**OpenGL - Предисловие**

*Эта книга посвящается  
памяти певца, поэта и   
композитора, гражданина   
России   
Игоря Талькова*

**Предисловие**

Эта книга является самоучителем по популярной библиотеке OpenGL. OpenGL - Open Graphics Library, открытая графическая библиотека. Термин "открытый" - означает независимый от производителей. Имеется спецификация (стандарт) на OpenGL, где все четко задокументировано и описано. Библиотеку OpenGL может производить кто угодно. Главное, чтобы библиотека удовлетворяла спецификации OpenGL и ряду тестов. Как следствие, в библиотеке нет никаких темных мест, секретов, недокументированных возможностей и т.п, те кто программировал под MSWindows или MSDOS понимают о чем я говорю. Библиотеку выпускают такие корпорации, как Microsoft, Silicon Graphics, а также просто группы программистов. Одним из таких примеров служит реализация Mesa. Эту библиотеку написали целый ряд программистов, главным автором является Brian Paul. Библиотека Mesa распространяется в исходных текстах на языке Си и собирается почти для любой операционной системы. Стандарт OpenGL развивается с 1992 года. Он разрабатывается фирмой Silicon Graphics. С тех пор библиотека завоевала огромную популярность и была интегрирована со множеством языков и систем разработки приложений. Вы можете писать программу с использованием OpenGL на Си, С++, Pascal, Java и многих других языках. Основное предназначение OpenGL - программирование трехмерной графики.

Библиотека OpenGL представляет из себя интерфейс программирования трехмерной графики. Единицей информации является вершина, из них состоят более сложные объекты. Программист создает вершины, указывает, как их соединять (линиями или многоугольниками), устанавливает координаты и параметры камеры и ламп, а библиотека OpenGL берет на себя работу создания изображения на экране. OpenGL идеально подходит для программистов, которым необходимо создать небольшую трехмерную сцену и не задумываться о деталях реализации алгоритмов трехмерной графики. Для профессионалов, занимающихся программированием трехмерной графики, библиотека тоже будет полезной, т.к. она представляет основные механизмы и выполняет определенную автоматизацию. Используя OpenGL вы с легкостью создадите трехмерные поверхности, наложите на них текстуры, осветите источниками света, сможете сделать эффект тумана, прозрачности, смешивания цветов, а также сможете наложить трафарет, передвигать объекты сцены, лампы и камеры по заданным траекториям, сделав тем самым анимацию. OpenGL непосредственно не поддерживает работу с устройствами ввода, такими как мышь или клавиатура, т.к. эта библиотека является платформенно-независимой. Но вы можете задействовать функции конкретной операционной системы, под которую вы пишите свою программу или воспользуйтесь надстройками над OpenGL, такими как библиотеки GLUT или GLAUX. Тем самым ваша программа заживет новой жизнью. Возможности просто безграничны!

Изначально задумывалась некая брошюра, что-то типа небольшого самоучителя. Потом проект сильно разросся, и я подумал: "А почему бы не написать книгу?". Программированием я занимаюсь на протяжении последних пяти лет. За это время я прочел огромное количество книг по данной тематике. Оглядывая свои полки, я с сожалением отмечаю, что на них практически отсутствую российские авторы. Многие российские авторы страдают тем, что зачастую просто не вдумываясь передирают хелп. В результате получается не учебное пособие, а справка по-русски. Кому такое творение нужно? Все программисты владеют техническим английским и в состоянии сами прочесть справку. Основное место на моей полке занимают такие, ставшие классикой книги, как: "Язык программирования Си" Б.Керниган и Д.Ритчи, "Основы Visual C++" Д. Круглински, "Введение в системы баз данных" К. Дейт, "Программирование в Windows для профессионалов" Рихтер, "Руководство системного администратора" Эви Немет, "Объектно-ориентированный анализ и проектирование" Гради Буч, Бъерн Страуструп - "Язык программирования С++", Герберт Шилдт - "Программирование на Си и С++ в Windows 95", Герберт Шилдт "Программирование для профессионалов в MSDOS", Герберт Шилдт "Справочник по Java", Том Сван "OWL & BC4" и много других не менее известных авторов. Герберт Шилдт и Тома Сван пишут очень много и далеко не все их труды хорошо сделаны. У Шилдта была книга по алгоритмам, у Тома Свана что-то по Java, которые, как мне показалось стали явной неудачей. Но эти авторы пишут для начинающих и определенный уровень все-таки держат. Я вижу главной причиной отсутствия наших авторов в том, что профессионалы писать не хотят. В следствие чего на этот рынок вышли халтурщики, которые просто занимаются надувательством, ибо их книги можно применять, только по одному назначению, они собственно говоря столько и стоят. Вы обратите внимание на такую важную деталь, почти во всех зарубежных книгах указывается, как можно связаться с автором. В наших я такого нигде не видел, в самом лучшем случае: "если вы хотите написать отзыв, пишите в редакцию". Я уже довольно много грязи вылил на наших авторов, возможно, мне просто не попадались хорошие книги. Моя книга задумывается как серьезный труд, как следствие она полностью открыта. Книга доступна в Интернет, все ваши замечания можете прислать мне по электронной почте, а также обсудить их между собой в форуме на нашем сайте. Ниже указаны мои адреса в Интернет и ФидоНет. Также имеется мой телефон, если у вас серьезное предложение. За полгода мне пришло несколько сот писем, два-три в день, я ответил на половину из них. Отвечал преимущественно на те, которые касались самой книги, иногда я не знал ответа, поэтому отвечать мне было нечего. Задавайте ваши вопросы у нас в форуме, может ответ напишет кто-нибудь другой. Подводя итог сказанному, я хочу сделать классную книжку, я приглашаю принять в этом участие и всех вас. Книга сделана в электронном виде, поэтому ее объем безграничен. Я хочу сделать хорошую книгу в России, мне хочется показать, что мы еще чего-то можем!

Я рассчитываю, что вы уже неплохо знакомы с программированием, в частности, с языком Си. Здесь, в основном, будет рассматриваться написание программ в среде Windows. Рассмотрю, конечно же, и особенности использования OpenGL в операционной системе Unix. Я опишу несколько различных типов приложений. Первый тип - это консольное приложение win32 с использованием библиотеки glaux.lib. Второе - обычное оконное приложение(Win32 Application). Третье - это приложение на базе MFC с архитектурой документ/вид. Четвертое - я покажу, как писать Java-апплеты с использованием библиотеки Magician. Magician - это библиотека java-классов с функциями, идентичными OpenGL. И пятое - Unix-приложение. Различия в написании этих приложений проявляются в начальной инициализации, префиксах и суффиксах названий функций. В остальном, на 90% все везде одинаково. Книга состоит из двух частей - "Основы OpenGL" и "OpenGL для профессионалов". Первая часть является скорее учебником, вторая - справочником по малоизвестным функциональным возможностям OpenGL, она будет содержать массу интересных программ, где наглядно будет продемонстрирована вся мощь OpenGL. Я хотел сделать каждую главу книги самодостаточной. Насколько мне это удалось, судить вам. Но я бы рекомендовал читать книгу последовательно. Главное, на что я делал упор, так это на простоту и понятность. Все примеры очень простые, и их размер не превышает нескольких килобайт. Все написано на чистом Си. В основу книги положен реальный опыт работы. И последнее, что надо сказать здесь, я не претендую на полноту и детальную теоретическую точность изложения материала. Книга ставит задачей научить читателя элементарным вещам, и поэтому здесь все объясняется простым и понятным языком на конкретных примерах с точки зрения практика, а не теоретика. Более точную информацию вы сможете найти в спецификации по OpenGL и на сайте Silicon Graphics [http://www.sgi.com/software/opengl](http://r.codenet.ru/?http://www.sgi.com/software/opengl)  . Но заранее предупрежу вас, что чтение справочной информации или того же RedBook не доставит вам массу удовольствия, очень мудрено там все изложено. В отличии от аналогичных книг данной тематики вы не найдете здесь пустого переписывания и перевода справочной информации и не к месту приведенного кода. Все примеры и упражнения к книге были придуманы мной и сделаны мной. Книга написана при помощи пакета LATEX2e в ОС Linux.

**Об авторе**

С раннего детства увлекался всевозможными головоломками, искал пищу для мозгов. Уже в восемь лет в пионерском лагере занял второе место по шахматам, это было одно из первых достижений. С 13 лет активно увлекся математикой. Так я попал в вечернюю школу "Авангард" при МИФИ. Через два года поступил в [спецшколу при МФТИ](http://r.codenet.ru/?http://www.inse.kiae.ru/school/)  . Надо сказать, что отсюда началась моя дорога в жизнь. Здесь я познакомился с персональным компьютером, с программированием, с Интернет, а также неплохо развил свое серое вещество головного мозга в плане физики и математики. В этот же период я закончил ЗФТШ при МФТИ и очень горжусь свидетельством об окончании с пятеркой по физике и четверкой по математике. После окончания школы был год работы и учебы в [Курчатовском институте](http://r.codenet.ru/?http://www.inse.kiae.ru/)  , но интерес к физике у меня к тому времени окончательно пропал, за что меня и выгнали. Кстати, о чем я не жалею, и даже очень рад, потому что сейчас я занимаюсь делом, которое мне по душе, у меня гораздо больше свободы. В настоящее время я работаю в центре "Современных Информационных Технологий в Математическом Образовании"(СИТМО) в Московском Государственном Институте Электроники и Математики(МГИЭМ), а также учусь на втором курсе параллельно на двух факультетах: информатики и телекоммуникаций и прикладной математики. Я являюсь системным администратором UNIX & NT, программистом (C/C++/Java) и отчасти менеджером в одном лице, а с недавнего времени еще и книжки пишу;-) Имею сертификаты мастера C программирования, С++ программиста, Java программиста, Unix-администратора. Со мной можно связаться по электронной почте:

\* FidoNet: 2:5020/370.2 620.20 1103.5

\* www: [itsoft.miem.edu.ru](http://r.codenet.ru/?http://itsoft.miem.edu.ru)  , [opengl.miem.edu.ru](http://r.codenet.ru/?http://opengl.miem.edu.ru)

\* Inet: [itarasov@rtuis.miem.edu.ru](mailto:itarasov@rtuis.miem.edu.ru)

\* Phone: 916-89-51, 916-89-63

Если у вас имеются дополнения или какие-то материалы, которые, по вашему мнению, желательно включить в книгу, то мы с удовольствием обсудим эту тему. Особенно я занят поиском материалов для второй части книги.

И еще, книга написана достаточно вольным стилем. Я пытаюсь не уйти в жаргон и в тоже время не писать заумным или официальным языком. Главное - это понятливость, я хочу, чтобы меня понимали, чтобы текст читался легко. Наверняка вы найдете ошибки в книге или вас удивит мой вольный стиль изложения, "ну и что?" - отвечу я - мне всего лишь 19 лет и я не волшебник, я только учусь.

**Благодарности**

Хочется поблагодарить огромную массу людей, без которых эта книга вряд ли бы состоялась. Прежде всего, хочу выразить благодарность непосредственному участнику Александру Терещенко за перевод книги из формата LATEX2e в html формат и последующую верстку книги. Отдельно хочется поблагодарить моих родителей за их ценные замечания и первую редакцию данного материала, а также за все остальное, что они для меня сделали.

Также я выражаю отдельное признание следующим людям без косвенного участия которых, я бы вряд ли что-либо написал. Это прежде всего мои учителя, которые дали мне путевку в жизнь: Сергей Васильевич Масалович, Сергей Ростиславович Тумковский, Георгий Петрович Путилов.

Это ребята, с которыми я вместе работаю в настоящее время: Александр Измайлов, Слава Сотников, Иван Жаднов, Кирилл Марченков, Андрей Плоскирев, Паша Воротилин.

Это друзья - фидошники: Олег Федоров, Эльвира Николаева, Игорь Щеколев, Паша Гродек, Олег Давыдов, Ринат Садретдинов, Сергей Зефиров и далее весь наш московский пойнтлист;-) Они столько раз говорили, что я не прав, я иду не той дорогой, что я теперь только и делаю, что иду, иду, делаю, делаю, только бы что-то им доказать, и может быть потом мы как-нибудь обсудим куда я все-таки пришел.;-)

Отдельный привет и спасибо моим друзьям по курчатнику: Тимур Тудоровский, Андрей Казанцев, Андрей Бажан и также все остальным.

Также не могу не вспомнить о своей деревне и своих самых старых друзьях: Виталька, Иван Иваныч и др. Деревня рулез-форева, если в России, еще и осталось, что-то от слова Родина, то это деревня, рыбалка, лес и грибной сезон, убежден ни в какой Америке нет такого чудного места, как наша русская деревня!

**OpenGL - Быстрый старт**

**2.1 Устанавливаем OpenGL**

Начнем с самого главного, установим необходимое программное обеспечение. Я предполагаю, что Windows у Вас уже установлен и работает, в противном случае вы купили не ту книгу. Во-первых, установите MSVisualC++6.0 и jdk113 или выше, если вам интересно узнать о написание java-апплетов с использованием OpenGL. Впрочем java понадобится вам только в седьмой главе. Во-вторых, нам понадобится реализация библиотеки OpenGL. Она входит в поставку Windows95/NT - это библиотеки opengl32.dll & glu32.dll. Вы также можете взять библиотеки OpenGL от Silicon Graphics. Инструкция по установке там прилагается. Вам придется скопировать opengl.dll и glu.dll в windows\system и положить opengl.lib, glu.lib в подкаталог Lib, где установлено MSVisualC++. В-третьих, вам понадобятся четыре моих программы-шаблона, которые представляют собой начальный скелет, который мы потом будем наполнять функциональностью. Где взять OpenGL от Silicon Graphics, Magician, jdk и мои программы-шаблоны, смотри в приложение 'А'.

**2.2 Давайте что-нибудь изобразим**

Самым простым объектом, с помощью которого можно увидеть всю мощь OpenGL, является сфера. Давайте попытаемся ее изобразить. Для этого надо создать новый проект в VisualC++, выполните следующие действия:

1. Запустите MSVisualC++6.0
2. Щелкните меню File->New->Win32 Console Application.
3. Выберете каталог и имя проекта, впишите - sphere, щелкните OK.

*Я все свои проекты держу на диске D в каталоге Projects. Projects ветвится дальше на подкаталоги с базами данных, утилитами, графикой и Java-приложениями. В общем, старайтесь присваивать разумные имена и вести разумную структуру каталогов. Это очень серьезная проблема.*

1. Выберете An Empty Project, щелкните Finish.
2. Cкопируйте в каталог вашего проекта мой шаблон glaux.c и переименуйте его в sphere.c
3. Присоедините его к проекту. Project->Add To Project->Files
4. Щелкните Build->Set Active Configuration и установите тип проекта sphere - Win32 Release
5. Далее, щелкайте Project->Settings->Link->Object/library modules и добавьте туда opengl32.lib, glu32.lib и glaux.lib
6. Вставьте в функцию display следующий код:
7. glColor3d(1,0,0);
8. auxSolidSphere(1);
9. Теперь откомпилируйте и запустите Вашу программу.  
   Меню Build->Execute Sphere.exe



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/glaux/glaux.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/glaux/release/glaux.exe)  .

Теперь поясню назначение тех двух функций, что вы вставили в пункте 9. Функция glColor3d устанавливает текущий цвет, которым будут рисоваться фигуры. Тут нужно пояснить, как устанавливается цвет и общую философию в названии функций OpenGL. Цвет устанавливается четырьмя параметрами: красный, синий, зеленый и прозрачность. Эти параметры варьируют в диапазоне от нуля до единицы. Четвертый параметр нам пока не нужен, поэтому мы вызвали glColor с тремя параметрами. В этом случае, значение четвертого параметра, прозрачности, по умолчанию считается равным единице, т.е. абсолютно непрозрачным, ноль - будет абсолютно прозрачным. Так как в языке Си нет перегрузки функций, то применяется следующий синтаксис вызова функций - FuncionName[n=число параметров][тип параметров]. Доступны следующие типы:

* b - GLbyte байт
* s - GLshort короткое целое
* i - GLint целое
* f - GLfloat дробное
* d - GLdouble дробное с двойной точностью
* ub - GLubyte беззнаковый байт
* us - GLushort беззнаковое короткое целое
* ui - GLuint беззнаковое целое
* v - массив из n параметров указанного типа

В нашем случае - glColor3d - означает, что в функцию передается три параметра типа GLdouble. Также можно было вызвать glColor3i, т.е. три параметра типа GLint. Если тип параметров короткое целое, целое, байт или длинное, то компонента цвета приводится к диапазону [0,1]. Приведение к этому диапазону осуществляется по следующим образом. В случае беззнакового типа наибольшее возможное значение приводится к единице, ноль к нулю. В случае знакового максимальное значение приводится к единице или к минус единице, а минус единица будет приведена к единице. Мудрено, сразу и не поймешь. На практике вы будете пользоваться одним из трех случаев, рассмотренных в качестве примера ниже. Например, для беззнакового байта приведение будет осуществлено по следующей формуле: значение\_переменной\_хранимой\_в\_байте/255, т.к. 255 максимальное число, хранимое в одном байте. Функция glColor3dv означает, что в качестве параметров передается массив из трех элементов типа GLdouble. Например:

double array[] = {0.5, 0.75, 0.3, 0.7};

...

glColor3dv(array);

glColor3ub(200,100,0); // приводится к 200/256, 100/256, 0,256

glColor3d(0.25,0.25,0); // темно-желтый

glColot3ub(0,100,0); // темно-зеленый

glColor3ub(0,0,255); // синий

Функция auxSolidSphere - рисует сферу в начале координат и единичным радиусом. Освещение, создание ламп и установка положения камеры мы рассмотрим чуть позже, а пока давайте получше освоимся и почувствуем насколько здесь все просто. Первую программу под моим руководством вы уже написали. Теперь выполните самостоятельные упражнения.

**2.3 Упражнение: "Трехмерные фигуры"**

Замените функцию auxSolidSphere на функцию, из указанных ниже с соответствующими параметрами. Значения параметров устанавливайте порядка единицы - 0.5-1.7. Если вы укажете слишком маленький размер, фигуру будет плохо видно; если слишком большой, то она получится урезанной. Это связано с тем, что ее край, как бы вылезет из монитора.

* auxSolidCube(width) // куб
* auxSolidBox(width, height, depth) // коробка
* auxSolidTorus(r,R) // тор
* auxSolidCylinder(r,height) // цилиндр
* auxSolidCone(r,height) // конус
* auxSolidIcosahedron(width) // многогранники
* auxSolidOctahedron(width)
* auxSolidTetrahedron(width)
* auxSolidDodecahedron(width)
* auxSolidTeapot(width) // рисует чайник

С помощью выше указанных функций вы можете рисовать сплошные фигуры. Если вам надо нарисовать проволочную, то вместо Solid пишите Wire.

Пример:

auxWireCube(1) // рисует проволочный куб

**2.4 Переход к новым координатам**

Продолжим рисовать трехмерные фигуры. В предыдущем параграфе вы научились рисовать примитивные трехмерные объекты. Но проблема в том, что они рисуются только в начале координат, т.е. в точке (0,0,0). Для того чтобы изобразить сферу в точке ( x0,y0,z0 ), надо переместить начало координат в эту точку, т.е. надо перейти к новым координатам. Эта процедура довольно распространенная при программировании графики и анимации. Часто, бывает очень удобно, сместить координаты в новую точку и повернуть их на требуемый угол, и ваши расчеты резко упростятся. Конкретный пример мы рассмотрим ниже, когда научимся программировать анимацию. А пока вы узнаете, как переходить к новым координатам. Для перехода к новым координатам в OpenGL есть две функции:

* glTranslated( x,y,z )
* glRotated(,x0,y0,z0)

Первая функция сдвигает начало системы координат на ( x,y,z ). Вторая - поворачивает на угол  против часовой стрелки вокруг вектора (x0,y0,z0). Теперь, стоит сказать еще о двух очень важных функциях:

* glPushMatrix()
* glPopMatrix()

Они предназначены для сохранения и восстановления текущих координат. Я не стал здесь приводить пример на то, как неудобно переходить от одной системы координат к другой и помнить все ваши переходы. Гораздо удобнее с помощью glPushMatrix() сохранить текущие координаты, потом сдвигаться, вертеться, как вам угодно, а после, вызовом glPopMatrix вернуться к старым координатам. Итак, настало время поэкспериментировать. Создайте новый проект, повторив пункты из раздела 2.2. Только назовите его sphere2. Сначала мы рассмотрим сдвиг координат. Вставьте в функцию display следующий код:

glPushMatrix(); // сохраняем текущие координаты

glTranslated(1.4,0,0); // сдвигаемся по оси Х на 1.4

glColor3d(0,1,0);

auxSolidSphere(0.5); // рисуем сферу в (1.4,0,0)

// в абсолютных координатах

glTranslated(1,0,0); // еще раз сдвигаемся

glColor3d(0,0,1);

auxSolidSphere(0.3);

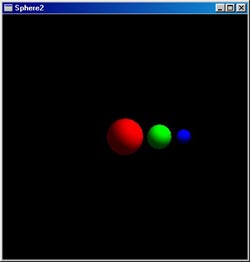
glPopMatrix(); // возвращаемся к старой системе координат

glColor3d(1,0,0);

auxSolidSphere(0.75); // рисуем сферу в точке (0,0,0)

// в абсолютных координатах

Теперь откомпилируйте и запустите вашу программу.  
Меню Build->Execute Sphere.exe



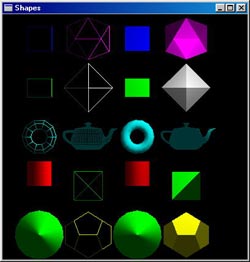
Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/sphere2/sphere2.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/sphere2/release/sphere2.exe)  .

В общем-то, из комментариев многое понятно. Обращаю ваше внимание только на два последних вызова auxSolidSphere. Перед вызовом glPopMatrix сфера рисуется в точке (2,0,0), а после, в точке (0,0,0).

**2.5 Упражнение: "Cписок трехмерных фигур"**

Используя список функций из предыдущего упражнения, нарисуйте эти фигуры в два столбца. Слева проволочные. Справа сплошные.

Примечание: тут хочу заметить, что в версии glaux.lib от фирмы Microsoft имеется следующий баг: цилиндр и конус рисуются всегда либо проволочными, либо сплошными. Если вы первый цилиндр/конус в программе нарисовали проволочным, то далее все цилиндры/конусы будут проволочными. Соответственно, если первой была сплошная фигура, то далее все будут сплошные. Поэтому, не стоит паниковать. Это ошибка Microsoft. Могу также вас порадовать, что ниже я расскажу, как обойти эту проблему.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/shapes/shapes.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/shapes/release/shapes.exe)  .

**2.6 Поворот координат**

Теперь рассмотрим вращение координат. Создайте новый проект с именем Rotate. Переименуйте glaux.c в rotate.c В функцию display вставьте следующий код:

glColor3d(1,0,0);

auxSolidCone(1, 2); // рисуем конус в центре координат

glPushMatrix(); // сохраняем текущие координаты

glTranslated(1,0,0); // сдвигаемся в точку (1,0,0)

glRotated(75, 1,0,0); // поворачиваем систему координат на 75 градусов

glColor3d(0,1,0);

auxSolidCone(1, 2); // рисуем конус

glPopMatrix(); // возвращаемся к старым координатам

Как видите, конус повернулся в абсолютных координатах. Так что, для того, чтобы нарисовать фигуру не в начале координат, надо:

1. сохранить текущие координаты
2. сдвинуть(glTranslated), повернуть(glRotated)
3. нарисовать то, что хотели
4. вернуться к старым координатам

Вызовы glPushMatrixglPopMatrix могут быть вложенными, т.е.:

glPushMatrix();

...

glPushMatrix();

glPopMatrix();

...

glPopMatrix();

Естественно число вызовов glPopMatrix должно соответствовать числу вызовов glPushMatrix, иначе у вас сцена улетит в неизвестном направление. Максимально допустимая глубина вложенности glPushMatrix/glPopMatrix определяется следующим образом:

int n;

glGetIntegerv(GL\_MAX\_MODELVIEW\_STACK\_DEPTH, &n);

printf("n=%d\n",n);

Спецификация на OpenGL гарантирует, что глубина стека не может быть меньше 32.

**2.7 Упражнение: "Снеговик"**

Используя функцию glRotate, нарисуйте снеговика. Три сферы, шапка - конус, нос - тоже конус, глаза - сфера, рот можно квадратным сделать - glBox.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/snowman/snowman.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/snowman/release/snowman.exe)  .

Примечание: Имеется еще один баг в glaux.lib от Microsoft. Кажется, последний из обнаруженных мной. Функция aux[Solid/Wire]Cylinder прежде, чем нарисовать цилиндр, сдвигает и поворачивает координаты. Так что, если вы уже сместили и повернули координаты, то цилиндр нарисуется совсем не там, где вы рассчитывали. Люди из Microsoft, конечно же, будут уверять, что это особенность, и предложат вам скачать очередной ServicePack.;-) А я ниже расскажу, как более правильно рисовать цилиндры и конусы. Если вам некогда ждать, то далее приведен исправленный код этих функций. Большинство из вас сейчас пропустят его и перейдут к следующему очень интересному разделу - "Анимация". И правильно сделаете. К исправлению ошибок вернетесь, когда немного освоитесь. Но я все же решил привести код по исправлению ошибок Microsoft именно здесь. Можете пересобрать glaux.lib, заменив соответствующий код в файле shapes.c. Где взять исходники, смотрите в приложение 'A'. По-моему, они есть в MSDN.

void auxWireCylinder (GLdouble radius, GLdouble height)

{

GLUquadricObj \*quadObj;

GLdouble \*sizeArray, \*tmp;

GLuint displayList;

sizeArray = (GLdouble \*) malloc (sizeof (GLdouble) \* 2);

tmp = sizeArray;

\*tmp++ = radius;

\*tmp++ = height;

displayList = findList (CYLINDERWIRE, sizeArray, 2);

if (displayList == 0) {

glNewList(makeModelPtr (CYLINDERWIRE, sizeArray, 2),

GL\_COMPILE\_AND\_EXECUTE);

quadObj = gluNewQuadric ();

gluQuadricDrawStyle (quadObj, GLU\_LINE);

gluCylinder (quadObj, radius, radius, height, 12, 2);

glEndList();

}

else {

glCallList(displayList);

free (sizeArray);

}

}

void auxSolidCylinder (GLdouble radius, GLdouble height)

{

GLUquadricObj \*quadObj;

GLdouble \*sizeArray, \*tmp;

GLuint displayList;

sizeArray = (GLdouble \*) malloc (sizeof (GLdouble) \* 2);

tmp = sizeArray;

\*tmp++ = radius;

\*tmp++ = height;

displayList = findList (CYLINDERSOLID, sizeArray, 2);

if (displayList == 0) {

glNewList(makeModelPtr (CYLINDERSOLID, sizeArray, 2),

GL\_COMPILE\_AND\_EXECUTE);

quadObj = gluNewQuadric ();

gluQuadricDrawStyle (quadObj, GLU\_FILL);

gluQuadricNormals (quadObj, GLU\_SMOOTH);

gluCylinder (quadObj, radius, radius, height, 12, 2);

glEndList();

}

else {

glCallList(displayList);

free (sizeArray);

}

}

void auxSolidCone (GLdouble base, GLdouble height)

{

GLUquadricObj \*quadObj;

GLdouble \*sizeArray, \*tmp;

GLuint displayList;

sizeArray = (GLdouble \*) malloc (sizeof (GLdouble) \* 2);

tmp = sizeArray;

\*tmp++ = base;

\*tmp++ = height;

displayList = findList (CONESOLID, sizeArray, 2);

if (displayList == 0) {

glNewList(makeModelPtr (CONESOLID, sizeArray, 2),

GL\_COMPILE\_AND\_EXECUTE);

quadObj = gluNewQuadric ();

gluQuadricDrawStyle (quadObj, GLU\_FILL);

gluQuadricNormals (quadObj, GLU\_SMOOTH);

gluCylinder (quadObj, base, (GLdouble)0.0, height, 15, 10);

glEndList();

}

else {

glCallList(displayList);

free (sizeArray);

}

}

**2.8 Анимация**

Давайте оживим нашего снеговика и добавим интерактивность. Для этого надо отрисовывать кадры и реагировать на внешние события от клавиатуры или мыши. Для отрисовки кадров их надо как-то различать. Для этого мы в функции display вводим переменную time типа int с модификатором static. Создайте новый проект и в функцию display введите следующее: "static int time=0;". Модификатор static означает, что значение переменной будет сохраняться при выходе из функции. Начальное значение мы устанавливаем в ноль. Если функция display не зависит от времени, то счетчик кадров можно и не вести. Теперь добавьте следующие строчки:

glPushMatrix();

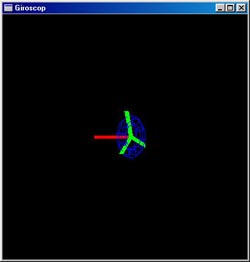
glTranslated( ((double)time)/100.0 ,0,0);

... // здесь код из предыдущего упражнения "Cнеговик"

glPopMatrix();

Запустив приложение, вы увидите, что снеговик пополз вдоль оси Х. Вот так вы можете делать анимацию. Т.е. теперь координаты объектов вычисляются в зависимости от времени. Я привел простой пример. Вообще говоря, для программирования сложной графики вам понадобится вычислять координаты каждого отдельного объекта в зависимости от времени.

Далее мы рассмотрим пример с более сложной анимацией. Здесь вращается тор и его ось вращения. Я приведу исходный текст с подробными комментариями. Пример программы "Гироскоп":



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/giroscop/giroscop.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/giroscop/release/giroscop.exe)  .

glPushMatrix(); // сохраняем текущие координаты, т.к. при выходе

// из функции нам нужно вернуться к абсолютным

// координатам

// попробуйте закомментировать вызов glPushMatrix

// здесь и glPopMatrix в конце и вы увидите, что

// из этого получится

glRotated(time/2, 0.0, 1.0, 0.0); // поворачиваем координаты

glLineWidth(5); // устанавливаем толщину линии - пять

glColor3f(1,0,0); // устанавливаем текущий цвет - красный

glBegin(GL\_LINES); // рисуем красную ось

glVertex3d(-0.3,0,0); // т.к. мы повернули координаты,

glVertex3d(1.5,0,0); // ось будет повернута в абсолютных

glEnd(); // координатах, и т.к. функция display

// вызывается в цикле, и переменная

// time увеличивается на единицу

// при каждом вызове display, мы

// получаем анимацию - вращающуюся ось

glPushMatrix(); // теперь относительно повернутых координат

// мы переходим к новым координатам

glRotated(2\*time, 1,0,0); // поворачиваем их на угол

// 2\*time вокруг красной оси

glTranslated(-0.3,0,0); // и сдвигаем на край оси

glColor3f(0,0,1); // устанавливаем синий цвет

glPushMatrix(); // теперь еще раз переходим к

// новым координатам,

glRotated(90,0,1,0); // чтобы развернуть тор на

// 90 градусов

// у вас крыша еще не поехала?

// немного?

// то-то же, тут не так все просто,

// но если понять что и как, то

// очень удобно

// программировать графику

glLineWidth(1);

auxWireTorus(0.2, 0.7);

glPopMatrix(); // нарисовав тор, мы возвращаемся

// к предыдущим

// координатам, если забыли,

// начало этих координат

// лежит на конце красной оси

glLineWidth(7);

glColor3f(0,1,0);

glBegin(GL\_LINES); // теперь рисуем три зеленых линии

glVertex3d(0,0,0); // что тут и как рисуется, я поясню в

glVertex3d(0,1,0); // следующей

glVertex3d(0,0,0); // главе, сейчас это не так важно

glVertex3d(0,-0.5,1);

glVertex3d(0,0,0);

glVertex3d(0,-0.5,-1);

glEnd();

glPopMatrix(); // теперь возвращаемся к повернутым

// координатам на угол time/2

glPopMatrix(); // ну и переходим к абсолютным координатам

time++; // увеличиваем счетчик кадров или время,

// называйте как хотите,

// на единицу

Как вы, наверное, догадались, надо создать очередной проект, скопировать туда мой шаблон, добавить его в проект, указать библиотеки opengl32.lib glu32.lib glaux.lib в Project->Setting->Link->Settings->Link->Object/library modules:, вставить этот код в функцию display. Еще вам нужно в начале функции display вставить строку static int time=0; и закомментировать строчку glEnable(GL\_LIGHTING) в функции main.

Запустив это приложение, вы увидите, как оно работает. Теперь закомментируйте соответствующие друг другу вызовы glPushMatrix и glPopMatrix и посмотрите на результат. Я бы рекомендовал такой способ для изучения и понимания работы программы: вы комментируете куски кода и смотрите, что получается.

Для того, чтобы реагировать на клавиатуру или мышь, вы должны определить функцию, которая будет вызываться при поступление событий от клавиатуры или мыши. Для кнопки клавиатуры вы определяете функцию со следующим прототипом void CALLBACK FunctionName(void) и устанавливаете ее как обработчик определенной кнопки - auxKeyFunc(key\_code, FunctionName); key\_code смотри в glaux.h. Пример: auxKeyFunc(AUX\_LEFT, FunctionName) Здесь вы устанавливаете FunctionName как функцию, которая будет вызываться при нажатии на клавиатуре клавиши "стрелка влево".

Для мыши вы устанавливаете свою функцию обработки событий мыши вызовом функции auxMouseFunc(int button,int action,AUXMOUSEPROC). Переменная button может принимать значения - AUX\_LEFTBUTTON, AUX\_MIDDLEBUTTON, AUX\_RIGHTBUTTON. Переменная action принимает следующие значения - AUX\_MOUSEDOWN, AUX\_MOUSEUP, AUX\_MOUSELOC. Функция AUXMOUSEPROC должна иметь прототип - void CALLBACK FunctionName(AUX\_EVENTREC \*event), где AUX\_EVENTREC определено как

typedef struct \_AUX\_EVENTREC

{

GLint event;

GLint data[4];

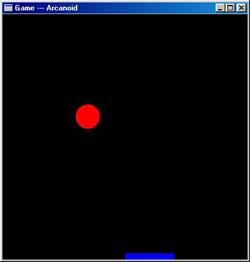
}AUX\_EVENTREC;

Для более детальной информации смотрите справочные руководства и исходные тексты библиотеки OpenGL Auxiliary library. Эта книга об OpenGL, а не о программировании интерфейсов пользователя. Поэтому, за подобной информацией вам придется лезть в соответствующие справочные руководства по Win API, MFC, Java и т.п.

В FunctionName вы можете изменить какую-нибудь глобальную переменную, и, соответственно, функция display будет рисовать кадры в зависимости от этой переменной.

**2.9 Упражнение:" Игра Арканоид"**

Напишите игру "Arconoid" - летающий шарик, снизу подставка, пользователь стрелками или мышкой управляет подставкой и отбивает шарик.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/arcanoid/arcanoid.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/arcanoid/release/arcanoid.exe)  .

**2.10 Резюме**

На этом эта глава книги закончена. Вы научились рисовать трехмерные объекты и делать интерактивную анимацию, используя эти объекты. Этого, в общем-то, достаточно для написания примитивных программ. В следующей главе мы научимся рисовать примитивные объекты: точки, линии, ломаные и многоугольники.

**OpenGL - Рисуем простые объекты**

**3.1 Общие положения**

Точки, линии, треугольники, четырехугольники, многоугольники - простые объекты, из которых состоят любые сложные фигуры. В предыдущей главе мы рисовали сферу, конус и тор. OpenGL непосредственно не поддерживает функций для создания таких сложных объектов, т.е. таких функций нет в opengl32.dll. Эти функции есть в библиотеки утилит glu32.dll, и устроены они следующим образом. Для того чтобы нарисовать сферу функция auxSolidSphere использует функции из библиотеки glu32.dll, а те в свою очередь, используют базовую библиотеку opengl32.dll и из линий или многоугольников строят сферу. Примитивы создаются следующим образом:

glBegin(GLenum mode); // указываем, что будем рисовать

glVertex[2 3 4][s i f d](...); // первая вершина

... // тут остальные вершины

glVertex[2 3 4][s i f d](...); // последняя вершина

glEnd(); // закончили рисовать примитив

Сначала вы говорите, что будете рисовать - glBegin с соответствующим параметром. Возможные значения mode перечислены ниже в таблице. Далее вы указываете вершины, определяющие объекты указанного типа. Обычно вы будете задавать вершину одним из четырех способов.

glVertex2d(x,y); // две переменных типа double

glVertex3d(x,y,z); // три переменных типа double

glVertex2dv(array); // массив из двух переменных типа double

glVertex3d(array); // массив из трех переменных типа double

И наконец, вы вызываете glEnd, чтобы указать, что вы закончили рисовать объекты типа, указанного в glBegin. Далее мы подробно разберем создание всех примитивов.

|  |  |
| --- | --- |
| Значение mode | Описание |
| GL\_POINTS | Каждый вызов glVertex задает отдельную точку. |
| GL\_LINES | Каждая пара вершин задает отрезок. |
| GL\_LINE\_STRIP | Рисуется ломанная. |
| GL\_LINE\_LOOP | Рисуется ломанная, причем ее последняя точка соединяется с первой. |
| GL\_TRIANGLES | Каждые три вызова glVertex задают треугольник. |
| GL\_TRIANGLE\_STRIP | Рисуются треугольники с общей стороной. |
| GL\_TRIANGLE\_FAN | Тоже самое, но по другому правилу соединяются вершины, вряд ли понадобится. |
| GL\_QUADS | Каждые четыре вызова glVertex задают четырехугольник. |
| GL\_QUAD\_STRIP | Четырехугольники с общей стороной. |
| GL\_POLYGON | Многоугольник. |

**3.2 Точки**

Вы можете нарисовать столько точек, сколько вам нужно. Вызывая glVertex3d, вы устанавливает новую точку. При создании точек вы можете изменять следующие параметры. Вы можете вызывать glColor3d внутри glBegin/glEnd. Размер точки можно устанавливать с помощью функции:

void glPointSize(GLfloat size)

Режим сглаживания можно устанавливать вызовом функции

glEnable(GL\_POINT\_SMOOTH)

Отключается соответственно вызовом glDisable() c этим параметром. Последние функции - glPointSize и glEnable/glDisable надо вызывать вне glBegin/glEnd, иначе они будут проигнорированы. Функции glEnable/glDisable включают/выключают множество опций, но вы должны учитывать, что некоторые опции влекут за собой большие вычисления и, следовательно, изрядно затормаживают ваше приложение, поэтому без надобности не стоит их включать. Очевидно, что совершенно не к чем включать освещение, наложение текстуру и сглаживания цветов при рисовании точек. Пока вы с этими возможностями OpenGL не познакомились, поэтому запомните это на будущее.

// рисуем точки

glPointSize(2);

glBegin(GL\_POINTS);

glColor3d(1,0,0);

glVertex3d(-4.5,4,0); // первая точка

glColor3d(0,1,0);

glVertex3d(-4,4,0); // вторая точка

glColor3d(0,0,1); // третья

glVertex3d(-3.5,4,0);

glEnd();

glPointSize(5);

glBegin(GL\_POINTS);

glColor3d(1,0,0);

glVertex3d(-2,4,0); // первая точка

glColor3d(0,1,0);

glVertex3d(-1,4,0); // вторая точка

glColor3d(0,0,1); // третье

glVertex3d(0,4,0);

glEnd();

glPointSize(10);

glEnable(GL\_POINT\_SMOOTH);

glBegin(GL\_POINTS);

glColor3d(1,0,0);

glVertex3d(2,4,0); // первая точка

glColor3d(0,1,0);

glVertex3d(3,4,0); // вторая точка

glColor3d(0,0,1); // третья

glVertex3d(4,4,0);

glEnd();

glDisable(GL\_POINT\_SMOOTH);

**3.3 Линии**

Для линий вы также можете изменять ширину, цвет, размер сглаживание. Если вы зададите разные цвета для начала и конца линии, то ее цвет будет переливающемся. OpenGL по умолчанию делает интерполяцию. Так же вы можете рисовать прерывистые линии, делается это путем наложения маски при помощи следующей функции:

void glLineStipple(GLint factor, GLushort pattern );

Второй параметр задает саму маску. Например, если его значение равно 255(0x00FF), то чтобы вычислить задаваемую маску воспользуемся калькулятором. В двоичном виде это число выглядит так: 0000000011111111, т.е. всего 16 бит. Старшие восемь установлены в ноль, значит тут линии не будет. Младшие установлены в единицу, тут будет рисоваться линия. Первый параметр определяет, сколько раз повторяется каждый бит. Скажем, если его установить равным 2, то накладываемая маска будет выглядеть так:

00000000000000001111111111111111

Далее приведен исходный текст с комментариями для наглядной демонстрации что к чему.

glLineWidth(1); // ширину линии

// устанавливаем 1

glBegin(GL\_LINES);

glColor3d(1,0,0); // красный цвет

glVertex3d(-4.5,3,0); // первая линия

glVertex3d(-3,3,0);

glColor3d(0,1,0); // зеленый

glVertex3d(-3,3.3,0); // вторая линия

glVertex3d(-4,3.4,0);

glEnd();

glLineWidth(3); // ширина 3

glBegin(GL\_LINE\_STRIP); // см. ниже

glColor3d(1,0,0);

glVertex3d(-2.7,3,0);

glVertex3d(-1,3,0);

glColor3d(0,1,0);

glVertex3d(-1.5,3.3,0);

glColor3d(0,0,1);

glVertex3d(-1,3.5,0);

glEnd();

glLineWidth(5);

glEnable(GL\_LINE\_SMOOTH);

glEnable(GL\_LINE\_STIPPLE); // разрешаем рисовать

// прерывистую линию

glLineStipple(2,58360); // устанавливаем маску

// пояснения см. ниже

glBegin(GL\_LINE\_LOOP);

glColor3d(1,0,0);

glVertex3d(1,3,0);

glVertex3d(4,3,0);

glColor3d(0,1,0);

glVertex3d(3,2.7,0);

glColor3d(0,0,1);

glVertex3d(2.5,3.7,0);

glEnd();

glDisable(GL\_LINE\_SMOOTH);

glDisable(GL\_LINE\_STIPPLE);

**3.4 Треугольники**

Для треугольника можно задавать те же параметры, что и для линии плюс есть еще одна функция glPolygonMode. Она устанавливает опции для отрисовки многоугольника. Первый параметр может принимать значения - GL\_FRONT, GL\_BACK и GL\_FRONT\_AND\_BACK. Второй параметр указывает, как будет рисоваться многоугольник. Он принимает значения - GL\_POINT(рисуются только точки), GL\_LINE(рисуем линии) и GL\_FILL(рисуем заполненный многоугольник). Первый параметр указывает: к лицевой, тыльной или же к обеим сторонам применяется опция, заданная вторым параметром. Треугольники можно рисовать, передав GL\_TRIANGLE\_STRIP или GL\_TRIANGLE\_FAN в glBegin. В первом случае первая, вторая и третья вершины задают первый треугольник. Вторая, третья и четвертая вершина - второй треугольник. Третья, четвертая и пятая вершина - третий треугольник и т.д. Вершины n, n+1 и n+2 определят n-ый треугольник. Во втором случае первая, вторая и третья вершина задают первый треугольник. Первая, третья и четвертая вершины задают второй треугольник и т.д. Вершины 1, n+1, n+2 определяют n-ый треугольник. Далее следует пример с комментариями.

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL); // см. выше

glBegin(GL\_TRIANGLES);

glColor3d(1,0,0); // рисуем треугольник

glVertex3d(-4,2,0);

glVertex3d(-3,2.9,0);

glVertex3d(-2,2,0);

glEnd();

glLineWidth(2);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE); //рисуем

// проволочные треугольники

glBegin(GL\_TRIANGLE\_STRIP); // обратите внимание на порядок

// вершин

glColor3d(0,1,0);

glVertex3d(1,2,0);

glVertex3d(0,2.9,0);

glVertex3d(-1,2,0);

glVertex3d(0,1.1,0);

glEnd();

glEnable(GL\_LINE\_STIPPLE);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glBegin(GL\_TRIANGLE\_FAN);

glColor3d(0,0,1);

glVertex3d(4,2,0);

glVertex3d(2.6,2.8,0);

glVertex3d(2,2,0);

glVertex3d(3,1.1,0);

glEnd();

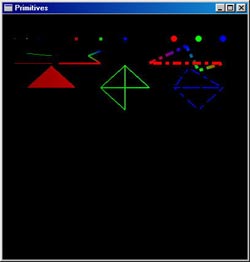
glDisable(GL\_LINE\_STIPPLE);

**3.5 Четырехугольники и многоугольники**

Четырехугольники рисуются вызовом функции glBegin с параметром GL\_QUADS или GL\_QUAD\_STRIP. Для первого случая каждые четыре вершины определяют свой четырехугольник. Во втором случае рисуются связанные четырехугольники. Первая, вторая, третья и четвертая вершина определяют первый четырехугольник. Третья, четвертая, пятая и шестая вершина - второй четырехугольник и т.д. (2n-1), 2n, (2n+1) и (2n+2) вершины задают n-ый четырехугольник. Многоугольники задаются вызовом glBegin с параметром GL\_POLYGON. Все вершины определяют один многоугольник. Для многоугольников можно задавать стили при помощи выше описанной функции glPolygonMode, толщину линии, толщину точек и цвет.

**3.6 Упражнение: "Примитивы"**

Изобразите точки, линии, треугольники, многоугольники в одном окне, как показано ниже.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter3/primitive/primitive.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter3/primitive/release/primitive.exe)  .

**3.7 Упражнение: "Многогранники"**

Реализуйте проволочные многогранники с помощью проволочных треугольников, многоугольников и линий.

**3.8 Резюме**

Ну вот, вы еще на один шаг продвинулись в изучение библиотеки OpenGL. Теперь вы имеете представление о том, как рисовать элементарные фигуры. Из примитивов вы можете составить фигуры любой сложности.

**OpenGL - Полезные и бесполезные мелочи**

**4.1 Построение поверхностей**

Существует набор функций для построения сфер, цилиндров и дисков. Эти функции представляют очень мощный контроль за построением трехмерных объектов. Непосредственно рисовать вы будете, используя следующие функции: gluSphere, gluCylinder, gluDisk и gluPartialDisk. В начале книги вы научились строить трехмерные объекты с помощью функций из библиотеки Auxilary Library. Функции aux[Solid/Wire]Sphere, aux[Solid/Wire]Cylinder и aux[Solid/Wire]Cone просто вызывают gluSphere и gluCylinder. Как я уже ранее говорил, в aux[Solid/Wire]Cylinder и aux[Solid/Wire]Cone фирмы Microsoft имеются баги. Здесь будет подробно рассмотрено построение сфер и цилиндров, так что потребность в aux[Solid/Wire]Cylinder и aux[Solid/Wire]Cone отпадет.

Первым параметром для gluSphere, gluCylinder, gluDisk является указатель на объект типа GLUquadricObj. Далее следуют параметры непосредственно создаваемой фигуры. Для сферы - это радиус; для цилиндра - радиус нижнего основания, радиус верхнего основания и высота; для диска - внутренний радиус, внешний радиус и для частичного диска - внутренний радиус, внешний радиус, угол, с которого начинать рисовать, длина дуги в градусах, которую рисовать. Последние два параметра у всех этих функций одинаковы. Это число разбиений вокруг оси Z и число разбиений вдоль оси Z. Как я уже говорил, все сложные объекты состоят из простых: точек, линий и многоугольников. Вы понимаете, что нарисовать/создать идеально гладкую сферу или цилиндр невозможно. Поэтому строится приближение из плоскостей. Для этого и нужно указать количество разбиений. Чем больше разбиение, тем лучше будет выглядеть ваша сфера. Однако, задавать здесь число с шестью нулями не стоит. Это лишено всякого смысла. Оптимальным, на мой взгляд, является число от 10 до 20. Чем больше объект, тем больше нужно разбиений. Число разбиений вдоль и поперек я выставляю одинаковыми. Сначала вы должны создать объект типа GLUquadricObj с помощью функции gluNewQuadric. Теперь устанавливаете свойства с помощью функции gluQuadricDrawStyle. Доступны стили: GLU\_FILL - рисуется сплошной объект, GLU\_LINE - проволочный объект, GLU\_POINT - рисуются только точки. Рисуете то, что хотите. И не забудьте удалить созданный объект, воспользовавшись gluDeleteQuadric.

Настало время поэкспериментировать. Создайте очередное консольное приложение и отредактируйте функцию display следующим образом.

void CALLBACK display(void)

{

GLUquadricObj \*quadObj;

quadObj = gluNewQuadric(); // создаем новый объект

// для создания сфер и цилиндров

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glPushMatrix();

glColor3d(1,0,0);

gluQuadricDrawStyle(quadObj, GLU\_FILL); // устанавливаем

// стиль: сплошной

gluSphere(quadObj, 0.5, 10, 10); // рисуем сферу

// радиусом 0.5

glTranslated(-2,0,0); // сдвигаемся влево

glRotated(45, 1,0,0); // поворачиваем

glColor3d(0,1,0);

gluQuadricDrawStyle(quadObj, GLU\_LINE); // устанавливаем

// стиль: проволочный

gluCylinder(quadObj, 0.5, 0.75, 1, 15, 15);

glPopMatrix();

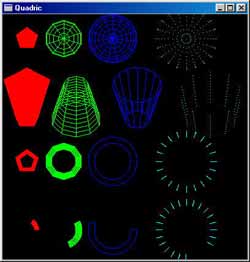
gluDeleteQuadric(quadObj);

auxSwapBuffers();

}

**4.2 Упражнение: "Сфера, цилиндр и диски"**

Доработайте код, приведенный выше, чтобы в первой строке показывались три сферы. Цвет и стили(GLU\_POINT, GLU\_LINE и GLU\_FILL) должны быть разными. В следующих трех строках должны быть цилиндры, диски и частичные диски.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/quadric/quadric.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/quadric/release/quadric.exe)  .

**4.3 Интерполяция цветов**

Когда вы создаете многоугольник, вы можете задать цвет для каждой его вершины. Если разрешено сглаживание цветов, то многоугольник будет переливаться. Поясню на примере. Режим сглаживания по умолчанию разрешен. Он переключается функцией glShadeModel с аргументами GL\_FLAT и GL\_SMOOTH. GL\_FLAT запрещает сглаживание. На мой взгляд, сглаживание редко нужно. Вещь красивая, но бесполезная. Я в своих неучебных программах этот режим никогда не использовал. Поэтому советую его отключать, особенно, при программировании анимированного приложения. Создайте очередной проект. В функцию main добавьте

glShadeModel(GL\_SMOOTH);

Функцию display отредактируйте следующим образом:

void CALLBACK display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glBegin(GL\_TRIANGLES);

glColor3d(1,0,0);

glVertex2d(0,0);

glColor3d(0,1,0);

glVertex2d(0,3);

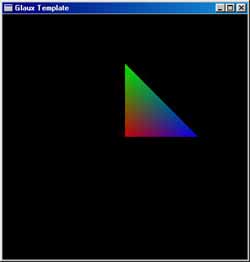
glColor3d(0,0,1);

glVertex2d(3,0);

glEnd();

auxSwapBuffers();

}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/shademodel/shademodel.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/shademodel/release/shademodel.exe)  .

**4.4 Прозрачность**

С помощью четвертого компонента цвета можно получать различные эффекты наложения объектов друг на друга, наложения цветов и т.п. Здесь я расскажу о наиболее нужном и распространенном эффекте - прозрачности объектов. Для того, чтобы разрешить обрабатывать четвертый компонент цвета вы должны вызвать функцию glEnable с аргументом GL\_ALPHA\_TEST. Для получения требуемого эффекта прозрачности нужно разрешить наложение цветов - glEnable(GL\_BLEND). И установить алгоритм, по которому будут смешиваться два цвета - glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA). Учтите, что эти режимы очень затормаживают вывод изображения, поэтому я не рекомендую устанавливать эти режимы глобально, для воспроизведения всех объектов. Выделите из ваших объектов те, которым требуется этот режим, включайте и отключайте его своевременно. Именно поэтому, эти тесты я разместил в функции display. Создайте новый проект с именем transperence и отредактируйте функцию display, как показано ниже.

void CALLBACK display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glEnable(GL\_ALPHA\_TEST);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glColor4d(1,0,0, 1);

auxSolidSphere( 1 );

glColor4d(0,1,0, 0.6);

auxSolidCylinder(2,3);

glDisable(GL\_BLEND);

glDisable(GL\_ALPHA\_TEST);

auxSwapBuffers();

}

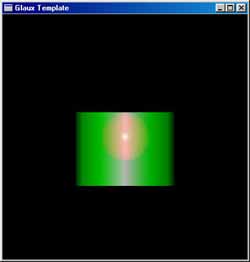
В заключение, добавлю, что если вы поменяете местами создание цилиндра и сферы, то никакого эффекта прозрачности не увидите. Это не баг, а фича - так задумано. Секрет прост: вы должны сначала рисовать дальние объекты, а потом ближние. Связано это с тем, что выполняется тест глубины. Теперь смотрите, что получается. Если вы сначала рисуете сферу(она удалена), а потом цилиндр, то выполняется это следующим образом.

1. Создаем сферу.
2. Выполняется тест глубины успешно, т.к. цилиндра пока нет.
3. В буфере рисуется сфера.
4. Создаем цилиндр.
5. Выполняется тест глубины успешно, т.к. стенка цилиндра ближе, чем сфера.
6. В буфере рисуется цилиндр, он закрывает сферу с учетом прозрачности.

Теперь смотрим, что происходит, если сначала нарисовать цилиндр, потом сферу.

1. Создаем цилиндр.
2. Выполняется тест глубины успешно.
3. В буфере рисуется цилиндр.
4. Создаем сферу.
5. Выполняется тест глубины аварийно, т.к. сфера создается за цилиндром.
6. В буфере ничего не рисуется.

В последнем случае вы увидите один цилиндр.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/transparence/transparence.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/transparence/release/transparence.exe)  .

**4.5 Упражнение: "Снег"**

Ранее рассматривалось приложение "Снеговик". Анимацию создавать вы тоже уже научились. Добавьте косой снег, только сделайте снеговика прозрачным, чтобы снежинки как бы пролетали сквозь него.

**4.6 Плоскости отсечения**

Если вам требуется нарисовать сферу или любой другой объект урезанным, то плоскости отсечения это то, что вам надо. Плоскостей отсечения может быть шесть штук. По умолчанию они все запрещены. Плоскость отсечения включается командой glEnable(GL\_CLIP\_PLANE0). Ноль на конце GL\_CLIP\_PLANE означает нулевую плоскость; можно указать один, два, три и т.д. Сама плоскость устанавливается функцией glClipPlane. Первый аргумент этой функции - это плоскость, второй - это массив из четырех элементов, в котором хранятся коэффициенты уравнения плоскости. Для тех, кто не помнит уравнения плоскости в трехмерном пространстве, напоминаю: A\*x+B\*y+C\*z+D = 0. Вот эти самые A,B,C,D и являются теми четырьмя коэффициентами. Создайте новый проект с именем ClipPlane и отредактируйте функцию display, как показано ниже.

void CALLBACK display(void)

{

GLdouble equation[4] = {-1,-0.25,0,2};

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glEnable(GL\_CLIP\_PLANE0);

glClipPlane(GL\_CLIP\_PLANE0, equation);

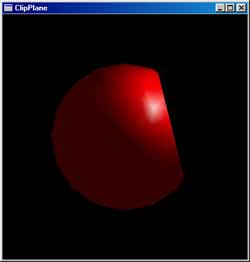
glColor3d(1,0,0);

auxSolidSphere( 3 );

glDisable(GL\_CLIP\_PLANE0);

auxSwapBuffers();

}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/clipplane/clipplane.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/clipplane/release/clipplane.exe)  .

**4.7 Упражнение: "Три плоскости"**

Добавьте еще две плоскости. Расположите их так, чтобы они отсекали двумерный угол [(p)/2].

**4.8 Трафарет I**

Зачем это может понадобиться, я не очень представляю. Рисовать прерывистую линию вы уже умеете. Трафарет - это то же самое, только для многоугольников. Вы разрешаете тест трафарета командой glEnable(GL\_POLYGON\_STIPPLE). Аналогично, как и в случае с линями, нужно загружать массив, который задает битовую маску. Размер трафарета строго оговорен - 32х32 пикселя. 32х32 равняется 1024. 1024 делим на восемь бит, получаем 128 байт, т.е., чтобы закодировать трафарет, нужен массив из 128-ми байт. Тут уже, как в случае с линиями, калькулятором не посчитаешь. Поэтому я написал утилиту - pcx\_2bpp, которая конвертирует pcx-файл в формате два бита на пиксель в массив на языке Си. Запускается она так:

>pcx\_2bpp filename.pcx >array.c

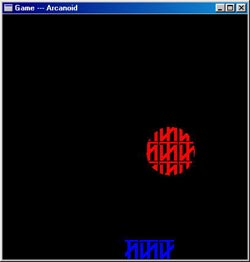
Утилиту можете взять на моем сервере в архиве с исходными текстами программ(см. приложения А). Знак "больше" означает перенаправление стандартного вывода в файл array.c. Эта утилита абсолютно переносима. Нарисовав в каком-нибудь графическом пакете рисунок в формате 32х32х2bpp, сохраните его и обработайте моей утилитой. Получите массив, который вставьте в свою программу. В функции main добавьте следующий код:

glEnable(GL\_POLYGON\_STIPPLE);

glPolygonStipple(array);

**4.9 Упражнение: "Совершенствуем Arcanoid"**

Не знаю, правильно ли сказал. Мне кажется, что уродуем, а не совершенствуем. В общем, наложите трафарет на подставку и шар. Когда увидите трафарет на шаре, то поймете, что "Уродуем".



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/arcanoid/arcanoid.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/arcanoid/release/arcanoid.exe)  .

**4.10 Туман**

Еще одна мелочь, которую мы рассмотрим - это включение тумана. За основу возьмем нашего снеговика. Надо сказать, что ведет туман себя довольно странно. Если его цвет установить белым, то при увеличении плотности тумана снеговик становится полностью белым, причем даже те его места, которые изначально были черными, т.к. не освещались. Когда я установил цвет тумана темно-серым - (0.25, 0.25, 0.25), то при увеличении плотности не освещенные его части как им и положено оставались черными, а видимые теряли яркость. Я скорее охарактеризовал бы средство тумана, как средство для изменения яркости, потому как на реальный туман, это к сожалению, не похоже. Итак, создайте новый проект с именем fog, скопируйте snowman.c в fog.c. Для того чтобы включить туман и поэкспериментировать с его влиянием на изображение мы добавим две функции, которые будут обрабатывать события от стрелок вверх\вниз на клавиатуре. Код этих функций выглядит так, включите его сразу после включения заголовочных файлов:

float density;

void CALLBACK Key\_UP(void )

{

density+=(float)0.1;

glFogf(GL\_FOG\_DENSITY, density);

}

void CALLBACK Key\_DOWN(void )

{

density-=(float)0.1;

glFogf(GL\_FOG\_DENSITY, density);

}

Я также ввел глобальную переменную density, в ней хранится плотность тумана. При нажатие стрелок вверх\вниз будет вызываться соответствующая функция и будет изменено значение тумана. Снеговик у нас все время перерисовывается, поэтому при следующей отрисовке кадра значение тумана обновится. Теперь надо отредактировать функцию main, добавленный код выделен серым фоном:

void main()

{

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

float fogcolor[4] = {0.25,0.25,0.25,1}; // цвет тумана

auxInitPosition( 50, 10, 400, 400);

auxInitDisplayMode( AUX\_RGB | AUX\_DEPTH | AUX\_DOUBLE );

auxInitWindow( "Snowman" );

auxIdleFunc(display);

auxReshapeFunc(resize);

auxKeyFunc(AUX\_UP, Key\_UP); // устанавливаем обработчик

auxKeyFunc(AUX\_DOWN, Key\_DOWN); // стрелок вверх/вниз

glEnable(GL\_FOG); // разрешаем туман

glGetFloatv(GL\_FOG\_DENSITY, &density); // получаем значение плотности

glFogfv(GL\_FOG\_COLOR, fogcolor); // устанавливаем цвет тумана

glEnable(GL\_ALPHA\_TEST);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

auxMainLoop(display);

}

Я полагаю здесь все ясно, прокомментирую тут только одну функцию:

glGetFloatv(GL\_FOG\_DENSITY, &density);

С помощью этой функции я получаю значение плотности тумана, установленное по умолчанию. Первый аргумент этой функции указывает OpenGL значение какого параметра мы хотим получить. Второй аргумент - это адрес в памяти куда будет записано значение данного параметра.

**4.11 Логические операции**

Логические операции позволяют вам складывать цвет фрагмента находящегося в буфере с цветом, который туда поступает. Этот режим разрешается и запрещается вызовом функций glEnable\glDisable с параметром GL\_COLOR\_LOGIC\_OP. Правило, по которому будут складываться цвета задается функцией glLogicOp. У нее один параметр - одна из следующих констант определенных в файле gl.h.

/\* LogicOp \*/

#define GL\_CLEAR 0x1500

#define GL\_AND 0x1501

#define GL\_AND\_REVERSE 0x1502

#define GL\_COPY 0x1503

#define GL\_AND\_INVERTED 0x1504

#define GL\_NOOP 0x1505

#define GL\_XOR 0x1506

#define GL\_OR 0x1507

#define GL\_NOR 0x1508

#define GL\_EQUIV 0x1509

#define GL\_INVERT 0x150A

#define GL\_OR\_REVERSE 0x150B

#define GL\_COPY\_INVERTED 0x150C

#define GL\_OR\_INVERTED 0x150D

#define GL\_NAND 0x150E

#define GL\_SET 0x150F

Мне не удалось найти разумного применения этой функции поэтому я приведу здесь лишь текст моей программы с комментариями, которая позволяет перебрать и посмотреть действие всех логических операций. Как и в предыдущем пункте у меня определена глобальная переменная и две функции реагирующие на нажатие стрелок на клавиатуре.

/\*

\* (c) Copyright 1995-2000, Igor Tarasov

\* FidoNet: 2:5020/370.2 620.20 1103.5

\* email: igor@itsoft.miem.edu.ru itarasov@rtuis.miem.edu.ru

\* Phone: (095)916-89-51 916-89-63

\*/

#include

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glaux.h>

int logicop = GL\_CLEAR;

void CALLBACK Key\_UP(void )

{

if(logicop<GL\_SET)

logicop++;

}

void CALLBACK Key\_DOWN(void )

{

if(logicop>GL\_CLEAR)

logicop--;

}

void CALLBACK resize(int width,int height)

{

glViewport(0,0,width,height);

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

glOrtho(-5,5, -5,5, 2,12);

gluLookAt( 0,0,5, 0,0,0, 0,1,0 );

glMatrixMode( GL\_MODELVIEW );

}

void CALLBACK display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glColor4d(1,0,0, 1);

auxSolidSphere( 1 );

glEnable(GL\_COLOR\_LOGIC\_OP);

glLogicOp(logicop);

glColor4d(0,1,0, 0.7);

auxSolidCylinder(2,3);

glDisable(GL\_COLOR\_LOGIC\_OP);

auxSwapBuffers();

}

void main()

{

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

GLfloat mat\_specular[] = { 1,1,1,1};

auxInitPosition( 50, 10, 400, 400);

auxInitDisplayMode( AUX\_RGB | AUX\_DEPTH | AUX\_DOUBLE );

auxInitWindow( "Glaux Template" );

auxIdleFunc(display);

auxReshapeFunc(resize);

auxKeyFunc(AUX\_UP, Key\_UP);

auxKeyFunc(AUX\_DOWN, Key\_DOWN);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

// glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

// glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialf(GL\_FRONT, GL\_SHININESS, 128);

auxMainLoop(display);

}

**4.12 Трафарет II**

Ранее вы познакомились с наложением трафарета состоящего из массива бит 32х32 точки. Недостатки такого трафарета в его небольшой площади и то, что каждые его элемент является битом, т.е. у такого трафарета всего две зоны(рисуем и не рисуем). Далее вы познакомитесь с полноценным трафаретом.

В OpenGL есть буфер трафарета, который предоставляет огромные возможности для творчества. Тест трафарета на мой взгляд бесспорно полезная вещь. С ее помощью реализуются самые разнообразные эффекты, начиная от простого вырезания одной фигуры из другой до реализации теней, отражений и прочих нетривиальных функций, требующих от вас уже не только знакомство с библиотекой OpenGL, но и понимания алгоритмов машинной графики. Здесь мы рассмотрим самое простое применение буфера трафарета. Пусть у нас есть два объекта на экране - сфера и куб, и пусть сфера находится внутри куба и немного из него выходит за его грани.

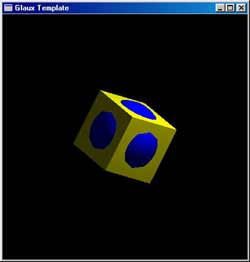


рис. 4.12.1

Мы поставим себе задачу изобразить три рисунка: куб минус сфера, сфера минус куб, пересечение куба и сферы. Как вы понимаете объединение куба и сферы выводится без всякого буфера трафарете по умолчанию. Собственно говоря, оно(объединение) изображено на рисунке 4.12.1. Теперь о том, что такое трафарет и как им пользоваться. Трафарет это двумерный массив целых переменных(тип int). Каждому пикселю в окне соответствует один элемент массива. Использование буфера трафарета происходит в два этапа. Сначала вы его заполняете, потом основываясь на его содержимом рисуете ваши объекты. Буфер трафарета заполняется следующим образом. Вы делите окно вывода на зоны и каждой зоне присваиваете свое значение. Например, для рисунка 4.12.1 область, где нет ничего будет заполнена нулями, область, где выведен куб заполнена единицами и область, где видна сфера двойками. Обратите внимание, что буфер трафарета - это двумерный массив, а не трехмерный. Теперь вы легко можете представить себе эти области, они собственно и изображены на приведенном рисунке. Также заметьте, что цвет здесь роли не играет. Я бы мог вывести все черным цветом, а буфер трафарета заполнился в зависимости от геометрических размеров фигур и от их пересечения. Далее мы рассмотрим функции библиотеки OpenGL для работы с трафаретом. Тест трафарета разрешается при помощи функций glEnable\glDisable с параметром GL\_STENCIL\_TEST. Очищается буфер трафарета при помощи функции glClear с параметром GL\_STENCIL\_BUFFER\_BIT. Заполнение буфера трафарета происходит при помощи следующих двух функций:

void glStencilFunc(GLenum func, GLint ref, GLuint mask)

void glStencilOp(GLenum fail, GLenum zfail, GLenum zpass)

Первая функция задает правило, по которому будет определяться пройден тест трафарета или нет. Переменная func может принимать одно из следующих значений:

|  |  |
| --- | --- |
| GL\_NEVER | Не проходит |
| GL\_LESS | Проходит if ( ref & mask) < ( stencil & mask) |
| GL\_LEQUAL | Проходит if ( ref & mask) ? ( stencil & mask) |
| GL\_GREATER | Проходит if ( ref & mask) > ( stencil & mask) |
| GL\_GEQUAL | Проходит if ( ref & mask) ? ( stencil & mask) |
| GL\_EQUAL | Проходит if ( ref & mask) = ( stencil & mask) |
| GL\_NOTEQUAL | Проходит if ( ref & mask) ? ( stencil & mask) |
| GL\_ALWAYS | Всегда проходит |

Если тест трафарета не пройден, то фрагмент (пиксели) фигуры не прорисовываются в данном месте, т.е. они не попадают в буфер кадра. Если тест пройден, то фигура рисуется. Вторая функция позволяет задать, как будет инициализироваться буфер трафарета. Параметры fail, zfail и zpass могут принимать одно из следующих значений:

|  |  |
| --- | --- |
| GL\_KEEP | Сохранить текущее значение в буфере трафарета |
| GL\_ZERO | Установить значение буфера трафарета в ноль |
| GL\_REPLACE | Заменить значение буфера трафарета на значение переменной ref, заданной функцией glStencilOp |
| GL\_INCR | Увеличить на единицу |
| GL\_DECR | Уменьшить на единицу |
| GL\_INVERT | Поразрядно инвертировать |

В случае непрохождения теста трафарета над фрагментом выполняется действие определенной параметром fail. Например, если мы хотим заполнить область трафарета, где рисуется куб единицами, то можно использовать следующий код:

glStencilFunc(GL\_NEVER, 1, 0); // значение mask не используется

glStencilOp(GL\_REPLACE, GL\_KEEP, GL\_KEEP);

auxSolidCube(2.5);

Объясняю подробней, первая функция говорит о том, что тест трафарета всегда проходит неудачно. Вторая функция задает, что в случае неудачного теста трафарета заменить значение хранящееся в буфере трафарета на значение переменной ref, а его мы задали равным единице. В результате, на экране ничего не нарисуется т.к. тест трафарета завершался неудачно, но в буфере трафарета мы получили проекцию куба из единичек, т.е. буфер трафарета заполнен не только нулями. Теперь мы хотим заполнить двойками область, где прорисовывается сфера. Здесь мы уже должны учитывать буфер глубины, иначе мы заполним двойками всю область, где у нас рисуется сфера. Для того чтобы учитывать буфер глубины тест трафарета должен завершиться положительно. Третий параметр zpass функции glStencilOp как раз указывает, что делать, если тест трафарета прошел, а тест глубины нет. Поэтому код выглядит так:

glStencilFunc(GL\_ALWAYS, 2, 0); // значение mask не используется

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_REPLACE);

auxSolidSphere(1.5);

В результате получаем буфер трафарета заполненный нулями, где ничего не было, единицами, где виден куб и двойками, где видна сфера. В последнем примере тест трафарета прошел успешно, поэтому на экране была нарисована сфера. Но это нам не мешает мы очистим буфер глубины и буфер цвета, но не буфер трафарета.

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

Теперь дело техники нарисовать тот же самый куб без сферы. Надо установить, что тест трафарета проходит, если значение находящееся в буфере трафарета совпадает со значением второго параметра функции glStencilFunc. Как вы помните куб в буфере трафарета имел значение единицы. Поэтому получаем:

glStencilFunc(GL\_EQUAL, 1, 255); // куб рисуется только там, где

// в буфере трафарета лежат единицы

glColor3d(1,1,1);

auxSolidCube(2.5);

Создайте новый проект с именем stencil. Скопируйте glaux.c в stencil.c и отредактируйте функцию display следующим образом:

void CALLBACK display(void){

// очищаем все буферы

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT);

// разрешаем тест трафарета

glEnable(GL\_STENCIL\_TEST);

// рисуем куб и заполняем буфер трафарета единицами

// в том месте, где рисуется куб

// тут у меня немного по другому, чем я выше было разобрано,

// но действие выполняется аналогичное

glStencilFunc(GL\_ALWAYS, 1, 0);

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_REPLACE);

auxSolidCube(2.5);

// заполняем буфер трафарета двойками

// в том месте, где сфера закрывает куб

glStencilFunc(GL\_ALWAYS, 2, 0);

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_REPLACE);

auxSolidSphere(1.5);

// очищаем буферы цвета и глубины

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glStencilFunc(GL\_EQUAL, 1, 255);

glColor3d(1,1,1);

auxSolidCube(2.5);

// вращаем сцену

glRotated(3, 1,0,0);

glRotated(5, 0,1,0);

glRotated(7, 0,0,1);

auxSwapBuffers();}



рис. 4.12.2

Теперь давайте немного приукрасим нашу программу. Давайте внутри куба нарисуем красный шар, который будут опоясывать зеленый и синий тор. Делается это следующим образом. Как и в предыдущей программе вы составляете буфер трафарета, после чего очищаете буфер глубины и цвета, запрещаете тест трафарета, т.к. нам надо просто вывести фигуры без трафарета, выводите шар с торами, и, наконец, включает тест трафарета и выводите куб с отверстиями. Вот код функции display:

void CALLBACK display(void){

// очищаем все буферы

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT);

// разрешаем тест трафарета

glEnable(GL\_STENCIL\_TEST);

// рисуем куб и заполняем буфер трафарета единицами

// в том месте, где рисуется куб

// тут у меня немного по другому, чем я выше было разобрано,

// но действие выполняется аналогичное

glStencilFunc(GL\_ALWAYS, 1, 0);

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_REPLACE);

auxSolidCube(2.5);

// заполняем буфер трафарета двойками

// в том месте, где сфера закрывает куб

glStencilFunc(GL\_ALWAYS, 2, 0);

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_REPLACE);

auxSolidSphere(1.5);

// очищаем буферы цвета и глубины

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// запрещаем тест трафарета и рисуем красную сферу

glDisable(GL\_STENCIL\_TEST);

glColor3d(1,0,0);

auxSolidSphere(0.5);

//синий тор

glColor3d(0,0,1);

auxSolidTorus(0.15, 0.6);

// зеленый тор, повернутый на 90 градусов относительно синего

glColor3d(0,1,0);

glPushMatrix();

glRotated(90, 1,0,0);

auxSolidTorus(0.15, 0.6);

glPopMatrix();

//снова разрешаем тест трафарета

glEnable(GL\_STENCIL\_TEST);

glStencilFunc(GL\_EQUAL, 1, 255);

glColor3d(1,1,1);

auxSolidCube(2.5);

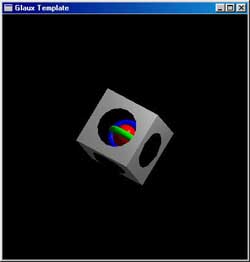
// вращаем сцену

glRotated(3, 1,0,0);

glRotated(5, 0,1,0);

glRotated(7, 0,0,1);

auxSwapBuffers();}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/stencil/stencil.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter4/stencil/release/stencil.exe)  .

**4.13 Упражнение "сфера минус куб"**

Напишите программу, в которой будет крутиться сфера минус куб.

**4.14 Упражнение "пересечение сферы и куба"**

Напишите программу, в которой будет крутиться пересечение сферы и куба.

**5.1 Общие слова**

Строить примитивные объекты вы уже научились. Но строить трехмерные сцены с использованием только примитивов вам вряд ли придется, да и выглядят они как-то схематично и тускло. Для того, чтобы их оживить, на примитивы накладывают картинки - текстуры. В качестве фона сцены тоже полезно использовать графическое изображение. Тем самым приложение сделается более живым и интересным. Так что в этой главе мы научимся работать с изображениями.

**5.2 Работа с изображениями**

Существует множество графических форматов - bmp, pcx, gif, jpeg и прочие. OpenGL напрямую не поддерживает не один из них. В OpenGL нет функций чтения/записи графических файлов. Но поддерживается работа с массивами пикселей. Вы загружаете графический файл, используя библиотеки других фирм, в память и работаете с ними средствами OpenGL. В массиве данные о пикселях могут располагаться разными способами: RGB, BGR, RGBA; могут присутствовать не все компоненты; каждый элемент массива может занимать один байт, два, четыре или восемь; выравнивание может быть по байту, слову или двойному слову. В общем, форматов расположения данных о графическом изображении в памяти очень много. Я рассмотрю один из них, наиболее часто применяемый, как мне кажется. Информация о каждом пикселе хранится в формате RGB и занимает три байта, выравнивание по байту. В Auxiliary Library есть функция auxDIBImageLoad(LPCSTR), которая загружает в память bmp-файл и возвращает указатель на структуру:

typedef struct \_AUX\_RGBImageRec {

GLint sizeX, sizeY;

unsigned char \*data;

} AUX\_RGBImageRec;

Для простоты я буду пользоваться именно этой функцией. Среди прилагаемых программ вы найдете мою утилиту для загрузки файлов из форматов pcx. Исходный текст этой утилиты абсолютно переносим на любую платформу с компилятором ANSI C.

В OpenGL имеются функции для вывода массива пикселей на экран(glDrawPixels), копирования(glCopyPixels), масштабирования(gluScaleImage). Здесь мы рассмотрим только glDrawPixels. Все остальные функции работы с изображениями устроены похожим образом. Для того, чтобы отобразить графический файл в окне OpenGL, вы должны загрузить его в память, указать выравнивание, установить точку, с которой начинается вывод изображения, и вывести его на экран. Раздобудьте где-нибудь свою фотографию в формате BMP. Можете взять фотографию своей девушки. Создайте новый проект. Объявите глобальную переменную -

AUX\_RGBImageRec \*image

Перед вызовом функции auxMainLoop в функции main вставьте строку:

image = auxDIBImageLoad("photo.bmp");

Выравнивание устанавливается вызовом функции glPixelStorei с параметром GL\_UNPACK\_ALIGNMENT и вторым параметром - целым числом, которое указывает выравнивание. Изображения выводятся прямо на экран. Поэтому все происходит в двухмерных координатах. Позиция, с которой начинается вывод изображения, указывается при помощи функции glRasterPos2d(x,y). Также вы можете установить размер пикселя, вызвав функцию glPixelZoom. Первый параметр этой функции - ширина, второй - высота пикселя. Я вызываю эту функцию с аргументами (1,1), что соответствует нормальному пикселю. Замените (1,1) на (3,2) и вы увидите, как картинка растянется в три раза по горизонтали и в два раза по вертикали. Это случилось, потому что теперь каждый пиксель изображения соответствует прямоугольнику 3х2 в окне. И наконец, вывод осуществляет функция glDrawPixels. Первые два параметра - это ширина и высота. Далее, вы указываете формат, в котором хранится информация в памяти, и тип элементов массива. Последним указывается массив данных. В функцию display вставьте следующий код:

glRasterPos2d(-4.5,-3); // нижний левый угол

glPixelZoom(1,1);

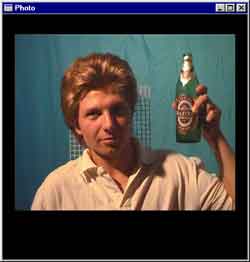
glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1); // выравнивание

glDrawPixels(image->sizeX, image->sizeY, // ширина и высота

GL\_RGB, GL\_UNSIGNED\_BYTE, // формат и тип

image->data); // сами данные

Также в OpenGL имеется функция glBitmap для отображения битовых массивов. Битовый массив - это последовательность байт, которые кодируют картинку из двух цветов. Соответственно, каждый байт кодирует 8 пикселей. Среди прилагаемых программ вы найдете мою утилиту pcx\_2bpp. Она читает pcx-файл формата один бит на пиксель и направляет на стандартный вывод массив на языке Си.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/photo/photo.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/photo/release/photo.exe)  . Моя фотография [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/igor.bmp)  .

**5.3 Упражнение: "Фон для игры Arcanoid"**

Найдите красивый фон в формате BMP. Можете опять взять свою фотографию. Подложите этот bmp-файл в качестве фона в игру Arcanoid. Закомментируйте разрешение всех тестов, кроме GL\_DEPTH\_TEST, в функции main. Возможно, я уже говорил о том, что вы всегда должны помнить, что дополнительные параметры затормаживают создание объекта, поэтому устанавливайте их очень осторожно. Часть из них можно установить в функции main. Другие же лучше устанавливать и отменять непосредственно при создании объекта в функции display.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/arcanoid/arcanoid.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/arcanoid/release/arcanoid.exe)  . Звездное небо [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/space.bmp)  .

**5.4 Создаем текстуру в памяти**

Одного вывода изображений недостаточно для создания полноценных трехмерных сцен. Часто возникает потребность накладывать изображение на трехмерные объекты и поворачивать/сдвигать их. Для этих целей существую текстуры. Также текстуры помогут вам покрыть весь объект в виде мозаики. Скажем, когда у вас имеется кирпичная стена, то вам не надо загружать изображение с кучей кирпичей. Достаточно загрузить один кирпич и указать, что эту текстуру нужно размножить по всей плоскости.

Сначала мы разберем создание и наложение текстур на плоскость. Затем рассмотрим наложение текстур на объекты, описанные в секции 4.1. И наконец, на все прочие, созданные из многоугольников; в частности, тор и чайник.

Для того, чтобы наложить текстуру на объект, вы должны:

1. Загрузить графический файл в память
2. Создать имя-идентификатор текстуры
3. Сделать его активным
4. Создать саму текстуру в памяти
5. Установить параметры текстуры
6. Установить параметры взаимодействия текстуры с объектом
7. Связать координаты текстуры с объектом

Первое вы уже научились делать в предыдущей секции. Создайте проект с именем Texture. Объявите следующие глобальные переменные:

unsigned int photo\_tex;

AUX\_RGBImageRec\* photo\_image;

unsigned int space\_tex;

AUX\_RGBImageRec\* space\_image;

Переменные photo\_tex и space\_tex будут служить идентификаторами текстур. А в photo\_image и space\_image мы загрузим bmp-файлы. Тут нужно отметить, что текстуры в OpenGL должны иметь размер 2n x 2m, где n и m целые числа. Это сделано для ускорения работы, т.к. сжимать или растягивать такие текстуры быстрее и удобней. Вы, конечно, можете загрузить изображение любого другого размера, но его придется масштабировать. На мой взгляд, это неправильно. Результат масштабирования вас может и не устроить. Так что я использую графические файлы с размером, кратным степени двойки. Мне удобнее отредактировать изображение в каком-нибудь графическом пакете, урезать его или наоборот дополнить, чем потом выяснять, почему оно искажается. Впрочем, тут многое зависит от конкретного случая. Художнику, который делает текстуры, все равно какого размера ее делать, поэтому легче попросить его сделать изображение с подходящими размерами. Вставьте следующий код в функцию main.

photo\_image = auxDIBImageLoad("photo.bmp");

space\_image = auxDIBImageLoad("space.bmp");

Картинки возьмите из моей программы - Texture. Фотографию можете взять свою.;-) Только размер ее желательно оставить 512x512.

Теперь вы должны создать имя-идентификатор текстуры. Его нужно создавать, когда у вас в приложении используется более одной текстуры, чтобы была возможность как-то их различать. В противном случае, когда текстура только одна, идентификатор ей не нужен. В следующем примере, при наложение текстуры на сферу, у нас будет ровно одна текстура, и я покажу вызовы каких функций необязательны. А пока, я предполагаю, что вам нужно использовать несколько текстур. Кстати, я просматривал много примеров при написание книги. Среди них были примеры из широко известной Red Book, примеры из MSDN, из интернета и других источников, но все, что касалось текстур, работало только с одной текстурой. Для элементарной программы-примера, конечно, подойдет и одна текстура, а вот для серьезных приложений вряд ли. Мы хотим написать серьезные приложения, поэтому нам придется использовать нескольких текстур. Функция glGenTextures принимает на вход два параметра. Первый указывает количество имен-идентификаторов текстур, которые нужно создать. Второй параметр - указатель на массив элементов типа unsigned int. Количество элементов в массиве должно совпадать с числом, указанным в качестве первого параметра. Например, следующий код создает десять имен текстур.

unsigned int names[10];

glGenTetures(10, names);

Хранить идентификаторы текстур в массиве не всегда удобно. Такой способ подходит для хранения задних фонов или типов стен - кирпичная, каменная и т.п. В общем, в массиве хранят те элементы, между которыми есть что-то общее. В нашем случае, два изображения связаны, т.к. используются в одном приложении, поэтому я создал два различных идентификатора. Так что добавьте следующий код в функцию main.

glGenTextures(1, &photo\_tex);

glGenTextures(1, &space\_tex);

Теперь мы привязываемся к текстуре фотографии, т.е. делаем ее активной. Для этого служит функция glBindTexture. Первый параметр должен быть GL\_TEXTURE\_2D или GL\_TEXTURE\_1D. Он показывает, с одномерным или двумерным изображением будем работать. Все примеры здесь касаются двумерных текстур. Для одномерной текстуры я просто не нашел красивого примера. Впрочем, в Red Book есть пример с одномерной текстурой. Там чайник разукрашивают красной лентой. Где взять эти примеры и многое другое смотрите в приложении 'A'. Второй параметр glBindTexture - идентификатор, который мы создали выше при помощи glGenTextures. Теперь добавьте вызов этой функции в main.

glBindTexture(GL\_TEXTURE\_2D, photo\_tex);

Теперь мы должны создать саму текстуру в памяти. Массив байт в структуре AUX\_RGBImageRec не является еще текстурой, потому что у текстуры много различных параметров. Создав текстуру, мы наделим ее определенными свойствами. Среди параметров текстуры вы указываете уровень детализации, способ масштабирования и связывания текстуры с объектом. Уровень детализации нужен для наложения текстуры на меньшие объекты, т.е. когда площадь на экране меньше размеров изображения. Нулевой уровень детализации соответствует исходному изображению размером 2nx2m, первый уровень - 2n-1x2m-1, k-ый уровень - 2n-kx2m-k. Число уровней соответствует min(n,m). Для создания текстуры имеется две функции glTexImage[1/2]D и gluBuild[1/2]DMipmaps.

glTexImage2D( gluBuild2DMipmaps(

GLenum target, GLenum target,

GLint lavel, GLint components,

GLint components, GLsizei width,

GLsizei width, GLsizei height,

GLsizei height, GLenum format,

GLint border, GLenum type,

GLenum format, const GLvoid\* pixels)

GLenum type,

const GLvoid\* pixels)

Основное различие в том, что первая функция создает текстуру одного определенного уровня детализации и воспринимает только изображения , размер которых кратен степени двойки. Вторая функция более гибкая. Она генерирует текстуры всех уровней детализации. Также эта функция не требует, чтобы размер изображения был кратен степени двойки. Она сама сожмет/растянет изображение подходящим образом, хотя возможно окажется, что и не вполне подходящим. Я воспользуюсь функцией glTexImage2D. Первый параметр этой функции должен быть GL\_TEXTURE\_2D. Второй - уровень детализации. Нам нужно исходное изображение, поэтому уровень детализации - ноль. Третий параметр указывает количество компонентов цвета. У нас изображение хранится в формате RGB. Поэтому значение этого параметра равно трем. Четвертый и пятый параметры - ширина и высота изображения. Шестой - ширина границы; у нас границы не будет, поэтому значение этого параметра - ноль. Далее, седьмой параметр - формат хранения пикселей в массиве - GL\_RGB и тип - GL\_UNSIGNED\_BYTE. И наконец, восьмой параметр - указатель на массив данных. Еще вы должны вызвать функцию glPixelStorei и задать, что выравнивание в массиве данных идет по байту. Добавьте следующий код в функцию main.

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3,

photo\_image->sizeX,

photo\_image->sizeY,

0, GL\_RGB, GL\_UNSIGNED\_BYTE,

photo\_image->data);

Аналогичный результат можно получить, вставив вызов gluBuild2DMipmaps с параметрами, указанными ниже.

gluBuild2DMipmaps(GL\_TEXTURE\_2D, 3,

photo\_image->sizeX,

photo\_image->sizeY,

GL\_RGB, GL\_UNSIGNED\_BYTE,

photo\_image->data);

Теперь нужно установить параметры текстуры. Для этого служит функция

glTexParameter[if](GLenum target, GLenum pname, GLenum param)

Первый параметр принимает значение GL\_TEXTURE\_1D или GL\_TEXTURE\_2D. Второй - pname - определяется параметр текстуры, который вы будете изменять. И третий параметр - это устанавливаемое значение. Если вы воспользовались gluBuild2DMipmaps вместо glTexImage2D, то вам не надо устанавливать следующие параметры, т.к. уже сформированы текстуры всех уровней детализации, и OpenGL сможет подобрать текстуру нужного уровня, если площадь объекта не совпадает с площадью текстуры. В противном случае, вы должны добавить следующие строки:

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

Вы указали, что для уменьшения и увеличения текстуры используется алгоритм GL\_NEAREST. Это означает, что цветом пикселя объекта, на который накладывается текстура, становится цвет ближайшего пикселя элемента текстуры. Вместо GL\_NEAREST можно указать GL\_LINEAR, т.е. цвет элемента объекта будет вычисляться как среднее арифметическое четырех элементов текстуры. Имеются еще четыре алгоритма вычисления цвета элемента объекта. Их можно устанавливать, когда вы создали текстуру со всеми уровнями детализации, т.к. применяют алгоритмы GL\_NEAREST и GL\_LINEAR к одному или двум ближайшим уровням детализации.

Еще вы можете установить взаимодействие текстуры с объектом. Тут имеются два режима при использовании трех компонентов цвета. Первый режим, установленный по умолчанию, когда у вас учитывается цвет объекта и цвет текстуры. Результирующий цвет получается перемножением компонентов цвета текстуры на компоненты цвета объекта. Скажем, если цвет текстуры - (r,g,b), а цвет объекта, на который она накладывается, - (r0,g0,b0), то результирующим цветом будет - (r\*r0,g\*g0,b\*b0). В случае, если цвет объекта черный - (0,0,0), то вы не увидите на нем текстуру, так как она вся будет черной. Второй режим взаимодействия, когда цвет объекта не учитывается. Результирующим цветом будет цвет текстуры. Эти параметры можно установить следующим образом.

glTexEnv(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE)

glTexEnv(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL)

По умолчанию, как я уже сказал, является режим GL\_MODULATE. Теперь сделайте активной текстуру space\_tex. И повторите для нее то же самое. На этом заканчивается создание текстуры. Осталось связать координаты текстуры с координатами объекта. Отредактируйте функцию display так:

glEnable(GL\_TEXTURE\_2D);

glColor3d(1,1,1);

glBindTexture(GL\_TEXTURE\_2D, space\_tex );

glBegin(GL\_QUADS);

glTexCoord2d(0,0); glVertex3d(-5,-5, -0.1);

glTexCoord2d(0,1); glVertex3d(-5, 5, -0.1);

glTexCoord2d(1,1); glVertex3d( 5, 5, -0.1);

glTexCoord2d(1,0); glVertex3d( 5,-5, -0.1);

glEnd();

glBindTexture(GL\_TEXTURE\_2D, photo\_tex);

glBegin(GL\_QUADS);

glTexCoord2d(0,0); glVertex2d(-4,-4);

glTexCoord2d(0,1); glVertex2d(-4, 4);

glTexCoord2d(1,1); glVertex2d( 4, 4);

glTexCoord2d(1,0); glVertex2d( 4,-4);

glEnd();

glDisable(GL\_TEXTURE\_2D);

Как вы, наверное, догадались, glTexCoord2d сопоставляет координаты текстуры вершинам четырехугольника. Скажу только, что нижний левый угол текстуры имеет координаты (0,0), а верхний правый - (1,1).



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/texture/texture.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/texture/release/texture.exe)  .  
Моя фотография [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/photo.bmp)  . Звездное небо [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/space.bmp)  .

**5.5 Повторение текстуры**

Размножить текстуру на плоскости не составляет большого труда. Давайте немного отредактируем программу из предыдущего раздела. Для того чтобы иметь возможность повторять текстуру, нужно установить параметр GL\_REPEAT для ее S и T координат. S-координата текстуры - это горизонтальная координата, T-координата - вертикальная. Второй параметр, который может быть установлен для координат, - GL\_CLAMP. Он гарантирует, что текстура не будет размножена. По умолчанию установлено GL\_REPEAT. Но я все-таки приведу соответствующий код, чтобы вы представляли, как устанавливать этот параметр. В функцию main добавьте следующие строки:

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

Теперь отредактируйте функцию display.

void CALLBACK display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glEnable(GL\_TEXTURE\_2D);

glColor3d(1,1,1);

glBindTexture(GL\_TEXTURE\_2D, space\_tex );

glBegin(GL\_QUADS);

glTexCoord2d(0,0); glVertex3d(-5,-5, -0.1);

glTexCoord2d(0,1); glVertex3d(-5, 5, -0.1);

glTexCoord2d(1,1); glVertex3d( 5, 5, -0.1);

glTexCoord2d(1,0); glVertex3d( 5,-5, -0.1);

glEnd();

glBindTexture(GL\_TEXTURE\_2D, photo\_tex);

glBegin(GL\_QUADS);

glTexCoord2d(0,0); glVertex2d(-4,-4);

glTexCoord2d(0,2); glVertex2d(-4, 4);

glTexCoord2d(3,2); glVertex2d( 4, 4);

glTexCoord2d(3,0); glVertex2d( 4,-4);

glEnd();

glDisable(GL\_TEXTURE\_2D);

auxSwapBuffers();

}

Функция glTexCoord привязывает координаты текстуры к вершинам объекта. Как я уже говорил, левый нижний угол текстуры имеет координату (0,0), а правый верхний - (1,1). Если вы указываете в качестве привязки значение больше единицы, то текстура повторяется. В нашем примере, координату (0,0) текстуры мы привязали к левой нижней вершине плоскости с координатой (-4,-4), а координату (3,2) текстуры к правой верхней вершине (4,4). Тем самым, мы получили размножение текстуры по горизонтали в количестве трех штук и по вертикали в количестве двух штук. Другие две вершины мы связали соответствующим образом. Если там указать не те числа, то изображение наклонится.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/repeat/repeat.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/repeat/release/repeat.exe)  .  
Моя фотография [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/photo.bmp)  . Звездное небо [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/space.bmp)  .

**5.6 Упражнение: "Вращаем текстуру"**

Завращайте плоскость с фотографией вокруг оси X и Y. Также пусть она равномерно колеблется вдоль оси Z от 3 до 7.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/rotating/rotating.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/rotating/release/rotating.exe)  .   
Моя фотография [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/photo.bmp)  . Звездное небо [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/space.bmp)  .

**5.7 Текстура на сфере**

Здесь я покажу, как работать с одной единственной текстурой и накладывать текстуры на сферы. Создайте новый проект с именем sphere. Добавьте глобальную переменную.

AUX\_RGBImageRec\* photo\_image;

В функции main загрузите изображение и создайте текстуру. Поскольку текстура у нас в этом приложении всего одна, то создавать идентификатор для нее не надо.

void main()

{

auxInitPosition( 50, 10, 400, 400);

auxInitDisplayMode( AUX\_RGB | AUX\_DEPTH | AUX\_DOUBLE );

auxInitWindow( "Shapes" );

auxIdleFunc(display);

auxReshapeFunc(resize);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_TEXTURE\_2D);

photo\_image = auxDIBImageLoad("photo.bmp");

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3,

photo\_image->sizeX,

photo\_image->sizeY,

0, GL\_RGB, GL\_UNSIGNED\_BYTE,

photo\_image->data);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

auxMainLoop(display);

}

Отредактируйте функцию display. Здесь все вам знакомо см. 4.1, кроме gluQuadricTexture. Эта функция разрешает или запрещает наложение текстуры на трехмерный объект. Второй параметр GL\_TRUE или GL\_FALSE. По умолчанию наложение текстуры запрещено.

void CALLBACK display(void)

{

GLUquadricObj \*quadObj;

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

quadObj = gluNewQuadric();

gluQuadricTexture(quadObj, GL\_TRUE);

gluQuadricDrawStyle(quadObj, GLU\_FILL);

glColor3d(1,1,1);

glRotated(5, 0,1,0);

glPushMatrix();

glRotated(-90, 1,0,0);

gluSphere(quadObj, 3, 16, 16);

glPopMatrix();

gluDeleteQuadric(quadObj);

auxSwapBuffers();

}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/shapes/shapes.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/shapes/release/shapes.exe)  .   
Моя фотография [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/photo.bmp)  .

**5.8 Упражнение "Текстуру в жизнь"**

Наложите текстуру на цилиндр, конус, диски и частичный диск. В качестве изображения возьмите какую-нибудь мозаику, горошек и т.п., потому как что-нибудь осмысленное будет смотреться не очень хорошо.

**5.9 Текстура на чайнике**

Текстуру можно наложить на объект любой сложности. Для этого надо разрешить автоматически генерировать координаты текстуры - glEnable(GL\_TEXTURE\_GEN\_S) и glEnable(GL\_TEXTURE\_GEN\_T). Далее, вы должны установить один из трех алгоритмов генерации координат текстур.

1. GL\_OBJECT\_LINEAR
2. GL\_EYE\_LINEAR
3. GL\_SPHERE\_MAP

Алгоритм генерации координат устанавливается с помощью функции glTexGeni. Первый параметр функции указывает тип координаты, для которой будет установлен алгоритм. GL\_S -горизонтальная координата, GL\_T - вертикальная. Второй параметр этой функции должен быть GL\_TEXTURE\_GEN\_MODE. И третий параметр - один из перечисленных выше алгоритмов. Создайте очередной проект с именем teapot. Отредактируйте функцию main, как в предыдущей программе, где мы накладывали изображение на сферу. Только добавьте там строчку glEnable(GL\_AUTO\_NORMAL), чтобы чайник лучше выглядел. Этот режим разрешает расчет векторов нормалей, что позволяет получать улучшенные изображения, однако занимает некоторое время. А функцию display отредактируйте, как показано ниже.

void CALLBACK display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glEnable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_GEN\_S);

glEnable(GL\_TEXTURE\_GEN\_T);

glColor3d(1,1,1);

glRotated(5,0,1,0);

glPushMatrix();

glTranslated(0,3,0);

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

auxSolidTeapot(2);

glTranslated(0,-3,0);

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);

glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);

auxSolidTeapot(2);

glTranslated(0,-3,0);

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);

glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);

auxSolidTeapot(2);

glPopMatrix();

glDisable(GL\_TEXTURE\_GEN\_S);

glDisable(GL\_TEXTURE\_GEN\_T);

glDisable(GL\_TEXTURE\_2D);

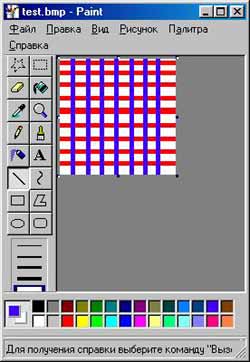
auxSwapBuffers();

}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/teapot/teapot.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/teapot/release/teapot.exe)  .   
Моя фотография [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/bin/photo.bmp)  .

Конечно, фотография накладывается не самым лучшим образом. А что вы хотели? Машина сама сгенерировать правильно координаты на кривой поверхности не может. Тем не менее, если в качестве текстуры взять изображение в горошек, то оно довольно неплохо ляжет на чайник. Я не великий художник и горошек в PaintBrush'e мне изобразить не удалось. Нет, не подумайте, что я совсем криворукий, просто, горошины у меня никак не хотели строиться в одну линию, поэтому я нарисовал следующую картину - "Клеточки". И наложил эту текстуру на чайник.





Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/common/common.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter5/common/release/common.exe)  .

**5.10 Упражнение "Текстуру на тор"**

Сделайте такую же программу только для тора.

# OpenGL - Освещение и все что с ним связано

## 6.1 Общие понятия

Создавать объекты и накладывать на них текcтуры вы научились. Осталась последняя из основных тем - это освещение объектов. Освоив освещение, вы сможете создавать полноценные трехмерные сцены. Освещение любого объекта зависит от двух факторов. Первый - это материал, из которого сделан объект. Второй - это свет, которым он освещен. В этой главе мы подробно рассмотрим все особенности OpenGL, касающиеся освещения объектов.

## 6.2 Модель освещения

По умолчанию освещение отключено. Включается оно командой glEnable(GL\_LIGHTING). В базовом шаблоне освещение я включил, потому что без освещения работать практически невозможно. Сфера всегда будет показываться как круг, а конус - как круг или треугольник. Если монотонное тело у вас равномерно освещено, то вы не можете увидеть его рельеф. Поэтому нам нужно использовать источники света. Сначала рассмотрим функцию, которая устанавливает базовые настройки. Когда вы разрешили освещение, то вы можете уже устанавливать фоновую освещенность. По умолчанию, значение фоновой освещенности равно (0.2, 0.2, 0.2, 1). Создайте новый проект, скопируйте туда шаблонный файл и отключите освещение. Вы с трудом сможете различить сферу на экране. С помощью функции glLightModel вы можете установить фоновое освещение. Если вы повысите его до (1,1,1,1), т.е. до максимума, то включать источники света вам не понадобится. Вы их действия просто не заметите, т.к. объект уже максимально освещен. И получится, что вы как бы отключили освещение. В общем, добавьте в main вызов следующей функции:

float ambient[4] = {0.5, 0.5, 0.5, 1};

...

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient);

Попробуйте изменить параметры и посмотрите на результат. Нулевую лампу (glEnable(GL\_LIGHT0)) лучше отключить.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter6/light/light.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter6/light/release/light.exe)  .

## 6.3 Материал

Материал может рассеивать, отражать и излучать свет. Свойства материала устанавливаются при помощи функции

glMaterialfv(GLenum face, GLenum pname, GLtype\* params)

Первый параметр определяет грань, для которой устанавливаются свойства. Он может принимать одно из следующих значений:

GL\_BACK задняя грань

GL\_FONT передняя грань

GL\_FRONT\_AND\_BACK обе грани

Второй параметр функции glMaterialfv определяет свойство материала, которое будет установлено, и может принимать следующие значения.

GL\_AMBIENT рассеянный свет

GL\_DIFFUSE тоже рассеянный свет, пояснения смотри ниже

GL\_SPECULAR отраженный свет

GL\_EMISSION излучаемый свет

GL\_SHININESS степень отраженного света

GL\_AMBIENT\_AND\_DIFFUSE оба рассеянных света

Ambient и diffuse переводятся на русский как "рассеянный". Разница между ними не очень понятна. Я использую только GL\_DIFFUSE. Третий параметр определяет цвет соответствующего света, кроме случая GL\_SHININESS. Цвет задается в виде массива из четырех элементов - RGBA. В случае GL\_SHININESS params указывает на число типа float, которое должно быть в диапазоне от 0 до 128. Я написал простенький пример с цилиндром и раскрасил его грани в разные цвета. Вам надо всего лишь модифицировать функцию display.

void CALLBACK display(void)

{

GLUquadricObj \*quadObj;

GLfloat front\_color[] = {0,1,0,1};

GLfloat back\_color[] = {0,0,1,1};

quadObj = gluNewQuadric();

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, front\_color);

glMaterialfv(GL\_BACK, GL\_DIFFUSE, back\_color);

glPushMatrix();

glRotated(110, -1,1,0);

gluCylinder(quadObj, 1, 0.5, 2, 10, 10);

glPopMatrix();

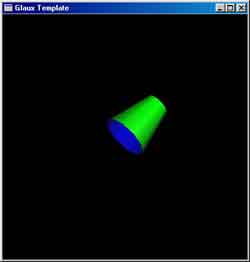
gluDeleteQuadric(quadObj);

auxSwapBuffers();

}

И вы должны разрешить режим освещенности для двух граней. По умолчанию он запрещен. Добавьте в функцию main следующую строчку.

glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter6/materials/materials.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter6/materials/release/materials.exe)  .

## 6.4 Лампы и их свойства

Все параметры лампы задаются с помощью функции glLight, которая имеет следующий прототип:

void glLight[if][v](

GLenum light,

GLenum pname,

GLfloat param)

Первый аргумент определяет номер лампы. Его можно задавать двумя способами. Первый - явно указать GL\_LIHGTi, где GL\_LIGHTi предопределено в файле gl.h следующим образом:

/\* LightName \*/

#define GL\_LIGHT0 0x4000

#define GL\_LIGHT1 0x4001

#define GL\_LIGHT2 0x4002

#define GL\_LIGHT3 0x4003

#define GL\_LIGHT4 0x4004

#define GL\_LIGHT5 0x4005

#define GL\_LIGHT6 0x4006

#define GL\_LIGHT7 0x4007

Второй способ - GL\_LIGHT0 + i, где i номер лампы. Такой способ используется, когда вам надо в цикле изменять параметры ламп. Второй аргумент определяет имя параметра, а третий его значение. Я здесь не буду перечислять всевозможные параметры и их допустимые значения, для этого есть справочник и MSDN. Я покажу и прокомментирую лишь небольшой пример, использующий три лампы. Для начала, давайте разберем шаблонный пример. С помощью следующих функций разрешаем освещение и включаем нулевую лампу.

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

Массивы pos и dir содержат координаты местоположения лампы и направления, куда она светит. Массив dir содержит три координаты - x,y,z. Массив pos - четыре, назначение четвертого мне не очень ясно. Если его значение отличается от нуля, то изображение вполне логичное получается. Если же он ноль, то получается что-то непотребное.

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

Создайте новый проект с именем lamps. Скопируйте шаблонный файл glaux.c. Отредактируйте функцию main:

void main()

{

float pos[4] = {3,3,3,1};

float color[4] = {1,1,1,1};

float sp[4] = {1,1,1,1};

float mat\_specular[] = {1,1,1,1};

auxInitPosition( 50, 10, 400, 400);

auxInitDisplayMode( AUX\_RGB | AUX\_DEPTH | AUX\_DOUBLE );

auxInitWindow( "Glaux Template" );

auxIdleFunc(display);

auxReshapeFunc(resize);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT3);

glEnable(GL\_LIGHT5);

glEnable(GL\_LIGHT6);

glLightfv(GL\_LIGHT3, GL\_SPECULAR, sp);

glLightfv(GL\_LIGHT5, GL\_SPECULAR, sp);

glLightfv(GL\_LIGHT6, GL\_SPECULAR, sp);

color[1]=color[2]=0;

glLightfv(GL\_LIGHT3, GL\_DIFFUSE, color);

color[0]=0;

color[1]=1;

glLightfv(GL\_LIGHT5, GL\_DIFFUSE, color);

color[1]=0;

color[2]=1;

glLightfv(GL\_LIGHT6, GL\_DIFFUSE, color);

glLightfv(GL\_LIGHT3, GL\_POSITION, pos);

pos[0] = -3;

glLightfv(GL\_LIGHT5, GL\_POSITION, pos);

pos[0]=0;pos[1]=-3;

glLightfv(GL\_LIGHT6, GL\_POSITION, pos);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialf(GL\_FRONT, GL\_SHININESS, 128.0);

auxMainLoop(display);

}

Здесь надо сделать пояснения для вызовов функции glLight с параметрами GL\_DIFFUSE и GL\_SPECULAR. GL\_DIFFUSE - определяет цвет света источника. В данном примере у нас три лампы - с красным источником света, с зеленым и с синим. GL\_SPECULAR определяет отраженный свет, см. предыдущий пункт.

Теперь измените функцию display:

glColor3d(1,1,1);

auxSolidSphere(2);



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter6/lamps/lamps.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter6/lamps/release/lamps.exe)  .

## 6.5 Тени

Тени напрямую не поддерживаются библиотекой OpenGL, поэтому этот материал для второй части книги. А если честно, то я с ними пока еще не разбирался.;-)

**OpenGL - Инициализация или как написать приложение с нуля**

**7.1 Общие положения**

В этой главе вы познакомитесь с самой главной частью программного кода - начальной инициализацией. На мой взгляд, это очень сложная тема. Я решил оставить ее на конец книги, когда вы уже будете знакомы с OpenGL. В противном случае, если бы я поместил эту главу в самом начале, я боюсь вы многого не поняли бы. Да и вообще, может не стали бы читать эту книжку.

Сначала, я расскажу в общих чертах, что нужно для инициализации библиотеки OpenGL. Далее мы рассмотрим несколько частных реализаций в среде Windows, Linux и межплатформенный вариант для Java. Для программирования графики в OpenGL вы должны иметь контекст воспроизведения. Это что-то типа контекста устройства в Windows или же магического адреса 0xA000 для графического или же 0xB800 для текствого режима MSDOS. Первое, что вы должны сделать, это подобрать и установить нужные вам параметры контекста воспроизведения. Потом создать сам контекст воспроизведения. И последнее, вы должны сделать его активным. Вообще говоря, вы можете иметь несколько контекстов воспроизведения, но активным может быть только один. Теперь вы можете уже что-нибудь рисовать. Но, возможно, вам не подходят настройки по умолчанию, которые предлагает OpenGL. Так что, придется еще немного потрудиться. Нужно сделать две вещи. Первое - это разрешить нужные вам опции с помощью glEnable и, здесь же, настроить параметры ламп, если вы используете освещение. Второе - надо обрабатывать сообщение об изменениях размера вашего окна. Тут вы указываете ту часть окна, где у вас будет располагаться контекст OpenGL. До этой главы у нас во всех примерах размер нашего окна и окна OpenGL совпадали, но, вообще говоря, это необязательно. Вывод OpenGL может занимать только часть окна, а в остальной части вы можете разместить свои компоненты: кнопки, поля ввода и т.п. Далее, вы должны указать тип проекции: перспективная или параллельная. В перспективной проекции две параллельных прямых будут сходиться вдалеке. В параллельной же проекции они всегда будут оставаться параллельными. И последнее, вы устанавливаете точку, где находится ваш глаз; точку, куда вы смотрите, и вектор, который принимается за направление вверх. У меня этот вектор во всех примерах - (0,1,0), т.е. ось Y направлена вверх.

**7.2 Консольное приложение - Win32 Console Application**

Достоинством приложений данного типа является переносимость на другие платформы при условии, что вы не пользовались другими платформеннозависимыми библиотеками. Реализация OpenGL Auxilary Library существует для большинства платформ. Также здесь значительно упрощена начальная инициализация, т.е. вы быстро можете начать программировать. Идеально подходит для начинающих. Именно поэтому, все примеры в этой книге я привязал к этому варианту приложения OpenGL. Хотя вы можете вставлять приводимый код в WinAPI-приложение и MFC-приложение, и все будет работать точно так же. Для Java-приложения вам придется добавить префиксы к функциям. Недостатком является урезанная функциональность. Вы не можете обрабатывать все сообщения, которые приходят вашему окну. Такой тип приложения идеально подходит для написания небольших портабельных утилит, размером до 25Kb.

Здесь, как я уже говорил, начальная инициализация самая простая. Библиотека сделает за вас большинство действий. От вас потребуется совсем немного по сравнению с другими типами приложений. Итак, давайте для начала создадим проект. Общепризнанно, что тремя наилучшими компиляторами считаются GNU C, Microsoft C и Watcom C. Inprise (Borland) отпадает. Компиляторами других фирм я не пользовался. Все мои задачи решались вышеуказанными четырьмя компиляторами. Правда, должен заметить, что с 1997 года я практически прекратил пользоваться компилятором фирмы Borland. Лишь изредка, когда нужно было перекомпилировать старые утилиты, написанные еще для MSDOS. Эта книга для начинающих, и я хочу сделать ее понятной большинству читателей. Поэтому я не буду рассматривать проекты для GNU C или Watcom C. Не хочу здесь городить непонятные многим начинающим makefile'ы. Однако, в конце данной главы будет разобрано приложение для UNIX, там уже от makefile'ов никуда не деться. Теперь вернемся к нашим баранам.

1. Запустите MSVisualC++6.0
2. Щелкните меню File->New->Win32 Console Application.
3. Выберете каталог и имя проекта задайте glaux, щелкните OK.
4. Выберете An Empty Project, щелкните Finish.
5. Создайте новый текстовый файл и сохраните его с именем glaux.c.
6. Присоедините его к проекту. Project->Add To Project->Files
7. Щелкните Build->Set Active Configuration и установите тип проекта glaux - Win32 Release
8. Далее щелкаете Project->Settings->Link->Object/library modules: и добавьте туда opengl32.lib, glu32.lib и glaux.lib

Проект у нас теперь есть, давайте писать glaux.c. Файл, в котором находится исходный код программы, желательно начинать с комментария. Это необязательно, но этого требует хороший стиль. В комментариях можно указать имя автора, способ связи - обычно, адрес электронной почты. Далее, можно кратко описать, что находится в этом файле. Неплохо вести некоторый дневник здесь же: что и когда вы добавили. У меня эти комментарии выглядят так:

/\*

\* (c) Copyright 1995-1999, Igor Tarasov

\* FidoNet: 2:5020/370.2 620.20 1103.5

\* Inet: itarasov@rtuis.miem.edu.ru

\* Phone: (095)916-89-51 916-89-63

\*/

Теперь надо включить заголовочные файлы:

#include <windows.h>

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glaux.h>

Давайте напишем функцию main(). Я посчитал, что наиболее правильно и понятно будет дать код и прокомментировать его подробно. У меня функция main() выглядит так:

void main()

{

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

// указываем координаты окна на экране

// верхний левый угол (50,10)

// ширина и высота - 400

auxInitPosition( 50, 10, 400, 400);

// устанавливаем параметры контекста OpenGL

//

auxInitDisplayMode( AUX\_RGB | AUX\_DEPTH | AUX\_DOUBLE );

// создаем окно на экране

auxInitWindow( "Glaux Template" );

// наше окно будет получать сообщения

// от клавиатуры, мыши, таймера или любые другие

// когда никаких сообщений нет

// будет вызываться функция display

// так мы получаем анимацию

// если вам нужна статическая картинка,

// то закомментируйте следующую строку

auxIdleFunc(display);

// при изменении размеров окна

// придет соответствующее сообщение

// в Windows - это WM\_SIZE

// мы устанавливаем функцию resize,

// которая будет вызвана

// при изменении размеров окна

auxReshapeFunc(resize);

// далее, я устанавливаю ряд тестов и параметров

// тест прозрачности, т.е. будет учитываться

// четвертый параметр в glColor

glEnable(GL\_ALPHA\_TEST);

// тест глубины

glEnable(GL\_DEPTH\_TEST);

// glColor будет устанавливать

// свойства материала

// вам не надо дополнительно

// вызывать glMaterialfv

glEnable(GL\_COLOR\_MATERIAL);

// разрешаем освещение

glEnable(GL\_LIGHTING);

// включаем нулевую лампу

glEnable(GL\_LIGHT0);

// разрешаем смешение цветов

// подробнее смотри главу "Полезные мелочи",

// далее в секции "прозрачность"

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

// устанавливаем положение нулевой лампы

// смотри главу "Освещение"

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

// и последнее, устанавливаем

// функцию display отрисовки окна

// эта функция будет вызываться всякий раз,

// когда потребуется перерисовать окно

// например, когда вы развернете окно на весь экран

// в windows - это обработчик сообщения WM\_PAINT

auxMainLoop(display);

}

Вот и все с функцией main(). Осталось написать код функции resize и функции display. Вставьте следующий код перед функцией main().

void CALLBACK resize(int width,int height)

{

// Здесь вы указываете ту часть окна,

// куда осуществляется вывод OpenGL.

glViewport(0,0,width,height);

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

// Устанавливаем тип проекции.

// glOrtho - параллельная

// glFrustum - перспективная

// Параметры у этих функций одинаковые.

// Они определяют объем, который вы видите.

// левая стенка - пять единиц влево

// правая - пять единиц вправо

// далее, нижняя стенка и верхняя

// и наконец, передняя и задняя

// см. ниже картинку

glOrtho(-5,5, -5,5, 2,12);

// Устанавливаем точку, в которой

// находится наш глаз ---(0,0,5)

// направление, куда смотрим --- (0,0,0)

// вектор, принимаемый за направление вверх --- (0,1,0)

// этим вектором является ось Y

gluLookAt( 0,0,5, 0,0,0, 0,1,0 );

glMatrixMode( GL\_MODELVIEW );

}

Здесь нужно сделать пояснения по поводу glMatrixMode. Функции glOrtho и glFrustum работают с матрицей, отвечающей за тип проекции. Они просто загружают соответствующую матрицу. Вы можете установить свой тип проекции, если вам это понадобится. Сначала вы говорите, что будете изменять матрицу проекции - glMatrixMode с параметром GL\_PROJECTION. Потом, с помощью glLoadMatrix загружаете соответствующую матрицу. Функции glTranslate/glRotate работают с матрицей вида. Ее мы загружаем последней строкой - glMatrixMode( GL\_MODELVIEW ).

В параметрах контекста воспроизведения мы установили AUX\_DOUBLE. Это значит, что рисоваться все будет сначала в буфер. Для того, что бы скопировать содержимое буфера на экран, вызывается функция auxSwapBuffers(). Если вы программировали анимацию для MSDOS или Windows, то наверняка использовали такой прием, чтобы избавиться от мерцания на экране. В функции display мы сначала очищаем буфер. Цвет, которым заполняется буфер при очищении, можно установить в функции main() вызовом glClearColor(r,g,b).

void CALLBACK display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

/\* remove next tree lines

\* and enter your code here

\*/

glTranslated(0.01,0,0);

glColor3d(1,0,0);

auxSolidSphere( 1 );

auxSwapBuffers();

}

Вот и все.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/glaux/glaux.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/glaux/release/glaux.exe)  .

**7.3 Windows-приложение - Win32 Application**

Достоинством является непосредственное взаимодействие с WinAPI. Начальная инициализация несколько усложняется, но зато вы имеете полноценное windows-приложение. Такой тип приложения подходит для написания серьезных больших программ. Кто-нибудь, конечно, скажет, что приложение непереносимо. Вам нужно написать работающее приложение для windows, а не неработающее, но переносимое приложение.

По поводу переносимости, хочу заметить следующее. В стандарте по языку Си сказано, что код на языке Си может быть платформенно независимым и платформенно зависимым. Из этого следует, что для обеспечения переносимости большой программы, вам придется делать несколько вариантов и затачивать ее под конкретные платформы. Код, относящийся к OpenGL, практически переносим. Непереносима только начальная инициализация. Конечно, вы можете попробовать Java-приложение, но тут возникают свои сложности. Так что, выбор за вами.

Создайте проект Win32 Application. Инструкции смотри в предыдущем разделе. Только имена дайте win и win.c. Теперь будем писать файл win.c. Внесите комментарии, заголовочные файлы и функции display и resize, см. предыдущий раздел. Из функций display и resize уберите слово CALLBACK. А в функции display замените auxSwapBuffers() на

glFinish();

SwapBuffers(wglGetCurrentDC());

После включения заголовочных файлов объявите следующие глобальные переменные.

HWND hWnd;

HGLRC hGLRC;

HDC hDC;

Теперь вставьте код функции, которая устанавливает параметры контекста воспроизведения OpenGL.

int SetWindowPixelFormat()

{

int m\_GLPixelIndex;

PIXELFORMATDESCRIPTOR pfd;

pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);

pfd.nVersion = 1;

pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW |

PFD\_SUPPORT\_OPENGL |

PFD\_DOUBLEBUFFER;

pfd.iPixelType = PFD\_TYPE\_RGBA;

pfd.cColorBits = 32;

pfd.cRedBits = 8;

pfd.cRedShift = 16;

pfd.cGreenBits = 8;

pfd.cGreenShift = 8;

pfd.cBlueBits = 8;

pfd.cBlueShift = 0;

pfd.cAlphaBits = 0;

pfd.cAlphaShift = 0;

pfd.cAccumBits = 64;

pfd.cAccumRedBits = 16;

pfd.cAccumGreenBits = 16;

pfd.cAccumBlueBits = 16;

pfd.cAccumAlphaBits = 0;

pfd.cDepthBits = 32;

pfd.cStencilBits = 8;

pfd.cAuxBuffers = 0;

pfd.iLayerType = PFD\_MAIN\_PLANE;

pfd.bReserved = 0;

pfd.dwLayerMask = 0;

pfd.dwVisibleMask = 0;

pfd.dwDamageMask = 0;

m\_GLPixelIndex = ChoosePixelFormat( hDC, &pfd);

if(m\_GLPixelIndex==0) // Let's choose a default index.

{

m\_GLPixelIndex = 1;

if(DescribePixelFormat(hDC,m\_GLPixelIndex,sizeof(PIXELFORMATDESCRIPTOR),&pfd)==0)

return 0;

}

if (SetPixelFormat( hDC, m\_GLPixelIndex, &pfd)==FALSE)

return 0;

return 1;

}

Информацию о структуре PIXELFORMATDESCRIPTOR смотрите в справочнике. Я пользуюсь MSDN. Сейчас MSDN входит в MS Developer Studio. Редактировать параметры этой структуры вам вряд ли придется. А если придется, то я не смогу тут описать все. Перевести справочник я, конечно, могу, но это вам вряд ли поможет. Книга не предназначена для этого. Здесь рассматриваются конкретные примеры и упражнения.

Теперь напишем функцию обработки сообщений нашего окна.

LRESULT CALLBACK WindowFunc(HWND hWnd,UINT msg,WPARAM wParam,LPARAM lParam)

{

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

PAINTSTRUCT ps;

switch(msg)

{

// сообщение WM\_CREATE приходит

// один раз при создании окна

case WM\_CREATE:

// получаем контекст устройства нашего окна

hDC = GetDC(hWnd);

// устанавливаем параметры контекста воспроизведения OpenGL

SetWindowPixelFormat();

// создаем контекст воспроизведения OpenGL

hGLRC = wglCreateContext(hDC);

// делаем его текущим

wglMakeCurrent(hDC, hGLRC);

// далее см. предыдущий раздел

glEnable(GL\_ALPHA\_TEST);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

break;

// это сообщение приходит при уничтожении окна

case WM\_DESTROY:

// удаляем созданный выше

// контекст воспроизведения OpenGL

if (hGLRC)

{

wglMakeCurrent(NULL, NULL);

wglDeleteContext(hGLRC);

}

// освобождаем контекст устройства нашего окна

ReleaseDC(hWnd, hDC);

PostQuitMessage(0);

break;

// это сообщение приходит всякий раз,

// когда нужно перерисовать окно

case WM\_PAINT:

BeginPaint(hWnd, &ps);

display();

EndPaint(hWnd, &ps);

break;

case WM\_SIZE:

resize( LOWORD(lParam), HIWORD(lParam) );

break;

default:

return DefWindowProc(hWnd,msg,wParam,lParam);

}

return 0;

}

И последнее, осталось написать функцию WinMain.

int WINAPI WinMain(HINSTANCE hThisInst,

HINSTANCE hPrevInst,

LPSTR str,int nWinMode)

{

MSG msg;

WNDCLASS wcl;

wcl.hInstance=hThisInst;

wcl.lpszClassName = "OpenGLWinClass";

wcl.lpfnWndProc = WindowFunc;

wcl.style = CS\_OWNDC | CS\_HREDRAW | CS\_VREDRAW;

wcl.hIcon = NULL;

wcl.hCursor = LoadCursor(NULL,IDC\_ARROW);

wcl.lpszMenuName = NULL;

wcl.cbClsExtra = 0;

wcl.cbWndExtra = 0;

wcl.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

RegisterClass(&wcl);

hWnd = CreateWindow(

"OpenGLWinClass",

"Win API Template",

WS\_OVERLAPPEDWINDOW | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS,

200,

150,

400,

420,

HWND\_DESKTOP, NULL,

hThisInst, NULL);

ShowWindow(hWnd,nWinMode);

UpdateWindow(hWnd);

while(1)

{

while( PeekMessage(&msg,NULL,0,0,PM\_NOREMOVE) )

if(GetMessage(&msg,NULL,0,0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

else

return 0;

display();

}

return 0;

}

OpenGL требует свойства WS\_CLIPCHILDREN и WS\_CLIPSIBLINGS для окна в Windows. Поэтому были добавлены эти свойства при создании окна в функцию Createwindow. Также обратите внимание, что функция display вызывается в бесконечном цикле. Она вызывается, когда в очереди сообщений окна нет ничего. Эта же функция вызывается, когда нужно отрисовать окно заново - обработчик WM\_PAINT.



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/win/win.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/win/release/win.exe)  .

**7.4 MFC-приложение - MFC AppWizard**

Этот тип приложения обладает всеми достоинствами и недостатками WinAPI-приложения, рассмотренного выше, так как MFC - это библиотека классов С++, т.е. надстройка над WinAPI. Кто-нибудь, конечно, скажет, что приложение на плюсах немеряно большое, работает медленно и MFC для ленивых. В общем, тут у каждого свое мнение. Каждый по-своему прав. Тем не менее, я считаю, что для каждой задачи требуется свой инструмент. Где-то лучше использовать MFC, где-то WinAPI. Кричать, что первое или второе является незаменимым средством на все случаи жизни было бы неверным. У MFC есть свои особенности, отнести которые к достоинствам или недостаткам однозначно нельзя. В зависимости от решаемой задачи они идут в плюс или минус. Согласитесь, что глупо забивать сапожный гвоздь кувалдой или же скобу сапожным молотком.

Создаем проект:

1. Запустите MSVisualC++6.0
2. Щелкните меню File->New->MFC AppWizard(exe).
3. Выберете каталог и имя проекта задайте mfc, щелкните OK.
4. Step1: Поставьте переключатель на Single document, далее OK.
5. Step3: Уберите флажок ActiveX Controls, далее OK.
6. Щелкните Finish.
7. Щелкните Build->Set Active Configuration и установите тип проекта MFC - Win32 Release
8. Далее щелкаете Project->Settings->Link->Object/library modules: и добавьте туда opengl32.lib, glu32.lib и glaux.lib

В CMFCView объявите закрытую(private) переменную hGLRC типа HGLRC. Там же объявите функцию int SetWindowPixelFormat(HDC) и открытую(public) функцию display. Вот, что должно получиться:

class CMFCView : public CView

{

private:

CClientDC \*pdc;

HGLRC hGLRC;

int SetWindowPixelFormat(HDC);

public:

void display();

...

Вставьте код этой функции в файл MFCView.cpp. Код возьмите из предыдущего раздела. Отредактируйте функцию CMFCView::PreCretaeWindow следующим образом:

BOOL CMFCView::PreCreateWindow(CREATESTRUCT& cs)

{

// TODO: Modify the Window class or styles here by modifying

// the CREATESTRUCT cs

cs.style |= (WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS);

return CView::PreCreateWindow(cs);

}

Добавьте также функцию display сюда:

void CMFCView::display()

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glTranslated(0.01,0,0);

glColor3d(1,0,0);

auxSolidSphere( 1 );

glFinish();

SwapBuffers(wglGetCurrentDC());

}

Теперь запустите View->Class Wizard и добавьте обработчик WM\_CREATE,WM\_DESTROY и WM\_SIZE в класс CMFCView. Отредактируйте их следующим образом:

int CMFCView::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CView::OnCreate(lpCreateStruct) == -1)

return -1;

pdc = new CClientDC(this);

if(SetWindowPixelFormat(pdc->m\_hDC)==FALSE)

return -1;

hGLRC = wglCreateContext(pdc->m\_hDC);

if(hGLRC == NULL)

return -1;

if(wglMakeCurrent(pdc->m\_hDC, hGLRC)==FALSE)

return -1;

glEnable(GL\_ALPHA\_TEST);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

return 0;

}

void CMFCView::OnDestroy()

{

if(wglGetCurrentContext()!=NULL)

wglMakeCurrent(NULL, NULL) ;

if(hGLRC!=NULL)

{

wglDeleteContext(hGLRC);

hGLRC = NULL;

}

delete pdc;

CView::OnDestroy();

}

void CMFCView::OnSize(UINT nType, int cx, int cy)

{

CView::OnSize(nType, cx, cy);

glViewport(0,0,cx,cy);

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

glOrtho(-5,5, -5,5, 2,12);

gluLookAt( 0,0,5, 0,0,0, 0,1,0 );

glMatrixMode( GL\_MODELVIEW );

}

Пояснения смотри в предыдущих разделах. В StdAfx.h включите заголовочные файлы, после строчки #include <afxext.h>

#include <gl/gl.h>

#include <gl/glu.h>

#include <gl/glaux.h>

Запустите еще раз Class Wizard и добавьте функцию OnIdle в класс CMFCApp. Эта функция вызывается всякий раз, когда очередь сообщений окна пуста. Отредактируйте ее:

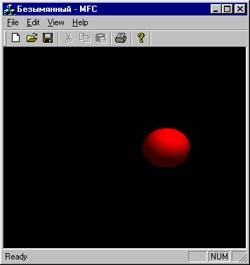
BOOL CMFCApp::OnIdle(LONG lCount)

{

( (CMFCView\*) ((CMainFrame\*)m\_pMainWnd)->GetActiveView() )->display();

return 1;//CWinApp::OnIdle(lCount);

}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/mfc/mfczip.exe)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/mfc/release/mfc.exe)  .

**7.5 Java-апплеты - Magician Library**

Достоинством данного типа приложения, конечно же, является переносимость( для тех платформ, для которых существует плагин) и незаменимое средство для web-программирования. Вы можете украсить свой web-сервер апплетами с трехмерной графикой. К вашим услугам все возможности OpenGL и объектно-ориентированного программирования. Недостатком является сложность программирования на языке Java. За короткое время(три месяца) работы с этим языком на меня свалилось очень много элементарных проблем: отсутствие форматированного ввода/вывода, непонятное поведение апплета - в разных броузерах по-разному; устаревшие методы, которые одним компилятором воспринимались нормально, а другой выдавал предупреждение, и прочие мелкие проблемы. Вообще, писать java-приложения, т.е. самостоятельные программы, я бы не советовал. Воспользуйтесь альтернативой - OpenGL Auxilary Library, рассмотренной в самом начале этой главы. Также можно воспользоваться аналогом GLAUX библиотекой GLUT, о которой пойдет речь ниже. Если же вам необходимо переносимое приложение, то возьмите его из примеров к Magician Library. Для java-апплетов - программ, исполняющихся в web-броузерах, альтернативы нет. Поэтому я и рассматриваю здесь программирование апплетов. У этого типа приложений имеются очень серьезные недостатки. Для запуска апплетов требуется плагин размером около двух мегабайт. Главным образом это связано с тем, что библиотека Magician осуществляет связку между классами Java и длл-модулями от Silicon Graphics, которые не входят в поставку Windows. Работают такие апплеты крайне нестабильно, во всяком случае на момент времени июнь 1999 года было много проблем. И самый главный недостаток, эта библиотека платная, если вы хотите распространять ее со своими коммерческими приложениями, то вам нужно купить лицензию. У меня на сайте находится пробная версия этой библиотеки. Сейчас вышла еще одна библиотека OpenGL для работы с java-апплетами, она называется GL4Java. Далее я ее рассмотрю.

Подробно описывать построение java-апплета, я думаю, излишне после рассмотренных здесь трех примеров приложений. Я лишь приведу здесь свой шаблонный файл template.java с комментариями.

// подключаем стандартные классы

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

//подключаем классы, которые осуществляют связь с opengl.dll и glu.dll

import com.hermetica.magician.\*;

import com.hermetica.util3d.\*;

public class template extends Applet implements GLEventListener

{

// в этом классе основные функции OpenGL, которые в СИ

// были с приставкой gl

private CoreGL gl\_ = new CoreGL();

// в этом классе объявлены функции из glu.dll

private CoreGLU glu\_ = new CoreGLU();

// это компонент, на котором отображается окно OpenGL

private GLComponent glc = null;

// размер компонента OpenGL

int width = 500;

public void init()

{

// создаем компонент OpenGL и устанавливаем его в центре апплета

glc = (GLComponent)GLDrawableFactory.createGLComponent(width, width);

add( "Center", glc );

// устанавливаем параметры контекста воспроизведения

GLCapabilities cap = glc.getContext().getCapabilities();

cap.setDoubleBuffered(GLCapabilities.DOUBLEBUFFER);

glc.addGLEventListener(this);

// инициализируем и стартуем

glc.initialize();

glc.start();

}

public void initialize( GLDrawable component )

{

float pos[] = {3,3,3,1};

float dir[] = {-1,-1,-1};

// тут префиксы немного другие

gl\_.glClearColor( 1.0f, 1.0f, 0.796875f, 1.0f );

gl\_.glShadeModel( GL.GL\_SMOOTH );

gl\_.glEnable(GL.GL\_ALPHA\_TEST);

gl\_.glEnable(GL.GL\_DEPTH\_TEST);

gl\_.glEnable(GL.GL\_COLOR\_MATERIAL);

gl\_.glEnable(GL.GL\_LIGHTING);

gl\_.glEnable(GL.GL\_LIGHT0);

gl\_.glEnable(GL.GL\_BLEND);

gl\_.glBlendFunc(GL.GL\_SRC\_ALPHA, GL.GL\_ONE\_MINUS\_SRC\_ALPHA);

gl\_.glLightfv(GL.GL\_LIGHT0, GL.GL\_POSITION, pos);

gl\_.glLightfv(GL.GL\_LIGHT0, GL.GL\_SPOT\_DIRECTION, dir);

}

public void reshape( GLDrawable component, int x, int y,

int width, int height)

{

// устанавливаем параметры сцены

gl\_.glViewport( component, 0, 0, height, height);

gl\_.glMatrixMode( GL.GL\_PROJECTION );

gl\_.glLoadIdentity();

gl\_.glOrtho(-5,5,-5,5,2,12);

glu\_.gluLookAt( 0,0,5, 0,0,0, 0,1,0 );

gl\_.glMatrixMode( GL.GL\_MODELVIEW );

}

public void display( GLDrawable component )

{

gl\_.glClear( GL.GL\_COLOR\_BUFFER\_BIT | GL.GL\_DEPTH\_BUFFER\_BIT );

// перемещаем сферу

gl\_.glTranslated(0.01,0,0);

gl\_.glColor3d(0,1,0);

shapes.solidSphere(0.6, 16, 16);

}

public void stop()

{

if(glc.isRunning())

glc.suspend();

}

public void start()

{

if(glc.isRunning())

glc.resume();

}

public void destroy()

{

glc.destroy();

}

public GL getGL()

{

return gl\_;

}

}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/java/template.java)  . Html файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/java/index.html)  .

**7.6 Java-апплеты GL4Java**

GL4Java еще одна библиотека, которая связывает классы Java с динамическими модулями opengl32.dll и glu32.dll. Недостатков у этого варианта гораздо меньше, чем у предыдущего. Библиотека имеется под разные платформы. Под Windows имеется инсталлятор. GL4Java работает с динамическими библиотеками OpenGL, которые входят в стандартную поставку Microsoft Windows, в следствие чего необходимый плагин занимает около 200Кб. Маленьким недостатком является требование java-машины версии 5.0.0.3186. Узнать версию вы можете в Internet Explorer'e, меню Ви->Консоль языка Java. GL4Java распространяется бесплатно, последнюю версию вы можете взять на сервере http://www.jausoft.com в разделе Products->GL4Java->Installation. Если этот плагин получит широкое распространение, а у него есть для этого все предпосылки, то написание java-апплетов с использованием OpenGL дело довольно перспективное.

Теперь немного теории о том, как такие библиотеки устроены. Вообще, из Java-апплетов нельзя обращаться к локальному диску пользователя. И тут, у вас может возникнуть вопрос, как же так получается, что из Интернета все-таки можно загрузить апплет и он будет работать с динамическими библиотеками установленными на вашем компьютере? Ведь в одном случае это может быть OpenGL, а в другом - вирус, делающий низкоуровневое форматирование вашего диска и перезаписывающий BIOS. Ответ очень прост, у вас на машине имеются доверительные динамические библиотеки и java-классы, которые с ними взаимодействуют. Просто одних библиотек недостаточно. Из самого java-апплета вы можете загрузить только java-классы на машине пользователя, к которым прописан CLASSPATH. Последним, в свою очередь, разрешено взаимодействовать с динамическими библиотеками. В результате, получается когда вы загружаете java-апплет с использованием OpenGL, то вся графика делается не на уровне виртуальной java-машины, а на уровне opengl32.dll, функционирование которой мало чем отличается от выполнения обычного исполняемого файла. Такие плагины, вообще говоря, являются потенциальной дырой для проникновения в вашу систему из вне. Если вы работаете в многопользовательской операционной системе с разграничением доступа, то не запускайте подозрительных программ от имени администратора. Например, в Unix или WindowsNT, если вы запустите какой-нибудь вирус от имени пользователя, отформатировать жесткий диск ему не удастся, прав у пользователя таких нет. Администраторов лопухов, я здесь не рассматриваю.

Теперь рассмотрим пример. У меня, среди прилагающихся программ, в директории template вы найдете поддиректории GL4Java. Здесь я приведу целиком исходный код с подробными комментариями.

// подключаем необходимые библиотеки

import gl4java.GLContext;

import gl4java.awt.GLCanvas;

import gl4java.awt.GLAnimCanvas;

// подключаем стандартные библиотеки java

import java.awt.\*;

import java.awt.event.\*;

import java.lang.Math;

import java.applet.\*;

// код компонента OpenGL, который мы разместим на нашем апплете

// компонентами являются кнопки, поля ввода и прочие элементы управления

// их программирование очень схоже

class BaseGL extends GLAnimCanvas

{

// в конструктор передаются параметры размеров компонента

// w - ширина, h - высота

public BaseGL(int w, int h){super(w, h);}

// устанавливаем параметр контекста воспроизведения OpenGL

// мы будем использовать анимацию и нам понадобится двойная буферизация

public void preInit(){doubleBuffer = true;}

// устанавливаем параметры сцены

public void init()

{

gl.glMatrixMode(GL\_PROJECTION);

gl.glLoadIdentity();

gl.glOrtho(-5,5,-5,5,2,12);

glu.gluLookAt( 0,0,5, 0,0,0, 0,1,0 );

gl.glMatrixMode(GL\_MODELVIEW);

glj.gljCheckGL();

glj.gljMakeCurrent(false);

}

// функция display выглядит почти также, как и в языке Си

// немного другие префиксы команд

public void display()

{

gl.glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

gl.glRotated(5, 0,0,1);

gl.glLineWidth(5);

gl.glBegin(GL\_LINES);

gl.glColor3d(1,0,0);

gl.glVertex2d(-2,0);

gl.glVertex2d(2,0);

gl.glColor3d(0,1,0);

gl.glVertex2d(0,-2);

gl.glVertex2d(0,2);

gl.glColor3d(0,0,1);

gl.glVertex3d(0,0,-2);

gl.glVertex3d(0,0,2);

gl.glEnd();

// этого своего рода аналог auxSwapBuffers

glj.gljSwap();

}//display

}

// теперь код апплета

public class BaseApp extends Applet

{

BaseGL baseGL = null;

String gljLib=null;

String glLib=null;

String gluLib=null;

public void init()

{

// Загружаем библиотеки, которые будут взаимодействовать c opengl32.dll

// эти библиотеки находятся на машине пользователя

if(GLContext.loadNativeLibraries(gljLib, glLib, gluLib)==false)

System.out.println("could not load native libs:"+

gljLib + ", " + glLib + ", " + gluLib);

else

System.out.println("load native libs:"+

gljLib + ", " + glLib + ", " + gluLib);

// устанавливаем алгоритм размещения компонентов на апплете

setLayout(null);

// устанавливаем размеры самого апплета

setSize(570,450);

// создаем компоненте OpenGL

baseGL = new BaseGL(150,150);

// устанавливаем его границы

baseGL.setBounds(120,0,450,450);

// присоединяем к апплету

add(baseGL);

}

// далее стандартные методы апплета

public void start()

{

baseGL.start();

}

public void stop()

{

baseGL.stop();

}

public void destroy(){

baseGL.stop();

baseGL.cvsDispose();

}

}

Html-код для запуска данного апплета выглядит следующим образом:

<HTML>

<HEAD>

<TITLE>Base class for GL4Java</TITLE>

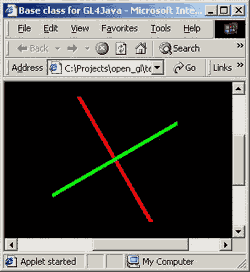
</HEAD>

<BODY>

<applet code="BaseApp.class" width=570 height=450></applet>

</BODY>

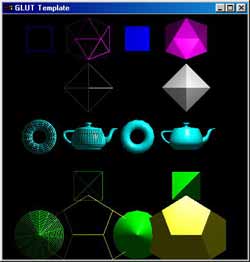
</HTML>



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/gl4java/BaseApp.java)  . Html файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/templates/gl4java/BaseApp.html)  .

**7.7 Библиотека GLUT**

Так получилось, что все примеры к этой книги я написал с использованием библиотеки GLAUX. Библиотека GLUT не менее, а может даже и более универсальная, чем GLAUX. Ее реализация доступна в исходных кодах на языке Си, и она собирается под самые различные платформы и реализации OpenGL. При написание программ для Windows недостатком является отсутствие библиотеки glut32.dll в стандартной поставке Windows. В следствие чего вам придется вместе со своей программой распространять еще и файл glut32.dll, который занимает около 150Кб. В то время как, с библиотекой GLAUX такой необходимости нет, ваш исполняемы файл будет работать на любой машине, где установлена операционная система Windows. Отмечу еще, что GLUT предоставляет больше возможностей по взаимодействия с операционной системой. Вы, например, можете создавать меню. Но в библиотеки GLUT нет таких функций, как [Solid/Wire]Box, [Solid/Wire]Cylinder. Правильные многогранники рисуются одного размера. Стоит еще заметить, что в ней нет ошибки Microsoft, когда у нас не получалось нарисовать на экране два конуса так, чтобы один из них был проволочный, а другой сплошной. Среди прилагающихся программ вы найдете приложение shapes\_glut. Сравните с рисунком из упражнения "Рисуем трехмерные объекты".



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/shapes_glut/shapes_glut.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter2/shapes_glut/release/shapes_glut.exe)  .

Я вам, вкратце, обрисовал все ЗА и ПРОТИВ, а вы уже сами решайте, что вам использовать в каждом конкретном случае. Для того чтобы создать приложение c использованием библиотеки GLUT выполните следующие действия:

1. Cоздайте новый проект - "Консольное приложение Win32"
2. В меню Build->Set Active Configuration выберите Release
3. Далее Project->Settings->Link->Objetc/library modules: впишите следующие библиотеки opengl32.lib glu32.lib glut32.lib.
4. Скопируйте мой шаблон glut.c и подключите его Project->Add To Project->Files

Далее я привожу исходный текст шаблонного файла glut.c с комментариями. Места выделенные серым цветом отличаются от того, что было в glaux.c, все остальное тоже самое.

/\*

\* (c) Copyright 1995-2000, Igor Tarasov

\* FidoNet: 2:5020/370.2 620.20 1103.5

\* email: igor@itsoft.miem.edu.ru itarasov@rtuis.miem.edu.ru

\* Phone: (095)916-89-51 916-89-63

\*/

#include <windows.h>

#include <GL/gl.h>

#include <GL/glu.h>

// подключаем заголовочный файл glut.h

#include <GL/glut.h>

// здесь не нужен больше модификатор CALLBACK

void resize(int width,int height)

{

glViewport(0,0,width,height);

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

glOrtho(-5,5, -5,5, 2,12);

gluLookAt( 0,0,5, 0,0,0, 0,1,0 );

glMatrixMode( GL\_MODELVIEW );

}

void display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

/\* remove next tree lines

\* and enter your code here

\*/

glTranslated(0.01,0,0);

glColor3d(1,0,0);

//изменились префиксы функций

// и появились некоторые дополнительные параметры,

// которые позволяют лучше контролировать

// выполнение того или иного действия

// в данном случае 15, 15 означает кол-во разбиений

// вдоль оси Z и поперек, т.е. дискретность с которой будет рисоваться

// сфера, чем выше эти параметры, тем более качественно будет сделана сфера,

// но на это уйдет и больше времени

glutSolidSphere(1, 15,15);

glutSwapBuffers();

}

void main()

{

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

GLfloat mat\_specular[] = {1,1,1,1};

// тут только префиксы поменялись

// назначения функций можете посмотреть в

// разделе по библиотеке GLAUX

glutInitWindowPosition(50, 10);

glutInitWindowSize(400, 400);

glutInitDisplayMode( GLUT\_RGB | GLUT\_DEPTH | GLUT\_DOUBLE );

glutCreateWindow( "GLUT Template" );

glutIdleFunc(display);

glutDisplayFunc(display);

glutReshapeFunc(resize);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialf(GL\_FRONT, GL\_SHININESS, 128.0);

/\*

\* Enter your cod here

\*/

glutMainLoop();

}

**7.8 Linux-приложение - Mesa Library**

**7.9 Упражнение: "Переносим игру Arcanoid"**

Перенесите во все, указанные здесь приложения, игру Arcanoid.

**OpenGL - Примеры интересных программ**

**8.1 Вращение сцены**

Когда построена сцена, наложены текстуры, включены источники света, присутствует анимация, кажется, что больше и желать нечего. Однако, вы наблюдаете сцену, все время из одной точки. Вы не видите ее с обратной стороны. Было бы интересно иметь возможность управлять положением камеры или вращать всю сцену целиком. На производительности это никак не скажется, т.к. каждый кадр рассчитывается заново. Для OpenGL все равно, где находится камера или как повернута сцена. В этом параграфе я продемонстрирую, как можно использовать мышь и клавиатуру для управления положением сцены в трехмерном пространстве.

Надо сразу отметить, что передвигать мышью объекты в трехмерном пространстве во всех направлениях с той же легкостью, с которой вы перебрасываете папки на рабочем столе Windows вам не удастся. Связано это с тем, что мышь на экране бегает в двумерном пространстве. И следовательно в вашу программу, как вы знаете, передается только два параметра передвижения мыши - х,у. Но для перемещения объектов в трехмерном пространстве требуется еще и z-координата. По хорошему, еще, конечно, нужен вектор нормали к объекту и угол поворота объекта вокруг вектора нормали. Объект в трехмерном пространстве может быть по-разному ориентирован относительно своего центра. Здесь мы ограничимся только вращением сцены вокруг двух осей координат. Для того чтобы можно было посмотреть на любую точку сцены этого вполне достаточно.

За основу мы возьмем нашего старого знакомого "Снеговика" из первых глав этой книги. Для работы с мышью и клавиатурой воспользуемся функциями библиотеки GLAUX, которая выступает связующим звеном между операционной системой Windows и библиотекой OpenGL. Нам потребуется две глобальных переменных для вращения сцены вокруг двух осей. Глобальными они сделаны потому, что к ним необходим доступ из нескольких разных функций, вообще же старайтесь избегать наличия глобальных переменных и сводите их количество к минимуму. Далее приведен исходный код с комментариями:

int alpha=0, beta=0; // объявляем переменные - углы поворота

//эта функция вызывается всякий раз, когда поступает сообщение от мыши

//см. также функцию main, в ней будут установлены функции-обработчики

//мыши и клавиатуры

void CALLBACK mouse(AUX\_EVENTREC \*event)

{

// в этих двух переменных будем хранить старые положения мыши

static int x0,y0=-12345;

//если в x0, y0 содержаться предыдущие координаты мыши,

//то прибавить переменным alpha и beta разницу между

//положениями мыши

if(y0!=-12345)

{

alpha += event->data[AUX\_MOUSEX] - x0;

beta += event->data[AUX\_MOUSEY] - y0;

}

//сохраняем положение мыши

x0 = event->data[AUX\_MOUSEX];

y0 = event->data[AUX\_MOUSEY];

}

//функции-обработчики стрелок клавиатуры

// с ними вы уже встречались в примерах arcanoid, fog, logicop

void CALLBACK Key\_LEFT(void)

{

alpha -= 5;

}

void CALLBACK Key\_RIGHT(void)

{

alpha += 5;

}

void CALLBACK Key\_UP(void)

{

beta += 5;

}

void CALLBACK Key\_DOWN(void)

{

beta -= 5;

}

// рисуем снеговика

void snowman()

{

glPushMatrix();

glColor3d(0.75,0.75,0.75);

glTranslated(0,-3,0);

auxSolidSphere(2.0);

glTranslated(0,3,0);

auxSolidSphere(1.5);

glTranslated(0,2,0);

auxSolidSphere(1);

glColor3d(0,0,0);

glTranslated(-0.3,0.3,1);

auxSolidSphere(0.1);

glTranslated(0.6,0,0);

auxSolidSphere(0.1);

glTranslated(-0.3,-0.3,0);

glColor3d(1,0,0);

auxSolidCone(0.3,0.5);

glTranslated(0,0.75,-1);

glColor3d(0,0,1);

glRotated(-90,1,0,0);

auxSolidCone(0.75,0.75);

glPopMatrix();

}

void CALLBACK display(void)

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

// сохраняем старое положение сцены

glPushMatrix();

//вращаемся вокруг осей Х и Y

glRotated(alpha, 0,1,0);

glRotated(beta, -1,0,0);

snowman();

//возвращаем координату на место

glPopMatrix();

auxSwapBuffers();

}

void main()

{

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

GLfloat mat\_specular[] = {1,1,1,1};

auxInitPosition( 50, 10, 400, 400);

auxInitDisplayMode( AUX\_RGB | AUX\_DEPTH | AUX\_DOUBLE );

auxInitWindow( "Controls" );

auxIdleFunc(display);

auxReshapeFunc(resize);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialf(GL\_FRONT, GL\_SHININESS, 128.0);

//устанавливаем функции-обработчик клавиатуры и мыши

auxKeyFunc(AUX\_LEFT, Key\_LEFT);

auxKeyFunc(AUX\_RIGHT, Key\_RIGHT);

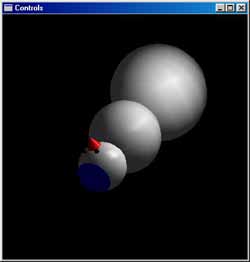
auxKeyFunc(AUX\_UP, Key\_UP);

auxKeyFunc(AUX\_DOWN, Key\_DOWN);

auxMouseFunc(AUX\_LEFTBUTTON, AUX\_MOUSELOC, mouse);

auxMainLoop(display);

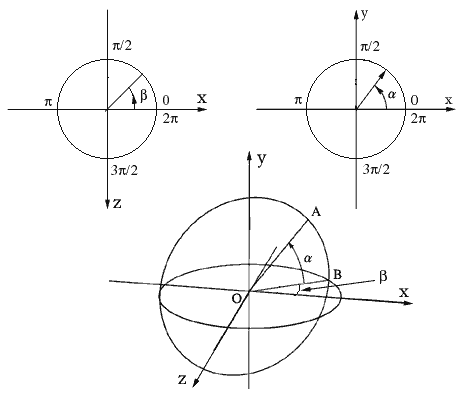
}



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter8/control/control.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter8/control/release/control.exe)  .

**8.2 Управление лампами**

Для управления лампами нам понадобится определить функции-обработчики стрелок: вверх, вниз, влево, вправо. Здесь все точно также как и в предыдущем пункте за исключением того, что управлять положением лампы гораздо удобнее в полярных координатах нежели в декартовых. Точка А в полярных координатах задается следующим образом, бэта - угол отклонения вектора ОВ в плоскости XZ, альфа - угол между вектором ОА и плоскостью XZ и третий параметр радиус - длина вектора.



Стрелки влево, вправо будут управлять углом бэта, а стрелки вверх, вниз - углом альфа. Соответственно, лампа будет перемещаться по сфере. Радиус мы изменять не будем. Для изменения положения лампы, нам понадобится выполнить следующие действия:

* получить текущие координаты лампы
* преобразовать их в полярные
* в полярных координатах изменить соответствующий угол
* преобразовать координаты из полярных в нормальные
* установить новое положение лампы

За основу мы возьмем программу с тремя источниками света направленными на сферу из главы "Освещение". Создайте новый проект и скопируйте в него файл lamps.c. Объявите функции-обработчики и отредактируйте соответствующим образом функцию main.

auxKeyFunc(AUX\_LEFT, Key\_LEFT);

auxKeyFunc(AUX\_RIGHT, Key\_RIGHT);

auxKeyFunc(AUX\_UP, Key\_UP);

auxKeyFunc(AUX\_DOWN, Key\_DOWN);

Код функции Key\_LEFT будет следующим:

void CALLBACK Key\_LEFT(void)

{

float nor[4];

float pol[3];

// получаем текущие координаты лампы

glGetLightfv(GL\_LIGHT3, GL\_POSITION, nor);

// конвертируем их в полярные

Normal2Polar(nor[0], nor[1], nor[2], pol);

// уменьшаем угол бэта на величину дельта

pol[1] -= delta;

// конвертируем обратно в нормальные координаты

Polar2Normal(pol[0], pol[1], pol[2], nor);

// устанавливаем новое положение лампы

glLightfv(GL\_LIGHT3, GL\_POSITION, nor);

}

После включение заголовочных файлов объявите две константы:

#define M\_PI 3.14159265358979323846

float delta=0.1;

Функции перевода из одной системы координат в другую:

void Polar2Normal(float a, float b, float r, float nor[3])

{

nor[0] = r\*cos(a)\*cos(b);

nor[1] = r\*sin(a);

nor[2] = -r\*cos(a)\*sin(b);

}

void Normal2Polar(float x, float y, float z, float pol[3])

{

pol[2] = sqrt(x\*x+y\*y+z\*z);

pol[0] = asin(y/pol[2]);

pol[1] = acos(x/sqrt(x\*x+z\*z));

if(z>0)

pol[1] = 2\*M\_PI - pol[1];

}

**8.3 Упражнение "Лампы"**

Допишите данную программу так, чтобы можно было изменять радиус и переключаться между лампами цифрами - 1,2,3.

**8.4 Управление камерой**

За основу данного проекта мы возьмем программу из упражнения "Список трехмерный фигур". Далее все примерно также, как и в в предыдущем пункте. Здесь нам потребуется три переменных x,y,z типа float для хранения текущего положения камеры, им также присвоим начальное значение (0,0,5). В функциях-обработчиках стрелок клавиатуры мы будем вызывать функцию resize с размерами окна 400х400, эти размеры мы становили в функции main. Соответственно надо модифицировать код функции resize так, чтобы она устанавливала камеру в точку с координатами x,y,z.

#define M\_PI 3.14159265358979323846

float delta=0.1;

float x=0,y=0,z=5;

void CALLBACK resize(int width,int height)

{

glViewport(0,0,width,height);

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

glOrtho(-5,5, -5,5, 2,12);

gluLookAt( x,y,z, 0,0,0, 0,1,0 );

glMatrixMode( GL\_MODELVIEW );

}

Теперь модифицируйте немного функцию перевода из полярных в обычные координаты. Это связано с тем, что в нее удобнее передавать не массив из трех переменных, а указатели на эти переменные. Указатели передаются за тем, чтобы функция смогла сохранить значение в эти переменные.

void Polar2Normal(float a, float b, float r,

float\* x, float\* y, float\* z)

{

\*x = r\*cos(a)\*cos(b);

\*y = r\*sin(a);

\*z = -r\*cos(a)\*sin(b);

}

Функцию перевода из нормальных координат в полярные оставляем без изменений.

void Normal2Polar(float x, float y, float z, float pol[3])

{

pol[2] = sqrt(x\*x+y\*y+z\*z);

pol[0] = asin(y/pol[2]);

pol[1] = acos(x/sqrt(x\*x+z\*z));

if(z>0)

pol[1] = 2\*M\_PI - pol[1];

}

Далее функции-обработчики кнопок.

void CALLBACK Key\_LEFT(void)

{

float pol[3];

Normal2Polar(x, y, z, pol);

pol[1] -= delta;

Polar2Normal(pol[0], pol[1], pol[2], &x, &y, &z);

resize(400,400);

}

void CALLBACK Key\_RIGHT(void)

{

float pol[3];

Normal2Polar(x, y, z, pol);

pol[1] += delta;

Polar2Normal(pol[0], pol[1], pol[2], &x, &y, &z);

resize(400,400);

}

void CALLBACK Key\_DOWN(void)

{

float pol[3];

Normal2Polar(x, y, z, pol);

pol[0] -= delta;

Polar2Normal(pol[0], pol[1], pol[2], &x, &y, &z);

resize(400,400);

}

void CALLBACK Key\_UP(void)

{

float pol[3];

Normal2Polar(x, y, z, pol);

pol[0] += delta;

Polar2Normal(pol[0], pol[1], pol[2], &x, &y, &z);

resize(400,400);

}

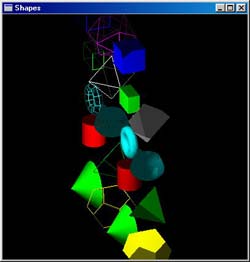
Не забудьте добавить в функцию main.

auxKeyFunc(AUX\_LEFT, Key\_LEFT);

auxKeyFunc(AUX\_RIGHT, Key\_RIGHT);

auxKeyFunc(AUX\_UP, Key\_UP);

auxKeyFunc(AUX\_DOWN, Key\_DOWN);



Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter8/camera/camera.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter8/camera/release/camera.exe)  .

**8.5 Рисуем кривые и поверхности**

В библиотеки glu имеются функции для создания кривых и поверхностей по точкам аппроксимируя их сплайнами. Все это по-моему также круто, как и бесполезно. Здесь я рассмотрю создание поверхности по точкам, которые могут быть заданы в виде массива точек - { (x0, y0, z0), (x1, y1, z1), (x2, y2, z2), ...} или в виде уравнения z(x,y). Задача, как правило, состоит в том, чтобы соединить каждые четыре соседние вершины многоугольником и приписать к ним координаты текстуры. В данной программе я наложу текстуру на поверхность, которая колеблется по синусу. Уравнение такой поверхности: z=sin(x+t). Параметр t - время, нужен для задания анимации. Создайте новый проект с именем flag и скопируйте туда шаблон glaux.c. Объявите в функции display эту переменную следующим образом:

static double t=0;

...

t+=0.1; // на каждом кадре увеличиваем ее значение.

auxSwapBuffers();

Код построения поверхности будет выглядеть так. Вы проходите в двойном цикле по некой области в плоскости XY и вычисляете z в зависимости от х и y. Каждые четыре соседние вершины соединяете многоугольником. В данном случае мы проходим в плоскости XY от точки (0,0) до точки (7,8). Текстуру можно было бы и не привязывать, но тогда бы был совсем не тот эффект. Привязка текстуры делается способом описанным в главе "Работа с картинками". Левая нижняя точка {0,0, z(0,0)} на поверхности соответствует точке {0,0} на текстуре, а правая верхняя точка на поверхности {7,8,z(7,8)} соответствует точке {1,1} текстуры. Поэтому, если точка на поверхности имеет координаты (x,y), то к ней надо привязать точку текстуры с координатой (x/Max\_X, y/Max\_Y), т.е. просто приводим к диапазону 0-1. Max\_X и Max\_Y, как вы понимаете, равны 7 и 8. Ниже следует код функции display c комментариями.

// определяем шаг

#define dx 0.7

#define dy 0.8

void CALLBACK display(void)

{

static double t=0;

double x,y,z;

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

// устанавливаем цвет поверхности белый

// с другим цветом будет складываться цвет текстуры

// а с белым будет видна сама текстура без искажений

glColor3d(1,1,1);

// немного повернем сцену

glPushMatrix();

glTranslated(-3,-3.5,0);

Проходим от точки (0,0) до (7-dy,8-dx)

for(y=0;y<9\*dy;y+=dy)

for(x=0;x<9\*dx;x+=dx)

{

// будем соединять, каждые четыре точки многоугольником

glBegin(GL\_POLYGON);

// вычисляем z от координаты x и от времени t

z = sin(x+t);

// привязываем координаты текстуры к координатам поверхности

glTexCoord2d(x/10/dx, y/10/dy);

glVertex3d(x,y,z);

// здесь значение z точно такое же

// т.к. z не зависит от y, a x у нас не изменилось

glTexCoord2d(x/10/dx, (y+dy)/10/dy);

glVertex3d(x,y+dy,z);

z = sin(x+dx+t);

glTexCoord2d((x+dx)/10/dx, (y+dy)/10/dy);

glVertex3d(x+dx,y+dy,z);

glTexCoord2d((x+dx)/10/dx, y/10/dy);

glVertex3d(x+dx,y,z);

glEnd();

}

glPopMatrix();

t+=0.1;

auxSwapBuffers();

}

Теперь измените положение камеры в функции resize:

gluLookAt( -2,3,5, 0,0,0, 0,1,0 );

И нам осталось загрузить текстуру. Объявите глобальную переменную

AUX\_RGBImageRec\* image;

Функцию main отредактируйте следующим образом.

void main()

{

float pos[4] = {3,3,3,1};

float dir[3] = {-1,-1,-1};

GLfloat mat\_specular[] = {1,1,1,1};

auxInitPosition( 50, 10, 400, 400);

auxInitDisplayMode( AUX\_RGB | AUX\_DEPTH | AUX\_DOUBLE );

auxInitWindow( "Glaux Template" );

auxIdleFunc(display);

auxReshapeFunc(resize);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_AUTO\_NORMAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_POSITION, pos);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, dir);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialf(GL\_FRONT, GL\_SHININESS, 128.0);

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

image = auxDIBImageLoad("photo.bmp");

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3,

image->sizeX,

image->sizeY,

0, GL\_RGB, GL\_UNSIGNED\_BYTE,

image->data);

glEnable(GL\_TEXTURE\_2D);

auxMainLoop(display);

}

Исходный файл смотрите [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter8/flag/flag.c)  . Исполняемый файл [здесь](http://r.codenet.ru/?http://opengl.org.ru/books/open_gl/chapter8/flag/release/flag.exe)  .

PS

В анимированном виде я смотрюсь гораздо лучше. Пивзавод балтика должен мне за рекламу продукта бочку пива поставить. И при том не одну.;-)

**OpenGL - Приложение A**

**Приложение A**

**Где взять OpenGL и другое ПО**

Здесь я перечислю необходимый инструментарий для работы с книгой, и где его можно раздобыть. По минимуму вам нужен сишный компилятор и библиотека OpenGL. Основная масса примеров рассматриваются с привязкой MSVisualC++6.0, однако вы можете использовать любой другой, например фирмы Inprise, в прошлом Borland. Все примеры переносимы на другие платформы, где имеется Auxiliary Library - заголовочный файл glaux.h и линковочный модуль в Windows glaux.lib, в другой ОС он может называться по-другому. Для работы с этой книгой я рекомендую вам все же использовать MSVisualC++6.0. В книге используется OpenGL от Microsoft. Также существует реализация OpenGL от Silicon Graphics. Эта библиотека работает на 15-20% быстрее, чем фирмы Microsoft. Неудобство ее использования состоит в том, что вам придется распространять ее вместе со своими приложениями, в то время как библиотека корпорации Microsoft входит в поставку Windows и поэтому имеется на всех компьютерах.

Компилятор языка Си, как вы понимаете, покупается у соответствующего производителя. Исходный текст библиотеки GLAUX вы найдете в MSDN(Platform SDK->Reference-> Code Samples->Graphica->OpenGL->GLAUX). Ниже перечислено программное обеспечение, которое можно скопировать в Интернет:

|  |  |
| --- | --- |
| OpenGL от Silicon Graphics | [http://itsoft.miem.edu.ru/files/opengl2.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/opengl2.exe) |
| Magician(OpenGL for Java) | [http://www.arcana.co.uk/products/magician](http://r.codenet.ru/?http://www.arcana.co.uk/products/magician) |
| У меня на сервере имеется дистрибутив именно, той версии, которая рассматривается в книге, а то боюсь, они что-нибудь новое выпустят и оно будет не совместимо со старой версией. В дистрибутив входит программка для проверки правильности установки. Ее можно запустить из windows нажав кнопку Start->Programms->Magician->Test\_Magician или Пуск->Программы->Magician->Test\_Magician | [http://itsoft.miem.edu.ru/files/magician.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/magician.exe) |
| Исходные тексты программ моей книги | [http://itsoft.miem.edu.ru/files/ogl\_sources.zip](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/ogl_sources.zip) |
| Дистрибутив библиотека GLUT для работы с Microsoft OpenGL(opengl32.dll, glu32.dll) | [http://itsoft.miem.edu.ru/files/glut\_ms.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/glut_ms.exe) |
| Дистрибутив библиотека GLUT для работы с Silicon Graphics OpenGL(opengl.dll, glu.dll) | [http://itsoft.miem.edu.ru/files/glut\_sgi.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/glut_sgi.exe) |
| Исходный код библиотеки GLUT | [http://itsoft.miem.edu.ru/files/glut\_src.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/glut_src.exe) |
| Дистрибутив библиотеки GLAUX с исправленными мной ошибками | [http://itsoft.miem.edu.ru/files/glaux\_it.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/glaux_it.exe) |
| Исходный код библиотеки GLAUX с исправленными мной ошибками | [http://itsoft.miem.edu.ru/files/glaux\_it\_src.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/glaux_it_src.exe) |
| Дистрибутив библиотеки GLAUX без исправлений | [http://itsoft.miem.edu.ru/files/glaux.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/glaux.exe) |
| Исходный код библиотеки GLAUX без исправлений | [http://itsoft.miem.edu.ru/files/glaux\_src.exe](http://r.codenet.ru/?http://itsoft.miem.edu.ru/files/glaux_src.exe) |
| Первоисточник книги и поддержка | [http://itsoft.miem.edu.ru](http://r.codenet.ru/?http://itsoft.miem.edu.ru) |

**Авторские права и условия распространения книги**

Книга состоит из двух частей. Первая часть содержит основные сведения, вторая еще не написана. В электронном виде книга распространяется бесплатно. Все права на данную книгу принадлежат мне(Игорю Тарасову). Никакая часть книги не может продаваться и использоваться для получения коммерческой выгоды без моего согласия. Вы можете свободно копировать и распространять электронную версию книги и программы, но только в том составе, который находится на моем сервере http://opengl.org.ru, с указанием первоисточника и моих авторских прав. Без моего согласия запрещается изменять содержание, гиперссылки и что-либо в книге. В мае 2000г выходит бумажный вариант, в нем будут дополнительная информация не опубликованная в электронном варианте. Рекомендую вам зарегистрировать у нас на сервере, чтобы я вас своевременно проинформировал об исправлениях и новых версиях. Книга постоянно пополняется новыми материалами. Я оставляю за собой право сократить электронную версию книги. Свои замечание и отзывы направляйте по адресу itarasov@rtuis.miem.edu.ru или опубликуйте их в гостевой книге или форуме. name=3>

**Глоссарий**

auxDIBImageLoad, 35  
auxIdleFunc, 56  
auxInitDisplayMode, 56  
auxInitPosition, 56  
auxInitWindow, 56  
auxKeyFunc, 18  
auxMainLoop, 56  
auxMouseFunc, 18  
auxReshapeFunc, 56  
auxSolidBox, 9  
auxSolidCone, 9  
auxSolidCube, 9  
auxSolidCylinder, 9  
auxSolidIcosahedron, 9  
auxSolidOctahedron, 9  
auxSolidSphere, 9  
auxSolidTeapot, 9  
auxSolidTorus, 9  
auxWireBox, 9  
auxWireCone, 9  
auxWireCube, 9  
auxWireCylinder, 9  
auxWireIcosahedron, 9  
auxWireOctahedron, 9  
auxWireTeapot, 9  
auxWireTorus, 9  
GL\_ALPHA\_TEST, 30  
GL\_AMBIENT, 50  
GL\_AMBIENT\_AND\_DIFFUSE, 50  
GL\_BACK, 25, 50  
GL\_BLEND, 30  
GL\_CLAMP, 43  
GL\_CLIP\_PLANE, 32  
GL\_DECAL, 42  
GL\_DIFFUSE, 50  
GL\_EMISSION, 50  
GL\_FILL, 25  
GL\_FLAT, 29  
GL\_FRONT, 25, 50  
GL\_FRONT\_AND\_BACK, 25, 50  
GL\_LIGHT\_MODEL\_TWO\_SIDE, 51  
GL\_LIGHTING, 49  
GL\_LINE, 25  
GL\_LINE\_LOOP, 23  
GL\_LINE\_STIPPLE, 23  
GL\_LINE\_STRIP, 23  
GL\_LINEAR, 41  
GL\_LINES, 23  
GL\_MODELVIEW, 59  
GL\_MODULATE, 42  
GL\_NEAREST, 41  
GL\_POINT, 25  
GL\_POINT\_SMOOTH, 22  
GL\_POINTS, 21  
GL\_POLYGON, 26  
GL\_POLYGON\_STIPPLE, 33  
GL\_PROJECTION, 59  
GL\_QUAD\_STRIP, 26  
GL\_QUADS, 26  
GL\_REPEATE, 43  
GL\_SHININESS, 50  
GL\_SMOOTH, 29  
GL\_SPECULAR, 50  
GL\_TEXTURE\_1D, 39  
GL\_TEXTURE\_2D, 39  
GL\_TRIANGLE\_FAN, 25  
GL\_TRIANGLE\_STRIP, 25  
GL\_TRIANGLES, 25  
glBegin, 21  
glBindTexture, 39  
glClipPlane, 32  
glColor, 8  
glCopyPixels, 36  
glDrawPixels, 36  
glEnd, 21  
glFrustum, 58  
glGenTextures, 39  
glLightModel, 49  
glLineStipple, 23  
glLineWidth, 16, 23  
glLoadIndentity, 58  
glMaterialfv, 50  
glMatrixMode, 58  
glOrtho, 58  
glPixelStore, 36  
glPixelZoom, 36  
glPointSize, 21  
glPolygonMode, 25  
glPolygonStipple, 33  
glPopMatrix, 10  
glPushMatrix, 10  
glRasterPos2d, 36  
glRotate, 10, 12  
glShadeModel, 29  
glTexCoord, 44  
glTexEnv, 42  
glTexImage2D, 40  
glTexParameter, 41  
glTranslate, 10  
GLU\_FILL, 28  
GLU\_LINE, 28  
GLU\_POINT, 28  
gluBuild2DMipmaps, 40  
gluCylinder, 27  
gluDeleteQuadric, 28  
gluDisk, 27  
gluLookAt, 58  
gluNewQuadric, 28  
gluPartialDisk, 27  
gluQuadricDrawStyle, 28  
GLUquadricObj, 27  
gluQuadricTexture, 45  
gluScaleImage, 36  
gluSphere, 27  
glVertex, 21  
glViewport, 58