СЕРГЕЙ ГАЙДУКОВ

bhv

OpenGL

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ ТРЕХМЕРНОЙ ГРАФИКИ НА C++

НОВЫЕ ВОЗМОЖНОСТИ OpenGL

СРЕДСТВА NVIDIA OpenGL SDK

ПРОГРАММИРОВАНИЕ ИГР

ЭКСПОРТ МОДЕЛЕЙ ИЗ 3ds max

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Сергей Гайдуков

OpenGL

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ ТРЕХМЕРНОЙ ГРАФИКИ НА C++

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1
Г12

Гайдуков С. А.

Г12 OpenGL. Профессиональное программирование трехмерной графики на C++. — СПб.: БХВ-Петербург, 2004. — 736 с.: ил.

ISBN 5-94157-363-4

Книга посвящена использованию новых возможностей графической библиотеки OpenGL версии выше 1.2 в приложениях, разрабатываемых на языке C++ в Microsoft Visual Studio .NET 2002. Описано применение средств NVIDIA OpenGL SDK для создания реалистичных трехмерных изображений. На примерах рассмотрены загрузка текстур из файлов форматов TGA и JPG, экспорт моделей из 3ds max, хранение данных в ZIP-архивах, отсечение невидимой геометрии, моделирование глянцевых объектов и др.

Прилагается компакт-диск с инструментальными средствами и демонстрационными версиями рассматриваемых примеров.

Для программистов

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Игорь Рыбинский*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.03.04.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 59,34.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в ФГУП ордена Трудового Красного Знамени "Техническая книга"
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций.
190005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-363-4

Содержание

Введение

Эта книга посвящена программированию компьютерной графики. Если точнее, она научит вас программировать с использованием расширений OpenGL в Microsoft Visual Studio.NET.

Базовая версия OpenGL 1.0 вышла в далеком 1992 году. С тех пор прошло более 10 лет. Это очень большой срок для компьютерной индустрии. Например, за это время операционные системы Windows прошли путь от 16-битной операционной оболочки Windows 3.11 до полноценной 32-битной операционной системы Windows XP. Тактовая частота процессоров Intel Pentium поднялась с 60 МГц до 3.2 ГГц, а на смену простеньким двумерным ускорителям ATI Mach32 корпорации ATI пришли "навороченные" ATI Radeon 9800 XT. По идее, за столь большой срок библиотека OpenGL должна была морально устареть, если бы не одно "но": создатели OpenGL заложили в нее очень важный принцип — расширяемость. Если производитель аппаратного обеспечения выпускал видеокарту, новые возможности которой не могли быть использованы в существующей версии OpenGL, то он мог выпустить свое расширение OpenGL. В результате, разработчики программного обеспечения могли использовать новые возможности этой видеокарты практически сразу после ее выхода, не дожидаясь выхода новой версии DirectX, как это случилось с видеокартой Radeon 9700 корпорации ATI. Другим классическим примером несбывшихся ожиданий могут служить печально известные пиксельные шейдеры GeForce2 GTS (GTS кстати расшифровывается, как Giga Texel Shader), когда компания Microsoft решила, что пиксельные шейдеры этого GPU таковыми не являются. В итоге, программа, использующая DirectX, не может задействовать все возможности GeForce2.

Любая следующая версия OpenGL отличается от предыдущей версии лишь списком обязательных расширений. Иными словами, OpenGL 1.4 можно описать, как OpenGL 1.0 + новые расширения версий 1.1, 1.2, 1.2.1, 1.3 и 1.4. Поэтому любая программа, написанная для ранних версий OpenGL, будет нормально работать с текущей версией OpenGL.

Однако использование расширений OpenGL связано с некоторыми проблемами. Самой большой трудностью является то, что компания Microsoft в настоящее время продвигает свой собственный API — DirectX. А поскольку OpenGL является прямым конкурентом DirectX, компания Microsoft не заинтересована в поддержке своими программными продуктами библиотеки OpenGL. В частности, в состав Microsoft Visual Studio .NET входят лишь за-

головочные файлы для OpenGL 1.1, причем сами файлы датируются 1996-м годом. С документацией дела обстоят аналогичным образом.

Из этого следует вывод, что для нормальной работы с расширениями OpenGL нам необходим дополнительный инструментарий. В качестве такого инструментария мы будем использовать NVIDIA SDK 5.21, который находится на CD-диске, сопровождающем книгу. В этот SDK входят NVIDIA Audio SDK, NVIDIA DirectX SDK и NVIDIA OpenGL SDK, причем, нас, разумеется, больше всего будет интересовать последний. NVIDIA OpenGL SDK содержит множество полезных вещей:

- набор заголовочных файлов для OpenGL версии 1.3;
- кроссплатформенную оконную библиотеку GLUT 3.7;
- библиотеку NVIDIA OpenGL Helper Library, которая фактически является объектно-ориентированной надстройкой над OpenGL и GLUT;
- математическую библиотеку NV_MATH;
- библиотеку утилит NV_UTIL, содержащую функции для работы с файлами форматов ZIP, ASE, JPG и TGA;
- множество других вспомогательных библиотек, облегчающих жизнь разработчику;
- множество примеров, демонстрирующих использование расширений OpenGL на практике;
- статьи и слайды докладов;
- многое другое.

Но у всего этого есть один недостаток: NVIDIA OpenGL SDK поставляется "как есть" — в исходных кодах и без какой-либо документации. Поэтому новичкам будет довольно сложно разобраться с этим продуктом.

Эту книгу я задумал как учебник, которого мне очень не хватало во время изучения NVIDIA OpenGL SDK и расширений OpenGL. Хотя книга является учебником и рассчитана на последовательное чтение, я решил разбить материал на тематические разделы. Книга не охватывает целиком ни NVIDIA OpenGL SDK, ни расширения OpenGL, т. к. эти темы очень объемные (одна только краткая спецификация расширений, поддерживаемых NVIDIA GeForce FX, занимает более 1000 страниц). Но, прочитав эту книгу, вы наверняка узнаете для себя очень много нового.

В этой книге основной упор делается на практические примеры. Из всех книг, которые я прочитал, наиболее полезными оказались те, которые содержали примеры и готовые решения. Книга содержит множество примеров, бóльшую часть которых я написал самостоятельно. При чтении этой книги постарайтесь просматривать эти примеры не очень бегло, иначе в какой-то момент времени вы рискуете потерять нить рассуждений.

На кого рассчитана эта книга

В первую очередь я адресую эту книгу студентам старших курсов и аспирантам первого года обучения, специализирующимся в области информатики и вычислительной техники. Книга будет полезна и профессионалам.

Книга рассчитана на читателей, имеющих опыт работы с Microsoft Visual C++ .NET (на уровне консольных приложений) и OpenGL 1.1. Очень желательно предварительно прочесть одну из книг, приведенных в списке использованной литературы ([1], [2], [3], [4], [5] или [6]). Кроме того, для понимания некоторых примеров понадобятся начальные знания Borland Delphi (на уровне первых 6-ти глав [7]).

Я принимаю нейтральную позицию в спорах на тему, что лучше: Visual C++ или Delphi. Каждый из этих языков имеет свои достоинства и недостатки, и программист просто должен уметь выбрать продукт, который лучше всего подходит для решения конкретной задачи. Visual C++ является идеальным инструментом для создания полноэкранных OpenGL-приложений, основным требованием к которым является быстродействие. Но в приложениях, где главную роль играет интерфейс, а вопросы производительности отходят на второй план, более рационально использовать Borland Delphi.

К книге прилагается CD-ROM, на котором вы найдете программное обеспечение, необходимое для работы с примерами из книги. Разумеется, вы не найдете на нем Microsoft Visual Studio .NET, Borland Delphi, 3D Studio MAX или какие-нибудь другие коммерческие продукты.

Структура книги

Книга состоит из двух частей. Первая часть посвящена вспомогательному инструментарию, без которого невозможно создать ни одну серьезную программу, использующую OpenGL, а вторая часть — собственно расширениям OpenGL.

Часть I. Использование NVIDIA OpenGL SDK

Главная задача первой части — познакомить читателя с основными компонентами NVIDIA OpenGL SDK и научить использовать их на практике.

Глава 1. Библиотека GLUT

Первая глава знакомит читателя с оконной библиотекой GLUT, позволяющей создавать кроссплатформенные OpenGL-приложения, не привязанные к определенной платформе. Даже если вы уже знакомы с этой библиотекой, я все равно советую вам бегло просмотреть эту главу, т. к. в ней приводятся

некоторые малоизвестные сведения о библиотеке GLUT, например, использование джойстика или создание полноэкранных приложений.

Глава 2. Библиотека GLH

Вторая глава посвящена одному из самых основных компонентов NVIDIA OpenGL SDK — библиотеке OpenGL Helper Library. Эта библиотека содержит множество классов, которые могут значительно облегчить жизнь программисту. Эти классы можно условно разделить на 3 большие группы:

- математические функции;
- объектно-ориентированная надстройка над GLUT, основанная на интеракторах;
- классы, инкапсулирующие функции OpenGL.

Кроме того, в этой главе рассматривается расширение библиотеки GLH — OpenGL Helper Library Extension (GLHE), добавляющее в библиотеку GLH ряд новых возможностей, в частности, интерактор консоли `glut_console` с поддержкой скриптов. Использование библиотеки GLHE позволяет значительно сократить размер демонстрационных программ: к примеру, программа, выводящая на экран цветной чайник, который пользователь может вращать и перемещать с использованием мыши и клавиатуры, менее 90 строк.

Глава 3. Библиотека NV_MATH

В третьей главе рассматривается математическая библиотека NV_MATH, являющаяся аналогом математической части библиотеки OpenGL Helper Library. Зачем она нужна? Если честно — не знаю. Скорее просто так исторически сложилось, что одни примеры из NVIDIA OpenGL SDK использовали библиотеку GLH_LINEAR, а другие — NV_MATH. В результате, для того, чтобы нормально ориентироваться в примерах NVIDIA OpenGL SDK, программисту очень желательно знать обе библиотеки. Поэтому если вы не собираетесь досконально разбирать примеры из состава NVIDIA OpenGL SDK, вы можете пропустить эту главу.

Глава 4. Библиотека NV_UTIL

В четвертой главе рассматривается библиотека NV_UTIL, которая содержит набор функций для работы с наиболее распространенными форматами файлов: TGA, JPG, ASE и ZIP. В принципе, этих форматов более чем достаточно для создания небольшой полноценной трехмерной игры. Кроме того, в этой главе рассматривается экспорт трехмерных сцен из 3D Studio MAX с использованием формата ASE при помощи библиотеки ASE Reader Library, являющейся объектной надстройкой над функциями NV_UTIL.

Часть II. Расширения OpenGL

Эта часть посвящена использованию новых расширений OpenGL версий 1.2 и выше, а также некоторых расширений, не вошедших в стандарт. Я решил не рассматривать расширения OpenGL версии 1.1, т. к. найти литературу на русском языке, посвященную OpenGL версий 1.0 и 1.1, не составит никакого труда.

Глава 5. Введение в расширения OpenGL

Пятая глава знакомит читателя с понятием расширения OpenGL и обучает использованию спецификации расширений. Кроме того, в этой главе рассматриваются семь относительно простых расширений OpenGL: EXT_separate_specular_color, ARB_window_pos, WGL_EXT_swap_control, SGIS_texture_lod, EXT_texture_lod_bias, EXT_texture_filter_anisotropic и SGIS_generate_mipmap. В заключение главы рассматривается инструментарий корпораций ATI и NVIDIA для быстрой инициализации расширений OpenGL: библиотеки ATI Extensions и NVIDIA OpenGL Helper Library.

Глава 6. Расширения EXT_texture_rectangle и NV_texture_rectangle

Шестая глава посвящена использованию текстур с разрешением, не кратным степени два.

Глава 7. Проверка видимости объектов с использованием расширений HP_occlusion_test и NV_occlusion_query

В седьмой главе рассматриваются расширения HP_occlusion_test и NV_occlusion_query, предназначенные для проверки видимости объектов на экране. Также рассматривается использование предварительной проверки видимости прямоугольных оболочек для отсечения невидимых объектов, что позволяет значительно повысить производительность приложения.

Глава 8. Использование внеэкранных буферов

Восьмая глава посвящена использованию внеэкранных буферов пикселов (pbuffer), представляющих собой специальную область видеопамяти, в которую программист может выводить изображение, как в обычное окно. При этом затрагиваются следующие расширения: WGL_ARB_pixel_format, WGL_ARB_pbuffer и ARB_render_texture. Кроме того, рассматривается практическое использование класса `PBuffer` из NVIDIA OpenGL SDK, значительно облегчающего работу с внеэкранными буферами pbuffer. В заключение главы создается мини-игра с видом от первого лица (FPS).

Глава 9. Сжатые текстуры

В девятой главе рассматривается использование сжатых текстур в OpenGL-программах, позволяющих сократить объем занимаемой видеопамяти, а также немного увеличить быстродействие программы. Кроме того, проводится исследование особенностей наиболее распространенного формата сжатия S3TC, а также его достоинства и недостатки. Примеры этой главы используют расширения ARB_texture_compression и EXT_texture_compression_s3tc. Кроме того, рассматриваются вопросы сохранения сжатых текстур на диске, а также использование формата DDS для хранения предварительно сжатых текстур с использованием формата S3TC.

Глава 10. Кубические текстурные карты

Десятая глава посвящена моделированию глянцевых объектов с использованием сферических и кубических текстурных карт. При этом подробно описываются все тонкости расширения ARB_texture_cube_map. Параллельно рассматриваются различные варианты хранения кубических текстур на диске, как с использованием классических графических форматов (TGA, JPG), так и с помощью формата DDS, позволяющего хранить в одном файле изображения всех граней кубической текстурной карты. Кроме того, затрагивается вопрос нетипичного использования кубических текстурных карт для моделирования закраски методом Фонга в реальном времени. В заключение рассматривается экспорт из 3D Studio MAX материалов, использующих текстурные карты отражения reflect/refract.

Требования к программному и аппаратному обеспечению

Для самостоятельного построения или модификации примеров программ, приведенных в книге, на вашем компьютере должно быть установлено следующее программное обеспечение.

- 32-разрядная операционная система семейства Windows, например Windows 2000 или XP.
- Microsoft Visual Studio .NET 2002.

Внимание

Часть примеров книги не компилируются с использованием Microsoft Visual Studio 6 или .NET 2003.

- Для сборки примеров, посвященных экспорту моделей из 3D Studio MAX, потребуется Discreet 3D Studio MAX 5.
- Для изменения разрешения текстур необходимо иметь Adobe Photoshop, ACDSee, IrfanView или аналогичную программу.

Остальное необходимое программное обеспечение находится на CD, сопровождающем книгу.

Для того чтобы на компьютере работали *все* примеры книги, вам понадобится видеокарта GeForce256 или выше. В табл. В1 приведены требования к видеокартам в зависимости от глав. Так как я в настоящее время располагаю только видеокартами корпораций ATI и NVIDIA, требования указаны только для видеокарт этих корпораций, хотя, скорее всего, бóльшая часть примеров будет работать и на видеокартах других фирм.

Таблица В1. *Требования к видеокартам по главам*

Глава	Минимально требуемая видеокарта
Глава 1	NVIDIA RivaTNT
Глава 2	NVIDIA RivaTNT
Глава 3	NVIDIA RivaTNT
Глава 4	NVIDIA RivaTNT2 32MB
Глава 5	NVIDIA GeForce256
Глава 6	NVIDIA GeForce256
Глава 7	NVIDIA GeForce256
Глава 8	NVIDIA GeForce256
Глава 9	NVIDIA GeForce256
Глава 10	NVIDIA GeForce256

К остальным компонентам компьютера не предъявляется каких-либо особенных требований. Это может быть, например, что-нибудь вроде Celeron 266MHz, 128MB RAM.

Все примеры этой книги проверялись на компьютере, конфигурация которого приведена в табл. В2. Часть "подозрительных" примеров тестировались на других компьютерах, на которых были установлены следующие видеокарты: NVIDIA GeForce4 Ti4600 128MB, NVIDIA GeForce2MX 32MB, NVIDIA RivaTNT M64 16MB, ATI Radeon 8500 и ATI 3D Rage128.

Таблица В2. *Конфигурация компьютера, на котором тестировались все примеры для книги*

Процессор	Intel Pentium-4 2.6C
Материнская плата	Asus P4B800 (i865PE)

Таблица B2 *(окончание)*

Оперативная память	1024MB PC3200
Процессор	Intel Pentium-4 2.6C
Видеокарты	ATI Radeon 9700 Pro 128MB
	NVIDIA GeForce FX 5800 Ultra 128MB
	NVIDIA GeForce2 MX 32MB
Операционная система	Microsoft Windows XP (Build 2600) Service Pack 1. English Version

Благодарности

За годы, которые потребовались на то, что бы разобраться с OpenGL, разработать учебный курс и написать эту книгу, я получил неоценимую помощь и поддержку от множества людей.

Во-первых, я хочу поблагодарить редакцию журнала "Программист", особенно Владимира Голубкова и Максима Туйкина, которые доверили неизвестному провинциальному программисту написание серии статей по OpenGL и прощали все задержки.

Отдельно хочется поблагодарить Михаила Краснова, автора книг "OpenGL. Графика в проектах Delphi" и "DirectX. Графика в проектах Delphi", который оказал мне неоценимую помощь в изучении OpenGL, подготовке книги и поиске издателя.

Я очень благодарен Геннадию Ригеру из ATI Technologies, который терпеливо отвечал на сотни моих вопросов по электронной почте, а также содействовал тому, чтобы примеры этой книги нормально работали на видеокартах ATI.

Алексей Лагуненко из ItLabs пригласил меня на конференцию NVIDIA для разработчиков. Именно это событие и вдохновило меня на написание книги.

Я признателен и многим другим людям, которые помогали мне при написании книги: Rev Lebaredian (NVIDIA), Rachel Lebaredian (Steamboat Software), Shawn Steiner (Discreet), Dan Lion (Turbo Squid) и Maxim Perminov (Intel).

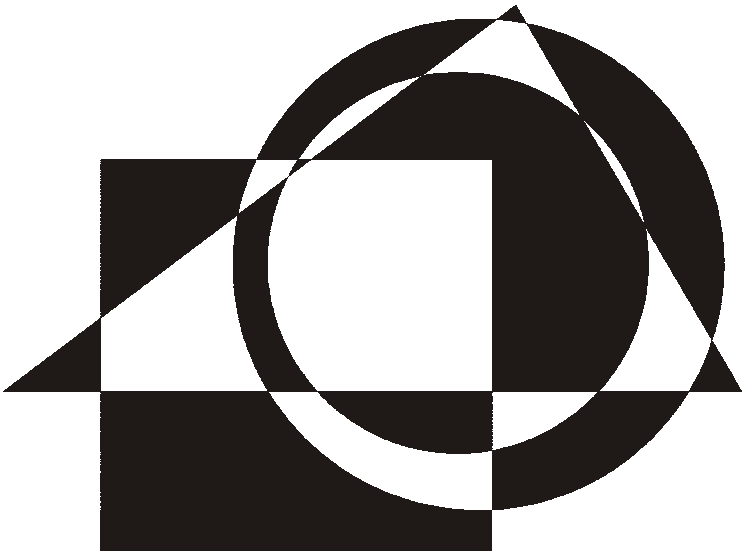
Мое понимание OpenGL формировалось под воздействием целого ряда публикаций, поэтому я очень хочу поблагодарить авторов статей на сайтах **www.ixbt.com**, **www.gamedev.ru**, **www.opengl.org**, **www.nvidia.com/developer**, **www.ati.com/developer**, а именно: Александра Медведева (iXBT), Андрея Воробьева (iXBT), Сергея Ваткина (GameDev), John Spitzer (NVIDIA), Sebastien Domine (NVIDIA) и Ричарда Хадди (ATI). Отдельная благодарность выражается сотрудникам NVIDIA Mark Kilgard и Cass Everitt, разработавшим библиотеки GLUT и OpenGL Helper Library, позволяющие на время забыть о существовании оконных функций, циклов обработки сообщений и прочих "ужасов" Win32.

Другим источником информации были форумы сайта iXBT (**forum.ixbt.com**), особенно форум разработчиков игр. Я бесконечно благодарен сообществу

этого форума за помощь в работе над книгой. Отдельная благодарность выражается Петру Попову, советы которого очень мне пригодились при написании книги.

В заключение хочу поблагодарить корпорации ATI и NVIDIA, которые предоставили в мое распоряжение видеокарты ATI Radeon 9700 Pro и NVIDIA GeForce FX 5800 Ultra, без которых написание этой книги было бы сильно затруднено.

Несмотря на поддержку и помощь других, ответственность за любые неточности и упущения полностью лежит на мне. Любые замечания и пожелания вы можете прислать по адресу **gsaf@sura.ru**.



ЧАСТЬ I

Использование NVIDIA OpenGL SDK

Глава 1

Библиотека GLUT

Как известно, библиотека OpenGL является кроссплатформенной: программа, написанная с использованием этой библиотеки, может работать под любой операционной системой, в которой имеется поддержка этой библиотеки. Но в действительности при реализации программ, написанных с ее помощью, возникает ряд трудностей.

Все дело в том, что в OpenGL отсутствуют средства для организации интерфейса с пользователем: инструменты для создания, работы с мышью и клавиатурой и т. д. Следовательно, при переносе программы на другую платформу ту часть программы, которая организует взаимодействие с пользователем, все равно придется переписывать.

Главная причина в том, что операции оконной подсистемы (создание окна, обработка сообщений, установка формата пикселов и многое другое) очень сильно зависят от операционной системы, и включение этих средств в OpenGL могло бы значительно сузить область применения этой библиотеки.

Поэтому существует потребность в надстройках над OpenGL, которые помогали бы в создании кроссплатформенных приложений. К ним относится библиотека GLUT (OpenGL Utility Toolkit). GLUT позволяет:

- создавать многооконные приложения;
- обрабатывать сообщения, используя процедуры обратного вызова (callback);
- работать с устройствами ввода информации, например, с мышью и клавиатурой;
- использовать таймеры;
- создавать всплывающие (pop-up) меню;
- работать со шрифтами и изображениями

и многое другое.

Программа, использующая GLUT версии 3.7, может работать в Windows 95, Windows NT, OS/2, UNIX и других ОС. Если программа использует только

возможности GLUT, то для переноса ее в любую операционную систему достаточно лишь перекомпилировать ее без изменения исходного текста. Показательно, что примеры, поставляемые с GLUT, работают во всех распространенных ОС без изменения исходного кода. Надо лишь использовать makefile от нужного компилятора.

Наряду с достоинствами у GLUT есть и недостатки. И как всегда, эти недостатки являются следствием достоинств. В данном случае это переносимость. Из-за переносимости приложения GLUT ограничены в функциональности и не имеют доступа ко всем возможностям операционной системы. В целом, библиотека является оптимальной для написания относительно небольших программ.

Но даже если вы не собираетесь использовать GLUT в своих программах, следует учитывать, что существует много программного обеспечения, которое использует эту библиотеку. Например, практически все примеры из NVIDIA OpenGL SDK написаны с использованием GLUT. Следовательно, чтобы разобраться в них, полезно будет знать основы GLUT.

1.1. Подключение GLUT к проекту

Создайте пустое приложение Win32:

1. В меню **File** выберите пункт **New | Project |Visual C++ Projects | Win32 Project**.
2. Выполните команду **Project | Properties | Linker | Command Line** (рис. 1.1).
3. В поле **Additional Options** впишите `glut32.lib` (разумеется, так же надо добавить и строку `opengl32.lib, glu32.lib /entry:"mainCRTStartup`, которая является обязательной для любой OpenGL-программы).

Все эти действия в окне **Property Pages** надо будет повторять каждый раз при создании нового проекта, иначе вы будете получать ошибки о неразрешенных ссылках при компоновке приложения.

Также надо учитывать, что для работы библиотеки необходим файл glut32.dll, который надо будет распространять вместе с программой, т. к. он не входит в стандартную поставку Windows. При установке программы его лучше всего размещать в каталоге \Windows\System 32.

Наконец, добавьте в проект новый файл (**Project | Add To Project | Files**) и назовите его, к примеру, main.cpp.

Теперь можно приступать к созданию нашей первой программы.

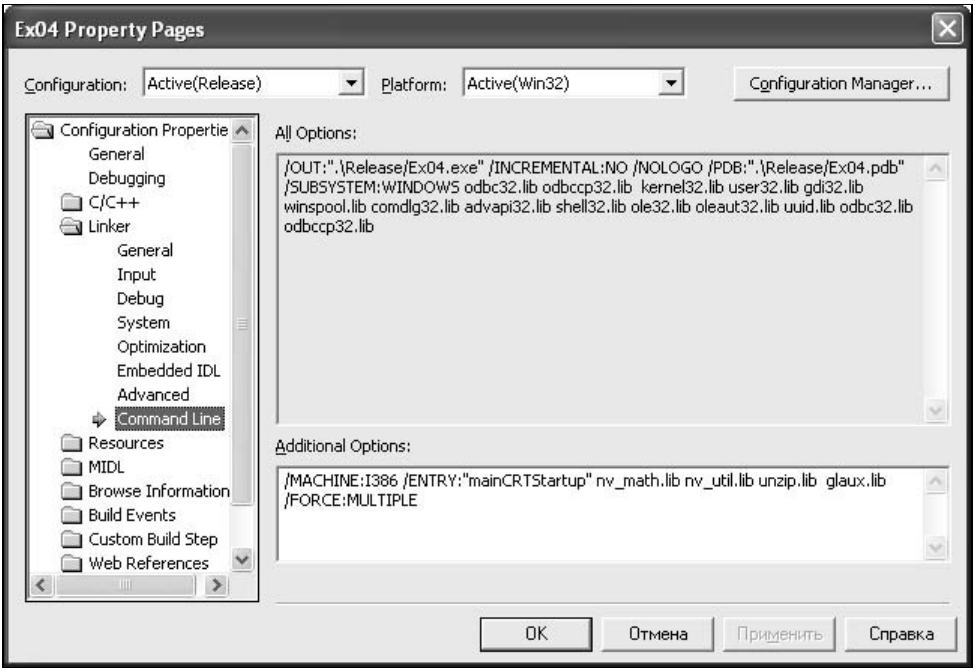


Рис. 1.1. Окно **Property Pages**

1.2. Пример простейшей программы, использующей GLUT

Далее приводится полный текст минимальной программы (листинг 1.1), использующей GLUT, которая рисует окно, заполняя его впоследствии синим цветом (Ex01) (рис. 1.2). Этот и все остальные примеры находятся на CD-диске, прилагаемом к книге. Бóльшая часть текстов примеров приводится частично.

Листинг 1.1

```
#include <gl\glut.h>

// Вывод изображения на экран
void Display()
{
  glClear(GL_COLOR_BUFFER_BIT);
// Смена переднего и заднего буферов экрана
```

```
  glutSwapBuffers();
}

int main(int argc, char* argv[])
{
  glutInit (&argc, argv);
// Задаем параметры окна
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
// Задаем размеры окна
  glutInitWindowSize(640, 480);
// Задаем позицию окна
  glutInitWindowPosition(100, 100);
// Создаем окно
  glutCreateWindow("My first OpenGL Application");

  glClearColor(0.5, 0.5, 0.75, 1);

// Задаем функцию обратного вызова изображения на экран
  glutDisplayFunc(Display);
  glutMainLoop();

  return 0;
}
```

Рис. 1.2. Простейшая GLUT-программа

Первая строка функции `main` вызывает команду `glutInit`, которая инициализирует библиотеку GLUT:

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

где:

- `argcp` — указатель на количество аргументов в командной строке;
- `argvv` — указатель на массив аргументов.

Обычно эти значения берутся из главной функции программы: `int main(int argc, char *argv[])`. Функция `glutInit` ищет в списке аргументов командной строки параметры, специфичные для GLUT, и инициализирует себя с учетом этих параметров. После чего убирает их из списка аргументов.

В настоящее время, ни один из параметров командой строки, распознаваемый функцией `glutInit`, не оказывает влияние на программу, работающую в операционных системах семейства Win32. Следовательно, если ваша программа будет работать только в операционных системах семейства Win32, вы можете опустить ее без ущерба для программы.

Следующая команда — `glutInitDisplayMode` — устанавливает формат пикселов окна программы.

```
void glutInitDisplayMode(unsigned int mode);
```

где:

- `mode` — битовая маска, каждый бит которой соответствует определенному атрибуту формата пикселов. Для более удобной установки формата пикселов в библиотеке GLUT имеется набор констант битовых масок (табл. 1.1), объединяя которые при помощи логической операции ИЛИ (OR), программист может создать требуемую ему битовую маску формата пикселов.

Таблица 1.1. *Основные предопределенные константы битовых масок формата пикселов библиотеки GLUT*

Константа	Описание
`GLUT_RGBA`	Необходим RGBA-видеорежим
`GLUT_RGB`	Псевдоним для `GLUT_RGB` (в библиотеке GLUT за поддержку альфа-канала отвечает битовая маска `GLUT_ALPHA`)
`GLUT_INDEX`	Необходим видеорежим цветовых индексов, т. е. целочисленные идентификаторы цветов должны использоваться в качестве индексов в палитре цветов
`GLUT_SINGLE`	Необходима поддержка одиночной буферизацией. Так как значение этой константы равно нулю, ее необязательно указывать

Таблица 1.1 *(окончание)*

Константа	Описание
GLUT_DOUBLE	Необходима поддержка двойной буферизации
GLUT_ACCUM	Необходим буфер аккумулятора
GLUT_ALPHA	Необходима поддержка альфа-канала
GLUT_DEPTH	Необходима поддержка буфера глубины
GLUT_STENCIL	Необходима поддержка буфера шаблона

В нашем случае флаг GLUT_RGBA указывает на то, что мы будем работать в режиме RGBA. В свою очередь флаг GL_DOUBLE свидетельствует о необходимости поддержки двойной буферизации.

Далее при помощи команд glutInitWindowSize и glutInitWindowPosition необходимо установить размеры окна и его позицию на экране:

```
void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
```

где:

- width — ширина окна в пикселах;
- height — высота окна в пикселах;
- x — положение левого верхнего угла окна (координата *x*);
- y — положение левого верхнего угла окна (координата *y*).

Мы устанавливаем размер окна (640 × 480) и положение его левого верхнего угла (100, 100). Собственно создание окна осуществляется функцией glutCreateWindow, которая создает окно с параметрами, заданными командами glutInitDisplayMode glutInitWindowSize и glutInitWindowPosition:

```
int glutCreateWindow(char *name);
```

где:

- name — заголовок создаваемого окна.

Наша программа создает окно с заголовком **My first OpenGL Application**.

Библиотека GLUT является событийно-ориентированной библиотекой, в результате чего любая GLUT-программа общается с внешним миром при помощи событий. Обработка событий в библиотеке GLUT реализована при помощи механизма функций обратного вызова (callback). Сначала программа передает библиотеке GLUT адрес функции обратного вызова, обрабатывающей данное событие, а затем, при наступлении этого события, биб-

лиотека вызывает указанную функцию обратного вызова, передавая ей, если необходимо, в качестве параметров информацию об особенностях события.

Для установки функции обратного вызова, отвечающей за перерисовку экрана, используется команда `glutDisplayFunc`:

```
void glutDisplayFunc(void (*func)(void));
```

где:

- `func` — адрес функции обратного вызова, отвечающей за обновление содержимого экрана. В Win32 эта функция фактически является обработчиком события `WM_PAINT`.

В нашей программе мы устанавливаем в качестве функции обратного вызова функцию `Display`. Функция `Display()` очень проста — она очищает экран и выводит полученное изображение, используя `glutSwapBuffers()`. Если при инициализации приложения мы используем флаг `GLUT_DOUBLE` вместо `GLUT_SINGLE`, то эту команду придется заменить командой `glFlush()`.

Предпоследняя строка функции `Main` при помощи команды `glutMainLoop()` запускает цикл обработки сообщений. В случае использования ОС семейства Windows команда `glutMainLoop` организует цикл обработки сообщений Win32.

Последняя строка функции `main (return 0)` — скорее дань традиции. В документации по библиотеке GLUT сказано, что функция `glutMainLoop()` никогда не возвращает управление программе. Поэтому если при завершении работы программы вам надо освободить системные ресурсы, то бессмысленно помещать этот код после вызова функции `glutMainLoop`.

Как видите, GLUT сильно облегчает написание простых программ. Кроме того, она делает их независимыми от платформы: в частности, эта программа после перекомпиляции сможет работать практически в любой операционной системе, имеющей поддержку OpenGL.

1.3. Работа с мышью и клавиатурой

В предыдущем разделе мы создали переносимое приложение, которое фактически ничего не делает. Сейчас мы попробуем добавить в него какие-нибудь дополнительные возможности. Для начала добавим в него обработку событий клавиатуры — завершение работы по нажатию клавиши <Esc>.

Для регистрации обработчика сообщений от клавиатуры в библиотеке GLUT используется команда `glutKeyboardFunc`:

```
void glutKeyboardFunc(void (*func)(unsigned char key,int x,   int y));
```

Как видно из определения этой функции, обработчик событий от клавиатуры принимает следующие параметры:

- `key` — идентификатор клавиши;
- `x`, `y` — координаты указателя мыши в момент нажатия клавиши.

Ниже приведен исходный код обработчика клавиатуры, который завершает работу программы при нажатии клавиши <Esc>

```
void Keyboard(unsigned char key, int x, int y)
{
  switch(key)
  {
  case 27:
  {
    exit(VK_ESCAPE);
    break;
  }
```

Работа с мышью организуется тоже очень просто (Ex02). Для регистрации обработчика события перемещения указателя мыши используются две команды: `glutMotionFunc` и `glutPassiveMotionFunc`:

```
void glutMotionFunc(void (*func)(int x, int y));
void glutPassiveMotionFunc(void (*func)(int x, int y));
```

Первая команда используется для обработки события перемещения мыши при нажатой кнопке мыши, вторая — при отжатой. Обработчик в качестве параметров принимает координаты (*x*, *y*) указателя мыши. Ниже приведен текст обработчика, который выводит в заголовке окна текущие координаты указателя мыши (рис. 1.3):

```
void MousePassiveMotion(int x, int y)
{
  char buf[80];
  sprintf(buf,"Mouse coords is: x=%d; y=%d", x, y);
  glutSetWindowTitle(buf);
}
glutPassiveMotionFunc(MousePassiveMotion);
```

Для установки заголовка окна используется функция `glutSetWindowTitle`:

```
void glutSetWindowTitle(char *name);
```

где:

- `name` — новый заголовок окна.

Рис. 1.3. Вывод точки на экран

Запустив программу на выполнение, обратите внимание на то, что при нажатой кнопке мыши обработка сообщения не происходит. Чтобы устранить этот недостаток, в функцию `main` необходимо добавить следующую строку:

```
glutMotionFunc(MousePassiveMotion);
```

Для того чтобы зарегистрировать обработчик событий нажатия кнопки мыши, применяется функция `glutMouseFunc`:

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
```

где:

- `button` — идентификатор кнопки мыши (`GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`, `GLUT_RIGHT_BUTTON` для левой, средней и правой соответственно);
- `state` — состояние кнопки мыши (`GLUT_DOWN` — нажата, `GLUT_UP` — отжата);
- `x`, `y` — координаты указателя мыши.

Приведенный далее фрагмент программы (часть кода опущена) при нажатии левой кнопки мыши рисует круглую точку размером 10 пикселов на месте курсора (листинг 1.2).

Листинг 1.2

```
#include <windows.h>
#include <gl\glut.h>
#include <stdio.h>

// Размеры окна
int WinWidth=640;
int WinHeight=480;
// Текущая позиция точки
int mx=0;
int my=0;

...

void Display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0, 1, 0);
// Рисуем точку
  glBegin(GL_POINTS);
    glVertex2f(mx, WinHeight-my);
  glEnd();

  glutSwapBuffers();
}

void Mouse(int button, int state, int x, int y)
{
// Если нажата левая кнопка мыши
  if ((button==GLUT_LEFT_BUTTON) | (state==GLUT_DOWN))
  {
// Устанавливаем координаты точки
```

```
    mx=x;
    my=y;
    glutPostRedisplay();
  }
}

int main(int argc, char* argv[])
{
...
  glClearColor(0.5, 0.5, 0.75, 1);

// Делаем точку круглой
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  glEnable(GL_BLEND);
  glEnable(GL_POINT_SMOOTH);
  glPointSize(10);
// Параметры проекции
  gluOrtho2D(0, WinWidth-1, 0, WinHeight-1);

  glutDisplayFunc(Display);
  glutMouseFunc(Mouse);
...
```

Заметьте, что в функции `Mouse` для перерисовки окна используется команда `glutPostRedisplay()`:

```
void glutPostRedisplay(void);
```

Эта функция посылает приложению сообщение о необходимости обновления окна, что в нашем случае приводит к вызову функции обратного вызова `Display()`.

Также обратите внимание на преобразование координат, которое выполняется в команде `glVertex3f`. Оно связано с тем, что системы координат окна Windows и OpenGL не совпадают (в Windows по умолчанию начало координат находится в левом верхнем углу, а в OpenGL — в левом нижнем).

Если вы поработаете некоторое время с этим примером, то увидите, что при изменении размеров окна приложение перестает корректно работать. Это связано с тем, что при изменении размеров окна надо менять матрицу модели командой `gluOrtho2D` и область вывода командой `glViewport`. Выходом из данной ситуации может служить использование команды `glutReshapeFunc`,