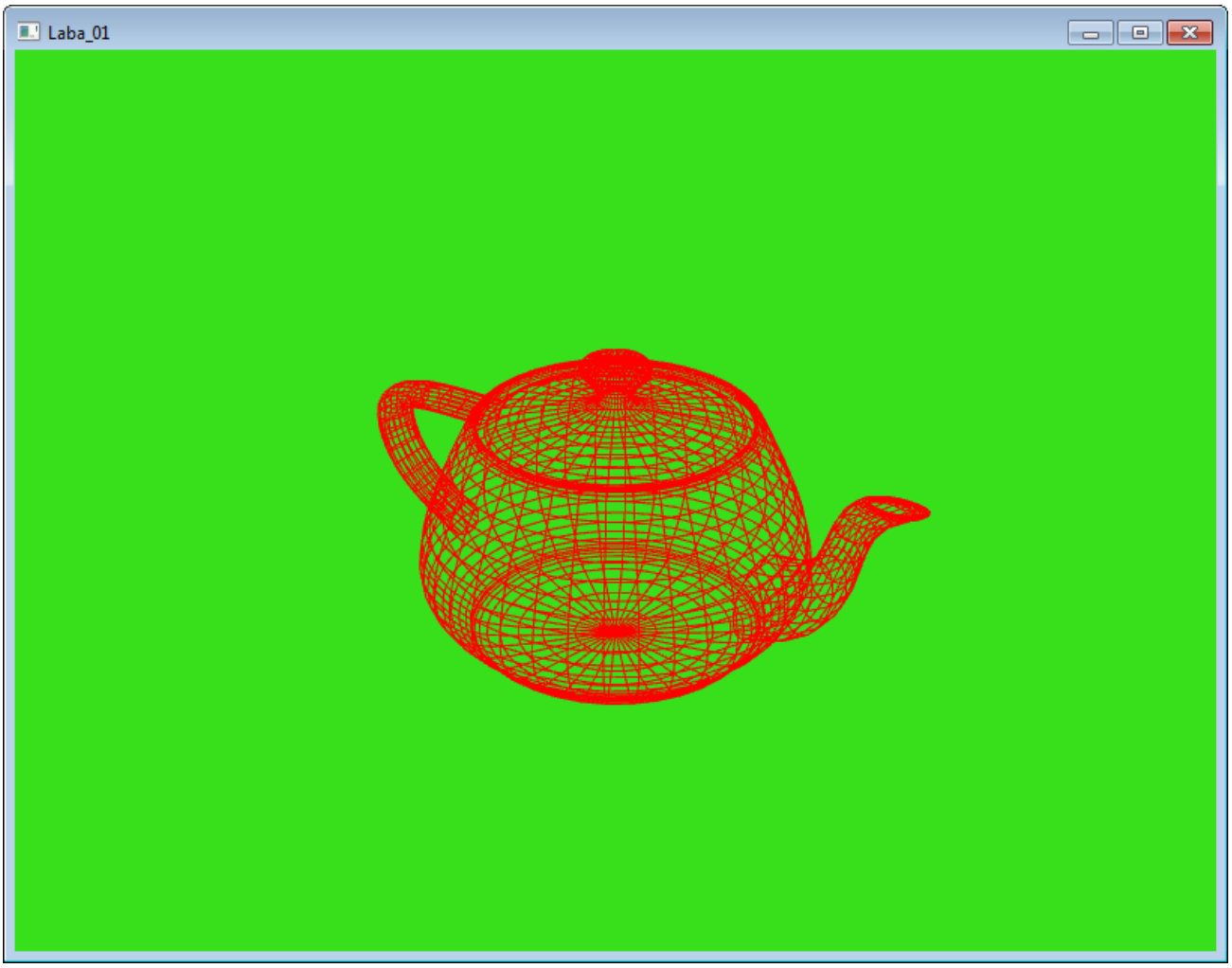
Лабораторная работа №1

Инициализация OpenGL.
Использование библиотеки freeglut.

Первая лабораторная работа призвана познакомить с библиотекой OpenGL, вспомогательной библиотекой GLUT (её более совершенной версией freeglut), а так же со средой программирования Visual Studio.

В качестве примера рассматривается создание первого проекта в среде Visual Studio 2015 (английская версия). Для других версий среды разработки основные этапы являются идентичными. К лабораторной работе прилагается исходный код, который можно использовать для ознакомления, однако желательно создать свой собственный проект «с нуля».



Порядок выполнения лабораторной работы.

При создании любой программы важно следовать от простого к сложному, на каждом шаге контролируя корректность данного этапа. Если сначала написать всю программу, а потом пытаться найти в ней ошибку, то можно столкнуться с большими трудностями, когда непонятно, какой именно этап работает некорректно.

Рассмотрим, для примера, написание алгоритма сортировки очереди. В этом случае работа сложного и объемного алгоритма по сортировке может работать неправильно только по тому, что неверно написана процедура добавления элемента в очередь. С другой стороны, если сначала написать процедуру добавления элемента в очередь и тщательно ее проверите, то при отладке можно положиться на эту процедуру и искать ошибку в других местах программы. Тщательная проверка подразумевает проверку, в том числе, и на граничные условия, в данном случае – это добавление элемента в пустую очередь, добавление максимально возможного количества элементов (если очередь имеет ограничения) и извлечение элемента из пустой очереди.

Разбиение задачи на части, помимо прочего, обеспечивает психологический комфорт, поскольку в каждый момент времени необходимо решить только небольшой, относительно простой участок задачи со строго заданными условиями, и не надо держать всю задачу, со всеми тонкостями и нюансами, в голове. Умение планировать работу, разделять сложную задачу на составные части, является одним из важнейших умений профессионального программиста.

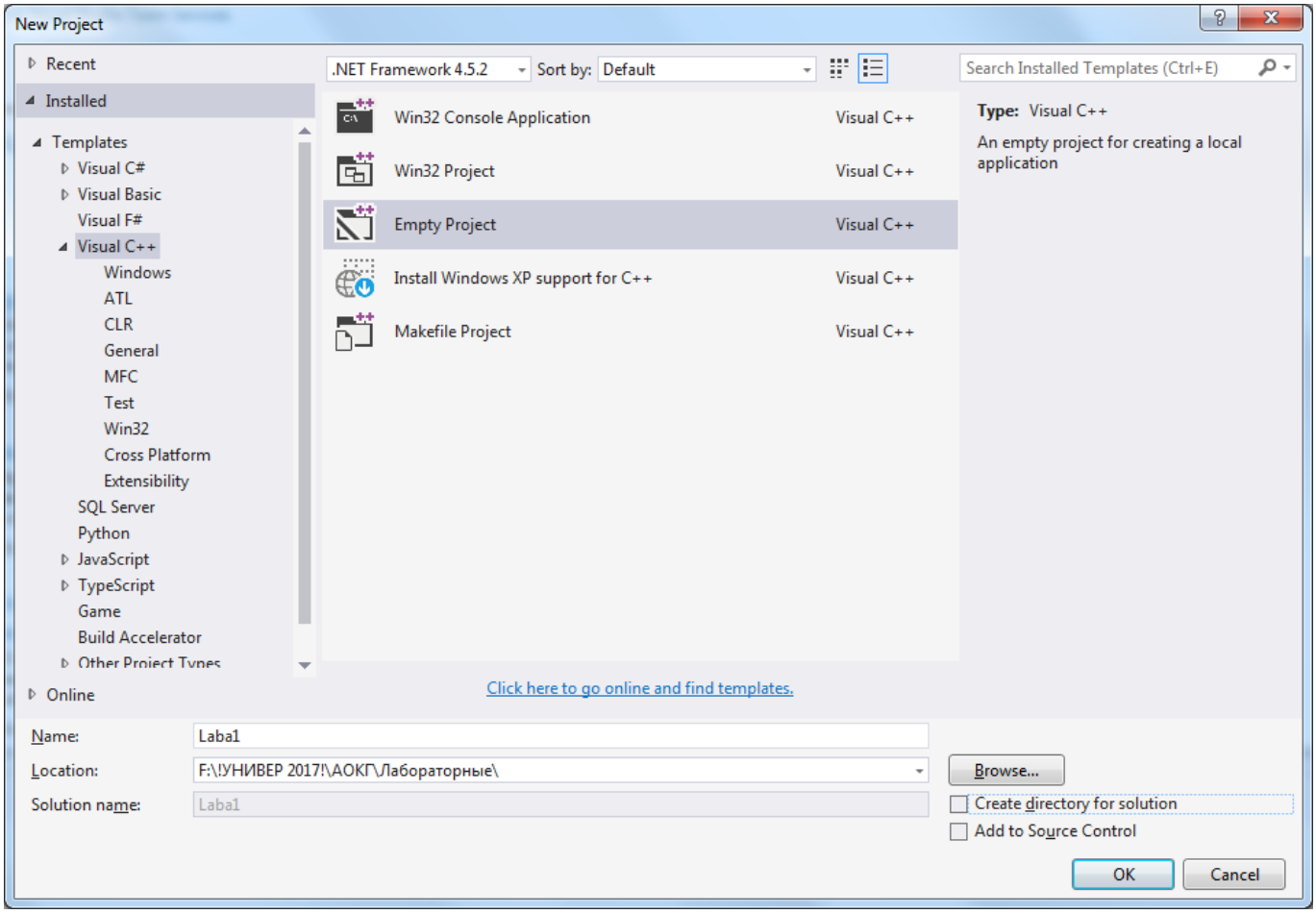
В данной серии лабораторных работ в каждой лабораторной работе будет предлагаться рекомендованный порядок выполнения лабораторной работы. В частности, данную лабораторную работу рекомендуется выполнять в следующей последовательности:

1. Создание пустого проекта;
2. Добавление к проекту главного файла с функцией main;
3. Подключение внешних библиотек (OpenGL и freeglut);
4. Изучение библиотеки freeglut;
5. Инициализация freeglut, создание окна, указание функций обратного вызова;
6. Написание функции Simulation (изменение параметров сцены);
7. Написание функции Reshape (измениние размеров окна);
8. Написание функции Display (перерисовка содержимого окна);
9. Написание функции KeyboardFunc (реакция на нажатие клавиши);
10. Реализация задания к лабораторной работе;
11. Написание отчета.

Создание проекта в среде разработки Microsoft Visual Studio 2015.

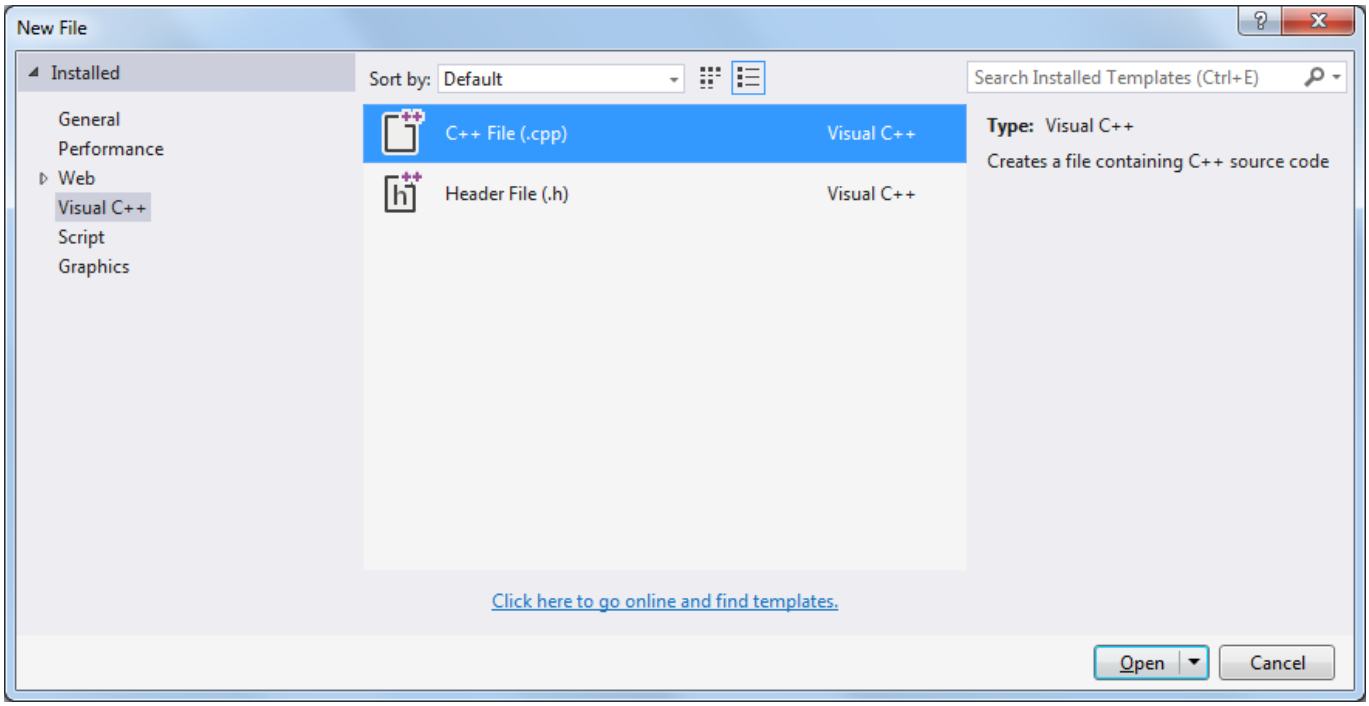
Прежде всего, необходимо разобраться с интегрированной средой разработки, если такого опыта еще нет. В данном случае обязательным является использование Microsoft Visual Studio. В этом разделе рассматривается создание проекта в интегрированной среде разработки Microsoft Visual Studio 2015.

Для большей наглядности создадим пустой проект. Для этого после загрузки программы необходимо в меню выбрать ***File>New>Project*** и ввести имя проекта и расположение файла проекта. При этом среда разработки самостоятельно создаст необходимый каталог:



После создания пустого проекта, необходимо добавить к нему хотя бы один файл с исходным кодом, который будет содержать функцию main, для того, чтобы проект можно было скомпилировать и получить работающую программу.

Для создания нового файла необходимо выбрать пункт меню ***File>New>File***, после чего, в появившемся окне необходимо выбрать тип создаваемого файла. В данном случае - это cpp файл. Содержимое окна приведено ниже:



Созданный файл пока не имеет ни имени, ни содержимого. Для начала введем небольшую демонстрационную программу, содержащую метод main, выводящую на экран стандартную фразу «Hello World!». Код программы приведен ниже:

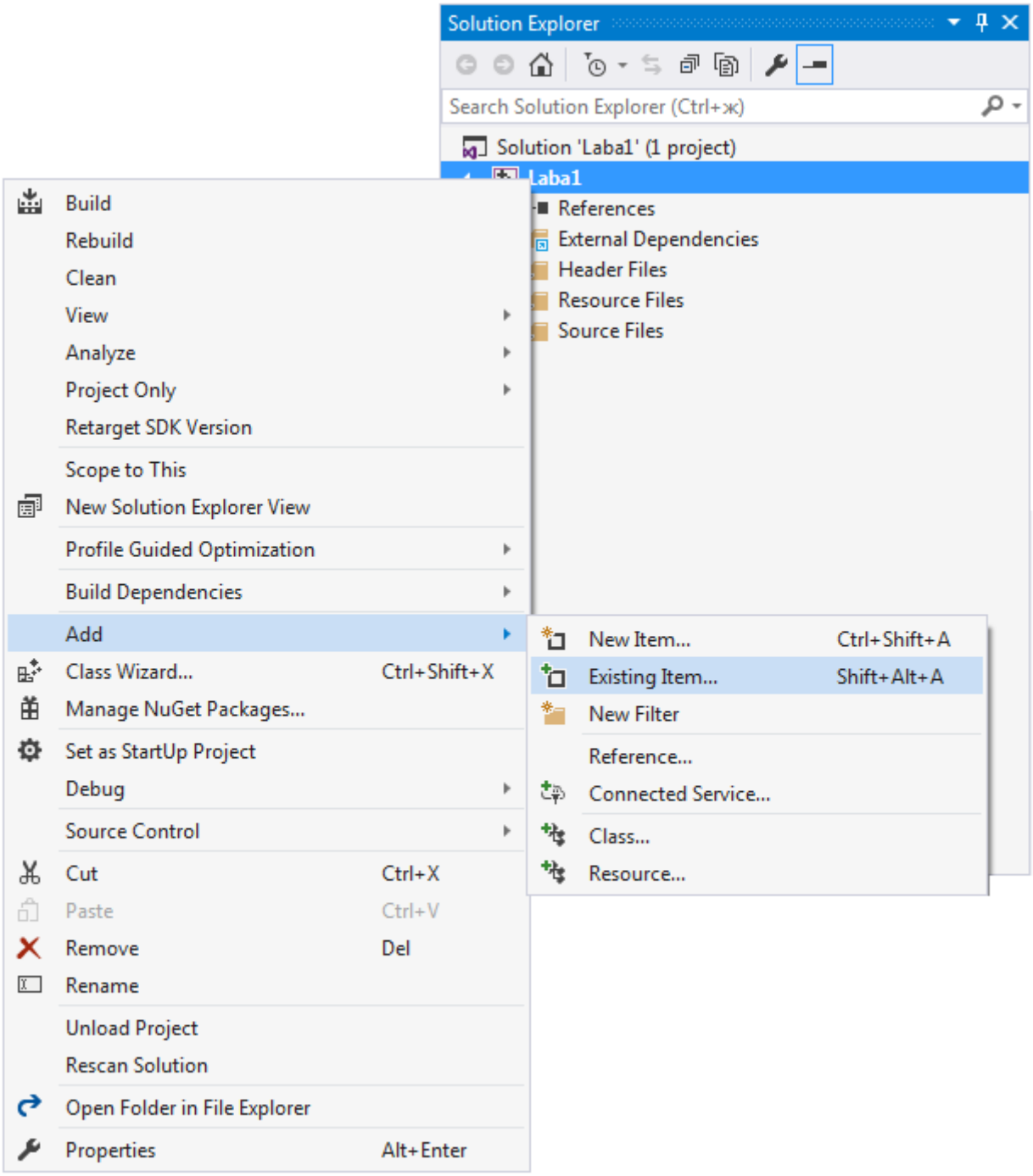
```
#include <stdio.h>
#include <windows.h>

void main(void) {
	printf("Hello World!");
	Sleep(2500);
}
```

Далее необходимо сохранить этот файл. При сохранении важно указать его месторасположение, так, чтобы он лежат в папке, которая была ранее создана для этого проекта. Имя файла можно указать любое, но, желательно, чтобы оно указывало на то, что именно этот файл является основным и содержит функцию main.

Следует отметить, что созданный файл, даже если он находится в той же директории на жестком диске, что и проект, пока не имеет к проекту никакого отношения. Для того чтобы указать, что этот файл относится к проекту, а значит при компиляции проекта в том числе будет компилироваться и указанный файл, его необходимо присоединить к проекту.

Для присоединения файла к проекту необходимо в окне обзора проекта (Solution Explorer) правой кнопкой мыши вызывать контекстное меню и выбрать ***Add>Existing Item***, после чего указать какой именно файл мы собираемся добавлять к проекту:

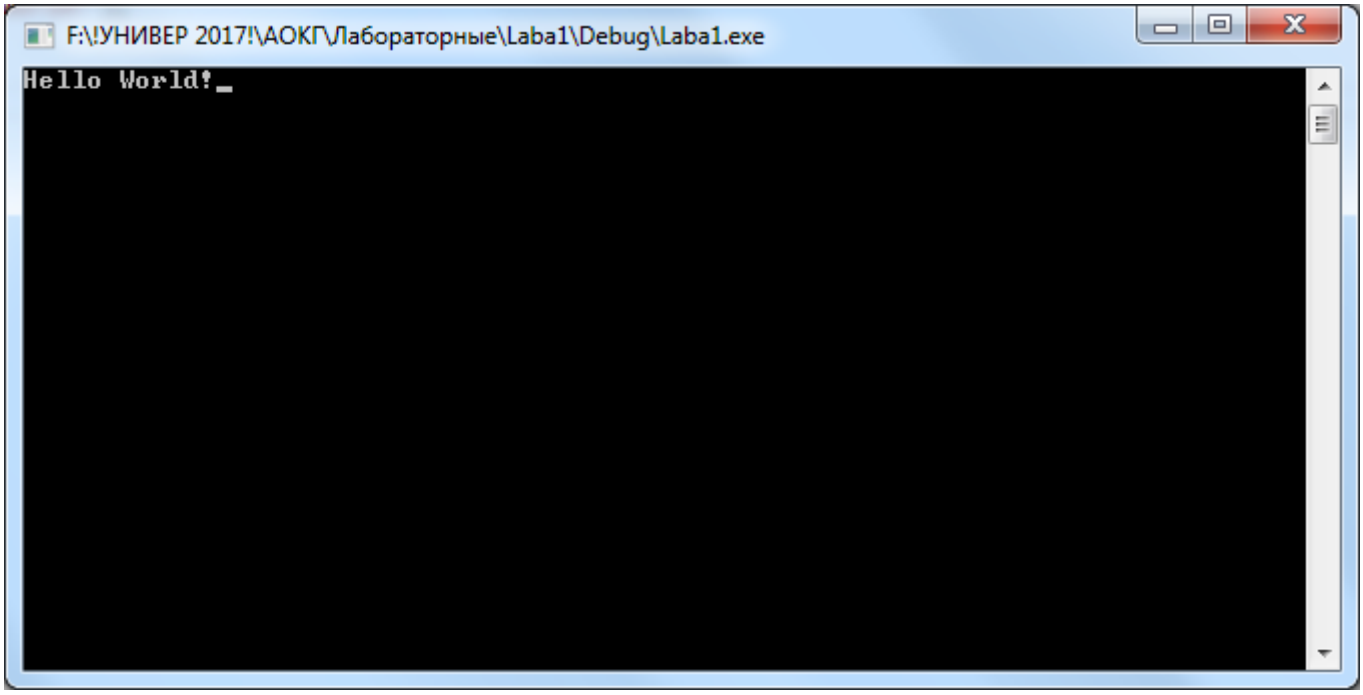


Окно Solution Explorer показывает все файлы, являющиеся частью проекта. Файлы, относящиеся к проекту, делятся на три группы:

1) Заголовочные файлы – файлы с расширением .h;
2) Файлы ресурсов;
3) Исполняемые файлы – файлы с расширением .c или .cpp;

После добавления файла к проекту он появится в соответствующей группе. В данном случае это группа Source Files. При нажатии на один из файлов в окне Solution Explorer он открывается в редакторе, после чего его можно просмотреть или изменить.

В завершении, после того как файл с функцией main был добавлен к проекту, необходимо скомпилировать проект и убедится что написанная нами программа работает. Выполняется это через главное меню, панель быстрого доступа или с использованием клавиши F5:



Для того, чтобы окно не закрывалось сразу по завершению программы, используется системная функция Sleep. Для использования этой функции требуется подключение бибилиотеки <Windows.h>.

Подключение библиотек OpenGL и freeglut.

Многие программы требуют подключения дополнительных внешних библиотек. Часть из них (например, стандартная библиотека Си) поставляется с каждым компилятором, другие необходимо устанавливать самостоятельно. В общем случае, библиотеки делятся на три части:

- Заголовочный файл (*.h) - нужен для того, чтобы компилятор мог распознать идентификаторы которые определены в библиотеке (например имена функций или типов данных). Указание заголовочных файлов позволяет компилятору проверить корректность синтаксиса.
- Файл реализации/объектный файл (*.lib) – это файл в котором непосредственно описаны реализации всех функций, определенных в заголовочном файле. На этапе линковки функции описанные в lib-файлах присоединяются к создаваемому исполняемому файлу.
- Файл динамической реализации (*.dll). В случае, если библиотека может использоваться несколькими программами одновременно, выгоднее поместить код функций из этой билиотеки в отдельный файл, нежели хранить эти функции в каждой программе. В этом случае исполняемый файл будет занимать гораздо меньше памяти, но для своего выполнения будет требовать динамическую библиотеку, то есть dll-файл.

При подключении новой библиотеки к проекту, прежде всего, необходимо подключить заголовочные файл (файлы с расширение *.h) в которых описываются все константы и функции, принадлежащие этому модулю. В первой лабораторной работе будут использоваться следующие библиотеки:

1. Прежде всего, для многих Windows-приложений необходимо подключить библиотеку <windows.h> которая необходима для использования функций WinAPI:

   ```
   #include <windows.h>
   ```

2. Для использования библиотеки OpenGL необходимо подключить заголовочные файлы этой библиотеки:

   ```
   #include <GL/gl.h>
   #include <GL/glu.h>
   ```

   Библиотека OpenGL поставляется со средой разработки Microsoft Visual Studio в качестве стандартной библиотеки, поэтому нет необходимости устанавливать ее отдельно. Однако следует отметить, что с тех пор, как в 2003 году корпорация Microsoft перестала принимать участие в развитие OpenGL, заголовочные файлы не обновляются. Поэтому в данной библиотеке по-прежнему используется версия OpenGL 1.1. Более современные версии можно использовать при помощи механизма расширений, о чем будет сказано в одной из следующих лабораторных работ.

3. В этой лабораторной работе так же планируется использовать библиотеку freeglut (более совершенный аналог библиотеки GLUT), поэтому необходимо подключить и её:

   ```
   #include "GL/freeglut.h"
   ```

   Данную бибилотеку необходимо установить самостоятельно, скопировав ее из папки преподавателя с сервера кафедры или с официальной страницы проекта http://freeglut.sourceforge.net.

При этом подключение последнего файла, а именно «freeglut.h», приводит к следующей ошибке времени компиляции:

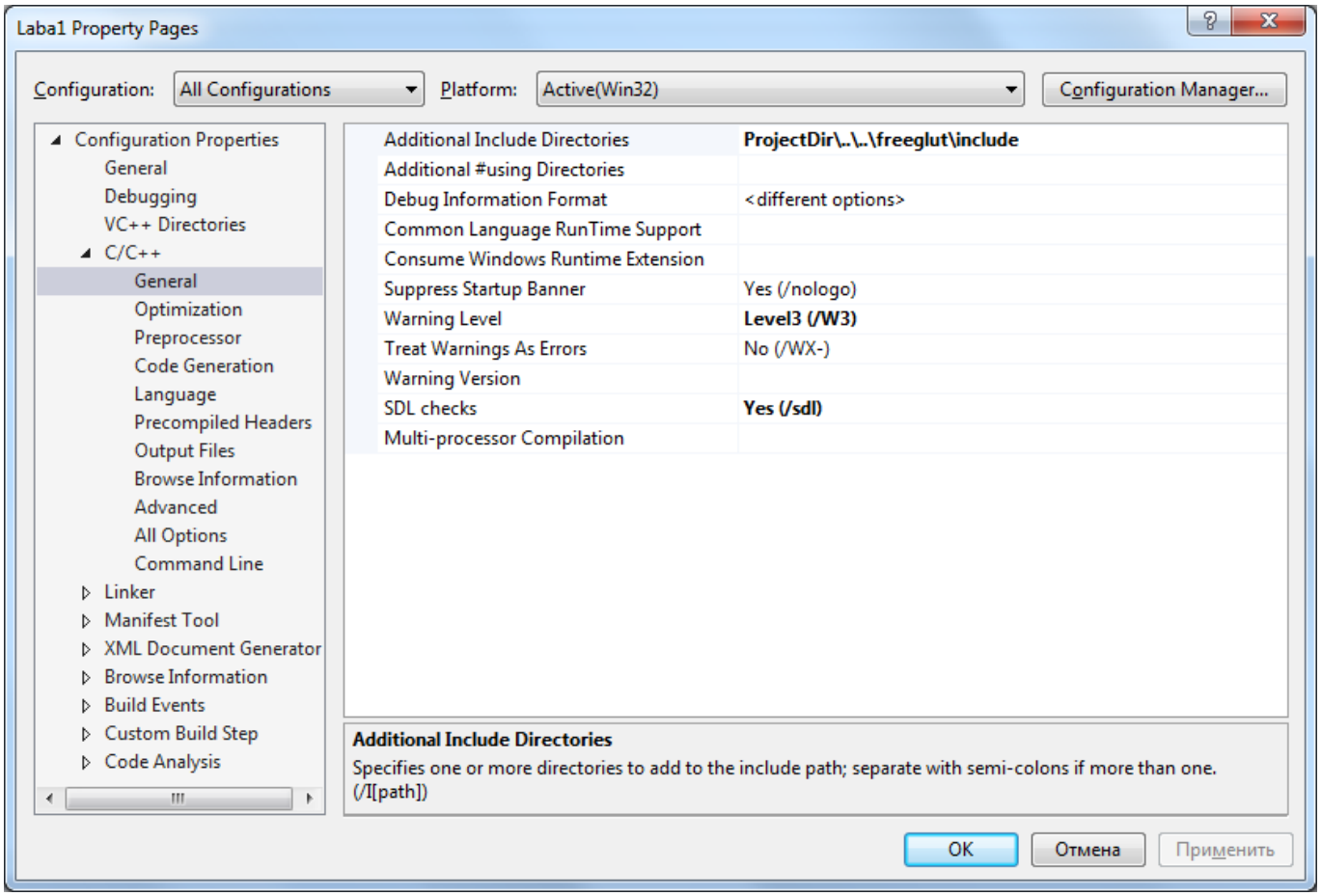
Severity	Code	Description	Project	File	Line	Suppression State
Error	C1083	Cannot open include file: 'GL/freeglut.h': No such file or directory	Laba1	f:\!универ 2017!\аокг\лабораторные\laba1\main.cpp	5	
Error (active)		cannot open source file "GL/freeglut.h"	Laba1	f:\!УНИВЕР 2017!\АОКГ\Лабораторные\Laba1\Main.cpp	5	

Данная ошибка сообщает о том, что компилятор не может найти указанный заголовочный файл, поскольку он не находится в стандартной папке подключаемых модулей. Чтобы компилятор знал, где искать подключаемые заголовочные файлы, ему необходимо указать папки, в которых могут лежать эти файлы. В данном случае необходимо распаковать библиотеку freeglut и указать компилятору путь к папке «freeglut\include». Делается это через настройки проекта. Следует отметить, что по умолчанию Visual Studio создает целых две конфигурации:

1. Версия для отладки (Debug) - в данном режиме к исходному коду добавляется дополнительная информация, которая позволяет отлаживать программу, например, устанавливать точки останова или просматривать содержимое памяти. В этом режиме, однако, программа не будет запускаться на компьютерах, на которых не установлен Visual Studio. Кроме того, программа не оптимизируется с точки зрения занимаемой памяти и скорости выполнения;
2. Финальная версия (Release) - в этом случае программа должна запускаться на любых компьютерах и будет оптимизированной с точки зрения как занимаемой памяти, так и скорости выполнения.

Настройки компиляции, в частности указание путей, используемых для поиска подключаемых заголовочных файлов, задается отдельно для каждой конфигурации. Для этого в окне обозревателя необходимо правой кнопкой мыши выбрать проект и перейти к его свойствам (Properties).В верхней части окна указывается, для какой именно конфигурации задаются свойства. Поскольку данный параметр является одинаковым как для ***debug***, так и для ***release*** версии то необходимо в выпадающем списке выбрать пункт «All Configurations.

Для указания путей, используемых для поиска подключаемых заголовочных файлов, используется пункт конфигурации Additional Include Directories, приведенный на рисунке ниже:



При этом можно задавать как абсолютные пути, так и пути, относительно текущего проекта. Последнее предпочтительно, так как при переносе проекта с одного компьютера на другой, не придется заново настраивать проект. Для этого существуют различные макросы, например:

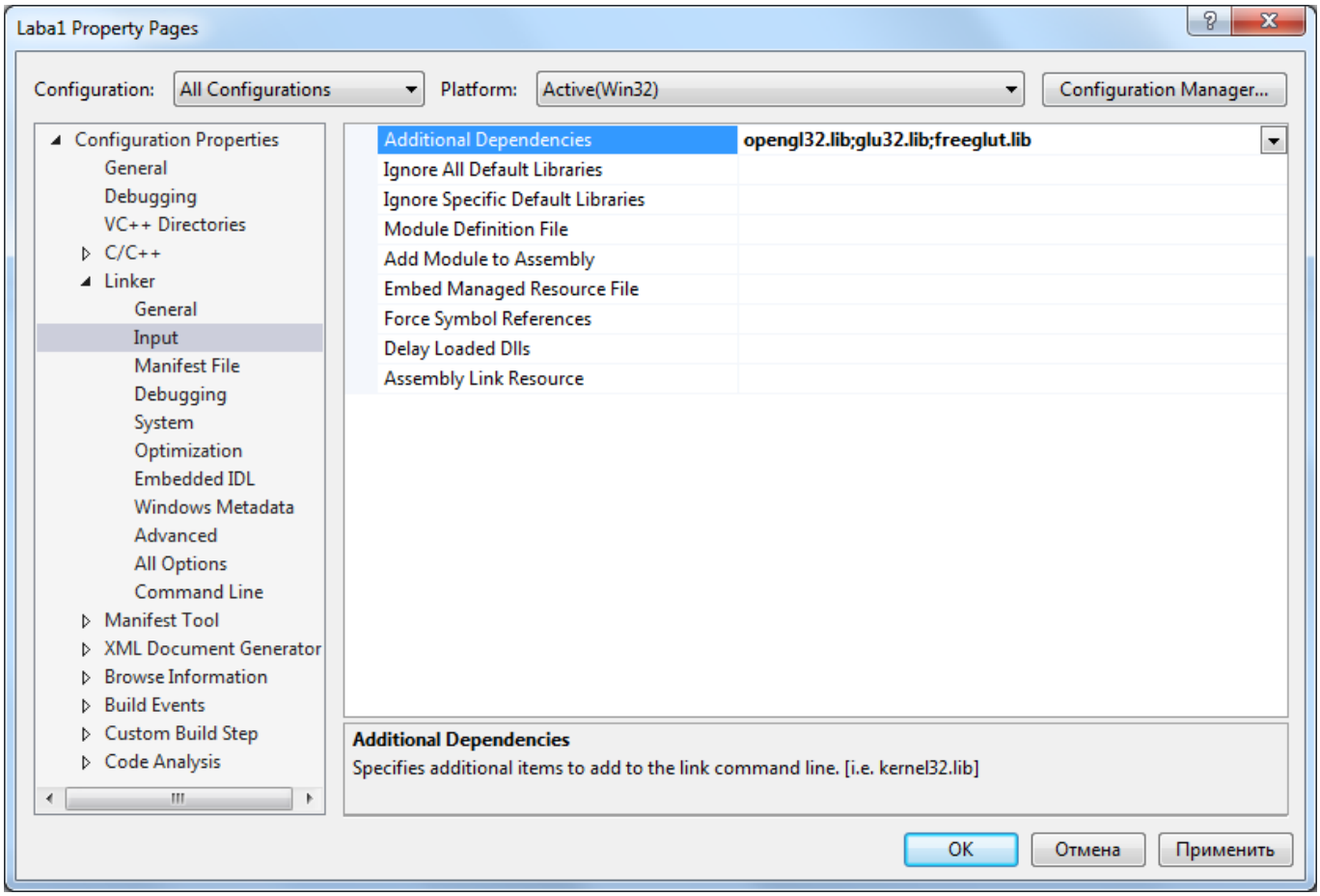
ProjectDir\.. – *искать в папке с проектом*
ProjectDir\..\.. – *искать в папке одним уровнем выше, чем папка с проектом*

Включение заголовочных файлов (*.h) позволяет использовать определенные в них константы, переменные и функции. При этом там содержаться только определения функций, а их реализации содержаться в файлах библиотек (*.lib). После подключения заголовочного файла и попытки скомпилировать проект, в котором используются функции из этих библиотек, может возникнуть ошибка, говорящая о том, что не удается найти реализацию соответствующей функции. Это происходит оттого, что необходимо указать в каком lib файле лежат соответствующие функции, а так же где лежат сами указанные lib файлы.

В данной лабораторной работе используется 3 библиотеки, для которых необходимо подключить следующие файлы:

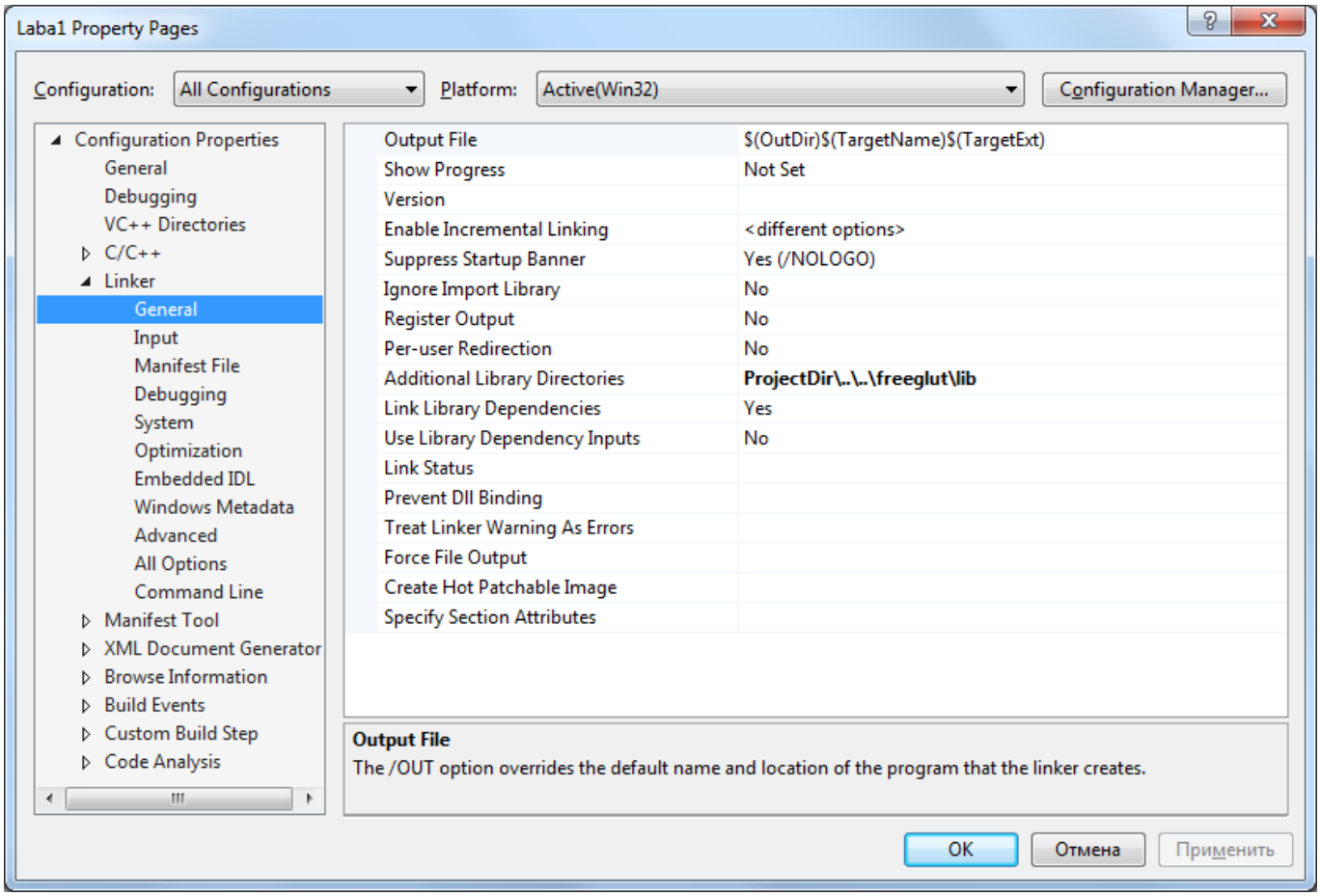
Используемый модуль	Заголовочный файл (*.h)	Файл библиотеки (*.lib)
freeglut	#include "freeglut.h"	freeglut.lib
GL	#include <GL/gl.h>	opengl32.lib
GLU	#include <GL/glu.h>	glu32.lib

Подключение lib файлов выполняется в два этапа. Прежде всего, необходимо указать, какие именно lib файлы требуется подключить. Выполняется это так же в свойствах проекта (в данном случае для всех конфигураций):

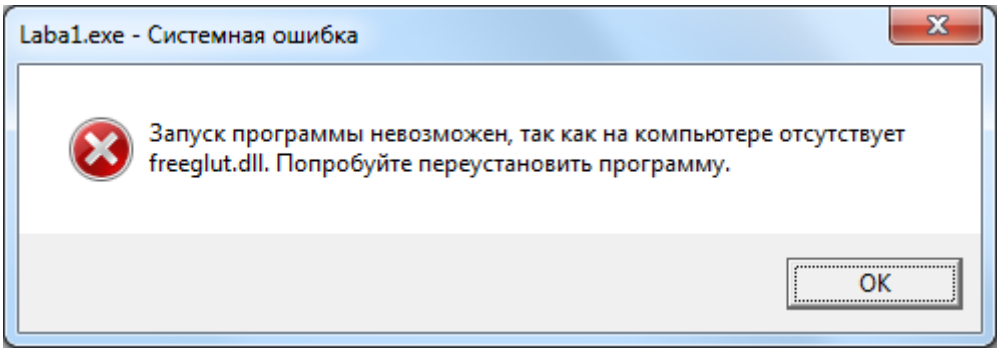


Далее необходимо указать каталоги, в которых лежат эти дополнительные lib файлы. При этом существует дополнительная тонкость, связанная с тем, что многие библиотеки используют различные файлы в зависимости от разрядности приложения (32 или 64-битные приложения). В частности, на рисунке, приведенном на следующей странице, в верхней части окна указанно, что мы создаем 32-х битную версию приложения (Win32), поэтому необходимо указать путь именно к 32-х битной версии lib файла. Для библиотеки freeglut 32-х битная версия лежит в каталоге «freeglut\lib». Здесь же, при необходимости, можно найти 64-битную версию библиотеки, расположенной в каталоге «freeglut\lib\x64».

При указании расположения lib файлов так же лучше использовать относительные пути. Файлы библиотек gl и glu лежат в стандартных папках VisualStudio, поэтом они видны всегда. Осталось указать путь для библиотеки freeglut:



Последнее, на что следует обратить внимание, это наличие динамически подключаемых библиотек (dll) в пределах доступности исполняемого файла (exe). В случае если при запуске программы необходимый dll файл не найден, возникает следующая ошибка:



Для того, чтобы избежать этой проблемы можно либо хранить dll-файл в той же директории что и исполняемый файл, либо записать его в одну из папок, в которых операционная система ищет динамические библиотеки по умолчанию, например в папку WINDOWS\SYSTEM32. Сам dll файл для библиотеки freeglut лежит в каталоге «freeglut\bin» для 32-битных или «freeglut\bin\x64» для 64-битных приложений.

Использование библиотеки GLUT.

Библиотека OpenGL, рассматриваемая в данном курсе лекций, является кроссплатформенной, то есть её можно использовать с различными языками программирования и операционными системами. Кроме того, библиотека OpenGL предназначена для решения задач, связанных только с трехмерной графикой, поэтому в самой библиотеке отсутствуют функции, относящиеся к тем задачам, которые невозможно решить без участия операционной системы. К таким задачам относятся: создание окна, инициализация контекста рендеринга или смена переднего и заднего буфера цвета. Эти задачи решаются дополнительными библиотеками, реализующими именно этот функционал. Например, для платформы Windows это дополнительная библиотека wgl, а для UNIX-систем – glx.

В процессе программирования приложений интерактивной компьютерной графики часто бывает необходимым решить и другие задачи, связанные со взаимодействием с операционной системой, такие как: получение кода нажатой пользователем клавиши, вызов функции через определенный промежуток времени, определение момента изменения размеров окна и так далее. Для решения вышеописанных задач часто используется библиотека GLUT (или более современная версия – freeglut).

OpenGL Utility Toolkit (GLUT) — библиотека утилит для приложений, использующих OpenGL, которая позволяет скрыть сложности взаимодействия с операционной системой и сосредоточится на работе с OpenGL. Библиотека GLUT, в частности, позволяет создать окна, обрабатывать сообщения операционной системы и предоставляет некоторые дополнительные функции, базирующиеся на OpenGL.

GLUT был создан Марком Килгардом (Mark Kilgard), во время его работы в Silicon Graphics Inc., однако библиотека давно не обновлялась, в связи с чем в настоящее время используются более современные библиотеки, например - freeglut. С целью обратной совместимости функции freeglut совпадают по названию и синтаксису с соответствующими функциями glut и имеют тот же префикс - glut.

Прежде чем приступать к программированию, необходимо изучить все функции, которые будут использоваться в данной лабораторной работе. В частности, здесь будут использоваться следующие функции:

Инициализация самой библиотеки и установка параметров дисплея:

glutInit
glutInitDisplayMode

Создание окна:

glutInitWindowPosition
glutInitWindowSize
glutCreateWindow
glutPostRedisplay

Задание адресов функций «обатного вызова» (callback functions):

Функции обратного вызова – это функции которые не вызываются самим программистом непосредственно, а передаются в качестве параеметров другим функциям для вызова в будущем. Применительно к интерактивным приложениям – это функции которые будут вызваны в ответ на какое-либо сообщение операционной системы. Например, можно указать, какую функцию необходимо будет вызвать в случае если пользователь изменит размеры окна. Для регистрации функций обратного вызова используются следующие функции glut:

glutDisplayFunc
glutReshapeFunc
glutKeyboardFunc
glutTimerFunc
и некоторые другие ...

Запуск «основного цикла программы»:

Это главная функция библиотеки glut. После выполнения всех необходимых действий по инициализации вызывается данная функция. Эта функция выполняет бесконечный цикл, в котором непрерывно проверяются сообщения от операционной системы и вызываются необходимые функции обратного вызова. Функции обратного вызова выполняют необходимые действия и возвращают управление обратно этой функции:

glutMainLoop

Смена переднего и заднего буфера:

glutSwapBuffers

Описание всех необходимых функций следует посмотреть в спецификации, на официальном сайте OpenGL или в других источниках. Краткое описание всех функций, а так же назначение параметров необходимо занести в таблицу отчета.

https://www.opengl.org/resources/libraries/glut/spec3/node1.html

Инициализация GLUT и создание окна.

Прежде всего, необходимо инициализировать саму библиотеку GLUT и установить параметры контекста рендеринга:

```
// инициализация библиотеки GLUT
glutInit(&argc, argv);
// инициализация дисплея (формат вывода)
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH | GLUT_MULTISAMPLE);
```

Первая функция инициализирует саму библиотеку GLUT. При этом ей передаются параметры командной строки, на тот случай, если в командной строке были заданы опции GLUT. Вторая функция инициализирует контекст рендеринга OpenGL. При этом параметры функции указывают, как именно инициализируется контекст рендеринга. В данном случае:

GLUT_RGBA	Данный параметр указывает, что цвет представляется в виде четверки чисел (red, green, blue, alpha).
GLUT_DOUBLE	Данный параметр указывает на то, что используется режим двойной буферизации для вывода на экран. Данный режим позволяет устранить мигание экрана в процессе построения изображения. Во всех лабораторных работах всегда будет использоваться режим двойной буферизации.
GLUT_DEPTH	Буфер глубины используется для корректного вывода заслоненных объектов и необходим для того, чтобы объекты, лежащие ближе к наблюдателю, не заслонялись дальними объектами независимо от порядка их вывода. Подробно использование буфера глубины будет рассмотрено в одной из следующих лабораторных работах.
GLUT_MULTISAMPLE	Этот параметр указывает, что для каждого пикселя экрана так же содержится дополнительная информация, необходимая для работы алгоритмов устранения ступенчатости (anti-aliasing). Использование алгоритмов будет рассмотрено в последующих лабораторных работах.

После инициализации GLUT, можно приступать к созданию окна. Для этого используются следующие функции, назначение и параметры которых довольно очевидны:

```
// создание окна:
// 1. устанавливаем верхний левый угол окна
glutInitWindowPosition(200, 200);
// 2. устанавливаем размер окна
glutInitWindowSize(600, 600);
// 3. создаем окно
glutCreateWindow("laba1");
```

Следует отметить, что все функции OpenGL выполняются в рамках своего собственного контекста выполнения. Поэтому перед вызовом любой функции OpenGL необходимо этот контекст создать. Иными словами, до того, как будет инициализирована библиотека OpenGL и создано окно, вызов функций OpenGL будет некорректным.

После того, как окно создано, необходимо зарагестрировать функции обратного вызова (callback functions):

```
// устанавливаем функцию, которая будет вызываться для перерисовки окна
glutDisplayFunc(Display);
// устанавливаем функцию, которая будет вызываться при изменении размеров окна
glutReshapeFunc(Reshape);
// устанавливаем функцию которая будет вызвана через 20 мс
glutTimerFunc(20, Simulation, 0);
// устанавливаем функцию, которая будет вызываться при нажатии на клавишу
glutKeyboardFunc(KeyboardFunc);
```

Первая функция определяет, какая функция будет вызываться всякий раз, когда требуется перерисовать окно. К таким ситуациям относятся, например, изменение положения окна или его размеров. Данная функция, в том числе, вызывается и тогда, когда окно только что было создано и, следовательно, его необходимо вывести на экран. В любой из этих ситуаций операционная система генерирует сообщение для приложения, библиотека GLUT обрабатывает это сообщение и вызывает функцию, которая была указана ранее. В данном случае это функция Display. Естественно, данная функция должны быть определена программистом.

Вторая функция определяет функцию, которая будет вызвана, когда пользователь изменит размеры окна. В данном случае это функция Reshape, которая должна переопределить порт просмотра и матрицу проекции, о чем будет сказано в последующих лабораторных работах.

Далее идет функция, которая указывает, что через определенное время (20 мс), необходимо вызвать функцию Simulation. По сути, запускается таймер, который сработает через 20 мс и запустит указанную функцию. В функции Simulation можно еще раз запустить таймер на то же самое время. Таким образом, можно добиться того, чтобы функция Simulation вызывалась каждые 20 мс (то есть 50 раз в секунду) и организовать в ней плавное изменение каких либо параметров, например автоматическую смену цветов чайника через заданный промежуток времени.

Последняя функция определяет функцию, которая будет обрабатывать нажатие клавиш. При нажатии клавиши будет вызвана указанная функция в которую в качестве параметров передаются код нажатой клавиши и координаты мышки (x, y) в момент нажатия.

Данные функции (glutDisplayFunc, glutReshapeFunc, glutTimerFunc и glutKeyboardFunc) просто указывают функции, которые будут вызваны при наступлении определенных событий. Они не вызывают данные функции напрямую. Кроме того, пока сообщения операционной системы не обрабатываются. Следует также иметь в виду, что существуют и другие функции обратного вызова, например, обработка движения мышки, обработка завершения программы и прочее.

Запуск основного цикла обработки сообщений операционной системы:

```
// основной цикл обработки сообщений ОС
glutMainLoop();
```

Внутри данной функции реализован бесконечный цикл, который получает все сообщения от операционной системы и вызывает функции обратного вызова. Из данной функции программа уже не выходит, оставаясь в ней, пока пользователь не закроет окно. Поэтому всю прочую инициализацию необходимо выполнить до вызова этой функции.

На этом заканчивается инициализация приложения. Осталось написать четыре функции, реализующие необходимую функциональность:

Simulation - изменение параметров сцены.

Reshape - измениние размеров окна

Display - перерисовка содержимого окна.

KeyboardFunc – обработка нажатия клавиши.

Функция Simulation.

Данная функция будет вызываться каждые 20 мс, то есть 50 раз в секунду. Это очень удобно, если требуется организовать плавное изменение какого-либо параметра. Например, если требуется изменить цвет чайника через 1 секунду, можно организовать счетчик, который будет увеличиваться на единицу при каждом вызове данной функции. Как только счетчик достигнет 50, можно считать, что прошло 50 промежутков по 20 мс, то есть ровно одна секунда.

В приведенном примере функция не выполняет практически никаких действий, но в реальном приложении здесь может быть расчет физики (падение или движение объектов под действием различных сил) или искусственного интеллекта. В данной лабораторной работе необходимо заново перезапустить таймер (поскольку он приводит к вызову функции только один раз) и генерировать принудительную перерисовку окна. Для этого используется функция GLUT – glutPostRedisplay, которая отмечает, что окно нуждается в перерисовке. Следовательно, в главном цикле программы (внутри функции glutMainLoop) будет вызвана необходимая функция обратного вызова, отвечающая за перерисовку окна.
Полный текст функции с комментариями приведен ниже:

```
// функция вызывается каждые 20 мс
void Simulation(int value)
{
	// устанавливаем признак того, что окно нуждается в перерисовке
	glutPostRedisplay();
	// эта же функция будет вызвана еще раз через 20 мс
	glutTimerFunc(20, Simulation, 0);
};
```

Функции Reshape.

Функция Reshape вызывается при изменении размеров окна, в том числе и при создании окна. Функция принимает два параметра - новую ширину и высоту окна. Функция, как правило, устанавливает новый порт просмотра (т.е. область, в которую будет выводиться изображение). Обычно изображение выводится во всё окно, поэтому размеры области просмотра равны размерам нового окна. Кроме того, данная функция должна установить новую матрицу проекции, которая отвечает за преобразование трехмерных координат в двухмерные координаты на экране. Важно устанавливать аспект матрицы проекции (отношение ширины к высоте) таким же, как и аспект порта просмотра. В этом случае все объекты сохранят свои пропорции. Например, круглые объекты будут казаться круглыми, а не сплющатся до эллипса. Подробнее про порт просмотра и матрицу проекции будет рассказано в следующих лабораторных работах. Полный текст функции с комментариями приведен ниже:

```
// функция, вызываемая при изменении размеров окна
void Reshape(int w, int h)
{
	// установить новую область просмотра, равную всей области окна
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);

	// установить матрицу проекции с правильным аспектом
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(25.0, (float)w/h, 0.2, 70.0);
};
```

Функция Display.

Данная функция занимается непосредственно выводом всей сцены на экран. ***Необходимо сделать важное замечание!*** Функция Display должна решать именно эту задачу – то есть вывод объектов на экран, используя глобальные данные, например: количество объектов, их цвет, их расположение и так далее. При этом изменением этих параметров должна заниматься другая функция (например, Simulation). Иными словами, в соответствии с первым принципом SOLID (single responsibility), необходимо четко разделить обязанности в программе. Одна функция обрабатывает данные (Simulation), другая функция занимается их выводом на экран (Display). Функция вывода трехмерной графики на экран должна состоять из следующих основных шагов:

1. Очистка буфера кадра (заднего буфера цвета и буфера глубины);
2. Определение параметров вывода, в том числе и позиция камеры, которая указывает, с какой точки наблюдатель смотрит на сцену;
3. Вывод всех объектов, при этом для каждого объекта указываются его параметры, например, цвет;
4. И, наконец, смена переднего и заднего буфера цвета.

Полный текст функции представленн ниже:

```
// функция вызывается при перерисовке окна
// в том числе и принудительно, по командам glutPostRedisplay
void Display(void)
{
	// отчищаем буфер цвета
	glClearColor(0.22, 0.88, 0.11, 1.0);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	// включаем тест глубины
	glEnable(GL_DEPTH_TEST);

	// устанавливаем камеру
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt(5, 5, 7.5, 0, 0, 0, 0, 1, 0);

	// выводим объект - красный (1,0,0) чайник
	glColor3f(1.0, 0.0, 0.0);
	glutWireTeapot(1.0);

	// смена переднего и заднего буферов
	glutSwapBuffers();
};
```

Функции KeyboardFunc.

Данная функция вызывается при нажатии пользователем клавиш клавиатуры и может использоваться для взаимодействия с программой. В приведенном примере функция просто выводит на консоль код нажатой клавиши:

```
// Функция обработки нажатия клавиш
void KeyboardFunc(unsigned char key, int x, int y)
{
	printf("Key code is %i\n", (unsigned int) key);
};
```

Задание к лабораторной работе.

В рамках данной лабораторной работы необходимо изучить функции GLUT для инициализации OpenGL и взаимодействия с операционной системой, разобраться с приведенным примером, а так же внести следующие изменения:

1. Используя функцию glutKeyboardFunc добавить функцию обработки события нажатия на клавишу;

2. Реализовать циклическое изменение цвета объекта по нажатию на клавишу, используя предварительно заданный массив цветов. В массиве цвета лежат в следующем порядке: черный, белый, синий и красный. Следует учесть, что во время защиты может потребоваться добавление новых цветов в массив;

3. Для защиты следует заранее подумать над тем, как организовать автоматическое изменение цветов с заданной скоростью;

Содержание отчета.

1. Титульный лист;

2. Задание к лабораторной работе (полный текст задания из данных методических указаний);

3. Таблица с описанием используемых функций, в которой необходимо кратко указать назначение функции и её параметров:

glutInit	
glutInitDisplayMode	
glutInitWindowPosition	
glutInitWindowSize	
glutCreateWindow	
glutPostRedisplay	
glutDisplayFunc	
glutReshapeFunc	
glutKeyboardFunc	
glutTimerFunc	
glutMainLoop	
glutSwapBuffers	
glutWireTeapot	

4. Текст программы с подробными комментариями.

5. Скриншот работы программы.

Критерии оценки и вопросы к защите.

Вопросы по теоретической части:

1.1 Что из себя представляет RGB модель представления цвета?
1.2 Растровый и векторный способ представления графики?
1.3 Что такое режим двойной буферизации?
1.4 Для чего необходим буфер глубины и тест глубины?
1.5 Что такое конвейер рендеринга?
1.6 Что такое контекст рендеринга?

Вопросы по практической части:

2.1 Как подключаются новые модули к проекту в среде Visual Studio 2015?
2.2 Что такое функции обратного вызова? Какие функции обратного вызова используются в вашей программе, и в каких случаях они вызываются?
2.3 Для чего необходима функция glutSwapBuffers?

Требования к программе:

3.1 Осмысленные имена переменных;
3.2 Отсутствие в программе «магических чисел», то есть непосредственно числовых констант, разбросанных по тексту программы. Например, количество цветов в массиве должно либо определяться автоматически, либо задаваться одной константой;
3.3 Комментарии ко всем переменным, функциям и крупным блокам кода;
3.4 Функция Display должна отвечать только за отрисовку изображения и не должна менять переменные, которые задают используемый для вывода чайника цвет.

GOOD CODERS...