

№ 4635

**004
К 637**

КОМПЬЮТЕРНАЯ ГРАФИКА С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OpenGL

Методические указания

**НОВОСИБИРСК
2016**

004
К 637

№ 4635

КОМПЬЮТЕРНАЯ ГРАФИКА С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OpenGL

Методические указания к лабораторным работам
для студентов III курса ФПМИ

НОВОСИБИРСК
2016

УДК 004.92(076.5)
К 637

Составители:

канд. техн. наук, доцент *А.Г. Задорожный*

канд. техн. наук, доцент *Д.В. Вагин*

канд. техн. наук, доцент *П.А. Домников*

Рецензент канд. техн. наук, доцент *Ю.В. Тракимус*

Работа подготовлена на кафедре прикладной математики НГТУ

© Новосибирский государственный
технический университет, 2016

ЛАБОРАТОРНАЯ РАБОТА № 1

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ OpenGL

ЦЕЛЬ РАБОТЫ

Ознакомиться с основами использования библиотеки OpenGL и работе с примитивами.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

OpenGL (Open Graphics Library) является одним из самых популярных программных интерфейсов (API) для разработки приложений в области двумерной и трехмерной графики. Работа с библиотекой построена по принципу «клиент-сервер»: приложение вызывает команды, а библиотека занимается их обработкой.

В общем случае OpenGL можно сравнить с конечным автоматом, состояние которого определяется множеством специальных констант и меняется посредством соответствующих команд. Вся эта информация применяется при поступлении в систему координат вершины текущего примитива.

Библиотека OpenGL была разработана как обобщенный аппаратно-независимый интерфейс к графической аппаратуре. По этой причине библиотека не включает в себя функции для работы с окнами и устройствами ввода. Также не включены и средства задания высокоуровневых описаний сложных моделей (чайник, сфера и т. п.) – для этих целей используются такие надстройки над OpenGL, как GLU (OpenGL Utility Library), GLUT (OpenGL Utility Toolkit) и т. д.

Синтаксис команд

Все команды (процедуры и функции) библиотеки начинаются с префикса «gl», а все константы – с префикса «GL_». Для библиотеки

GLU соответствующие префиксы команд и констант обозначаются как «glu» и «GLU_», для библиотеки GLUT – «glut» и «GLUT_», для библиотеки GLAUX – «aux» и «AUX_» и т. д.

Кроме того, в имена команд входят суффиксы, несущие информацию о числе и типе передаваемых параметров (аргументов). В частности, суффикс «-v» указывает на то, что список аргументов представляет собой массив элементов заданного типа и передается одним параметром (указателем).

Разбор структуры имени команды для задания фиолетового цвета `glColor3ub(255,0,255)` приведен в табл. 1. В табл. 2 приводится ряд вводимых библиотекой типов данных, аналогичные им стандартные типы языка C++ и соответствующие суффиксы.

Таблица 1

Команда `glColor3ub(255, 0, 255)`

Часть имени	Описание
<code>gl</code>	Имя библиотеки, где описана функция (OpenGL)
<code>Color</code>	Имя самой функции задания цвета
<code>3</code>	Количество передаваемых аргументов
<code>ub</code>	Тип передаваемых аргументов
<code>255,0,255</code>	Список аргументов (RGB-компоненты)

Таблица 2

Некоторые типы данных

Суффикс	Описание типа		Тип в OpenGL	Тип в C++
<code>b</code>	Целый	1 байт	<code>GLbyte</code>	<i>signed char</i>
<code>ub</code>			<code>GLubyte</code>	<i>unsigned char</i>
<code>i</code>		4 байта	<code>GLint</code>	<i>signed int</i>
<code>ui</code>			<code>GLuint</code>	<i>unsigned int</i>
<code>f</code>	Вещественный	4 байта	<code>GLfloat</code>	<code>float</code>
<code>d</code>		8 байт	<code>GLdouble</code>	<code>double</code>

Задание геометрических примитивов

Для задания геометрических примитивов необходимо указать набор вершин, определяющих данный объект. Для этого служат процедуры `glBegin(*)` и `glEnd()`.

Тип примитива задается параметром процедуры `glBegin(*)` и может принимать значения, представленные в табл. 3.

Таблица 3

Типы примитивов

GL_POINTS	Каждая вершина определяет <i>отдельную точку</i>
GL_LINES	Каждая отдельная пара вершин определяет <i>отрезок</i>
GL_LINE_STRIP	Вершины определяют <i>незамкнутую ломаную</i>
GL_LINE_LOOP	Вершины определяют <i>замкнутую ломаную</i>
GL_TRIANGLES	Каждая отдельная тройка вершин определяет <i>треугольник</i>
GL_TRIANGLE_STRIP	Каждая следующая вершина вместе с двумя предыдущими определяет <i>треугольник</i>
GL_TRIANGLE_FAN	Первая вершина и каждая следующая пара определяет <i>треугольник</i>
GL_QUADS	Каждая отдельная четверка вершин определяет <i>четырёхугольник</i>
GL_QUAD_STRIP	Каждая следующая пара вершин вместе с предыдущей парой определяет <i>четырёхугольник</i>
GL_POLYGON	Вершины определяют выпуклый <i>многоугольник</i>

Пример кода для задания разноцветного двумерного треугольника с использованием различных типов данных:

```
GLubyte blue[3] = {0, 0, 255};
float coord[] = {2, 1};
glColor3f (1.0, 0., 0);
glBegin (GL_TRIANGLES);
    glVertex2f (0.0, 0.0);
    glColor3ub (0, 255, 0);    glVertex2i (1, 2);
    glColor3ubv (blue);       glVertex2fv (coord);
glEnd();
```

Структура консольного приложения

Консольное приложение удобно писать с помощью кросс-платформенной библиотеки GLUT.

Инициализация приложения состоит из двух этапов: начальная инициализация самой библиотеки функцией `glutInit(*)` и инициализация буфера кадра функцией `glutInitDisplayMode(*)`.

Для создания стандартного окна с заголовком используется функция `glutCreateWindow(*)`, а размеры окна устанавливаются функцией `glutInitWindowSize(*)`.

Для создания меню используется команда `glutCreateMenu(*)`, добавление в меню пунктов и подменю выполняется функциями `glutAddMenuEntry(*)` и `glutAddSubMenu(*)` соответственно. Назначение меню на правую клавишу мыши осуществляется командой `glutAttachMenu(GLUT_RIGHT_BUTTON)`.

Регистрация ряда пользовательских callback-функций (функций обратного вызова) для реакции на нажатие клавиш (Keyboard, Mouse), изменения размера окна (Reshape), необходимости обновления окна (Display) осуществляется следующими командами:

- `glutKeyboardFunc(Keyboard)`
- `glutMouseFunc(Mouse)`
- `glutDisplayFunc(Display)`
- `glutReshapeFunc(Reshape)`

Контроль всех событий и автоматический вызов соответствующих функций выполняется командой `glutMainLoop()`.

В том случае когда необходим принудительный (досрочный) вызов функции обновления окна, рекомендуется использовать команду `glutPostRedisplay()`.

Пример приложения с использованием библиотек GLUT и STL

В данном примере рисуется множество точек, задаваемых левым кликом мыши, а по правому клику происходит удаление последней точки. Сдвиг всех точек осуществляется клавишами “w”, “s”, “a”, “d”, а смена цвета – “r”, “g”, “b”. Для хранения множества точек используется контейнер `vector` из библиотеки STL (Standard Template Library).

```
#include "glut.h"
#include <vector>
using namespace std;
```

```

GLint Width = 512, Height = 512;
GLubyte ColorR = 255, ColorG = 255, ColorB = 255;

struct type_point
{
    GLint x, y;
    type_point(GLint _x, GLint _y) { x = _x; y = _y; }
};
vector <type_point> Points;

/* Функция вывода на экран */
void Display(void)
{
    glClearColor(0.5, 0.5, 0.5, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(ColorR, ColorG, ColorB);
    glPointSize(6); glEnable(GL_POINT_SMOOTH);
    glBegin(GL_POINTS);
        for (int i = 0; i<Points.size(); i++)
            glVertex2i(Points[i].x, Points[i].y);
    glEnd();
    glFinish();
}

/* Функция изменения размеров окна */
void Reshape(GLint w, GLint h)
{
    Width = w;    Height = h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, w, 0, h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

```



```

/* Функция обработки сообщений от клавиатуры */
void Keyboard(unsigned char key, int x, int y)
{
    int i, n = Points.size();

    /* Изменение RGB-компонент цвета точек */
    if (key == 'r') ColorR += 5;
    if (key == 'g') ColorG += 5;
    if (key == 'b') ColorB += 5;

    /* Изменение XY-ординат точек */
    if (key == 'w') for (i = 0; i<n; i++) Points[i].y += 9;
    if (key == 's') for (i = 0; i<n; i++) Points[i].y -= 9;
    if (key == 'a') for (i = 0; i<n; i++) Points[i].x -= 9;
    if (key == 'd') for (i = 0; i<n; i++) Points[i].x += 9;

    glutPostRedisplay();
}

/* Функция обработки сообщения от мыши */
void Mouse(int button, int state, int x, int y)
{
    /* клавиша была нажата, но не отпущена */
    if (state!= GLUT_DOWN) return;

    /* новая точка по левому клику */
    if (button == GLUT_LEFT_BUTTON)
    {
        type_point p(x, Height - y);
        if(Points.Size()) Points.push_back(p);
    }

    /* удаление последней точки по правому клику */
    if (button == GLUT_RIGHT_BUTTON) Points.pop_back();

    glutPostRedisplay();
}

```

```

/* Головная программа */
void main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowSize(Width, Height);
    glutCreateWindow("Рисую точки");
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);
    glutMouseFunc(Mouse);
    glutMainLoop();
}

```

В функции `Display()` подготовка буфера кадра (экрана) начинается с того, что командами `glClearColor(*)` и `glClear(*)` очищается весь буфер, после чего задаются примитивы, а завершение процесса (отображение буфера на экране) выполняется командой `glFinish()`.

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Отобразить в окне множество примитивов (вершины которых задаются кликами мыши) в соответствии с вариантом задания.

2. Для завершения текущего (активного) набора (множества) примитивов и начала нового зарезервировать специальную клавишу (пробел или правый клик).

3. Для текущего набора примитивов предоставить возможность изменения цвета и координат его вершин.

4. Текущее множество примитивов выделять среди других, например, изменением размера его вершин командой `glPointSize(*)`.

5. Использовать контейнер `vector` из библиотеки STL для хранения набора примитивов и множества вершин каждого примитива, а для хранения атрибутов рекомендуется использовать стандартный класс `struct`.

6. Предусмотреть возможность удаления последнего примитива и последнего набора примитивов.

7. Продублировать команды в меню, созданном с помощью библиотеки GLUT.

ВАРИАНТЫ ЗАДАНИЙ

№	Тип примитивов
1	GL_POINTS
2	GL_LINES
3	GL_LINE_STRIP
4	GL_LINE_LOOP
5	GL_TRIANGLES
6	GL_TRIANGLE_STRIP
7	GL_TRIANGLE_FAN
8	GL_QUADS
9	GL_QUAD_STRIP
10	GL_POLYGON

Дополнительные задания.

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

- 1) изменение не только координат и цвета вершин примитивов, но и режимов сглаживания, шаблона закрашивания примитива, ...;
- 2) изменение параметров (в том числе и удаление) не только текущего набора примитивов, но и произвольного.
- 3) изменение параметров произвольного примитива в наборе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Архитектура и основные функции библиотек OpenGL.
2. Организация конвейера OpenGL.
3. Функции обратного вызова.
4. Примитивы и атрибуты.
5. Структуры данных.

ЛАБОРАТОРНАЯ РАБОТА № 2

РАСТЕРИЗАЦИЯ ПРИМИТИВОВ

ЦЕЛЬ РАБОТЫ

Ознакомиться с принципами растеризации примитивов и наложения цветов и текстур.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Растровые изображения – это набор *пикселей*, которые представляют собой наименьшую единицу растрового изображения. Векторные же изображения задаются функциями (отрезок, окружность и т. д.) с такими параметрами, как например, толщина и цвет контура. Соответственно для вывода векторного изображения на растровый экран оно должно быть сначала *растеризовано*.

Растеризация отрезка

Алгоритм Брезенхема (Bresenham's line algorithm) – это основной алгоритм *растеризации отрезка* (алгоритм, определяющий, какие точки двумерного раstra нужно закрасить, чтобы получить приближение заданного отрезка). Принцип работы алгоритма для отрезка с углом наклона менее 45° заключается в следующем: для каждого столбца пикселей найти и закрасить пиксель, ближайший по вертикали к соответствующей точке отрезка, т. е. критерием ошибки является расстояние между у-координатой исходного отрезка и у-координатой пикселя (оно не должно превышать половину высоты пикселя). Иллюстрация результата работы алгоритма представлена на рис. 1.

Для других вариантов расположения отрезка алгоритм легко модифицируется соответствующим образом.

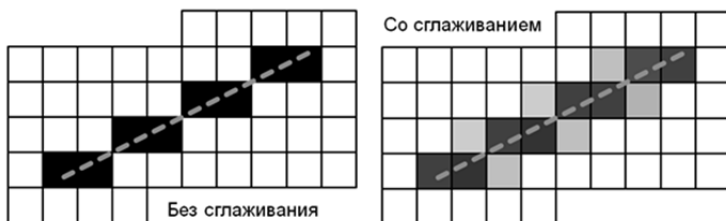


Рис. 1. Пример работы алгоритма Брезенхема

Заполнение многоугольника

Для того чтобы заполнить (закрасить) многоугольник, необходимо найти все пиксели из внутренней части многоугольника. Определить же принадлежность точки многоугольнику можно различными алгоритмами, простейшим (но не самым эффективным) из которых является метод углов: сумма всех углов, образованных рассматриваемой точкой и парой соседних вершин многоугольника, равна 360° только в том случае, когда эта точка находится внутри.

Угол между векторами \vec{a} и \vec{b} можно рассчитать по формуле скалярного произведения:

$$\arccos \left(\frac{(\vec{a}, \vec{b})}{\|\vec{a}\| \cdot \|\vec{b}\|} \right).$$

Логические операции над цветами

Включение и выключение логических операций над цветами выполняются командами `glEnable(GL_COLOR_LOGIC_OP)` и `glDisable(GL_COLOR_LOGIC_OP)` соответственно.

Выбор конкретной логической операции задается параметром функции `glLogicOp(*)`.

Таблица 4

Типы логических операций с цветом

Логическая операция		Действие над цветами s и d
GL_AND	AND	$s \& d$
GL_NAND	NOT AND	$! (s \& d)$
GL_OR	OR	$s d$
GL_NOR	NOT OR	$! (s d)$
GL_XOR	XOR	$s \wedge d$
GL_EQUIV	NOT XOR	$! (s \wedge d)$

Текстурирование

Текстура – это графическое изображение (обычно в виде двумерной картинке в формате bmp), используемое для нанесения изображения на объект, составленный из множества примитивов. Целью применения текстур является повышение реалистичности изображения за счет экономии вычислительных ресурсов.

Пиксели текстуры называют *текселями*. Вообще говоря, тексели не обязательно являются пикселями изображения: например, тексели можно рассматривать как карту высот при Bump Mapping.

В общем случае для наложения текстур на примитив необходимо выполнить этапы, приведенные из табл. 5.

Таблица 5

Этапы наложения текстур

Описание этапа			Команда
Создание массива текстур	1	Чтение текстур из файлов	LoadImage(*), fread(*), ...
	2	Генерация массива номеров текстур	glGenTextures(*)
Установка параметров учета выбранной текстуры	3	Выбор текущей текстуры по номеру	glBindTexture(*)
	4	Задание режима выравнивания	gluBuild2DMipmaps(*)
	5	Задание параметров растяжения и дублирования	glTexParameterf(*)
	6	Задание режима учета параметров материала	glTexEnvf(*)
Наложение выбранной текстуры	7	«Привязывание» текстуры к примитиву	glTexCoord2f(*)
	8	Разрешение отображения текстур	glEnable(GL_TEXTURE_2D)

В OpenGL нет функций для загрузки и конвертации текстур. Если текстура представляет собой простое изображение в формате bmp 24 bits, тогда для загрузки текстуры можно использовать средства C++

или встроенную в Windows функцию `LoadImage(*)`. В противном случае необходимо подключить дополнительную библиотеку работы с графическими изображениями типа `GLAUX`, `FreeImage` и т. п.

Пример кода для задания (с настройками по умолчанию) квадрата с «простой» текстурой, прочитанной из файла «texture.bmp»:

```
/* Чтение текстуры из файла */
GLubyte data54 [54];
FILE *file = fopen("texture.bmp","rb");
fread(data54, 54, 1, file);
int width = *(data54 + 18);
int height = *(data54 + 22);
GLubyte *pixels = new GLubyte [width * height * 3];
fread(pixels, width * height * 3, 1, file);
fclose(file);

/* Установка параметров текстуры */
GLuint tex;
glGenTextures (1, &tex);
glBindTexture (GL_TEXTURE_2D, tex);
gluBuild2DMipmaps (GL_TEXTURE_2D, 3, width, height,
                  GL_BGR_EXT, GL_UNSIGNED_BYTE, pixels);

/* Наложение текстуры на квадрат */
glEnable (GL_TEXTURE_2D);
glBegin (GL_QUADS);
    glTexCoord2f(0,0); glVertex2i(10,10);
    glTexCoord2f(0,1); glVertex2i(10,90);
    glTexCoord2f(1,1); glVertex2i(90,90);
    glTexCoord2f(1,0); glVertex2i(90,10);
glEnd();
```

Модельно-видовые преобразования

К модельно-видовым преобразованиям относят перенос, поворот и изменение масштаба (табл. 6).

Модельно-видовые преобразования

Команда OpenGL	Описание
<code>glTranslatef(dx,dy,dz)</code>	Перенос вдоль вектора (dx, dy, dz)
<code>glRotatef(angle, ox,oy,oz)</code>	Поворот на угол angle вокруг осей Ox , Oy и Oz
<code>glScalef(sx,sy,sz)</code>	Масштабирование вдоль осей в sx,sy и sz раз соответственно

Эти преобразования изменяют текущую систему координат, которая определяется модельно-видовой матрицей, устанавливаемой командой `glMatrixMode(GL_MODELVIEW)`. Отменить все изменения, т. е. восстановить исходное (единичное) состояние текущей матрицы, можно командой `glLoadIdentity()`.

В том случае когда требуется применить модельно-видовые преобразования не для всех объектов, а только для некоторых, необходимо перед их заданием:

- 1) сохранить текущее состояние командой `glPushMatrix()`;
- 2) применить нужное модельно-видовое преобразование;
- 3) задать эти объекты в исходной форме;
- 4) вернуть исходное состояние командой `glPopMatrix()`.

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. В соответствии с заданием отобразить объекты (правильные многоугольники), центр и «радиус» которых определяются кликами мыши.

2. Предоставить возможность изменять форму и положение объектов модельно-видовыми преобразованиями.

3. Предоставить возможность изменять цвет и текстуру текущего (активного) объекта.

4. Предоставить возможность включать и выключать логические операции смешивания цветов (в соответствии с вариантом задания).

5. Вывести на экране динамически меняемую крупную сетку условных пикселей и на ней растеризовать данные объекты в режимах «контур» и «с заполнением».

6. При растеризации пересекающихся объектов цвет области пересечения определять в соответствии с заданной операцией смешения цветов.

7. Продублировать команды в меню, созданном с помощью библиотеки GLUT.

ВАРИАНТЫ ЗАДАНИЙ

№	Объекты	Логические операции		
1	Правильные четырехугольники	AND	+	NOT AND
2	Правильные пятиугольники	AND	+	NOT AND
3	Правильные шестиугольники	AND	+	NOT AND
4	Правильные восьмиугольники	AND	+	NOT AND
5	Правильные четырехугольники	OR	+	NOT OR
6	Правильные пятиугольники	OR	+	NOT OR
7	Правильные шестиугольники	OR	+	NOT OR
8	Правильные восьмиугольники	OR	+	NOT OR
9	Правильные четырехугольники	XOR	+	NOT XOR
10	Правильные пятиугольники	XOR	+	NOT XOR
11	Правильные шестиугольники	XOR	+	NOT XOR
12	Правильные восьмиугольники	XOR	+	NOT XOR

Дополнительные задания

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

- 1) алгоритм сглаживания;
- 2) растеризацию дополнительных объектов – круга и окружности;
- 3) растеризацию объектов, чья геометрия была изменена модельно-видовыми преобразованиями.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Векторные и растровые изображения.
2. Алгоритмы растеризации отрезка.
3. Алгоритмы растеризации окружности.
4. Алгоритмы сглаживания.
5. Текстурирование.
6. Модельно-видовые преобразования.
7. Логические операции над цветами.

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).

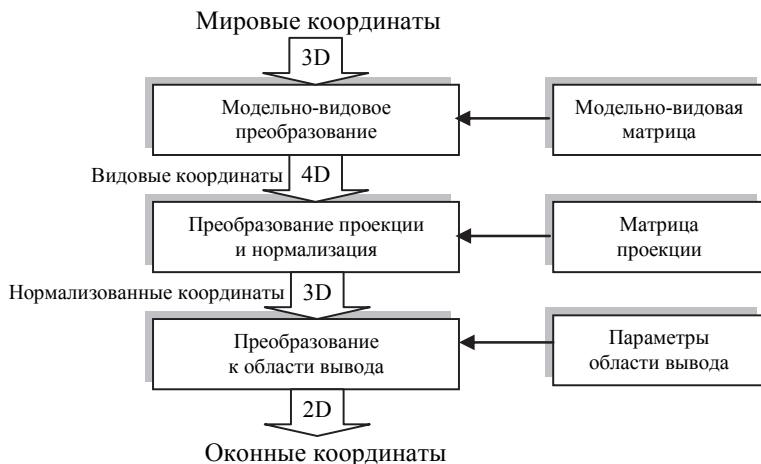


Рис. 2. Конвейер систем координат

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.

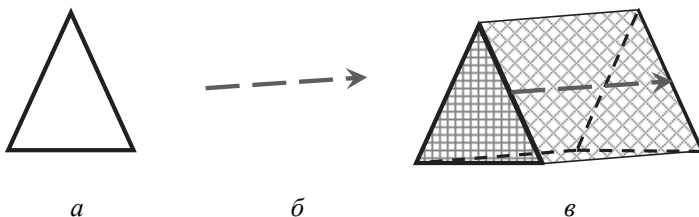


Рис. 3. Тиражирование вдоль отрезка
а – сечение; б – траектория; в – фигура в 3D

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.

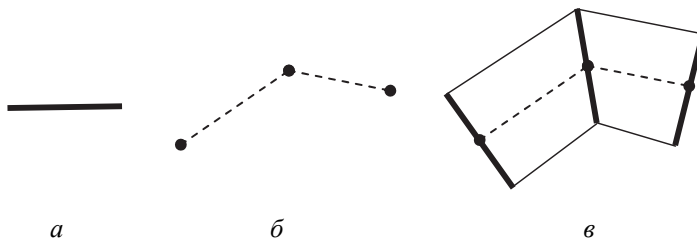


Рис. 4. Тиражирование вдоль ломаной траектории (каркас, вид сверху)

a – 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)`.

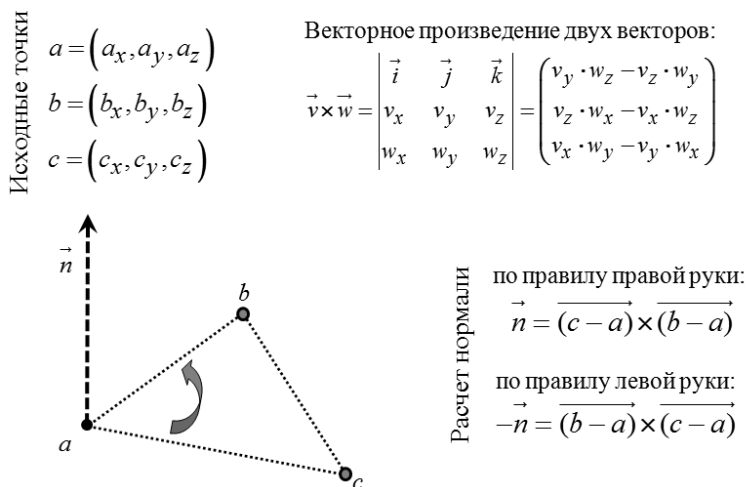


Рис. 5. Расчет нормалей по трем вершинам

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.

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

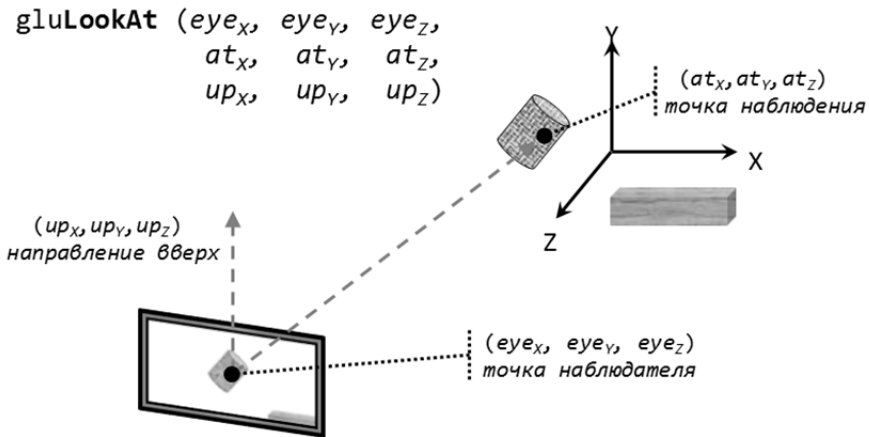


Рис. 7. Смена положения камеры (наблюдателя)

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.

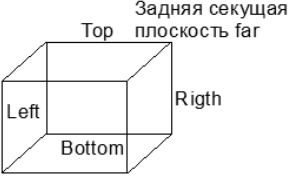
$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{far-near} & t_z \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad \begin{aligned} t_x &= -\frac{right+left}{right-left} \\ t_y &= -\frac{top+bottom}{top-bottom} \\ t_z &= -\frac{far+near}{far-near} \end{aligned}$$


Рис. 8. Ортогографическая проекция

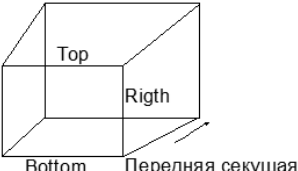
$$\begin{pmatrix} \frac{2 \text{ near}}{right-left} & 0 & A & 0 \\ 0 & \frac{2 \text{ near}}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{pmatrix}, \quad \begin{aligned} A &= \frac{right+left}{right-left} \\ C &= -\frac{far+near}{far-near} \\ B &= \frac{top+bottom}{top-bottom} \\ D &=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


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.

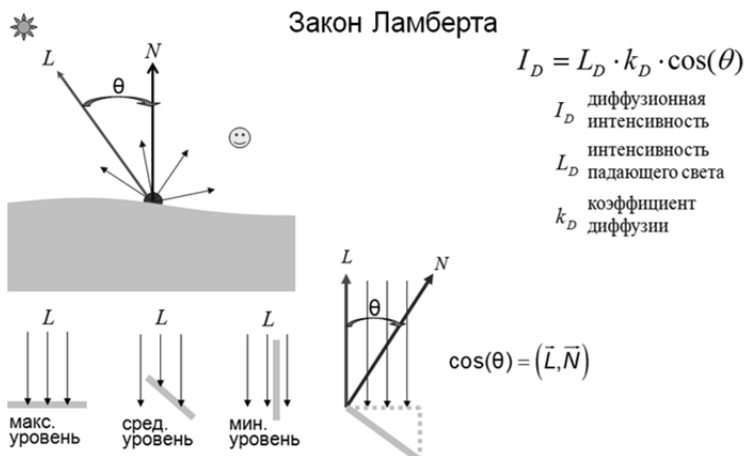
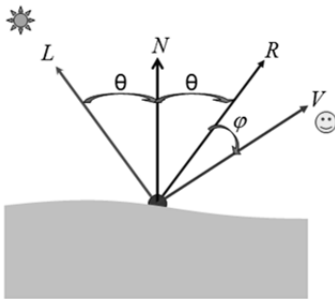


Рис. 10. Диффузная компонента освещения

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.

ЛАБОРАТОРНАЯ РАБОТА № 4

ТРАССИРОВКА ЛУЧЕЙ

ЦЕЛЬ РАБОТЫ

Ознакомиться с основными аспектами метода трассировки лучей.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Трассировка лучей (ray tracing) – это один из методов геометрической оптики (исследование оптических систем путем отслеживания взаимодействия отдельных лучей с поверхностями). Метод трассировки лучей используется для получения высокореалистичных изображений с учетом отражений и преломлений света.

К основным достоинствам метода можно также отнести возможность *рендеринга* гладких объектов без аппроксимации их полигонами, высокую алгоритмическую распараллеливаемость, корректность учета поля зрения и отсечения невидимых поверхностей. Серьезным же недостатком метода является производительность.

У метода трассировки лучей есть множество модификаций, которые в общем можно разделить на два класса: *прямую* трассировку и *обратную* (рис. 13). В методе прямой трассировки лучи испускается из источника света. В методе обратной трассировки лучи выпускаются из глаза наблюдателя через каждый пиксель экрана в сцену.

Алгоритм обратной трассировки лучей

1. Через каждый пиксель экранной плоскости выпускается луч в сцену и ищется точка его ближайшего пересечения с объектами сцены.

2. Из этой точки выпускаются лучи ко всем источникам света (для определения их видимости), вычисляется нормаль и рассчитывается освещенность данной точки по выбранной модели освещения (Фонга, Кука–Торренса и др.).

3. Для определения световой энергии вторичного освещения из этой же точки выпускаются также отраженный и преломленный лучи, каждый из которых трассируется для определения точки ближайшего пересечения. Затем снова может потребоваться трассировка возникающих отраженных и преломленных лучей.

4. Критерии останова рекурсии: достижение заданного уровня рекурсии или веса луча, убывающего с расстоянием.

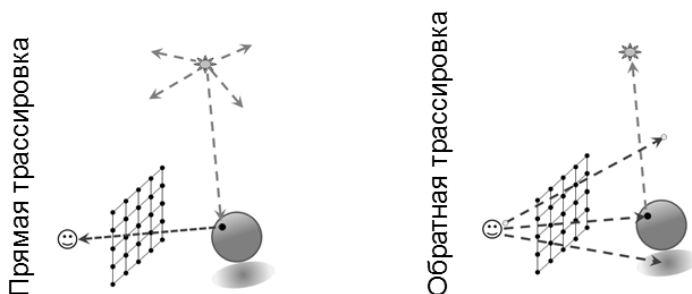


Рис. 13. Прямая и обратная трассировка лучей

Построение луча и поиск пересечений с объектами

Принцип построения уравнения луча $r(t)$, испускаемого из глаза наблюдателя (точка *Eye*) и проходящего через (r, c) – пиксель, представлен на рис. 14.

Такие объекты, как сфера и цилиндр удобно задавать в наиболее простом (каноническом) виде – в неявной форме вида $F(x, y, z) = 0$, что позволяет проще решать уравнение $F(x(t), y(t), z(t)) = 0$. Тогда вместо пересечения исходного луча $r(t)$ с преобразованным объектом достаточно легко найти пересечение обратно преобразованного луча с каноническим объектом: $F(M^{-1}r(t))=0$, где M – это некоторое модельно-видовое преобразование, преобразующее объект из канонической формы. На рис. 15–17 представлены примеры уравнений и расчетов точки пересечения луча с канонической сферой, базовой плоскостью и каноническим цилиндром соответственно.

Если, например, требуется найти пересечение луча с треугольником, тогда по трем вершинам строится уравнение плоскости (в которой лежит треугольник), ищется точка пересечения луча с этой плоскостью, а потом (в случае успеха) уже определяется принадлежность данной точки треугольнику. Построение уравнения плоскости по трем точкам a, b и c показано на рис. 18.

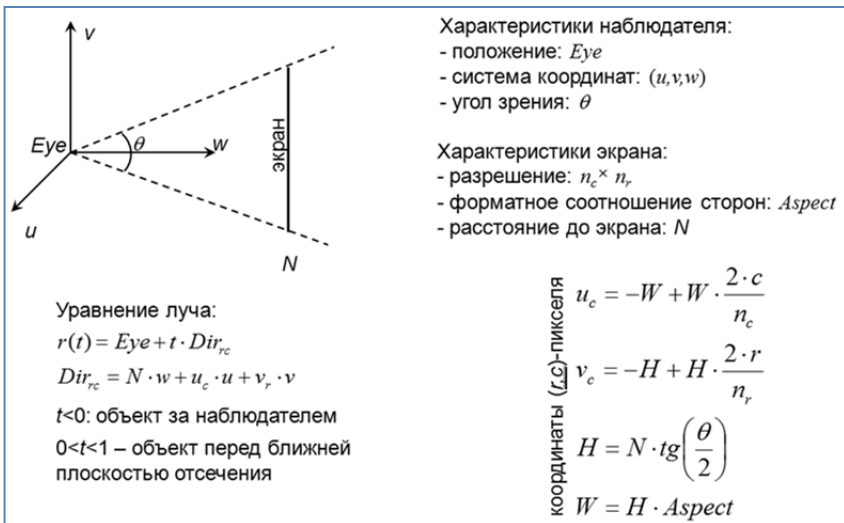


Рис. 14. Расчет уравнения луча

Общее уравнение сферы: $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$

Каноническое уравнение сферы: $x^2 + y^2 + z^2 = 1$

Параметрическое уравнение базовой сферы: $F(\bar{P}) = (\bar{P}, \bar{P}) - 1$

1. Подставляем уравнение луча: $(e + dt)^2 - 1 = 0$

2. Получаем квадратное уравнение: $at^2 + 2bt + c = 0$

где: $a = (d, d)$, $b = (e, d)$, $c = (e, e) - 1$,

дискриминант $s = b^2 - 4ac$

3. Находим корни: $t = \begin{cases} \otimes, & \text{если } s < 0 \text{ (нет пересечений)} \\ -\frac{b}{a}, & \text{если } s = 0 \text{ (одно пересечение)} \\ \frac{-b \pm \sqrt{s}}{a}, & \text{если } s > 0 \text{ (два пересечения)} \end{cases}$

Рис. 15. Пересечение луча со сферой

Общее уравнение плоскости: $ax^2 + bx + c = 0$

Уравнение базовой плоскости (xy-плоскости): $F(x, y, z) = z$

Луч $r(t)$ пересекает плоскость $z=0$ когда: $e_z + d_z t = 0$

Решение уравнения:

$$t = \begin{cases} \otimes, & \text{если } d_z = 0 \text{ (нет пересечений, луч параллелен плоскости)} \\ -\frac{e_z}{d_z}, & \text{если } d_z \neq 0 \end{cases}$$

Точка пересечения с плоскостью: $T = e - d \frac{e_z}{d_z}$

Рис. 16. Пересечение луча с базовой плоскостью

Каноническое уравнение цилиндра:

$$F(P) = P_x^2 + P_y^2 - [1 + (s-1)P_z^2]^2$$

$$0 < z < 1$$

$$s = 1 \text{ (цилиндр)}$$

$$s = 0 \text{ (конус)}$$

Уравнение для боковой поверхности ($P_z \in [0,1]$):

$$at^2 + bt + c = 0,$$

$$\text{где: } a = Dir_x^2 + Dir_y^2 - d^2$$

$$b = Eye_x Dir_x + Eye_y Dir_y - fd$$

$$c = Eye_x^2 + Eye_y^2 - f^2$$

$$d = (s-1)Dir_z$$

$$f = 1 + (s-1)Eye_z$$

Уравнение для нижнего основания ($P_x^2 + P_y^2 < 1$): $Eye_z + Dir_z t = 0$

Уравнение для верхнего основания ($P_x^2 + P_y^2 < s^2$): $Eye_z + Dir_z t - 1 = 0$

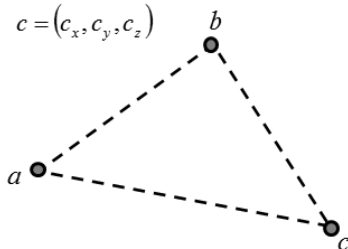
Рис. 17. Пересечение луча с каноническим цилиндром

Стандартное уравнение плоскости: $Ax + By + Cz + D = 0$

$$a = (a_x, a_y, a_z)$$

$$b = (b_x, b_y, b_z)$$

$$c = (c_x, c_y, c_z)$$



Вектор нормали: (A, B, C)

Уравнение плоскости
в координатном виде:

$$\begin{vmatrix} x - a_x & y - b_x & z - c_x \\ a_y - a_x & b_y - b_x & c_y - c_x \\ a_z - a_x & b_z - b_x & c_z - c_x \end{vmatrix} = 0$$

Расчет
коэффициентов
уравнения

$$A = \begin{vmatrix} 1 & a_y & a_z \\ 1 & b_y & b_z \\ 1 & c_y & c_z \end{vmatrix}$$

$$B = \begin{vmatrix} a_x & 1 & a_z \\ b_x & 1 & b_z \\ c_x & 1 & c_z \end{vmatrix}$$

$$C = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$$

$$D = - \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}$$

Рис. 18. Уравнение плоскости по трем точкам

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Считывать из файла (в зависимости от варианта)
 - а) тип объекта;
 - б) координаты и размер объектов;
 - в) параметры материала объектов.
2. Выполнить трассировку первичных лучей.
3. Добавить зеркальную плоскость и учесть отраженные лучи.
4. Предусмотреть возможность включения/исключения объектов.
5. Предусмотреть возможность изменения положения источника света.

ВАРИАНТЫ ЗАДАНИЙ

№	Объекты № 1	Объекты № 2
1	Сферы	Параллелепипеды
2	Сферы	Тетраэдры
3	Сферы	Прямоугольные призмы
4	Конусы	Параллелепипеды
5	Конусы	Тетраэдры
6	Конусы	Прямоугольные призмы
7	Цилиндры	Параллелепипеды
8	Цилиндры	Тетраэдры
9	Цилиндры	Прямоугольные призмы
10	Конические цилиндры	Параллелепипеды
11	Конические цилиндры	Тетраэдры
12	Конические цилиндры	Прямоугольные призмы

Дополнительные задания

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

- 1) учет текстур;
- 2) учет прозрачности;
- 3) сглаживание;
- 4) Модель Кука–Торренса (вместо модели Фонга).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Структура трассировщика лучей.
2. Построение теней и отражений.
3. Реализация прозрачности.
4. Способы ускорения расчетов.
5. Сглаживание.

РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

ИНТЕРАКТИВНОЕ СОЗДАНИЕ КРИВЫХ И ПОВЕРХНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ СПЛАЙНОВ

ЦЕЛЬ РАБОТЫ

Реализовать программу, отображающую график функции, получаемой в результате использования соответствующего сплайна.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Входными данными является набор точек на экране, задаваемый в произвольном порядке с помощью мыши. При добавлении новой точки сплайн должен автоматически перестраиваться.

Программа должна предоставить возможность движения по сцене и масштабирования с помощью клавиатуры/мыши.

Координатные оси и координатная сетка должны быть отображены и подписаны, причем при масштабировании размер ячеек сетки не должен меняться – меняются только числовые подписи и масштабируется сам график.

Для заданий с дифференцированием (интегрированием) на экране должен отображаться график не только самого сплайна, но и его производной (первообразной).

В том случае когда недостаточно контрольных точек для построения сплайна, следует выдавать соответствующее предупреждение, а сам сплайн, например, отображать ломаной.

Для повышения уровня сложности работы и получения дополнительных баллов нужно в своем варианте строить сплайн-поверхность. Входные данные можно считывать из файла.

ВАРИАНТЫ ЗАДАНИЙ

1. Интерполяционный сплайн на основе полиномов Лагранжа степени 2.
2. Интерполяционный сплайн на основе полиномов Лагранжа степени 3.
3. Интерполяционный сплайн на основе полиномов Лагранжа степени n .
4. Интерполяционный сплайн с использованием лагранжевых элементов второго порядка.
5. Интерполяционный сплайн с использованием лагранжевых элементов третьего порядка.
6. Интерполяционный сплайн с использованием эрмитовых элементов третьего порядка и производными, построенными с помощью полинома Лагранжа.
7. Интерполяционный сплайн с использованием эрмитовых элементов третьего порядка и непрерывными вторыми производными.
8. Сглаживающий сплайн с использованием эрмитовых элементов третьего порядка.
9. Сплайн Безье.
10. Замкнутый квадратичный B-сплайн.
11. Замкнутый квадратичный B-сплайн.
12. Замкнутый кубический B-сплайн.
13. Замкнутый кубический B-сплайн.
14. Замкнутый B-сплайн переменной степени.
15. Замкнутый B-сплайн переменной степени.
16. Замкнутый квадратичный NURBS-сплайн.
17. Замкнутый квадратичный NURBS-сплайн.
18. Замкнутый кубический NURBS-сплайн.
19. Замкнутый кубический NURBS-сплайн.
20. Замкнутый NURBS-сплайн переменной степени.
21. Замкнутый NURBS-сплайн переменной степени.
22. Дифференцирование с использованием сплайна из варианта 1.
23. Дифференцирование с использованием сплайна из варианта 2.
24. Дифференцирование с использованием сплайна из варианта 3.
25. Дифференцирование с использованием сплайна из варианта 4.

26. Дифференцирование с использованием сплайна из варианта 5.
27. Дифференцирование с использованием сплайна из варианта 6.
28. Дифференцирование с использованием сплайна из варианта 7.
29. Дифференцирование с использованием сплайна из варианта 8.
30. Интегрирование с использованием сплайна из варианта 1.
31. Интегрирование с использованием сплайна из варианта 2.
32. Интегрирование с использованием сплайна из варианта 3.
33. Интегрирование с использованием сплайна из варианта 4.
34. Интегрирование с использованием сплайна из варианта 5.
35. Интегрирование с использованием сплайна из варианта 6.
36. Интегрирование с использованием сплайна из варианта 7.
37. Интегрирование с использованием сплайна из варианта 8.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Соловейчик Ю.Г.* Метод конечных элементов для решения скалярных и векторных задач: учеб. пособие / Ю.Г. Соловейчик, М.Э. Рояк, М.Г. Персова. – Новосибирск: Изд-во НГТУ, 2007. – 896 с.
2. *Порев В.Н.* Компьютерная графика / В. Порев. – СПб., 2005. – 428 с.
3. *Коцюбинский А.О.* Компьютерная графика: практ. пособие. – М., 2001. – 750 с.
4. *Пономаренко С.И.* Пиксел и вектор. Принципы цифровой графики. – СПб., 2002. – 477 с.
5. *Петров М.Н.* Компьютерная графика: учеб. пособие / М.Н. Петров, В.П. Молочков. – СПб., 2002. – 735 с.: ил. + 1 CD-ROM.
6. *Дружинин А.И.* Алгоритмы компьютерной графики: учеб. пособие / А.А. Дружинин, В.В. Вихман. – Новосибирск: Изд-во НГТУ, 2003. – 54с.
7. *Дружинин А.И.* Алгоритмы компьютерной графики. Ч. 2: учеб. пособие / А.И. Дружинин. – Новосибирск: Изд-во НГТУ, 2007.
8. *Дружинин А.И.* Алгоритмы компьютерной графики. Ч. 3: учеб. пособие / А.И. Дружинин, Т.А. Дружинина. – Новосибирск: Изд-во НГТУ, 2009.

ОГЛАВЛЕНИЕ

Лабораторная работа № 1. Введение в программирование с использованием OpenGL	3
Лабораторная работа № 2. Растеризация примитивов	11
Лабораторная работа № 3. Трехмерная визуализация в режиме реального времени	17
Лабораторная работа № 4. Трассировка лучей	29
Расчетно-графическое задание. Интерактивное создание кривых и поверхностей с использованием сплайнов	35
Библиографический список	38

**КОМПЬЮТЕРНАЯ ГРАФИКА
С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OpenGL**

Методические указания

Редактор *И.Л. Кескевич*
Выпускающий редактор *И.П. Брованова*
Компьютерная верстка *Л.А. Веселовская*

Налоговая льгота – Общероссийский классификатор продукции
Издание соответствует коду 95 3000 ОК 005-93 (ОКП)

Подписано в печать 05.12.2016. Формат 60 × 84 1/16. Бумага офсетная. Тираж 100 экз.
Уч.-изд. л. 2,32. Печ. л. 2,5. Изд. № 353/15. Заказ № 40. Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20