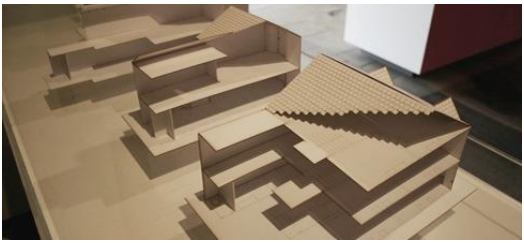
ФГБОУ ВПО «Воронежский государственный технический университет»
Кафедра компьютерных интеллектуальных технологий проектирования

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 1-3 по дисциплине «Программирование трехмерной графики» для студентов направления 09.03.02 «Информационные системы и технологии» всех форм обучения



Воронеж 2013

Составители: канд. техн. наук А.Н. Юров,
канд. техн. наук М.В. Паринов,
ст. преп. В.А. Рыжков, канд.
техн. наук А.С. Левченко

УДК 004.9

Методические указания к лабораторным работам № 1-3 по дисциплине «Программирование трехмерной графики» для студентов направления 09.03.02 «Информационные системы и технологии» всех форм обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А.Н. Юров, М.В. Паринов, В.А. Рыжков, А.С. Левченко. Воронеж, 2013. 57 с.

Методические указания содержат практический материал по работе с графическими компонентами OpenGL и GLUT.

Предназначены для студентов 2 курса. Ил. 19. Библиогр.: 8 назв.

Рецензент д-р техн. наук, проф. А.В. Кузовкин

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Печатается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Разработка приложений с интерактивным интерфейсом и графическим представлением данных является основным критерием при создании программных средств. Форма доступа к информации, которая отображена в виде графиков и диаграмм, воспринимается естественнее при вводе исходных данных, а также в процессе анализа результатов. Однако программные реализации при изучении той или иной технологии графического представления информации с использованием языка программирования, с которым у начинающих разработчиков уже был опыт общения, вызывают определенные трудности. Чтобы ознакомиться с основами графического вывода и представления расчетных результатов учащимся предлагается к изучению библиотека утилит для приложений glut, а также набор компонентов OpenGL с использованием языка программирования C++. В методических указаниях будет подробно изложена методика настройки производственных сред для разработки glut приложений в операционных системах Windows и Linux, создан каркас приложения для последующего развития, дана необходимая информация по функциям и командам, приведены примеры приложений с пояснениями. Кроме того, ряд лабораторных работ позволит приобрести и закрепить полученные навыки на контрольных примерах и заданиях.

РАЗРАБОТКА ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ GLUT

1.1. Функциональные возможности пакета glut

Использование библиотеки glut преследует две цели. Во-первых, это создание кроссплатформенного кода. Вовторых, glut позволяет облегчить изучение OpenGL. Чтобы начать программировать под OpenGL, используя glut, требуется всего страница кода. В свою очередь написание аналогичных вещей на API требует несколько страниц, написанных со знанием механизма управления окнами конкретной операционной системы.

Характерными особенностями OpenGL, которые обеспечили распространение и развитие указанного графического стандарта, являются следующие подходы[3]:

-Стабильность. Все изменения в стандарте сочетаются с имеющимися правилами разработки, тем самым обеспечивается совместимость с разработанным ранее ПО.

-Надежность и переносимость. Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой ОС и организации отображения информации. В методических указаниях приводятся примеры для ОС Linux и Windows.

-Легкость применения. Стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет быстрее создавать объектно-ориентированные приложения и подсистемы на их основе. Необходимые функции для обеспечения совместимости с оборудованием реализованы

на уровне библиотеки и значительно упрощают разработку программ.

Пакет инструментов для создания графических приложений glut позволяет разработчикам решать следующие задачи:

-использовать одни и те же функции в различных операционных системах;

-получать доступ и обрабатывать сообщения от устройств ввода информации (клавиатура, мышь и т.д.); -работать со шрифтами и изображениями;

-добавлять в программу события от таймера;

-использовать многооконные приложения с всплывающими меню;

-библиотека проста в использовании, а создание программ возможно на следующих языках программирования: C, C++, Fortran и т.д.

Набор программных средств glut доступен по следующему адресу в сети интернет:
http://www.opengl.org/resources/libraries/glut/glut_downloads.php
На указанной странице размещены инструменты графической библиотеки для ряда операционных систем. Кроме того в сети имеется открытая альтернатива инструментария glut - freeglut, которая позволяет пользователю создавать и управлять окнами, предоставляющими контекст OpenGL на широком спектре платформ, а также взаимодействовать с клавиатурой и мышью. freeglut предназначена для полной замены glut, и имеет очень немного отличий от неѐ. Адрес в сети интернет, по которому можно больше узнать о проекте, а также загрузить необходимые компоненты для разработки приложений следующий: http://freeglut.sourceforge.net.

Функции glut могут быть классифицированы на несколько групп согласно своей функциональности:

-инициализация;

-начало обработки событий;

-управление окнами;

-управление перекрытием;

-управление меню;

-регистрация вызываемых (callback) функций;

-управление индексированной палитрой цветов;

-чтение состояния;

-отображение шрифтов;

-отображение геометрических фигур;

1.2. Настройка сред разработки glut-проектов

Прежде чем приступить к написанию проектов с использованием glut, необходимо загрузить необходимые компоненты с интернет - ресурсов (ссылки приведены выше) для требуемых операционных систем. Причем есть пакеты, подготовленные для вставки непосредственно в интегрированную среду разработки приложений посредством бинарных сборок (bin), а есть варианты собрать и сконфигурировать glut (freeglut) самостоятельно по исходным файлам (src). Первый способ позволяет быстрее настроить производственную среду разработки glut. Рассмотрим содержимое готовой сборки для семейства операционных систем Microsoft. В бинарной сборке по ОС Win32 присутствует ряд файлов, которые отвечают за следующие действия:

-README-win32 - описание библиотеки. В этом файле перечислены описание модулей и файлов, особенности сборки программ, возможные ошибки, список последних изменений в

библиотеке и др. информация; glut.def - файл сведений о модуле dll (список функций,
которые есть в библиотеке); glut.h - заголовочный файл, в котором объявлены функции и константы, дополняющие функциональность языка. Его нужно скопировать в папку include среды, в которой планируется разработка, и включить в файл исходного кода программы строкой #include <GL/glut.h>. Кроме того файл может быть размещен в директории, где находится проектный файл - в таком случае в исходный код поставляется строка #include "glut.h"; glut32.dll - динамическая библиотека, включающая скомпилированные версии. Она нужна для использования функций в процессе выполнения программы, так как именно в этом файле присутствует реализация ключевых функций; glut32.lib - файл библиотеки, используемый при компиляции. Библиотеку указывают компоновщику, чтобы последний ссылался на прототипы функций с их реализацией. (пример: link example.obj glut32.lib (example.obj скомпилированный cpp программы, которая создается).

Теперь рассмотрим порядок настройки и сборки glut проектов в ОС Linux (Ubuntu) и Microsoft Windows 7

1.2.1. Сборка проектов в консольном режиме в Linux

Для выполнения glut проектов достаточно загрузить необходимые компоненты графических инструментов посредством Ubuntu Software Center или через терминал команд. Порядок установки компонентов для версии Ubuntu 12.04 посредством консольной установки следующий (узнать версию ОС можно следующим образом: вызываем консольное

окно терминала команд (Ctrl+Alt+T) и набираем команду `lsb_release -a`):

-вызвать консоль команд (Ctrl+Alt+T) и набрать следующую строку с перечнем команд: `sudo apt-get install freeglut3 freeglut3-dev libglew1.5 libglew1.5-dev libglu1-mesa libglu1-mesa-dev libgl1-mesa-glx libgl1-mesa-dev mesa-common-dev`.

При этом потребуется в консоли ввести пароль для того, чтобы нужные графические пакеты были установлены. Чтобы оптимизировать работу в терминале при копировании команд с электронного варианта методических указаний, рекомендуется использовать следующие комбинации клавиш - Shift+Ins – вставить текст и Ctrl+Ins – копировать текст. Если операция по вводу выполнена успешно, начнется установка графических библиотек. Процесс установки представлен на рисунке 1.

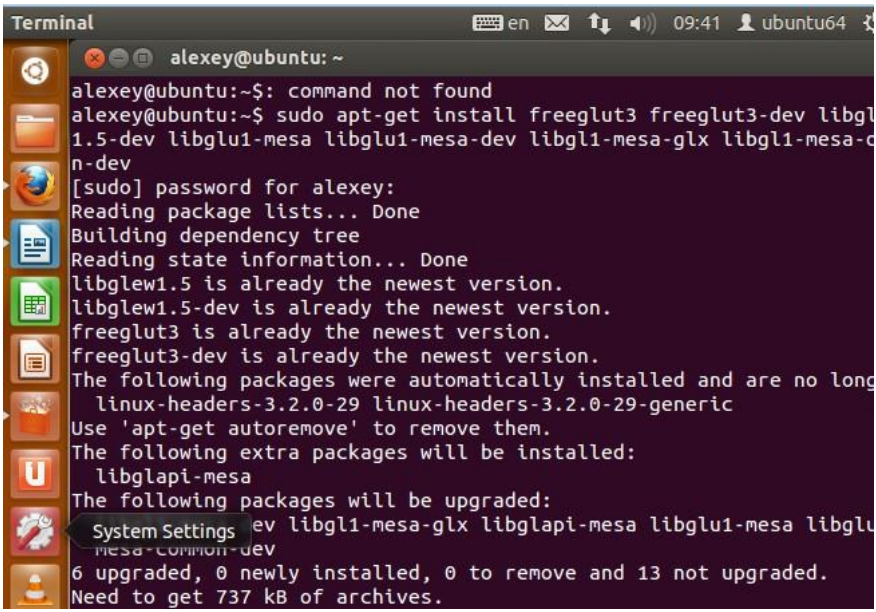


Рис.1. Установка пакетов OpenGL и glut в Ubuntu

-следующим шагом является установка компилятора под Linux. Процесс загрузки компилятора в Ubuntu выполняется следующим образом: `sudo apt-get install g++` и `sudo apt-get install gcc`.

-для подготовки исходных текстов программы использовать консольный режим необходимости нет, так как есть ряд текстовых редакторов (например, встроенный в Midnight Commander). Можно подготовить текстовый файл, воспользовавшись текстовым редактором gedit, установить который можно следующим образом: `sudo apt-get install gedit`.

-выполнить сборку в консольном режиме, подготовив тем самым исполняемый файл к запуску в Linux, можно следующим образом: `g++ file_name.cpp –lglut –o run_file`, где g++ компилятор и сборщик текстов на языке C++; file_name.cpp-имя файла с расширением .cpp-
определяется пользователем;

-lglut подключение библиотеки glut (-l опция для подключения библиотеки, сокращенно от library);

-o подготовить файл к запуску, однако можно таким образом задать уровень оптимизации компилируемого приложения, например -o3;

-run_file имя исполняемого файла, который будет создан после сборки проекта, при условии, что в тексте программы ошибок не было.

Рассмотрим пример по сборке приложения в консоли Ubuntu. Наберем следующий текст программы в gedit, листинг которого представлен на рисунке 2. Запишем набранный текст программы в файл base с расширением cpp. Подготовим исполняемый файл следующей строкой: `g++ base.cpp –`

`lglut -o firststart` и запустим на выполнение созданный файл с помощью файлового менеджера (рисунки 3 и 4).

1.2.2. Сборка проектов в IDE среде для ОС Linux

Создать приложение в IDE средах под Linux с использованием glut проще и эффективнее, если есть уже настроенная производственная среда разработки. Мы будем ориентироваться на IDE Code::Blocks-среда, которая доступна для установки из цента установки программного обеспечения Ubuntu. Выбираем вкладку по обновлению ПО и в строке поиска указываем название продукта C::B, затем производим установку (рисунок 5). Надо отметить, что IDE установится при условии доступа к сети Internet. На рисунке 5 отмечено, что приложение можно только удалить, что означает следующее - IDE установлена в системе и готова к работе. Установка отдельных элементов среды, если по каким-либо причинам они не были развернуты в системе, производится следующим образом:

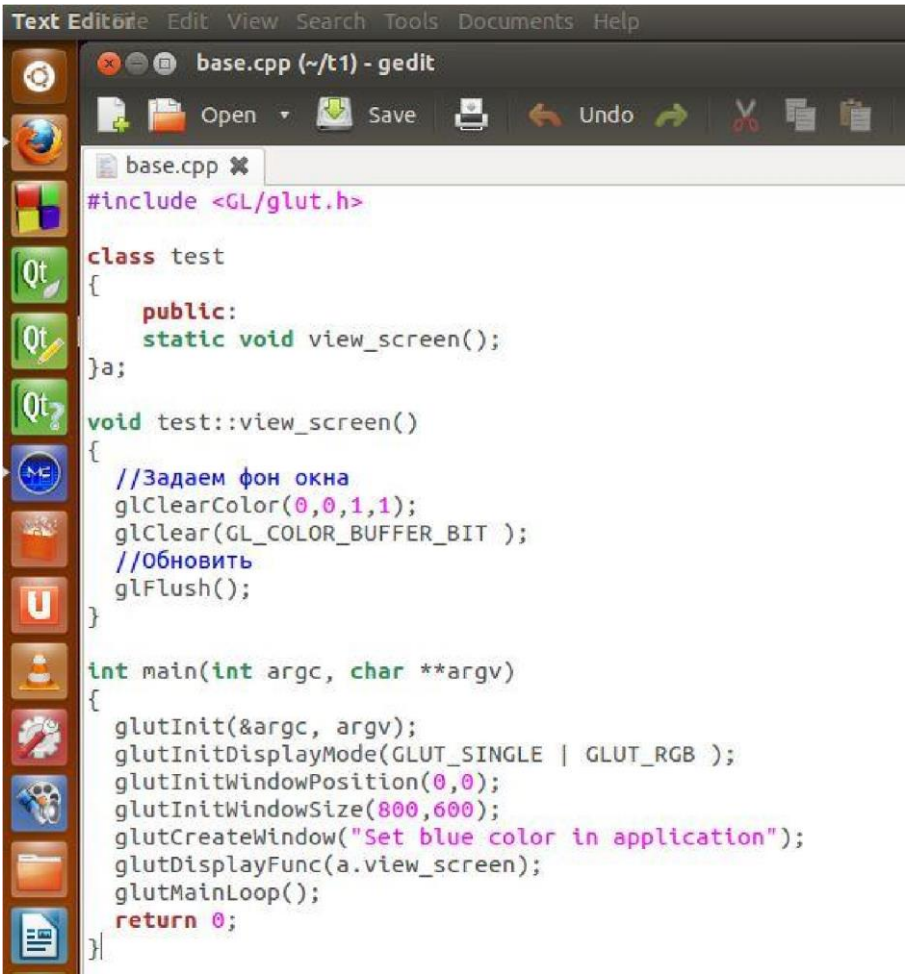


Рис. 2. Листинг тестового приложения

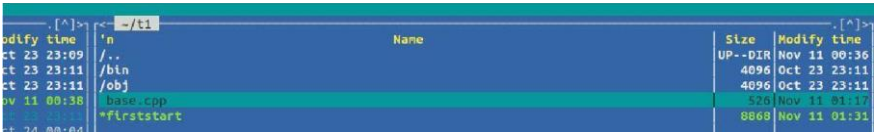


Рис. 3. Создание и запуск приложения из файлового менеджера

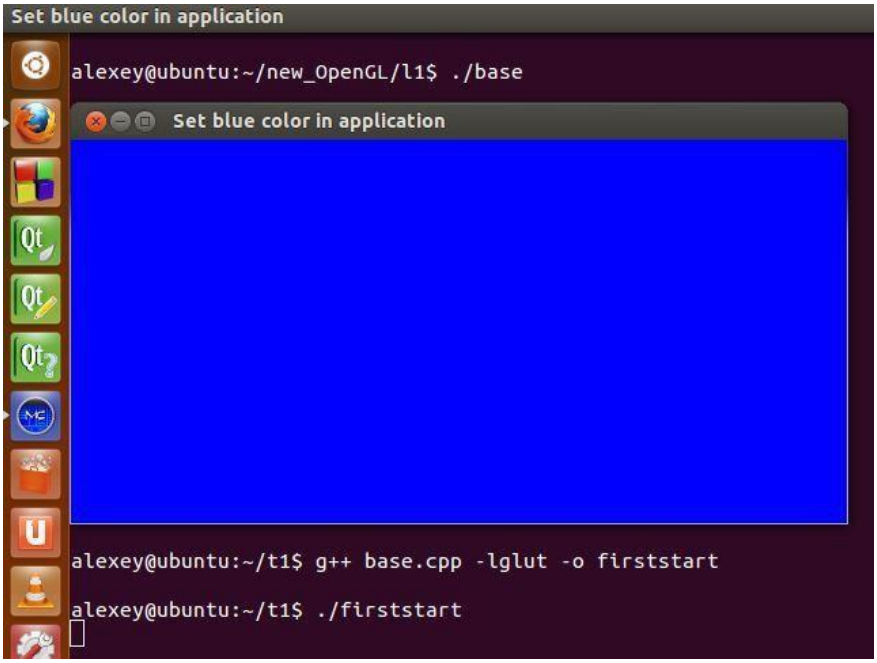


Рис. 4. Приложение glut под Ubuntu в действии

1.Установка из теримнала компилятора: `sudo aptget install build-essential`

2.Установка отладчика: `sudo apt-get install gdb`

3.Установка библиотек, а также компонентов wxWidgets: `sudo apt-get install libwxgtk2.8-0` 4.Установка документации по библиотеке визуальных компонентов: `sudo apt-get install wx2.8-doc`

5.Установка среды Code::Blocks из терминала: `sudo aptitude install codeblocks codeblocks-contrib`

После установки запускаем Code::Blocks IDE. При первом запуске у вас спросят, какой компилятор использовать по умолчанию, здесь просто выберем «GNU GCC compiler».

Остальные действия по настройке glut компонентов необходимо выполнить из предыдущего раздела. Если выбрано построение консольного проекта, необходимо зайти в раздел Project и выбрать секцию build option с добавлением glut, как показано на рисунке 6.

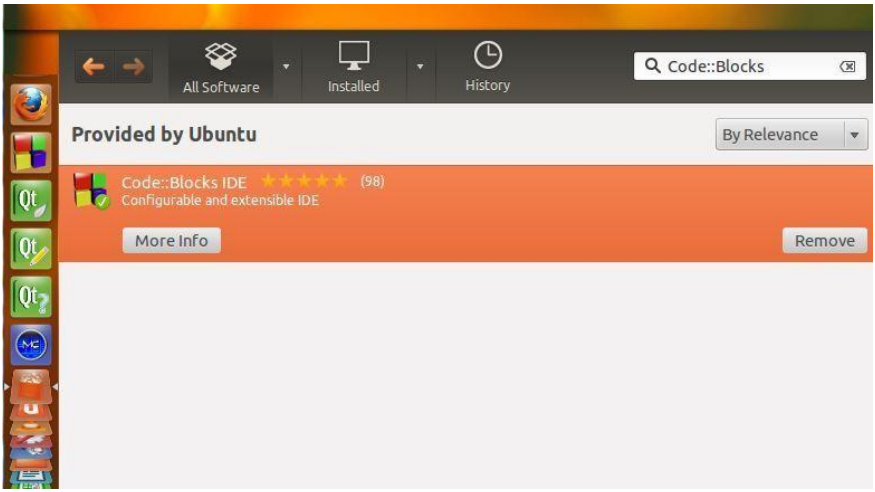


Рис. 5. Установка Code::Blocks средствами Ubuntu

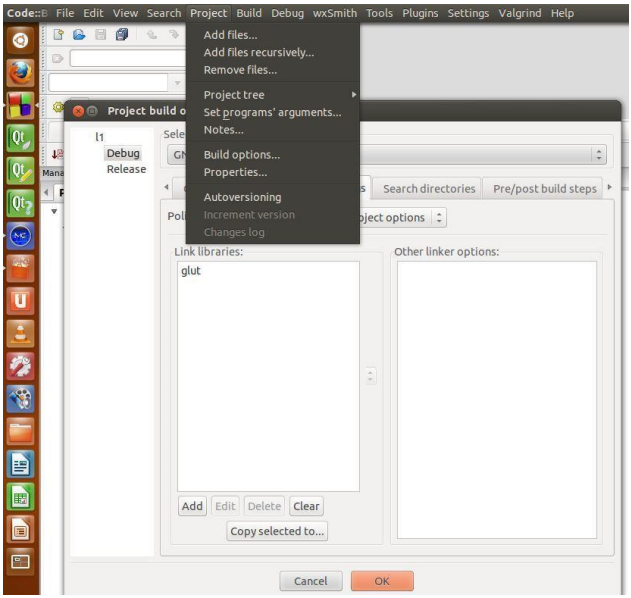


Рис. 6. Настройка проекта под glut в IDE Code::Blocks

На рисунке 7 показана сборка проекта. При этом интегрированная среда разработки позволяет в любой момент времени воспользоваться отладочными средствами и редактором исходного кода.

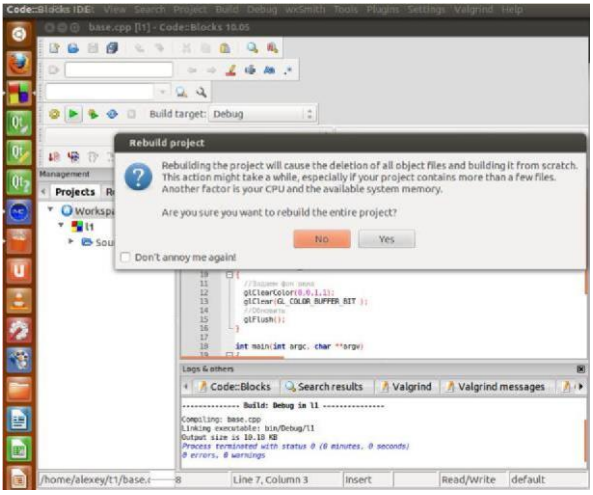


Рис. 7. Создание и работа с glut проектом в Code::Blocks

1.2.3. Создание проектных решений glut в Code::Blocks в ОС Windows

Подготовить приложение glut под Windows в Code::Blocks рекомендуется следующим образом:

-производим установку дистрибутива Code::Blocks для ОС Windows с встроенным MinGW. C::B может использовать произвольный компилятор для сборки приложений, поэтому есть версия дистрибутива, состоящая из графической оболочки с меню - она несколько компактнее в размере.

Проект можно создать пустым “Empty project”, при этом указав название и местоположение проекта и назначить компилятор. По умолчанию доступен GNU GCC Compiler C/C++ при условии, что была произведена установка в конфигурации с данным компилятором. Далее необходимо указать библиотеки, которые нужны для сборки OpenGL и glut. Для этого выбираем Projects/Build options, появляется окно, слева необходимо выбрать самое первое, которое содержит название проекта. Теперь переходим к вкладке Linker Settings и в Link libraries, добавляем соответствующие библиотеки, но их добавлять нужно таким же образом, как и при обычной сборке но без –l: SDL GL GLU glut (добавлять нужно по- одному).

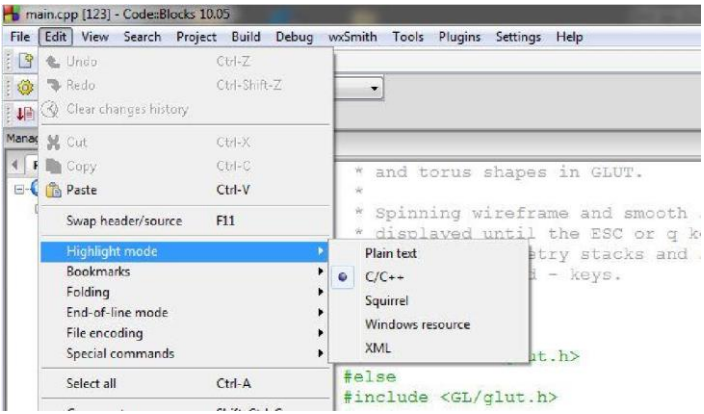


Рис. 8. Изменение стиля подсветки кода программы

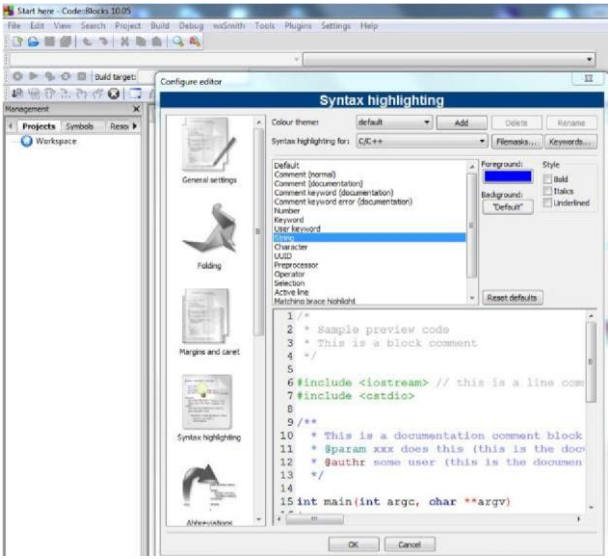


Рис. 9. Переопределение режимов отображения текста

Кроме того, если необходимо изменить систему подсветки листинга программы, выбираем режим отображения текстов для C/C++ (рисунок 8), а в случае переопределения цветов на другие, выполняем действия согласно рисунку 9.

При сборке и выполнении проекта выбираем пункт меню Build and run, либоF9.

Порядок настройки и выполнения glut приложений для Windows 7 (64 bit) следующий:

1.Копируем файл glut32.dll в папку установки Windows, например c:\windows\.

2.Содержимое папки freeglut-MinGW-2.8.0-1.mp (все файлы подготовлены уже для работы с Windows). копируем по месту установки Code::Blocks в директорию
MinGW: bin->bin; include->include; lib->lib. Для Windows 7 64bit путь установки Code::Blocks C:\Program Files (x86)\CodeBlocks\MinGW.

3.При создании нового проекта мастером приложений OpenGL glut необходимо указать на расположение необходимых библиотек в папке MinGW, т.е. прописать, например, C:\Program Files (x86)\CodeBlocks\MinGW. В процессе выбора пути для скопированных файлов возможно потребуется ссылка проекта на glut.h в подпапке include\GL. Следует отметить, что в готовых проектах среды Code::Blocks при указании опций компилятора для Windows и
Linux могут отличаться формы записи подключаемых библиотек: для Windows в настройках (Project->Build options>Linker settings->Link libraries), например, требуется указывать glu32, а для ОС Linux по аналогичному подходу: GLU.

ЛАБОРАТОРНАЯ РАБОТА №1
БАЗОВЫЕ ВОЗМОЖНОСТИ OPENGL И GLUT

Цель работы: освоить приемы использования простейших графических примитивов при создании кроссплатформенных приложений.

Задачи и требования к проекту разработки:

1.Изучить базовый набор функций графических библиотек и использовать их при создании приложений в ОС Windows и Linux.

2.Предложить собственную объектно-ориентированную модель графического приложения.

3.Подготовить решение и создать программную реализацию согласно предложенному заданию.

Теоретические сведения

Проектирование объектно-ориентированной модели графического приложения

Подготовим простейшее приложение, отображающее треугольник на плоскости. Сборку приложения выполним в среде Code::Blocks под Ubuntu. Листинг программы к выполнению представлен следующим кодом:

```
//#include <GL/glut.h>
#include <GL/freeglut.h>
#include <iostream>
//#include <GL/gl.h>
#define rotation 1
//Описание методов класса по построению объектов
class CoreGraph
```

```
{ public:
static void OutGraph();
CoreGraph();
};
//Реализация
void CoreGraph::OutGraph()
{
glClear (GL_COLOR_BUFFER_BIT);
glColor3f (0.0, 0.0, 0.0); //Построение
треугольника по точкам
glBegin(GL_TRIANGLES);
glVertex3f (rotation-0.5, rotation-0.25, 0.0);
glVertex3f (rotation-0.75, rotation-0.75, 0.0);
glVertex3f (rotation-0.25,rotation- 0.75, 0.0);
glEnd(); glFlush();
}
CoreGraph::CoreGraph()
{
//установим белый фон glClearColor
(1.0, 1.0, 1.0, 0.0);
//определение условий вывода
glMatrixMode(GL_PROJECTION); glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0, 0.0, 1.0);
}
//Основная функция по созданию консольного
//приложения
int main(int argc, char **argv)
{
glutInit(&argc, argv); glutInitDisplayMode
( GLUT_RGB); glutInitWindowSize (250,
250); glutInitWindowPosition (100, 100);
glutCreateWindow ("Base Graphics with GLUT");
CoreGraph *pnt=new CoreGraph();
glutDisplayFunc(pnt->OutGraph); glutMainLoop();
return 0;
}
```

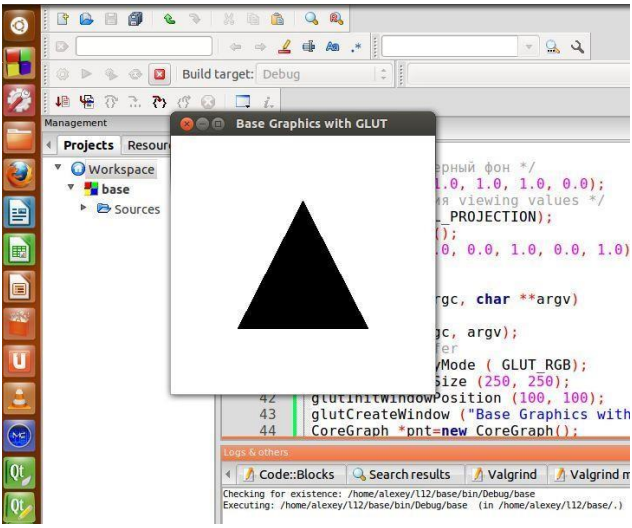


Рис. 10. Выполнение glut приложения

Результатом работы программы будет изображение, представленное на рисунке 10. Для того чтобы по выбранному базовому консольному проекту промежуточное окно не появлялось на экране, производим настройку в среде Code::Block согласно рисунку 11.

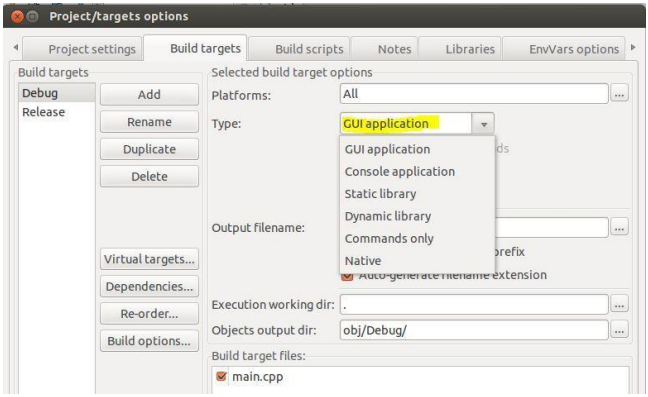


Рис. 11. Скрытие консольного окна при выводе изображения на экране ЭВМ

Разберем построчно программный код. Строки директив предпроцессора (начинаются с символа #), позволяют подключить необходимые glut компоненты к приложению. Причем необходимо написать `#include <GL/freeglut.h>`, либо `<GL/glut.h>` в зависимости от того, какие заголовочные файлы установлены и используются в среде разработки. Ссылаться на заголовочный файл `gl.h`, включающий функции OpenGL нет необходимости, потому что предыдущие директивы обеспечивают по умолчанию его подключение. `#define rotation 1`-обеспечивает поворот объекта за счет назначения точкам новых координат (данная реализация будет рассмотрена позднее).

Следующий шаг по созданию приложения - построение класса `CoreGraph`. Класс содержит метод, отвечающий за вывод графического объекта на экран `void CoreGraph::OutGraph()` и конструктор класса `CoreGraph::CoreGraph()`, где производится базовая инициализация параметров объекта и первоначальные установки. Рассмотрим действие команд метода OutGraph().

Строка `glClear (GL_COLOR_BUFFER_BIT)` предназначена для опустошения некоторой области памяти. Возможно использование данной функции OpenGL со следующими параметрами:

`GL_COLOR_BUFFER_BIT` – производится очистка текущего буфера цвета, который активен на момент использования;

GL_DEPTH_BUFFER_BIT - производится очистка буфера глубины;

GL_ACCUM_BUFFER_BIT - производится очистка аккумулирующего буфера;

GL_STENCIL_BUFFER_BIT - очищает буфер трафарета. Если буфер для очистки не представлен, то glClear не вызывает никакого эффекта.

Строка glColor3f (0.0, 0.0, 0.0) позволяет определить цвет объекта. Функция имеет три параметра- составляющие красного, зеленого и синего оттенков, которые определяются вещественными значениями от 0 до 1.

Например, определив glColor3f (0.7, 0.35, 0.7), мы зададим светло-фиолетовый оттенок для отображаемых объектов.

Строка glBegin(GL_TRIANGLES) определяет границы, внутри которых заданы вершины примитива или группы примитивов, в данном случае треугольников. glEnd() завершает указанные действия. glBegin() может иметь следующие параметры:

GL_POINTS- определяется набор точек glVertex, которые необходимо отобразить на экране;

GL_LINES-каждая пара вершин позволяет построить отрезок;

GL_LINE_STRIP-отображается набор отрезков-начало каждого определяется конечными координатами предыдущего отрезка;

GL_LINE_LOOP-рисуется ломанная, причем ее последняя точка соединяется с первой;

GL_TRIANGLES-каждые три вызова glVertex задают треугольник, можно подготовить набор треугольников;

GL_TRIANGLE_STRIP-рисуются треугольники с общей стороной;

GL_TRIANGLE_FAN -тоже самое, но по другому правилу соединяются вершины;

GL_QUADS-каждые четыре вызова glVertex задают четырехугольник;

GL_QUAD_STRIP-четырехугольники с общей стороной;

GL_POLYGON-многоугольник задается набором вершин.

glFlush()-функция позволяет принудительно обеспечить вызов ряда команд OpenGL, которые были заданы до этого, гарантируя их выполнение за некоторое время. В ряде случаев, когда в приложении определены буферы памяти под хранение объектов, данная функция увеличивает производительность по отображению. Если же требуется дополнительная синхронизация процессов отображения, можно использовать функцию `glFinish()`. Указанная функция не завершится до тех пор, пока не будут выполнены все предыдущие команды. Однако это функция менее производительна в работе, чем `glFlush()`.

В конструкторе `CoreGraph::CoreGraph()` по функции `glClearColor (1.0, 1.0, 1.0, 0.0)` производится установка фона экрана приложения в белый цвет (значения определены по принципу RGB, который был рассмотрен ранее). Последний параметр в функции отвечает за прозрачность объекта.

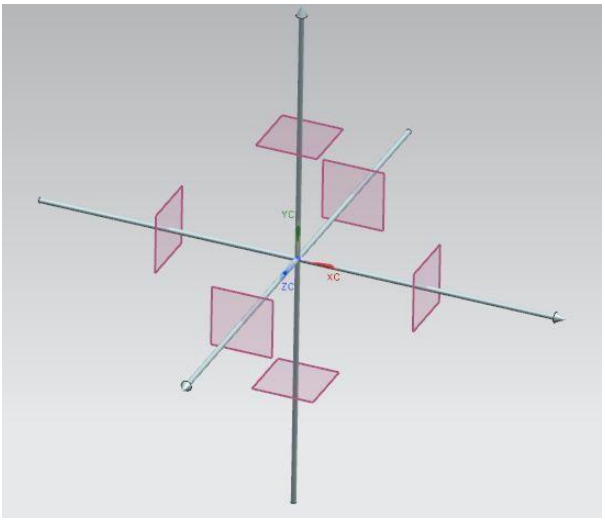


Рис. 12. Представление объектов на сцене

Перечень команд, состоящий из функций `glMatrixMode(GL_PROJECTION);glLoadIdentity();glOrtho(0.0, 1.0, 0.0, 1.0, 0.0, 1.0);` позволяет определиться с системой отображения объектов на экране компьютера, причем первые 2 функции обновляют систему координат, а `glOrtho(0.0, 1.0, 0.0, 1.0, 0.0, 1.0);`определяет объект на сцене по принципу задания некоторого места, заданного для оси x,y,z минимальным и максимальным значением согласно рисунку 12.

Параметры `glOrtho (x1,x2,y1,y2,z1,z2)` следующие:

x1-крайная левая координата объема отсечения; x2-крайная правая координата объема отсечения; y1-крайная нижняя координата объема отсечения; y2-крайная верхняя координата объема отсечения; z1-ближайшая координата от наблюдателя по оси z; z2-удаленная координата от наблюдателя по оси z;

Инициализация графики в основной функции main() производится рядом команд. При этом используется следующий подход: программа OpenGL, использующая GLUT, должна начинаться с инициализации GLUT-машины. Все функции инициализации имеют префикс glutInit. Главная инициализирующая функция называется glutInit:

glutInit(int **argcp, char **argv);

argcp - это указатель на еще не измененную переменную argc главной функции программы (main). После возврата из функции, значение, на которое указывает argcp, может измениться, так как glutInit использует все опции командной строки, относящиеся к библиотеке GLUT, например, в системе X-Window все опции, относящиеся к управлению окнами, ассоциируются с GLUT.

argv - это еще не измененная переменная argv главной функции.

glutInit позаботится об инициализации переменных состояния GLUT и откроет сессию с системой управления окнами. Есть всего лишь несколько функций, которые могут быть вызваны перед glutInit; это только те функции, которые имеют префикс glutInit-. Данные функции могут быть использованы для установки начального состояния окна.

glutInitWindowPosition(int x, int **y);

glutInitWindowSize(int width, int **height);

x,y = позиция окна на экране в пикселях (определяется позиция левого верхнего угла окна) width,

height - ширина и высота окна в пикселях.

Есть еще одна функция, которая присутствует практически во всех приложениях OpenGL-

glutInitDisplayMode(). Функция glutInitDisplayMode(unsigned int mode) имеет один входящий параметр, который может быть получен с помощью двоичной ИЛИ -комбинацией режимов GLUT (каждый режим представляет собой двоичную маску). Возможные значения режимов:

GLUT_RGBA-режим RGBA. Используется по умолчанию, если не указаны явно режимы GLUT_RGBA или GLUT_INDEX;

GLUT_RGB- аналогичный режим, что и GLUT_RGBA;

GLUT_INDEX-режим индексированных цветов. Отменяет GLUT_RGBA;

GLUT_SINGLE-окно с одиночным буфером. Используется по умолчанию;

GLUT_DOUBLE- окно с двойным буфером. Отменяет GLUT_SINGLE;

GLUT_ACCUM- окно с аккумулирующим буфером;

GLUT_ALPHA-окно с альфа-компонентой к цветовым буферам;

GLUT_DEPTH-окно с буфером глубины;

GLUT_STENCIL-окно с буфером трафаретов;

GLUT_MULTISAMPLE-окно с поддержкой multisampling;

GLUT_LUMINANCE-дублирующее окно с яркостной ("luminance") моделью цветов.

Так как в рассмотренном приложении создан класс, а функции графических библиотек используют обратные вызовы, то методы класса обязаны быть статичными, в противном случае компилятор выдаст ошибку.

Последняя команда `glutMainLoop()` главной функции `main()` обеспечивает основной цикл glut по обработке событий. В цикл могут попасть сообщения о нажатии клавиш с

клавиатуры, использование таймера, сообщение от перемещения манипулятора мыши, изменении размера окна и т.д. Функция используется до тех пор, пока программа не будет завершена.

Задания на самостоятельную работу:

1.Написать приложение, которое построено на основе разработанной объектно-ориентированной модели и позволяет визуально представить графики следующих функций: $y=\sin(x);y=x^2;y=tg(x)$.

2.Взяв за основу приложение из теоретической части и изменив вид матрицы проекции, подготовить вывод на экран пирамиды и куба.

ЛАБОРАТОРНАЯ РАБОТА №2
РАБОТА С КЛАВИАТУРОЙ, МАНИПУЛЯТОРОМ МЫШЬ И ТАЙМЕРОМ

Цель работы: использовать в разработанном приложении устройства ввода, а также обрабатывать сообщения от таймера вычислительной системы.

Задачи и требования к проекту разработки:

1. Подготовить в приложении методы, обеспечивающие обработку сообщений, поступающие от клавиатуры и манипулятора мышь.

2.Используя таймер, выполнить анимацию с базовыми геометрическими примитивами.

3.Создать проект, в котором предусмотрена обработка событий устройств, поступающих от пользователя в процессе работы приложения.

Теоретические сведения

Для обработки сообщений от клавиатуры в библиотеке glut используется функция `glutKeyboardFunc(unsigned char, int, int)`. Параметры указанной функции следующие: символьный тип отвечает за код клавиши (американская стандартная кодировочная таблица для печатных символов и некоторых специальных кодов-ASCII); второй и третий параметр относится к координатам указателя мыши на момент нажатия клавиши - могут не использоваться в проекте, однако в прототипе функции обязаны быть.

Сообщения от манипулятора мышь можно получить аналогичным образом, применив в проекте функцию `glutMouseFunc (int button, int state, int x, int y)`, где:

`button`- идентификатор кнопки мыши. Возможные значения для клавиш манипулятора мышь: `GLUT_LEFT_BUTTON`-левая клавиша, `GLUT_RIGHT_BUTTON`-правая клавиша, `GLUT_MIDDLE_BUTTON`-колесо, работающее в режиме кнопка; `state` - состояние кнопок мыши (`GLUT_DOWN`-нажата, `GLUT_UP` –отжата); `x, y`-координаты указателя мыши.

С учетом описанных функций спроектируем приложение, в котором производится построение треугольника, при нажатии левой клавиши мыши треугольник начинает вращение вокруг некоторой оси, при нажатии правой клавиши мыши осуществляется его остановка. Выход из программы происходит при нажатии на клавишу ESC. Листинг программы с комментариями приведен ниже:

```
      #include <GL/glut.h>
      #include <GL/freeglut.h>
      #include <iostream>
      #include <GL/gl.h>
//Модель glut приложения
class rotation {
//Переменная, отвечающая за угол вращения
      static GLfloat spin;      public:
//Метод первоначальных установочных параметров
//приложения      static void
      init();
//Метод, отвечающий за поворот объекта на
//плоскости      static void
      spinDisplay();
//Метод, в котором производится построение
```

```
//объекта     static void
      display();
//Метод по обработке событий от манипулятора мышь
      static void mouse(int button,int
state,int x,int y);
//Метод по представлению в пространстве объекта
      static void reshape(int w, int h);
//Метод по обработке сообщений от клавиатуры
      static    void    keyboard(unsigned    char
k,int,int);
      };

      GLfloat rotation::spin=0.0;

      void rotation::keyboard(unsigned char k,int
a,int b) {
          if (k==27) exit(0);
      }
      void rotation::init()
      {
        glClearColor(0.0,0.0,0.0,0.0);
      glShadeModel(GL_FLAT);
      }
      void rotation::display()
      {
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glPushMatrix();   glRotatef(spin,0.0,0.0,1.0);
glColor3f(0.0f,1.0f,0.0f);
       glBegin(GL_TRIANGLES);
              glVertex3f( 0.0f, 1.0f, 0.0f);
            glVertex3f(-1.0f,-1.0f, 0.0f);
             glVertex3f( 1.0f,-1.0f, 0.0f);
           glEnd();
      glutSwapBuffers(); glPopMatrix();
      }
      void rotation::spinDisplay()
```

```
      {
         spin=spin+1.0;
         if(spin>360.0) spin=spin-360.0;
      usleep(1000);
      //Функция перерисовки окна
      glutPostRedisplay();
      }
      void rotation::reshape(int w, int h)
      {
         glViewport(0,0,(GLsizei) w, (GLsizei) h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
         glOrtho(-5.0,5.0,-5.0,5.0,-1.0,1.0);
glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      }
      void     rotation::mouse(int     button,int
state,int x,int y)
      {
      switch(button)
      { case
      GLUT_LEFT_BUTTON:
       if
      (state==GLUT_DOWN)
      glutIdleFunc(spinDisplay); break; case
      GLUT_RIGHT_BUTTON:
       if (state==GLUT_DOWN) glutIdleFunc(NULL);
break;
      }
      } int main(int argc, char
**argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glutInitWindowSize(250,250);
glutInitWindowPosition(100,100);
glutCreateWindow("around triangles");
```

```
class rotation *d=new rotation(); d-
>init();
glutDisplayFunc(d->display);
glutReshapeFunc(d->reshape);
glutMouseFunc(d->mouse);
glutKeyboardFunc(d-
>keyboard); glutMainLoop();
return 0;
}
```

Функция `usleep(1000)` производит аппаратную задержку, эквивалентную приблизительно 1 секунде времени. Необходимость задержки нужна лишь в том случае, если в ЭВМ установлен дополнительный графических ускоритель графики. Для Windows следует применять `Sleep(int num)`, предварительно подключив заголовочный файл <windows.h> Результаты работы программы показаны на рисунке 13.

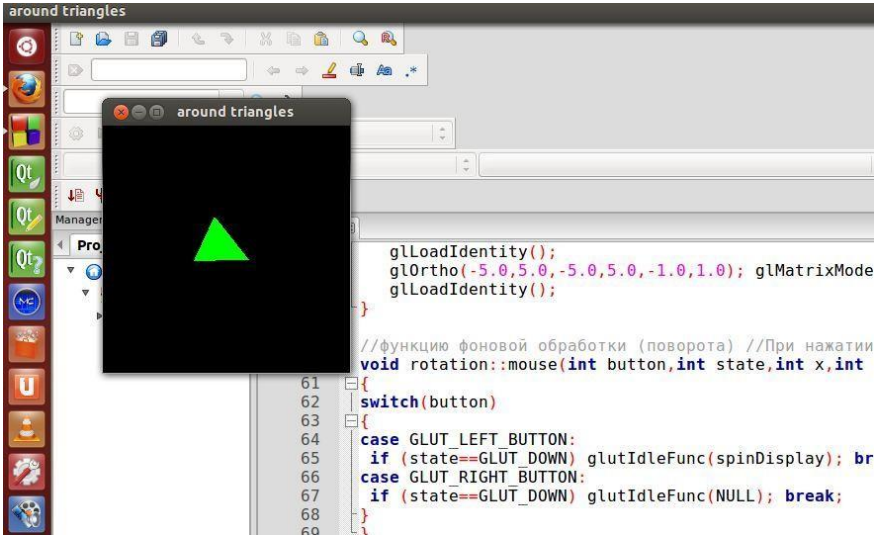


Рис. 13. Использование анимации при построении объектов

Подготовим пример с анимацией объектов, для запуска которой не требуется нажатия клавиш мыши. В качестве

графических объектов созданы треугольник и квадрат, причем фигуры по отдельности являются вписанными в предыдущие, тем самым вызывая несколько необычный визуальный эффект Пример, представленный на листинге, подготовлен на базе уже использованной ООП модели, содержит комментарии по созданию графических примитивов, а результаты работы программы в Ubuntu приведены на рисунке 14.

```
      #include <GL/glut.h>
      #include <GL/freeglut.h>
      #include <iostream>
      #include <GL/gl.h>

      class rotation
      {
//Маштабирование графического объекта
      static int dis;
          //Углы поворота мировой матрицы
      static float theta;     static
      float theta1;     public:
//Метод, отвечающий за поворот объекта на
//плоскости     static void
      drive();
//Метод, в котором производится построение
//графических объектов
      static void display();
//Метод по представлению в пространстве объекта
      static void reshape(int w, int h);
//Метод по обработке сообщений от клавиатуры
      static    void    keyboard(unsigned    char
k,int,int); }
      *d;
//Определяемые установочные параметры для углов
//пересчета и коэффициента масштабирования
      int rotation::dis=2; float
```

```
      rotation::theta=0.0f; float
      rotation::theta1=30.0f;

      void rotation::keyboard(unsigned char k,int
a,int b) {
          if (k==27) exit(0);
      }
      void rotation::display()
      {
       //Координаты базового треугольника
      float xA,yA,xB,yB,xC,yC;
       //Координаты рассчитанного треугольника
      float xxA,yyA,xxB,yyB,xxC,yyC; //Коэффициенты
      смещения новой точки (точек)   float p,q;
        glClearColor(0.0,0.0,0.0,0.0);
      glShadeModel(GL_FLAT);

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glPushMatrix();
       glRotatef(theta,1.0,0.0,1.0);
       glColor3f(0.0f,1.0f,0.0f);
            p=0.95;
                 q=1-p;
                  xA=0.0;xB=8.0;xC=4.0;
            yA=0.0;yB=0.0;yC=8.0;
            for(int i=0;i<50;i++)
                 {
                 glBegin (GL_LINE_LOOP);
      glColor3f(0.0f,1.0f,0.0f);
      glVertex2f(xA/dis,yA/dis);
glColor3f(1.0f,1.0f,0.0f);
glVertex2f(xB/dis,yB/dis);
glColor3f(1.0f,0.0f,1.0f);
glVertex2f(xC/dis,yC/dis);
xxA=p*xA+q*xB;yyA=p*yA+q*yB
;
xxB=p*xB+q*xC;yyB=p*yB+q*yC
```

```
;
xxC=p*xC+q*xA;yyC=p*yC+q*yA
; xA=xxA;xB=xxB;xC=xxC;
yA=yyA;yB=yyB;yC=yyC;
glEnd(); }
//Задаем угол вращения отдельно
glRotatef(theta1,1.0,0.0,0.0);
glColor3f(0.0f,1.0f,1.0f);
//Координаты базового прямоугольника     float
xA1,yA1,xB1,yB1,xC1,yC1,xD1,yD1; //Координаты
рассчитанного прямоугольника float
xxA1,yyA1,xxB1,yyB1,xxC1,yyC1,xxD1,yyD1;
xA1=-2.0;xB1=-8.0;xC1=-8.0;xD1=-2.0;
yA1=0.0;yB1=0.0;yC1=6.5;yD1=6.5;
for(int i=0;i<100;i++)
                  { glBegin
(GL_LINES);
glVertex2f(xA1/dis,yA1/dis);
glVertex2f(xB1/dis,yB1/dis);
glVertex2f(xB1/dis,yB1/dis);
glVertex2f(xC1/dis,yC1/dis);
glVertex2f(xC1/dis,yC1/dis);
glVertex2f(xD1/dis,yD1/dis);
glVertex2f(xD1/dis,yD1/dis);
glVertex2f(xA1/dis,yA1/dis);
xxA1=p*xA1+q*xB1;yyA1=p*yA1+q*yB1;
xxB1=p*xB1+q*xC1;yyB1=p*yB1+q*yC1;
xxC1=p*xC1+q*xD1;yyC1=p*yC1+q*yD1;
xxD1=p*xD1+q*xA1;yyD1=p*yD1+q*yA1;
xA1=xxA1;xB1=xxB1;xC1=xxC1;xD1=xxD1;
yA1=yyA1;yB1=yyB1;yC1=yyC1;yD1=yyD1;
glEnd();
                 } glutSwapBuffers();
     glPopMatrix();
     glutIdleFunc(drive);
     }
```

```
//Пересчет углов для 2х объектов void
rotation::drive()
      {
         theta +=.2f;
      theta1+=.2f;
      usleep(1000);
      glutPostRedisplay();
      } void rotation::reshape(int
w, int h)
      {
         glViewport(0,0,(GLsizei) w, (GLsizei) h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
         glOrtho(-5.0,5.0,-5.0,5.0,-15.0,15.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      }
      int main(int argc, char **argv)
      {
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
      glutInitWindowSize(600,600);
      glutInitWindowPosition(100,100);
      glutCreateWindow("Animation");
      glutDisplayFunc(d->display);
      glutReshapeFunc(d->reshape);
      glutKeyboardFunc(d->keyboard);
      glutMainLoop(); return 0;
      }
```

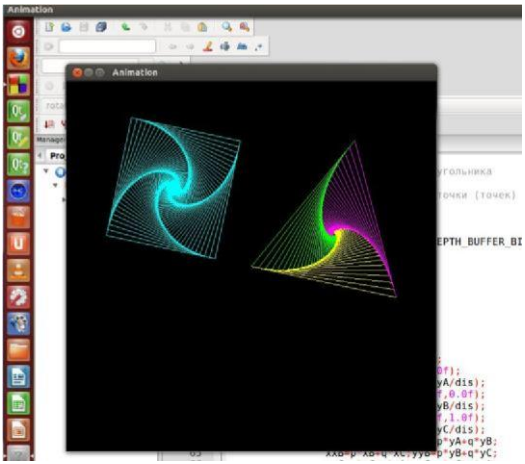


Рис. 14. Использование анимации при построении объектов

В анимационный процесс вращения объектов можно добавить готовые функции glut, которые позволяют вывести сферу, куб, конус, тор, додекаэдр, икосаэдр и прочие, причем возможен вывод контурный (функции имеют вложение, состоящее из слова Wire), так и полностью закрашенный (Solid). Пример построения икосаэдра в контурном виде следующий: `glutWireIcosahedron()`. Видно, что функция определена без параметров. Описание остальных готовых 3D примитивов построения можно найти в документации по glut.

На рисунке 15 показано добавление икосаэдра в проект и вывод его наряду с другими построениями.

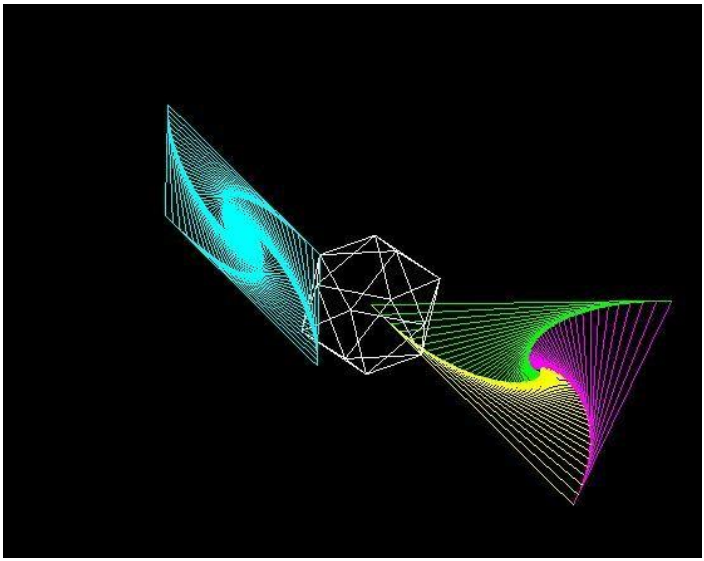


Рис. 15. Анимационные решения на основе готовых 3D примитивов **Задания на самостоятельную работу:**

1.Написать приложение, которое построено на основе разработанной объектно-ориентированной модели и позволяет визуально выполнить анимацию прямоугольника, оставляющего ”трек полета” в виде набора точек. Управление прямоугольником организовать с клавиатуры.

2.В приложении создать некоторый графический объект, имеющий несколько высот (вершин). Обеспечить вывод в консольное окно координат курсора манипулятора мышь при попадании указателя в вершину (вершины) имеющегося объекта.

ЛАБОРАТОРНАЯ РАБОТА №3
ГРАФИЧЕСКИЕ ПОСТРОЕНИЯ НА ПЛОСКОСТИ (ГРАФИКИ И СПЛАЙНЫ)

Цель работы: создать приложение GLUT и выполнить построение кривой заданного вида.

Задачи и требования к проекту разработки:

1.Рассмотреть приемы построения кривых на плоскости посредством геометрических примитивов OpenGL и GLUT.

2.Подготовить объектно-ориентированную модель приложения с выводом на экран кривой заданного вида.

Теоретические сведения

Кривая может быть представлена совокупностью точек. Если точки расположены близко друг от друга, то, соединяя их отрезками прямой, можно получить изображение некоторой кривой линии заданного вида (рисунок 16-а).

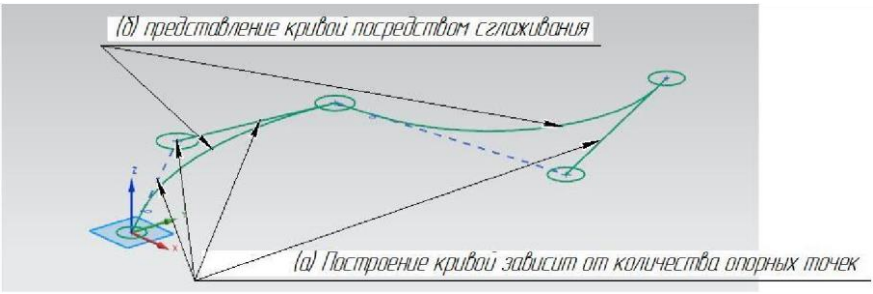


Рис. 16. Построение кривой

Однако вид кривой будет зависеть от количества точек и, следовательно, улучшить его можно увеличением

промежуточных точек, посредством различных методов по сглаживанию (рисунок 16-б), аналитическим путем задания геометрии кривой. Рассмотрим приложение GLUT, в котором выполнено построение части окружности тремя способами согласно рисунку 17 (построение координатных осей в приложении не предусмотрено). Согласно указанным формулам [2] подготовим 3 функции в пределах графического класса.

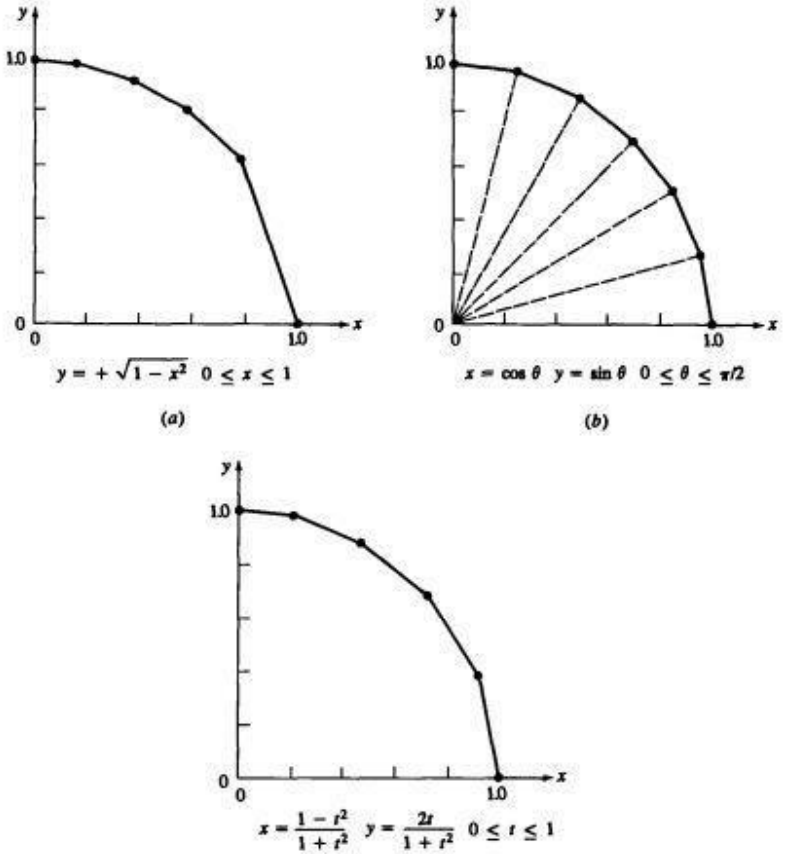


Рис. 17. Представление окружности для первого квадранта

```
#include <GL/glut.h>
#include <iostream>
#include <GL/gl.h>
#include <cmath>
class DrawCircles
{
protected:
//Способ построения части окружности в
//непараметрическом виде //0<=x<=1;y=SQRT(1-x^2)
static void build_circle_1(float a,float b,int c)
   {
 //Масштабный коэффициент для построения
int mashtab=c;
 //Точка с расчетными координатами x и y
 //a и b-пределы
float x=0,y;
  for (x=a;x<=b;x+=0.05)
{
  y=sqrt(1-x*x);
//Размер точек
glPointSize(3);
  //Указываем контрольные точки
glBegin(GL_POINTS);   glColor3d(0,1,0);
  glVertex3d(x*mashtab,y*mashtab,0);
glEnd();
}
  //Соединяем точки линиями for
(x=a;x<=b;x+=0.05)
{
  glBegin(GL_LINES);   y=sqrt(1-
x*x);
  glVertex3d(x*mashtab,y*mashtab,0);
x+=0.05;   y=sqrt(1-x*x);
  glVertex3d(x*mashtab,y*mashtab,0);
glEnd();   x-=0.05;
}
```

```
   }
//Способ построения части окружности в
//параметрическом виде
//x=cos(q);y=sin(q);0<=q<=PIE/2
static void build_circle_2(float a,float b,int c)
   {
 //Масштабный коэффициент для построения
int mashtab=c;
 //Точка с расчетными координатами x и y
 //a и b-пределы
float x,y,q;
  for (q=a;q<=b;q+=b/10)
{   y=sin(q);
x=cos(q);
//Размер точек
glPointSize(4);
  //Указываем контрольные точки
glBegin(GL_POINTS);   glColor3d(0,1,0);
  glVertex3d(x*mashtab,y*mashtab,0);
glEnd();
}
//Соединяем точки линиями   for
(q=a;q<=(b-b/10);q+=b/10)
{   y=sin(q);
x=cos(q);
glBegin(GL_LINES);
  glVertex3d(x*mashtab,y*mashtab,0);
q+=b/10;   y=sin(q);   x=cos(q);
  glVertex3d(x*mashtab,y*mashtab,0);
glEnd();   q-=b/10;
}
   }

//Способ построения части окружности в
//параметрическом виде
static void build_circle_3(int a,int b,int c)
   {
```

```
 //Масштабный коэффициент для построения
int mashtab=c;
 //Точка с расчетными координатами x и y
 //a и b-пределы
float x,y,t;   for
(t=a;t<=b;t+=0.1)
{
  x=(1-t*t)/(1+t*t);
y=2*t/(1+t*t);   //Размер
точек   glPointSize(5);
  //Указываем контрольные точки
glBegin(GL_POINTS);   glColor3d(0,1,0);
  glVertex3d(x*mashtab,y*mashtab,0);
glEnd();
}
  //Соединяем точки линиями for
(t=a;t<=b-0.1;t+=0.1)
{
  glBegin(GL_LINES);   x=(1-
t*t)/(1+t*t);   y=2*t/(1+t*t);
  glVertex3d(x*mashtab,y*mashtab,0);
t+=0.1;   x=(1-t*t)/(1+t*t);
y=2*t/(1+t*t);
  glVertex3d(x*mashtab,y*mashtab,0);
glEnd();   t-=0.1; }
   } };
class base:public DrawCircles
{
public:
   static void OnReshape(int w, int h)
    {
     if (h==0)
     h=1;
//установить область окна отображения
glViewport(0,0,w,h);
//установить матрицу проекции
glMatrixMode(GL_PROJECTION);    glLoadIdentity();
```

```
//используем перспективную проекцию
gluPerspective(45,(float)w/h,0.1,100);  //возврат
к матрице моделирования, чтобы была //возможность
перемещать построенный объект
glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
   }
  static void OnDraw()
  {
 glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_
BIT);
      glLoadIdentity();
gluLookAt( 0,0,30,
                     0,0,0,
                     0,1,0);
//Вызов функции моделирования
//непараметрический вид части окружности
build_circle_1(0,1,10); //Вызов функции
моделирования
//параметрический вид части окружности
//представление через sin и cos
build_circle_2(0,M_PI/2,8); //Вызов
функции моделирования
//параметрический вид части окружности
//представление соотношение сторон
build_circle_3(0,1,6);
      glutSwapBuffers();
}
static void OnInit()
{
      glEnable(GL_DEPTH_TEST);
}
};
int main(int argc,char** argv)
{
base *pointer=new base(); glutInit(&argc,argv);
```

```
glutInitDisplayMode(GLUT_DEPTH|GLUT_RGBA|GLUT_DOU
BLE);
glutInitWindowSize(800,600);
glutCreateWindow("circles 1 2 3");
glutDisplayFunc(pointer->OnDraw);
glutReshapeFunc(pointer->OnReshape);
pointer->OnInit(); glutMainLoop();
      return 0;
}
```

Результат работы приложения представлен рисунком 18.

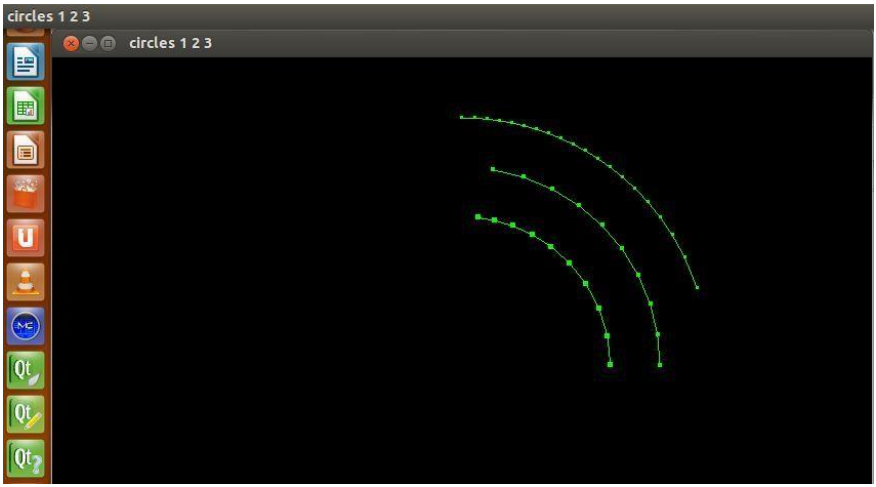


Рис. 18. Построение окружности для первого квадранта в GLUT

В следующем приложении представлены приемы построения кривых Безье и Nurbs относительно предопределенного набора точек. Точки определены в пространстве координатами x,y,z (в одном случае построения кривой координата z не используется) и записаны в массивы. Кривые формируются из отрезков - соответственно чем больше отрезков, тем кривая имеет более плавный вид переходов. В программе предусмотрена возможность задать количество

отрезков для сглаживания кривых посредством клавиатурных клавиш. Текст приложения представлен кодом, подготовленным в Code::Blocks в ОС Windows и содержит подробное описание, а результаты работы показаны на рисунке 19.

```
#include <stdlib.h>
#include <GL/glut.h>
//основной класс по использованию функций
//инициализации, изменению окна вывода и
//клавиатурного обработчика
class base { protected:
//уровни детализации кривых
static unsigned int LOD; static
unsigned int LOD1;
static void OnKeyPress(unsigned char key,int,int)
{
      switch(key) {
      // увеличение значения LOD
case '+':
           ++LOD;
LOD1+=2;
           break;

      // уменьшение значения LOD
case '-':
           --LOD;
           LOD1-=2;
    //установить минимальное значение LOD и LOD1
           if (LOD<3)
                LOD=3;
           if (LOD1<4)
                LOD1=4;
```

```
            break;
case 27 :     case
'q':
exit(0);
          break;
default:
            break;
      }
     //запросить glut перестроить экран
      glutPostRedisplay();
}
static void OnInit() {
      // тест буфера глубины
glEnable(GL_DEPTH_TEST);
}
static void OnReshape(int w, int h)
{
      if (h==0)
      h=1;
//установить размер вывода изображения в окне
      glViewport(0,0,w,h);
//установить матрицу проекции
glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
//использовать перспективную проекцию
gluPerspective(45,(float)w/h,0.1,100); //переход к
матрице моделей, если требуется
//работать с объектом построения
glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
}
};
//Определение статических элементов за пределами
//класса
unsigned int base::LOD=20; unsigned
int base::LOD1=LOD; //Класс по
```

```
проектированию кривых class
geometry:public base
{
static float g_Knots[11]; static
float g_Points[7][3]; static
unsigned int g_num_cvs; static
unsigned int g_degree; static
unsigned int g_order; static
unsigned int g_num_knots;

static float CoxDeBoor(float u,int i,int k,const
float* Knots) {
      if(k==1)
      {
            if( Knots[i] <= u && u <= Knots[i+1] )
{
                  return 1.0f;
            }
            return 0.0f;
      }
      float Den1 = Knots[i+k-1] - Knots[i];
float Den2 = Knots[i+k] - Knots[i+1];
      float Eq1=0,Eq2=0;
      if(Den1>0) {
                Eq1 = ((u-Knots[i]) / Den1) *
CoxDeBoor(u,i,k-1,Knots);
      }
      if(Den2>0) {
                Eq2 = (Knots[i+k]-u) / Den2 *
CoxDeBoor(u,i+1,k-1,Knots);
      }
      return Eq1+Eq2;
}
static void GetOutpoint(float t,float OutPoint[])
```

```
{     for(unsigned int i=0;i!=g_num_cvs;++i) {
//вычисление требуемых точек на кривой float
Val = CoxDeBoor(t,i,g_order,g_Knots);
if(Val>0.001f) {

OutPoint[0] += Val * g_Points[i][0];
OutPoint[1] += Val * g_Points[i][1];
OutPoint[2] += Val * g_Points[i][2];
           }
      }
}
//Массив точек для построения кривой Безье
static float Points[4][3];  public:
geometry()
{ }
static void OnDraw() {
    //Очистить экран и буфер глубины
      glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_
BIT);
//Обнулить предыдущие элементы пространства
//объектов
      glLoadIdentity();
//Установить камеру просмотра в заданную позицию
gluLookAt( 1,10,30, // позиция взгляда
                      0,0,0,// основная точка
                     0,1,0);// направление
//взгляда (первое-по x,второе -по y, третье-по z)

      glColor3f(1,0,1);
//отрисовка множества малых отрезков, которые в
//совокупности создадут кривую
glBegin(GL_LINE_STRIP);

      for(unsigned int i=0;i!=LOD;++i) {
//используется параметрическое значение t с
//диапазоном изменения от 0 до 1
```

```
            float t = (float)i/(LOD-1);
//используем преобразование для того, что в
//расчетах часто требуется изменение значения на
//обратное
            float it = 1.0f-t;
          //Вычисление функций сглаживания
            float b0 = t*t*t;
float b1 = 3*t*t*it;
float b2 = 3*t*it*it;
float b3 =  it*it*it;
//вычисление координат x,y,z точек кривой
//суммированием
            float x = b0*Points[0][0] +
                        b1*Points[1][0] +
              b2*Points[2][0] +
                 b3*Points[3][0] ;
            float y = b0*Points[0][1] +
                        b1*Points[1][1] +
              b2*Points[2][1] +
                 b3*Points[3][1] ;
            float z = b0*Points[0][2] +
                        b1*Points[1][2] +
              b2*Points[2][2] +
                 b3*Points[3][2] ;
          //заданная точка
            glVertex3f( x,y,z );
      }
      glEnd();
//отрисовка приграничных областей
glColor3f(0,1,0);
glPointSize(3);
glBegin(GL_POINTS);
for(int i=0;i!=4;++i) {
                glVertex3fv( Points[i] );
```

```
            }
      glEnd();
      //отрисовка контура кривой
glColor3f(0,1,1);
glBegin(GL_LINE_STRIP);
for(int i=0;i!=4;++i) {
glVertex3fv( Points[i] );
            }
      glEnd();
      //Построение nubrs кривой
glColor3f(1,1,0);     glBegin(GL_LINE_STRIP);
      for(unsigned int i=0;i!=LOD1;++i) {

             float t  = g_Knots[g_num_knots-1] * i
/ (float)(LOD1-1);
            if(i==LOD-1)
                  t-=0.001f;
            float Outpoint[3]={0,0,0};
GetOutpoint(t,Outpoint);
            glVertex3fv(Outpoint);
      }
      glEnd();
glColor3f(1,0,0);
glPointSize(3);
glBegin(GL_POINTS);
      for(unsigned int i=0;i!=g_num_cvs;++i) {
            glVertex3fv(g_Points[i]);
      }
      glEnd();
//сделать видимым требуемое построение можно так:
//изображение готовится в теневом буфере, а затем
//переносится на передний план
      glutSwapBuffers();
}
   static void other()
```

```
   {
//вызов функции по обработке изменений размера
//экрана
      glutReshapeFunc(OnReshape);
//установка обработчика нажатий на клавиши
//клавиатуры
      glutKeyboardFunc(OnKeyPress);
//вызов начальных установок
      OnInit();
   }
};
//Инициализация массивов за пределами класса float
geometry:: Points[4][3] = {
      { 10,-7,0 },
      {  5,-2,0 },
      { -5,-15,0 },
      {-10,-10,0}};
float geometry::g_Points[7][3] = {
      { 10,10,0 },
      {  5,10,2 },
      { -5,5,0 },
      {-10,5,-2},
      {-4,10,0},
      {-4,5,2},
      {-8,1,0}
};
float            geometry::g_Knots[]            =
{0.0f,0.0f,0.0f,0.0f,1.0f,2.0f,3.0f,4.0f,4.0f,4.0
f,4.0f};
unsigned int geometry::g_num_cvs=7; unsigned
int geometry::g_degree=3;
unsigned int geometry::g_order=g_degree+1;
unsigned                                        int
geometry::g_num_knots=g_num_cvs+g_order;
```

```
 int main(int argc,char** argv)
{     geometry value_class;
//установка GLUT
glutInit(&argc,argv);
//задание режима буфера глубины, дисплейного
//режима RGBA,двойной буферизации
glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE);
//установка размеров окна
glutInitWindowSize(640,480);
//создание окна
         glutCreateWindow("Other Curves: +/- to
Change Level of Detail");
//вызов функции по построению графической сцены
glutDisplayFunc(value_class.OnDraw); //доступ к
функциям по обработке клавиатурных //клавиш и т.д.
      value_class.other();
//бесконечный цикл, чтобы обеспечить работу
//приложения
glutMainLoop();
      return 0;
}
```

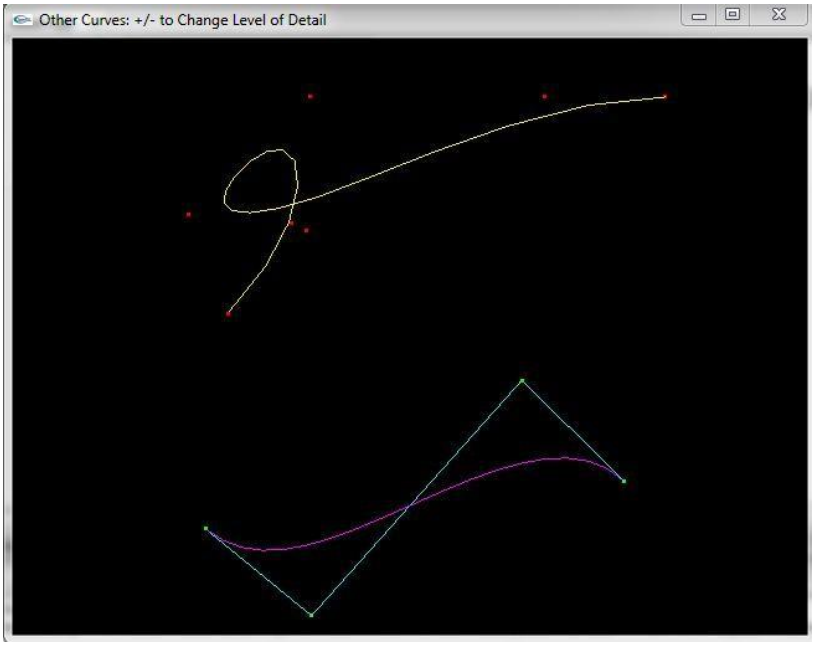


Рис. 19. Построение кривых в GLUT

Задания на самостоятельную работу:

1. Случайным образом создать несколько точек (от 5 до 10) и соединить их кривой таким образом, чтобы в заданной точке наблюдался радиус сглаживания.

2.Построить несколько точек и вокруг них вывести кривую, отстоящую от точек на заданном расстоянии. Предусмотреть в программе изменение расстояния с помощью клавиатурных клавиш.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Райт-мл Р.С. OpenGL. Суперкнига, 3-е изд. / Р.С. Райтмл, Б.Липчак -М.: ООО ”И.Д. Вильямс”, 2006. – 1040 с.
2. Роджерс Д. Математические основы машинной графики/ Д. Роджерс, Дж. Адамс- М.: Мир, 2001. – 604 с.
3. Баяковский Ю.М. Графическая библиотека OpenGL. Учебно-методическое пособие / Ю.М. Баяковский, А.В. Игнатенко, А.И. Фролов –М.: Изд. отдел факультета Вычислительной Математики и Кибернетики МГУ им. Ломоносова, 2003.-132 с.
4. Прата С. Язык программирования С++. Лекции и упражнения / С. Прата. 5-е изд. – М.: ООО ”И.Д. Вильямс”, 2007. – 1184 с.
5. Страуструп Б Язык программирования С++ /Б. Страуструп. - М.: Бином, 2011. – 1136 с.
6. Шилдт Г. С++ Базовый курс / Г. Шилдт. 3-е изд. – М.: ООО ”И.Д. Вильямс”, 2010. – 624 с.
7. Roberge J. A laboratory course in C++ structures. 2ed./ J. Roberge, S. Brandl, D. Whittington. Jones and Bartlett, 2003. -411 p.
8. London J. Modeling Derivatives in C++ / London J. Wiley, 2005. -841p.

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 1-3 по дисциплине «Программирование трехмерной графики» для студентов направления 09.03.02 «Информационные системы и технологии» всех форм обучения

Составители:
Юров Алексей Николаевич
Паринов Максим Викторович
Рыжков Владимир Анатольевич
Левченко Александр Сергеевич

В авторской редакции

Компьютерный набор А.Н. Юрова

Подписано к изданию 27.01.2013.
Уч.-изд. л. 3,0. «С»

ФГБОУ ВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14