_x(\mathbf{n}, \overline{SP}) \\ Q_y(\mathbf{n}, \overline{SP}) \\ Q_z(\mathbf{n}, \overline{SP}) \\ (\mathbf{n}, \overline{SP}) \end{pmatrix} = \begin{pmatrix} S_x(\mathbf{n}, \overline{SP}) - [D + (\mathbf{n}, S)]\overline{SP}_x \\ S_y(\mathbf{n}, \overline{SP}) - [D + (\mathbf{n}, S)]\overline{SP}_y \\ S_z(\mathbf{n}, \overline{SP}) - [D + (\mathbf{n}, S)]\overline{SP}_z \\ (\mathbf{n}, \overline{SP}) \end{pmatrix} =$$

$$= \begin{pmatrix} S_x[AP_x + BP_y + CP_z - (\mathbf{n}, S)] - [D + (\mathbf{n}, S)](P_x - S_x) \\ S_y[AP_x + BP_y + CP_z - (\mathbf{n}, S)] - [D + (\mathbf{n}, S)](P_y - S_y) \\ S_z[AP_x + BP_y + CP_z - (\mathbf{n}, S)] - [D + (\mathbf{n}, S)](P_z - S_z) \\ AP_x + BP_y + CP_z - (\mathbf{n}, S) \end{pmatrix} =$$

$$= \begin{pmatrix} P_x[AS_x - D - (\mathbf{n}, S)] + P_y[BS_x] + P_z[CS_x] + DS_x \\ P_x[AS_y] + P_y[BS_y - D - (\mathbf{n}, S)] + P_z[CS_y] + DS_y \\ P_x[AS_z] + P_y[BS_z] + P_z[CS_z - D - (\mathbf{n}, S)] + DS_z \\ P_xA + P_yB + P_zC - (\mathbf{n}, S) \end{pmatrix} =$$

$$= \begin{pmatrix} AS_x - D - (\mathbf{n}, S) & BS_x & CS_x & DS_x \\ AS_y & BS_y - D - (\mathbf{n}, S) & CS_y & DS_y \\ AS_z & BS_z & CS_z - D - (\mathbf{n}, S) & DS_z \\ A & B & C & -(\mathbf{n}, S) \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} = \mathcal{M}P.$$

□

1.2 Занятие 2

Как правило, задачей программы, использующей OpenGL, является обработка трёхмерной сцены и интерактивное отображение в буфере кадра. Сцена состоит из набора трёхмерных объектов, источников света и виртуальной камеры, определяющей текущее положение наблюдателя.

Обычно приложение OpenGL в бесконечном цикле вызывает функцию обновления изображения в окне. В этой функции и сосредоточены вызовы основных команд OpenGL. Если используется библиотека GLUT, то это будет функция с обратным вы зовом, зарегистрированная с помощью вызова `glutDisplayFunc()`. GLUT вызывает эту функцию, когда операционная система информирует приложение о том, что содержимое окна необходимо перерисовать (например, если окно было перекрыто другим). Создаваемое изображение может быть как статичным, так и анимированным, т.е. зависеть от каких-либо параметров, изменяющихся со временем. В этом случае лучше вызывать функцию обновления самостоятельно. Например, с помощью команды `glutPostRedisplay()`.

Приступим, наконец, к тому, чем занимается типичная функция обновления изображения. Как правило, она состоит из трёх шагов:

- очистка буферов OpenGL;
- установка положения наблюдателя;

- преобразование и рисование геометрических объектов.

Очистка буферов производится с помощью команды:

```
void glClearColor(clampf r , clampf g, clampf b, clampf a)
void glClear(bitfield buf)
```

Команда `glClearColor` устанавливает цвет, которым будет заполнен буфер кадра. Первые три параметра команды задают R, G и B компоненты цвета и должны принадлежать отрезку $[0, 1]$. Четвертый параметр задаёт так называемую альфа компоненту. Как правило, он равен 1. По умолчанию цвет — чёрный (0,0,0,1).

Команда `glClear` очищает буферы, а параметр `buf` определяет комбинацию констант, соответствующую буферам, которые нужно очистить. Типичная программа вызывает команду

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

для очистки буферов цвета и глубины.

Установка положения наблюдателя и преобразования трёхмерных объектов (поворот, сдвиг и т.д.) контролируются с помощью задания матриц преобразования.

Сейчас сосредоточимся на том, как передать в OpenGL описания объектов, находящихся в сцене. Каждый объект является набором примитивов OpenGL.

1.2.1 Вершины и примитивы

В OpenGL вершина (vertex) является атомарным графическим примитивом и определяет точку, конец отрезка, угол многоугольника и т.д. Все остальные примитивы формируются с помощью задания вершин, входящих в данный примитив. Например, отрезок определяется двумя вершинами, являющимися концами отрезка.

С каждой вершиной ассоциируются её атрибуты. В число основных атрибутов входят положение вершины в пространстве, цвет вершины и вектор нормали.

1.2.2 Положение вершины в пространстве

Положение вершины определяются заданием её координат в двух-, трёх-, или четырёхмерном пространстве (однородные координаты). Это реализуется с помощью нескольких вариантов команды `glVertex`:

```
void glVertex [2 3 4] [ s i f d] (type coords )
void glVertex [2 3 4] [ s i f d] v (type *coords )
```

Каждая команда задаёт четыре координаты вершины: x, y, z, w. Команда `glVertex2*` получает значения x и . Координата z в таком случае устанавливается по умолчанию равной 0, координата w — равной 1. `glVertex3*` получает координаты x, y, z и заносит в координату w значение 1. `glVertex4*` позволяет задать все четыре координаты.

Для ассоциации с вершинами цветов, нормалей и текстурных координат используются текущие значения соответствующих данных, что отвечает организации OpenGL

как конечного автомата. Эти значения могут быть изменены в любой момент с помощью вызова соответствующих команд.

Для задания текущего цвета вершины используются команды

```
void glColor [3 4] [b s i f ] (GLtype components)
void glColor [3 4] [b s i f ]v (GLtype components)
```

Первые три параметра задают R, G, B компоненты цвета, а последний параметр определяет коэффициент непрозрачности (так называемая альфа-компонента) . Если в названии команды указан тип «f» (float), то значения всех параметров должны принадлежать отрезку [0,1], при этом по умолчанию значение альфа-компоненты устанавливается равным 1.0, что соответствует полной непрозрачности. Тип «ub» (unsigned byte) подразумевает, что значения должны лежать в отрезке $[0, 255]$.

Вершинам можно назначать различные цвета, и, если включен соответствующий режим, то будет проводиться линейная интерполяция цветов по поверхности примитива.

Для управления режимом интерполяции используется команда

```
void glShadeModel (GLenum mode)
```

вызов которой с параметром `GL_SMOOTH` включает интерполяцию (установка по умолчанию), а с `GL_FLAT` — отключает.

Определить нормаль в вершине можно, используя команды

```
void glNormal3 [b s i f d] ( type coords )
void glNormal3 [b s i f d] v ( type coords )
```

Для правильного расчёта освещения необходимо, чтобы вектор нормали имел единичную длину. В OpenGL существует специальный режим, при котором задаваемые нормали будут нормироваться автоматически. Его можно включить командой `glEnable(GL_NORMALIZE)`.

Режим автоматической нормализации должен быть включен, если приложение использует модельные преобразования растяжения/сжатия, так как в этом случае длина нормалей изменяется при умножении на модельно-видовую матрицу.

Однако применение этого режима уменьшает скорость работы механизма визуализации OpenGL, так как нормализация векторов имеет заметную вычислительную сложность (взятие квадратного корня). Поэтому лучше сразу задавать единичные нормали.

Отметим, что команды

```
void glEnable (GLenum mode)
void glDisable (GLenum mode)
```

производят включение и отключение того или иного режима работы конвейера OpenGL. Эти команды применяются достаточно часто, и их возможные параметры будут рассматриваться в каждом конкретном случае.

1.2.3 Операторные скобки glBegin / glEnd

Мы рассмотрели задание атрибутов одной вершины. Однако чтобы задать атрибуты графического примитива, одних координат вершин недостаточно. Эти вершины надо объединить в одно целое, определив необходимые свойства. Для этого в OpenGL используются так называемые операторные скобки, являющиеся вызовами специальных команд OpenGL. Определение примитива или последовательности примитивов происходит между вызовами команд

```
void glBegin (GLenum mode)
void glEnd (void)
```

Параметр `mode` определяет тип примитива, который задаётся внутри и может принимать следующие значения:

- `GL_POINTS` — каждая вершина задаёт координаты некоторой точки.
- `GL_LINES` — каждая отдельная пара вершин определяет от-резок; если задано нечетное число вершин, то последняя вершина игнорируется.
- `GL_LINE_STRIP` — каждая следующая вершина задаёт от резок вместе с предыдущей.
- `GL_LINE_LOOP` — отличие от предыдущего примитива толь-ко в том, что последний отрезок определяется последней и первой вершиной, образуя замкнутую ломаную.
- `GL_TRIANGLES` — каждые отдельные три вершины определяют треугольник; если задано не кратное трем число вершин, то последние вершины игнорируются.
- `GL_TRIANGLE_STRIP` — каждая следующая вершина задаёт треугольник вместе с двумя предыдущими.
- `GL_TRIANGLE_FAN` — треугольники задаются первой вершиной и каждой следующей парой вершин (пары не пересекаются) .
- `GL_QUADS` — каждая отдельная четвёрка вершин определяет четырёхугольник; если задано не кратное четырём число вершин, то последние вершины игнорируются.
- `GL_QUAD_STRIP` — четырёхугольник с номером n определяется вершинами с номерами $2n-1$, $2n$, $2n+2$, $2n+1$.
- `GL_POLYGON` — последовательно задаются вершины выпуклого многоугольника.

Например, чтобы нарисовать треугольник с разными цветами в вершинах, достаточно написать:

```
GLfloat blueColor[3] = {0,0,1};
glBegin(GL_TRIANGLES);
  glColor3f(1.0, 0.0, 0.0);  /* красный */
  glVertex3f(0.0, 0.0, 0.0);
  glColor3ub(0, 255, 0);     /* зеленый */
```

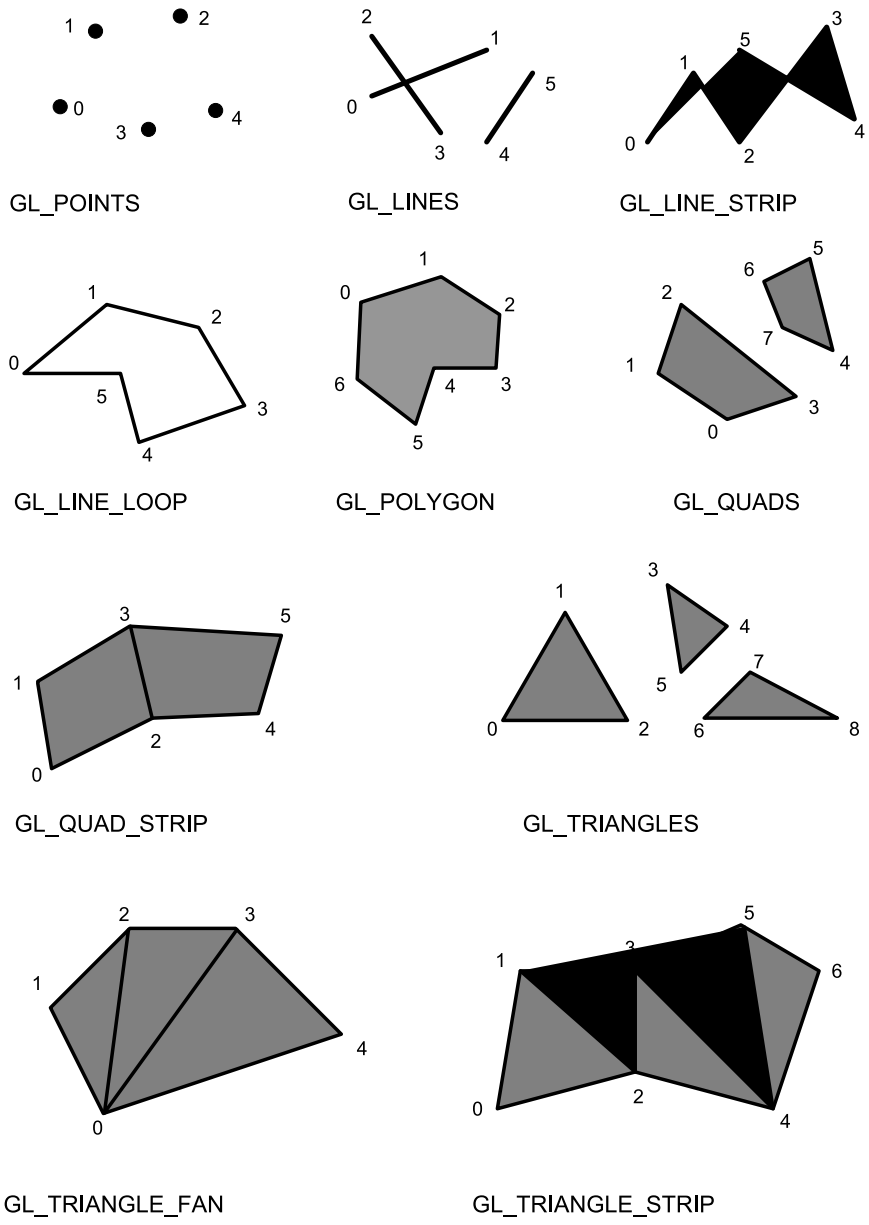


Рис. 2.3: Примитивы OpenGL.

```
  glVertex3f(1.0, 0.0, 0.0);
  glColor3fv(blueColor);      /* синий */
  glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Как правило, разные типы примитивов имеют различную скорость визуализации на разных платформах. Для увеличения производительности предпочтительнее использовать примитивы, требующие меньшее количество информации для передачи на сервер, такие как GL_TRIANGLE_STRIP, GL_QUAD_STRIP, GL_TRIAGLE_FAN.

Кроме задания самих многоугольников, можно определить метод их отображения на экране. Однако сначала надо определить понятие лицевых и обратных граней.

Под гранью понимается одна из сторон многоугольника, и по умолчанию лицевой считается та сторона, вершины которой обходятся против часовой стрелки. Направление обхода вершин лицевых граней можно изменить вызовом команды

```
void glFrontFace (GLenum торе)
```

со значением параметра торе равным GL_CW (clockwise), а вернуть значение по умолчанию можно, указав GL_CCW (counter clockwise).

Чтобы изменить метод отображения многоугольника используется команда

```
void giPolygonMode (GLenum face , GLenum mode)
```

Параметр торе определяет как будут отображаться многоугольники, а параметр face устанавливает тип многоугольников, к которым будет применяться эта команда и может принимать следующие значения:

- GL_FRONT — для лицевых граней;
- GL_BACK — для обратных граней;
- GL_FRONT_AND_BACK — для всех граней.

Параметр торе может быть равен:

- GL_POINT — отображение только вершин многоугольников;
- GL_LINE — многоугольники будут представляться набором отрезков;
- GL_FILL — многоугольники будут закрашиваться текущим цветом с учётом освещения, и этот режим установлен по умолчанию.

Также можно указывать какой тип граней отображать на экране. Для этого сначала надо установить соответствующий режим вызовом команды glEnable(GL_CULL_FACE), а затем выбрать тип отображаемых граней с помощью команды

```
void glCullFace (GLenum торе)
```

Вызов с параметром GL_FRONT приводит к удалению из изображения всех лицевых граней, а с параметром GL_BACK — обратных (установка по умолчанию).

Кроме рассмотренных стандартных примитивов в библиотеках GLU и GLUT описаны более сложные фигуры, такие как сфера, цилиндр, диск (в GLU) и сфера, куб, конус, тор, тетраэдр, додекаэдр, икосаэдр, октаэдр и чайник (в GLUT). Автоматическое наложение текстуры предусмотрено только для фигур из библиотеки GLU.

1.2.4 Модели освещения

Модель освещённости Фонга является наиболее простой и наиболее популярной моделью для 3-мерной компьютерной графики. Её популярность обусловлена:

- возможностью достижения многих визуальных эффектов;
- простотой реализации, как в виде программы, так и в виде микросхемы.

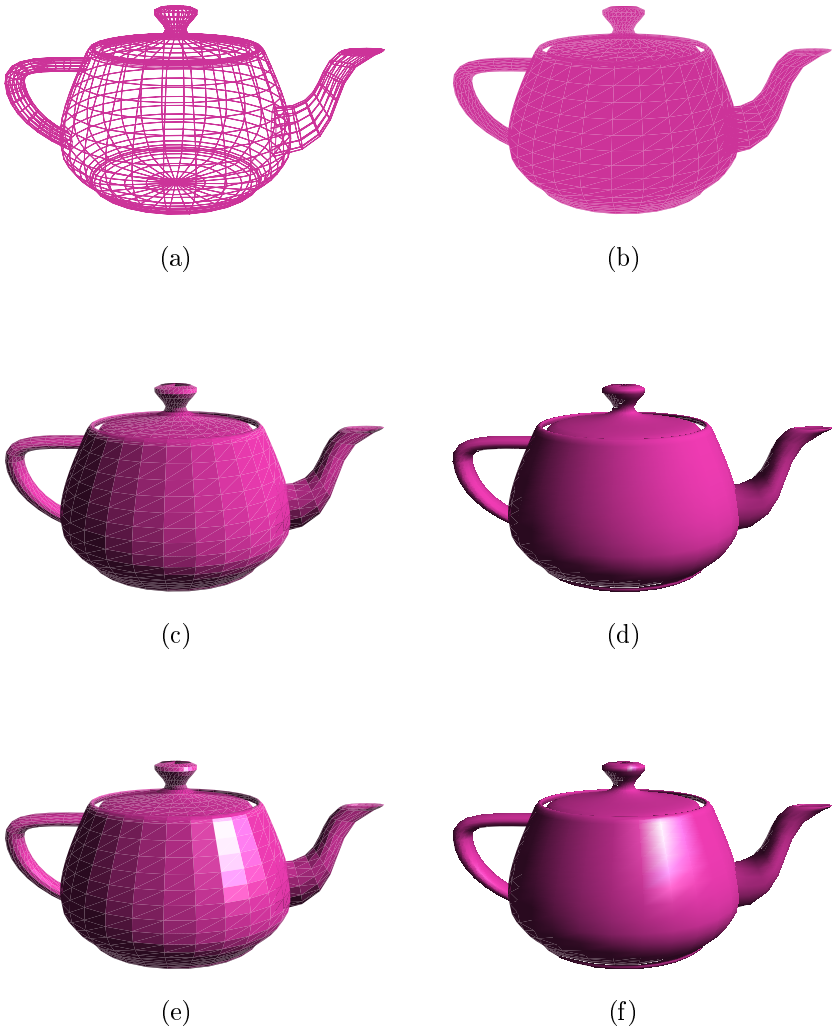


Рис. 2.4: Шесть вариантов освещения и затенения. (a) Остов. (b) Раскраска сплошным цветом без учёта освещения и затенения. (c) Дискретное затенение, фоновое и рассеянное освещение. (d) Метод Гуро с фоновым и рассеянным освещением. (e) Дискретное затенение с фоновым, рассеянным и зеркальным освещением. (f) Метод Гуро с фоновым, рассеянным и зеркальным освещением.

Модель Фонга описывает то, как поверхность объекта на сцене отражает свет. Предполагается, что все источники света точечные (то есть без специальных мер нельзя представить, например, похожую на стержень лампу дневного света). Цвет

состоит из трёх дискретных компонент (красного, зелёного и синего). Каждый источник света имеет только чистую красную, чистую зелёную и чистую синюю составляющие. *Принцип суперпозиции* позволяет вычислять и интенсивность отражённого света независимо для всех источников света и для каждого из трёх цветов.

Модель Фонга допускает два типа отражения:

- *Рассеянное отражение.* При этом лучи отражаются от поверхности одинаково во всех направлениях (рис. 2.5). Рассеянное отражение хорошо описывает матовые поверхности.
- *Зеркальное отражение.* Лучи отражаются от поверхности как от зеркала. Как показано на рис. 2.6, отражённые лучи составляют с поверхностью приблизительно тот же угол, и падающий луч. Зеркальное отражение хорошо описывает полированные и глянцевые поверхности. При зеркатоном отражение на поверхности могут образовываться *блики* — участки поверхности с наибольшей интенсивностью отражённого света.

В модели Фонга свет бывает трёх видов:

- *Зеркальный свет* от точечного источника влияет только на зеркальное отражение.
- *Рассеянный свет* от точечного источника влияет только на рассеянное отражение.
- *Внешний свет* не порождается точечным источником. Во всех точках сцены, во всех направлениях внешний свет достигает поверхностей объектов с равной интенсивностью. Это своего рода приближение множественного отражения лучей между объектами. В модели Фонга не учитывается свет, отражённый от других объектов.

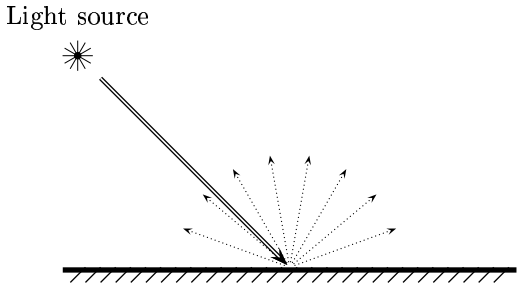


Рис. 2.5: Рассеянно отражённый свет одинаково яркий во всех направлениях. Двойной стрелкой обозначен падающий луч, пунктир — отражённый свет.

Цвет моделируется с помощью красной, зелёной и синий компонент, каждая из которых имеет зеркальную, рассеянную и внешнюю составляющие.

В модели Фонга любая поверхность обладает свойствами *материала*. Эти свойства характеризуют то, как материал отражает свет. Они задаются независимо для трёх цветов. Свойства материала:

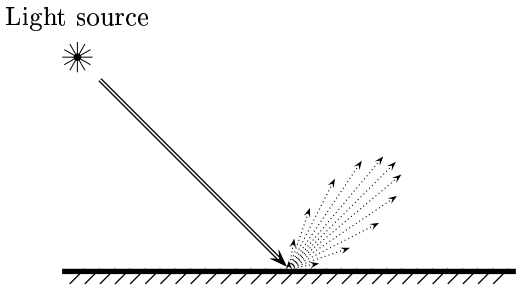


Рис. 2.6: В случае зеркального отражения направления отражённых лучей составляют углы с нормалью поверхности близкие к углу падения света. Двойной стрелкой обозначен падающий луч, пунктир — отражённый свет. Чем длиннее пунктирная стрелка, тем ярче свет в соответствующем направлении.

- *Коэффициент зеркального отражения* ρ_{s}, *коэффициент рассеянного отражения* ρ_{d}, *коэффициент внешнего отражения* ρ_{a} определяют, какая часть света отразится соответственно для зеркальной составляющей света, рассеянной и внешней.

- Излучательность поверхности I_{e} определяет свет, исходящий от поверхности при отсутствии любых источников света. Возможность материала излучать свет не делает поверхность источником света в терминах модели Фонга. Причина в том, что излучённый свет не может быть отражён от других поверхностей. Свет излучения действует только на наблюдателя.

Схема отражения в модели Фонга изображена на рис. 2.7. Определённая точка поверхности освещена точечным источником света и видна наблюдателю. Ориентация поверхности задана единичным вектором нормали $\mathbf{n}$. Направление лучей обозначено единичным вектором ℓ, направленным от точки на поверхности к источнику света. Направление обзора задано единичным вектором $\mathbf{v}$, направленным от точки поверхности к наблюдателю. Эти три вектора вместе со свойствами материала позволяют рассчитать освещённость точки в модели Фонга.

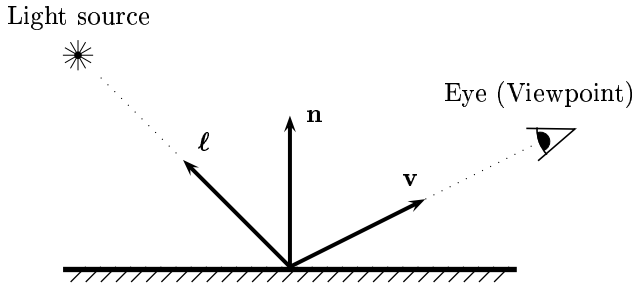


Рис. 2.7: Основные вектора модели освещения Фонга: $\mathbf{n}$ —нормаль к поверхности; ℓ — направление, где расположен точечный источник света; $\mathbf{v}$ — направление, где расположен наблюдатель. Все вектора $\mathbf{n}$, ℓ $\mathbf{v}$ единичные и необязательно лежат в одной плоскости.

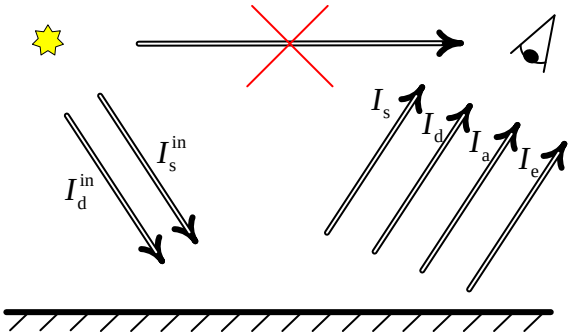


Рис. 2.8: Лучи источника обязательно должны отразиться от какой-либо поверхности.

Считаем, что свет от точечного источника света имеет интенсивность I^{in}. По интенсивностью света будем понимать энергию, сообщаемую светом единице площади, перпендикулярной лучам.

Лучи источника обязательно должны отразиться от какой-либо поверхности (рис. 2.8).

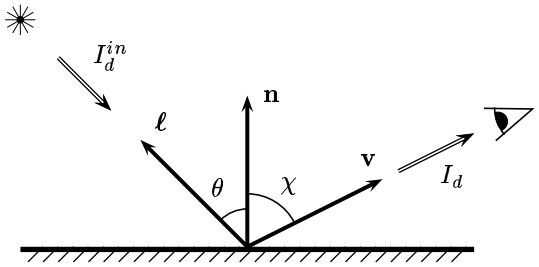


Рис. 2.9: Рассеянное отражение в модели Фонга. Угол падения равен θ. Интенсивности света падающего на поверхность и достигшего наблюдателя обозначены соответственно I_d^{in} и I_d.

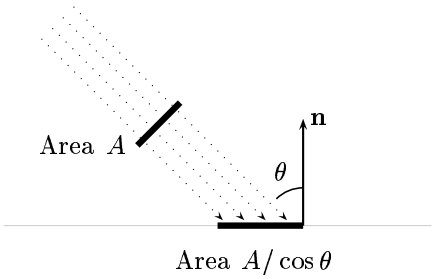


Рис. 2.10: Перпендикулярная потоку света площадка A. При наклонении на угол θ площадь возрастает в $1/\cos\theta$ раз.

$$I = I_{\mathrm{a}} + I_{\mathrm{d}} + I_{\mathrm{s}} + I_{\mathrm{e}} = \rho_{\mathrm{a}} I_{\mathrm{a}}^{in} + \rho_{\mathrm{d}} I_{\mathrm{d}}^{in}(\ell, \mathbf{n}) + \rho_{\mathrm{s}} I_{\mathrm{s}}^{in}(\mathrm{r}, \mathbf{v})^f + I_{\mathrm{e}}.$$

$$\mathbf{I} = \rho_{\mathrm{a}} * \mathbf{I}_{\mathrm{a}}^{in} + \rho_{\mathrm{d}} * \mathbf{I}_{\mathrm{d}}^{in}(\ell, \mathbf{n}) + \rho_{\mathrm{s}} * \mathbf{I}_{\mathrm{s}}^{in}(\mathrm{r}, \mathbf{v})^f + \mathbf{I}_{\mathrm{e}}.$$

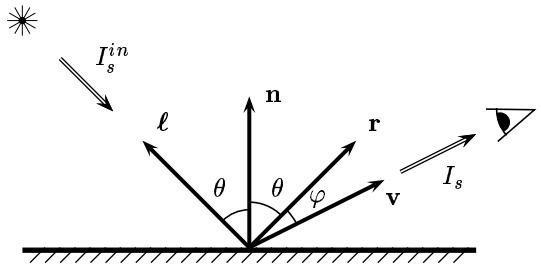


Рис. 2.11: Зеркальное отражение в модели Фонга. Угол падения равен θ. Направление наилучшего отражения: $\mathbf{r}$. Интенсивности зеркального света, падающего на поверхность и достигшего наблюдателя обозначены соответственно I_s^{in} и I_s.

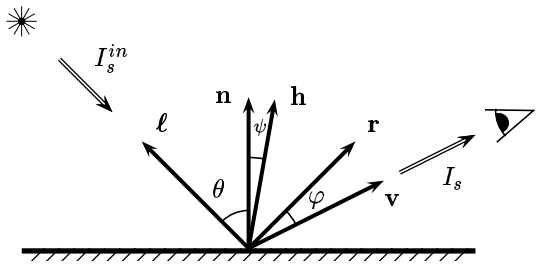
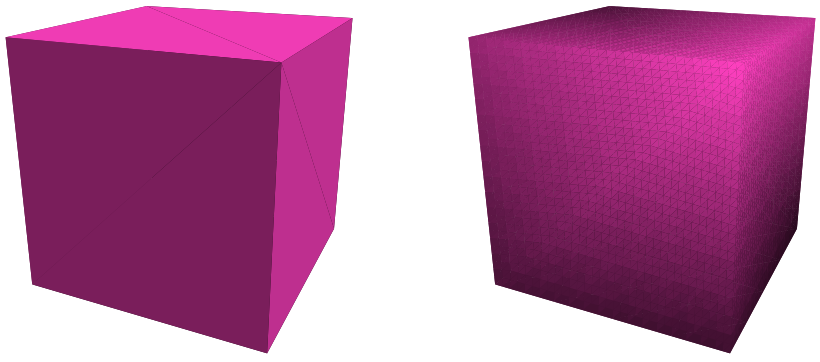


Рис. 2.12: Расчёт зеркального отражения с помощью промежуточного вектора $\mathbf{h}$. Этот вектор имеет единичную длину и промежуточное между ℓ и $\mathbf{v}$ направление.



(a) (b)

Рис. 2.13: Два куба (a) с нормалями перпендикулярными граням и (b) с нормалями исходящими из центра куба. В обоих случаях использован метод Гуро.

1.3 Занятие 3

1.4 Занятие 4

1.4.1 Барицентрические координаты

Обозначим для краткости $\mathbf{e}_1 = \overline{AB}$, $\mathbf{e}_2 = \overline{AC}$ и $\mathbf{f} = \overline{AP}$ (рис. 4.14). Рассмотрим направленный внутрь треугольника вектор n такой, что $|\mathbf{n}| = 1$ и $\mathbf{n} \perp \mathbf{e}_2$. Для получения $\mathbf{n}$ найдём составляющую $\mathbf{m}$ вектора $\mathbf{e}_1$ перпендикулярную $\mathbf{e}_2$

$$\mathbf{m} = \mathbf{e}_1 - \frac{(\mathbf{e}_1, \mathbf{e}_2)\mathbf{e}_2}{(\mathbf{e}_2, \mathbf{e}_2)}, \qquad \mathbf{n} = \frac{\mathbf{m}}{|\mathbf{m}|}.$$

(Деление на $(\mathbf{e}_2, \mathbf{e}_2)$ необходимо, так как $\mathbf{e}_2$ может оказаться неединичным.) Если считать $\mathbf{e}_2$ основанием $\triangle ABC$, то его высота будет равна $(\mathbf{n}, \mathbf{e}_1)$. Отсюда получим площадь

$$S_{ABC} = \frac{1}{2}(\mathbf{n}, \mathbf{e}_1)|\mathbf{e}_2| = \frac{(\mathbf{m}, \mathbf{e}_1)|\mathbf{e}_2|}{2|\mathbf{m}|}.$$

Аналогично получим площадь $\triangle ACP$

$$S_{ACP} = \frac{1}{2}(\mathbf{n}, \mathbf{f})|\mathbf{e}_2| = \frac{(\mathbf{m}, \mathbf{f})|\mathbf{e}_2|}{2|\mathbf{m}|}.$$

Следовательно,

$$\beta = \frac{S_{ACP}}{S_{ABC}} = \frac{(\mathbf{m}, \mathbf{f})}{(\mathbf{m}, \mathbf{e}_1)}$$

Если в последней формуле поменять ролями $\mathbf{e}_1$ и $\mathbf{e}_2$, то получится формула для γ.

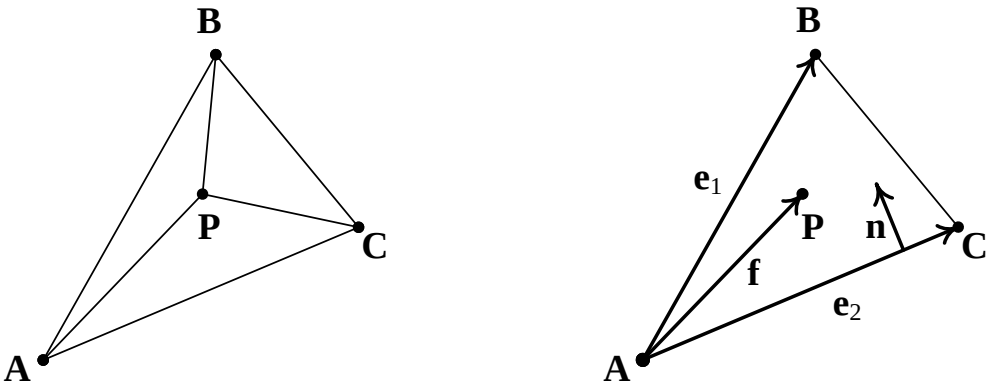


Рис. 4.14: Вычисление барицентрических координат в $\mathbb{R}^3$

Так как точка $\mathbf{P}$ может меняться внутри $\triangle ABC$, а сам $\triangle ABC$ остаётся при этом неизменным, то удобно в самом начале вычислений получить два вспомогательных вектора

$$\mathbf{u}_\beta = \frac{(\mathbf{e}_2, \mathbf{e}_2)\mathbf{e}_1 - (\mathbf{e}_1, \mathbf{e}_2)\mathbf{e}_2}{(\mathbf{e}_1, \mathbf{e}_1)(\mathbf{e}_2, \mathbf{e}_2) - (\mathbf{e}_1, \mathbf{e}_2)^2} \quad \text{и} \quad \mathbf{u}_\gamma = \frac{(\mathbf{e}_1, \mathbf{e}_1)\mathbf{e}_2 - (\mathbf{e}_1, \mathbf{e}_2)\mathbf{e}_1}{(\mathbf{e}_1, \mathbf{e}_1)(\mathbf{e}_2, \mathbf{e}_2) - (\mathbf{e}_1, \mathbf{e}_2)^2}. \tag{4.6}$$

С помощью этих векторов можно для любой точки $\mathbf{P}$ *быстро* рассчитать барицентрические координаты

$$\beta = (\mathbf{u}_\beta, \mathbf{f}), \quad \gamma = (\mathbf{u}_\gamma, \mathbf{f}) \quad \text{и} \quad \alpha = 1 - \beta - \gamma. \tag{4.7}$$

Вычислять вектора m и n нет необходимости, достаточно получить только u_β и u_γ. Формулы (4.6), (4.7) дают эффективный способ получения барицентрических координат, работающий в пространстве любой размерности, не только в $\mathbb{R}^3$.

4.4 Пусть $A = \langle 0, 0\rangle$, $B = \langle 2, 3\rangle$ и $= \langle 3, 1\rangle$. Определить барицентрические координаты следующих точек: а) $P = \langle 2, 3\rangle$; б) $P = \langle 1\frac{1}{3}, 2\rangle$; в) $P = \langle \frac{3}{2}, \frac{3}{2}\rangle$; д) $P = \langle 1, 0\rangle$.

1.4.2 Билинейная интерполяция

Часто возникает потребность интерполировать между 4 точками. Очевидно, 4 точки можно рассматривать как две группы по 3 точки (рис. 4.15) и применить к каждой группе интерполяцию по трём точкам. Если все 4 точки лежат в одной плоскости, то результат не будет зависеть от разбиения. Однако, если 4 точки лежат не на плоской двумерной поверхности, то результаты интерполяции по трём точкам для разбиений на рис. 4.15.*а* и 4.15.*б* будут отличаться.

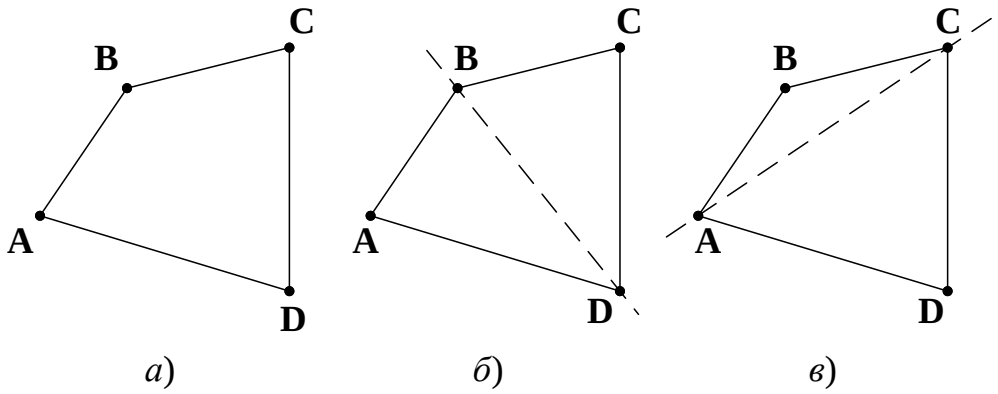


Рис. 4.15: Интерполяция между 4 точек. *а*) Исходные 4 точки; *б*),*в*) два варианта разбиения 4 точек на две группы из 3 точек.

Рассмотрим пример поверхности, являющейся объединением непересекающихся четырёхугольников (возможно с непрямыми углами в вершинах и неплоских, например, рис. 4.16). Требуется параметризовать четырёхугольники величинами α и β из отрезка $[0, 1]$. Такая задача возникает при наложении изображения (текстуры) на поверхность.

Чтобы интерполировать между 4 вершинами существует метод *билинейной интерполяции*. Метод позволяет получить гладкую поверхность между 4 вершинами. Пары вершин, «не лежащих на диагоналях»,соединены между собой отрезками прямых.

Рассмотрим 4 точки, изображённые на рис. 4.17. Нам нужно параметризовать точки внутри поверхности двумя скалярными величинами α и β из $[0, 1]$. Другими словами требуется построить отображение, переводящее квадрат $[0, 1]\times[0, 1]$ в поверхность так, чтобы вершины и рёбра квадрата и поверхности совпадали (рис. 4.18).

Билинейная интерполяция определяется следующим образом

$$\begin{aligned}\mathbf{P} &= (1-\beta)\cdot[(1-\alpha)\mathbf{A} + \alpha\mathbf{B}] + \beta\cdot[(1-\alpha)\mathbf{D} + \alpha\mathbf{C}] = \\ &= (1-\alpha)\cdot[(1-\beta)\mathbf{A} + \beta\mathbf{D}] + \alpha\cdot[(1-\beta)\mathbf{B} + \beta\mathbf{C}] = \\ &= (1-\alpha)(1-\beta)\mathbf{A} + \alpha(1-\beta)\mathbf{B} + \alpha\beta\mathbf{C} + (1-\alpha)\beta\mathbf{D},\end{aligned} \tag{4.8}$$

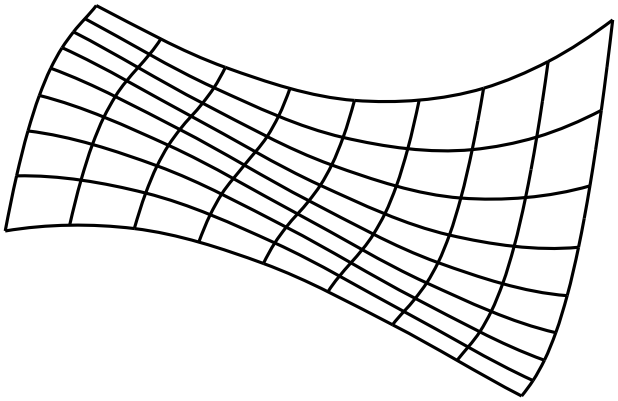


Рис. 4.16: Пример поверхности, являющейся объединением непересекающихся четырёхугольников.

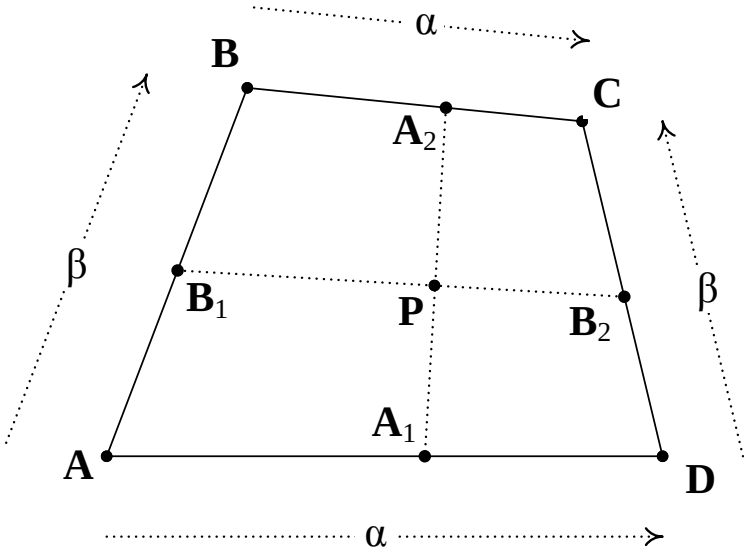


Рис. 4.17: Точка $\mathbf{P} = \mathbf{P}(\alpha, \beta)$ получена с помощью билинейной интерполяции со скалярными параметрами α и β. Точки $\mathbf{A}_1$ и $\mathbf{A}_2$ интерполяцией с параметром α, точки $\mathbf{B}_1$ и $\mathbf{B}_2$ интерполяцией с параметром β.

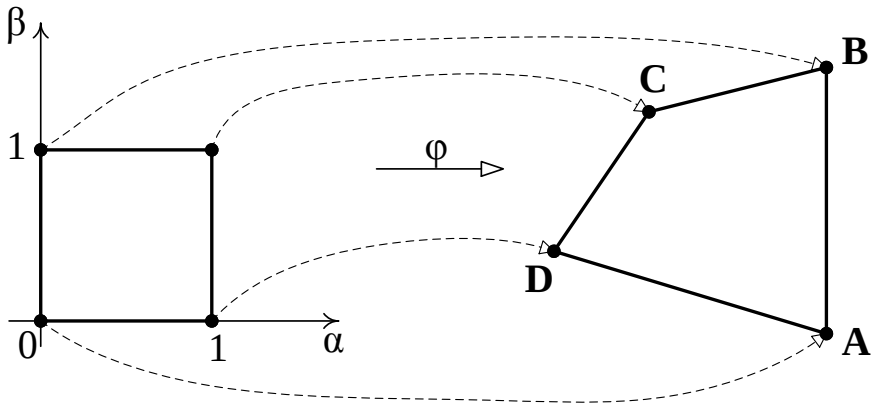


Рис. 4.18: Пример отображения $\varphi: \ [0,1] \times [0,1] \mapsto ABCD$.

где $0 \leqslant \alpha, \beta \leqslant 1$. Получаем, что $\mathbf{P}$ — это взвешенное среднее вершин $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ и $\mathbf{D}$.

Мы определили билинейное отображение тремя эквивалентными способами, чтобы подчеркнуть возможность вначале линейно интерполировать по α, а затем линейно интерполировать по β, или наоборот, вначале по β, затем по α.

Билинейная интерполяция может быть использована для интерполяции значений функции $f(\mathbf{x})$. Если известны значения функции в четырёх точках A, B, C и D, то значение функции в точке P можно приблизить выражением, походим на (4.8),

$$f(P) = (1-\alpha)(1-\beta)f(A) + \alpha(1-\beta)f(B) + \alpha\beta f(C) + (1-\alpha)\beta f(D).$$

4.5 $A = \langle 0, 0\rangle$, $B = \langle 4, 0\rangle$, $C = \langle 5, 3\rangle$ и $D = \langle 0, 2\rangle$. Для заданных α и β определить точку при билинейной интерполяции: а) $\alpha = 1$ и $\beta = 0$; б) $\alpha = \frac{1}{3}$ и $\beta = 1$; в) $\alpha = \frac{1}{2}$ и $\beta = \frac{1}{4}$; д) $\alpha = \frac{2}{3}$ и $\beta = \frac{1}{3}$.

1.4.3 Обращение билинейной интерполяции

Рассмотрим вначале случай билинейной интерполяции на плоскости. Для обращения билинейной интерполяции рассмотрим два вектора

$$\mathbf{s}_1(\beta) = \mathbf{x} + \beta\mathbf{s}_1 \text{ и } \mathbf{s}_2(\beta) = \mathbf{y} + \beta\mathbf{s}_2,$$

как показано на рис. 4.19. Требуется определить значение $\beta \in [0, 1]$. Для этого достаточно потребовать, чтобы точки $\mathbf{s}_1(\beta)$, $\mathbf{u}$ и $\mathbf{s}_2(\beta)$ лежали на одной прямой.

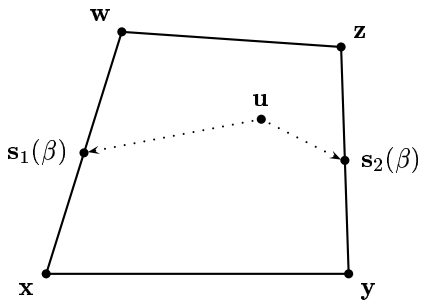


Рис. 4.19: При истинном значении β три точки $\mathbf{s}_1(\beta)$, $\mathbf{u}$ и $\mathbf{s}_2(\beta)$ лежат на одной прямой. Значение β на рисунке меньше истинного значения для точки $\mathbf{u}$.

Два вектора коллинеарны, если и только если их векторное произведение равно $\mathbf{0}$. Поэтому точки $\mathbf{s}_1(\beta)$, $\mathbf{u}$ и $\mathbf{s}_2(\beta)$ лежат на одной прямой, когда

$$\begin{aligned}\mathbf{0} = (\mathbf{s}_1(\beta) - \mathbf{u}) \times (\mathbf{s}_2(\beta) - \mathbf{u}) &= (\beta\mathbf{s}_1 - (\mathbf{u} - \mathbf{x})) \times (\beta\mathbf{s}_2 - (\mathbf{u} - \mathbf{y})) = \\ &= (\mathbf{s}_1 \times \mathbf{s}_2)\beta^2 + [\mathbf{s}_2 \times (\mathbf{u} - \mathbf{x}) - \mathbf{s}_1 \times (\mathbf{u} - \mathbf{y})]\beta + (\mathbf{u} - \mathbf{x}) \times (\mathbf{u} - \mathbf{y}). \quad (4.9)\end{aligned}$$

Обозначим

$$\begin{aligned}A &= \mathbf{s}_1 \times \mathbf{s}_2 = (\mathbf{w} - \mathbf{x}) \times (\mathbf{z} - \mathbf{y}),\\ B &= (\mathbf{z} - \mathbf{y}) \times (\mathbf{u} - \mathbf{x}) - (\mathbf{w} - \mathbf{x}) \times (\mathbf{u} - \mathbf{y}),\\ C &= (\mathbf{u} - \mathbf{x}) \times (\mathbf{u} - \mathbf{y}).\end{aligned}$$

Тогда (4.9) перейдёт в

$$A\beta^2 + B\beta + C = 0.$$

Откуда

$$\beta = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}.$$

Из двух корней нужно выбрать один, удовлетворяющий условию $0 \leqslant \beta \leqslant 1$. Оказывается, что это делать необязательно, так как истинное значение β всегда равно

$$\beta = \frac{-B - \sqrt{B^2 - 4AC}}{2A}. \tag{4.10}$$

Сделаем теперь небольшое изменение с целью получения более устойчивых к погрешности округления результатов. Если $B < 0$ и $B^2 \gg 4AC$, то в числителе (4.10) будет разность двух больших чисел. В этом случае компьютер может вернуть число близкое к нулю. Выйти из положения позволяет формула

$$\beta = \frac{2C}{-B + \sqrt{B^2 - 4AC}}. \tag{4.11}$$

Заметим, что формулы (4.10) и (4.11) эквиваленты.

Когда значение получено, займёмся поиском α. Точка $\mathbf{u}$ есть взвешенная сумма $\mathbf{s}_1(\beta)$ и $\mathbf{s}_2(\beta)$. Отсюда

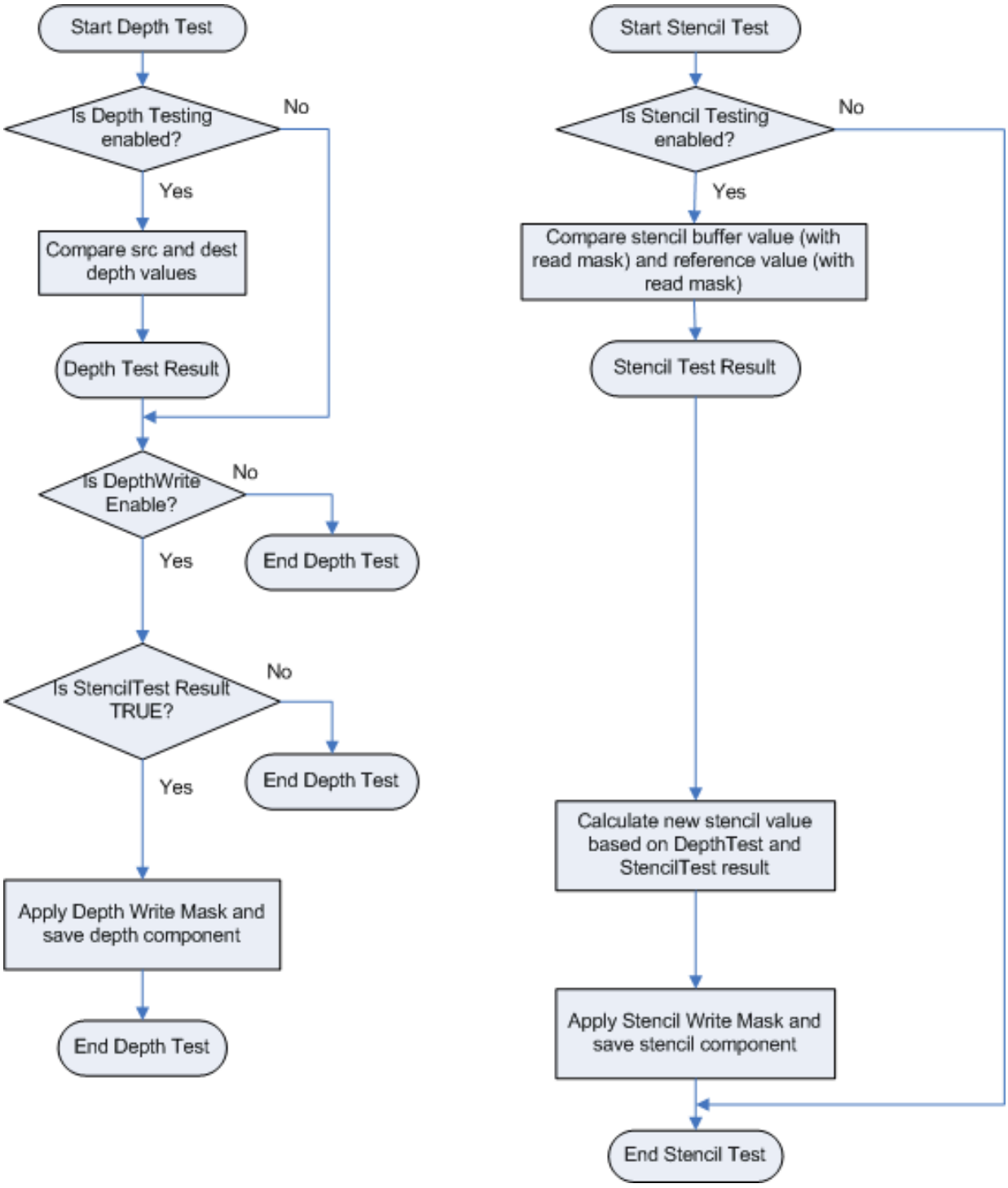
$$\alpha = \frac{(\mathbf{u} - \mathbf{s}_1(\beta)) \cdot (\mathbf{s}_2(\beta) - \mathbf{s}_1(\beta))}{(\mathbf{s}_2(\beta) - \mathbf{s}_1(\beta))^2}.$$

Для случая трехмерного пространства алгоритм сохраняет свою силу, но требуется слегка изменить значения A, B и C. Сформулируем теперь алгоритм обращения билинейной интерполяции:

```
// x, y, x, w, u ∈ ℝ
BilinearInvert( u, x, y, z, w ) {
    n = (z − x) × (w − y);
    A = n · ((w − x) × (z − y));
    B = n · ((z − y) × (u − x) − (w − x) × (u − y));
    C = n · ((u − x) × (u − y));
    if ( B > 0 ) {
        β = (−B − √(B² − 4AC)) / 2A;
    } else {
        β = 2C / (−B + √(B² − 4AC));
    }
    s1,β = (1 − β)x + βw;
    s2,β = (1 − β)y + βz;
    α = ((u − s1,β) · (s2,β − s1,β)) / (s2,β − s1,β)²;
    return α и β; // обращение билинейной интерполяции
}
```

1.5 Занятие 5

1.5.1 Буферы глубины (z-buffer) и трафарета (stencil buffer)



1.5.2 Рельефное текстурирование (bump mapping)

Рельефное текстурирование используется, чтобы добавить к гладкой поверхности неровности и зазубрины. Представление всех маленьких зазубрин и неровностей с

помощью полигонов обычно оказывается затратным и неоптимальным решением, так как необходимо использовать огромное количество многоугольников. В рельефном текстурировании применяется совершенно иной подход, а именно на гладкую поверхность накладывается специальная текстура, которая называется *картой высот*. Это напоминает географическую карту, где высота местности обозначается различными цветами. Карта высот — это необычная текстура. Её наложение на поверхность меняет не цвет последней, а её нормали. Как известно, в модели Фонга отражение света поверхностью определяется направлением нормалей в её вершинах. В результате получается имитация неровностей и зазубрин на гладкой поверхности. *В отличие от обычной текстуры, при перемещении наблюдателя, освещенность рельефа будет изменяться, так как изменяются углы между нормалями к поверхности и направлением на наблюдателя.*

Рис. 5.20: Тор с рельефной текстурой. Обратите внимание на отсутствие неровностей на силуэте (контуре). Сцену освещает 4 источника белого света, а также слабое внешнее освещение.

На рис. 5.20 представлен пример наложения рельефной текстуры. Хорошо видно, что силуэт (контур) тора гладкий без неровностей. Это доказывает, что геометрическая модель поверхности гладкая. Видимые выпуклости — это реакция модели Фонга на то, что нормали вершин были модифицированы картой высот.

Рельефное текстурирование впервые описал Blinn в 1978 году. Рассмотрим основные этапы этого метода. Пусть поверхность задана параметрической функцией $\mathbf{p}(u, v)$. Предположим, что частные производные $\mathbf{p}_u = \frac{\partial \mathbf{p}}{\partial u}$ и $\mathbf{p}_v = \frac{\partial \mathbf{p}}{\partial v}$ определены в области определения функции поверхности $\mathbf{p}(u, v)$, отличны от нуля, и мы можем их вычислить (например, численно). Единичный вектор нормали к поверхности равен

$$\mathbf{n}(u, v) = \frac{\mathbf{p}_u \times \mathbf{p}_v}{|\mathbf{p}_u \times \mathbf{p}_v|}.$$

Рельефная текстура — это матрица скалярных величин $d(u, v)$, представляющая собой смещение в направлении нормали. То есть каждая точка поверхности $\mathbf{p}(u, v)$ получает мнимое смещение $d(u, v)$ в направлении нормального вектора (рис. 5.21). Следует особо подчеркнуть, что смещение мнимое. В действительности при отрисовке на компьютере поверхность $\mathbf{p}$ не искажается, текстурой. Мы лишь мысленно допускаем эти искажения, чтобы рассчитать нормали к искажённой поверхности.

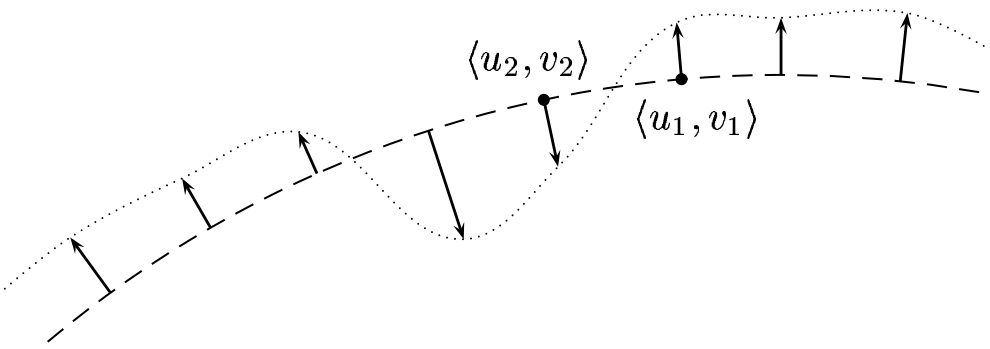


Рис. 5.21: Штриховая линия обозначает номер разреза двумерной поверхности. Поверхность перпендикулярна расстоянию $d(u,v)$, формирующему кривую из точек. Лицевая сторона поверхности находится сверху. Поэтому значение $d(u_1,v_1) > 0$, а значение $d(u_2,v_2) < 0$.

Формула для искажённой поверхности

$$\mathbf{p}^*(u,v) = \mathbf{p} + d\mathbf{n}.$$

Нормали к искажённой поверхности вычислим следующим образом. Вначале найдём первые производные

$$\frac{\partial p^*}{\partial u} = \frac{\partial p}{\partial u} + \frac{\partial d}{\partial u}\mathbf{n} + d\frac{\partial \mathbf{n}}{\partial u}, \tag{5.12}$$

$$\frac{\partial p^*}{\partial v} = \frac{\partial p}{\partial v} + \frac{\partial d}{\partial v}\mathbf{n} + d\frac{\partial \mathbf{n}}{\partial v}. \tag{5.13}$$

Если векторно перемножить (5.12) и (5.13), получим нормаль к искажённой поверхности. Упростим частные производные, отбросив последнее слагаемое

$$\frac{\partial \mathbf{p}^*}{\partial u} \approx \frac{\partial \mathbf{p}}{\partial u} + \frac{\partial d}{\partial u}\mathbf{n},$$

$$\frac{\partial \mathbf{p}^*}{\partial v} \approx \frac{\partial \mathbf{p}}{\partial v} + \frac{\partial d}{\partial v}\mathbf{n}.$$

Такое отбрасывание можно обосновать тем, что расстояния $d(u,v)$, формирующие рельеф, малы (размер зазубрин и неровностей небольной). Кроме этого частные производные $\mathbf{n}$ невелики, если основная поверхность $\mathbf{p}(u,v)$ относительно гладкая. Заметим однако, что $\frac{\partial d}{\partial u}$ и $\frac{\partial d}{\partial v}$ нельзя считать малыми, так как неровности рельефа могут иметь существенный наклон (крутизну). С учётом всего выше сказанного, мы можем приблизить нормаль к искажённой поверхности как

$$\mathbf{m} \approx \left(\frac{\partial \mathbf{p}}{\partial u} + \frac{\partial d}{\partial u}\mathbf{n}\right) \times \left(\frac{\partial \mathbf{p}}{\partial v} + \frac{\partial d}{\partial v}\mathbf{n}\right) =$$

$$= \left(\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}\mathbf{n}\right) + \left(\frac{\partial d}{\partial u}\mathbf{n} \times \frac{\partial \mathbf{p}}{\partial v}\right) - \left(\frac{\partial d}{\partial v}\mathbf{n} \times \frac{\partial \mathbf{p}}{\partial u}\right). \tag{5.14}$$

Вектор $\mathbf{m}$ перпендикулярен к искажённой поверхности, но ненормированный. Поэтому рассмотрим вектор $\mathbf{n}^* = \mathbf{m}/|\mathbf{m}|$.

Отметим, что в (5.14) входят только частные производные функции $d(u,v)$, а значения самой $d(u,v)$ не используются. Частные производные можно было бы вычислить с помощью разделённых разностей[1]. Однако более прямой путь — это не хранить матрицу значений $d(u,v)$, а использовать две матрицы частных производных.

Алгоритм вычисления $\mathbf{n}^*$ не работает, если $\frac{\partial \mathbf{p}}{\partial u}$ или $\frac{\partial \mathbf{p}}{\partial v}$ равны нулю. Такое явление происходит в особых точках и на практике случается довольно часто. Например, северный и южный полюсы сферы, заданной в сферической или полярной системе координат, являются особыми точками. Таким образом следует проявлять осторожность при рельефном текстурировании вблизи точек с нулевыми частными производными.

Выше считалось, что функция рельефа $d(u,v)$ зависит от переменных u и v. Иногда более удобно вместо u и v использовать текстурные координаты s и t. При этом вместо $d(u,v)$ будем рассматривать функцию $D(s,t)$. Текстурные координаты можно выразить через u и v так $s = s(u,v)$ и $t = t(u,v)$, где зависимости линейные или билинейные. Отсюда

$$d(u,v) = D(s(u,v), t(u,v)),$$
$$\frac{\partial d}{\partial u} = \frac{\partial D}{\partial s}\frac{\partial s}{\partial u} + \frac{\partial D}{\partial t}\frac{\partial t}{\partial u},$$
$$\frac{\partial d}{\partial v} = \frac{\partial D}{\partial s}\frac{\partial s}{\partial v} + \frac{\partial D}{\partial t}\frac{\partial t}{\partial v}.$$

Частные производные функций s и t — это константы в случае линейной интерполяции или линейная комбинация u и v в случае билинейной интерполяции.

При рельефном текстурировании возникает проблема ступенчатости, когда от наблюдателя до поверхности значительное расстояние. Особенно, когда зазубрины целиком умещаются в один пиксель экрана. Для решения проблемы динамически уменьшают высоты зазубрин при удалении поверхности от наблюдателя.

1.5.3 Эффект зеркального отражения (environment mapping или reflection mapping)

Зеркального отражение с помощью текстуры (environment mapping или reflection mapping) предназначено для отображения на экране блестящих объектов, к которых отражается окружающая их сцена. Зеркальное отражение на основе текстуры не является ресурсоёмким методом в сравнении с методом трассировки лучей, но позволяет достигать хороших результатов для некрупных блестящих объектов.

Основная идея зеркального текстурирования следующая. Пусть у нас есть небольшой объект из материала со свойством зеркального отражения. Например, маленькое плоское или сферическое зеркало (как зеркало заднего вида в автомобиле) или небольшой объект с зеркальной поверхностью такой как блестящий чайник, ложка,

[1] *Разделённые разности* позволяют численно получать значения производной функции $y(x)$, например, для малых значений h справедливо

$$y'(x) = \frac{y(x+h) - y(x)}{h} + O(h) \qquad \text{и} \qquad y'(x) = \frac{y(x+h) - y(x-h)}{2h} + O(h^2).$$

Рис. 5.22: Изображение окружающего мира для сферической проекции. Этот тип зеркального текстурирования поддерживается OpenGL. Сцена та же что и на рис. 5.23. Отметим, что передняя стена имеет (с надписью VIEWER) имеет наибольшую достоверность, а задняя — наименьшую. Из-за этого сферическое наложение текстуры лучше всего подходит, когда направление обзора близко к направлению, которое используется при подготовке зеркальной текстуры.

хромированный водопроводный кран или серебряный бокал. Далее мы мысленно переносимся в центр объекта и фотографируем оттуда окружающий мир. На основе этих фотографий формируется зеркальная текстура (рис. 5.22 и 5.23).

Когда компьютер рисует вершину зеркального объекта, в расчёт берутся положение наблюдателя, координаты вершины, нормаль к поверхности в данной вершине. В начале определяется *отражённое направление обзора*, то есть вектор от вершины до наблюдателя зеркально отражается относительно нормали к вершине. Если представить, что от наблюдателя вышел луч света к вершине, то после отражения его направление совпадёт с этим отражённым направлением обзора. С помощью последнего вектора определяется координата на зеркальной текстуре.

Рассмотрим реализацию в OpenGL:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

При этом используются следующие формулы:

$$\mathbf{f} = \mathbf{u} - 2\mathbf{n}'\mathbf{n}'^{\top}\mathbf{u},$$

где $\mathbf{u}$ — вектор от начала видовых координат к вершине, $\mathbf{n}'$ — текущая нормаль после преобразования в видовые координаты, $\mathbf{f} = (f_x, f_y, f_z)^{\top}$ — результирующий вектор, исходя из которого определяется соответствующая координата:

$$s = f_x/m + 0{,}5 \quad \text{и} \quad t = f_y/m + 0{,}5,$$

где $m = 2\sqrt{f_x^2 + f_y^2 + (f_z + 1)^2}$.

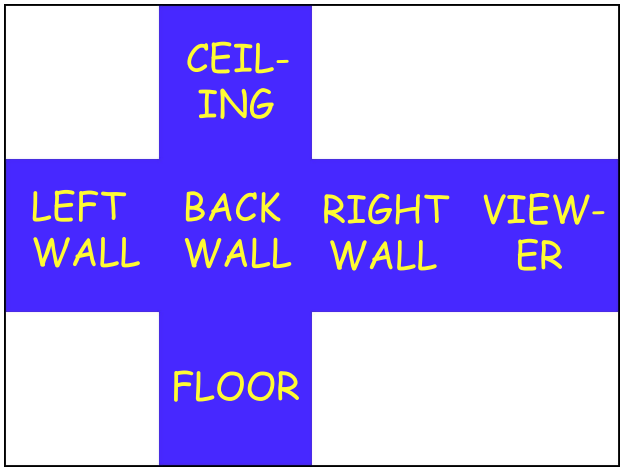


Рис. 5.23: Изображение окружающей мира для коробочной проекции состоит из 6 видов, соответствующих 6 граням куба. Куб развёрнут в плоскую картинку. На текстуре представлено отражение комнаты, полученное из центра комнаты. Стены, пол и потолок окрашены в синий цвет за исключением жёлтых надписей. Белые области текстуры не используются.

1.6 Занятие 6

1.7 Занятие 7

1.8 Занятие 8

1.9 Занятие 9

1.10 Занятие 10

Множество точек пространства $\mathbb{R}^n$ называется *выпуклым*, если оно содержит вместе с любыми двумя точками соединяющий их отрезок. *Выпуклой оболочкой* множества G называется наименьшее выпуклое множество, содержащее G.

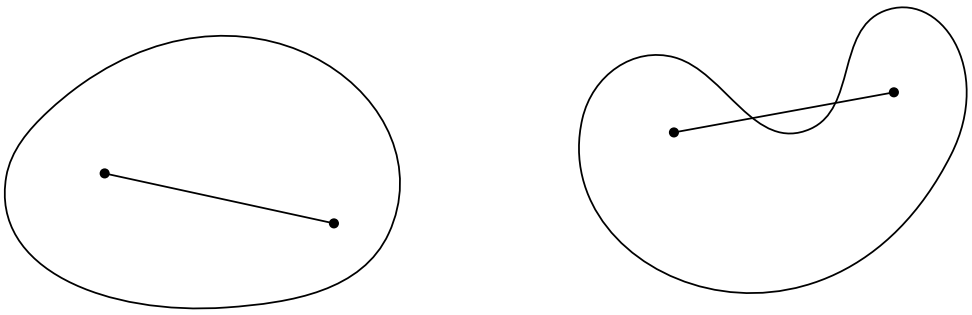


Рис. 10.24: Примеры выпуклового G_1 и невыпуклового G_2 множеств.

Многочлены Бернштейна степени k имеют вид

$$B_i^k(u) = C_k^i u^i (1-u)^{k-i}, \quad C_k^i = \frac{k!}{i!(k-i)!}.$$

Кривая Безье степени k ($k \geqslant 1$), заданная своей $k+1$ опорной точкой $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k$ имеет вид

$$\mathbf{q}(u) = \sum_{i=0}^{k} B_i^k(u)\mathbf{p}_i, \quad u \in [0,1].$$

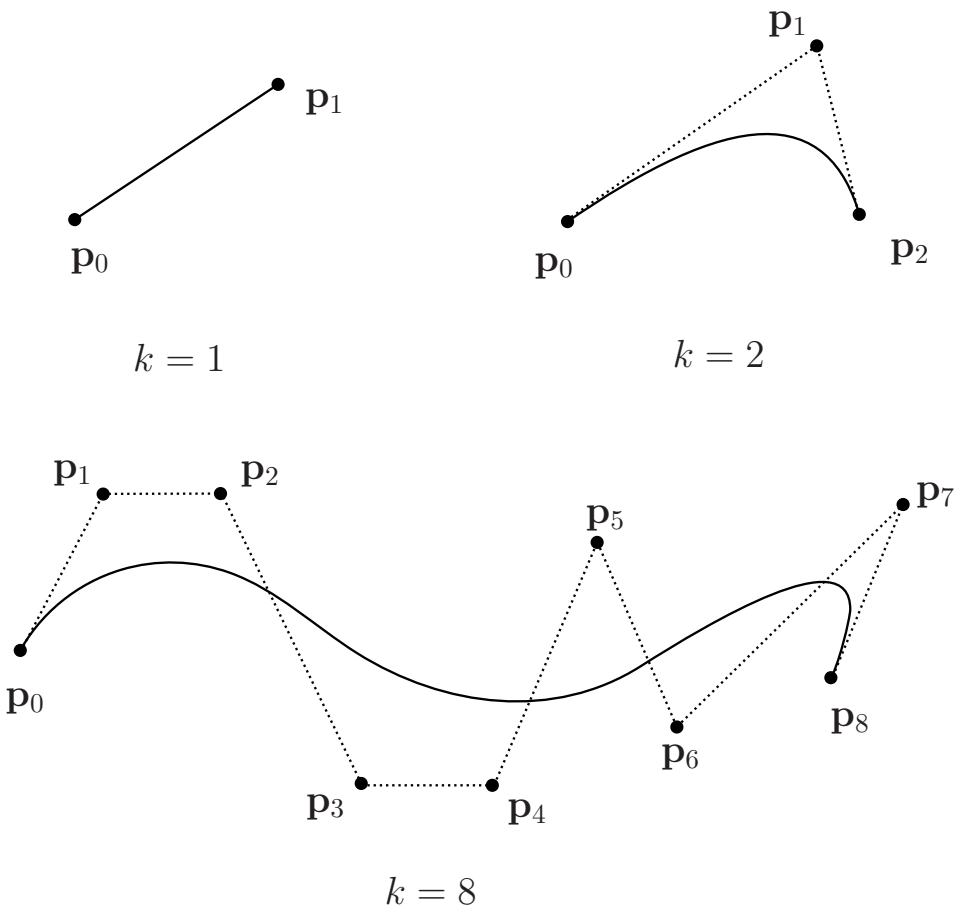


Рис. 10.25: Примеры кривых Безье различной степени k.

Теорема 0.1. *Кривая Безье обладает следующими свойствами:*

(а) $B_0^k(0) = 1 = B_k^k(1)$;

(б) $\sum_{i=0}^{k} B_i^k(u) = 1$ *для любого* u;

(в) $B_i^k(u) \geqslant 0$ *для всех* $0 \leqslant u \leqslant 1$.

Теорема 0.2. *Пусть* $\mathbf{q}(u)$ *есть кривая Безье степени* k *с опорными точками* $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k$. *Тогда*

$$\mathbf{q}'(u) = k \cdot \sum_{i=0}^{k-1} B_i^{k-1}(u)(\mathbf{p}_{i+1} - \mathbf{p}_i).$$

Следствие 0.3. *Для кривой Безье степени k справедливо следующее:*

$$\mathbf{q}'(0) = k(\mathbf{p}_1 - \mathbf{p}_0), \qquad \mathbf{q}'(1) = k(\mathbf{p}_k - \mathbf{p}_{k-1}).$$

Назовём *опорной ломаной* совокупность отрезков, соединяющих опорные точки $\mathbf{p}_{i-1}$ и $\mathbf{p}_i$, $i = 1, 2, \ldots, k$. На рис. 10.25 опорная ломаная изображена пунктиром. Кривая Безье, заданная в пространстве $\mathbb{R}^n$ обладает свойством *наименьшей вариации*: Всякая гиперплоскость в $\mathbb{R}^n$ пересекает кривую Безье небольшее число раз, чем та же гиперплоскость пересекает опорную ломаную. Например, в $\mathbb{R}^2$ гиперплоскость представляет из себя прямую. На рис. 10.26 количество пересечений прямой l и кривой Безье равно 3, что не превосходит количество пересечений прямой l и опорной ломаной, то есть 5.

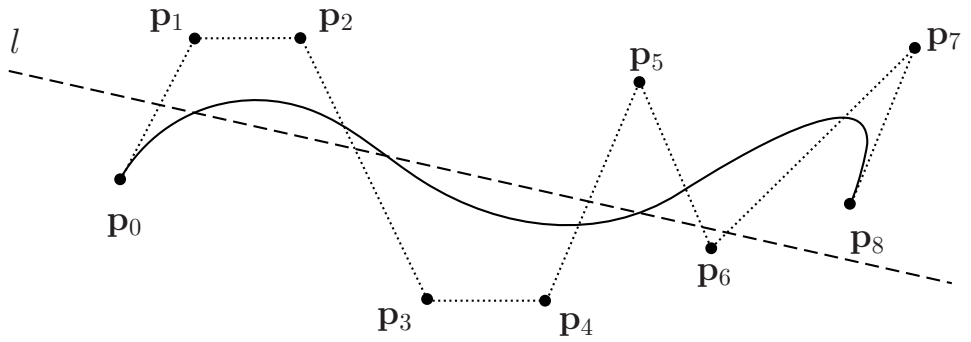


Рис. 10.26: Иллюстрация свойства наименьшей вариации кривой Безье в $\mathbb{R}^2$. Количество пересечений прямой l и кривой Безье равно 3. Количество пересечений прямой l и опорной ломаной равно 5. Справедливо, что $3 \leqslant 5$.

1.10.1 Метод де Кастельжо для кривой Безье произвольной степени.

Пусть кривая Безье $\mathbf{q}(u)$ степени k построена по опорным точкам $\mathbf{p}_1$, $\mathbf{p}_2$, $\ldots$, $\mathbf{p}_k$. Для $u \in [0, 1]$ определим по индукции последовательность точек $\mathbf{p}_i^r(u)$. Положим

$$\begin{aligned} &\mathbf{p}_i^0(u) = \mathbf{p}_i, \\ &\mathbf{p}_i^r(u) = (1-u)\mathbf{p}_i^{r-1}(u) + u\mathbf{p}_{i+1}^{r-1}(u), \quad i = 0, 1, \ldots, k-r. \end{aligned}$$

Заметим, что с ростом индекса r диапазон индекса i сужается. Так при $r = k$ мы имеем единственную точку $\mathbf{p}_0^k(u)$.

Теорема 0.4. *Для определённых выше кривой $\mathbf{q}(u)$ и системы точек $\mathbf{p}_i^r(u)$ справедливо равенство $\mathbf{q}(u) = \mathbf{p}_0^k(u)$.*

1.10.2 Рекурсивное деление кривой Безье.

Пусть кривая Безье $\mathbf{q}(u)$ степени k определяется контрольными точками $\mathbf{p}_i$, $i = \overline{0, k}$. Зафиксируем $u \in [0, 1]$. Определим точки $\mathbf{p}_i^r(u)$ следующим образом. Во-первых, при $r = 0$ положим $\mathbf{p}_i^0(u) = \mathbf{p}_i$. Во-вторых, при $r > 0$ и $0 \leqslant i \leqslant k - r$ положим

$$\mathbf{p}_i^r(u) = (1-u)\mathbf{p}_i^{r-1}(u) + u\mathbf{p}_{i+1}^{r-1}(u).$$

Кривая Безье $\mathbf{q}(u)$ степени k точкой $\mathbf{q}_0 = \mathbf{q}(u_0)$ делится на две части

$$\mathbf{q}_1(u) = \mathbf{q}(u_0 u) \qquad \mathbf{q}_2(u) = \mathbf{q}(u_0 + (1 - u_0)u).$$

Теорема 0.5. *Для определённых выше кривых* $\mathbf{q}$, $\mathbf{q}_1$, $\mathbf{q}_2$ *и точек* $\mathbf{p}_i^r$ *справедливо следующее:*

(а) кривая $\mathbf{q}_1(u)$ *есть кривая Безье степени* k*, построенная на опорных точках* $\mathbf{p}_0^0, \mathbf{p}_0^1, \mathbf{p}_0^2, \ldots, \mathbf{p}_0^k$*;*

(б) кривая $\mathbf{q}_2(u)$ *есть кривая Безье степени* k*, построенная на опорных точках* $\mathbf{p}_0^k, \mathbf{p}_1^{k-1}, \mathbf{p}_2^{k-2}, \ldots, \mathbf{p}_k^0$.

1.10.3 Повышение степени

Пусть задана кривая Безье $\mathbf{q}(u)$ степени k. Может оказаться так, эту же кривую $\mathbf{q}(u)$ можно рассматривать как кривую Безье степени $k+1$. Если последнее выполнимо, то говорят о *повышении степени* кривой. Повышение степени может оказаться полезным, когда по каким-то причинам требуется кривая большей степени. Например, окружность можно приблизить кривой Безье 2-ой степени, но язык PostScript поддерживает только кривые степени 3. В последнем случае требуется повысить степень кривой на единицу.

Сформулируем более строго задачу повышения степени кривой: *Дана кривая Безье* $\mathbf{q}(u)$ *степени* k*, построенная по* k *опорным точкам* $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_k$*. Требуется найти* $k+1$ *опорную точку* $\widehat{\mathbf{p}}_0, \widehat{\mathbf{p}}_1, \ldots, \widehat{\mathbf{p}}_{k+1}$ *новой кривой* $\widehat{\mathbf{q}}(u)$ *так, чтобы* $\widehat{\mathbf{q}}(u) = \mathbf{q}(u)$.

Оказывается, что степень любой кривой Безье всегда можно повысить.

Пример.

$$\mathbf{q}'(0) = 2(\mathbf{p}_1 - \mathbf{p}_0)$$

$$\mathbf{q}'(1) = 2(\mathbf{p}_2 - \mathbf{p}_1)$$

$$\widehat{\mathbf{p}}_1 = \widehat{\mathbf{p}}_0 + \frac{1}{3}\mathbf{q}'(0) = \frac{1}{3}\mathbf{p}_0 + \frac{2}{3}\mathbf{p}_1$$

$$\widehat{\mathbf{p}}_2 = \widehat{\mathbf{p}}_3 - \frac{1}{3}\mathbf{q}'(1) = \frac{2}{3}\mathbf{p}_1 + \frac{1}{3}\mathbf{p}_2$$

В случае кривой произвольной степени k формулы повышения степени имеют следующий вид:

$$\widehat{\mathbf{p}}_0 = \mathbf{p}_0, \qquad \widehat{\mathbf{p}}_i = \frac{i}{k+1}\mathbf{p}_{i-1} + \frac{k-i+1}{k+1}\mathbf{p}_i, \qquad \widehat{\mathbf{p}}_{k+1} = \mathbf{p}_k.$$

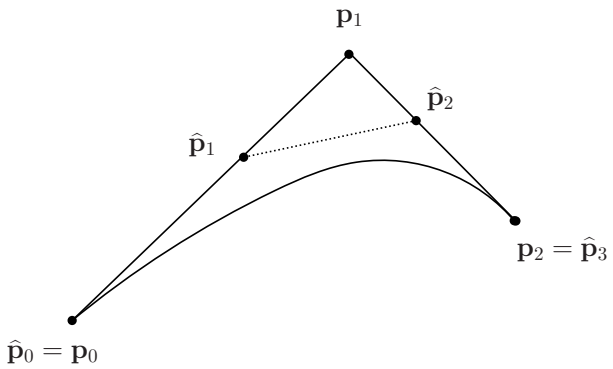


Рис. 10.27: Кривая $\mathbf{q}(u)$ 2-ой степени построена на $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$, кривая $\widehat{\mathbf{q}}(u)$ 3-ей степени построена на $\widehat{\mathbf{p}}_0, \widehat{\mathbf{p}}_1, \widehat{\mathbf{p}}_2, \widehat{\mathbf{p}}_3$. Кривые совпадают: $\mathbf{q}(u) = \widehat{\mathbf{q}}(u)$.

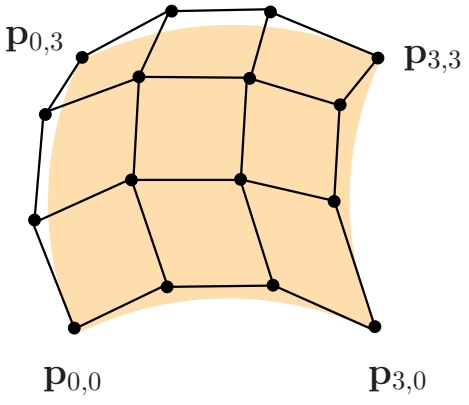


Рис. 10.28: Поверхность Безье степени 3.

1.10.4 Поверхности Безье.

$$\mathbf{q}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_i^3(v)\mathbf{p}_{i,j}.$$

1.10.5 Поддержка работы с объектами Безье в OpenGL

В составе OpenGL имеются средства поддержки работы с кривыми и поверхностями Безье — Безье-вычислитель, которые позволяют вычислять значения полиномов Безье любого порядка. Безье вычислитель можно использовать для работы с полиномами от одной, двух, трёх и четырёх переменных.

Для каждого значения u и v формула $\mathbf{q}(u)$ или $\mathbf{q}(u,v)$ вычисляет точку на кривой или поверхности. При использовании Безье-вычисления сначала выбирают функцию $\mathbf{q}(u)$ или $\mathbf{q}(u,v)$, включают её (Безье-вычислитель), а затем используют команду `glEvalCoord1()` или `glEvalCoord2()` вместо команды `glVertex*()`. В этом случае вершина кривой или поверхности может использоваться точно также, как и любая другая вершина, например, для формирования точки или линии. Кроме того, другие команды автоматически генерируют серии вершин, образующих пространство регулярной однородной сетки по оси u (или по осям u и v).

Функция обработки полинома одной переменной настраивается в процессе инициализации OpenGL- программы посредством вызова функции

`glMap1f(type, u_min, u_max, stride, order, pointarray)`.

Аргумент `type` задаёт тип объекта, который будет представлен полиномом Безье. Можно назначить в качестве значения этого аргумента константы, задающие трёх- и четырёхмерные геометрические точки, цвет в формате RGBA, нормали, индексированные цвета и координаты текстур (от одно- до четырёхмерных).

Parameter	**Meaning**
`GL_MAP1_VERTEX_3`	x, y, z координаты вершин
`GL_MAP1_VERTEX_4`	x, y, z, w координаты вершин
`GL_MAP1_INDEX`	индекс цвета
`GL_MAP1_COLOR_4`	R, G, B, A
`GL_MAP1_NORMAL`	координаты нормали
`GL_MAP1_TEXTURE_COORD_1`	s координаты текстуры
`GL_MAP1_TEXTURE_COORD_2`	s, t координаты текстуры
`GL_MAP1_TEXTURE_COORD_3`	s, t, r координаты текстуры
`GL_MAP1_TEXTURE_COORD_4`	s, t, r, q координаты текстуры

Таблица 1.1: Типы опорных точек для `glMap1*()`

Указатель на массив опорных точек полинома передаётся функции через аргумент `point_array`. Аргументы `u_min`, `u_max` определяют область существования парамет-

ра полинома. Аргумент `stride` представляет собой количество значений параметра между сегментами кривой. Значение аргумента `order` должно быть равно количеству опорных точек. Для формирования кубической трёхмерной кривой, определённой на интервале $(0, 1)$, функции `glMap1f`() следует передать такой набор аргументов:

```
point data[]= {.};
glMap1f(GL_MAP_VERTEX_3, 0.0, 1.0, 3, 4, data);
```

После настройки функция активизируется посредством вызова `glEnable`(`type`). Если функция расчета активизирована, то можно получить от неё значения полинома, вызвав функцию `glEvalCoord1f`(`u`). Таким образом, обращение к `glEvalCoord1f`() может заменить обращение к функциям `glVertex`(), `glColor`(), `glNormal`(). Пусть, например, функция расчёта настроена на формирование кривой Безье на интервале $(0, 10)$ по некоторому массиву опорных точек. Набор из 100 точек кривой, равноотстоящих на этом интервале можно получить с помощью такого фрагмента программы:

```
glBegin(GL_LINE_STRIP)
   for(i=0; i<100; i++) glEvalCoord1f((float)i/100.);
glEnd();
```

Если значения параметра u распределены равномерно, то для вычисления точек на кривой следует использовать функции `glMapGrid1f`() и `glEvalMesh1`(), например:

```
glMapGrid1f(100, 0.0, 10.0);
glEvalMesh1(GL_LINE, 0, 100);
```

После вызова `glMapGrid1f`() устанавливается равномерная сетка в 100 отсчётов, а после вызова функции `glEvalMesh1`() будет сформирована кривая.

Поверхности Безье формируются в OpenGL примерно по той же методике, что и кривые, только роль функции инициализации играет не `glMap1`*(), а `glMap2`*(), а для считывания результатов следует обращаться к функции `glEvalCoord2`*() вместо `glEvalCoord1`*(). В обеих функциях нужно специфицировать данные, относящиеся к двум независимым параметрам u и v. Например, функция `glMap2f`() имеет такой формат вызова:

`glMap2f`(`type`, `u_min`, `u_max`, `u_stride`, `u_order`,
`v_min`, `v_max`, `v_stride`, `v_order`, `point_array`).

Настройка функции вычисления на работу с бикубической поверхностью Безье, определенной на области $(0, 1) \times (0, 1)$, выполняется таким вызовом `glMap2f`():

`glMap2f`(`GL_MAP_VERTEX_3`, 0.0, 1.0, 3, 4, 0.0, 1.0, 12, 4, `data`);

Для обоих независимых переменных нужно задать порядок полинома (аргументы `u_order` и `v_order`) и количество значений параметра между сегментами (аргументы `u_stride` и `v_stride`), что обеспечивает дополнительную гибкость при формировании поверхности. Обратите внимание на то, что значение `v_stride` для второго параметра равно 12, поскольку в массиве опорных точек `data` данные хранятся по строкам. Поэтому для перехода к следующему элементу этой же строки нужно «перешагнуть»

три числа в формате `float`, а для перехода к следующему элементу в этом же столбце нужно «перешагнуть» через $3 \times 4 = 12$ чисел в формате `float`. Способ вызова программы расчёта зависит от того, какой результат мы хотим получить,- вывести на экран сеть или сформировать многоугольники для последующего раскрашивания. Если ставится задача сформировать на экране сеть, то соответствующий фрагмент программы должен выглядеть примерно так:

```
for(j=0; j<100; j++) {
    glBegin(GL_LINE_STRIP);
    for(i=0; i<100; i++)
        glEvalCoord2f((float)i/100.0, (float)j/100.0);
    glEnd();
    glBegin(GL_LINE_STRIP);
    for(i=0; i<100; i++)
        glEvalCoord2f((float)j/100.0, (float)i/100.0);
    glEnd();
}
```

Если же желательно сформировать множество многоугольников, то фрагмент должен выглядеть так:

```
for(j=0; j<99; j++) {
    glBegin(GL_QUAD_STRIP)
    for(i=0; i<=100; i++) {
        glEvalCoord2f( (float)i/100.0, (float)j/100.0);
        glEvalCoord2f( (float)(i+1)/100.0, (float)j/100.0);
    }
    glEnd();
}
```

Для работы на равномерной сетке следует использовать функции `glMapGrid2*`() и `glEvalMesh2`(). Тогда в самое начало программы, в ту часть, которая отвечает за инициализацию, нужно включить такой фрагмент:

`glMapGrid2f`(100, 0.0, 1.0, 100, 0.0, 1.0);

В функции отображения `display`() нужно вызвать `glEvalMesh2`():

`glEvalMesh2`(`GL_FILL`, 0, 100, 0, 100);

Для работы алгоритмов тонирования (закрашивания) сформированной поверхности при настройке режима учёта освещения нужно дополнительно вызвать функцию `glEnable`(), передав ей в качестве аргумента константу `GL_AUTO_NORMAL`:

`glEnable`(`GL_AUTO_NORMAL`);

Это позволит OpenGL автоматически вычислять вектор нормали к каждому участку формируемой поверхности и использовать этот вектор при закрашивании участков этой поверхности.

1.11 Занятие 11. Пересечение луча с объектами сцены.

Проверка пересечения составляет основу для трассировки лучей, когда требуется проверить пересечение большого числа лучей с большим числом объектов на сцене.

Бóльшая часть вычислительного времени при трассировке лучей тратится на проверку пересечений. Существуют также и другие приложения алгоритмов проверки пересечений, например, организация движения объектов, проверка соударений объектов.

Проверка соударений — это очень обширная тема. Мы коснёмся лишь вопроса пересечения лишь с простыми объектами.

При работе алгоритмов проверки пересечений важную роль играют следующие два фактора: 1. Точность 2. *Скорость*. Часто проверка пересечений составляет существенную часть программы. Например, при трассировке лучей почти всё время тратится именно на неё. Причина этого кроется в большом количестве объектов на сцене (от сотен до тысяч). В трассировке лучей предполагается, что каждый луч может попасть в любой объект на сцене. В действительности луч попадает только в один объект.

Также важно, чтобы алгоритм умел *быстро* отвечать на вопрос, когда пересечения не происходит. Фактически намного важнее уметь быстро определять отсутствие пересечения, чем быстро определять наличие последнего.

Если идёт проверка пересечения луча с многими объектами, то может оказаться полезным иерархически упорядочить эти объекты, либо разбить пространство с помощью октавных деревьев или бинарных разбивающих пространство деревьев.

1.11.1 Пересечение луча и сферы

Пусть луч света и сфера фиксированы. Луч определяется своим началом $\mathbf{p}$ и единичным вектором $\mathbf{u}$, задающим направление луча. Таким образом луч — это множество точек

$$\{\mathbf{p} + \alpha\mathbf{u} \,|\, \alpha \geqslant 0\}.$$

Будем также говорить о прямой, содержащей луч, которую назовём прямой луча.

Сфера задаётся своим центром $\mathbf{c}$ и радиусом $r > 0$.

Найдем, прежде всего, точку $\mathbf{q}$ на прямой луча, ближайшую с центру сферы (рис. 11.29). Очевидно, $\mathbf{q} = \mathbf{p} + \alpha\mathbf{u}$, где величина α определяет расстояние от $\mathbf{q}$ до $\mathbf{p}$. Из построения имеем

$$(\mathbf{q} - \mathbf{c}) \perp \mathbf{u} \quad \Rightarrow \quad 0 = (\mathbf{q} - \mathbf{c}) \cdot \mathbf{u} = (\mathbf{p} + \alpha\mathbf{u} - \mathbf{c}) \cdot \mathbf{u}.$$

Так как $\mathbf{u} \cdot \mathbf{u} = 1$, то $\alpha = -(\mathbf{p} - \mathbf{c}) \cdot \mathbf{u}$. Отсюда $\mathbf{q} = \mathbf{p} - ((\mathbf{p} - \mathbf{c}) \cdot \mathbf{u})\mathbf{q}$.

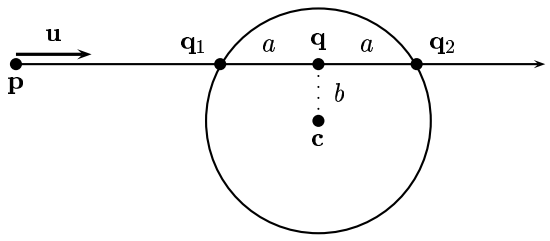


Рис. 11.29: Луч из точки $\mathbf{p}$ с направлением $\mathbf{u}$ имеет ближайшую к центру сферы точку $\mathbf{q}$. Луч пересекает сферу в точках $\mathbf{q}_1$ и $\mathbf{q}_2$.

Когда точка $\mathbf{q}$ найдена, можно проверить, лежит ли $\mathbf{q}$ внутри сферы, то есть

$$|\mathbf{q} - \mathbf{c}| \leqslant r. \tag{11.15}$$

В процессе проверки последнего неравенства требуется вычислять квадратный корень[2]. Чтобы не тратить время процессора на вычисление корня, лучше вместо (11.15) проверить

$$|\mathbf{q}-\mathbf{c}|^2 \leqslant r^2$$

(квадрат длины вектора требует для своего вычисления только 3 умножения, 2 сложения, что эффективнее $\sqrt{\ }$).

Если $\mathbf{q}$ не лежит внутри сферы, что очевидно пересечения нет. Пусть $\mathbf{q}$ лежит внутри сферы, положим $b = |\mathbf{q}-\mathbf{c}|$ и $a = \sqrt{r^2-b^2}$. В общем случае луч пересекает сферу в двух точках:

$$\mathbf{q}_1 = \mathbf{p} + (\alpha - a)\mathbf{u} \quad \text{и} \quad \mathbf{q}_2 = \mathbf{p} + (\alpha + a)\mathbf{u}.$$

Если $\alpha \geqslant a$, то луч падает на сферу в точке $\mathbf{q}_1$. Если $\alpha < a$, то $\mathbf{q}_1$ лежит на отрицательном продолжении луча (позади точки $\mathbf{p}$). В последнем случае проверим неравенство $\alpha + a > 0$. Если оно выполняется, то $\mathbf{p}$ лежит внутри сферы, и $\mathbf{q}_2$ — точка падения луча на внутреннюю поверхность сферы.

Объединим всё сказанное в виде алгоритма:

```
// Пересечение луча и сферы:
// Вход: Луч из точки p в направлении u (|u| = 1).
//       Сфера с центром c и радиуса r.
α = −(p − c) · u;
q = p + αu;
bSq = |q − c|^2;
if ( bSq > r^2 ) {
    return ∅; // пересечения нет
}
a = √(r^2 − bSq);
if ( α ⩾ a ) {
   q1 = q − au;
   return q1;
}
if ( α + a > 0 ) {
   q2 = q + au;
   return q2;
}
return ∅; // пересечения нет
```

Как упоминалось ранее, наиболее важный аспект работы алгоритма — это скорость определения отсутствия пересечения. Предложенный алгоритм для определения непересечения требует (первые 4 строки программы): 9 умножений, 11 сложений, 1 сравнение. Величину r^2 можно вычислить заранее (считаем, что радиус сферы не меняется между кадрами).

Если известно, что $\mathbf{p}$ не может оказаться внутри сферы, имеет смысл добавить проверку $\alpha > 0$, так как $\alpha \leqslant 0$ означает, что луч вообще не направлен в сторону сферы.

[2] Для любого вектора $\mathbf{u}$ его длина равна $|\mathbf{u}| = \sqrt{u_x^2 + u_y^2 + u_z^2}$.

Как видно проверка пересечения луча и сферы для компьютера довольно проста. Из-за этого сферу удобно использовать для определения пересечений луча со объектами сложной формы. Заключим такой объект целиком в сферу. Если луч не пересекает сферу, то он не пересекает и объект. Если же луч сферу пресекает, то далее проверяем пересечение луча с самим объектом. Преимущество такого подхода состоит в том, что мы можем быстро отсеить случаи непересечения.

Отсюда возникает — это довольно простой с точки зрения компьютерных вычислений объект

1.11.2 Пересечение луча и плоскости

Плоскость Γ задаётся вектором нормали $\mathbf{n}$ и числом d. Точки $\mathbf{x}$ плоскости удовлетворяют уравнению $\mathbf{x} \cdot \mathbf{n} = d$.

Пусть по-прежнему задан парой $\mathbf{p}$ и $\mathbf{u}$. Найдём вначале точку $\mathbf{q}$ пересечения (если оно есть) прямой луча и плоскости. Положим $\mathbf{q} = \mathbf{p} + \alpha\mathbf{u}$, $\alpha \in \mathbb{R}$. Очевидно,

$$\mathbf{q} \in \Gamma \quad \Leftrightarrow \quad d = \mathbf{q} \cdot \mathbf{n} = \mathbf{p} \cdot \mathbf{n} + \alpha \mathbf{u} \cdot \mathbf{n}.$$

Откуда

$$\alpha = \frac{d - \mathbf{p} \cdot \mathbf{n}}{\mathbf{u} \cdot \mathbf{n}}.$$

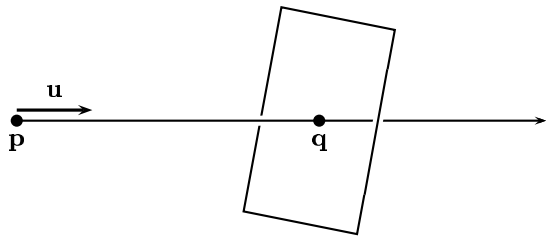


Рис. 11.30: Луч из точки $\mathbf{p}$ с направлением $\mathbf{u}$ пересекает плоскость в точке $\mathbf{q}$.

Плоскость разбивает пространство на две части. Назовём полупространство, куда направлена нормаль $\mathbf{n}$ пространством «над» плоскостью, а оставшееся пространство пространством «под» плоскостью. Несложно проверить следующее:

$$d - \mathbf{p} \cdot \mathbf{n} \begin{cases} < 0, \text{ точка } \mathbf{p} \text{ находится над плоскостью;} \\ > 0, \text{ точка } \mathbf{p} \text{ находится под плоскостью;} \\ = 0, \text{ луч параллелен плоскости.} \end{cases} .$$

В случае $\mathbf{u} \cdot \mathbf{n} = 0$ луч параллелен плоскости. Если луч лежит в плоскости, договоримся это не считать пересечением.

α — это расстояние между $\mathbf{q}$ и $\mathbf{p}$ со знаком. Если $\alpha < 0$, луч не пересекает плоскость.

```
// Пересечение луча и плоскости:
// Вход: Луч из точки p в направлении u (|u| = 1).
//       Плоскость с нормалью n и параметром d.
c = u · n;
```

```
if ( c == 0 ) { // пересечения нет
    return ∅;  // (луч параллелен плоскости)
}
α = (d − p · n)/c;
if ( α < 0 ) {
    return ∅; // пересечения нет
}
q = p + αu;
return q;
```

1.11.3 Пересечение луча и треугольника

Многие поверхности можно приблизить совокупностью плоских фрагментов, в частности, треугольников. Поэтому, проверка пересечения луча и треугольника является фундаментальной операцией. Луч задан своим началом $\mathbf{p}$ и направлением $\mathbf{u}$ ($|\mathbf{u}| = 1$). Треугольник можно задать 3 вершинами $\mathbf{v}_0$, $\mathbf{v}_1$ и $\mathbf{v}_2$. Считаем, что вершины не принадлежат одной прямой.

Первый шаг — проверка пересечения луча и плоскости, в которой лежит треугольник. Эта плоскость задаётся нормалью $\mathbf{n}$ и скалярной величиной d:

$$\mathbf{n} \cdot \mathbf{x} = d.$$

Для определённости будем считать, что если смотреть на плоскость «сверху», что вершины упорядочены против часовой стрелки. Найдём $\mathbf{n}$ и d:

$$n = (\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0), \quad d = \mathbf{n} \cdot \mathbf{v}_0.$$

Точку $\mathbf{q}$ можно определить, используя алгоритм пересечения луча и плоскости (см. предыдущий параграф). Последний алгоритм даёт также расстояние α от $\mathbf{p}$ до $\mathbf{q}$. Если $\alpha < 0$, то пересечения нет. Иначе, проверяя знак $\mathbf{u} \cdot \mathbf{n}$ или $d - \mathbf{p} \cdot \mathbf{n}$, можно определить, лежит ли $\mathbf{p}$ «над» или «под» плоскостью.

Когда точка $\mathbf{q}$ найдена, остаётся проверить принадлежность её треугольнику. Множество точек треугольника есть выпуклая оболочка его вершин.

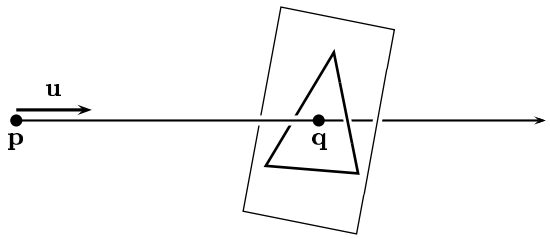


Рис. 11.31: Пересечение луча и треугольника. Первый шаг — это найти точку $\mathbf{q}$ пересечения луча с плоскостью треугольника.

```
// Пересечение луча и треугольника:
// Вход: Луч из точки p в направлении u (|u| = 1).
//       Треугольник с вершинами v0, v1 и v2.
// Вычислим заранее n, d, uβ, uγ:
e1 = v1 − v0;
```

```
e2 = v2 − v0;
n = e1 × e2;
d = n · v0;
a = e1 · e1;
b = e1 · e2;
c = e2 · e2;
D = ac − b^2;
A = a/D;
B = b/D;
C = c/D;
uβ = Ce1 − Be2;
uγ = Ae2 − Be1;


// Основной алгоритм:
q = алгоритм пересечения луча и плоскости(p, u, n, d);
if ( q == ∅ ) {
    return ∅; // пересечения нет
}
r = q − v0;
β = uβ · r;
if ( β < 0 ) {
    return ∅; // пересечения нет
}
γ = uγ · r;
if ( γ < 0 ) {
    return ∅; // пересечения нет
}
α = 1 − β − γ;
If ( α < 0 ) {
    return ∅; // пересечения нет
}
return q;
```

1.11.4 Пересечение луча и выпуклого многогранника

Выпуклый многогранник в $\mathbb{R}^3$ — это часть пространства, ограниченная конечным набором плоскостей[3]. Примеры: куб, призма, пирамида и т.д. В случае $\mathbb{R}^2$ вместо ограничивающих плоскостей присутствуют ограничивающие прямые. Вообще понятие выпуклого многогранника можно обобщить для любой размерности p. При этом гранями являются гиперплоскости. Алгоритм ниже работает для любого p, но для определённости пусть речь идёт о $\mathbb{R}^3$.

Так как мы не рассматриваем невыпуклые многогранники, то слово «выпуклый» в дальнейшем будем опускать.

Пусть многогранник имеет k граней, заданных своими нормалями $\mathbf{n}_i$ и скалярами d_i, $i = \overline{1,k}$. Договоримся, что все нормали направлены наружу. Таким образом, многогранник — это множество точек:

$$\mathbf{x} \cdot \mathbf{n}_i \leqslant d_i, \quad i = \overline{1,k},$$

[3]На практике часто требуется, чтобы многогранник имел конечные размеры. Описываемый ниже алгоритм подходит также для бесконечно больших многогранников.

— пересечение полупространств «под» всеми плоскостями.

Найдём пересечения луча с каждой плоскостью. То есть получим набор $\mathbf{q}_i$, α_i, $i = \overline{1,k}$. Далее выясним по какую сторону от каждой грани лежит точка $\mathbf{p}$. Напомним, что $\mathbf{p}$ лежит «над» i-ой плоскостью, если $\mathbf{u}\cdot\mathbf{n}_i < 0$. Иначе при $\mathbf{u}\cdot\mathbf{n}_i > 0$ точка $\mathbf{p}$ лежит «под» i-ой плоскостью.

Случаю параллельных луча и плоскости договоримся сопоставлять $\alpha = -\infty$. В этом случае всё равно требуется различать пересечения «сверху» и «снизу».

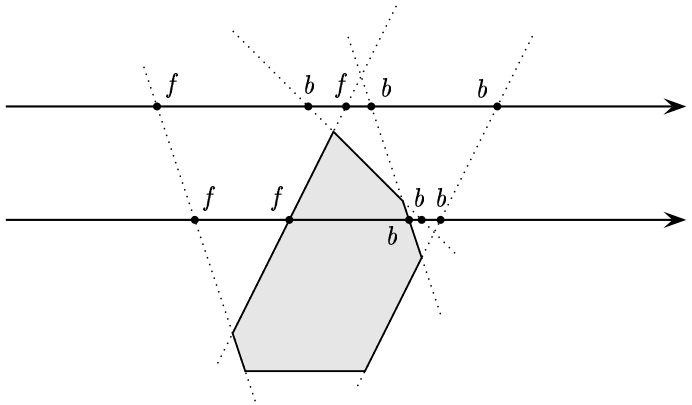


Рис. 11.32: Пересечение двух лучей и ограничивающих плоскостей многогранника. Пересечения «сверху» отмечены буквой f (front), пересечения «снизу» — буквой b (back). Последовательность букв для верхнего луча: $fbfbb$. Так как f встречается после b, то верхний луч не пересекает многогранник. У нижнего луча в последовательности $ffbbb$ все буквы f предшествуют b, поэтому пересечение есть.

Лемма 0.6. *Пусть $\mathbf{q}_i$ и α_i соответствуют i-ой плоскости, как было определено выше. Пусть $\mathbf{p}(\beta) = \mathbf{p} + \beta\mathbf{u}$. Тогда*

- *в случае пересечения «сверху» $\mathbf{p}(\beta)$ лежит полупространстве «ниже» i-ой плоскости тогда и только, когда $\beta \geqslant \alpha_i$;*
- *в случае пересечения «снизу» $\mathbf{p}(\beta)$ лежит полупространстве «выше» i-ой плоскости тогда и только, когда $\beta \leqslant \alpha_i$.*

Доказательство. очевидно. □

В лемме заключается основная идея алгоритма. Вычисляем все значения α_i и $\mathbf{q}_i$ и определяем тип пересечения с i-ой плоскостью (сверху или снизу). Далее положим:

$$\texttt{fMax} = \max\{\alpha_i \,|\, \text{луч пересекает } i\text{-ую плоскость «сверху»}\},$$
$$\texttt{bMin} = \min\{\alpha_i \,|\, \text{луч пересекает } i\text{-ую плоскость «снизу»}\}.$$

Луч пересекает многогранник тогда и только тогда, когда $\texttt{fMax} \leqslant \texttt{bMin}$. Если $\texttt{fMax} \geqslant 0$, то луч достигает поверхности многогранника в точке $\mathbf{p}(\texttt{fMax})$. Если $\texttt{fMax} < 0 \leqslant \texttt{bMin}$, то начало луча $\mathbf{p}$ находится внутри многогранника, и луч пересекает последний изнутри в точке $\mathbf{p}(\texttt{bMin})$. Если $\texttt{bMin} < 0$ или $\texttt{bMin} < \texttt{fMax}$, то пересечения вообще нет. Рисунок 11.32 иллюстрирует работу алгоритма.

Объединим всё выше сказанное в виде алгоритма. Символами $\pm\infty$ обозначены очень большие по абсолютной величине положительные и отрицательные числа.

```
// Пересечение луча и многогранника:
// Вход: Луч из точки p в направлении u (|u| = 1).
//       Набор k плоскостей с нормалями n_i и
//       параметрами d_i, i = 1,k.
fMax = −∞;
bMin = +∞;
for (int i = 0; i<=k; i++) {
    // Пересечение луча и плоскости
    s = u · n_i;
    if (s == 0) { // луч и плоскость параллельны
        if ( p · n_i > d_i ) {
            return ∅; // пересечения нет
        } else {
            continue;
        }
    }
    // Если непараллельно плоскости
    α = (d_i − p · n_i)/s;
    if ( u · n_i < 0 ) { // если пересечение "сверху"
        if ( α > fMax ) {
            if ( α > bMin ) {
                return ∅; // пересечения нет
            }
            fMax = α;
        }
    } else { // если пересечение "снизу"
        if ( α < bMin ) {
            if ( α < 0 or α < fMax ) {
                return ∅; // пересечения нет
            }
            bMin = α;
        }
    }
} // for
α = fMax > 0 ? fMax : bMin;
return q = p + αu;
```

Существует несколько важных случаев, когда работу алгоритма можно существенно ускорить. Речь идёт о следующих многогранниках: кубы, прямоугольные призмы, параллелепипеды и т.д. У этих геометрических тел каждая грань имеет параллельную ей пару. Такие пары можно рассматривать одновременно.

1.11.5 Пересечение луча и цилиндра

Проверка пересечения луча и цилиндра сочетает в себе подходы при проверке пересечений луча со сферой и плоскостью. Рассмотрим для простоты случай прямого кругового цилиндра. Такой цилиндр можно задать радиусом r, направлением оси $\mathbf{v}$ ($|\mathbf{v}| = 1$), центром $\mathbf{c}$ и высотой h.

Точки цилиндра лежат не дальше чем r от оси цилиндра и заключены в простран-

ство между плоскостями, содержащими основания:

$$\{\mathbf{x} : |\mathbf{x} - ((\mathbf{x}-\mathbf{c})\cdot\mathbf{v})\mathbf{v} - \mathbf{c}|^2 \leqslant r^2\} \cap \{\mathbf{x} : ((\mathbf{x}-\mathbf{c})\cdot\mathbf{v})^2 \leqslant (h/2)^2\}.$$

То есть цилиндр есть пересечение множеств точек двух типов: точек бесконечного цилиндра и точек между двумя параллельными плоскостями. Очевидно, цилиндр является выпуклым множеством.

«сверху» и «снизу».

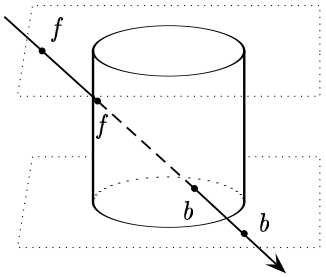


Рис. 11.33: Цилиндр — это пересечение бесконечно высокого цилиндра и пространства между плоскостями, содержащими основания цилиндра. Луч на рисунке входит «сверху» в верхнюю плоскость, затем входит в (бесконечный) цилиндр, выходит из цилиндра и наконец проходит через нижнюю плоскость «снизу». Все пересечения луча «сверху» предваряют пересечения «снизу» (в последовательности $ffbb$ сначала идут только f, затем только b), следовательно, луч пересекает цилиндр.

Во-первых, аналогично параграфу 1 проверим пересечение луча с бесконечно высоким цилиндром. Это даёт нам 0, 1 или 2 точки пересечений.

Упражнение. *Написать алгоритм определения пересечения луча и прямого кругового цилиндра.*

Упражнение. *Написать алгоритм определения пересечения луча и произвольного цилиндра.*

1.12 Занятие 12. Метод трассировки лучей.

1.12.1 Основы трассировки лучей

1.12.2 Щупальца тени

1.12.3 Отражение лучей

1.12.4 Преломление лучей

1.12.5 Локальная освещение и отражение лучей

$$\mathbf{r}_\mathbf{v} = 2(\mathbf{v}\cdot\mathbf{n})\mathbf{n} - \mathbf{v}$$

$$I = I_{\text{local}} + \rho_{\text{rg}} I_{\text{reflect}}$$

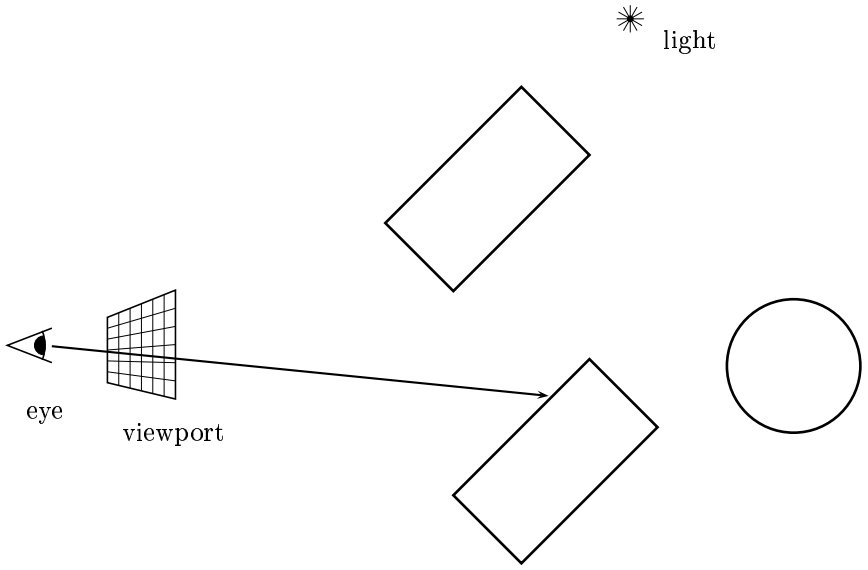


Рис. 12.34: Простейший тип трассировки лучей, состоящий в испускании лучей из положения наблюдателя через каждый пиксель на экране (то есть на плоскости проектирования). Для вычисления освещённости точки, куда падает луч, используется локальная модель.

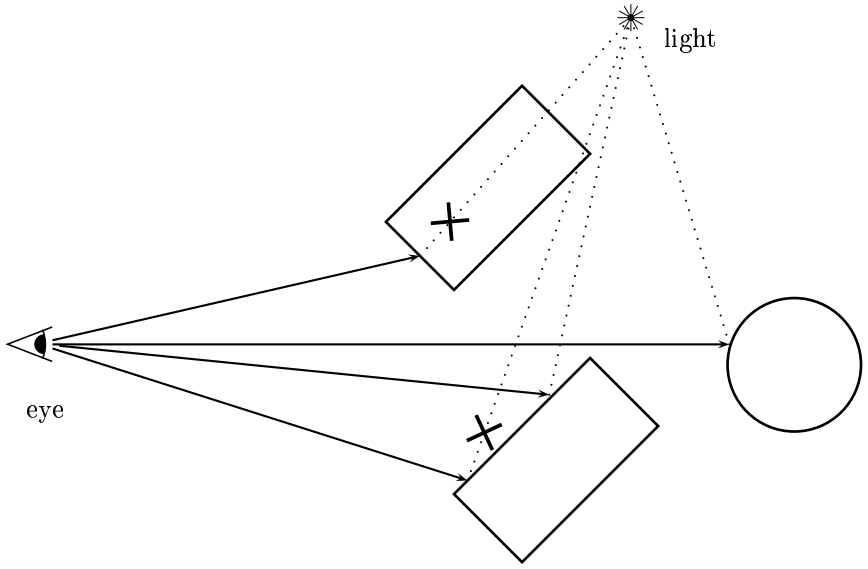


Рис. 12.35: Щупальца тени. Проводится луч от положения наблюдателя до объекта на сцене, куда луч падает. От точки пересечения луча и объекта проводится щупальце тени (отмечена пунктиром) в сторону источника света, чтобы узнать, освещена ли точка напрямую или находится в тени. Два щупальца, отмеченные символом $\times$, напрямую неосвещены.

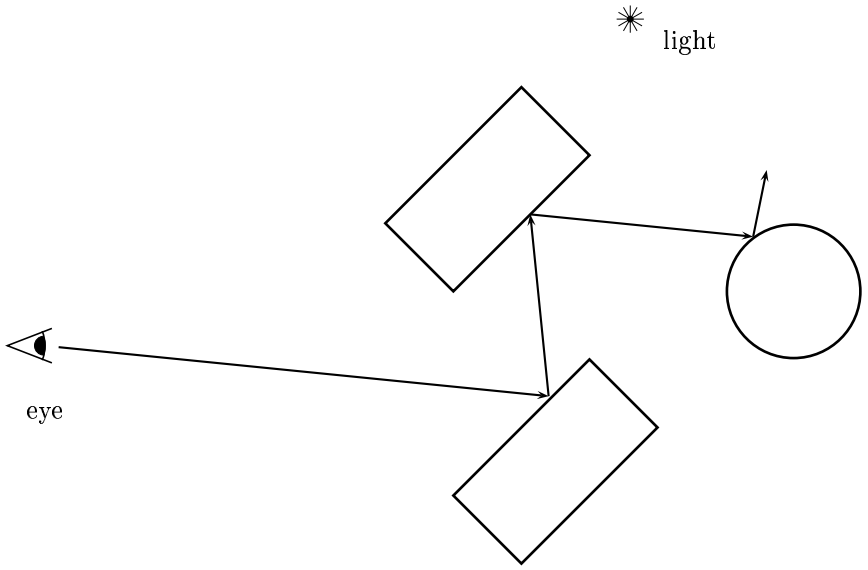


Рис. 12.36: Отражение света. Луч, выпущенный из положения наблюдателя, претерпевает несколько отражений. Эта даёт приближение эффекта множественного отражения в глобальной модели освещённости.

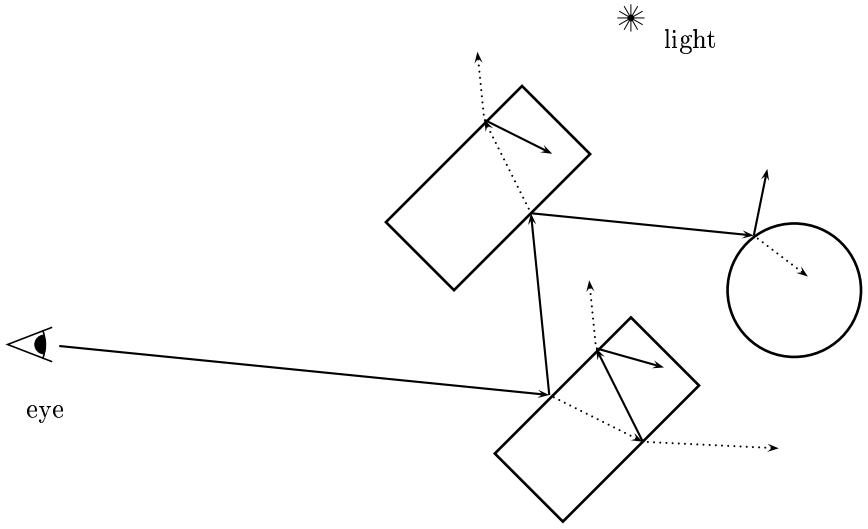


Рис. 12.37: Преломление и отражение света. Луч, выпущенный из положения наблюдателя, претерпевает несколько отражений и преломлений. Отражённые лучи обозначены сплошной линией, а преломлённые —пунктиром. Щупальца тени используются, но на рисунке не отмечены.

$$\mathbf{I}_{\text{local}} = \rho_{\text{a}}\mathbf{I}_{\text{a}}^{\text{in}} + \rho_{\text{d}}\sum_{i=1}^{k}\delta_i\mathbf{I}_{\text{d}}^{\text{in},i}(\ell_i\cdot\mathbf{n}) + \rho_{\text{s}}\sum_{i=1}^{k}\delta_i\mathbf{I}_{\text{s}}^{\text{in},i}(\mathbf{r}_{\text{v}}\cdot\ell_i)^f + \mathbf{I}_{\text{e}}$$

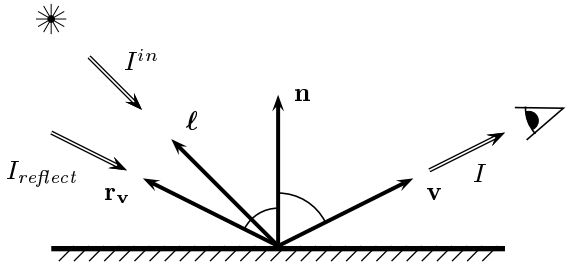


Рис. 12.38: Положение отражённого луча при трассировке лучей. Вектор $\mathbf{v}$ противоположен направлению входящего луча. Направление идеального отражения обозначено вектором $\mathbf{r_v}$. Вектор ℓ направлен на источник света. I — интенсивность исходящего света, видимого в направлении $\mathbf{v}$. I_{reflect} — интенсивность в направлении $\mathbf{r_v}$. I^{in} — интенсивность источника света.

1.12.6 Преломление лучей

Закон Снеллиуса

$$\frac{\sin\theta_v}{\sin\theta_t} = \eta$$

$$\mathbf{v}_{\text{lat}} = \mathbf{v} - (\mathbf{v}\cdot\mathbf{n})\mathbf{n}$$

$$|\mathbf{t}_{\text{lat}}| = \sin\theta_t = \frac{1}{\eta}\sin\theta_v = \frac{1}{\eta}|\mathbf{v}_{\text{lat}}|$$

$$\mathbf{t}_{\text{lat}} = -\frac{1}{\eta}\mathbf{v}_{\text{lat}}$$

$$\cos\theta_t = \sqrt{1-\sin^2\theta_t} = \sqrt{1-|\mathbf{t}_{\text{lat}}|^2}$$

$$\mathbf{t}_{\text{perp}} = -\sqrt{1-|\mathbf{t}_{\text{lat}}|^2}\cdot\mathbf{n}$$

Окончательно имеем

$$\mathbf{t} = \mathbf{t}_{\text{lat}} + \mathbf{t}_{\text{perp}}$$

$$\mathbf{t} = \frac{1}{\eta}((\mathbf{v}\cdot\mathbf{n})\mathbf{n} - \mathbf{v}) - \sqrt{1-\frac{1}{\eta^2}(1-(\mathbf{v}\cdot\mathbf{n})^2)}\cdot\mathbf{n}$$

Обобщим всё сказанное в виде алгоритма:

```
CalcTransmissionDirection( v, n, η ) {
    t_lat = ((v · n)n − v)/η;
    sinSq = |t_lat|²; // sin² θ_t
    if ( sinSq>1 ) {
        return "Преломления нет: полное внут. отраж.";
    }
    t = t_lat − √(1 − sinSq) · n;
    return t;
}
```

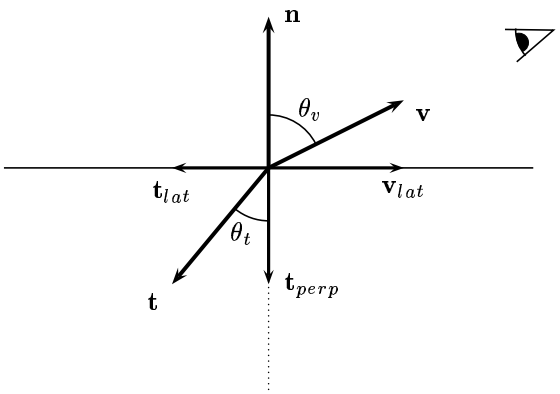


Рис. 12.39: Вычисление направления преломления света $\mathbf{t}$. Горизонтальная линия обозначает поверхность прозрачного материала (гранизу двух сред с различным коэффициентом преломления) с единичным вектором нормали $\mathbf{n}$. Вектор $\mathbf{v}$ направлен противоположно падающему лучу. Направление идеального отражения обозначено вектором $\mathbf{t}$. Вектора $\mathbf{v}_{\text{lat}}$ и $\mathbf{t}_{\text{lat}}$ (от англ. lateral, то есть боковой, поперечный, направленный в сторону) есть составляющие векторов $\mathbf{v}$ и $\mathbf{t}$ вдоль поверхности, а $\mathbf{v}_{\text{perp}}$ и $\mathbf{t}_{\text{perp}}$ (от англ. perpendicular) — соответствующие составляющие в направлении нормали.

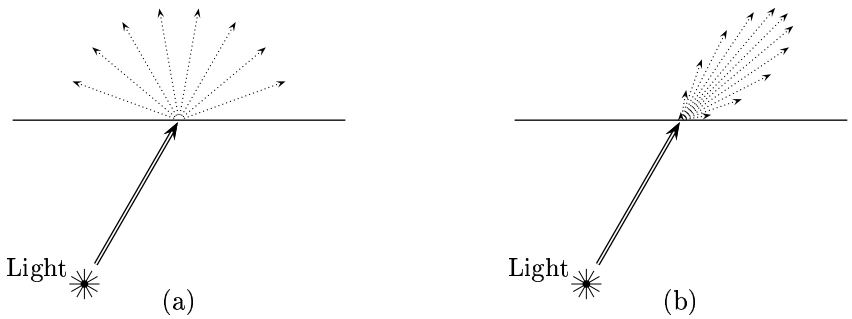


Рис. 12.40: (a) Диффузное преломление света. (b) Зеркальное преломление света. Зеркально преломлённый свет сконцентрирован вокруг направления преломления из закона Снеллиуса.

$\mathbf{t} = \eta^{-1}((\mathbf{v},\mathbf{n})\mathbf{n} - \mathbf{v}) - \sqrt{1 - \eta^{-2}(1 - (\mathbf{v},\mathbf{n})^2)} \cdot \mathbf{n}$

Формула полного локального освещения, включающая в себя отражённый и сквозной свет:

$$\mathbf{I}_{\text{local}} = \rho_{\text{a}}\mathbf{I}_{\text{a}}^0 + \underbrace{\rho_{\text{d}} \sum_{i=1}^{k} \delta_i \mathbf{I}_{\text{d}}^{0,i}(\ell_i, \mathbf{n}) + \rho_{\text{s}} \sum_{i=1}^{k} \delta_i \mathbf{I}_{\text{d}}^{0,i}(\mathbf{v_r}, \ell_i)^f}_{\text{отраженный свет}} +$$

$$+ \underbrace{\rho_{\text{d}} \sum_{i=1}^{k} \delta'_i \mathbf{I}_{\text{d}}^{0,i}(\ell_i, -\mathbf{n}) + \rho_{\text{s}} \sum_{i=1}^{k} \delta'_i \mathbf{I}_{\text{d}}^{0,i}(\mathbf{t}, \ell_i)^f}_{\text{сквозной свет}} + \mathbf{I}_{\text{e}}$$

1.12.7 Общий алгоритм

Объединим результаты предыдущих параграфов в один алгоритм. Основной цикл перебирает все пиксели на экране и проводит луч сквозь каждый пиксель из положения наблюдателя. Параметры этого луча передаются в функцию `RayTrace`, которя делает следующее:

1. Находит точку первого соприкосновения луча с объектами на сцене. Если луч не падает ни на один из объектов, то закрашиваем пиксель в фоновый цвет (составляющая света ambient) и завершаем функцию.
2. Вычисляет освещённость точки соприкосновения, используя локальную модель освещённости.
3. Выпускает отражённый и преломлённый лучи.
4. Дважды вызывает рекурсивно саму себя с параметрами сначала отражённого луча, затем преломлённого.
5. Объединяет освещённость из пункта 2 с освещённостями отражённого и преломлённого лучей.

Чтобы рекурсия могла прекратиться необходим критерий останова. Положим в качестве такового некоторое положительное число `maxDepth`, обозначающее наибольшее возможное число соударений луча с объектами сцены (сколько раз путь луч в нашей модели может претерпевать изломы).

```
// ОСНОВНОЙ АЛГОРИТМ
// x - положение камеры
// maxDepth ∈ ℕ - глубина трассировки
Цикл по всем пикселям на экране {
    u = (p−x)/|p−x|;// единичный вектор в направлении от x до p
    Цвет пикселя p = RayTrace(x, u, maxDepth);
}


// РЕКУРСИВНАЯ ФУНКЦИЯ ТРАССИРОВКИ ЛУЧЕЙ
// s - начальная точка луча
```

```
// u - единичный вектор в направлении луча
// depth - глубина трассировки
// Результат - цвет в виде тройки значений (R,G,B)
(цвет) RayTrace(s, u, depth ) {
    // Часть I: нерекурсивные вычисления
    Проверить пересечение луча из точки s в
    направлении u с поверхностями объектов на сцене.
    Если пересечение есть, пусть z - точка первого
    пересечения луча с одним из объектов, n - нормаль к
    поверхности в точке пересечения.
    if (z == ∅) { // луч не упал ни на один объект
        return "цвет фона"; //т.е. составляющая ambient
    }

    Цикл по всем источникам света {
        Выпустить щупальце тени из точки z к источнику
        света. Проверить, пересекает ли щупальце
        какой-нибудь объект. Вычислить δ_i и δ'_i.
    }

    color = I_local; // См. уравнение (*)

    // Часть II: рекурсивные вычисления
    if ( depth==0 ) {  // Достигнута максимальная
        return color; // глубина трассировки.
    }

    // Вычислить направление отражения и
    // добавить цвет отражения
    if (ρ_rg ≠ 0 ) { // если ненулевое отражение
        r = u − 2(u, n)n; // См. (**), где v = −u.
        color = color +ρ_rg*RayTrace(z, r, depth-1);
    }

    // Вычислить направления сквозного луча
    // (если он имеет место) и прибавить его цвет
    if (ρ_tg ≠ 0) { // Если есть прозрачность.
                    // Пусть η - индекс преломления.
        t=CalcTransmissionDirection(−u, n, η);
        if (t определён) { // если это неслучай полного
                           // внутреннего отражения
            color = color +ρ_tg*RayTrace(z, t, depth-1);
        }
    }
    return color;
}
```

1.13 Занятие 13. Метод излучений.

Метод *излучений* (МИ) (от англ. radiosity) — это глобальная модель освещённости, которая отслеживает распределение рассеянного света на сцене. Глобальность модели состоит в том, что она реализует эффект множественного отражения (учитывает влияние объектов на сцене друг на друга). В отличие от МТЛ, который отслеживает

только зеркальную составляющую света, ИМ работает только с диффузной составляющей.

Задача МИ — это вычислить уровень освещённости каждой поверхности сцены.

Пример. Аудитория с флуоресцентным источником света,крашенными стенами и неблестящий кафель на полу, парты и другая мебель. Предполагается, что нет блестящих поверхностей, и поэтому зеркальная составляющая света невелика. Весь свет на сцене порождается лампами на потолке. Свет диффузно отражается от объектов сцены, особенно от стен и пола, которые становятся вторичными источниками света. Например, часть пола под столом неосвещена напрямую источником света, но она не является полностью тёмной областью. Аналогично, потолок получает очень мало света от закреплённых на нём ламп, но и он достаточно светлый. Даже нижняя сторона крышки стола получает часть отражённого света отражённого от пола.

Пример. Сцена с непрямым освещением, таким как торшер. Торшер — это источник рассеянного света, его лампа направлена в потолок или скрыта абажуром. Света торшера достаточно, чтобы осветить комнату целиком, несмотря на то, что ни один объект в комнате не освещён напрямую лампой.

МИ дополняет МТЛ. Действительно, МТЛ — модель глобального зеркального освещения, а МИ — глобальная модель рассеянного освещения.

МИ лучше подходит для сцен с «мягким» освещением с нечёткими тенями. МТЛ — для сцен с резкими тенями. МИ отслеживается путь света *от источника* света. МТЛ, напротив, отслеживает путь лучей *от наблюдателя*. В связи с последним замечанием несложно понять, что результат работы МТЛ зависит от положения наблюдателя, а МИ никак не связан с наблюдателем. Получается, что можно всего один раз рассчитать освещённость всей сцены с помощью МИ, а затем без существеных вычислений в режиме реального времени отрисовывать сцену для различных положений наблюдателя. Для сравнения МТЛ не позволяет в режиме реального времени перемещать наблюдателя, так как новое положение наблюдателя требует полного пересчёта освещённости сцены.

Перед построением строгого математического алгоритма дадим общее описание метода.

ИМ начинается с разбиения поверхностей объектов сцены на множество многоугольников, называемых элементами (patch). Например, рис. 13.41. Основная цель ИМ — вычислить освещённость или яркость каждого элемента разбиения. Предполагается, что каждый многоугольник освещён равномерно, и отражение только рассеивающее. Таким образом имеет значение только суммарное освещение элементов, а направление этого освещения игнорируется. Некоторые элементы разбиения являются источниками света. Назовём их *излучающими* элементами. Для остальных элементов нужно определить уровни их освещённости. Будем следить за потоками света от излучающих элементов до всех остальных, учитывая возможные отражения света.

После разбиения на элементы определяется, сколько света отражается от каждого элемента в другие по отдельности. Для каждой пары элементов это будет часть отражённого первым элементом света, который достигает второй элемент. Например, пусть первый элемент лежит на стене комнаты, а второй — это плитка пола. Тогда

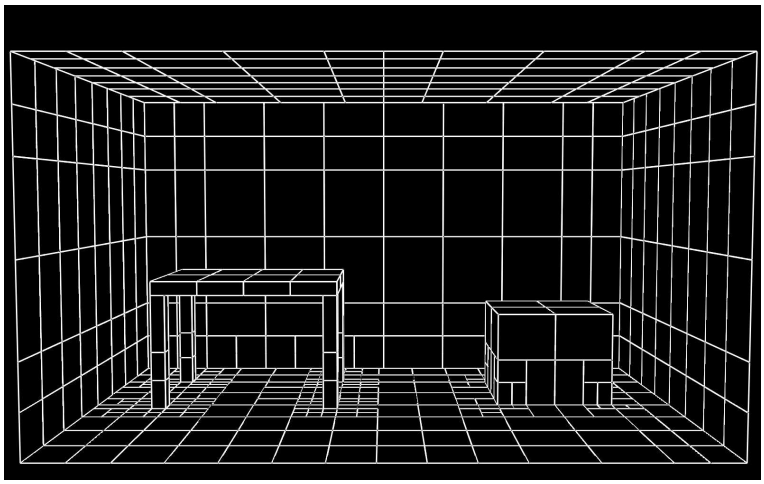


Рис. 13.41: Разбиение поверхностей сцены на области в методе ИМ.

речь идёт о процентах света, отражённого от элемента стены и достигшего плитки пола.

Когда взаимные влияния элементов друг на друга определены, можно записать систему уравнений, где неизвестными являются освещённости элементов разбиения. Получается большая система линейных уравнений, которую нужно решить. На рисунке 13.42 привидён пример сцены, где освещённости элементов разбиения определены с помощью ИМ. На рисунке каждый элемент освещён равномерно, от чего сцена не выглядит достоверно. Заключительным этапом ИМ является усреднение освещённостей для соседних элементов (рис. 13.43).

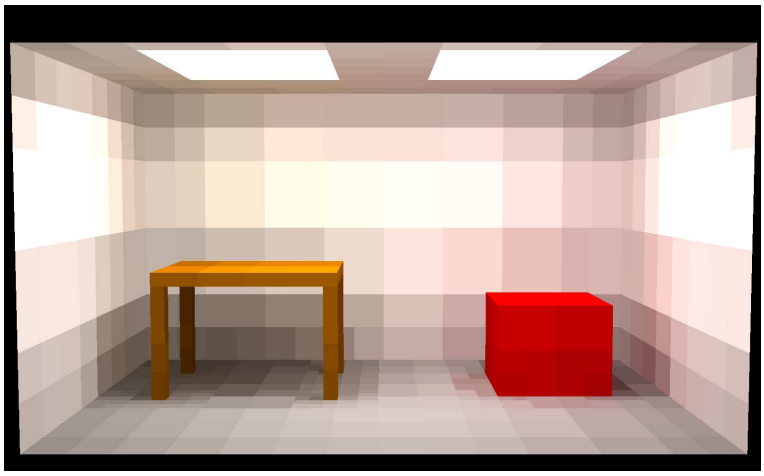


Рис. 13.42: Освещённость сцены вычислена с помощью ИМ без сглаживания цветов. Очевидно, изображение основано на разбиении рисунка 13.41.

В приведённом изложении метода речь шла только о яркости. Но алгоритм также годится для определения яркости освещения элементов светом с фиксированной длиной волны (обычно речь идёт о красном, зелёном и синем свете). Как правило для различных длин волн используют одинаковый коэффициент взаимного отражения элементов разбиения.

Рис. 13.43: Освещённость сцены вычислена с помощью ИМ с последующим сглаживанием. Красный цвет коробки отражается от стены в виде красноватого оттенка. Изображение основано на разбиении рисунка 13.41.

1.14 Элементы разбиения, освещённость, коэффициенты отражения

Первый шаг ИМ — это разбиение поверхностей объектов сцены на плоские многоугольники (элементы). Метод использует предположение о том, что каждый элемент освещён равномерно. В конце работы метода для лучшего восприятия сцены глазом будет усредняющее сглаживание. Важно выбирать элементы достаточно малого размера, чтобы предположение равномерной освещённости элементов не породило много ошибок. Поэтому, уровень освещённости внутри одного элемента, либо соседних элементов не должен меняться очень сильно. Иначе ИМ даст плохие результаты.

Пример разбиения представлен на рис. 13.41. Вся сцена разбита на прямоугольные элементы. Это случай очень простого, что обусловлено прямоугольной формой всех поверхностей на сцене. Однако, сложности возникают даже здесь. Например, уровни освещённости около углов интенсивнее меняются, поэтому разумнее использовать в углах элементы меньшего размера.

В процессе разбиения полезно иметь ввиду следующие соображения. Чем меньше размер элементов разбиения, тем лучше ИМ рассчитывает освещённость сцены. С другой стороны ИМ имеет сложность $O(n^2)$, где n — это количество элементов в разбиении. То есть время вычислений квадратично зависит количества элементов в разбиении. Расходы оперативной памяти также порядка $O(n^2)$. Таким образом, приходится использовать различные методы адаптивного разбиения (разбиения, которое адаптированы к условиям конкретной задачи), где часть элементов большие, остальные — маленькие. На рисунках 13.41–13.43 использованы малые разбиения около углов комнаты, около ящика и под столом. Основной принцип: элементы должны быть меньше там где освещённость меняется быстрее.

Обозначим элементы разбиения буквами $P_1, P_2, \ldots, P_n$, а соответствующие им площади — $A_1, A_2, \ldots, A_n$.

Цель ИМ — определить среднюю яркость каждого элемента P_i. Обозначим через B_i световую энергию, исходящую из P_i, отнесённую к площади P_i. То есть B_i — это световая энергия в единице площади P_i. Значение B_i не совпадает с рассматриваемой ранее интенсивностью света I. Величина I определялась, как количество света, пересекающее единицу площади, перпендикулярную световому потоку. Напротив, B_i есть количество света в единице площади (без учёта ориентации света и площадки, перпендикулярность уже не нужна). Так как мы работаем только с рассеянным светом, и все поверхности Ламбертовы, то свет от P_i в направлении наблюдателя пропорционален B_i и не зависит от ориентации P_i.

Обозначим через B_i^{in} свет, достигающий P_i. Тогда получим уравнение, связывающее падающий и излучаемы свет:

$$B_i = E_i + R_i \cdot B_i^{\text{in}}, \tag{14.16}$$

где E_i — это излучательная способность P_i (от англ. emissiveness), а R_i — коэффициент отражения (от англ. reflectivity). По аналогии с величинами B_i и B_i^{in} излучательность E_i измеряется в единицах световой энергии на единицу площади. Значение E_i обозначает количество света производимого P_i. Если P_i — источник света, то $E_i > 0$, в остальных случаях $E_i = 0$. Коэффициент отражения определяет цвет элемента P_i. Так как мы говорим только о яркости света, то R_i — это скалярная величина, равная части падающего света, которую отражает элемент разбиения. Полагаем, что $0 \leqslant R_i < 1$. Если $R_i = 0$, то элемент свет не отражает, а значения R_i близкие к 1 означают высокую отражающую способность. Близкая к 0 величина R_i соответствует чёрным поверхностям, близкая к 1 — белым поверхностям.

Уравнение (14.16) позволяет вычислить свет, связанный с конкретным элементом. Нам также потребуется выразить влияние исходящего из элемента света на другие элементы. Свет, приходящий в P_j, равен общему излучению всех других элементов P_j, освещающих P_i. Пусть F_{ij} — это часть излучения P_i, которая достигает P_j напрямую (без промежуточного отражения). В случае $i = j$ полагаем $F_{ii} = 0$. Совокупность F_{ij}, $i, j = \overline{1, n}$ называется *формфактором* (form-factors).

Так как освещение измеряется в единицах световой энергии, отнесённых к единице площади, то суммарный свет, исходящий из P_j равен $A_j B_j$. Аналогично свет, падающий на P_i равен $A_i B_i^{\text{in}}$. Получаем

$$A_i B_i^{\text{in}} = \sum_{j=1}^{n} F_{j,i} A_j B_j. \tag{14.17}$$

В дальнейшем мы получим следующее уравнение

$$A_i F_{i,j} = A_j F_{j,i}.$$

С учётом последнего (14.17) можно переписать как

$$B_i^{\text{in}} = \sum_{j=1}^{n} F_{i,j} B_j. \tag{14.18}$$

Объединяя (14.17) и (14.18), получим уравнение *излучения* (УИ):

$$B_i = E_i + R_i \sum_{j=1}^{n} F_{i,j} B_j. \tag{14.19}$$

Последнее можно записать в матричном виде. Обозначим

$$\mathbf{B} = \begin{pmatrix} B_1 \\ \vdots \\ B_n \end{pmatrix}, \quad \mathbf{E} = \begin{pmatrix} E_1 \\ \vdots \\ E_n \end{pmatrix}, \quad M = (R_i F_{i,j})_{i,j} = \begin{pmatrix} R_1F_{11} & R_1F_{12} & \cdots & R_1F_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ R_nF_{n1} & R_1F_{12} & \cdots & R_nF_{nn} \end{pmatrix}.$$

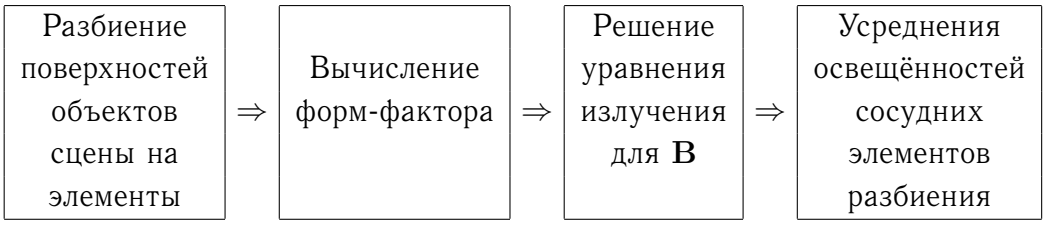
В итоге

$$\mathbf{B} = \mathbf{E} + M\mathbf{B}.$$

Пусть I обозначает единичную матрицу размера $n \times n$. Тогда последнее уравнение можно переписать в виде

$$(I - M)\mathbf{B} = \mathbf{E}.$$

Вектор излучательности $\mathbf{E}$ и коэффициенты R_i считаются известными. Далее требуется определить F_{ij}. После этого остаётся вычислить вектор освещённостей $\mathbf{B}$ с помощью итерационных методов решения СЛАУ (например, методом Якоби или Зейделя). Основные этапы работы МИ представлены на следующей диаграмме:



1.15 Вычисление формфактора

Напомним, что F_{ij} — это доля света, исходящего из P_i и достигающего P_j непосредственно (без отражения от других объектов). Ясно, что $F_{i,j} \in [0, 1]$. Считаем, что у нас замкнутое пространство. То есть свет, исходящий из элемента полностью распределяется среди других элементов. Иными словами:

$$\sum_{j=1}^{n} F_{i,j} = 1.$$

Последнее не ограничивает общности, так как при необходимости некоторые элементы можно считать чёрными и неотражающими свет.

$$F_{i,j} = \frac{(\cos\varphi_i)(\cos\varphi_j)A_j}{\pi d^2}. \tag{15.20}$$

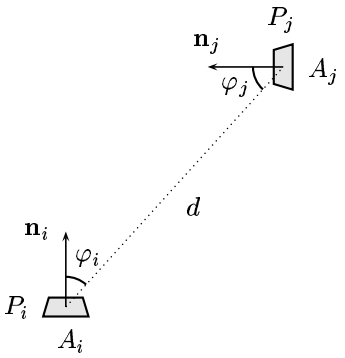


Рис. 15.44: Два бесконечно малых элемента разбиения P_i и P_j, площади которых равны A_i и A_j соответственно. Нормали элементов суть $\mathbf{n}_i$ и $\mathbf{n}_j$.

1.15.1 Метод трассировки лучей для расчёта форм-фактора

Для вычисления форм-фактора можно использовать трассировку лучей. Подсчёт $F_{i,j}$ основан на (15.20), но вводится дополнительно коэффициент *видимости* V_{ij}. Последнее представляет собой долю элемента P_i, которая видна из элемента P_j. Ясно, что $V_{i,j} = V_{j,i}$. Для определения $V_{i,j}$ выпустим лучи из k точек элемента P_i в k точек элемента P_j. Трассировка лучей позволяет обнаружить факт пересечение луча с промежуточными объектами. Полагаем $V_{i,j}$ равным доле лучей, которые не прерывают промежуточные объекты.

На каждом из элементов P_i и P_j выбираются равномерно отдалённые друг от друга k точек. Затем случайным образом устанавливается соответствие между точками P_i и P_j. Другими словами мы формируем пары точек, беря первую из P_i, а вторую из P_j. Через каждую пару точек пропускаем луч и запускаем МТР.

Пусть $\mathbf{c}_i$ и $\mathbf{c}_j$ — это центры P_i и P_j соответственно. Пусть d — это расстояние от $\mathbf{c}_i$ до $\mathbf{c}_j$, а φ_i и φ_j — углы между нормалями и вектором $\mathbf{c}_j - \mathbf{c}_i$. В итоге:

$$F_{i,j} = V_{i,j} \frac{(\cos\varphi_i)(\cos\varphi_j)A_j}{\pi d^2}.$$

1.15.2 Метод полукуба

Метод полукуба (МП), введённый в 1985 Коэном и Гринбергом позволяет вычислить форм-фактор с помощью буфера глубины. Алгоритмы работы с буфером глубины реализованы в большинстве графических подсистем на аппаратном уровне. Таким образом, работу алгоритма можно ускорить аппаратно.

Для расчёта коэффициентов $F_{i,j}$, $j = \overline{1,n}$ для элемента P_i наблюдатель (временно) помещается внутрь P_i. Далее рассчитывается сцена, и получается двумерное цветное изображение (проекция). С помощью этого изображения можно определить, какая часть элемента P_i занята элементом P_j. Если кроме этого учесть расстояние от P_i до P_j, а также взаимную ориентацию P_i и P_j, получим $F_{i,j}$.

Основная идея МП представлена на рисунке 15.20. Здесь виртуальный полукуб помещён над центром бесконечно малого элемента P_i. Полукуб — это верхняя половина куба со стороной 2. Элемент P_j проецируется в направлении элемента P_i на полукуб.

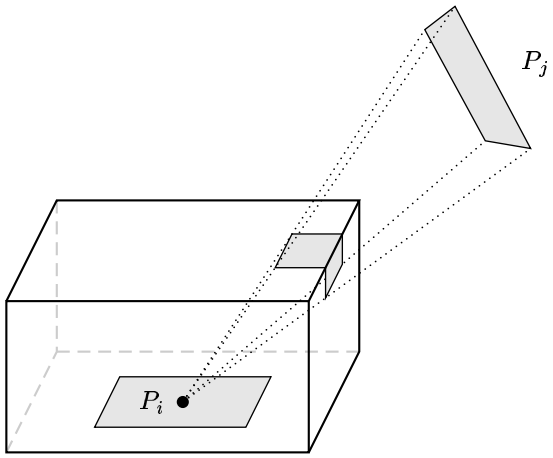


Рис. 15.45: Проекция на полукуб.

Мы ещё не проверяли виден ли элемент P_j, если на него смотреть из положения P_i. Для этого попросим графическую подсистему рассчитать сцену 5 раз, проецируя объекты сцены на верхнюю грань полукуба и его 4 боковые стороны. Упомянутые 5 граней выступают в роле 5 «экранов». На каждом таком «экране» проекция произвольного элемента P_j может занимать 0 или более «пикселей». При этом механизм буфера глубины автоматически упорядочивает объекты сцены по глубине (дальности расположения от P_i) и отсекает невидимые части элемента P_j или даже целиком P_j. Полученное двумерное изображение не выводится на настоящий экран. Вместо этого анализируется, какие пиксели каким элементам P_j соответствуют. Здесь применим следующий поход. Элементы разбиения P_k, $k = \overline{1,n}$ временно окрашиваются не в свои настоящие цвета, а некоторые другие так, чтобы различным элементам P_j соответствовали различные цвета C_j. Далее P_j проектируется на грани куба и предстают в виде набора «пикселей» C_j. Обычно используется $(256)^3$ различных цветов. Этого должно хватить для раскрашивания всех элементов разбиения.

МП получает $F_{i,j}$ как сумму форм-факторов отражения из P_j в пиксели, соответствующие P_j:

$$F_{i,j} = \sum_{\substack{\text{пиксели,}\\ \text{соответст-}\\ \text{вующие } P_j}} (\text{доля излучения } P_j, \text{ достигающая пикселя}). \tag{15.21}$$

На рис. 15.46 показаны пиксели на верхней грани полукуба. Если начало координат поместить в центр $\mathbf{c}_j$ основания полукуба, а само основание лежит в плоскости Oxy, то пиксели цвета C_j имеют координаты $(x, y, 1)$. Расстояние от $\mathbf{c}_j$ до пикселя равно $d = \sqrt{x^2 + y^2 + 1}$. Очевидно $\varphi_i = \varphi_j$ и $\cos\varphi_i = 1/d$. С учётом (15.20) выражение 15.21

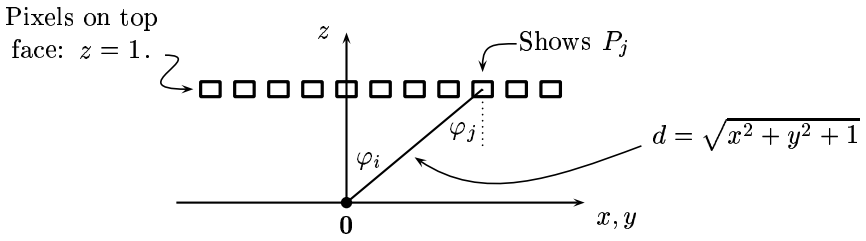


Рис. 15.46: Ряд пикселей на верхней гране полукуба. Один из пикселей соответствует элементу P_j. Начало координат помещено в центр P_i. Верхняя грань лежит в плоскости $z = 1$.

примет вид:

$$\sum_{\substack{\text{пиксели,}\\ \text{соответст-}\\ \text{вующие } P_j}} \frac{(1/d)(1/d)(\text{площадь пикселя})}{\pi d^2} = \sum_{\substack{\text{пиксели,}\\ \text{соответст-}\\ \text{вующие } P_j}} \frac{(\text{площадь пикселя})}{\pi d^4},$$

где $d^4 = (x^2+y^2+1)$. «Площадь пикселя» равна площади, занимаемой одним пикселем на верхней гране полукуба. Так как вершина куба имеет площадь 2×2, площадь пикселя раввна $4/(wh)$, где w и h суть ширина и высота невидимого «экрана».

Пиксели, соответствующие боковым поверхностям полукуба вносят похожий вклад в $F_{i,j}$. В соответствии с рис. 15.47 имеем

$$\sum_{\substack{\text{пиксели,}\\ \text{соответст-}\\ \text{вующие } P_j}} \frac{z(\text{площадь пикселя})}{\pi d^4},$$

где z есть z-компонента координат пикселя.

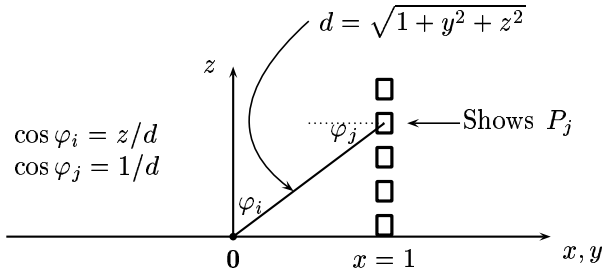


Рис. 15.47: Ряд пикселей той грани полукуба, которая лежит в плоскости $x = 1$.

1.15.3 Пример расчёта формфактора для двумерного случая

Требуется определить F_{ij}, т.е. часть излучения элемента разбиения P_i, которая достигает элемента P_j. Будем считать P_i и P_j отрезками прямых с граничными точками $\mathbf{q}_1$, $\mathbf{q}_2$ и $\mathbf{q}_3$, $\mathbf{q}_4$ соответственно (рис. 15.49). Рассмотрим произвольную точку $\mathbf{q} \in P_i$. Из

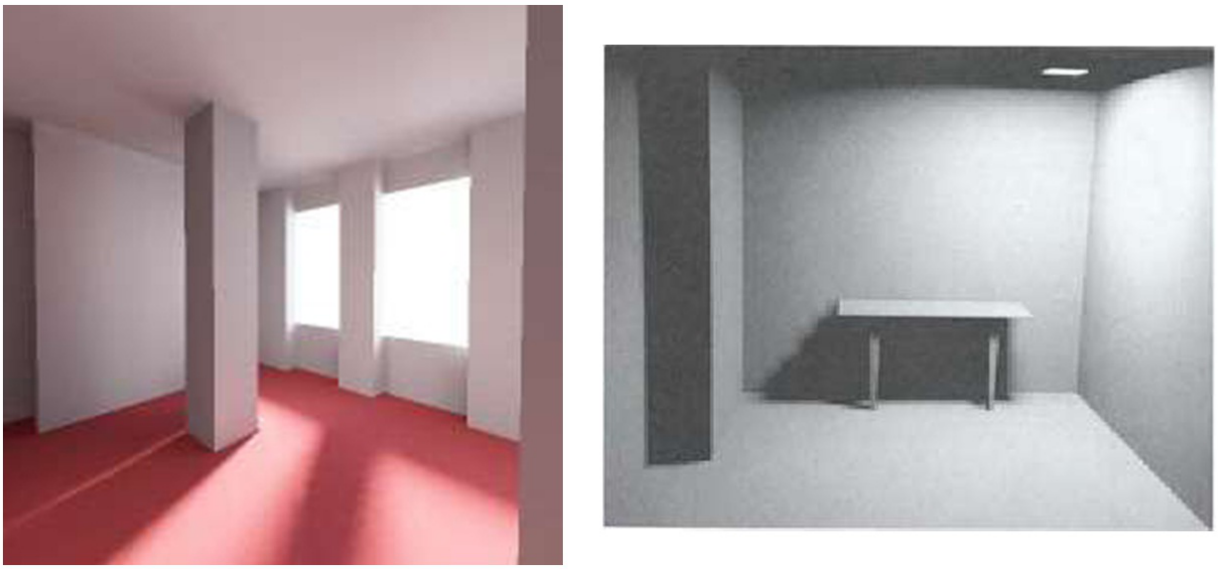


Рис. 15.48: Результат работы МИ. Справа особенно заметна проблема блочных теней.

неё исходит рассеянное излучение равномерно во всех направлениях по одну сторону P_i. Все направления лучей из точки $\mathbf{q}$ образуют угол π. Элемента R_j достигают лучи, попадающие внутрь угла $\gamma = \angle(\mathbf{q}_1\mathbf{q}_2, \mathbf{q}_3\mathbf{q}_4)$. Таким образом, только γ/π лучей из $\mathbf{q}$ достигнут P_j.

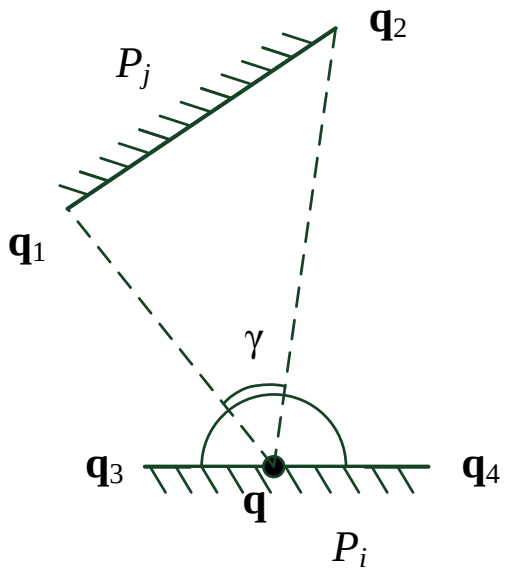


Рис. 15.49: Пример расчёта формфактора для двумерного случая.

Чтобы получить значение F_{ij}, остается просуммировать вклады всех точек элемента разбиения P_i:

$$F_{ij} = \frac{1}{\pi|\mathbf{q}_3\mathbf{q}_4|} \int\limits_{\mathbf{q}_3\mathbf{q}_4} \gamma(\mathbf{q})\,|d\mathbf{q}|. \tag{15.22}$$

Значение угла $\gamma(\mathbf{q})$ легко получить из скалярного произведения векторов $\mathbf{qq}_1$ и $\mathbf{qq}_2$:

$$\gamma(\mathbf{q}) = \arccos\left(\frac{(\mathbf{qq}_1, \mathbf{qq}_2)}{|\mathbf{qq}_1| \cdot |\mathbf{qq}_2|}\right).$$

Если между P_i и P_j находятся какие-то объекты, которые препятствуют части лучей из P_i достигать P_j, то следует внести соответствующие поправки в расчёт F_{ij}. Например на рис. 15.50 лучи из точки $\mathbf{q}$ достигнут P_j только, если их направления попадут либо внутрь угла γ_1, либо внутрь угла γ_2. Доля таких лучей составит $(\gamma_1 + \gamma_2)/\pi$. Выражение (15.22) претерпит соответствующие изменения.

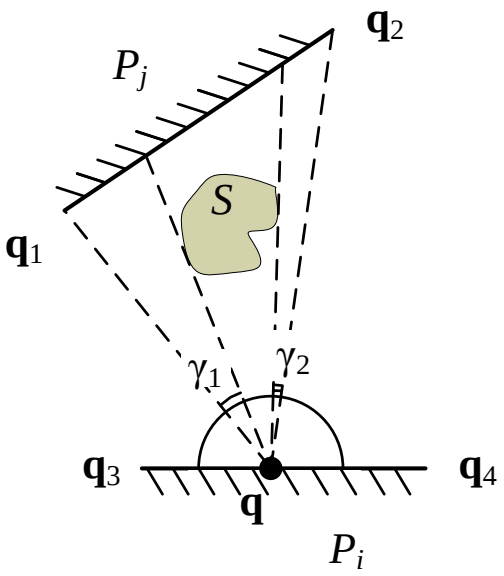


Рис. 15.50: Пример расчёта формфактора для двумерного случая, когда нет прямой видимости между элементами разбиения.

1.16 Занятие 14

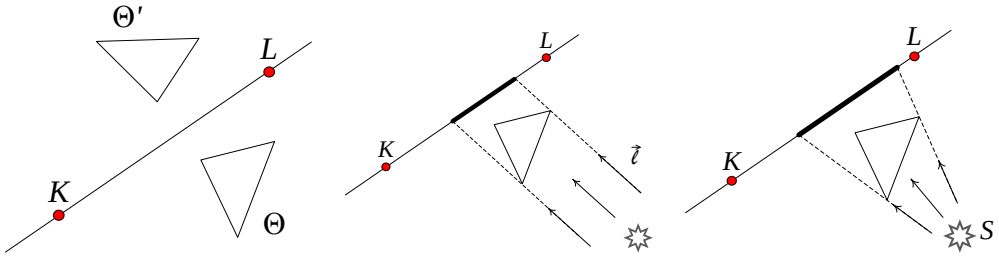
Глава 2

Лабораторные работы

2.1 Лабораторная работа 1

Всё сказанное далее относится к случаю двумерного пространства (плоскости). Программу следует реализовать в MATLAB.

Прямая Λ проходит через две случайно заданные точки K и L. Задан некоторый объект Θ (например, ломаная или многоугольник).



а) б) в)

Рис. 1.1: а) Отражение; б) тень при бесконечно удалённом источнике света; в) тень при источнике света в конечной точке.

③ Построить зеркальное отражение Θ' объекта относительно прямой (рис. 1.1 а). Программа должна вывести на экран прямую, объект и его отражение.

④ Источник света находится в бесконечно удалённой точке, направление лучей задано вектором $\bar{\ell}$. Построить тень, которую бросает объект на прямую (рис. 1.1 б). Программа должна вывести на экран источник света, прямую, объект и его тень.

⑤ Источник света находится в конечной точке S. Построить тень, которую бросает объект на прямую (рис. 1.1 в). Программа должна вывести на экран источник света, прямую, объект и его тень.

Указания. Для наглядности при рисовании следует использовать различные цвета и типы линии. Каждое задание сводится к поиску подходящей матрицы $\mathcal{M}$ и умножении на её на вектор однородных координат точек объекта:

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \mathcal{M} \begin{pmatrix} x \\ y \\ w \end{pmatrix}.$$

Если объект состоит из набора вершин $P_1, \ldots, P_n$, то преобразования координат оптимальнее выполнить одновременно для всех вершин. Т.е. вместо

$$P'_1 = \mathcal{M}P_1, \ldots, P'_n = \mathcal{M}P_n$$

лучше использовать

$$[P'_1, \ldots, P'_n] = \mathcal{M}[P_1, \ldots, P_n],$$

где

$$[P_1, \ldots, P_n] = \begin{pmatrix} P_x^1 & P_x^2 & \cdots & P_x^n \\ P_y^1 & P_y^2 & \cdots & P_y^n \\ P_w^1 & P_w^2 & \cdots & P_w^n \end{pmatrix}.$$

Штрихованная матрица определяется аналогично.

Пример неудачной программы:

```
for i=1:n
    temp = M*[X(i); Y(i); 1];
    X2[i] = temp(1)/temp(3); % x = x / w
    Y2[i] = temp(2)/temp(3); % y = y / w
end
```

Пример хорошей программы:

```
temp = M*[X; Y; ones(1,n)];
X2 = temp(1,:)./temp(3,:); % x = x / w
Y2 = temp(2,:)./temp(3,:); % y = y / w
```

Перед выводом точек на экран нужно перейти от однородных координат к обычным:

$$(x, y, w) \rightarrow (x/w, y/w).$$

2.2 Лабораторная работа 2

Пользователь может отпределить до восьми источников света и задать их свойства такие, как цвет, положение и направление. Для задания этих свойств служит функция

```
void glLight{if}[v](GLenum light, GLenum pname, TYPE param);
```

Эта функция задаёт параметры для источника света `light`, принимающего значения `GL_LIGHT0`, `GL_LIGHT1`, ..., `GL_LIGHT7`. Параметр `pname` определяет характеристику источника света, которая задаётся последним параметром. Возможные значения для `pname` приведены в таблице:

Значение	*Значение по умолчанию*	*Комментарий*
`GL_AMBIENT`	$(0, 0, 0, 1)$	Фоновая RGBA-освещённость
`GL_DIFFUSE`	$(1, 1, 1, 1)$	Рассеянная RGBA-освещённость
`GL_SPECULAR`	$(1, 1, 1, 1)$	Зеркальная RGBA-освещённость
`GL_POSITION`	$(0, 0, 1, 0)$	(x, y, z, w) позиция истоника света
`GL_SPOT_DIRECTION`	$(0, 0, -1)$	(x, y, z) направление для конических источников света
`GL_SPOT_EXPONENT`	0	Показатель степени в формуле Фонга
`GL_SPOT_CUTOFF`	180	Половина угла для конических источников света
`GL_CONSTANT_ATTENUATION`	Отсутствует	Параметр K_c
`GL_LINEAR_ATTENUATION`	Отсутствует	Параметр K_l
`GL_QUADRATIC_ATTENUATION`	Отсутствует	Параметр K_q

Для использования источников света надо разрешить применение расчёта освещённости командой

```
glEnable(GL_LIGHTING);
```

и разрешить (включить) соответствующий источник света при помощи команды `glEnable`, например,

```
glEnable(GL_LIGHT0);
```

Глобальное фоновое освещение можно задать при помощи команды

```
void glLightModel{if}[v](G_LIGHT_MODEL_AMBIENT, ambientColor);
```

Местонахождение наблюдателя оказывается влияние на блики на объектах. По умолчанию при расчётах освещённости считается, что наблюдатель находится в бесконечно удалённой точке, т.е. направление на наблюдателя постоянно для любой вершины. Можно включить более реалистическое освещение, когда направление на наблюдателя будет вычисляться отдельно для каждой вершины; для этого служит команда

```
void glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

Свойства материала, из которого сделан объект, задаются при помощи следующей функции:

```
void glMaterial{if}[v](GLenum face, GLenum pname, TYPE param);
```

Параметр `face` указывает, для какой из сторон грани задаётся свойство, и принимает одно их следующих значений: `GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`. Параметр `pname` указывает, какое именно свойство материала задаётся. Возможные значения приведены в таблице:

Значение	*Значение по умолчанию*	*Комментарий*
`GL_AMBIENT`	$(0.2, 0.2, 0.2, 1.0)$	Фоновый цвет материала
`GL_DIFFUSE`	$(0.8, 0.8, 0.8, 1.0)$	Рассеянный цвет материала
`GL_AMBIENT_AND_DIFFUSE`		Фоновый и рассеянный цвет материала
`GL_SPECULAR`	$(0.0, 0.0, 0.0, 1.0)$	Зеркальный цвет материала
`GL_SHININESS`	0	Коэффициент Фонга для бликов
`GL_EMISSION`	$(0.0, 0.0, 0.0, 1.0)$	Цвет свечения материала

$$\text{color} = E + K_a I_a + \sum S_i \frac{1}{k_c + k_l d + k_q d^2} (K_a I_{ai} + \max\{(\ell, \mathbf{n}), 0\} I_{dt} K_d + (\max\{(\mathbf{h}, \mathbf{n}), 0\})^p I_{si} K_s),$$

где

E — собственная светимость материала (`GL_EMISSION`);

I_a — глобальная фоновая освещённость;

K_a — фоновый цвет материала (`GL_AMBIANT`);

S_i — член, отвечающий за ослабление света в силу того, что источник имеет коническую направленность; он принимает следующие значения: 1, если источник неконический; 0, если источник конический и вершина лежит вне конуса освещённости,

$$(\max\{(\mathbf{v}, \ell), 0\})^e,$$

где $\mathbf{v}$ — единичный вектор от источника света к вершине; ℓ — единичный вектор направления для источника света (`GL_SPOT_DIRECTION`); e — коэффициент `GL_SPOT_EXPONENT`;

k_c — коэффициент `GL_CONSTANT_ATTENUATION`;

k_l — коэффициент `GL_LINEAR_ATTENUATION`;

d — расстояние до источника света;

k_q — коэффициент `GL_QUADRATIC_ATTENUATION`;

I_{ai} — фоновая освещённость от i-го источника света;

ℓ — единичный вектор направления на источник света;

$\mathbf{n}$ — единичный вектор нормали;

I_{di} — рассеянная освещённость от i-го источника света;

K_d — рассеянный цвет материала (GL_DIFFUSE);

p — коэффициент Фонга (GL_SHININESS);

I_{si} — зеркальная освещённость от i-го источника света;

K_s — цвет бликов (GL_SPECULAR).

После проведения всех вычислений цветовые компоненты отсекаются по отрезку $[0, 1]$.

Фрагмент программы для настройки источника света:

```
// фоновое освещённость для источника Light0
GLfloat lightAmb[] = {0.0, 0.0, 0.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);

// рассеянное освещённость для источника Light0
GLfloat lightDif[] = {1.0, 1.0, 1.0, 1.0};
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDif);

// положение источника Light0
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

glEnable(GL_LIGHT0);   // включить источник Light0
glEnable(GL_LIGHTING); // включить освещение
```

Фрагмент программы для настройки материала:

```
float matDif[]     = {0.0f, 0.2f, 0.0f};
float matAmb[]     = {0.2f, 0.2f, 0.2f};
float matSpec[]    = {0.6f, 0.6f, 0.6f};
float matShininess = 0.1f*128;

glMaterialfv(GL_FRONT, GL_AMBIENT,   matAmb);
glMaterialfv(GL_FRONT, GL_DIFFUSE,   matDif);
glMaterialfv(GL_FRONT, GL_SPECULAR,  matSpec);
glMaterialf (GL_FRONT, GL_SHININESS, matShininess);
```

1. Убедиться, что источник света сам по себе не виден наблюдателю.
2. Убедиться, что, если между объектом и источником света находится другой объект, то второй объект не бросает тени на первый и не препятствует прохождению сквозь себя света.

③ Нарисовать два куба, задавая вершины и нормали (рис. 2.2):

```
glBegin(...);
...
glNormal3f(...);
glVertex3f(...);
```

```
...
glEnd();
```

В первом кубе нормали к вершинам должны быть перпендикулярны граням. Во втором кубе нормали к вершинам должны быть направлены от центра куба к его вершинам.

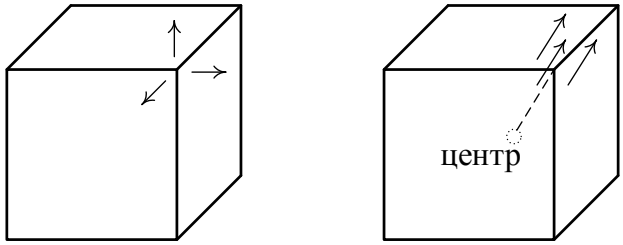


Рис. 2.2: Два куба: а) нормали к вершинам перпендикулярны граням; б) нормали к вершинам направлены от центра куба к его вершинам.

③ Задать свойства материала у граней куба. Задать свойства источника света. Рассмотреть 4 комбинации свойств материала куба и свойств источника света:

	Зеркальный свет	*Рассеянный свет*
Зеркальный материал	Клавиша «1»	Клавиша «2»
Рассеивающий материал	Клавиша «3»	Клавиша «4»

Переключение между комбинациями осуществлять по нажатию на клавиши «1»–«4», например,

```
void key(unsigned char key, int x, int y) {
   if (key == '1') { // реакция на клавишу "1"
      // меняем свойства материала и источника света
      ...
      glutPostRedisplay(); // просим библиотеку перерисовать изображение
   }
   ...
}
```

③ Продемонстрировать и прокомментировать работу режимов
`GL_LIGHT_MODEL_AMBIENT`,
`GL_LIGHT_MODEL_LOCAL_VIEWER`.

④ Продемонстрировать и прокомментировать работу режимов
`GL_EMISSION`,
`glShadeModel`.

⑤ Попытаться вращением сцены (манипулируя мышью) получить блик внутри грани куба. Какой при этом должен быть наблюдатель: локальный или бесконечно удалённый? Возможно ли получить блик внутри грани, заданной с помощью четырёх вершин при интерполяции Гуро? Разбить поверхности кубов, введя сетку (рис. 2.3). В каждой вершине сетки рассчитать нормаль как в случае интерполяции Фонга. При этом нормаль в точке P есть взвешенная сумма нормалей исходных четырех вершин A, B, C и D

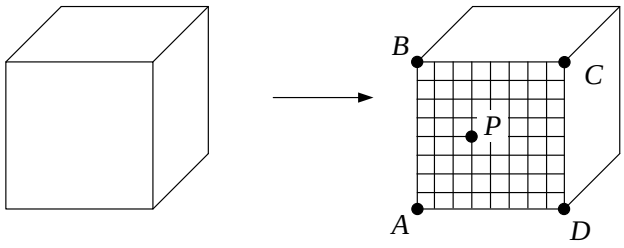


Рис. 2.3: Получение блика внутри грани куба при разбиении грани (сетка) по аналогии с интерполяцией Фонга.

$$\mathbf{n}_P = \frac{a\mathbf{n}_A + b\mathbf{n}_B + c\mathbf{n}_C + d\mathbf{n}_D}{|a\mathbf{n}_A + b\mathbf{n}_B + c\mathbf{n}_C + d\mathbf{n}_D|}.$$

Коэффициенты a, b, c, d — барицентрические координаты точки P внутри системы точек A, B, C и D.

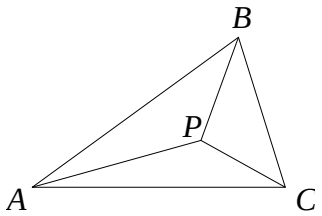


Рис. 2.4: Вычисление барицентрических координат точки P.

В случае треугольника для вычисления барицентрических координат можно использовать следующие формулы:

$$P = aA + bB + cC, \quad a = \frac{S_{BCP}}{S_{ABC}},\ b = \frac{S_{ACP}}{S_{ABC}},\ c = \frac{S_{ABP}}{S_{ABC}}.$$

2.3 Лабораторная работа 3

③ Создать функции, рисующие следующие объекты в начале координат: а) стол (крышка и 3–4 ножки); б) книга (параллелепипед, т.е. растянутый куб); в) ручка (вытянутый цилиндр); г) чайник. Для рисования использовать функции библиотеки GLUT: `glutSolidSphere`, `glutSolidCube`, `glutSolidCone`, `glutSolidTorus`, `glutSolidTeapot`. Нарисовать сцену со столом, на котором лежат книга, ручка и стоит чайник. Для создания сцены использовать полученные функции и встроенные в OpenGL функции для работы с матрицами: `glLoadIdentity`, `glScalef`, `glRotatef`, `glTranslatef`, `glMultMatrixf`, `glLoadMatrixf` и др.

④ Нарисовать аудиторию, состоящую из трёх рядов столов, по 4 стола в каждом ряду. На столах в случайном положении находятся книги, ручки и чайники.

③ Исследовать работу функций проектирования `glOrtho`, `gluPerspective` и `glFrustum`. Продемонстрировать обрезание объектов сцены.

④ Реализовать приближение и удаление камеры при нажатии на произвольные две клавиши. Для этого необходимо изменить параметры функции `gluLookAt`.

⑤ С помощью встроенных матричных функций (`glScalef`, `glRotatef`, `glTranslatef` и др.) нарисовать аудиторию и её зеркальное отражение относительно плоскости стены, где висит доска.

2.4 Лабораторная работа 4

③ Наложить текстуру на плоский объект.

④ Наложить текстуру на неплоский объект (сфера или тор). При наложении текстуры требуется вручную пересчитывать координаты текстуры. При это может потребоваться также вручную нарисовать сам неплоский объект. Использовать `gluNewQuadric` в данном задании не следует.

⑤ Текстура в роли зеркала (эффект отражения окружающих объектов с помощью текстуры).

Если некоторый объект состоит из материала с большой зеркальной составляющей, то для реалистичности этот объект должен отражать окружающие его объекты. Эффект отражения можно достичь, если наложить на объект текстуру, представляющую из себя изображение окружающих объектов.

При таком наложении координаты текстуры не должны жестко привязываться к координатам вершин объектов. Иначе при сдвиге зеркальной поверхности отражение не изменится. В реальном мире отражение зависит от положения зеркала по отношению к наблюдателю. Следовательно, нужен автоматический расчет координат текстуры для каждой вершины. Такой автоматический режим расчета координат текстуры в зависимости от координаты вершины объекта и положения наблюдателя реализован в OpenGL и представляет собой сферическую проекцию.

Чтобы заработало зеркальное текстурирование, нужно кроме включения текстур и их загрузки в память выполнить следующие команды:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

Напомним, что положение пикселя в текстуре определяется двумя координатами: s и t.

При зеркальном текстурировании для определения какой пиксель (s, t) текстуры наложить на заданную вершину используются направление нормали к поверхности объекта, положение наблюдателя и направление, в котором смотрит наблюдатель.

2.5 Лабораторная работа 5

Работа с буфером трафарета (stencil).

④ Постоить тень от объекта, падающую на неплоский объект, например, на лестницу.

③ Постоить зеркальное отражение сцены в прямоугольном зеркале, и ⑤ в зеркале в виде эллипса.

2.6 Лабораторная работа 6

См. практическое занятие №10

④ Изобразить на сцене с помощью двух кривых Безье приближение окружности.

⑤ Изобразить на сцене с помощью двух поверхностей Безье приближение сферы.

Указание: использовать, например, следующие контрольные точки:

$\{2,-2,0\}$	$\{3,-1,0\}$	$\{3,1,0\}$	$\{2,2,0\}$
$\{1,-3,0\}$	$\{2,-3,5\}$	$\{2,3,5\}$	$\{1,3,0\}$
$\{-1,-3,0\}$	$\{-2,-3,5\}$	$\{-2,3,5\}$	$\{-1,3,0\}$
$\{-2,-2,0\}$	$\{-3,-1,0\}$	$\{-3,1,0\}$	$\{-2,2,0\}$

2.7 Лабораторная работа 7

Знакомство с пакетом для трассировки лучей `Ray Trace 3.0`.

③ Развернуть пакет `Ray Trace 3.0` в среде Microsoft Visual Studio:

1. Скачать пакет http://miet.aha.ru/cg/RayTrace_3.3.zip и распаковать архив.
2. Открыть в Microsoft Visual Studio файл `RayTraceKd\RayTraceKd.sln`.
3. Добавить пути к файлам: `glut.h` и `glut32.lib`, скомпилировать и запустить проект.

Управление проектом:

- Поворот сцены — клавиши $\leftarrow$, $\rightarrow$, $\uparrow$, $\downarrow$.
- Клавиша `g` — рассчитать сцену по модели Фонга (обычный для OpenGL режим).
- Клавиша `G` — рассчитать сцену, используя трассировку лучей.

Измерить и сравнить время, затраченное для расчёта сцены по модели Фонга и с помощью трассировки лучей.

④ Попробовать работу программы в трёх режимах:

1. Сцена задается программно. Все объекты сцены, источники света, материалы, текстуры задаются программно с помощью специальных функций.
2. Сцена задается файлом `*.nff`. Все объекты сцены и источники света перечислены в специальном файле `*.nff`.
3. Сцена задается файлом `*.obj` аналогично предыдущему случаю.

Режим задается константой `MODE` внутри функции `InitializeSceneGeometry` из файла `RayTraceKd\RayTraceKd.cpp`. В директории `RayTraceKd` доступны следующие файлы с описанием сцены:

- `balls_2_1.nff`
- `balls_3_1.nff`

- `balls_4_1.nff`
- `balls_5_1.nff`
- `jacks_2_1.nff`
- `jacks_3_1.nff`
- `jacks_4_1.nff`
- `jacks_5_1.nff`
- `f15.obj`.

⑤ В режиме `MODE=1` нарисовать одну сферу, затем две сферы. Для этого модифицировать функцию `SetUpViewableObjects` из файла `RayTraceKd\RayTraceSetup2.cpp`. Сравнить увеличение времени, затраченного на трассировку лучей во втором случае. Объяснить результаты.

2.8 Приложение. Команды GLUT

solid — сплошная фигура, wire — каркасная фигура.

- `glutSolidSphere, glutWireSphere` — сфера;
- `glutSolidCube, glutWireCube` — куб;
- `glutSolidCone, glutWireCone` — конус;
- `glutSolidTorus, glutWireTorus` — тор;
- `glutSolidDodecahedron, glutWireDodecahedron` — додекаэдр;
- `glutSolidOctahedron, glutWireOctahedron` — октаэдр;
- `glutSolidTetrahedron, glutWireTetrahedron` — тетраэдр;
- `glutSolidIcosahedron, glutWireIcosahedron` — икосаэдр;
- `glutSolidTeapot, glutWireTeapot` — чайник.

Литература

[1] *Баяковский Ю. М., Игнатенко А. В.* Начальный курс OpenGL // М.: Планета знаний, 2007.

[2] *Роджерс Д., Адамс Дж.* Математические основы машинной графики // М.: Мир, 2001.

[3] *Buss Samuel R.* 3-D Computer Graphics (A Mathematical Introduction with OpenGL) // Cambridge University Press, 2003.