

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

Ю. Н. Косников, А. В. Кузьмин

ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ
И ВИЗУАЛИЗАЦИЯ ДВУХМЕРНЫХ
ОБЪЕКТОВ СРЕДСТВАМИ ОТКРЫТОЙ
ГРАФИЧЕСКОЙ БИБЛИОТЕКИ

Учебное пособие

Пенза
Издательство ПГУ
2023

УДК 004.925
К71

Р е ц е н з е н т ы :

доктор технических наук, профессор, профессор кафедры
«Системы автоматизированного проектирования»
Пензенского государственного университета
А. С. Бождай;

доктор технических наук, профессор, ученый секретарь
Акционерное общество «Научно-производственного
предприятия "Рубин"» (г. Пенза)
М. М. Бутаев

Косников, Юрий Николаевич.

К71 Геометрическое моделирование и визуализация двухмер-
ных объектов средствами открытой графической библиотеки :
учеб. пособие / Ю. Н. Косников, А. В. Кузьмин. – Пенза :
Изд-во ПГУ, 2023. – 68 с.

ISBN 978-5-907752-01-6

Изложены способы и приемы геометрического моделирования
двухмерных сцен. Описаны возможности библиотеки OpenGL, которые
необходимо использовать для отображения статических и динамических
графических объектов. Даны рекомендации по рациональному исполь-
зованию команд библиотеки. Показана настройка среды программиро-
вания для визуализации графики средствами библиотеки OpenGL.

Издание подготовлено на кафедре «Информационно-вычисли-
тельные системы» ПГУ и предназначено для обучающихся по направле-
нию подготовки 09.03.03 «Прикладная информатика» и специальности
09.05.01 «Применение и эксплуатация автоматизированных систем спе-
циального назначения».

УДК 004.925

ISBN 978-5-907752-01-6

© Пензенский государственный
университет, 2023

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	4
ВВЕДЕНИЕ.....	5
1. ПРОЕКТИРОВАНИЕ РАСПОЛОЖЕНИЯ ОБЪЕКТОВ В МИРОВОЙ СИСТЕМЕ КООРДИНАТ	7
Контрольные вопросы к разделу 1	9
2. СОЗДАНИЕ ГЕОМЕТРИЧЕСКИХ ФОРМ.....	10
2.1. Определение состава объекта	10
2.2. Определение координат характерных вершин объекта	14
2.3. Выбор команд библиотеки для отображения объекта	15
2.4. Выбор цветовых решений	16
2.5. Запись программы	18
Задание по созданию геометрических форм для самостоятельного выполнения (углубленное задание).....	18
Контрольные вопросы к разделу 2	19
3. РАССТАНОВКА ГРАФИЧЕСКИХ ОБЪЕКТОВ В СЦЕНЕ.....	20
3.1. Дисплейный список	20
3.2. Команды геометрических преобразований	21
3.3. Расстановка объектов в сцене с применением дисплейных списков	23
3.4. Использование стековой памяти	24
3.5. Расстановка объектов по глубине.....	26
Задание по расстановке объектов для самостоятельного выполнения (углубленное задание).....	27
Контрольные вопросы к разделу 3	28
4. НАЛОЖЕНИЕ ТЕКСТУРЫ НА ДВУХМЕРНЫЕ ОБЪЕКТЫ.....	29
4.1. Процесс наложения текстуры	29
4.2. Разработка закона текстурирования.....	30
4.3. Программирование наложения текстуры	32
4.4. Дополнительные возможности текстурирования.....	36
Задание по текстурированию для самостоятельного выполнения (углубленное задание).....	38
Контрольные вопросы к разделу 4	38
5. ОТОБРАЖЕНИЕ ДИНАМИЧЕСКИХ СЦЕН.....	39
5.1. Описание динамики объектов.....	40
5.2. Программирование динамики графических объектов	47
Контрольные вопросы к разделу 5	48
ПРИЛОЖЕНИЕ А. Подключение библиотеки OpenGL в среде Lazarus	49
ПРИЛОЖЕНИЕ Б. Контрольные задания по геометрическому моделированию и визуализации двухмерных объектов	60
СПИСОК ЛИТЕРАТУРЫ	66

ПРЕДИСЛОВИЕ

В информационных источниках OpenGL (Open Graphics Library) называют графической библиотекой, графическим интерфейсом, графическим стандартом. Все названия верны. OpenGL – это набор готовых к использованию макропроцедур, которые облегчают программисту разработку приложений в области двухмерной и трехмерной графики, освобождают его от написания графических программ для конкретного оборудования и одинаково выполняются под управлением разных операционных систем и в разных средах программирования.

Библиотека имеет хорошо продуманную организацию: она включает базовый набор команд, который не меняется со временем, а в процессе совершенствования библиотеки к нему добавляются дополнительные наборы – расширения (GLU, GLUT, GLAUX и др.). Такое строение библиотеки облегчает ее освоение, что и обусловило ее выбор в качестве графического инструмента учебного назначения.

Описанию средств библиотеки OpenGL посвящено много книг и сайтов сети Интернет. Однако зачастую применение команд библиотеки описывается по принципу «делай как я», т.е. без пояснений. В то же время разработка графических программ, в том числе написанных с применением OpenGL, подчинена определенной логике, которую программист должен понимать. В учебном пособии описаны последовательные шаги разработки графической программы, предназначенной для отображения двухмерных сцен. Все конструкции программы снабжены достаточно подробными пояснениями. Свое понимание приемов рисования двухмерной сцены студенты могут проверить с помощью контрольных вопросов и заданий. Для желающих освоить графические процедуры более глубоко даны усложненные задания, а также рекомендации по отображению динамических сцен.

Библиотека OpenGL объемна и разнообразна. С помощью учебного пособия студенты могут получить первый опыт по ее применению. Более глубокие знания можно почерпнуть из информационных источников, приведенных в конце пособия.

Авторы с благодарностью примут критику и предложения по электронной почте: jkos@pnzgu.ru.

ВВЕДЕНИЕ

В геометрическом моделировании и компьютерной графике с помощью двумерных (плоских) объектов представляют результаты решения многих практических задач. Это отображение графиков и диаграмм, планов и карт, диспетчерской обстановки и мнемосхем. Двумерные объекты привязаны к некоторой плоскости, которую можно назвать картинной или экранной. Каждая точка объекта имеет две координаты, обозначаемые обычно x и y . Можно представить плоский объект и в трехмерном пространстве, где третья координата (координата z) всех его точек одинакова, как правило, $z = 0$.

Множество объектов, которые относятся к решаемой в данный момент задаче и последовательно или параллельно во времени представляются наблюдателю, называют сценой. Ясно, что любую сцену, содержащую двумерные объекты, можно нарисовать в графическом редакторе. Однако такая сцена будет статичной. В лучшем случае объекты можно заставить двигаться по раз и навсегда заданной траектории. Для этого служит сценарная анимация. Если же возникает задача управлять движением объектов сцены извне, следить за их эволюциями во времени, то приходится прибегать к программированию.

Программирование графики – непростое дело, и для его облегчения специалистами созданы предметно-ориентированные средства. Это графические библиотеки. В их состав входят готовые графические процедуры – команды. Каждая команда выполняет какую-то графическую операцию: задает режим рисования, описывает форму объекта или его цвет, управляет местоположением и наклоном объекта, накладывает на него сделанный заранее узор и т.д. Команды графических библиотек встраиваются в программы, написанные на языках высокого уровня, и в сочетании с их возможностями позволяют отображать динамические сцены. В настоящее время наибольшей популярностью пользуются две графические библиотеки: DirectX, разработанная фирмой Microsoft, и OpenGL, созданная корпорацией Silicon Graphics Inc. Как развитие OpenGL консорциумом Khronos Group позднее создана еще одна библиотека – Vulkan.

В процессе проектирования графической сцены необходимо решить четыре основные задачи:

- описание геометрической формы объектов;
- расстановку объектов в статической сцене;

– наложение на фон и объекты сцены характерного узора (текстуры);

– изменение геометрических характеристик сцены во времени.

В учебном пособии изложены способы и приемы геометрического моделирования, которые необходимо использовать для решения этих задач. Описаны возможности библиотеки OpenGL и ее команды, позволяющие отображать двухмерные графические объекты. Даны рекомендации по рациональному использованию команд библиотеки. Применение средств геометрического моделирования и отображения объектов рассмотрено на примере формирования сцены, состоящей из объектов в виде стрелок.

В процессе изучения материала студенты могут самостоятельно написать графическую программу. В первом приложении к учебному пособию помещена «заготовка» правильно написанной программы (в среде Lazarus) и содержатся необходимые для ее понимания пояснения. Во втором приложении содержатся контрольные задания на отображение двухмерных объектов. С их помощью студенты могут проверить понимание материала пособия.

1. ПРОЕКТИРОВАНИЕ РАСПОЛОЖЕНИЯ ОБЪЕКТОВ В МИРОВОЙ СИСТЕМЕ КООРДИНАТ

Сцена представляет собой совокупность объектов, имеющих заданные форму, расположение и внешний вид. Описание сцены «закладывается» в компьютер в виде набора графических команд с обоснованно выбранными параметрами. Прежде чем будет определен этот набор команд, нужно принять решение по внешнему виду и параметрам сцены, т.е. спроектировать ее.

Проектирование имеет смысл начинать с выбора системы координат. Библиотека OpenGL по умолчанию вводит декартову трехмерную систему координат, которая называется мировой (МСК). Начало МСК размещается в центре кубического объема видимости, оси МСК направлены так: ось x – вправо, ось y – вверх, ось z – на наблюдателя. Размеры куба видимости по умолчанию составляют ± 1 по каждой координатной оси. Объекты, размещаемые за пределами куба, не видны наблюдателю, а объекты, лежащие на границах куба, отсекаются его плоскостями и видны частично. Наблюдатель по умолчанию размещается в начале координат, а его взор направлен вдоль отрицательной полуоси z . В этом случае плоские объекты располагаются в плоскости $z = 0$. Для отображения большинства двухмерных сцен достаточно работать с такими установками.

Система координатных отсчетов МСК может быть изменена командой

`glOrtho(left, right, bottom, top, near, far);` .

Она вводит параллелепипед видимости, координаты четырех плоскостей которого – левой, правой, нижней и верхней относительно наблюдателя – задаются параметрами команды `left`, `right`, `bottom` и `top` соответственно. Параметры `near` и `far` определяют удаление ближней и дальней плоскостей параллелепипеда от наблюдателя. Если эти плоскости находятся «за спиной» наблюдателя, то соответствующие параметры принимают отрицательные значения. Это особенность параллельной (ортографической) проекции объектов в OpenGL: она позволяет наблюдателю видеть все объекты, попавшие внутрь объема видимости, в том числе и находящиеся «за спиной». Исходное положение куба видимости устанавливается командой

`glOrtho(-1,1,-1,1,-1,1);` .

Примечательно, что в этой команде значение $near = -1$ (со знаком минус) устанавливает ближнюю плоскость отсечения на положительной полуоси z , а значение $far = +1$ (со знаком плюс) устанавливает дальнюю плоскость отсечения на отрицательной полуоси z .

Таким образом, пространство, в котором работает библиотека OpenGL, – трехмерное, но при проектировании плоских сцен его можно считать двумерным и представлять как квадратное окно с размерами 2×2 усл. ед. Начало МСК – в центре окна, оси идут вправо (ось x) и вверх (ось y). Окно можно представить как результат сечения куба видимости плоскостью, перпендикулярной оси глубины. В этом двумерном окне размещаются объекты сцены.

Спроектировать сцену – значит определить форму, размеры, состав и внешний вид объектов, а также разместить их в окне вывода. Удобно проектировать сцену визуально, т.е. рисовать ее в масштабе с привязкой к окну вывода. Пусть сцена содержит три однотипных объекта в виде стрелок. Их геометрические и видовые параметры могут быть различными и в рассматриваемом примере выбраны так, как показано на рис. 1.

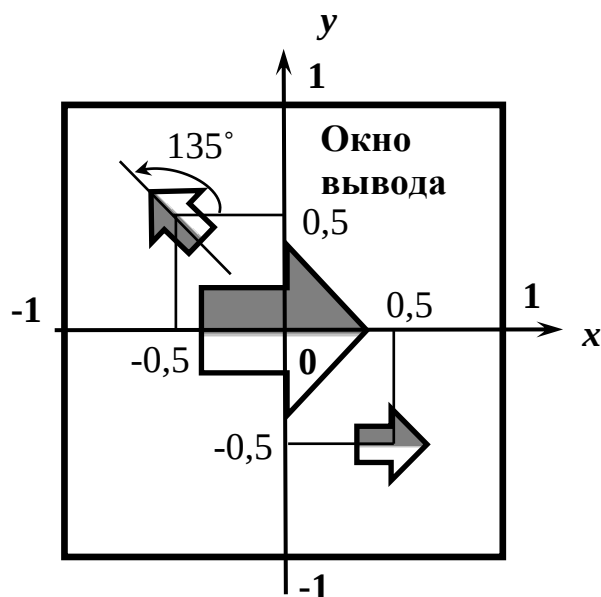


Рис. 1. Пример отображаемой сцены

Размеры центральной стрелки на рис. 1 по каждой координате в два раза больше размеров двух других стрелок, хотя форма всех стрелок (соотношение размеров) одинакова.

Контрольные вопросы к разделу 1

1. Какая система координат применяется в OpenGL? Как в ней размещается наблюдатель по умолчанию?
2. Для чего служит команда `glOrtho`?
3. Что такое параллелепипед видимости, куб видимости в OpenGL?
4. Какие настройки устанавливают команды `glOrtho(-2,2, -2,2, -2, -1)`; `glOrtho(-2,2, -2,2, 1,2)`? Попробуйте нарисовать их.
5. Как с помощью команды `glOrtho` задать размещенный в первом октанте МСК куб видимости с ребром, равным единице, причем одна из вершин куба совпадает с началом МСК?

2. СОЗДАНИЕ ГЕОМЕТРИЧЕСКИХ ФОРМ

В процессе создания геометрических форм необходимо:

- определить состав объекта, выделив в нем геометрические примитивы;
- поместить объект в выбранную систему координат;
- определить координаты характерных точек каждого геометрического примитива в выбранной системе координат;
- выбрать команды библиотеки OpenGL, пригодные для отображения каждого примитива, и задать параметры этих команд;
- выбрать цветовые решения для фона (окна вывода) и примитивов с целью повышения наглядности отображения в процессе отладки программы;
- записать фрагменты программы.

2.1. Определение состава объекта

Объекты сложной конфигурации в компьютерной графике представляются в виде сочетания простых фигур – геометрических примитивов. Примитивы задаются своими характерными точками. Библиотека OpenGL располагает целым набором таких примитивов, которые перечислены в табл. 1. Там же показаны способы описания этих примитивов и принципы их отображения.

Таблица 1

Геометрические примитивы библиотеки OpenGL

Наименование	Идентификатор в OpenGL	Чем задается	Принцип отображения
1	2	3	4
Единичные примитивы			
Точка	GL_POINTS	Координатами точки	По заданным координатам засвечивается один или несколько пикселей экрана
Отрезок	GL_LINES	Координатами начальной и конечной точек	Начальная и конечная точки соединяются прямолинейным отрезком

1	2	3	4
Треугольник	GL_TRIANGLES	Координатами трех угловых точек	Каждая последующая точка соединяется прямолинейным отрезком с предыдущей точкой в последовательности задания этих точек в программе
Четырех-угольник	GL_QUADS	Координатами четырех угловых точек	То же
Много-угольник (N-угольник)	GL_POLYGON	Координатами N угловых точек	То же
Последовательности примитивов			
Ломаная линия	GL_LINE_STRIP	Первый отрезок ломаной задается координатами начальной и конечной точек, каждый последующий отрезок – координатами конечной точки	Каждая последующая точка соединяется прямолинейным отрезком с последней предыдущей точкой
Замкнутый контур	GL_LINE_LOOP	Так же, как ломаная линия	Так же, как для ломаной линии, но конечная точка автоматически соединяется с первой прямолинейным отрезком
Стрип треугольников	GL_TRIANGLE_STRIP	Первый треугольник задается координатами трех угловых точек, каждый последующий – координатами одной точки	Каждая последующая точка соединяется прямолинейными отрезками с двумя последними предыдущими точками
Веер треугольников	GL_TRIANGLE_FAN	Так же, как стрип треугольников	Каждая последующая точка соединяется прямолинейными отрезками с последней предыдущей и первой точками

1	2	3	4
Стрип четырёх- угольников	GL_QUAD_ STRIP	Первый четы- реугольник задается коорди- натами четырех точек, каждый последующий – координатами двух угловых точек	Каждая последующая пара точек соединяется прямолинейными отрез- ками с двумя послед- ними предыдущими точками в последова- тельности задания точек в программе

Принципы отображения геометрических примитивов библиотеки OpenGL иллюстрируются рис. 2, где цифрами обозначена последовательность задания характерных точек в программе рисования.

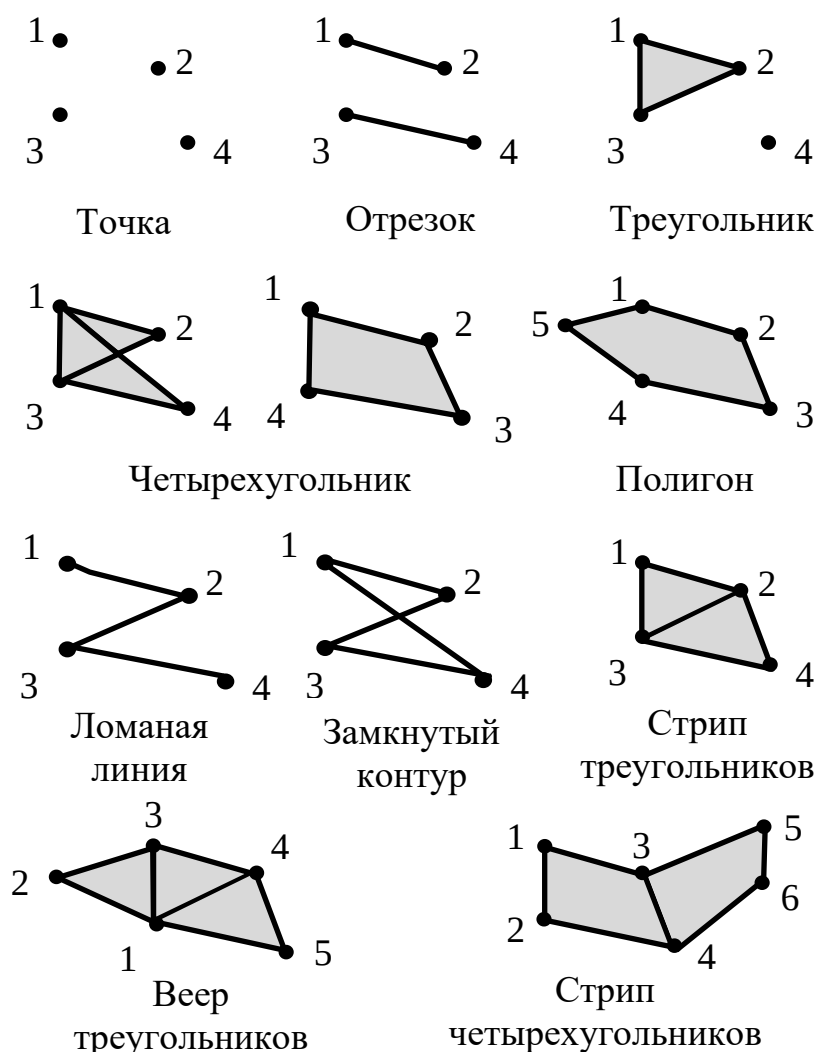


Рис. 2. Принцип отображения геометрических примитивов

На рис. 2 показано, что примитивы, являющиеся многоугольниками, закрашиваются. Следует обратить внимание на различные законы рисования четырехугольника и последовательности четырехугольников. Чтобы четырехугольник в обоих случаях рисовался правильно, его характерные точки нужно задавать в первом случае, последовательно обходя периметр, а во втором случае – «попарно», как показано на рис. 2.

При разбиении объекта на примитивы нужно стремиться к максимально простому решению. Очевидно, что рассматриваемый в качестве примера объект – стрелка – может быть составлен из двух отдельных примитивов: четырехугольника и треугольника. Можно использовать и примитив-полигон. Каждый примитив задается характерными точками, в OpenGL они называются вершинами (Vertex). Есть два принципиально различных способа размещения примитивов объекта в окне вывода. По первому способу каждый примитив описывается координатами своих вершин в МСК. Для этого координаты вершин должны быть рассчитаны, получены путем измерения или найдены иным образом по усмотрению проектировщика. Найти координаты не всегда бывает просто, например, координаты «носика» повернутой стрелки на рис. 1. В соответствии со вторым способом для описания объекта вводится вспомогательная объектная система координат (ОСК). По умолчанию она совпадает с МСК. Примитивы объекта размещаются в ОСК максимально простым образом. Например, стрелка может быть размещена так, как показано на рис. 3.

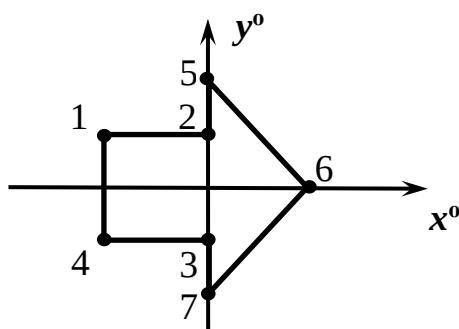


Рис. 3. Размещение примитивов объекта в ОСК

Тогда почти половина координат характерных вершин объекта принимает нулевое значение, причем эти вершины располагаются симметрично относительно оси x^0 , что облегчает поиск остальных координат. При использовании такого способа описания объекта его желаемое расположение в МСК достигается сдвигом и поворотом всей ОСК.

Для этого в библиотеке OpenGL есть средства, рассмотренные далее. Например, повернутая стрелка, изображенная на рис. 1 и размещенная в своей ОСК в соответствии с рис. 3, будет располагаться в МСК так, как показано на рис. 4.

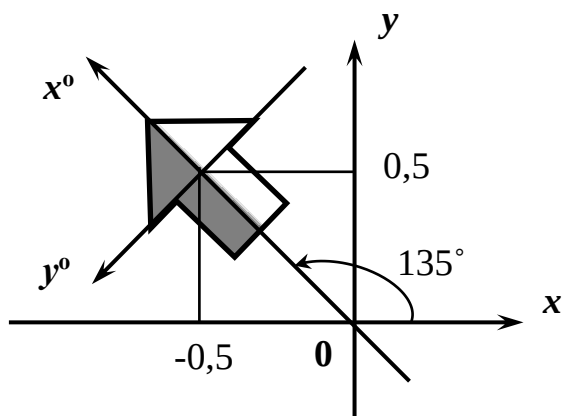


Рис. 4. Расположение ОСК объекта-стрелки в МСК

Второй способ описания объектов является более грамотным и находит широкое применение на практике, поэтому в учебном пособии далее используется именно он.

2.2. Определение координат характерных вершин объекта

Если графический объект является образом реального объекта, найти координаты его характерных точек в ОСК несложно. Нужно просто спроецировать его в масштабе в ОСК. В другом случае координаты вершин понятны из их расстановки в ОСК. В любом случае проектировщик выполняет эту работу, исходя из каких-то своих соображений. Пусть в рассматриваемом примере координаты семи вершин стрелки (вершины помечены на рис. 3 номерами) имеют значения, приведенные в табл. 2.

Таблица 2

Координаты вершин объекта-стрелки в ОСК

Номер вершины i	1	2	3	4	5	6	7
Координата x_i	-0,4	0	0	-0,4	0	-0,4	0
Координата y_i	0,2	0,2	-0,2	-0,4	0,4	0	-0,4

2.3. Выбор команд библиотеки для отображения объекта

Для отображения геометрического примитива необходимо задать его характерные вершины и указать, как их нужно соединять (задать тип примитива). Вершина в пространстве задается командой

```
glVertex2f(x,y); или glVertex3f(x,y,z);
```

Число 2 или 3 указывает, в каком пространстве – двумерном или трехмерном – представляется вершина. Буква f означает тип параметра, в данном случае с плавающей точкой (float), но может использоваться и другой тип: i – целый, d – двойной точности и др. В скобках помещаются параметры команды, это две или три координаты вершины. Они могут задаваться константами (например, glVertex3f(0.5,-0.4,0.0);) или переменными. Во втором случае переменные до использования их в команде glVertex должны получить численные значения.

Тип геометрического примитива задается в качестве параметра команды, которая представляет собой пару операторных скобок

```
glBegin(<параметр>);  
...  
glEnd(); .
```

В качестве параметра используется идентификатор примитива из табл. 1. Между операторными скобками помещаются команды glVertex, задающие координаты характерных вершин. Например, команды

```
glBegin(GL_TRIANGLES);  
    glVertex2f(0.0,0.4);  
    glVertex2f(0.4,0.0);  
    glVertex2f(0.0,-0.4);  
glEnd();
```

описывают треугольник, представляющий собой часть стрелки на рис. 3.

Пара скобок glBegin – glEnd заносит описываемое в ней изображение (в виде кодов цветояркостности) в буфер кадра видеосистемы компьютера. При опросе буфера по закону растра это изображение выводится на экран. Следовательно, можно считать, что скобки glBegin – glEnd рисуют изображение, только не на экране, а пока в памяти компьютера.

В одной паре скобок glBegin – glEnd можно описать несколько однотипных примитивов. Так, набор команд

```
glBegin(GL_LINES);  
    glVertex2f(0.0,0.0);  
    glVertex2f(0.1,0.1);  
    glVertex2f(0.4,0.1);  
    glVertex2f(0.3,0.0);  
    glVertex2f(0.5,0.5);  
glEnd();
```

рисует два прямолинейных отрезка, заданных первой–второй и третьей–четвертой командами glVertex, а пятая команда glVertex игнорируется, так как описываемой ей вершине нет «пары».

2.4. Выбор цветовых решений

Выбор цветовых решений рисования объектов не относится к геометрии сцены, но для повышения наглядности картинки в процессе отладки программы бывает полезно раскрасить ее определенным образом. При создании графической сцены различают цвет фона (цвет закрашки окна вывода) и цвет пера (цвет линий и закрашки примитивов). По умолчанию цвет фона – черный, а цвет пера – белый, но их можно менять специальными командами. Цвет фона в OpenGL задается командой

```
glClearColor(R,G,B,A); ,
```

где R, G, B – доли красного, зеленого и синего основных цветов в цвете фона (лежат в диапазоне от 0 до 1); A – коэффициент прозрачности фона (0 – прозрачный, 1 – непрозрачный фон). Если использовать только нулевые и единичные значения интенсивности основных цветов, можно задать восемь ярких цветов фона в соответствии с табл. 3. Пропорционально уменьшая значения интенсивности, можно сделать цвет фона менее насыщенным, например, при $R = G = B = 0,5$ получается серый цвет.

Окно вывода закрашивается заданным цветом командой

```
glClear(GL_COLOR_BUFFER_BIT); ,
```

которая заносит в буфер цвета (и в буфер кадра) заданные коды цветояркости для всех пикселей окна.

Для задания цвета пера используется команда

`glColor3f(R,G,B);` ,

где R, G, B – тоже доли красного, зеленого и синего цветов, но теперь – в цвете геометрической фигуры.

Таблица 3

**Набор цветов, полученный при использовании нулевых
и единичных интенсивностей основных цветов**

R	G	B	Цвет
0	0	0	Черный (black)
0	0	1	Синий (blue)
0	1	0	Зеленый (green)
0	1	1	Бирюзовый (cyan)
1	0	0	Красный (red)
1	0	1	Пурпурный (magenta)
1	1	0	Желтый (yellow)
1	1	1	Белый (white)

Эта команда устанавливает так называемый текущий цвет пера. Он применяется при рисовании всех объектов, описанных в программе (в скобках `glBegin` – `glEnd`) ниже установки этого цвета. В нужный момент текущий цвет пера меняется с помощью повторного применения команды `glColor`. Например, приведенный ниже фрагмент программы рисует на светло-бирюзовом фоне объект, состоящий из двух разноцветных примитивов: красного треугольника и зеленого четырехугольника:

`glClearColor(0.0,0.5,0.5,1.0);` //задание светло-бирюзового
цвета фона

```
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0); //задание красного цвета пера
glBegin(GL_TRIANGLES);
    glVertex2f(x1,y1);
    glVertex2f(x2,y2);
    glVertex2f(x3,y3);
glEnd();
glColor3f(0.0,1.0,0.0); //задание зеленого цвета пера
glBegin(GL_QUADS);
    glVertex2f(x4,y4);
    glVertex2f(x5,y5);
    glVertex2f(x6,y6);
    glVertex2f(x7,y7);
glEnd(); .
```

2.5. Запись программы

Структура графической программы включает следующие фрагменты:

- раздел описаний;
- задание графического режима (задание формата пикселя);
- задание начальных установок;
- рисование объектов;
- завершение программы.

Раздел описаний, как обычно, нужен для описания используемых в программе модулей, типов, констант, переменных и других конструкций, предусматриваемых языком программирования.

Описания задания графического режима и завершения программы даны в приложении А. Начальные установки задают текущие настройки, разрешают или запрещают какие-то действия в зависимости от замысла разработчика. Из рассмотренных команд к начальным установкам относятся команды выбора и применения цвета фона. В следующих разделах учебного пособия рассмотрены другие возможные начальные установки.

Рисующий фрагмент программы включает уже рассмотренные команды описания геометрических примитивов и задания их цвета. Нужно только иметь в виду, что рисование примитивов (занесение кодов в буфер кадра) осуществляется в той последовательности, в которой примитивы описаны в программе. Это означает, что в случае перекрытия примитивов сверху (в буфере и на экране) окажется примитив, описанный в программе позднее.

Задание по созданию геометрических форм для самостоятельного выполнения (углубленное задание)

1. Попробуйте внести команду задания цвета `glColor3f(1.0,0.0,0.0);` внутрь операторных скобок `glBegin – glEnd`, рисующих какой-либо примитив. Поставьте ее перед первой, потом – перед второй, потом – перед третьей командой `glVertex`. Попытайтесь объяснить эффект.

2. Перед операторными скобками `glBegin – glEnd`, рисующими некоторый примитив с заливкой, вставьте команду `glShadeModel(GL_FLAT);` и снова внесите внутрь скобок команду задания цвета,

поставив ее сначала перед первой, потом – перед второй, потом – перед третьей командой glVertex. Посмотрите описание команды glShadeModel в литературе. Поясните результаты.

Контрольные вопросы к разделу 2

1. Что такое геометрический примитив? Назовите геометрические примитивы OpenGL.

2. Для чего служит команда glVertex? Какие параметры она имеет?

3. Чем различаются примитивы, задаваемые строками GL_LINE и GL_LINE_STRIP, GL_TRIANGLE и GL_TRIANGLE_STRIP, GL_QUAD и GL_QUAD_STRIP?

4. Нарисуйте фигуры, заданные командами:

```
glBegin(<ИДЕНТИФИКАТОР ПРИМИТИВА>);  
    glVertex2f(0.0,0.0);  
    glVertex2f(0.4,0.0);  
    glVertex2f(0.6,-0.4);  
    glVertex2f(0.2,-0.6);  
glEnd(); ,
```

если идентификатор примитива – это строка GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN.

5. Чтобы закрасить фон окна вывода выбранным цветом, нужно применить две команды: glClearColor и glClear. Чем они различаются?

6. Можно ли считать пару операторных скобок glBegin – glEnd() и команды, записанные между ними, рисующими командами? Обоснуйте свое мнение.

7. Какое действие выполняет OpenGL при выполнении команды glColor3f(R,G,B);? Как могут выглядеть параметры этой команды?

3. РАССТАНОВКА ГРАФИЧЕСКИХ ОБЪЕКТОВ В СЦЕНЕ

3.1. Дисплейный список

Как отмечалось в разд. 2, графический объект описывается в своей системе координат – ОСК. Расположение объекта в заданном месте окна вывода осуществляется путем сдвига и поворота всей ОСК, как это показано на рис. 4. Для изменения размеров объекта применяется масштабирование ОСК. Таким образом, объект рисуется на экране с привязкой к своей ОСК, но сама эта ОСК установлена в МСК и промасштабирована требуемым образом.

Если в разных местах сцены отображается несколько объектов, имеющих одинаковую форму, т.е. одинаково описанных в своей ОСК, то для их отображения в соответствии с описанным подходом необходимо:

- задать с помощью операций сдвига и поворота положение ОСК первого объекта;
- задать с помощью операции масштабирования размеры первого объекта;
- нарисовать первый объект (скобки glBegin – glEnd);
- задать положение ОСК второго объекта;
- задать размеры второго объекта;
- нарисовать второй объект и т.д.

Легко понять, что параметры расстановки у всех объектов различны, а процесс рисования одинаков. Это означает, что в программе несколько раз будут вызываться одни и те же последовательности рисующих команд в скобках glBegin – glEnd. В программировании в таких случаях применяют процедуры. В библиотеке OpenGL повторяющаяся последовательность рисующих команд тоже может быть оформлена в виде своеобразной процедуры, которая носит название дисплейного списка.

Дисплейный список – это любая последовательность команд, имеющая самостоятельное значение и получившая уникальное имя. Эта последовательность команд заключается в операторные скобки glNewList – glEndList. Команда

```
glNewList(1, GL_COMPILE);
```

открывает (создает) дисплейный список, а беспараметрическая команда

`glEndList();`

его закрывает.

Параметрами команды открытия являются имя дисплейного списка (это целое число, например, 1) и режим его компиляции. Если этот режим `GL_COMPILE`, то дисплейный список компилируется, но не выполняется, а ждет своего вызова специальной командой. Она имеет вид

`glCallList(1);` .

Если же режим компиляции дисплейного списка задается строкой `GL_COMPILE_AND_EXECUTE`, то дисплейный список компилируется и тут же выполняется.

В рассматриваемом примере в виде дисплейного списка имеет смысл оформить рисование стрелки, привязанной к своей ОСК. В этот дисплейный список войдут команды задания цвета и рисования (в скобках `glBegin` – `glEnd`) составляющих стрелку примитивов.

3.2. Команды геометрических преобразований

Для расстановки и масштабирования объектов в сцене применяются команды, задающие геометрические преобразования: `glTranslate`, `glRotate` и `glScale`. Команда

`glTranslatef(±Δx,±Δy,±Δz);`

задает сдвиг ОСК вдоль ее координатных осей на расстояния Δx , Δy и Δz относительно предыдущего положения. Эта команда, в принципе, работает в трехмерном пространстве, поэтому имеет три параметра. Знаки приращений Δx , Δy , Δz определяют направление сдвига: в направлении положительной (плюс) или отрицательной (минус) координатной полуоси. Буква *f* (float) в описании команды означает, что сдвиги задаются вещественными числами, хотя могут использоваться и другие типы параметров команды. Например, если ОСК наклонной стрелки изначально совпадает с МСК, то команда

`glTranslatef(−0.5,0.5,0.0);`

устанавливает начало ОСК в точку $(-0.5, 0.5)$, как показано на рис. 4.

Команда

`glRotatef( $\pm$ alfa,  $n_x$ ,  $n_y$ ,  $n_z$ );`

поворачивает ОСК на угол α против часовой стрелки (знак плюс) или по часовой стрелке (минус) вокруг некоторой оси. Ось поворота рассматривается как вектор, выходящий из начала координат ОСК и имеющий координаты n_x , n_y , n_z . Это означает, что команда поворота, как и команда сдвига, работает в трехмерном пространстве. Напомним, что координатами вектора в трехмерном пространстве являются проекции его на координатные оси, а в числах – координаты точки, в которую он направлен. Например, на рис. 5 показан вектор $\vec{N}$ с координатами $(-3, 2, 1)$, там же изображены его проекции n_x , n_y , n_z на координатные оси.

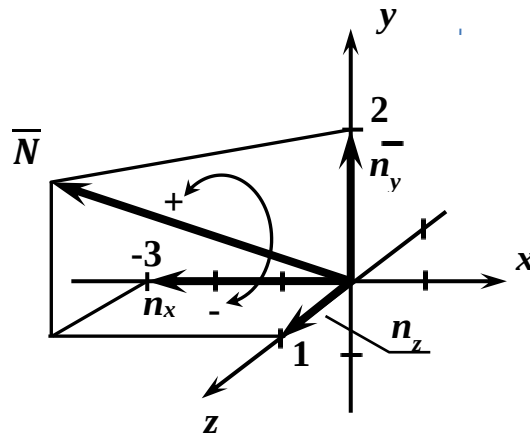


Рис. 5. Пример вектора оси поворота

Очевидно, чтобы повернуть объект в плоскости экрана (в плоскости $z = 0$), нужно использовать нулевые координаты n_x , n_y и ненулевую координату n_z , а чтобы правильно задать угол поворота объекта, нужно смотреть на поворот с «острия» вектора оси поворота. Так, если в качестве оси поворота объекта используется вектор $\vec{N}$, то положительное или отрицательное направление поворота устанавливается так, как показывает стрелка на рис. 5.

Команда `glRotate` поворачивает ОСК относительно ее предыдущего состояния, т.е. угол поворота может накапливаться. Величина угла α в команде выражается в градусах.

Команда

`glScalef( $k_x$ ,  $k_y$ ,  $k_z$ );`

сжимает или растягивает ОСК, а вместе с ней и размещенный в ней объект, вдоль координатных осей. Величину деформации задают коэф-

эффициенты масштабирования k_x , k_y , k_z . При выполнении команды `glScale` координаты всех точек ОСК умножаются на масштабные коэффициенты. Это означает, что при использовании $k_x, k_y, k_z > 1$ происходит увеличение объекта на экране, а при $k_x, k_y, k_z < 1$ – его уменьшение. Равенство какого-то коэффициента нулю обращает в нуль соответствующую координату точек объекта. Различные сочетания масштабных коэффициентов (неравенство их друг другу) приводят к искажению геометрической формы объекта.

3.3. Расстановка объектов в сцене с применением дисплейных списков

Графические объекты (стрелки), рассматриваемые в качестве примера, имеют одинаковую форму. Это означает, что они могут быть описаны с помощью одного дисплейного списка. Пусть он имеет имя в виде числа 2 и описывает объект в ОСК так, как это показано на рис. 3. Для того чтобы расставить графические объекты в соответствии с рис. 1, нужно для каждого из них задать местоположение командой сдвига, задать угол наклона командой поворота, задать размер командой масштабирования и вызвать на исполнение дисплейный список.

Центральная стрелка не нуждается в геометрических преобразованиях, так как ее положение и размеры соответствуют рис. 1 (напомним, что по умолчанию ОСК совпадает с МСК). Поэтому для рисования центральной стрелки нужно просто вызвать дисплейный список:

```
glCallList(2); .
```

Теперь можно перейти к рисованию повернутой стрелки. Для нее команды геометрических преобразований и вызов дисплейного списка выглядят так:

```
glTranslatef(-0.5,0.5,0.0);  
glRotatef(135,0.0,0.0,1.0);  
glScalef(0.5,0.5,1.0);  
glCallList(2); .
```

В приведенном фрагменте программы важна последовательность преобразований. Что будет, если ее изменить, например, поменять местами поворот и сдвиг? Операция поворота изменит направления координатных осей, и тогда сдвиг, который выполняется вдоль этих координатных осей, даст не ожидаемый, а совсем другой результат. Вместо перемещения влево вверх стрелка сдвинется вниз. Это легко

понять, построив эволюции стрелки в масштабе на бумаге. Критично и место размещения в программе операции масштабирования. Действительно, если масштабирование предшествует сдвигу, то сдвиг произойдет в измененной системе координат и желаемая величина перемещения не будет достигнута.

3.4. Использование стековой памяти

После размещения второго объекта – повернутой стрелки – нужно перейти к отображению третьего объекта. Координаты начала его ОСК – (0.5,0.5). Однако просто применить команду `glTranslate` с такими параметрами не получится, так как перед этим в процессе установки второй стрелки положение ОСК было изменено. Напомним, что команды сдвига, поворота и масштабирования выполняют свои преобразования ОСК относительно ее предыдущего состояния, т.е. работают по приращениям. Значит, для установки третьей стрелки нужно найти такие параметры сдвига, поворота и масштабирования, которые позволят перейти от состояния ОСК второй стрелки к состоянию ОСК третьей стрелки, что, в общем случае, непросто. Гораздо проще было бы задать положение третьей стрелки, отталкиваясь от исходного состояния ОСК, т.е. в центре экрана, без поворота, без масштабирования. Для этого можно было бы после установки второй стрелки вернуть ОСК в исходное состояние, выполнив обратные геометрические преобразования, а затем приступить к установке третьей стрелки. Однако это неэффективный прием, так как наборы преобразований, в общем случае, могут быть большими. В библиотеке OpenGL есть аппарат стековой памяти, специально созданный для выполнения этой работы.

Стек – это память, действующая по принципу «последним вошел – первым вышел». Если в некоторый момент выполнения программы применить команду

`glPushMatrix();` ,

то состояние системы координат, действующее на данный момент, будет запомнено. Далее это состояние можно изменять командами сдвига, поворота, масштабирования, но запомненное исходное состояние будет храниться в стеке. В нужный момент времени запомненное состояние ОСК может быть возвращено – считано из стека – беспараметрической командой

`glPopMatrix();` .

В принципе, объем стека позволяет, если это необходимо, сохранять в нем несколько состояний системы координат – на разные моменты времени. Для этого каждый раз применяется команда `glPushMatrix`. Для извлечения из стека очередного состояния ОСК каждый раз применяется команда `glPopMatrix`.

Работа со стеком имеет одну особенность. После того как командой `glPopMatrix` некоторое состояние ОСК извлекается из стека, оно становится доступным для изменения (сдвига, поворота, масштабирования), но в самом стеке оно пропадает – стирается. Следовательно, если это состояние ОСК в будущем понадобится еще раз, его нужно еще раз запомнить с помощью команды `glPushMatrix`.

Применив аппарат стековой памяти, расстановку стрелок в соответствии с рис. 1 можно описать следующим образом:

```
//запоминание исходного состояния ОСК
glPushMatrix();
//отображение первой стрелки
glCallList(2);
//установка второй стрелки
glTranslatef(-0.5,0.5,0.0);
glRotatef(135,0.0,0.0,1.0);
glScalef(0.5,0.5,1.0);
//отображение второй стрелки
glCallList(2);
//возврат исходного состояния ОСК
glPopMatrix();
//повторное запоминание исходного состояния ОСК
glPushMatrix();
//установка третьей стрелки
glTranslatef(0.5, -0.5,0.0);
glScalef(0.5,0.5,1.0);
//отображение третьей стрелки
glCallList(2); .
```

С помощью стека можно запоминать и возвращать любое состояние системы координат. Однако часто бывает необходимо установить именно исходное состояние ОСК – без каких-либо сдвигов, поворотов и масштабных изменений. Для этого служит команда

```
glLoadIdentity(); .
```

Ее удобно использовать в начале программы. Дело в том, что в процессе работы программы часто возникает необходимость выполнить ее тело еще раз, т.е. перерисовать изображение. Это происходит, например, при изменении размеров окна вывода или его сворачивании-разворачивании. При перерисовке в программе сохраняется текущее состояние ОСК, которое существовало на момент завершения предыдущего цикла рисования. Это означает, что последующие геометрические преобразования начнутся не от исходного состояния ОСК и их выполнение не приведет к желаемому результату. Команда `glLoadIdentity` позволяет предотвратить такую неправильность работы программы.

3.5. Расстановка объектов по глубине

При работе с двумерными объектами координату глубины (координату z) можно не использовать. Однако все команды геометрических преобразований работают в трехмерном пространстве и потому имеют своеобразие, которое следует учитывать и в двумерной графике.

В процессе расстановки объекты могут располагаться с частичным перекрыванием. Оно возникает даже тогда, когда координата z объектов одинакова. В этом случае по умолчанию сверху окажется объект, нарисованный программой позже. Коды его цветояркости позже попадут в буфер кадра и заменят собой коды, занесенные туда ранее. При этом неважно, какую глубину имеют объекты. Но в библиотеке OpenGL есть и другой режим вывода перекрывающихся объектов, основанный на так называемом тесте глубины. В этом режиме глубина объектов, т.е. координата z их элементов, фиксируется в выделенной для этого памяти – буфере глубины. В процессе вывода на экран перекрывающихся объектов расстояния от них до наблюдателя сравниваются между собой с помощью специального алгоритма, который упорядочивает объекты по координате z . Точнее, по глубине упорядочиваются элементы объектов, претендующие на закраску каждого пикселя окна вывода. В результате более близкий к наблюдателю объект на экране заслонит более далекие объекты. В случае равенства координат z двух объектов (их элементов) последовательность их вывода

зависит от настройки алгоритма буфера глубины. Это и есть тест глубины.

По умолчанию тест глубины не работает. Для его включения нужно применить разрешающую команду. Вообще в OpenGL имеется две команды, которые разрешают и запрещают различные возможности. Это `glEnable(<параметр>);` и `glDisable(<параметр>);`. Их параметр – это текстовая строка, которая определяет, что разрешается или запрещается. Чтобы разрешить упорядочение объектов по глубине, в начало программы нужно вставить команду

```
glEnable(GL_DEPTH_TEST); .
```

В любой момент времени это разрешение можно отменить командой

```
glDisable(GL_DEPTH_TEST); ,
```

после чего объекты снова будут выводиться на экран в последовательности рисования их программой.

Чтобы подготовить буфер глубины к работе, его нужно очистить командой

```
glClear(GL_DEPTH_BUFFER_BIT); ,
```

которая также вставляется в начало программы. После такой подготовки буфер глубины можно использовать. По умолчанию настройка теста глубины такова, что выводится на экран тот объект, элементы которого ближе к наблюдателю, чем элементы других объектов, т.е. координата *z* проходящего тест объекта строго меньше координаты *z* других объектов. Можно настроить тест глубины и по-другому. Например, на передний план может выходить объект, координата *z* которого меньше или равна, а может быть и больше координаты *z* других объектов. Настройка теста глубины выполняется с помощью команды `glDepthFunc`.

Задание по расстановке объектов для самостоятельного выполнения (углубленное задание)

1. Примените к рисованию первой стрелки команду масштабирования с отрицательным значением первого параметра, например со значением (-1) . Поясните эффект.

2. Определите набор и последовательность геометрических преобразований, необходимых для поворота на заданный угол всей сцены, показанной на рис. 1.

3. Определите последовательность команд, которая позволяет придать каждой стрелке сцены свой цвет.

Контрольные вопросы к разделу 3

1. Для чего служит дисплейный список в OpenGL?

2. Объясните различие в выполнении следующих команд:

```
glNewList(1, GL_COMPILE); ... glEndList();
```

```
glNewList(1, GL_COMPILE_AND_EXECUTE); ... glEndList();
```

3. Какая команда вызывает дисплейный список на исполнение?

4. Для чего служат команды `glTranslate`, `glRotate`, `glScale`?

Сколько параметров они могут иметь? Что задают их параметры?

5. Для чего предназначена стековая память OpenGL? Как пользоваться командами `glPushMatrix()`; и `glPopMatrix()`;

6. Когда целесообразно применять команду `glLoadIdentity()`;

7. Что такое тест глубины, буфер глубины?

8. Как включить и выключить тест глубины?

4. НАЛОЖЕНИЕ ТЕКСТУРЫ НА ДВУХМЕРНЫЕ ОБЪЕКТЫ

4.1. Процесс наложения текстуры

Текстура в компьютерной графике – это двухмерный массив разноцветных элементов – текселей (texel – texture element). Текстура располагается в своей системе координат – СКТ, где каждый тексель получает две текстурные координаты s и t . Геометрический примитив, как известно, описывается своими характерными точками, каждая из которых имеет две геометрические координаты (координату z при текстурировании можно не учитывать). Суть процесса текстурирования заключается в установлении соответствия между координатами текселей и геометрическими координатами. Другими словами, каждому элементу геометрического примитива ставится в соответствие тексель. При выводе изображения на экран каждому элементу примитива сопоставляется пиксель окна вывода. Следовательно, в процессе наложения текстуры для каждого пикселя, попавшего внутрь контура примитива, находится тексель текстуры. При этом в случае наложения большой текстуры на маленький примитив текстура будет сжата, в противном случае – растянута. Закон сопоставления характерных точек примитива и текстуры выбирает разработчик, а нахождение текселей для каждого пикселя изображения примитива выполняют команды графической библиотеки. Текстурный рисунок обычно изготавливается в растровом графическом редакторе и сохраняется в одном из стандартных форматов. В графической библиотеке текстура представляется и используется в некотором внутреннем формате, который не совпадает ни с одним стандартным. Следовательно, в графической программе должно осуществляться конвертирование рисунка во внутренний формат библиотеки, т.е. преобразование текстурного рисунка в текстурный образец.

С учетом приведенных сведений можно сформулировать последовательность действий, необходимых для наложения текстуры на геометрический примитив:

- изготовить рисунок текстуры;
- разработать закон наложения текстуры;
- определить текстурные координаты текселей, соответствующих характерным точкам примитива;

- разрешить текстурирование;
- конвертировать текстурный рисунок в текстуру (текстурный образец);
- используя команды графической библиотеки, задать соответствие текстурных и геометрических координат;
- записать фрагмент программы, рисующей текстурированный примитив.

Из семи перечисленных действий три первые требуют принятия творческих решений, а четыре последние – применения графических технологий.

4.2. Разработка закона текстурирования

При изготовлении текстурного рисунка нужно соблюдать несколько простых правил:

- размеры холста в пикселях должны быть кратны степени числа 2, т.е. годятся размеры 64×64 , 64×512 , 256×8 и другие подобные;
- сохранять рисунок нужно в формате bmp в той же папке, где сохраняется графическая программа;
- нужно учитывать соотношение между пиксельными размерами примитива и текстурного рисунка, т.е. для примитива с большими размерами нужно создавать и больший рисунок.

Несоблюдение этих правил приводит к усложнению, а порой и к невозможности правильного текстурирования.

Для выбора закона текстурирования удобно изобразить текстурный рисунок в СКТ и наложить на него проекцию геометрического примитива или целого объекта (зависит от замысла разработчика). В качестве примера можно рассмотреть наложение на объект-стрелку простой текстуры в виде двухцветного поля. На рис. 6 показаны различные законы текстурирования стрелки.

На рис. 6,а одна текстура накладывается на весь объект (стрелку), а на рис. 6,г,д показано раздельное текстурирование частей стрелки, позволяющее разнообразить вид текстурированного объекта.

При выборе законов, показанных на рис. 6,а,б, вид текстурированного объекта понятен из рисунков. Рисунок 6,в показывает только соответствие геометрических и текстурных координат и не свидетель-

ствуется об искажении формы стрелки. Стрелка командами рисования будет отображена в соответствии с заданными геометрическими координатами, и форма ее не изменится. А вот текстура будет наложена на нее с искажениями: вдоль одних сторон полигонов текстура будет сжата, вдоль других – растянута. Искажения будут особенно заметны, если текстура будет иметь вид регулярного узора, например шахматного поля.

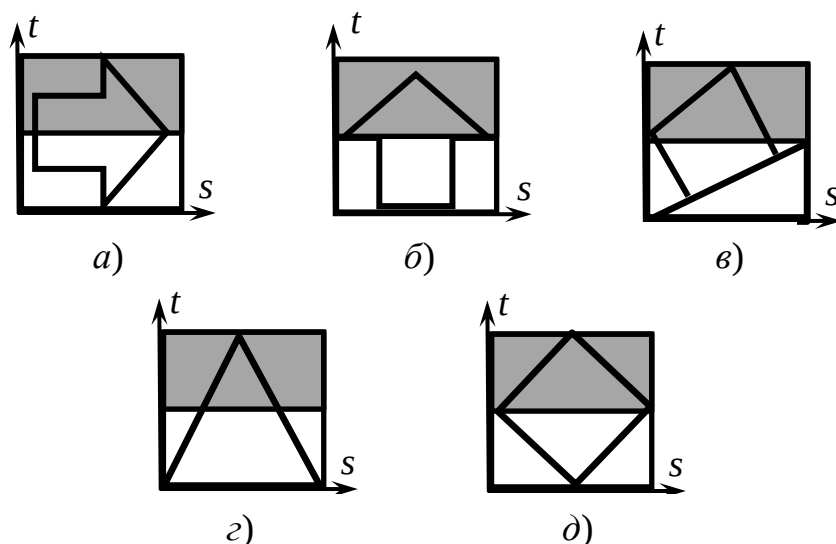


Рис. 6. Варианты текстурирования объекта

Выбрав закон наложения текстуры, можно найти текстурные координаты s , t для всех характерных точек примитивов. В библиотеке OpenGL текстура, независимо от ее размеров, располагается в диапазоне от 0 до 1 по каждой координате СКТ. Введя по этим координатам шкалу отсчетов, легко найти текстурные координаты характерных точек объекта по рисунку. Например, для закона текстурирования, сходного с показанным на рис. 6,б, эти координаты видны из рис. 7.

Чтобы проверить, насколько правильно выбран закон наложения текстуры и не приведет ли он к искажению текстуры на изображении, нужно сравнить отношения геометрических и соответствующих им текстурных диапазонов координат для характерных участков объекта. Например, на рис. 3 длина и высота стрелки в геометрических координатах одинаковы: 0,8 единицы, а в текстурных координатах на рис. 7 они не одинаковы: 0,8 и 1,0. Это означает, что на изображении стрелки текстура будет немного растянута по координате y^o (по рис. 3).

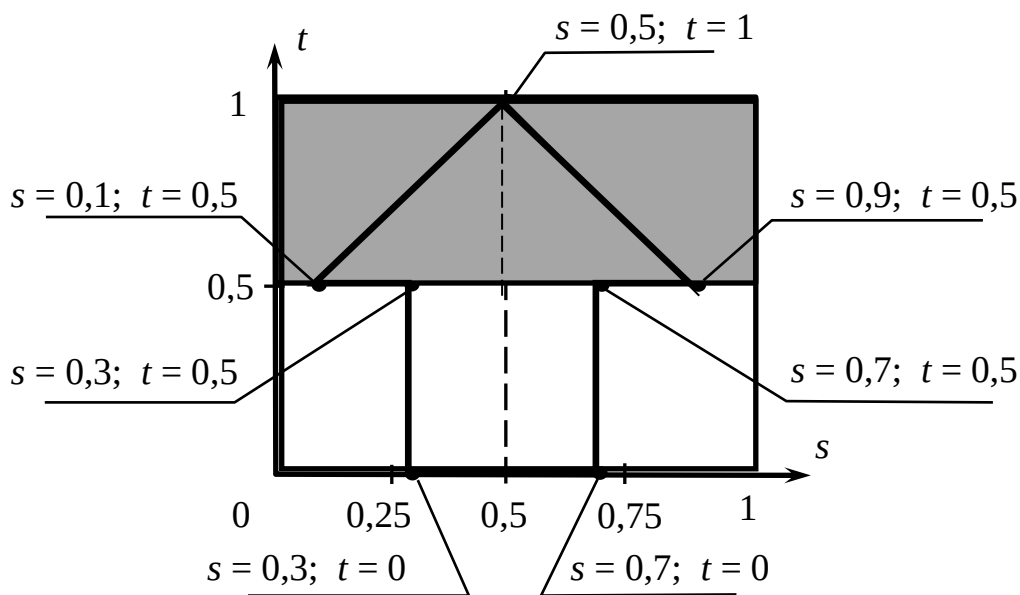


Рис. 7. Нахождение текстурных координат для характерных точек объекта

4.3. Программирование наложения текстуры

По умолчанию примитивы сцены получают однотонную заливку, цвет которой задается командой `glColor`. Для того чтобы получить возможность накладывать текстуру, нужно воспользоваться уже известной разрешающей командой `glEnable` с параметром `GL_TEXTURE_2D`. Команда вставляется в графическую программу до команд рисования. Тем самым включается режим текстурирования, который действует до тех пор, пока не будет выключен. Выключить (запретить) текстурирование можно в любой момент, применив команду `glDisable` с тем же параметром `GL_TEXTURE_2D`, при этом вновь будет включен режим цветной заливки. Таким образом, применяя названные команды в нужные моменты времени, можно чередовать отображение различных объектов сцены с текстурированием и с заливкой.

Изготовленный в графическом редакторе рисунок должен быть конвертирован графической программой во внутренний формат OpenGL. Для этого в одном из расширений библиотеки (расширении GLAUX) имеются встроенные средства. К ним относится функция

`auxDIBImageLoad(<параметр>); ,`

параметром которой является текстовая строка, содержащая имя файла рисунка с расширением bmp и путь к файлу. Функция выполняет конвертирование рисунка, размещает его в некоторой области памяти с определенной структурой и возвращает указатель на эту область. Переменная-указатель описывается в программе как указатель на тип AUX_RGBImageRec, который задает структуру памяти текстуры.

Указанная функция хорошо работает в программах, написанных на С-подобных языках. В программах, написанных в среде Lazarus, для конвертации с успехом может быть применен свободно распространяемый программный модуль Textures.pas, автором которого является голландский программист Ян Хорн. Модуль преобразует рисунки формата bmp, jpeg, tga в текстурные образцы библиотеки OpenGL. Для использования его нужно присоединить к программе в разделе uses.

В состав модуля Textures входит процедура конвертации форматов:

```
LoadTexture('<имя рисунка и путь>', <имя текстуры>,
            <признак ресурса>); ,
```

имеющая три параметра. Первый – текстовая строка в апострофах, которая содержит путь к файлу рисунка (диск, каталог, подкаталог...) и его имя с расширением. Если рисунок сохранен в той же папке, где хранится программа, путь указывать не нужно. Второй параметр команды – это имя текстуры (текстурного образца), в которую конвертируется рисунок. В качестве текстурных имен в OpenGL используются целые числа, поэтому в программе нужно ввести переменную целого типа (встроенного типа OpenGL GLuint – беззнаковое целое – или типа word, поддерживаемого средой Lazarus). Третьим параметром процедуры LoadTexture является булевский признак TRUE или FALSE. Если рисунок сохранен в одной папке с программой в формате bmp и его имя TexRisl, а имя текстуры TexName1, то процедура конвертации вызывается так:

```
LoadTexture('TexRisl.bmp', TexName1, FALSE); .
```

После создания текстуры она становится текущей, т.е. может тут же использоваться для текстурирования. Если в сцене несколько объектов имеют различные текстуры, процедуру конвертирования приходится применять неоднократно. Это можно сделать двумя способами. Во-первых, процедуру конвертации можно вызывать перед текстурированием каждого очередного объекта, т.е. действовать по схеме:

- конвертация текстуры 1;
- наложение текстуры 1;

- конвертация текстуры 2;
- наложение текстуры 2 и так далее.

В этом случае в программе достаточно иметь одну переменную для имени текстуры, и эта переменная поочередно будет связываться со всеми текстурными образцами, которые будут заноситься в одну и ту же область памяти. Такой вариант прост, но несколько «тормозит» программу, так как конвертирование рисунков (особенно, большого размера) в ходе рисования требует дополнительного времени. Второй вариант конвертации грамотнее и эффективнее. Он предполагает, что все текстуры заготавливаются заранее и получают свои уникальные имена. Для этого процедуры конвертации вызываются в начале программы с различными значениями второго параметра. Теперь для того, чтобы выбрать необходимую для некоторого объекта текстуру, например с именем `TexName`, ее надо перед наложением сделать текущей. Для этого служит следующая команда библиотеки OpenGL:

```
glBindTexture(GL_TEXTURE_2D, TexName); .
```

В этом случае возникает такая схема текстурирования:

- конвертация текстуры 1;
- конвертация текстуры 2 и так далее;
- вызов текстуры 1 в качестве текущей;
- наложение текстуры 1;
- вызов текстуры 2 в качестве текущей;
- наложение текстуры 2 и так далее.

Следует отметить, что для программы, написанной в среде Lazarus, команда `glBindTexture` является внешней процедурой, описание которой содержится в файле библиотеки `opengl32.dll`. Поэтому для использования команды эта процедура должна быть упомянута в разделе программы `implementation`:

```
procedure glBindTexture(target: GLenum; texture: GLuint);
    stdcall; external opengl32; ,
```

здесь `GLenum` и `GLuint` – встроенные типы данных библиотеки OpenGL, в Lazarus им соответствует тип `Cardinal`.

Когда текстурный образец сделан текущим, можно приступать к непосредственному наложению текстуры на примитивы. Для этого нужно каждой характерной точке примитива сопоставить ее текстурные координаты. Геометрические координаты характерных вершин

задаются командами glVertex. Для привязки к ним текстурных координат перед каждой командой glVertex вставляется команда

```
glTexCoord2f(s,t); ,
```

в которой s и t – соответствующие текстурные координаты, заданные константами или переменными вещественного (float) типа. В качестве примера ниже приведен фрагмент программы, рисующий текстурированный треугольник стрелки. Геометрические координаты вершин треугольника взяты из табл. 2 с учетом рис. 3, текстурные – с учетом рис. 7:

```
glEnable(GL_TEXTURE_2D);
LoadTexture('TexRis1.bmp',TexName1, FALSE);
//задание режима наложения текстуры
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
//задание белого цвета «подложки» текстуры
glColor3f(1.0,1.0,1.0);
//рисование с текстурированием
glBegin(GL_TRIANGLES);
    glTexCoord2f(0.1,0.5);
    glVertex3f(0.0,0.4,0.0);
    glTexCoord2f(0.5,1.0);
    glVertex3f(0.4,0.0,0.0);
    glTexCoord2f(0.9,0.5);
    glVertex3f(0.0,-0.4,0.0);
glEnd(); .
```

Некоторые команды в приведенном фрагменте нуждаются в пояснениях.

Как уже отмечалось, текстура может накладываться на примитив со сжатием или с растяжением. В первом случае на изображение могут пропасть мелкие элементы, во втором может появиться ступенчатость наклонных линий. Для повышения качества текстурирования применяют различные фильтры. Их суть в том, что цветояркость каждого пикселя, принадлежащего примитиву, определяется не одним текселем текстуры, а находится с помощью усреднения цветояркости несколь-

ких текселей из некоторой окрестности. Закон усреднения может быть различным. Чем он сложнее, тем выше качество изображения, но ниже скорость его обработки. Оптимальным можно считать линейный закон усреднения (GL_LINEAR), который и задается в приведенном фрагменте программы командами glTexParameter для случая растяжения (MAG_FILTER) и сжатия (MIN_FILTER) текстуры.

Если в программе перед текстурированием командой glColor задан цвет пера, отличный от белого, то при текстурировании примитива произойдет смешивание цветов текстуры и цветовой «подложки» (заливки) примитива. Такой режим наложения текстуры установлен в библиотеке OpenGL по умолчанию. В результате цветовая гамма текстуры на изображении окажется измененной. Этот эффект можно использовать для повышения разнообразия цветовых решений. Если же требуется, не меняя настроек, получить на изображении текстуру в исходном виде, то перед текстурированием следует задать белый цвет пера, что и сделано в приведенном фрагменте программы. Режим наложения текстуры может быть изменен командой glTexEnv. Например, следующий вариант ее написания включает режим замены цвета заливки примитива цветами текстуры:

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,  
          GL_REPLACE); .
```

4.4. Дополнительные возможности текстурирования

Как уже отмечалось, текстурные координаты s , t в СКТ лежат в диапазоне от 0 до 1. Однако это не означает, что параметры команды glTexCoord могут лежать только в этом диапазоне. Задавая их величину больше единицы или отрицательной, можно реализовать дополнительные возможности текстурирования. Если, например, наложить текстуру на четырехугольник следующим набором команд:

```
glBegin(GL_QUADS);  
    glTexCoord2f(0.0,0.0);  
    glVertex2f(0.0,0.0);  
    glTexCoord2f(0.0,1.0);  
    glVertex2f(0.0,0.5);
```

```
glTexCoord2f(5.0,1.0);
glVertex2f(0.5,0.5);
glTexCoord2f(5.0,0.0);
glVertex2f(0.5,0.0);
```

```
glEnd(); ,
```

то произойдет пятикратное наложение (размножение) текстуры в горизонтальном направлении. Размножением текстуры часто пользуются при отображении протяженных поверхностей с хаотичным рисунком (камни, песок, трава). Применение размножения в этом случае позволяет раскрасить поверхность больших размеров, используя небольшую текстуру, чем достигается экономия памяти вычислительной системы.

Использование отрицательных значений текстурных координат позволяет накладывать текстуру в обратном направлении, т.е. в зеркальном отражении. Возможности и логика размножения и зеркального отражения текстуры будут более понятными, если принять образ размещения текстуры (на примере цветка) в СКТ в соответствии с рис. 8.

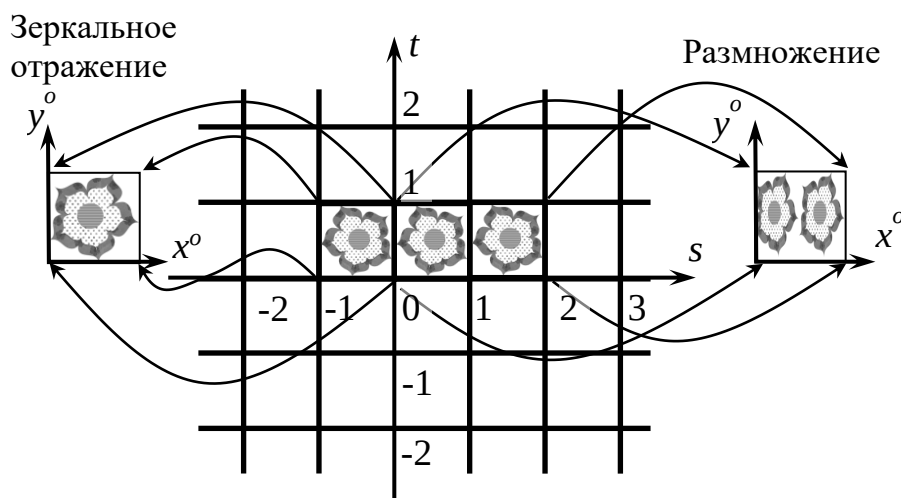


Рис. 8. Размещение текстуры в СКТ

Стрелками на рис. 8 показано установление соответствия между геометрическими и текстурными координатами. Следует отметить, что при текстурировании можно задавать дробные значения текстурных координат. Возникающий при этом эффект можно понять, пользуясь рисунком, аналогичным рис. 8.

Задание по текстурированию для самостоятельного выполнения (углубленное задание)

1. Создайте текстурный фон сцены в виде нескольких разноцветных кругов. Подсказка: текстура не может существовать сама по себе, она должна быть наложена на какой-либо геометрический примитив.
2. Наложите текстуру на квадрат в соответствии с рис. 9.

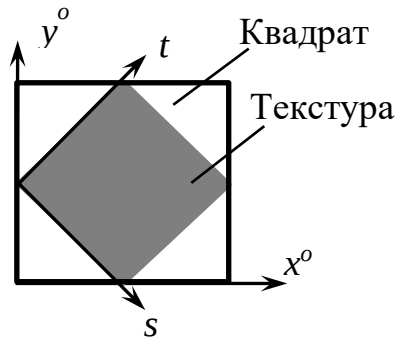


Рис. 9. Задание по наложению текстуры

Контрольные вопросы к разделу 4

1. В чем заключается суть процесса текстурирования?
2. Чем отличается текстурный образец OpenGL от рисунка, сделанного в графическом редакторе?
3. В чем заключается разработка закона наложения текстуры?
4. Как наложить текстуру на объект без искажения?
5. Где нужно располагать команду `glTexCoord2f(s,t)`? Что она выполняет?
6. Почему имя процедуры `LoadTexture`, которая служит для конвертации текстуры в формат OpenGL, не имеет префикса?
7. Для чего нужна команда `glBindTexture`? Когда ее нужно использовать?
8. В каком диапазоне лежат текстурные координаты OpenGL?
9. В каких случаях применяются следующие команды OpenGL:
`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);`

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); ?
```

10. К чему приведет использование отрицательных значений текстурных координат?

11. Что нужно сделать, чтобы размножить текстуру?

5. ОТОБРАЖЕНИЕ ДИНАМИЧЕСКИХ СЦЕН

5.1. Описание динамики объектов

Динамика объекта может проявляться в виде изменения его размеров или местоположения на экране без изменения формы, в виде изменения формы объекта без изменения его местоположения и в виде одновременного изменения упомянутых параметров в разных сочетаниях. Любая динамика привязывается ко времени. Сутью ее программирования является циклическая перерисовка объекта с изменением его геометрических параметров в каждом цикле.

Изменение местоположения объекта

Геометрические преобразования объектов, как известно, задаются командами `glTranslate`, `glRotate`, `glScale`. Их параметры при каждой перерисовке могут меняться с постоянным приращением, например: для сдвига $x:=x+dx$; для поворота $ALFA:=ALFA+dALFA$; для масштабирования $kx:=kx+dkx$. Команды геометрических преобразований применяются ко всей ОСК. При использовании приращений неизменной величины закон преобразований будет линейным. Более интересным случаем является нелинейная динамика, при которой параметры команд должны вычисляться. Например, на практике часто применяют круговую или, в общем случае, эллиптическую замкнутую траекторию движения объектов.

Движение объекта по эллиптической траектории может происходить без поворота («параллельно самому себе») и с поворотом. Образно можно представить себе первый вариант на примере движения кабины колеса обозрения, а второй вариант – на примере картинке, приклеенной к концу секундной стрелки часов. Для реализации первого варианта нужно циклически вычислять координаты точек, принадлежащих эллиптической траектории, и использовать их как параметры команды сдвига. Сдвиг должен применяться ко всей ОСК, в которой описан объект. При этом не надо забывать, что команда сдвига в OpenGL работает по приращениям.

Со школы известно каноническое уравнение эллипса с центром в начале координат:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

где a и b – полуоси эллипса.

Для применения в компьютерной графике оно весьма неудобно. Действительно, приняв за аргумент координату x и изменяя ее в интервале от $(-a)$ до $(+a)$, получим

$$y_{1,2} = \pm \frac{b}{a} \sqrt{a^2 - x^2}.$$

Для каждого значения x нужно найти два значения y : y_1 и y_2 , так как эллипс описывается неоднозначной квадратической функцией. Тогда для последовательного перебора точек эллиптической траектории придется сначала с постоянным шагом изменять x от $(-a)$ до $(+a)$ и вычислять y_1 , а затем изменять x от $(+a)$ до $(-a)$ и вычислять y_2 . Алгоритм получается усложненным, а точки на траектории будут расставлены неравномерно. Недостатки можно минимизировать, используя параметрическое уравнение эллипса:

$$\begin{aligned} x &= a \cos\left(\frac{2\pi}{T}t\right) + x_0, \\ y &= b \sin\left(\frac{2\pi}{T}t\right) + y_0, \end{aligned}$$

где x_0, y_0 – координаты центра эллипса; t – параметр-время, который изменяется в интервале от 0 до T ; T – длительность однократного прохождения объектом эллиптической траектории.

В качестве параметра t здесь выступает время, тогда приведенные выражения получают понятный смысл. Они описывают поведение (изменение) координат x и y во времени. Траекторию нужно разделить на некоторое количество N шагов и установить временной интервал T , за который траектория будет пройдена. Тогда удобно принять

$$t = \frac{n}{N}T = n\Delta t,$$

где n – номер шага по траектории ($n = 0, \dots, N$); Δt – длительность шага ($\Delta t = T/N$).

Если объект движется по эллиптической траектории с поворотом, нужно после команды сдвига использовать команду поворота. Угол по-

ворота в общем случае нужно вычислять, однако при круговой траектории это сделать просто: он равен $\left(\frac{2\pi}{T}t\right)$. Более того, в случае круговой траектории с центром в точке (x_0, y_0) и радиусом R алгоритм отображения упрощается. В очередном цикле отображения нужно сначала выполнить сдвиг ОСК в точку (x_0, y_0) , затем повернуть ОСК на угол $\left(\frac{2\pi}{T}t\right)$, а затем сдвинуть ее по оси y^0 на величину радиуса R . Логика алгоритма легко понять, нарисовав результаты перечисленных преобразований.

Более общим случаем динамики объектов является их движение по траектории произвольной формы. В качестве примера можно рассмотреть движение объекта по криволинейной траектории, показанной на рис. 10.

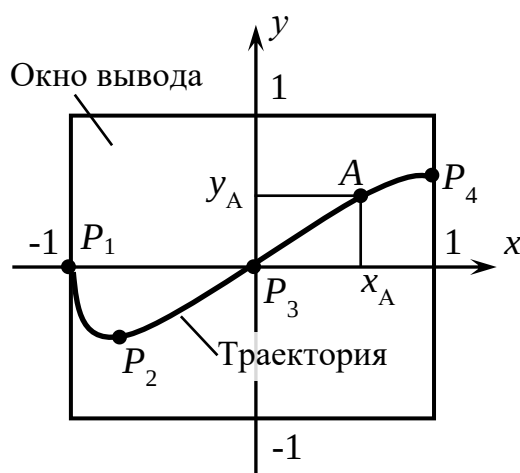


Рис. 10. Нелинейная траектория движения объекта в мировой системе координат

Подобная траектория не может иметь более двух экстремумов и, следовательно, описывается уравнением третьей степени:

$$y = ax^3 + bx^2 + cx + d,$$

где a, b, c, d – коэффициенты формы, определяющие своеобразие кривой.

Для каждого значения x кривая имеет одно значение y , т.е. одну точку. Например, для $x = x_A$ это $y = y_A$, что соответствует точке A . В этом случае коэффициенты формы можно найти, наложив ограничения на четыре точки кривой. Пусть это точки $P_1, \dots, P_4$, координаты которых заданы разработчиком так, как показано в табл. 4.

Координаты характерных точек траектории

Координата	Точка			
	P_1	P_2	P_3	P_4
x	-1	-0,5	0	1
y	0	-0,5	0	0,7

Тогда, подставляя координаты этих точек в уравнение траектории, можно составить систему из четырех уравнений с четырьмя неизвестными коэффициентами формы:

$$\begin{aligned} 0 &= -a + b - c + d, \\ -0,5 &= -0,125a + 0,25b - 0,5c + d, \\ 0 &= d, \\ 0,7 &= a + b + c + d. \end{aligned}$$

Решив систему любым известным методом, можно найти численные значения коэффициентов формы и, подставив их в уравнение траектории, получить расчетное выражение. Теперь в графической программе нужно организовать изменение аргумента x с некоторым шагом. Каждый шаг привязывается к отсчету системного времени (обращение к системному таймеру выполняется своими средствами в каждой среде программирования). Вычисленные значения величин x и y передаются в команду сдвига в качестве параметров.

Более сложным случаем является движение объекта по траектории, описываемой многозначной функцией. Для нее каждому значению аргумента x соответствует несколько значений y , как это показано на рис. 11.

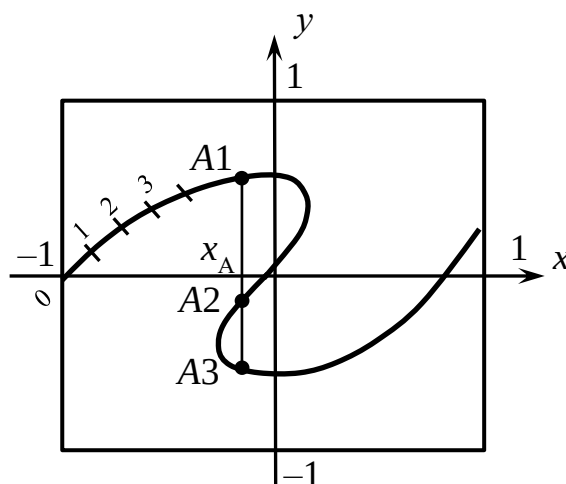


Рис. 11. Криволинейная траектория, описываемая многозначной функцией

Здесь придется применить параметрическое описание траектории, например:

$$x = a_x t^3 + b_x t^2 + c_x t + d_x,$$

$$y = a_y t^3 + b_y t^2 + c_y t + d_y,$$

где t – параметр-время; $a_x, \dots, d_y$ – коэффициенты формы.

Для нахождения коэффициентов формы можно применить уже знакомый прием:

- исходя из желаемого вида траектории, задать на ней четыре характерные точки, для которых найти координаты $x_i, y_i (i = 1, \dots, 4)$. Точки должны быть привязаны к значениям t_i , т.е. моментам времени, в которые они будут достигнуты;

- подставляя координаты и значения времени в параметрическое описание траектории, получить две системы из четырех уравнений с четырьмя неизвестными каждая. При этом в первой системе уравнений будут использоваться только x_i , а во второй – только y_i ;

- решив системы уравнений, найти значения коэффициентов формы для описания каждой координаты траектории.

При программной реализации траектории шаги по ней нужно привязать к отсчетам системного таймера, расставленным с интервалом Δt . Вся траектория должна составлять N шагов.

При отладке программы часто возникает необходимость изменить форму траектории. Даже если эти изменения невелики, они вызывают необходимость пересчета всех коэффициентов формы. Применив другую форму описания траектории, можно существенно облегчить процесс ее редактирования. Речь идет о кривой Безье, предложенной французским специалистом по компьютерной графике Пьером Безье (Pierre Étienne Bézier). В форме Безье траектория описывается четырьмя характерными (опорными) точками P_1, P_2, P_3, P_4 , две из которых (P_1, P_4) принадлежат концам кривой, а две другие (P_2, P_3) – лежат на касательных, проведенных через конечные точки, и задают изгиб кривой. Кривая Безье показана на рис. 12, а ее математическое описание в параметрической форме имеет вид

$$x = (1-t)^3 x_1 + 3t(1-t)^2 x_2 + 3t^2(1-t) x_3 + t^3 x_4,$$

$$y = (1-t)^3 y_1 + 3t(1-t)^2 y_2 + 3t^2(1-t) y_3 + t^3 y_4,$$

где $x_1, \dots, x_4$ и $y_1, \dots, y_4$ – координаты четырех опорных точек; t – параметр, изменяющийся в диапазоне от 0 до 1.

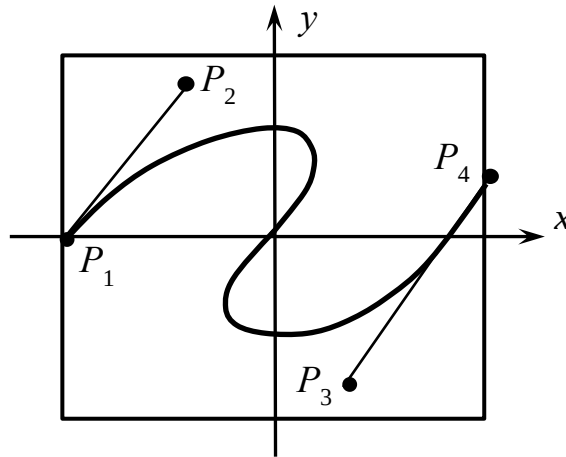


Рис. 12. Криволинейная траектория объекта в виде кривой Безье

Выбрать P_1 и P_4 просто: это начало и конец желаемой траектории. Для нахождения P_2 и P_3 нужно знать значения производных $\partial x / \partial t$ и $\partial y / \partial t$ в начальной и конечной точках. Тогда

$$\left(\frac{\partial x}{\partial t} \right)_{x=x_1} = 3(x_2 - x_1), \left(\frac{\partial x}{\partial t} \right)_{x=x_4} = 3(x_4 - x_3),$$

$$\left(\frac{\partial y}{\partial t} \right)_{y=y_1} = 3(y_2 - y_1), \left(\frac{\partial y}{\partial t} \right)_{y=y_4} = 3(y_4 - y_3),$$

откуда можно найти координаты точек P_2, P_3 . Найти значения частных производных по t сложно, поэтому лучше применить простой прием нахождения координат точек P_2, P_3 . Желаемую траекторию нужно нарисовать в МСК, ориентировочно провести касательные к кривой в ее начальной и конечной точках и поставить на них точки P_2 и P_3 . При расстановке этих точек следует пользоваться простым правилом: чем сильнее вытянута траектория в направлении касательной, тем дальше от крайней точки P_1 (P_4) должна стоять на этой касательной средняя опорная точка P_2 (P_3). Получив в результате графических построений координаты опорных точек, их надо подставить в описание кривой Безье. Теперь эти выражения можно использовать для нахождения промежуточных точек траектории. Для редактирования формы траектории достаточно изменить координаты опорных точек и посмотреть на результат.

Изменение формы объекта

Для изменения формы объекта нужно управлять положением его отдельных вершин. В качестве примера можно рассмотреть превращение треугольника в квадрат, его иллюстрирует рис. 13. В процессе превращения происходит изменение положения вершин треугольника. Поскольку число вершин у квадрата на единицу больше, чем у треугольника, в треугольнике нужно заранее предусмотреть дополнительную вершину (на рис. 13 это P_2), которая станет еще одной вершиной квадрата (S_2). Изменение положения вершин показано на рис. 13 стрелками. Для описания перемещения вершины P_i в новое положение S_i удобно применить линейную интерполяцию.

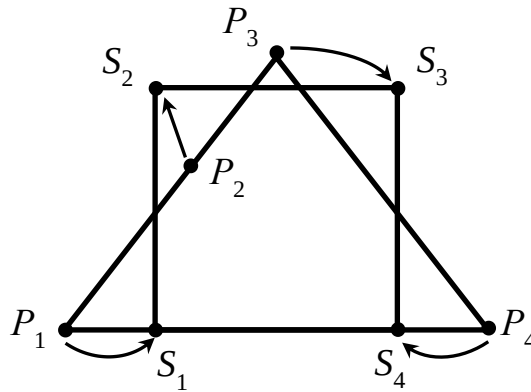


Рис. 13. Движение вершин треугольника при его преобразовании в квадрат

Текущие координаты новой вершины вычисляются по интерполяционным выражениям:

$$x = (1 - \alpha)x_p + \alpha x_s,$$

$$y = (1 - \alpha)y_p + \alpha y_s, \alpha = 0, \dots, 1,$$

где α – интерполяционный коэффициент; x_p, y_p – координаты вершины P_i ; x_s, y_s – координаты вершины S_i .

При $\alpha = 0$ вершина находится в положении P_i , при $\alpha = 1$ – в положении S_i , при $1 > \alpha > 0$ вершина находится в некотором среднем положении на прямой, соединяющей P_i и S_i . Подставив значение α в зависимость от времени t , можно записать:

$$\alpha = \frac{t}{T} = \frac{n\Delta t}{T}, t = 0, \dots, T, n = 0, \dots, N,$$

где T – запланированная длительность перехода вершины из положения P_i в положение S_i ; n – номер шага по траектории перехода вершины от P_i к S_i ; Δt – длительность шага; N – запланированное число шагов перехода вершины от P_i к S_i .

Чем меньше Δt и больше N , тем плавнее будет двигаться вершина. На практике следует выбирать $\Delta t \leq 0,03$ с. Число N зависит от расстояния перехода и выбирается так, чтобы длина одного шага на экране была соизмерима с размером пикселя. Координаты вершин, вычисляемые в цикле по интерполяционным выражениям, используются в качестве параметров команд `glVertex`.

Изменение формы объектов можно получить, применив команду масштабирования `glScale` с неодинаковыми масштабными коэффициентами. Процесс деформации будет плавным, если масштабные коэффициенты вычислять в цикле, например по интерполяционным выражениям:

$$k_x = (1 - \alpha)k_{x1} + \alpha k_{x2},$$

$$k_y = (1 - \alpha)k_{y1} + \alpha k_{y2},$$

где k_{x1} , k_{y1} – начальные, а k_{x2} , k_{y2} – конечные значения масштабных коэффициентов.

Коэффициент интерполяции зависит от времени и определяется так, как было показано для изменения положения вершин.

5.2. Программирование динамики графических объектов

Динамические изображения выводятся на экран по фазам движения. Если частота смены фаз достаточно велика (больше 30 Гц), наблюдатель видит плавную динамику. Для формирования динамических изображений требуются два буфера кадра, в вычислительной системе они называются задним и передним буферами. Пока в заднем буфере накапливается очередная фаза динамики изображения, из переднего на экран выводится изображение предыдущей фазы. После накопления в заднем буфере очередной фазы она быстро переписывается в передний буфер, и процесс повторяется. Если же использовать один буфер кадра, он попеременно должен работать на создание изображения и считывание его на экран, что снижает цветовую насыщенность изображения, а то и делает его мелькающим. По умолчанию в вычислительной системе работает именно один буфер – задний. Чтобы использовать двойную буфериза-

цию, нужно в программе по окончании рисования инициировать подключение переднего буфера. Это делает процедура

SwapBuffers;

или команда расширения GLUT

glutSwapBuffers(); .

Для отображения объекта, движущегося по некоторой траектории, его экранные координаты циклически пересчитываются и используются в команде сдвига `glTranslate`. При этом нужно помнить, что командой сдвига задаются не экранные координаты объекта, а расстояния его перемещения. Каждый следующий сдвиг выполняется относительно предыдущего положения объекта. Это означает, что каждый раз для перемещения объекта в очередную точку траектории нужно либо вычислять приращения координат объекта относительно предыдущего положения, либо «сбрасывать» результат предыдущего сдвига и вычислять новые координаты объекта относительно его исходного положения. Второй вариант легче реализовать: для этого достаточно воспользоваться уже известными командами `glPopMatrix`, `glPushMatrix`. То же можно сказать и про работу с командой поворота `glRotate`.

Контрольные вопросы к разделу 5

1. Чем не подходит для компьютерной графики каноническое уравнение эллипса?
2. В каких случаях можно и в каких случаях нельзя использовать в компьютерной графике явную форму записи криволинейной траектории?
3. Как найти коэффициенты формы кубической кривой?
4. Чем задается форма кривой Безье? Чем это лучше по сравнению с описанием кубической кривой с коэффициентами формы?
5. Как получить изменение формы объекта во времени?
6. Почему для формирования динамических изображений в графической системе нужно использовать два буфера кадра?
7. Какую последовательность действий нужно выбрать, чтобы описать круговое перемещение объекта вокруг некоторого центра с поворотом вокруг своей оси?
8. Как в программе задать требуемую скорость динамики объекта на экране?

ПРИЛОЖЕНИЕ А

Подключение библиотеки OpenGL в среде Lazarus

А.1. Установка компонента OpenGL

Перед началом использования функций библиотеки *OpenGL* в среде *Lazarus* необходимо установить дополнительный компонент *TOpenGLControl*, который по умолчанию не установлен. Для этого необходимо установить библиотеку *OpenGLContext*:

1. В пункте меню «Пакет» (*Package*) выбрать команду «Установить/Удалить пакеты» (*Install/Uninstall Packages*).

2. В открывшемся окне установки новых пакетов отметить «*LazOpenGLContext 0.0.1*» как показано на рис. А.1.

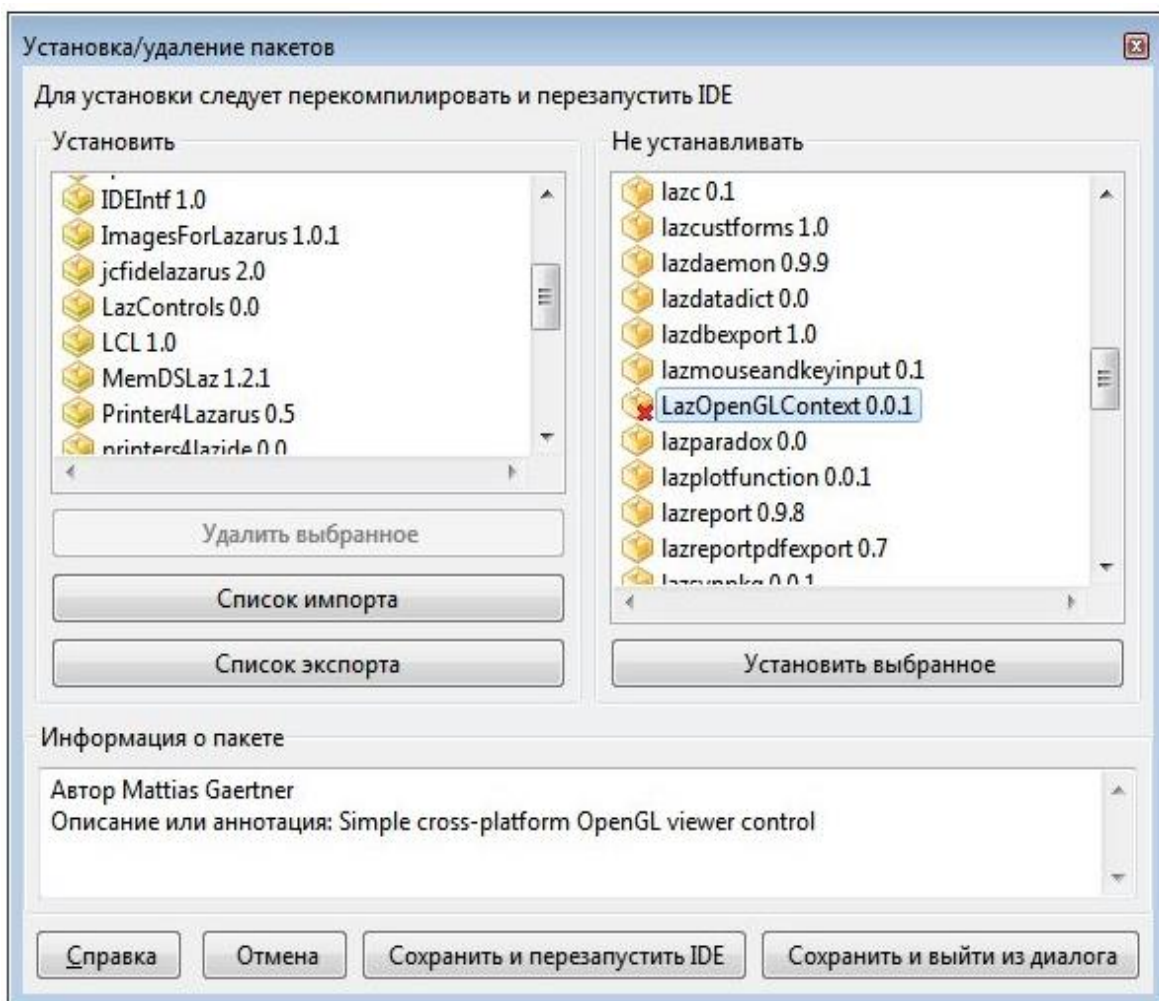


Рис. А.1. Выбор пакета

3. Нажать кнопку «Установить выбранное» (*Install selection*), при этом установка не запускается сразу – пакет только добавляется в список для установки, непосредственно установка произойдет позже. Нужный компонент появится в левой колонке (рис. А.2).

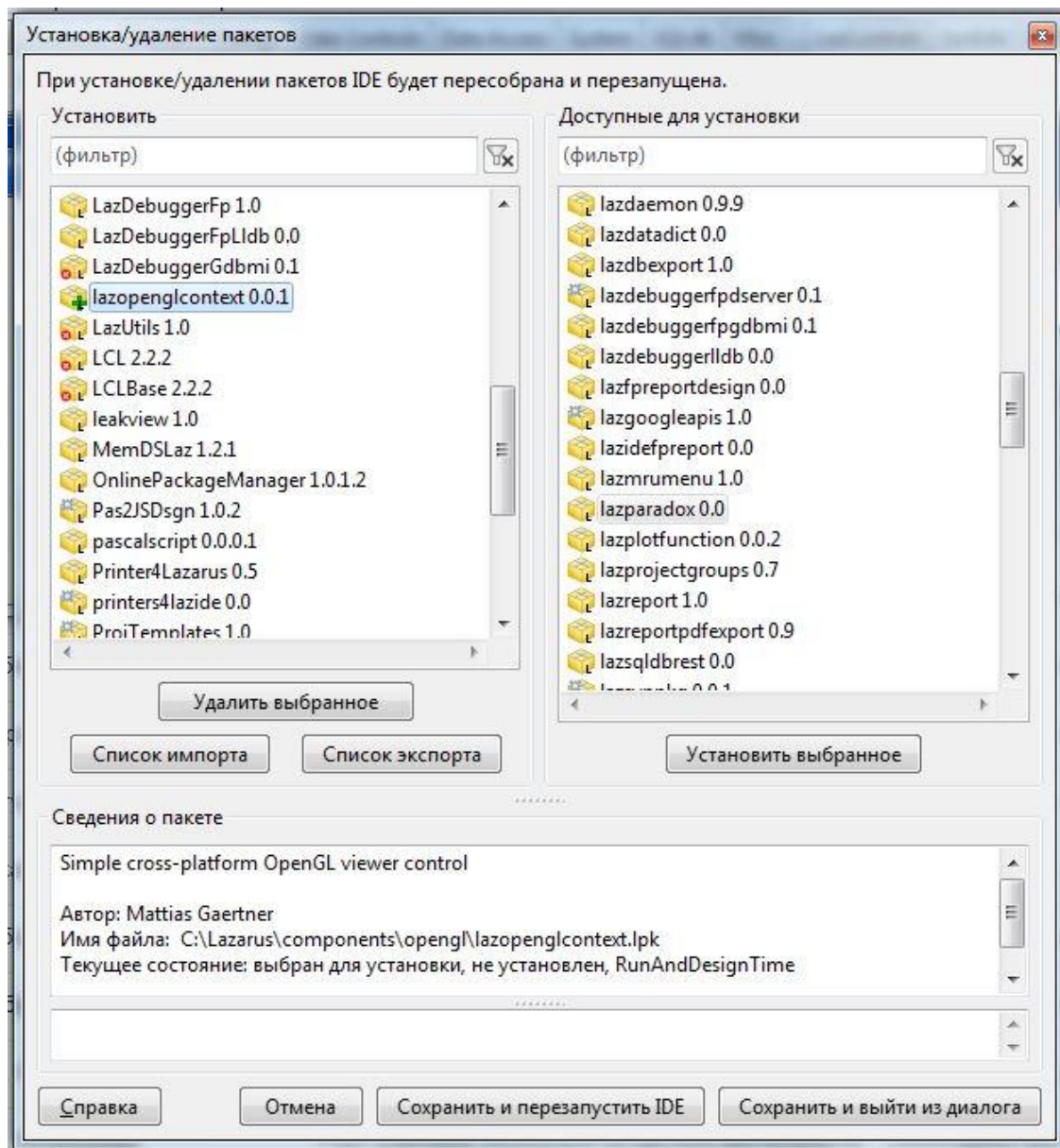


Рис. А.2. Подготовка пакета к установке

4. Нажать кнопку «Сохранить и перезапустить IDE» (*Save and Rebuild IDE*), для установки и пересборки среды Lazarus с компонентом *TOpenGLControl* (*IDE – Integrated Development Environment*) – нажать кнопку «Продолжить» (*Continue*) в окне подтверждения установки нового пакета, который должен присутствовать в списке (рис. А.3).

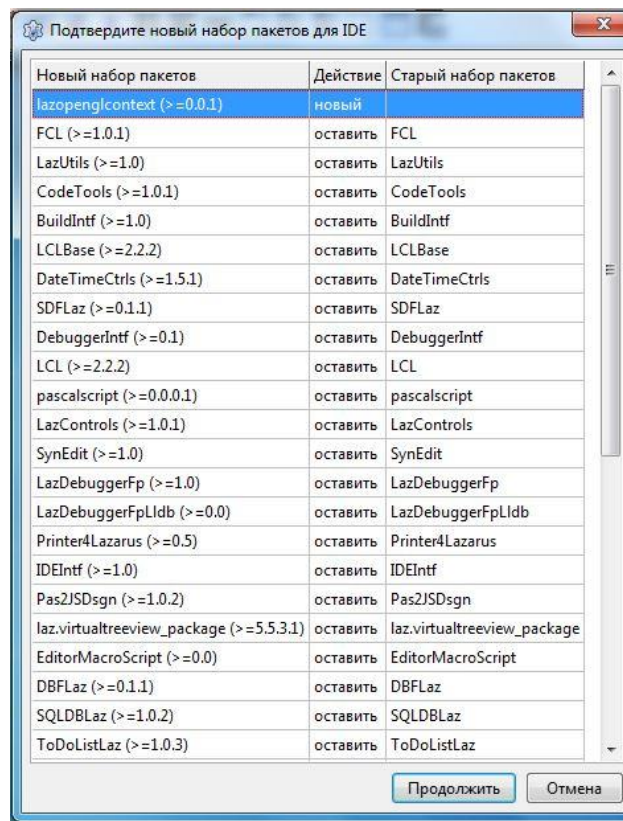


Рис. А.3. Список пакетов к установке

5. Дождаться окончания процесса пересборки (результат показан на рис. А.4).

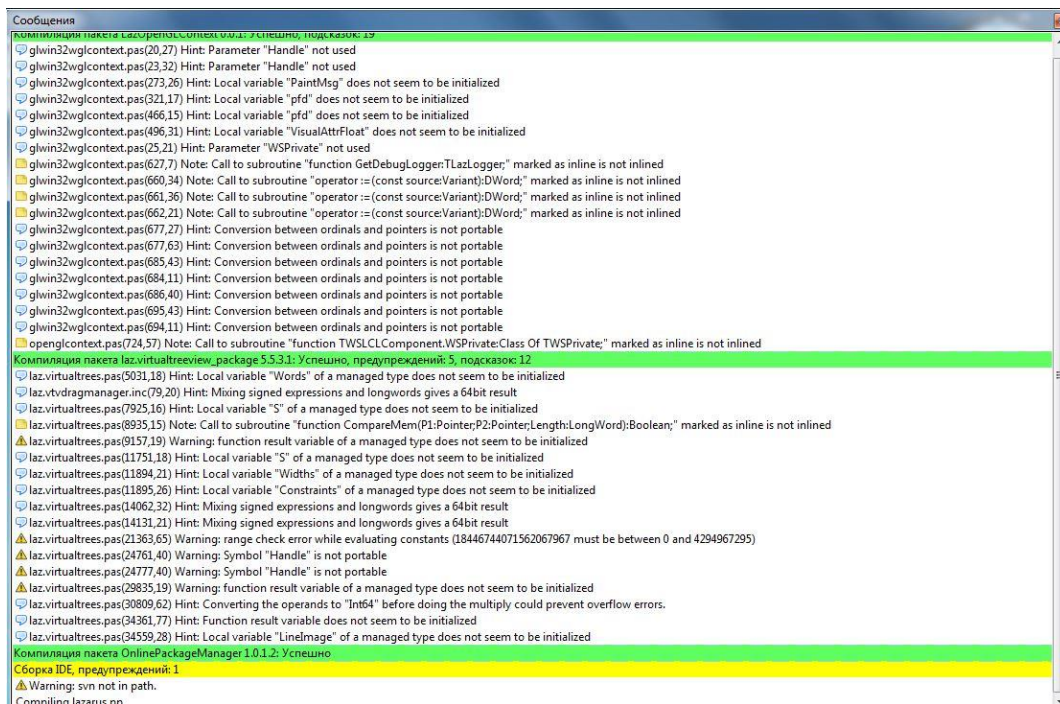


Рис. А.4. Результат процесса пересборки

После успешной установки компонента в библиотеке доступных визуальных компонентов появится новая вкладка *OpenGL*, содержащая один визуальный компонент *TOpenGLControl* (рис. А.5).

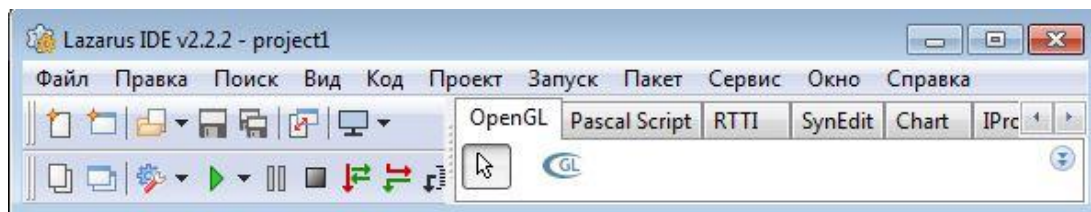


Рис. А.5. Новый компонент в библиотеке компонентов

А.2. Создание проекта с использованием библиотеки OpenGL

Следующим шагом после установки компонента *OpenGL* является его подключение к проекту. Рассмотрим это на примере простого приложения, использующего возможности графической библиотеки *OpenGL*. Приложение должно отображать на экране цветной треугольник. Для этого требуется выполнить шаги, описанные ниже. **Для создания работоспособного приложения требуется выполнение всех шагов в приведенной последовательности – простое копирование текста модуля в новый проект приведет в ошибке!**

- В среде *Lazarus* необходимо создать новый проект: в выпадающем меню «Файл – Создать» выбрать «Приложение». В результате будет создан новый проект с одной главной формой *Form1* (рис. А.6).

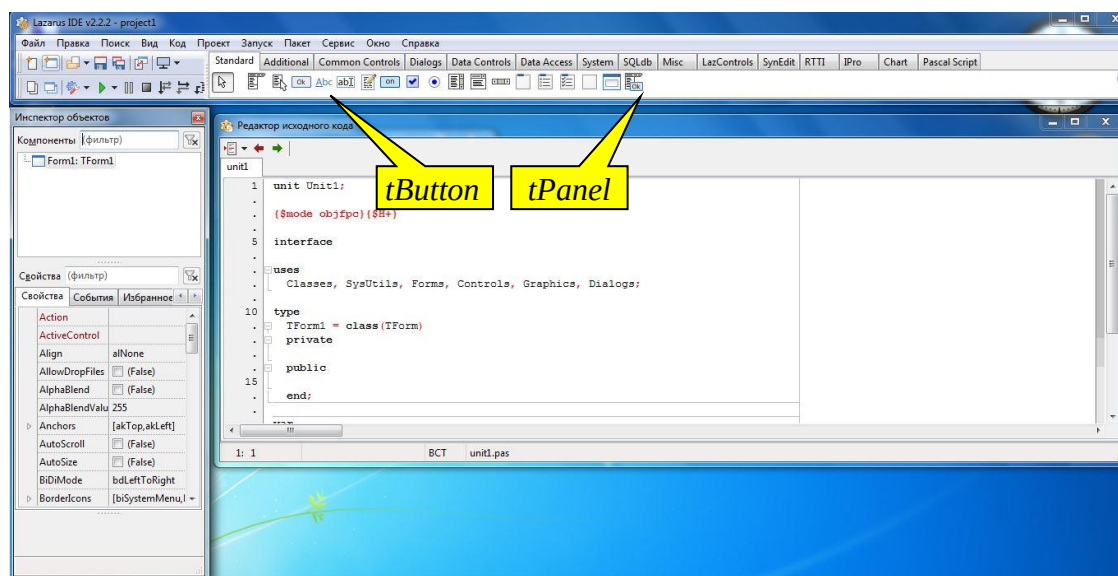


Рис. А.6. Общий вид проекта

- Поместить на форму *Form1* компонент *OpenGLControl1* из библиотеки компонентов. Компонент находится на вкладке *OpenGL* палитры компонентов (рис. А.7) – данная вкладка появляется после выполнения установки компонента, описанной в предыдущем разделе. Для выбора нужного компонента на него нужно кликнуть мышью, а затем для помещения его на форму – кликнуть по форме в выбранном месте. В результате форма будет выглядеть, как на рис. А.8. При этом компонент *OpenGLControl1* будет невидим для пользователя после запуска программы, его можно перекрывать другими визуальными компонентами или уменьшать до минимального размера – это не влияет на работоспособность программы.

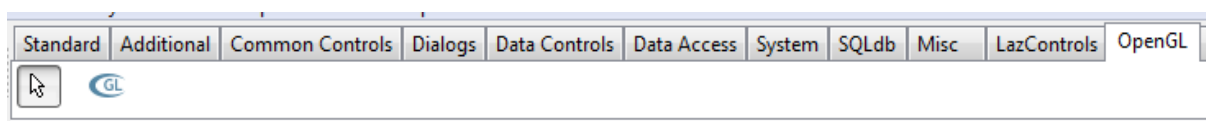


Рис. А.7. Палитра компонентов. Вкладка *OpenGL*

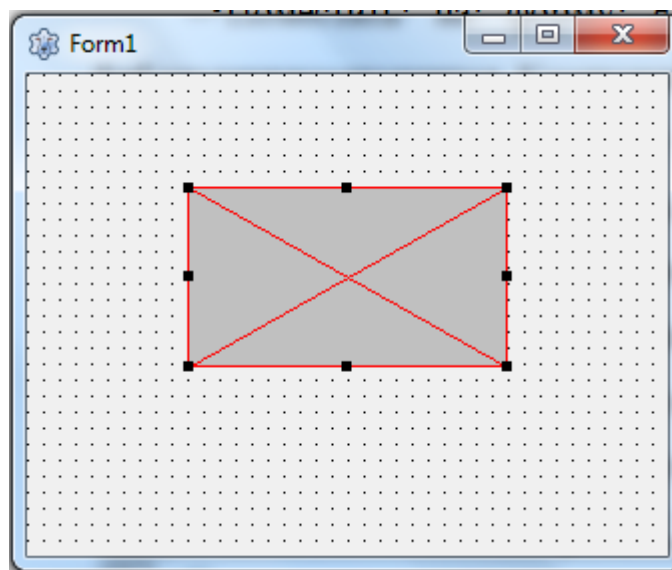


Рис. А.8. Вид главной формы с компонентом *OpenGLControl1*

- Поместить на форму *Form1* два визуальных компонента из библиотеки компонентов: панель *Panel1* и кнопку *Button1*. Компоненты находятся на вкладке *Standard* палитры компонентов (см. выноски на рис. А.6). Для выбора нужного компонента на нем нужно кликнуть мышью, а затем для помещения его на форму – кликнуть по форме в выбранном месте. В результате форма будет выглядеть, как на рис. А.9.

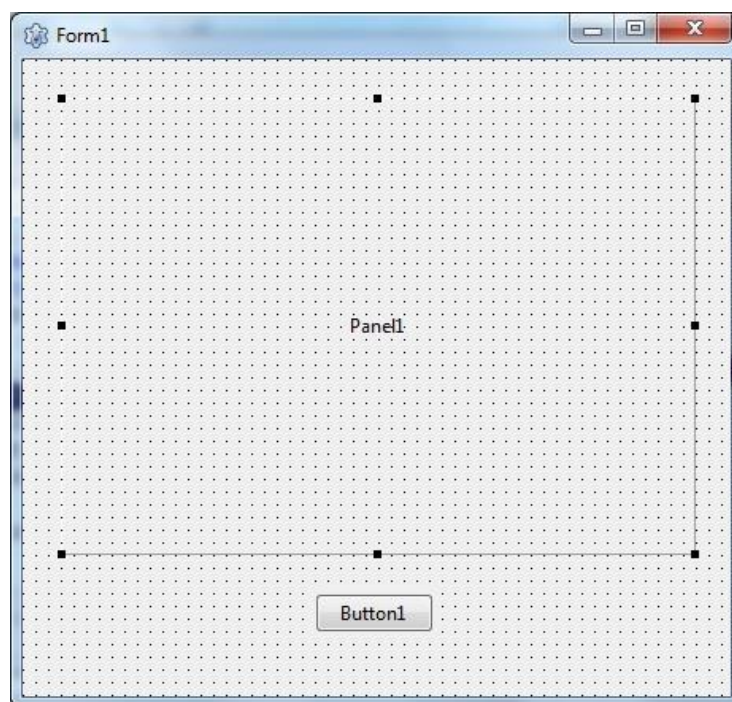


Рис. А.9. Вид главной формы с визуальными компонентами

- Открыть модуль с программным кодом *Unit1*. Для открытия модуля кликнуть значок «Модули» (*Modules*) на панели инструментов или нажать сочетание клавиш <CTRL> + <F12> и в открывшемся окне (рис. А.10) выбрать *unit1.pas*.

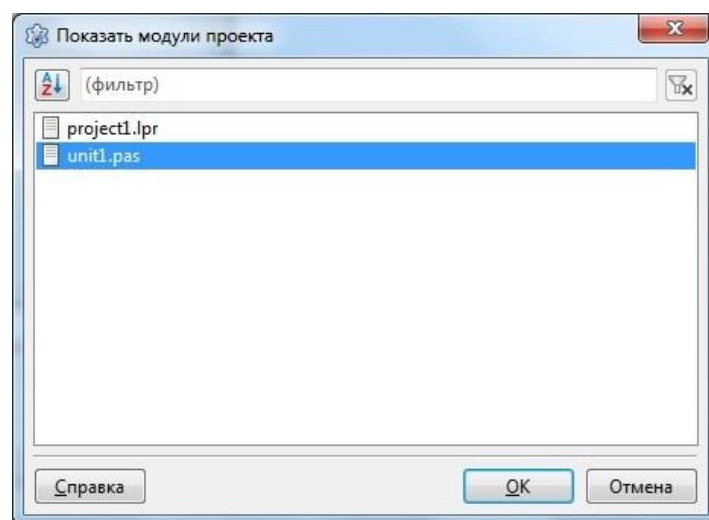


Рис. А.10. Окно выбора модуля

- Проверить подключение библиотеки *OpenGLContext* (объявление добавляется автоматически в конец раздела *uses*) и объявление объекта *TOpenGLControl1* в описании класса *TForm1* (также появляется автоматически после добавления на форму компонента *OpenGLControl1*):

```

TForm1 = class(TForm)
    Button1: TButton;
    OpenGLControl1: TOpenGLControl;
    Panel1: TPanel;

```

- Подключить библиотеку *OpenGL*. Для этого в разделе *uses* дописать имя библиотеки «*GL*».
- Создать обработчик события создания главной формы *FormCreate*. Для этого выбрать форму *Form1*, кликнув по ней левой клавишей мыши, в инспекторе объектов выбрать вкладку «События», на которой выбрать событие *OnCreate* (рис. А.11). Нажав кнопку с многоточием, создать в тексте модуля *Unit1* заготовку процедуры *FormCreate*.

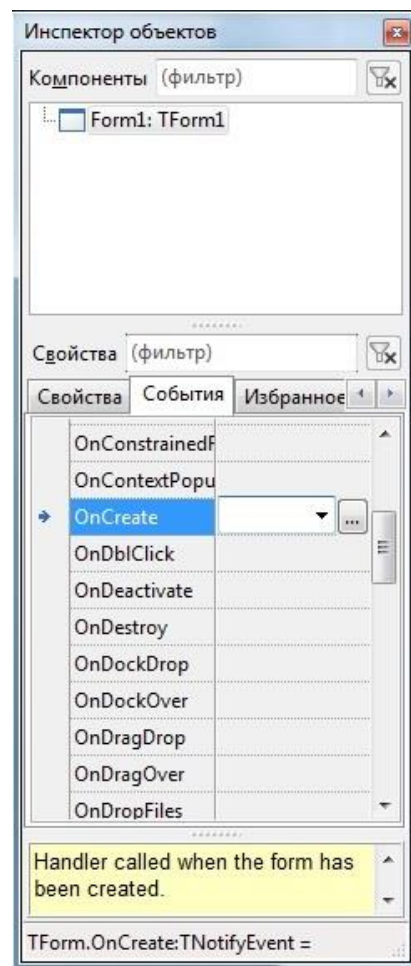


Рис. А.11. Инспектор объектов, событие *OnCreate*

- В процедуре *FormCreate* написать код создания объекта *OpenGLControl1*:

```

OpenGLControl1:= TOpenGLControl.Create(Self);
//Автоматическое масштабирование поля вывода
OpenGLControl1.AutoSizeViewport := true;
//Установка родительского компонента – Panel1
OpenGLControl1.Parent           := Panel1;
//Вывод изображения на всю видимую область компонента
OpenGLControl1.Align            := alClient;.

```

- Создать обработчик события нажатия кнопки *Button1* на главной форме – *Button1Click*. Для этого выбрать кнопку *Button1*, кликнув по ней левой клавишей мыши, в инспекторе объектов выбрать вкладку «События», на которой выбрать событие *OnClick* (рис. А.12). Нажав кнопку с многоточием, создать в тексте модуля *Unit1* заготовку процедуры *Button1Click*.

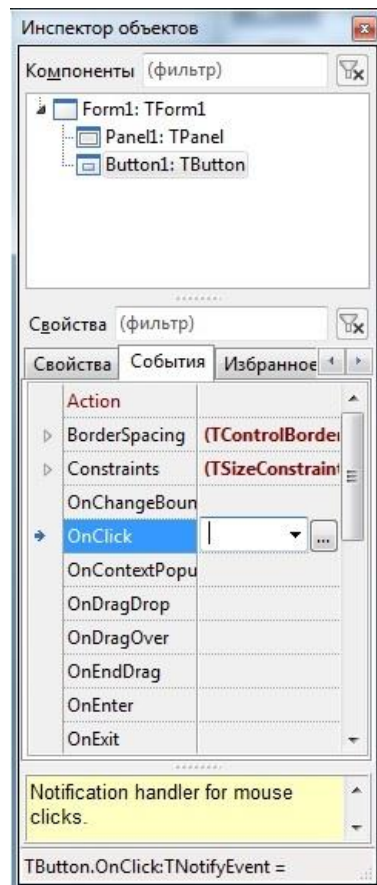


Рис. А.12. Инспектор объектов, событие *OnClick*

- В процедуре *Button1Click* написать код рисования цветного треугольника на панели *Panel1*:

```

glClearColor (0.0, 0.8, 0.8, 1.0); //установка бирюзового
цвета фона
glClear (GL_COLOR_BUFFER_BIT); //очистка буфера цвета - за-
краска фона
glMatrixMode(GL_MODELVIEW); //выбор модельно-видовой
матрицы
glLoadIdentity(); //установка единичной матрицы
glOrtho(-1, 1, -1, 1, -1, 1); //установка объема видимости

glBegin(GL_TRIANGLES); //рисование треугольника
    glColor3f(1, 0, 0); //задание цвета вершины тре-
угольника
    glVertex3f(0.0, 1.0, 0.0); //задание положения вершины
    glColor3f(0, 1, 0);
    glVertex3f(-1.0, -1.0, 0.0);
    glColor3f(0, 0, 1);
    glVertex3f( 1.0, -1.0, 0.0);
glEnd;
OpenGLControl1.SwapBuffers; //вывод буфера кадра на экран

```

- Горизонтальный и вертикальный размеры панели *Panel1* на форме (размеры окна вывода изображения – Width, Height) рекоменду-ется выбрать одинаковыми в инспекторе объектов (вкладка «Свойства»).
- Запустить приложение. Результатом выполнения программы является появление после нажатия кнопки *Button1* изображения, пока-занного на рис. А.13.

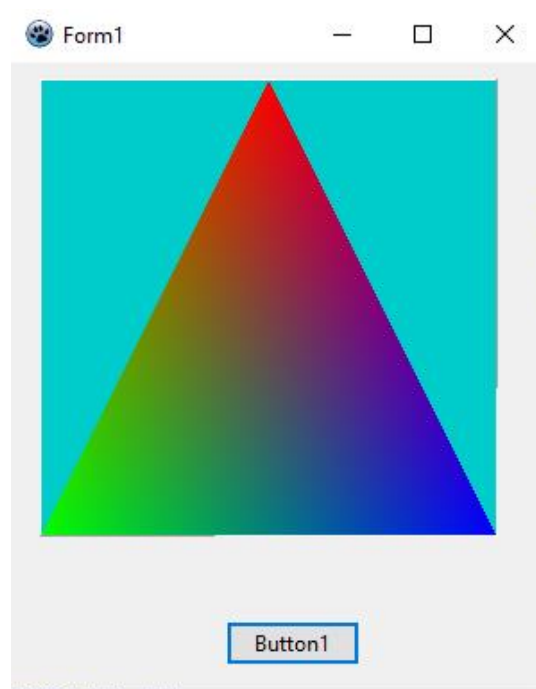


Рис. А.13. Результат работы программы

Полный текст модуля *unit1.pas*, содержащего программный код, приведен ниже:

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
  Dialogs, ExtCtrls, StdCtrls, OpenGLContext, gl;

type
  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    OpenGLControl1: TOpenGLControl;
    Panel1: TPanel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
  public
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenGLControl1:= TOpenGLControl.Create(Self);
  OpenGLControl1.AutoSizeViewport := true;
  OpenGLControl1.Parent           := Panel1;
  OpenGLControl1.Align            := alClient;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
glClearColor (0.0, 0.8, 0.8, 1.0);
glClear (GL_COLOR_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glOrtho(-1, 1, -1, 1, -1, 1);

glBegin(GL_TRIANGLES);
glColor3f(1, 0, 0);
    glVertex3f(0.0, 1.0, 0.0);
    glColor3f(0, 1, 0);
    glVertex3f(-1.0,-1.0, 0.0);
    glColor3f(0, 0, 1);
    glVertex3f( 1.0,-1.0, 0.0);
glEnd;

OpenGLControl1.SwapBuffers;

end;

end.
```

ПРИЛОЖЕНИЕ Б

Контрольные задания по геометрическому моделированию и визуализации двухмерных объектов

Задание 1. Система координат и примитивы OpenGL

Цель выполнения задания: изучить на практике отображение простых геометрических фигур в пределах нормализованного объема видимости.

Изучаемые команды

Команды инициализации: `auxInitWindowPosition`, `auxInitWindowSize`, `glutInitWindowPosition`, `glutInitWindowSize`.

Команды настройки цветов фона и изображения: `glClearColor`, `glClear(GL_COLOR_BUFFER_BIT)`, `glColor`.

Команды рисования геометрических примитивов: `glVertex`, `glBegin` – `glEnd`.

Команды задания стиля вычерчивания: `glLineWidth`, `glEnable(GL_LINE_SMOOTH)`, `glEnable(GL_POINT_SMOOTH)`.

Команда задания стиля (модели) заливки примитивов: `glShadeModel(GL_FLAT)`; .

Варианты заданий приведены в табл. Б2.1.

Таблица Б2.1

Номер варианта	Фигура	Геометрический примитив
1	2	3
1	Треугольник, вписанный в прямоугольник	Треугольник (GL_TRIANGLES), связанные отрезки (GL_LINE_STRIP)
2	Смежные пятиугольник и треугольник	Полигон (GL_POLYGON)
3	«Флажок» – вытянутый прямоугольник и треугольник	Четырехугольник (GL_QUADS), треугольник (GL_TRIANGLES)
4	«Флажок» – вектор и невыпуклый пятиугольник	Отрезок (GL_LINES), полигон (GL_POLYGON)
5	«Кораблик» – трапеция и вектор	Четырехугольник (GL_QUADS), отрезок (GL_LINES)

1	2	3
6	«Молоток» – трапеция и вытянутый прямоугольник	Полигон (GL_POLYGON), четырехугольник (GL_QUADS)
7	«Лопата» – выпуклый пятиугольник и вытянутый прямоугольник	Полигон (GL_POLYGON), отрезок (GL_LINES)
8	«Дупель один-один» – два смежных квадрата с точками в центрах	Четырехугольник (GL_QUADS), точка (GL_POINTS),
9	Пятиугольник с двумя диагоналями	Треугольник (GL_TRIANGLES), отрезок (GL_LINES)
10	«Часы» – квадрат, два узких прямоугольника разной длины и крупная точка в центре	Связанные отрезки (GL_LINE_STRIP), четырехугольник (GL_QUADS), точка (GL_POINTS)
11	«Домик» – смежные треугольник и квадрат	Треугольник (GL_TRIANGLES), четырехугольник (GL_QUADS)
12	Правильный шестиугольник с диагоналями и точкой большого размера в центре	Связанные треугольники – «веер» (GL_TRIANGLE_FAN), точка (GL_POINTS)

Программа выполнения задания:

1. Спроектировать изображение, выбрав координаты вершин, которые ограничивают примитивы. Координаты x и y вершин не должны превышать ± 1 , координата z берется равной нулю. Выбрать размеры окна вывода. Определить цвет фона (не черный) и цвет изображения – разный для каждого фрагмента изображения.

2. В программу-заготовку добавить команды задания окна и рисования изображения. Использовать формат команды задания вершин `glVertex3f`. Получить изображение заданных фигур на экране.

3. Изменяя координаты x и y вершин в диапазоне $(-5, +5)$, определить границы объема видимости по соответствующим координатам, а также направления координатных осей и точку их начала.

4. Задать координату z одной из вершин равной $+0,5$, затем $-0,5$. Объяснить наблюдаемый эффект. Задать координату z одной из вершин равной $+5$, затем -5 . Объяснить наблюдаемый эффект.

Задание 2. Расстановка объектов в сцене

Цель выполнения задания: практическое изучение средств, необходимых для выполнения сценарных преобразований.

Изучаемые команды

Команды задания геометрических преобразований объектов: `glLoadIdentity`, `glTranslate`, `glRotate`, `glScale`.

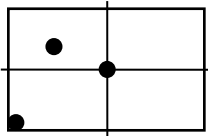
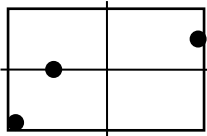
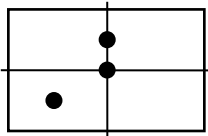
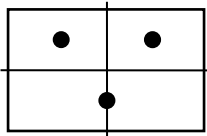
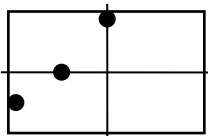
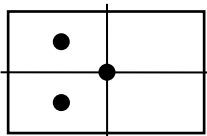
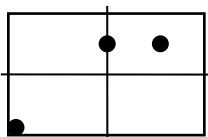
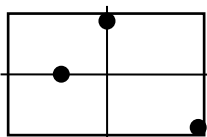
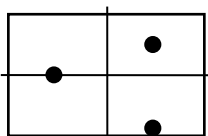
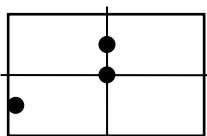
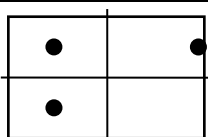
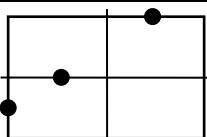
Команды работы с дисплейным списком: `glNewList`, `glEndList`, `glCallList`.

Команды работы со стеком: `glPushMatrix`, `glPopMatrix`.

Команды работы с буфером глубины: `glEnable(GL_DEPTH_TEST)`, `glClear(GL_DEPTH_BUFFER_BIT)`.

Варианты задания приведены в табл. Б2.2. В ней точками показано требуемое расположение трех объектов в поле вывода. В графе «Поворот объектов» заданы углы поворота этих объектов в плоскости экрана. Если объектов больше трех, остальные расставляются по усмотрению студента.

Таблица Б2.2

Номер варианта	Расположение объектов в сцене	Поворот объектов	Номер варианта	Расположение объектов в сцене	Поворот объектов
1		$A1 = 150^\circ$ $A2 = -70^\circ$ $A3 = 45^\circ$	7		$A1 = 160^\circ$ $A2 = -80^\circ$ $A3 = 60^\circ$
2		$A1 = 170^\circ$ $A2 = -90^\circ$ $A3 = 70^\circ$	8		$A1 = 220^\circ$ $A2 = -100^\circ$ $A3 = -80^\circ$
3		$A1 = 230^\circ$ $A2 = -110^\circ$ $A3 = 90^\circ$	9		$A1 = 240^\circ$ $A2 = 120^\circ$ $A3 = -100^\circ$
4		$A1 = 250^\circ$ $A2 = -130^\circ$ $A3 = 110^\circ$	10		$A1 = 0^\circ$ $A2 = -140^\circ$ $A3 = -120^\circ$
5		$A1 = -260^\circ$ $A2 = 150^\circ$ $A3 = -145^\circ$	11		$A1 = 270^\circ$ $A2 = -160^\circ$ $A3 = -10^\circ$
6		$A1 = 280^\circ$ $A2 = 170^\circ$ $A3 = -20^\circ$	12		$A1 = -290^\circ$ $A2 = 20^\circ$ $A3 = -45^\circ$

Программа выполнения задания:

1. Число объектов в сцене – не менее трех. В качестве объектов сцены использовать геометрические фигуры из задания 1, которые для получения требуемого числа объектов рисуются несколько раз. Применить дисплейные списки. Спроектировать сцену, выбрав необходимые команды и их параметры. Разработать последовательность геометрических преобразований, необходимых для поворота на заданный угол:

а) всей сцены;

б) каждого объекта сцены в отдельности.

2. Пояснить различие. Выбрать цвета фона и фигур.

3. В программу-заготовку ввести команды сдвига и поворота системы координат. Чтобы упростить размещение объектов в сцене, использовать стек модельно-видовых матриц. Размеры фигур при необходимости изменить командой масштабирования. Получить изображение сцены. Надлежащим выбором параметров команд сдвига добиться размещения объектов сцены в соответствии с заданием.

4. Задать такие параметры команд `glTranslate`, `glRotate`, чтобы графические объекты сцены частично перекрывались. Проанализировать вид сцены с включенным и выключенным буфером глубины.

Задание 3. Текстурирование объектов

Цель выполнения задания: повышение реалистичности отображения объектов за счет наложения на их поверхность характерного узора – текстуры; изучение средств текстурирования.

Изучаемые команды

Команды подготовки текстур: `glEnable(GL_TEXTURE_2D)`, `glBindTexture`, `glTexImage2D`, `auxDIBImageLoad`.

Команды наложения текстуры: `glTexParameter`, `glTexCoord`.

Задания на текстурирование предусматривают применение в сцене двух текстур: для фоновой заставки и для геометрических объектов. Текстура заставки накладывается на отдельный четырехугольный примитив и имеет тему – лесную, морскую и т.д. Фон, элементы и детальность текстуры определяются студентами самостоятельно, но исходя из темы. Фоновая текстура выполняется инструментами графического редактора и дополняется фамилиями исполнителей. Текстура для объектов сцены имеет геометрический рисунок, выполняемый инструмен-

тами растрового графического редактора. Цветовые решения текстур определяются студентами.

Варианты задания на текстурирование приведены в табл. Б2.3.

Таблица Б2.2

Номер варианта	Тема текстуры для фона	Текстура для объектов
1	Осенняя	Поле с треугольниками
2	Космическая	Поле с ломаной линией
3	Зимняя	Поле с разноцветными эллипсами
4	Техническая	Концентрические пятиугольники
5	Праздничная	Пляшущие человечки
6	Медицинская	Поле с разноцветными трапециями
7	Вулканическая	Знаки зодиака
8	Транспортная	Разноцветные синусоиды
9	Компьютерная	Параллельные полосы разной ширины и цвета
10	Музыкальная	Разноцветные треугольные флажки
11	Библиотечная	Треугольник, вписанный в окружность
12	Военная	Векторы, выходящие из одной точки («пауки»)

Программа выполнения задания:

1. Спроектировать сцену, используя объекты задания 2 и добавив общую заставку. Изготовить текстуры средствами растрового графического редактора. Размеры текстур выбрать кратными степени числа 2. Текстуры сохранить в формате bmp.

2. В программу-заготовку ввести команды подготовки и наложения текстур на объекты сцены. В программах на C-подобных языках использовать команды расширения GLAUX, в программах на Free Pascal (в среде Lazarus) – свободно распространяемый модуль Textures.pas.

Для использования модуля Textures его необходимо присоединить к программе в разделе uses. Загрузка текстуры из файла, расположенного на диске в одной папке с программой, осуществляется с помощью вызова процедуры

LoadTexture ('<имя>.bmp', TexName, False); ,

где <имя> – имя bmp-файла текстуры; TexName (типа GLuint) – имя текстурного объекта.

Нужно обратить внимание на то, что имя и расширение текстурного файла заключаются в апострофы. После загрузки наложение текстуры на графический объект выполняется обычным образом (`glTexCoord`).

3. Для лучшего восприятия текстурированных объектов их цвет перед наложением текстур нужно сделать белым (задать единичные значения R-G-B-компонентам в команде `glColor`).

4. Получить изображение сцены с текстурированными объектами.

5. Изменить расположение текстуры на примитиве, повернув ее на заданный угол.

СПИСОК ЛИТЕРАТУРЫ

1. Эйнджел Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGL : пер. с англ. 2-е изд. М. : Вильямс, 2001. 590 с.
2. Херн Д., Бейкер М.П . Компьютерная графика и стандарт OpenGL : пер. с англ. 3-е изд. М. : Вильямс, 2005. 1168 с.
3. Ву М., Девис Т., Нейдер Дж., Шрайнер Д. OpenGL. Руководство по программированию. Библиотека программиста. 4-е изд. СПб. : Питер, 2006. 624 с.
4. Баяковский Ю. М., Игнатенко А. В. Начальный курс OpenGL. М. : Планета знаний, 2007. 221 с.
5. OpenGL. Программирование с использованием OpenGL. URL: <http://www.opengl.org.ru/>
6. Программирование магических игр. URL: <http://pmg.org.ru/nehe/index.html>
7. OpenGL Tutorial/ru // Lazarus.Freepascal. URL: https://wiki.lazarus.freepascal.org/OpenGL_Tutorial/ru

Учебное издание

**Косников Юрий Николаевич,
Кузьмин Андрей Викторович**

**ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ
И ВИЗУАЛИЗАЦИЯ ДВУХМЕРНЫХ
ОБЪЕКТОВ СРЕДСТВАМИ ОТКРЫТОЙ
ГРАФИЧЕСКОЙ БИБЛИОТЕКИ**

Редактор *А. Г. Темникова*
Технический редактор *Р. Б. Бердникова*
Компьютерная верстка *Р. Б. Бердниковой*

Подписано в печать 19.07.2023. Формат 60×84¹/₁₆.
Усл. печ. л. 3,95.
Заказ № 319. Тираж 43.

Издательство ПГУ.
440026, г. Пенза, ул. Красная, 40.
Тел.: (8412) 66-60-49, 66-67-77; e-mail: iic@pnzgu.ru

Вниманию авторов!

Издательство ПГУ выпускает учебную, научную и художественную литературу, презентационную и акцидентную продукцию, а также полноцветные юбилейные и мемориальные издания в соответствии с ГОСТ 7.60–2003.

Издательство ПГУ принимает к изданию рукописи, подготовленные с использованием текстового редактора Microsoft Word for Windows версий **2003 и выше**. Формат – А4, основной шрифт – Times New Roman, 14–16 pt через одинарный интервал (минимальный размер шрифта в таблицах и сносках – 12,5 pt). Тип файла в электронном виде – doc, docx.

Работа должна содержать индекс УДК, аннотацию.

Аннотация (ГОСТ 7.86–2003, ГОСТ 7.9–1995) включает характеристику основной темы, проблемы объекта, цели работы и ее результаты. В аннотации указывают, что нового несет в себе данный документ в сравнении с другими, родственными по тематике и целевому назначению. Аннотация может включать сведения о достоинствах произведения. Текст аннотации начинают фразой, в которой сформулирована главная тема документа. Заканчивается аннотация читательским адресом.

Рисунки и таблицы должны быть размещены в тексте после ссылки на них (растровые рисунки предоставляются в виде отдельных файлов в формате jpg, BMP с разрешением 300 dpi, векторные рисунки в формате Corel Draw с минимальной толщиной линии 0,75 pt. Рисунки должны быть доступны для правки!). Рисунки должны сопровождаться подрисуночными подписями, на все рисунки и таблицы в тексте должны быть ссылки.

Формулы в тексте выполняются только в редакторе формул **MathType версия 5.0** и выше. Символы греческого и русского алфавита должны быть набраны прямо, нежирно; латинского – курсивом, нежирно; обозначения векторов и матриц – прямо, жирно; цифры – прямо, нежирно. Наименования химических элементов набираются прямо, нежирно. Эти же требования необходимо соблюдать и в рисунках.

В списке литературы **нумерация источников** должна соответствовать очередности ссылок на них в тексте ([1], [2], ...). Номер источника указывается в квадратных скобках. Требования к оформлению списка литературы на русские и иностранные источники (ГОСТ Р 7.0.5–2008): для книг – фамилия и инициалы автора, название, город, издательство, год издания, том, количество страниц; для журнальных статей, сборников трудов – фамилия и инициалы автора, название статьи, полное название журнала или сборника, серия, год, том, номер, страницы; для материалов конференций – фамилия и инициалы автора, название статьи, название конференции, город, издательство, год, страницы.

К материалам **должна** прилагаться следующая информация: фамилия, имя, отчество, контактные телефоны.

Контакты Издательства ПГУ: т.: (8412) 66-60-49, 66-67-77, e-mail: iic@pnzgu.ru

