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
`gl`	Имя библиотеки, где описана функция (OpenGL)
`Color`	Имя самой функции задания цвета
`3`	Количество передаваемых аргументов
`ub`	Тип передаваемых аргументов
`255,0,255`	Список аргументов (RGB-компоненты)

Таблица 2

Некоторые типы данных

Суффикс	Описание типа		Тип в OpenGL	Тип в C++
`b`	Целый	1 байт	`GLbyte`	*signed* `char`
`ub`			`GLubyte`	*unsigned* `char`
`i`		4 байта	`Glint`	*signed* `int`
`ui`			`Gluint`	*unsigned* `int`
`f`	Вещественный	4 байта	`GLfloat`	`float`
`d`		8 байт	`GLdouble`	`double`

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

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-кординат 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("Рисуем 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.