

ЛАБОРАТОРНАЯ РАБОТА № 3

ТРЕХМЕРНАЯ ВИЗУАЛИЗАЦИЯ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

ЦЕЛЬ РАБОТЫ

Ознакомиться с методом тиражирования сечений (основным способом задания полигональных моделей) и средствами трехмерной визуализации (системы координат, источники света, свойства материалов).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Для получения двумерного изображения трехмерного объекта необходимо осуществить преобразование системы мировых координат в систему оконных координат (рис. 2).

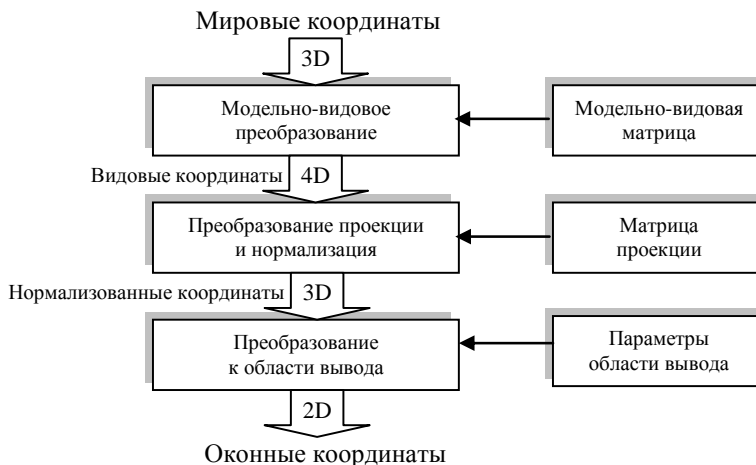


Рис. 2. Конвейер систем координат

Матричные операции

В OpenGL предусмотрены три типа матриц: видовая, проекций и текстуры. Выбор текущей (активной) матрицы задается командой `glMatrixMode(*)`:

- `glMatrixMode(GL_MODELVIEW)` // выбор видовой матрицы
- `glMatrixMode(GL_PROJECTION)` // выбор матрицы проекции
- `glMatrixMode(GL_TEXTURE)` // выбор текстурной матрицы

Если в качестве параметра команды `glMatrixMode(*)` указать не `GL_MODELVIEW`, а `GL_TEXTURE`, тогда указанные преобразования будут применяться не к координатам примитива, а к текстурным координатам.

Замена текущей матрицы на единичную (восстановление исходного состояния) осуществляется командой `glLoadIdentity()`. Для того чтобы сохранить в стеке или восстановить из стека состояние матрицы можно использовать команды `glPushMatrix()` и `glPopMatrix()`, а для умножения – команду `glMultMatrix(*)`.

Получить текущее значение (содержимое) матрицы можно путем вызова команды `glGetFloatv(*)`, указав интересующую матрицу и передав массив размерностью 16. Необходимо иметь в виду, что для быстрого действия OpenGL хранит матрицу не по строкам, а по столбцам.

Метод тиражирования сечений

Трехмерный объект получается тиражированием (последовательным перемещением) заданного двумерного сечения вдоль некоторой траектории и закраской боковых и торцевых поверхностей (граней) получаемого тела. Сечение в общем случае должно быть перпендикулярно траектории тиражирования, пример приведен на рис. 3.

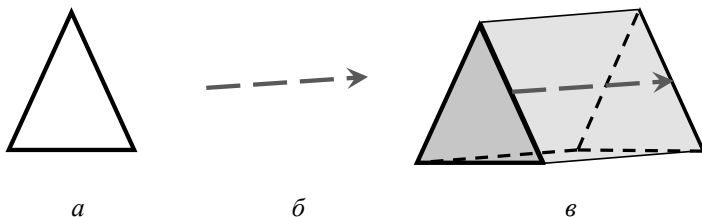


Рис. 3. Тиражирование вдоль отрезка

а – сечение; б – траектория; в – фигура в 3D

В том случае когда траектория представляет собою ломаную, в узлах общее сечение тоже поворачивается на угол, равный половине угла поворота ломаной в данной вершине, как показано на рис. 4.

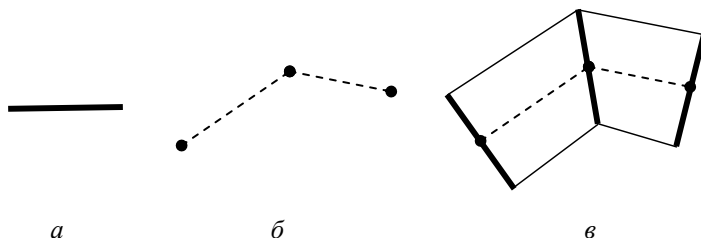


Рис. 4. Тиражирование вдоль ломаной траектории (каркас, вид сверху)

а – сечение; *б* – траектория; *в* – каркас фигуры

Часто использует метод «процентной траектории», когда дополнительные (внутренние) сечения ставятся не в узлах ломаной, а на указанных (в процентах относительно всей длины траектории) расстояниях от начальной точки. В этом случае начальное сечение имеет позицию 0 %, а конечное – 100 %.

Буфер глубины

Буфер глубины (Z-buffer, depth buffer) – двумерный массив данных, дополняющий двумерное изображение (буфер кадра), где каждому пикселю (фрагменту) изображения ставится в соответствии «глубина», т. е. расстояние от наблюдателя до соответствующей точки поверхности отрисовываемого объекта. Если пиксели двух рисуемых объектов перекрываются, то их значения глубины сравниваются и рисуется тот, который ближе, а его значение глубины сохраняется в Z-буфере. Поддержка буфера осуществляется на аппаратном уровне.

Для работы с буфером используются следующие команды:

- `glutDisplayMode (GLUT_DEPTH)` // создание
- `glClear (GL_DEPTH_BUFFER_BIT)` // очистка
- `glEnable (GL_DEPTH_TEST)` // включение
- `glDisable (GL_DEPTH_TEST)` // выключение

Двойная буферизация

Экран имеет два цветовых буфера – передний (front) и задний (back). Из переднего буфера происходит вывод на экран, а в это время

в заднем буфере готовится новое изображение. При поступлении запроса на обмен содержимое заднего буфера и содержимое и переднего буфера *логически* меняются местами. В результате обеспечивается гладкость анимации.

Режим включается командой `glutDisplayMode(GLUT_DOUBLE)`. Команда `glutSwapBuffers()`, которая заменяет собой команду `glFinish()`, используется для переключения буферов.

Расчет нормалей

Нормали в вершинах каждой грани полученного объекта должны быть перпендикулярны самой грани. В любой вершине примитива нормали вычисляются с помощью *векторного произведения* векторов, образованных данной вершиной и двумя смежными с ней (рис. 5).

Соответственно если примитивы имеют общую вершину, то в ней будут указаны несколько разных нормалей (от каждого из примитивов) – это задает *плоский* (несглаженный) режим. В режиме *сглаживания* нормаль в этой общей вершине рассчитывается как векторная сумма «плоских» нормалей.

Нормаль в вершине задается командой `glNormal3f(*)`. Поскольку для корректного расчета освещения необходимо, чтобы вектор нормали имел единичную длину, можно включить режим автоматического нормирования нормалей командой `glEnable(GL_NORMALIZE)`.

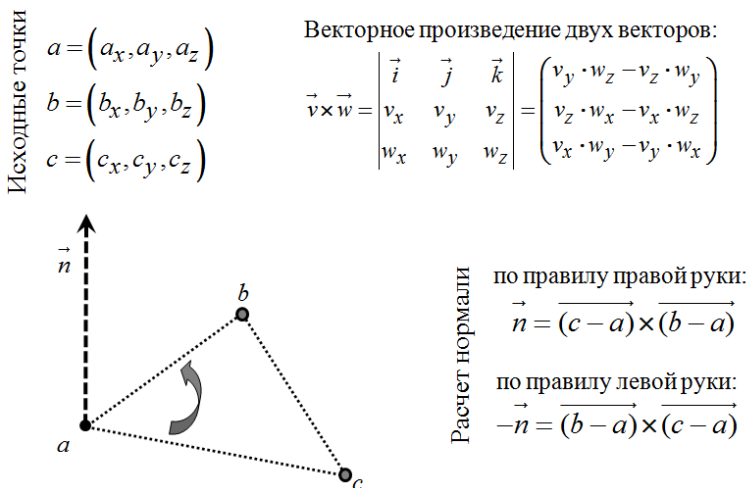


Рис. 5. Расчет нормалей по трем вершинам

Преобразования вида и проекции

Для проведения модельно-видовых и проекционных преобразований координат достаточно умножить на соответствующую матрицу каждую вершину объекта и получить измененные координаты этой вершины.

Вид матриц, соответствующих модельно-видовым преобразованиям (сдвиг, масштабирование и поворот), представлен на рис. 6.

<code>glTranslatef (dx, dy, dz)</code> $\tilde{x} = x + dx$ $\tilde{y} = y + dy$ $\tilde{z} = z + dz$	$M_{trans} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Преобразование сдвига
<code>glScalef (sx, sy, sz)</code> $\tilde{x} = x \cdot sx$ $\tilde{y} = y \cdot sy$ $\tilde{z} = z \cdot sz$	$M_{scale} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Преобразование масштаба
<code>glRotatef (angle, rx, ry, rz)</code> $M_{rotate} = M_{OZ} \cdot M_{OY} \cdot M_{OX}$ $c = \cos(angle)$ $s = \sin(angle)$	$M_{OX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $M_{OY} = \begin{bmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $M_{OZ} = \begin{bmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Преобразование поворота

Рис. 6. Преобразования сдвига, поворота и вращения

Кроме изменения положения самого объекта, часто бывает необходимо изменить положение наблюдателя, что также приводит к изменению модельно-видовой матрицы. Это можно сделать с помощью команды `gluLookAt(*)`, представленной на рис. 7. Строго говоря, эта команда совершает перенос и поворот объектов сцены, но в таком виде задавать параметры бывает удобнее. Следует отметить, что вызывать данную команду имеет смысл перед определением преобразований объектов, когда модельно-видовая матрица равна единичной.

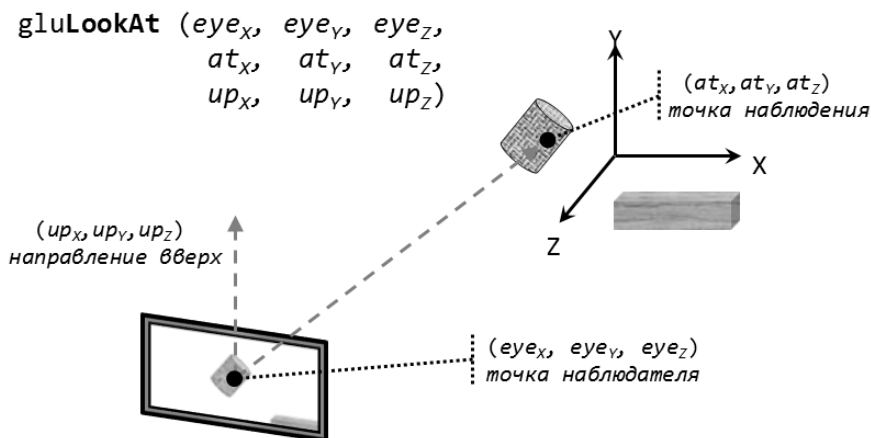


Рис. 7. Смена положения камеры (наблюдателя)

Результирующая матрица преобразований является произведением указанных выше матриц. В общем случае матричные преобразования в OpenGL нужно записывать в обратном порядке. После этих преобразований текущей системы координат уже можно задавать сам объект в исходной форме.

Также в OpenGL существуют команды для задания ортографической (параллельной) и перспективной (центральной) проекций в *левосторонней системе координат*:

- `glOrtho (left, right, bottom, top, near, far)`
- `glFrustum (left, right, bottom, top, znear, zfar)`
- `gluPerspective (angle, aspect, znear, zfar)`

Если OpenGL используется для рисования двумерных объектов, то в этом случае положение вершин можно задавать, используя команды типа `glVertex2f(*)`, а задание ортогографической проекции выполнять командой `gluOrtho2D(left,right,bottom,top)`, т. е. значения `near` и `far` устанавливаются равными `-1` и `1` соответственно.

Данные преобразования основаны на понятии *куб видимости* (он же *объем видимости*, *пирамида видимости*). Пирамида видимости (view frustum) – это часть пространства (сцены), в которой находятся все объекты, видимые из данной точки в данный момент. Она определяется шестью гранями усеченной пирамиды (для перспективного преобразования) или куба (для ортогографической проекции). Соответствующие виды матриц и объема видимости представлены на рис. 8 и 9.

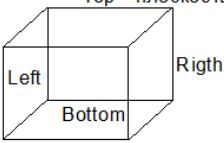
$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{far-near} & t_z \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad \begin{aligned} t_x &= -\frac{right+left}{right-left} \\ t_y &= -\frac{top+bottom}{top-bottom} \\ t_z &= -\frac{far+near}{far-near} \end{aligned}$$


Рис. 8. Ортогографическая проекция

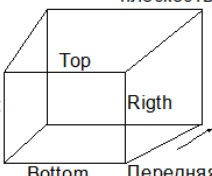
$$\begin{pmatrix} \frac{2 \text{ near}}{right-left} & 0 & A & 0 \\ 0 & \frac{2 \text{ near}}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{pmatrix}, \quad \begin{aligned} A &= \frac{right+left}{right-left} \\ C &= -\frac{far+near}{far-near} \\ B &= \frac{top+bottom}{top-bottom} \\ D &= -\frac{2 \text{ far near}}{top-bottom} \end{aligned}$$


Рис. 9. Перспективная проекция

После применения матрицы проекции получаются усеченные координаты, которые затем нормализуются и преобразуются к оконным.

Область вывода, которая представляет собой прямоугольник пикселей размером `width*height` в оконной системе координат, задается командой `glViewport (x,y, width,height)`.

Освещение и материалы

В OpenGL используется модель освещения, в которой цвет точки определяется следующими факторами:

- свойствами материала и текстуры,
- величиной и направлением нормали в этой точке,
- положением источника света и наблюдателя.

При этом закраска граней осуществляется только *по методу Гуро*, когда цвет в каждом пикселе грани вычисляется при помощи линейной интерполяции цвета по грани. Интенсивность же отраженного света в вершинах граней вычисляется в соответствии с *моделью освещения Фонга*, по которой свет представляет собой простую сумму трех компонент $I = I_A + I_D + I_S$:

- *фоновой* (ambient) $I_A = k_A * L_A$;
- *диффузной* (diffuse) I_D (рис. 10);
- *зеркальной* (specular) I_S (рис. 11).

Эта формула применяется для каждой RGB-компоненты освещения. Параметры модели Фонга для некоторых материалов приведены в табл. 7.

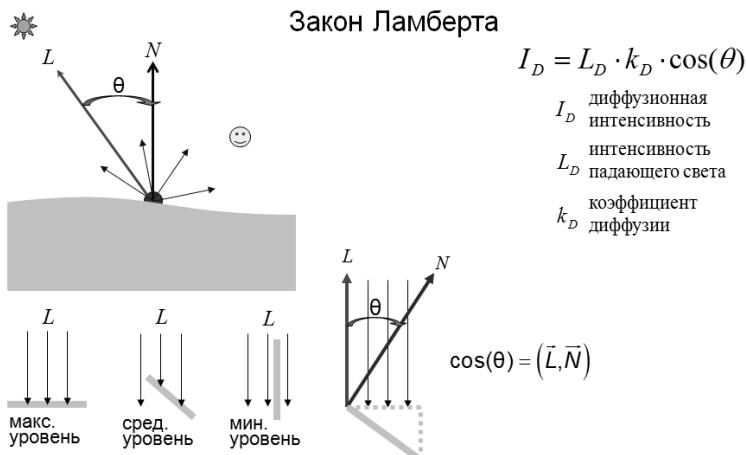
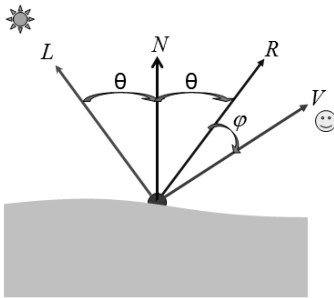


Рис. 10. Диффузная компонента освещения

Закон Фонга



$$I_s = L_s \cdot k_s \cdot \cos^b(\varphi)$$

I_s зеркальная
интенсивность
 L_s интенсивность
падающего света
 k_s коэффициент
зеркальности
 b коэффициент
блеска материала

$$\bar{R} = 2 \cdot (\bar{L}, \bar{N}) \cdot \bar{N} - \bar{L}$$

$$\cos^b(\varphi) = (\bar{R}, \bar{V})$$

Рис. 11. Зеркальная компонента освещения

Таблица 7

Некоторые значения коэффициентов в модели Фонга

Материал	k_A			k_D			k_S			b
	R	G	B	R	G	B	R	G	B	
Латунь	0.32	0.22	0.02	0.78	0.56	0.11	0.99	0.94	0.80	28
Бронза	0.21	0.12	0.05	0.71	0.42	0.18	0.39	0.27	0.16	26
Хром	0.25	0.25	0.25	0.40	0.40	0.40	0.77	0.77	0.77	77
Медь	0.19	0.07	0.02	0.70	0.27	0.08	0.25	0.13	0.08	13
Золото	0.24	0.19	0.07	0.75	0.60	0.22	0.62	0.55	0.36	51
Олово	0.10	0.05	0.11	0.42	0.47	0.54	0.33	0.33	0.52	10
Серебро	0.19	0.19	0.19	0.50	0.50	0.50	0.50	0.50	0.50	51

Поскольку в реальности свет с расстоянием может затухать, светить не во все стороны, то в библиотеке OpenGL предусмотрены различные виды источников света (рис. 12).

Модель освещения задается командой `glLightModelfv(*)`. В частности, с ее помощью можно включить или отключить двусторонний режим расчета освещенности для лицевых и обратных (нелицевых) граней объектов.

Всего в OpenGL доступно 8 источников. Включение источника под номером 2 выполняется командой `glEnable(GL_LIGHT2)`. Выбор типа источника света выполняется командой `glLightfv(*)`.

Материал объекта задается командой `glMaterialfv(*)`. Разрешение учета цвета материала при включенном источнике выполняется командой `glEnable(GL_COLOR_MATERIAL)`.

Тип источника света	Источник излучения	Описание действия	Дополнительные характеристики
Фоновый	Само пространство		Интенсивность
Эмиссионный	Сам объект (не источник)	Подсвечивает только себя	Интенсивность в точке
Точечный	ИСТОЧНИК СВЕТА	Излучает равномерно по всем направлениям	Интенсивность Положение
Прожекторный		Излучает свет направленным пучком	Направление Угол Функция распределения
Удаленный Направленный		Расположен в бесконечности, т. е. все излучаемые лучи параллельны	Интенсивность Вектор направления

Рис. 12. Типы источников в OpenGL

Код примера задания освещения (считается, что нормали уже определены) в нулевом источнике:

```
glEnable ( GL_LIGHTING );
glEnable ( GL_LIGHT0 );
float pos[] = { 0, 0, 1, 1 };
float mat[] = { 0.2, 0.2, 0.2, 1 };
glLightModel ( GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE );
glLightfv ( GL_LIGHT0, GL_POSITION, pos );
glMaterialfv ( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat );
glEnable ( GL_COLOR_MATERIAL );
```

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Считывать из файла (в зависимости от варианта):
 - а) 2D-координаты вершин сечения (считающегося выпуклым);
 - б) 3D-координаты траектории тиражирования;
 - в) параметры изменения сечения.
2. Построить фигуру в 3D по прочитанным данным.
3. Включить режимы:
 - а) буфера глубины;
 - б) двойной буферизации;
 - в) освещения и материалов.
4. Предоставить возможность показа:
 - а) каркаса объекта;
 - б) нормалей (например, отрезками);
 - в) текстур, «обернутых» вокруг фигуры.
5. Предоставить возможность переключения между режимами ортографической и перспективной проекции.
6. Обеспечить навигацию по сцене с помощью модельно-видовых преобразований, сохраняя положение источника света.
7. Предоставить возможность включения / выключения режима сглаживания нормалей.

ВАРИАНТЫ ЗАДАНИЙ

№	Сечение	Изменение сечения
1	Треугольник	Поворот
2	Треугольник	Масштабирование
3	Четырехугольник	Поворот
4	Четырехугольник	Масштабирование
5	Пятиугольник	Поворот
6	Пятиугольник	Масштабирование
7	Шестиугольник	Поворот
8	Шестиугольник	Масштабирование
9	Восьмиугольник	Поворот
10	Восьмиугольник	Масштабирование
11	Девятиугольник	Поворот
12	Девятиугольник	Масштабирование

Дополнительные задания

Для повышения уровня сложности работы и получения дополнительных баллов нужно в своем варианте дополнительно реализовать (на выбор):

- 1) построение фигуры с использованием процентной траектории;
- 2) использование различных источников освещения;
- 3) использование сечения произвольной формы.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Конвейер систем координат.
2. Метод тиражирования сечений.
3. Буферы, используемые в OpenGL.
4. Модели освещения.
5. Алгоритмы удаления невидимых линий.