Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Т. А. Рак, О. О. Шатилова, С. Н. Нестеренков

ДВУМЕРНАЯ ВИЗУАЛИЗАЦИЯ. ОСНОВЫ РАБОТЫ С OpenGL

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для направления специальности 1-40 05 01-12 «Информационные системы и технологии (в игровой индустрии)»

Библиотека БГУИР

Минск БГУИР 2018

УДК 004.92(076)
ББК 32.972.13я73
Р19

Рецензенты:

кафедра экономической информатики учреждения образования
«Белорусский государственный экономический университет»
(протокол №1 от 30.08.2017);

директор ООО «Тотал Геймз» А. А. Фирсов

Рак, Т. А.

Р19 Двумерная визуализация. Основы работы с OpenGL : учеб.-метод. пособие / Т. А. Рак, О. О. Шатилова, С. Н. Нестеренков. – Минск : БГУИР, 2018. – 83 с. : ил.

ISBN 978-985-543-435-2.

Содержит теоретический обзор курса, а также индивидуальные задания по лабораторному практикуму по курсу «Двумерная визуализация».

УДК 004.92(076)
ББК 32.972.13я73

ISBN 978-985-543-435-2

© Рак Т. А., Шатилова О. О., Нестеренков С. Н., 2018
© УО «Белорусский государственный университет информатики и радиоэлектроники», 2018

Библиотека БГУИР

СОДЕРЖАНИЕ

ТЕМА №1. ЗНАКОМСТВО С *OPENGL*. РИСОВАНИЕ ПРОСТЕЙШИХ ГЕОМЕТРИЧЕСКИХ ФИГУР С ПОМОЩЬЮ ГРАФИЧЕСКИХ ПРИМИТИВОВ 4

1.1. Общие теоретические сведения 4
1.2. Индивидуальные задания 14

ТЕМА №2. АЛГОРИТМ ЦДА ДЛЯ ПОСТРОЕНИЯ ЛИНИЙ 16

2.1. Общие теоретические сведения 16
2.2. Индивидуальные задания 26

ТЕМА №3. АЛГОРИТМ БРЕЗЕНХЕМА ДЛЯ ПОСТРОЕНИЯ ПРЯМЫХ 27

3.1. Общие теоретические сведения 27
3.2. Индивидуальные задания 30

ТЕМА №4. АЛГОРИТМЫ ПОСТРОЕНИЯ ОКРУЖНОСТЕЙ 32

4.1. Общие теоретические сведения 32
4.2. Индивидуальные задания 37

ТЕМА №5. ПОСТРОЕНИЕ ЭЛЛИПСА И ФИГУР, ПРОИЗВОДНЫХ ИЗ ЭЛЛИПСА 38

5.1. Краткие теоретические сведения 38
5.2. Индивидуальные задания 48

ТЕМА №6. МЕТОДЫ ОПРЕДЕЛЕНИЯ ВИДИМОСТИ ДЛЯ КАРКАСНЫХ ИЗОБРАЖЕНИЙ. ЗАКРАШИВАНИЕ МНОГОУГОЛЬНИКОВ 49

6.1. Общие теоретические сведения 49
6.2. Индивидуальные задания 61

ТЕМА №7. АТРИБУТЫ ПРИМИТИВОВ 63

7.1. Теоретические сведения 63
7.2. Индивидуальные задания 81

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 82

Библиотека БГУИР

Тема №1. Знакомство с *OpenGL*. Рисование простейших геометрических фигур с помощью графических примитивов

Цель работы: изучить основные функции пакета *OpenGL*; написать и отладить простейшую программу для визуализации примитивов.

1.1. Общие теоретические сведения

В графических пакетах общего назначения пользователям предлагают ряд функций для создания рисунков и выполнения действий над ними.

Эти функции можно классифицировать по тому, имеют ли они дело с графическим выходом, входом, атрибутами, преобразованиями, визуализацией, делением изображений или общим контролем.

Основные блоки, из которых составляются изображения, называются графическими выходными примитивами. К ним относятся символьные строки и геометрические объекты, такие как точки, прямые линии, закрашенные цветные участки (как правило, многоугольники) и формы, которые задаются массивами цветных точек. В некоторых графических пакетах предлагаются функции для изображения более сложных форм – окружностей, цилиндров, конусов. Функции для создания результирующих примитивов представляют основные средства для создания изображений.

Атрибуты – свойства результирующих примитивов. То есть атрибут описывает, как следует изображать отдельный примитив. Сюда относится описание цвета, типа линии, шрифт текста и узоры заполнения отдельных участков.

Графические функции

Размер, положение и ориентацию объекта на сцене можно менять с помощью геометрических преобразований. В некоторых графических пакетах есть дополнительный набор функций для преобразований моделирования, которые используют для построения сцены, где описания отдельных объектов задаются в локальных координатах. В таких пакетах предлагается механизм описания сложных объектов (например, электрического контура или велосипеда) с помощью иерархической структуры (дерева). В других пакетах просто предоставляются функции геометрических преобразований, подробности моделирования предоставляются программисту.

После того как сцена составлена, с помощью функций для описания формы объектов и их атрибутов графический пакет создает проекцию изображения на устройстве вывода. Преобразования наблюдения помогают выбрать точку наблюдения на экране, вид проекции, которую следует использовать в данном случае, и место на мониторе, где будет располагаться изображение. Существуют и другие стандартные функции для управления областью изображения на экране путем задания его координат, размера и структуры. Для 3D-сцен определяются видимые объекты и налагаются соответствующие условия освещения.

Библиотека БГУИР

В интерактивных графических приложениях используются различные типы входных устройств, в том числе мышь, планшет или джойстик. Для проверки обработки потока данных, поступающего от этих интерактивных устройств, используют функции ввода.

Некоторые графические пакеты предлагают также функции для деления описания изображения на именованные наборы составляющих частей.

Наконец, в графических пакетах содержится ряд служебных задач, таких как закрашивание экрана монитора заданным цветом и инициализация параметров. Функции, выполняющие эту работу, можно объединить под названием операции управления.

Стандарты программного обеспечения

Цель, преследуемая в стандартизированных графических программах, – универсальность. Когда разрабатываются пакеты со стандартными графическими функциями, такие программы легко переносятся с одной машины на другую и могут использоваться в различных реализациях и приложениях.

Международные и национальные организации по составлению стандартов во многих странах объединили свои усилия с целью разработки общепринятого стандарта КГ. В результате в 1984 г. была создана базовая графическая система (*Graphical Kernel System – GKS*). Эта система принята в качестве первого стандарта графического программного обеспечения Международной организацией по стандартизации (*International Standards Organization – ISO*).

Вторым разработанным и принятым стандартом программного обеспечения был *PHIGS* (*Programmer`s Hierarchical Interactive Graphics Standard*) – иерархическая интерактивная графическая система программиста. Этот стандарт отличался более широким диапазоном возможностей иерархического моделирования объектов, задания цветов, закрашивания поверхностей и выполнения действий над изображением.

Далее появилось продолжение *PHIGS* – стандарт *PHIGS*+, в котором предоставлялись возможности трехмерного закрашивания поверхностей, которых не было в *PHIGS*.

Во времена разработки *GKS* и *PHIGS* очень популярными стали рабочие станции производства *Silicon Graphics*, *Inc*., *SGI*. Эти рабочие станции выпускались вместе с набором стандартных функций *GL* (*Graphics Library*), который скоро стал довольно популярным в кругах, где дело имелось с КГ. Функции *GL* разрабатывались для быстрого закрашивания в реальном времени, и вскоре этот программный пакет распространился и на другие программные средства. В итоге в начале 1990-х гг. был разработан пакет *OpenGL* как аппаратно-независимая версия пакета *GL*. В августе 2014 года была опубликована спецификация *OpenGL* 4.5. Разработчик – компания *Silicon Graphics* (*SGI*), которая в 1992 г. возглавила *OpenGL ARB* – группу компаний по разработке спецификации *OpenGL*.

Библиотека *OpenGL* разработана специально для эффективной обработки трехмерных данных, но может работать и с описанием 2D-сцен как с частным случаем 3D-изображения.

Библиотека БГУИР

Графические функции в любом пакете задаются как набор описаний, которые не зависят от какого-либо ЯП. Затем указывается привязка к языку для определенного ЯП высокого уровня. Эта привязка задает синтаксис, который позволяет пользоваться различными графическими функциями этого языка. Привязка к ЯП задается так, чтобы максимально использовать соответствующие возможности языка и управлять моментами, связанными с синтаксисом, такими как типы данных, задание параметров и обработка ошибок. Спецификация для реализации графического пакета в определенном языке устанавливается *ISO*.

Знакомство с OpenGL

Основная библиотека функций в пакете *OpenGL* предлагается для спецификации графических примитивов, атрибутов, геометрических преобразований, преобразований наблюдения и многих других преобразований.

OpenGL разрабатывался, чтобы быть независимым от аппаратных средств, поэтому многие операции, такие как функции ввода и вывода, не входят в число функций основной библиотеки. Но стандартные функции ввода-вывода, а также множество дополнительных функций, доступны из вспомогательных библиотек, разработанных для *OpenGL*.

Перед именами функций основной библиотеки *OpenGL* ставят префикс *gl*, а каждое слово, которое входит в имя функции, начинается с заглавной буквы (*glBegin*, *glClear*, *glCopyPixels*, *glPolidonMode*).

Определенные функции требуют, чтобы одному или нескольким из ее аргументов присваивалось значение символьной константы, обозначающей, например, имя параметра, значение параметра или определенного режима. Все такие константы начинаются с заглавных букв *GL*. Кроме того, слова, составляющие имя этой константы, пишутся заглавными буквами, а в качестве разделителя между словами, составляющими одно имя, используют нижнее подчеркивание (*GL_2D*, *GL_RGB*, *GL_CCW*, *GL_POLYGON*, *GL_AMBIENT_AND_DIFFUSE*).

Функции пакета *OpenGL* также воспринимают особые типы данных. Например, параметр функции *OpenGL* может воспринимать значения, которые задаются как 32-разрядные целые числа. Но размер спецификации целого числа на разных машинах может отличаться. Для обозначения особого типа данных в пакете OpenGL используются специальные встроенные названия типов данных: *GLbyte*, *GLshort*, *Glint*, *GLfloat*, *GLdouble*, *GLboolean*.

Родственные библиотеки

Кроме основной библиотеки *OpenGL* существует еще ряд связанных с ней библиотек для выполнения специальных операций. Набор программ *OpenGL* (*OpenGL Utility – GLU*) предоставляет стандартные функции, позволяющие настраивать матрицы проекций и визуализации, описывать сложные объекты через приближения прямых и прямоугольников, изображать квадратичные и бисплайны с помощью линейного приближения, закрашивать поверхности и др.

Библиотека БГУИР

Каждая реализация пакета *OpenGL* включает библиотеку *GLU*, а все имена функций *GLU* начинаются с префикса *glu*.

Чтобы создать графическое изображение с помощью пакета *OpenGL*, прежде всего необходимо открыть на рабочем столе окно изображения – прямоугольную область экрана, на которой будет строиться наше изображение.

Окно изображения нельзя создать непосредственно с помощью основных функций пакета *OpenGL*, т. к. эта библиотека содержит только независящие от прибора графические функции, а операции управления зависят от компьютера, на котором мы работаем. Однако существует несколько библиотек систем окон, которые поддерживают функции для различных машин.

Расширение *OpenGL* для системы *X-Windows* (*GLX*) предлагает набор стандартных функций, которые начинаются с префикса *glX*.

Расширение *OpenGL* для системы *Apple* (*AGL*) предлагает набор стандартных функций, которые начинаются с префикса *agl*.

Для системы *Microsoft Windows* стандартные функции *WGL* предлагает интерфейс *Windows-to-OpenGL*. Эти функции начинаются с префикса *wgl*.

OpenGL Utility Toolkit (*GLUT*) предлагает библиотеку функций для работы с любой системой окон на экране. Библиотеке функций *GLUT* соответствует префикс *glut*. Кроме того, в этой библиотеке содержатся методы, позволяющие описывать и закрашивать кривые и поверхности второго порядка.

Так как *GLUT* – интерфейс для других систем окон со специальными устройствами, мы можем воспользоваться программой *GLUT* и сделать так, чтобы наши программы не зависели от приборов.

Файлы заголовков

Во всех наших графических программах должен быть файл заголовка для корневой библиотеки *OpenGL*. Для большинства приложений нужна еще и библиотека *GLU*. И мы должны включать файл заголовка для системы окон. Например, для системы *Microsoft Windows* файл заголовка для доступа к стандартным функциям *WGL* – *windows.h*. Этот файл заголовка должен находиться перед файлами заголовков для библиотек *OpenGL* и *GLU*, т. к. в нем записан макрос, необходимый для библиотек *OpenGL* версии *Microsoft Windows*.

Исходный файл может начинаться так:

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

Однако, если для выполнения операций управления окном пользоваться библиотекой *GLUT*, то не нужно включать заголовочные файла *gl.h* и *glu.h*, т. к. в библиотеке *GLUT* уже подразумевается, что они будут учтены правильным способом. Поэтому можно заменить файлы заголовков библиотек *OpenGL* и *GLU* на

```
#include <GL/glut.h>
```

Библиотека БГУИР

Управление окнами изображений с помощью библиотеки GLUT

Так как мы будем пользоваться библиотекой *GLUT*, нашим первым шагом будет инициализация *GLUT*. Эта функция инициализации может обрабатывать любые аргументы в командной строке. Инициализация *GLUT* осуществляется с помощью команды

glutInit (&argc, argv);

Затем на экране необходимо создать окно изображения с соответствующим названием в строке заголовка:

glutCreateWindow("Пример рисования линии");

Здесь единственный аргумент – строковая константа, которая содержит название нашего окна изображения.

Далее необходимо задать, что конкретно будет содержаться в нашем окне изображения. Для этого с помощью функции *OpenGL* создается изображение и передается определение *GLUT glutDisplayFunc*, которое связывает наш рисунок с окном изображения.

В качестве простого примера предположим, что у нас есть код *OpenGl* для описания отрезка прямой в процедуре с названием *my_line*. Затем с помощью следующей функции описание линейного сегмента передается в окно изображения:

glutDisplayFunc(my_line);

Но окна изображения все еще нет на экране. Чтобы завершить операции по обработке окна, нам нужна еще одна функция из библиотеки *GLUT*. После выполнения следующего оператора активизируются все окна изображений, которые были созданы вместе со своим графическим содержанием:

glutMainLoop();

Эта функция должна размещаться в крайней точке нашей головной функции. Она служит для изображения начальных графических элементов и вводит программу в бесконечный цикл, в котором проверяются входные данные, поступающие от таких устройств, как мышь и клавиатура. Наш первый пример не будет интерактивным, поэтому программа будет изображать нарисованную нами линию до тех пор, пока мы не закроем окно изображения.

Также мы можем задать размер и местоположение нашего окна изображения, несмотря на то, что эти параметры заданы по умолчанию. С помощью функции *glutInitWindowPosition* из библиотеки *GLUT* мы можем задать исходное положение левого верхнего угла окна изображений. Это положение определяется через целые значения координат экрана. Начало отсчета находится в верхнем левом углу экрана. Например,

glutInitWindowPosition(0, 100);

прижмет наше окно изображения к левому краю экрана и расположит его на 100 пикселей ниже верхней границы экрана.

Аналогично функция *glutInitWindowSize* используется, чтобы задать начальную ширину и высоту окна изображения в пикселях.

После того, как окно изображений появится на экране, мы имеем возможность поменять его начальное местоположение и размер.

Библиотека БГУИР

Также мы имеем возможность задать ряд других опций окна изображений, таких как буферизация и выбор цветового режима, использовав функцию *glutInitDisplayMode*. Аргументами этой функции являются символьные константы библиотеки *GLUT* (табл. 1.1).

Таблица 1.1

Аргументы функции glutInitDisplayMode

Константа	Значение
GLUT_RGB	Для отображения графической информации используются три компоненты цвета *RGB*
GLUT_RGBA	То же что и *RGB*, но используется также четвертая компонента *ALPHA* (прозрачность)
GLUT_INDEX	Цвет задается не с помощью *RGB*-компонентов, а с помощью палитры. Используется для старых дисплеев, где количество цветов, например, 256
GLUT_SINGLE	Вывод в окно осуществляется с использованием одного буфера. Обычно используется для статического вывода информации
GLUT_DOUBLE	Вывод в окно осуществляется с использованием двух буферов. Применяется для анимации, чтобы исключить эффект мерцания
GLUT_ACCUM	Использовать также буфер накопления (*Accumulation Buffer*). Этот буфер применяется для создания специальных эффектов, например, отражения и тени
GLUT_ALPHA	Использовать буфер *ALPHA*. Этот буфер, как уже говорилось, используется для задания четвертого компонента цвета – *ALPHA*. Обычно применяется для таких эффектов, как прозрачность объектов и *anti-aliasing*
GLUT_DEPTH	Создать буфер глубины. Этот буфер используется для отсечения невидимых линий в 3D-пространстве при выводе на плоский экран монитора
GLUT_STENCIL	Буфер трафарета используется для таких эффектов, как вырезание части фигуры, делая этот кусок прозрачным. Например, наложив прямоугольный трафарет на стену дома, вы получите окно, через которое можно увидеть, что находится внутри дома
GLUT_STEREO	Этот флаг используется для создания стереоизображений

Например, с помощью следующей функции задается, что для окна изображения будет использоваться один буфер регенерации, а для выбора цветовых значений – цветовой режим *RGB* (красный, зеленый, синий):

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

Значения постоянных, которые передаются этой функции, объединяются с помощью логической операции ИЛИ. На самом деле эту функцию можно было не прописывать в нашем примере, потому что указанные параметры задаются по умолчанию.

Управление окнами изображений с помощью библиотеки GLUT

Итак, текст нашей головной функции имеет вид

*void main(int argc, char** argv)*

Библиотека БГУИР

```
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowPosition(0, 100);
        glutInitWindowSize(400, 300);
        glutCreateWindow("Пример рисования линии");
        init();
        glutDisplayFunc(my_line);
        glutMainLoop();
}
```

Но у нас еще не все подготовлено и реализовано.

Выбрать цвет фона для окна изображения, а также создать процедуру, в которой содержатся функции, подходящие для изображения линии, которую мы задумали изобразить:

void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);

где *red*, *green*, *blue*, *alpha* – значения, используемые для очистки буфера цвета. Значения по умолчанию 0.

glClearColor устанавливает значение *RGBA* компонент цвета для последующего их использования функцией *glClear*.

Значения должны быть в рамках [0;1]. Если бы мы задали первые три параметра равными 0, то получили бы черный фон.

Четвертый параметр называется альфа-фактором для заданного цвета. Одно из назначений этого параметра – «смешивание». Когда активизируются операции смешивания пакета *OpenGL*, значение альфа-фактора может использоваться для определения результирующего цвета двух перекрывающихся объектов. Значение альфа-фактора 0 говорит о том, что объект будет полностью прозрачным, а значение 1 указывает на полностью непрозрачный объект.

Хотя команда *glClearColor* присваивает цвет окну изображения, она не перемещает его на экран. Чтобы увидеть окно изображений на экране, нужно вызвать следующую функцию *OpenGl*:

glClear(GL_COLOR_BUFFER_BIT);

Эта функция фактически заполняет соответствующие буферы (в нашем случае окно изображения) значением, выбранным в функциях *glClearColor*, *glClearIndex*, *glClearDepth*, *glClearStencil* и *glClearAccum* .

Параметрами функции могут быть:

1) *GL_COLOR_BUFFER_BIT* – очищает текущий буфер цвета, выбранный для записи;
2) *GL_DEPTH_BUFFER_BIT* – очищает буфер глубины;
3) *GL_ACCUM_BUFFER_BIT* – очищает аккумулирующий буфер;
4) *GL_STENCIL_BUFFER_BIT* – очищает буфер трафарета.

Если буфер для очистки не представлен, то *glClear* не вызывает никакого эффекта.

glClear(GL_COLOR_BUFFER_BIT);

Библиотека БГУИР

что означает, что в буфере цвета (буфере регенерации) находятся значения битов, которые следует присвоить переменным, указанным в функции *glClearColor*.

Кроме определения цвета фона для окна изображения можно выбирать различные цветовые схемы для объектов, которые мы хотим изобразить на экране. Для нашей первой программы мы просто зададим цвет объекта (линии) –красный – с помощью функции

glColor3f(1.0, 0.0, 0.0);

Эта функция задания текущего цвета вершины. Параметры соответствуют красному, зеленому и синему цвету. Уровень прозрачности в этой функции 1, что означает, что цвет полностью непрозрачен. *F* указывает на то, что значения всех трех параметров варьируются в пределах от 0 до 1.

Далее нам необходимо сообщить программе *OpenGL*, как мы хотим спроектировать нашу линию на окно изображения, поскольку создание двумерного изображения рассматривается *OpenGL* как частный случай трехмерного. *OpenGL* будет обрабатывать нашу линию с помощью операции полной трехмерной визуализации. Мы можем задать вид проекции (режим) и другие параметры визуализации, которые нам нужны, с помощью следующих двух функций:

glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0, 200.0, 0.0, 150.0);

Для задания различных преобразований объектов сцены в *OpenGL* используются операции над матрицами, при этом различают три типа матриц: видовая, проекций и текстуры. Все они имеют размер 4×4. Видовая матрица определяет преобразования объекта в мировых координатах, таких как параллельный перенос, изменение масштаба и поворот. Матрица проекций задает, как будут проецироваться трехмерные объекты на плоскость экрана (в оконные координаты), а матрица текстуры определяет наложение текстуры на объект.

Для того чтобы выбрать, какую матрицу надо изменить, используется команда

void glMatrixMode(GLenum mode);

вызов которой со значением параметра *mode*, равным *GL_MODELVIEW*, *GL_PROJECTION*, *GL_TEXTURE*, включает режим работы с матрицами – видовой, проекций и текстуры – соответственно. Для вызова команд, задающих матрицы того или иного типа, необходимо сначала установить соответствующий режим.

В *OpenGL* существуют ортографическая (параллельная) и перспективная проекции. Первый тип проекции может быть задан командами:

1) *void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)*;

2) *void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)*.

Первая команда создает матрицу проекции в усеченный объем видимости (параллелограмм видимости) в левосторонней системе координат. Параметры

Библиотека БГУИР

команды задают точки (*left*, *bottom*, *-near*) и (*right*, *top*, *-near*), которые отвечают левому нижнему и правому верхнему углам окна вывода. Параметры *near* и *far* задают расстояние до ближней и дальней плоскостей отсечения по дальности от точки (0, 0, 0) и могут быть отрицательными.

Во второй команде, в отличие от первой, значения *near* и *far* устанавливаются равными –1 и 1 соответственно.

Исходя из вышесказанного, наши команды

glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0, 200.0, 0.0, 150.0);

означают, что для отображения содержимого 2D-прямоугольной области со внешними координатами на экран следует использовать ортогональную проекцию и что значения координаты x этого прямоугольника должны лежать в диапазоне от 0 до 200, а значения y – в диапазоне от 0 до 150. Какие бы объекты мы не задавали в пределах этого прямоугольника внешних координат, они будут попадать в окно изображения. Ничто, находящееся вне описанного диапазона, отображаться не будет.

Таким образом, с помощью функции *gluOrtho2D* задается, что система координат окна изображения будет такой: точка с координатами (0, 0) находится в нижнем левом углу окна изображений, а точка с координатами (200, 150) – в верхнем правом углу. Так как мы описываем 2D-объект, действие ортогональной проекции будет ограничиваться вставкой нашего рисунка в окно изображения.

И, наконец, нам нужно вызвать соответствующие стандартные функции *OpenGL*, чтобы создать линию.

```
glBegin(GL_LINES);
    glVertex2i(0, 0);
    glVertex2i(50, 145);
glEnd();
```

Этот код задает двумерный прямолинейный отрезок с целочисленными декартовыми координатами концов (0, 0) и (50, 145).

Давайте подробнее разберемся, что здесь написано.

Чтобы задать какую-нибудь фигуру, одних координат вершин недостаточно, и эти вершины надо объединить в одно целое, определив необходимые свойства. Для этого в *OpenGL* используется понятие примитивов, к которым относятся точки, линии, связанные или замкнутые линии, треугольники и так далее. Задание примитива происходит внутри командных скобок:

void glBegin(GLenum mode);
void glEnd(void);

Параметр *mode* определяет тип примитива, который задается внутри и может принимать следующие значения:

1) *GL_POINTS* – каждая вершина задает координаты некоторой точки;

2) *GL_LINES* – каждая отдельная пара вершин определяет отрезок; если задано нечетное число вершин, то последняя вершина игнорируется;

3) *GL_LINE_STRIP* – каждая следующая вершина задает отрезок вместе с предыдущей;

Библиотека БГУИР

4) *GL_LINE_LOOP* – отличие от предыдущего примитива только в том, что последний отрезок определяется последней и первой вершиной, образуя замкнутую ломаную;

5) *GL_TRIANGLES* – каждая отдельная тройка вершин определяет треугольник; если задано не кратное трем число вершин, то последние вершины игнорируются;

6) *GL_TRIANGLE_STRIP* – каждая следующая вершина задает треугольник вместе с двумя предыдущими;

7) *GL_TRIANGLE_FAN* – треугольники задаются первой и каждой следующей парой вершин (пары не пересекаются);

8) *GL_QUADS* – каждая отдельная четверка вершин определяет четырехугольник; если задано не кратное четырем число вершин, то последние вершины игнорируются;

9) *GL_QUAD_STRIP* – четырехугольник с номером n определяется вершинами с номерами $2n – 1$, $2n$, $2n + 2$, $2n + 1$;

10) *GL_POLYGON* – последовательно задаются вершины выпуклого многоугольника.

Определение атрибутов вершины

Под вершиной понимается точка в трехмерном пространстве, координаты которой можно задавать следующим образом:

```
void glVertex[2 3 4][s i f d](type coords);
void glVertex[2 3 4][s i f d]v(type *coords);
```

Координаты точки задаются максимум четырьмя значениями: x, y, z, w, при этом можно указывать два (x, y) или три (x, y, z) значения, а для остальных переменных в этих случаях используются значения по умолчанию: $z = 0$, $w = 1$. Как уже было сказано выше, число в названии команды соответствует числу явно задаваемых значений, а последующий символ – их типу.

Координатные оси расположены так, что точка (0, 0) находится в левом нижнем углу экрана, ось x направлена влево, ось y – вверх, а ось z – из экрана. Это расположение осей мировой системы координат, в которой задаются координаты вершин объекта $P – p^*$.

Теперь мы готовы оформить вспомогательные функции – процедуры.

Все функции инициализации и присваивания соответствующих одноразовых параметров поместим в функцию *init*:

```
void init(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
```

Геометрическое описание нашего рисунка поместим в процедуру

```
void my_line(void)
{
```


Библиотека БГУИР

```
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINES);
        glVertex2i(0, 0);
        glVertex2i(50, 145);
        glEnd();
        glFlush();
}
```

В конце этой процедуры находится функция *glFlush*(), которая ускоряет выполнения функций *OpenGL*, записанных в буферах, которые находятся в различных местах вычислительной системы.

Особенности создания приложения с OpenGL в среде MVS2015

Создаем пустой консольный проект.

Заходим в *Свойства Проекта –> Компоновщик –> Ввод –> Дополнительные зависимости* и прописываем следующие команды:

opengl32.lib
glu32.lib
*путь до**Glaux.lib*

В папку с проектом *имя проекта**имя проекта* необходимо добавить файлы *glut.h*, *glut32.lib* и *glut32.dll*.

Непосредственно в исходнике прописать:

```
// WinAPI
#include <Windows.h>
#include <tchar.h>
// OpenGL
#pragma comment(lib, "OpenGL32.lib")
// GLUT
#pragma comment(lib, "glut32.lib")
#include "glut.h"
```

1.2. Индивидуальные задания

Изучив предложенные в теоретическом описании функции, получить окно изображений с заданными размерами и фигурой. Уточненные сведения находятся в табл. 1.2.

Таблица 1.2

Варианты индивидуальных заданий к теме 1

№ варианта	Ширина окна изображений	Высота окна изображений	Фигура
1	**2**	**3**	**4**
1	20	40	Квадрат
2	100	100	Прямая

Библиотека БГУИР

Окончание табл. 1.2

1	**2**	**3**	**4**
3	30	30	Ромб
4	42	50	Квадрат
5	70	100	Ромб
6	55	50	Параллелограмм
7	88	100	Квадрат
8	20	40	Параллелограмм
9	60	60	Трапеция
10	100	50	Квадрат
11	30	30	Прямая
12	40	70	Ромб
13	60	100	Квадрат
14	20	30	Параллелограмм
15	60	50	Трапеция

Библиотека БГУИР

Тема №2. Алгоритм ЦДА для построения линий

Цель работы: изучить алгоритм ЦДА для растрового построения прямых линий; написать и отладить программу визуализации прямой.

2.1. Общие теоретические сведения

Координатные представления

Для создания изображения с помощью программного пакета необходимо задать геометрическое описание объекта, который следует изобразить. Это описание определяет положение и форму объекта. Например, прямоугольник задается через положение его углов (граней), а сфера – через положение центра и радиус. В программных пакетах общего назначения необходимо, чтобы геометрическое описание задавалось в стандартной правосторонней системе координат. Если значения координат рисунка задаются в какой-либо другой системе координат (сферической, гиперболической и т. д.), то, прежде чем вводить их значения в программный пакет, их необходимо преобразовать в декартовы координаты.

В общем случае в процессе создания и изображения сцены используется несколько различных декартовых координатных систем. Во-первых, можно задавать формы отдельных объектов в отдельной системе координат для каждого объекта.

Такие системы координат называют *координатами моделирования*, локальными или главными координатами. Задав формы отдельных объектов, можно составить сцену путем расстановки объектов по соответствующим местам в системе координат сцены, которая называется внешней системой координат. Этот этап подразумевает преобразование отдельных систем координат моделирования в координаты с заданным положением и ориентацией относительно внешней системы координат. Для описания не слишком сложных сцен часть объектов можно добавлять непосредственно в общую структуру сцены во внешних координатах. Геометрическое описание в системе координат моделирования и во внешней системе координат могут задаваться в любой удобной форме как целые числа или как числа с плавающей точкой, без учета ограничений для отдельных устройств ввода.

После того как заданы все элементы сцены, чтобы создать изображение, общее описание во внешних координатах обрабатывают различными программами в одной или нескольких системах координат устройств ввода. Этот процесс называется *конвейером наблюдения* (*viewing pipeline*). Вначале внешние координаты преобразуются в координаты наблюдения, соответствующие тому изображению сцены, которое мы хотим увидеть. В основе этой системы координат лежит положение и ориентация гипотетической камеры. После этого координаты объекта преобразуются в двумерную проекцию сцены, которая соответствует

Библиотека БГУИР

тому, что мы увидим на устройстве вывода. Затем эта сцена записывается в нормированных координатах, где значение каждой координаты попадает в диапазон от –1 до 1 или от 0 до 1, в зависимости от системы. Нормированные координаты еще называют нормированными координатами прибора, т. к. такое описание делает графический пакет независимым от диапазона координат специального устройства вывода.

Еще необходимо определить видимые поверхности и обрезать части рисунка, выходящие за пределы поля зрения. Наконец, стандарты развертки рисунка преобразовываются и попадают в буфер регенерации растровой системы, чтобы превратится в изображение. Систему координат устройства изображения обычно называют координатами устройства, или экранными координатами. Часто и нормированные координаты, и координаты экрана описываются в левосторонней системе координат, так что увеличение положительного расстояния от плоскости *OXY* (экрана или плоскости изображения) можно интерпретировать как удаление от точки наблюдения.

Системы координат

Для того чтобы, к примеру, прорисовать прямолинейный отрезок, необходимо задать положение его двух концов, а для прорисовки многоугольника необходимо задать набор координат его вершин. Значения координат этих точек хранятся в описании сцены вместе с остальной информацией об этих объектах, таких как цвет и координатных границы, т. е. минимальные и максимальные значения координат x, y, z для каждого объекта. Набор координатных границ называют также ограничивающим прямоугольником данного объекта. Затем объекты изображаются – информация о сцене передается стандартным процедурам визуализации, которые определяют видимые поверхности и в конечном итоге ставят в соответствие объектам значения координат на экране. Для записи информации о сцене, такой как коды цвета, в определенные места буфера кадров используется процесс преобразования в стандарт развертки, и на устройстве вывода изображаются сцены.

Экранные координаты

Местоположение на экране выражается через целочисленные экранные координаты, которые соответствуют положениям пикселей в буфере кадра. Значения координат пикселей дают номер строки развертки (значение координаты y) и номер столбца (значение координаты x). При аппаратном выполнении таких процессов, как обновление экрана, как правило, положение пикселей отсчитывается от левого верхнего угла экрана. Тогда строкам развертки присваиваются значения от 0 (верхняя строка экрана) до какого-то целочисленного значения $ymax$ (нижняя строка экрана), а положение пикселей в каждой строке развертки нумеруются от 0 до $xmax$ в направлении слева направо. В то же время с помощью программных команд можно задать любую систему отсчета положений на экране. Например, можно задать диапазон точек с целочисленными координатами и началом координат в нижнем левом углу экрана или воспользоваться для

Библиотека БГУИР

описания рисунка нецелочисленными декартовыми координатами. Затем значения координат, используемые для описания геометрии сцены, с помощью стандартных процедур визуализации преобразуются в целые значения положений пикселей в буфере кадра.

В алгоритмах для строк развертки графические примитивы задаются через координаты представления, т. е. определяются положения пикселей, которые следует изображать. Например, если заданы координаты конечных точек линейного отрезка, алгоритм построения изображения должен вычислить положения тех пикселей, которые лежат на прямой, соединяющей точки. Так как пиксель занимает конечную площадь экрана, это должно учитываться при выполнении алгоритмов. В настоящее время центр области, занимаемой пикселем, принято сопоставлять с каждым целочисленным значением координаты на экране.

Определив положение пикселей для данного объекта, в буфер кадра необходимо записать соответствующие коды цвета. Чтобы сделать это, воспользуемся низкоуровневой процедурой вида

setPixel (x, y, color);
setPixel (x, y);

Эта функция задает цвет пикселя в изображении с координатами (x, y) относительно произвольно выбранного на экране начала отсчета.

Абсолютные и относительные координаты

Рассмотренные выше системы координат формулировались через значения абсолютных координат, т. е. задается действительное положение точек в используемой системе координат.

Но в некоторых графических пакетах положение точек можно также задавать с помощью относительных координат. Такой способ удобен для построения чертежей с помощью перьевых графопостроителей, создания художественных изображений, а также для издательских целей и печатной работы. Воспользовавшись этим способом, можно задавать координаты точки относительно последнего положения, к которому обращалась система (к текущему положению). Например, если точка с координатами (3, 8) – последнее положение, к которому обращалась программа, то относительные координаты (2, –1) соответствуют абсолютным координатам (5, 7). В таком случае, перед тем как задать какие-либо координаты для функции примитива, используется дополнительная функция, устанавливающая текущее положение.

Тогда, чтобы описать такой объект, как набор соединенных между собой прямолинейных отрезков, нужно задать только последовательность относительных координат (смещений) после того, как будет установлено исходное положение. Графические системы могут предлагать опции для задания положения точки с помощью относительных/абсолютных координат. Далее будем считать, что все координаты задаются в абсолютных системах отсчета.

В нашей первой программе мы использовали команду

void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);

Библиотека БГУИР

которая представляет функцию, используемую для задания любой двумерной декартовой системы координат. Аргументы этой функции – значения, определяющие границы изменения координат *x* и *y* для того рисунка, который требуется изобразить. Так как *gluOrtho2D* описывает ортогональную проекцию, необходимо убедиться в том, что значения координат помещены в проекционную матрицу *OpenGL*. Кроме того, перед определением диапазона внешних координат в качестве проекционной матрицы можно использовать единичную матрицу. Это гарантирует, что значения координат не будут прибавляться к значениям, которые, возможно, были ранее внесены в проекционную матрицу. Таким образом, систему координат для использованных ранее примеров можно задать с помощью операторов:

glMatrixMode (GL_PROJECTION); // определяем вид матрицы
glLoadIdentity (); // устанавливаем текущую матрицу единичной
gluOrtho2D (xmin, xmax, ymin, ymax); // устанавливаем размеры плоскости

При этом левый нижний угол на экране будет описываться координатами (*xmin*, *ymin*), а правый верхний угол – координатами (*xmax*, *ymax*).

После этого можно задать один или несколько графических примитивов, при изображении которых используется система координат, описанная в *gluOrtho2D*. Если размеры этих примитивов будут вписываться в координатные пределы окна на экране, то будут изображены все примитивы. В противном случае будут видны только те части примитивов, которые попадают в диапазон координат окна. Кроме того, при задании геометрического описания рисунка все координаты примитивов *OpenGL* должны выражаться в абсолютной системе координат относительно системы отсчета.

Функции точек в OpenGL

Чтобы описать геометрию точки, ее положение задается во внешней системе координат. Затем эти значения вместе с другими геометрическими параметрами, которые могут описывать данную сцену, обрабатываются стандартными процедурами визуализации. Пока не заданы другие значения атрибутов, примитивы *OpenGL* будут изображаться с размерами и цветом, определенными по умолчанию. Изначально цвет примитивов – белый, а размер точки равен размеру одного пикселя на экране.

Чтобы задать координаты единственной точки, используется следующая функция *OpenGL*:

glVertex();*

Знак «*» означает, что для данной функции необходимо указать индексные коды, обозначающие размерность пространства, тип числовых данных, которые используются в качестве значения координат, и возможность представления координат в виде вектора. Функция *glVertex* должна находиться в программе между функциями *glBegin* и *glEnd*. Аргумент *glBegin* определяет тип графического примитива, который следует изобразить, а функция *glEnd* не требует аргументов.

Библиотека БГУИР

При выводе на экран точки в списке аргументов функции *glBegin* появляется символьная константа *GL_POINTS*. Таким образом, положение точки в *OpenGL* описывается так:

```
glBegin (GL_POINTS);
        glVertex* ();
glEnd ();
```

Хотя термином *Vertex* (вершина) в строгом смысле называют «угловую» точку многоугольника, точку пересечения сторон угла, точку пересечения эллипса с его главной осью или другие подобные точки геометрических фигур, функция *glVertex* описывает положение любой точки. Таким образом, для задания точек, прямых линий и многоугольников применяется одна функция, а для описания объектов, составляющих сцену, чаще всего используются прямоугольные участки.

Координаты точек в *OpenGL* могут задаваться в двух, трех или четырех измерениях. Для определения размерности используемого пространства необходимо указать индекс 2, 3 или 4 в функции *glVertex*. Четырехмерное описание указывает на представление с помощью однородных координат, где однородный параметр *h* (четвертая координата) – масштабный коэффициент для декартовых координат.

Далее необходимо обозначить, какой тип данных используется для описания числовых значений координат. Это осуществляется с помощью второго индекса функции *glVertex*: *i* (*integer*), *s* (*short*), *f* (*float*), *d* (*double*). Значения координат в функции *glVertex* могут перечисляться в явном виде, или может использоваться единственный аргумент, который задает положения координат в виде массива. Если применяется описание координат в виде массива, то необходимо также добавить третий индекс – *v* (*vector*).

Допустим, необходимо вывести на экран три точки на одинаковом расстоянии друг от друга на двумерном прямолинейном отрезке с тангенсом угла наклона 2:

```
glBegin ( GL_POINTS );
        glVertex2i ( 50 , 100 );
        glVertex2i ( 75 , 150 );
        glVertex2i ( 100 , 200 );
glEnd ();
```

Или можно сделать все то же самое, но определить координаты в виде массивов и использовать их вместо параметра функции прорисовки:

```
int  point1[] = 50,100;
int  point2[] = 75,150;
int  point3[] = 100,200;
.....
glBegin ( GL_POINTS );
        glVetrex2iv (  point1 );
        glVetrex2iv ( point2 );
        glVetrex2iv ( point1 );
```


Библиотека БГУИР

```
glEnd ();
```

А вот пример задания двух точек в трехмерном пространстве:

```
glBegin ( GL_POINTS );
        glVetrex3f (-78.05, 909.72 , 14.60 );
        glVetrex3f (261.91, -5200.67, 188.33);
glEnd ();
```

В графических пакетах, как правило, предлагаются функции для описания одного или нескольких прямолинейных участков, причем каждый из этих участков определяется координатами двух его концов. В пакете *OpenGL* координаты одного конца выбирают с помощью *glVertex*, точно так же, как это делалось для координат точки, а в среду *glBegin\glEnd* заключается список функций *glVetrex*. Однако теперь в качестве параметра *glBegin* используется символьная константа, которая указывает, что данный перечень координат нужно понимать как координаты концов отрезков. Существуют три символьные константы *OpenGL*, которые можно использовать для определения того, как следует соединять точки данного перечня, чтобы получился набор прямолинейных отрезков. По умолчанию каждая символьная переменная дает изображение сплошных линий белого цвета.

Алгоритмы построения прямых в OpenGL

Прямолинейный отрезок на сцене определяется координатами его концов. Чтобы изобразить прямую на растровом мониторе, графическая система сперва должна спроектировать положения концов отрезка, переводя их в целочисленные значения экранных координат, и определить ближайшие положения пикселей, лежащих вдоль линии, соединяющей эти концы. Затем в буфер кадра загружается цвет линии с соответствующими координатами пикселей. Считывая информацию из буфера кадра, видеоконтроллер изображает пиксели на экране. В ходе этого процесса происходит цифровая обработка прямой линии и преобразование ее в целочисленные значения координат, которые в общем случае только приблизительно передают настоящую форму линии. Рассчитанные координаты прямой (10.48; 20.51) дают координаты пикселя (10; 21). Такое округление значений координат до целых чисел приводит к тому, что все линии, кроме горизонтальных и вертикальных, изображаются в виде «зубцов» (рис. 2.1).

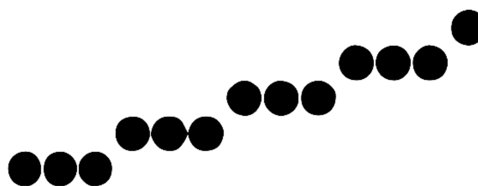


Рис. 2.1. Отображение округленных значений координат

Положение пикселей вдоль прямой линии определяется исходя из геометрических свойств прямой линии. Декартово уравнение прямой линии имеет вид

$$y = a \cdot x + b, \tag{2.1}$$

где a – тангенс угла наклона прямой;

Библиотека БГУИР

b – точка ее пересечения с осью ординат.

Если известно, что два конца отрезка заданы как точки с координатами (x_0, y_0) и (x_{end}, y_{end}), то можно найти значения тангенса угла наклона a и точки b пересечения с осью y по следующим формулам:

$$a = \frac{y_{end} - y_0}{x_{end} - x_0}; \tag{2.2}$$

$$b = y_0 - a \cdot x_0. \tag{2.3}$$

Алгоритмы изображения прямых линий основаны на уравнении прямой и на последних двух формулах.

Для любого заданного интервала координат x (∂x) вдоль линии из уравнения (2.2) можно найти соответствующий интервал координат y (∂y):

$$\partial y = a \cdot x(\partial x). \tag{2.4}$$

Аналогично можно найти интервал оси $x(\partial x)$, соответствующий заданному ∂y:

$$\partial x = \frac{\partial y}{a}. \tag{2.5}$$

Эти уравнения составляют основу для определения отклоняющих напряжений в таких аналоговых дисплеях, как системы векторного сканирования, где возможны относительно небольшие изменения величины отклоняющего напряжения. Для прямых с тангенсом угла $|a| < 1$ ∂x может устанавливаться пропорциональным небольшому горизонтальному отклоняющему напряжению, и тогда соответствующее вертикальное отклонение задается пропорционально ∂y, что можно рассчитать по формуле (2.4). Для прямых, тангенс угла которых больше 1, ∂y можно выбирать пропорционально небольшому вертикальному отклоняющему напряжению, при этом соответствующее горизонтальное отклонение задается пропорционально ∂x, которое рассчитывается по формуле (2.5). Для прямых с тангенсом наклона угла, равным 1, и $\partial y = \partial x$, и напряжения горизонтального и вертикального отклонения равны между собой. В каждом из этих случаев между заданными точками изображается гладкая прямая с тангенсом угла наклона a.

В растровых системах прямые линии строятся по пикселям, и размер шага в горизонтальном и вертикальном направлениях ограничивается разрешением пикселей. Это значит, что нужно «провести выборку» точек прямой линии с дискретными значениями и определить пиксели, самые близкие к данным прямой для каждого элемента выборки. Этот процесс отражен на рисунке, где элементы выборки с дискретными координатами расположены вдоль оси x (рис. 2.2).

Библиотека БГУИР

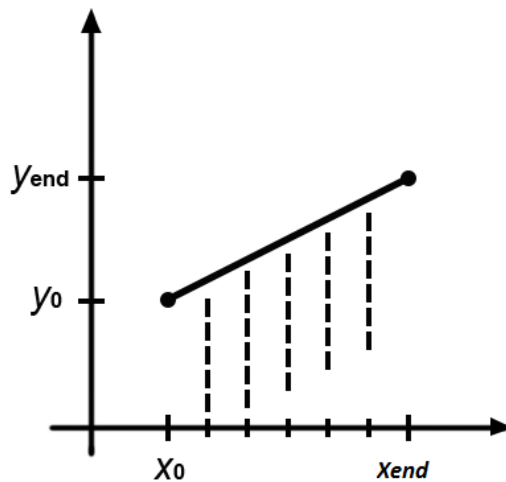


Рис. 2.2. Построение растрового изображения

Цифровой дифференциальный анализатор (ЦДА) – алгоритм преобразования стандартов развертки прямой линии, основанный на вычислении либо ∂x, либо ∂y по уравнениям (2.4) или (2.5). Прямая разбивается на единичные отрезки по одной из координат, а для другой координаты определяются соответствующие целые значения, ближайшие к данной прямой.

Рассмотрим прямую линию с положительным тангенсом угла наклона.

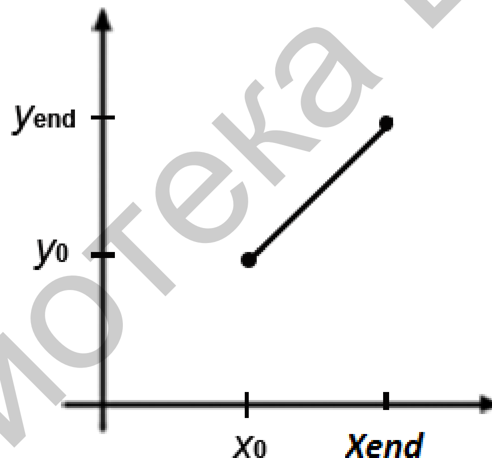


Рис. 2.3. Прямая с положительным углом наклона

Если тангенс угла наклона меньше или равен 1, прямая разбивается на единичные отрезки по координате x ($\partial x = 1$), и последовательно вычисляются значения y:

$$y_{k+1} = y_k + a\,. \tag{2.6}$$

Индекс k пробегает целые числа от 0 (первая точка) и увеличивается на 1 до тех пор, пока не будет достигнута последняя точка. Поскольку a может быть любым действительным числом от 0 до 1, каждое рассчитанное значение следует округлять до ближайшего целого числа, соответствующего положению пикселя на экране в обрабатываемом столбце x.

Библиотека БГУИР

Для прямых с положительным тангенсом угла наклона, превышающим 1, координаты x и y необходимо поменять местами. Прямая разбивается на единичные отрезки по y ($\partial y = 1$) и последовательно вычисляются значения:

$$x_{k+1} = x_k + \frac{1}{a}. \quad (2.7)$$

В основе уравнений (2.6) и (2.7) лежит предположение, что линии обрабатываются в направлении от левого до правого конца. Если обработку выполнять в обратном направлении, т. е. слева направо, то либо $\partial x = -1$ и

$$y_{k+1} = y_k - a, \quad (2.8)$$

либо $\partial y = -1$ и

$$x_{k+1} = x_k - \frac{1}{a}. \quad (2.9)$$

Аналогичные вычисления выполняются с помощью формул (2.6) – (2.9), что позволяет определить положения пикселей на прямой с отрицательным тангенсом угла наклона. Таким образом, если абсолютное значение тангенса угла наклона меньше 1, а начальная точка находится слева, то полагаем, что $\partial x = 1$, и вычисляем значения y с помощью (2.6). Если начальная точка находится справа, тангенс угла наклона положительный и меньше 1, то полагаем, что $\partial x = -1$, и находим положения y с помощью (2.8). Для отрицательного тангенса угла наклона с абсолютным значением больше 1 мы используем $\partial y = -1$ и (2.9) или $\partial y = 1$ и (2.7).

Этот алгоритм сведен к следующей процедуре, входом которой служат два целочисленных значения экранных координат концов отрезка. Параметрам ∂x и ∂y присваиваем значения горизонтальной и вертикальной разностей между точками – концами отрезков. Большую из разностей обозначаем *step*. Начиная с координат (x_0, y_0) определяем смещение, необходимое на каждом шаге для того, чтобы найти следующее положение этой прямой. Этот процесс выполняется *step* раз. Если ∂x больше, чем ∂y, а x_0 меньше, чем x_{end}, то значения приростов по направлениям x и y будут равны 1 и a соответственно. Если же разность по координате x больше, но при этом x_0 больше, чем x_{end}, то для создания следующей точки на прямой используются декременты –1 и $-a$. В любом случае в направлении y используется единичный прирост, а в направлении x – прирост $1/a$.

```
#include <stdlib.h>
#include <cmath>
void init(void){
        glClearColor(0, 0, 0, 0.0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(0.0, 20.0, 0.0, 15.0);
}
void setPixel(int x, int y){
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
```


Библиотека БГУИР

```
    glFlush();
}
inline int round_k(const float a){
    return int(a + 0.5);}
void lineCDA(int x0, int y0, int xend, int yend){
    int dx = xend - x0, dy = yend - y0, step, k;
    float xInc, yInc, x = x0, y = y0;
    step = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);
    xInc = float(dx) / float(step);
    yInc = float(dy) / float(step);
    setPixel(round_k(x), round_k(y));
    for (k = 0; k < step; k++){
        x += xInc;
        y += yInc;
        setPixel(round_k(x), round_k(y));
        }
}
void myDisplay(void){
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 1, 1);
    lineCDA(0, 0, 20, 40);
}
void main(int argc, char** argv){
    glutInit( &argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(0, 100);
    glutInitWindowSize(400, 300);
    glutCreateWindow("Пример ");
    init();
    glutDisplayFunc(myDisplay);
    glutMainLoop();
}
```

Алгоритм ЦДА – более быстрый способ вычисления положений пикселей, чем тот, при котором непосредственно используется уравнения прямой. В нем операция умножения, фигурирующая в уравнении прямой, исключена за счет использования растровых характеристик, так что при перемещении по прямой и переходе от одного пикселя к другому прибавляются нужные приросты по направлениям x и y. Тем не менее, если отрезки достаточно длинные, то из-за накопления ошибок округления при последовательном прибавлении прироста возможно смещение положения пикселя относительно фиксированного направления прямой. Более того, операции округления и арифметика с плавающей точкой требуют больших временных затрат. Выполнение алгоритма ЦДА можно ускорить, разделив приросты a и $1/a$ на целую и дробную части, чтобы все вычисления сводились к операциям с целыми числами.

Библиотека БГУИР

2.2. Индивидуальные задания

Самостоятельно вывести алгоритм ЦДА для построения прямой с координатами (x_0, x_1). Реализовать алгоритм с помощью компьютерных приложений и получить изображение прямых в соответствии с вариантом, предложенным в табл. 2.1.

Таблица 2.1

Варианты индивидуальных заданий к теме 2

№ варианта	Начальная точка отрезка, x_0	Конечная точка отрезка, x_1
1	(0, 9)	(4, 4)
2	(2, 12)	(6, 6)
3	(–5, 8)	(–1, 2)
4	(–1, 10)	(4, 6)
5	(1, 13)	(5, 6)
6	(–4, 9)	(0, 8)
7	(–2, 7)	(2, 2)
8	(4, 14)	(8, 8)
9	(3, 13)	(7, 7)
10	(4, 9)	(0, 3)
11	(–4, 9)	(0, 3)
12	(2, 11)	(6, 6)
13	(0, 17)	(7, 10)
14	(2, 12)	(6, 8)
15	(0, 10)	(4, 4)

Библиотека БГУИР

Тема №3. Алгоритм Брезенхема для построения прямых

Цель работы: изучить алгоритм Брезенхема для растрового построения прямых линий; написать и отладить программу визуализации прямой.

3.1. Общие теоретические сведения

Алгоритм Брезенхема – точный и эффективный растровый алгоритм создания прямых линий, в котором вычисляются только целочисленные значения приростов. Кроме того, алгоритм можно адаптировать для изображения окружностей и других кривых.

Рассмотрим процесс преобразования стандартов развертки для прямой с положительным тангенсом угла наклона меньше 1. В этом случае положение пикселей на прямой определяется разбиением на единичные отрезки по координате x. Начиная с левого конца (x_0, y_0) данной прямой, при переходе к соседнему столбцу наносится пиксель, который по своему номеру строки развертки y является самым близким к направлению прямой (рис. 3.1).

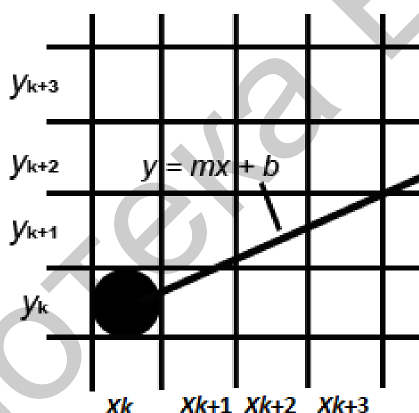


Рис. 3.1. Визуализация прямой

Предположим, что мы определили, что следует наносить пиксель с координатами (x_k, y_k). Далее необходимо решить, какой пиксель изображать в столбце $x_{k+1} = x_k + 1$. Выбор необходимо сделать из пикселей с координатами ($x_k + 1$, y_k), ($x_k + 1$, $y_k + 1$).

В точке выборки с координатами $x_k + 1$ обозначим расстояния по вертикали от пикселей до математической прямой как *d_lower* и *d_upper*. Координата y математической прямой в столбце пикселей $x_k + 1$ находятся как

$$y = a \cdot x_k + 1) + b. \tag{3.1}$$

Тогда

$$d_lower = y - y_k = a \cdot (x_k + 1) + b - y_k, \tag{3.2}$$

$$d_upper = (y_k + 1) - y - = y_k + 1 - a \cdot (x_k + 1) - b. \tag{3.3}$$

Библиотека БГУИР

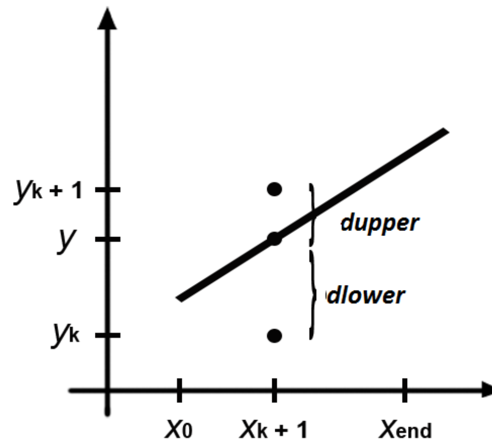


Рис. 3.2. Графическая интерпретация *d_lower* и *d_upper*

$$p_k = \partial x \cdot \left(d_lower - d_upper \right) = 2 \cdot \partial y \cdot x_k - 2 \cdot \partial x \cdot y_k + c.$$

Чтобы определить, какой из этих пикселей ближе к заданной прямой, можно провести эффективную проверку, основанную на разности двух расстояний до пикселей:

$$d_lower - d_upper = 2 \cdot a \cdot (x_k + 1) - 2 \cdot y_k + 2 \cdot b - 1. \quad (3.4)$$

Параметр p_k – параметр принятия решения для *k*-го шага алгоритма построения прямой линии находится путем такого преобразования уравнения (3.4), чтобы в него входили только целочисленные расчеты. Это можно сделать, проведя подмену $a = \partial y / \partial x$, где ∂ – вертикальные и горизонтальные расстояния между концами отрезка, и вычислить параметр принятия решения как

$$p_k = \partial x \cdot (d_lower - d_upper) = 2 \cdot \partial y \cdot x_k - 2 \cdot \partial x \cdot y_k + c. \quad (3.5)$$

Знак параметра p_k будет таким же, как и знак *d_lower* – *d_upper*. Параметр c – постоянная со значением $2 \cdot \partial y + \partial x \cdot (2 \cdot b - 1)$, которая не зависит от координаты пикселя и находится в ходе рекурсивных вычислений, необходимых для расчета p_k. Если пиксель с координатой y_k окажется ближе к реальному направлению прямой, чем пиксель с координатой y_{k+1} (т. е. *d_lower* < *d_upper*), то параметр принятия решений окажется отрицательным, и на график наносится нижний пиксель. Если p_k оказывается положительным, то на график наносится верхний пиксель.

Изменение значения координаты вдоль направления прямой происходит при единичном шаге как в направлении x, так и в направлении y. Следовательно, с помощью схемы целочисленного прироста можно найти значения последующих параметров принятия решения. На шаге k+1 параметр находится из уравнения (3.5) как

$$p_{k+1} = 2 \cdot y \cdot \left(x_k + 1 \right) - 2 \cdot \partial x \cdot \left(y_k + 1 \right) + c.$$

Вычитая (3.5) из предыдущего уравнения, получим

$$p_{k+1} - p_k = 2 \cdot \partial y \cdot \left(x_{k+1} - x_k \right) - 2 \cdot \partial x \cdot \left(y_{k+1} - y_k \right)$$

или

Библиотека БГУИР

$$p_{k+1} = p_k + 2 \cdot \partial y - 2 \cdot \partial x \cdot (y_{k+1} - y_k), \tag{3.6}$$

где $y_{k+1} - y_k$ равен или 0, или 1 в зависимости от знака параметра p_k.

Такой рекурсивный расчет параметров принятия решения выполняется в каждой точке с целым значением координаты x, начиная с координаты левого конца отрезка. Первый параметр p_0 находится из уравнения (3.5) в начальной точке с координатами (x_0, y_0), при этом a находится как $\partial y/\partial x$:

$$p_0 = 2 \cdot \partial y - \partial x. \tag{3.7}$$

Итак, при выполнении преобразования стандартов развертки постоянные $2 \cdot \partial$у и $2 \cdot \partial$у $- 2 \cdot \partial x$ рассчитываются один раз для прямой.

Алгоритм Брезенхема:

1) вводим два конца отрезка, помечая левый конец отрезка как (x_0, y_0);
2) задаем в буфере кадра цвет пикселя (x_0, y_0);
3) вычисляем постоянные ∂x, ∂y, $2 \cdot \partial y$ и $2 \cdot \partial y$, $2 \cdot \partial x$ и находим начальное значение параметра принятия решения $p_0 = 2 \cdot \partial y - \partial x$;
4) для каждого x_k вдоль прямой, начиная с $k = 1$, проводим проверки: если $p_k < 0$, то следующую точку следует изобразить на месте пикселя (x_{k+1}, y_k) и $p_{k+1} = p_k + 2 \cdot \partial y$; если $p_k > 0$, то следующую точку следует изобразить на месте пикселя (x_{k+1}, y_{k+1}) и $p_{k+1} = p_k + 2 \cdot \partial y - 2 \cdot \partial x$;
5) выполняем этап ∂x 1 раз.

```
void linesBrasenhem(int x0, int y0, int xend, int yend){
        int dx = abs(xend – x0), dy = abs(yend – y0);
        int p = 2 * dy – dx;
        int x, y;
        if (x0 > xend){
                x = xend;
                y = yend;
                xend = x0;
                }
        else
                {
                x = x0;
                y = y0;
                }
        setPixel(x, y);
        }
        while (x < xend){
                x++;
                if (p < 0)
                        p += 2 * dy;
                else
```

Библиотека БГУИР

```
            {
            y++;
            p += 2 * (dy – dx);
            }
        setPixel(x, y);
    }
```

Алгоритм Брезенхема можно обобщить для прямых линий с произвольным тангенсом угла наклона, рассмотрев симметрию различных октантов и квадрантов плоскости xy. Для прямой с положительным тангенсом угла наклона, превышающим 1, роли x и y меняются местами. Это означает, что в направлении координаты y делаются единичные шаги, и при этом последовательно вычисляются координаты x, ближайшие к направлению прямой. Кроме того, программу можно изменить таким образом, чтобы построение прямой начиналось с другого конца. Если в качестве исходной точки прямой с положительным тангенсом угла наклона берется правый конец отрезка, то x и y уменьшаются при каждом шаге слева направо. Чтобы гарантировать, что независимо от начальной точки будут изображаться одни и те же пиксели, когда вертикальные расстояния от пикселей до прямой равны ($d_lower = d_upper$), всегда выбирается верхний (или нижний) пиксель.

При отрицательном тангенсе угла наклона алгоритм аналогичный, только на этот раз одна координата увеличивается, а вторая, наоборот, уменьшается.

Наконец, можно отдельно рассматривать некоторые частные случаи: горизонтальные ($\partial y = 0$), вертикальные ($\partial x = 0$), и диагональные прямые ($|\partial y| = |\partial x|$) можно непосредственно заносить в буфер кадров без обработки с помощью алгоритма построения простых линий.

3.2. Индивидуальные задания

Самостоятельно вывести алгоритм Брезенхема для построения прямой с координатами (x_0, x_1). Реализовать алгоритм с помощью компьютерных приложений и получить изображение прямых в соответствии с вариантом, предложенным в табл. 3.1.

Таблица 3.1

Варианты индивидуальных заданий к теме 3

№ варианта	Начальная точка отрезка, x_0	Конечная точка отрезка, x_1
1	**2**	**3**
1	(0, 9)	(4, 4)
2	(2, 12)	(6, 6)
3	(–5, 8)	(–1, 2)
4	(–1, 10)	(4, 6)
5	(1, 13)	(5, 6)

Библиотека БГУИР

Окончание табл. 3.1

1	2	3
6	(–4, 9)	(0, 8)
7	(–2, 7)	(2, 2)
8	(4, 14)	(8, 8)
9	(3, 13)	(7, 7)
10	(4, 9)	(0, 3)
11	(–4, 9)	(0, 3)
12	(2, 11)	(6, 6)
13	(0, 17)	(7, 10)
14	(2, 12)	(6, 8)
15	(0, 10)	(4, 4)

Библиотека БГУИР

Тема №4. Алгоритмы построения окружностей

Цель работы: изучить растровые алгоритмы построения окружностей; написать и отладить программу визуализации окружности.

4.1. Общие теоретические сведения

В корневую библиотеку *OpenGL* не включены в качестве функций-примитивов функции создания таких кривых основных типов, как окружности и эллипсы. Однако в этой библиотеке есть функции для изображения сплайнов Безье, представляющих собой полиномы, которые задаются набором дискретных точек.

В библиотеке *OpenGL Utility* (*GLU*) есть процедуры для создания трехмерных поверхностей второго порядка, таких как сферы и цилиндры, а также функции для построение рациональных би-сплайнов, т. е. сплайнов, которые включают более простые кривые Безье. С помощью рациональных би-сплайнов можно изображать окружности, эллипсы и другие двухмерные поверхности второго порядка. Кроме того, в пакете *OpenGL Utulity Toolkit* (*GLUT*) есть процедуры, с помощью которых можно изображать такие трехмерные поверхности второго порядка, как сферы и конусы, а также некоторые другие фигуры.

Еще один способ создания изображения простой кривой – аппроксимировать ее с помощью ломаной линии. Нужно всего лишь задать набор точек, расположенных на этой кривой, а затем соединить их. Причем чем больше будет этих точек, тем более гладким будет изображение этой кривой.

Третий способ – написать собственную функцию для построения кривой, основанную на алгоритмах, которые будут нами рассмотрены в дальнейшем.

Поскольку окружность – элемент, который часто используется в рисунках и графиках, во многие графические пакеты включена процедура изображения или полных окружностей, или различных дуг. Кроме того, иногда в графической библиотеке содержатся функции изображения различных типов кривых, включая окружности и эллипсы.

Окружность определяется как геометрическое место точек, удаленных на заданное расстояние r от центра с координатами (x_c, y_c) (рис. 4.1). Для любой точки окружности взаимосвязь с этим расстоянием в декартовых координатах выражается через теорему Пифагора:

$$(x - x_c)^2 + (y - y_c)^2 = r^2. \quad (4.1)$$

Этим уравнением можно воспользоваться для того, чтобы определить положения точек на окружности, перемещаясь по оси x с единичным шагом от $x_c - r$ до $x_c + r$, вычисляя в каждой точке соответствующее значения y:

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}\,. \quad (4.2)$$

Библиотека БГУИР

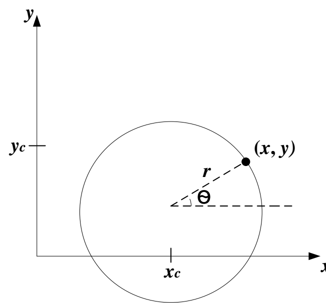


Рис. 4.1. Окружность

Можно регулировать величину этого промежутка, меняя местами *x* и *y* (делая шаг по оси *y* и вычисляя соответствующий *x*), когда абсолютное значение тангенса угла наклона окружности больше 1. Но это увеличит объем операций и время на обработку.

Еще один способ устранения неодинаковых промежутков – найти точки на границе круга с помощью полярных координат *r* и Θ, в которых уравнение окружности имеет вид

$$x = x_c + r \cdot \cos\Theta,$$
$$y = y_c + r \cdot \sin\Theta. \quad (4.3)$$

Если изображение создается с помощью этих уравнений при фиксированном шаге по углу, окружность строится из точек, расположенных по кругу через одинаковые интервалы. Чтобы уменьшить объем вычислений, можно сделать угловое расстояние между точками на окружности большим и соединять эти точки прямолинейными отрезками, аппроксимируя ими окружность. Для того чтобы на растровом экране получить более гладкую линию, размер углового шага можно задать равным $1/r$. При этом пиксели будут располагаться приблизительно на одинаковом расстоянии друг от друга. Из минусов такого подхода – необходимость использовать трудозатратные тригонометрические функции.

При любом из описанных способов прорисовки окружностей можно уменьшить количество вычислений, учитывая симметрию окружности. Если определить положение кривой в первом квадранте, можно построить часть окружности во втором сегменте, отразив кривую симметрично относительно оси *y*. И так далее по аналогии. Этим путем можно пройти еще дальше и воспользоваться симметрией октантов.

На рис. 4.2 видно, что, зная координаты точки (*x*, *y*) на одной восьмой части окружности, можно изобразить еще семь точек этой окружности.

Зная это, можно прорисовать лишь точки на секторе от $x = 0$ до $x = y$.

Но определение координат пикселей окружности с использованием симметрии и уравнений (4.1) и (4.3) требует существенного объема вычислений.

Более эффективные алгоритмы построения окружностей основаны на вычислении прироста параметра принятия решения, как это было в алгоритме Брезенхема, куда входят только простые операции с целыми числами.

Библиотека БГУИР

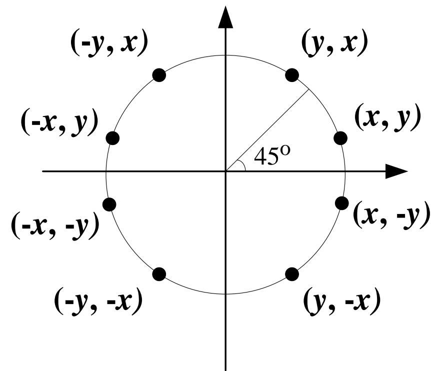


Рис. 4.2. Симметрия октантов окружности

Алгоритм Брезенхема для построения прямой линии можно адаптировать для построения окружностей, устранив на каждом шаге выборки параметры принятия решения для определения ближайшего к окружности пикселя.

Здесь мы будем разбивать на единичные отрезки аппроксимируемую линию и на каждом шаге будем определять координаты пикселей, которые находятся ближе всего к заданной окружности. При известном радиусе r и координатах центра окружности на экране (x_c, y_c) вначале можно организовать алгоритм таким образом, чтобы рассчитать координаты пикселей вокруг окружности с центром в (0, 0). После этого каждое рассчитанное положение (x, y) перемещается в соответствующую ему точку на экране, для чего x_c прибавляется к x, y_c – к y. На дуге окружности от $x = 0$ до $x = y$ в первом квадранте тангенс угла наклона кривой меняется от 0 до – 1. Таким образом, в этом октанте можно брать единичный шаг в положительном направлении координаты x и на основе параметра принятия решения определять, какой из пикселей в каждом столбце находится ближе по вертикали к заданной окружности. После этого, используя симметрию, находим координаты пикселей окружности в остальных семи октантах.

Чтобы воспользоваться методом средней точки, зададим функцию окружности как

$$f_{circl}(x, y) = x^2 + y^2 - r^2. \tag{4.4}$$

Любая точка (x, y), которая лежит на окружности радиусом r, удовлетворяет уравнению $f_{circl}(x, y) = 0$. Если точка находится внутри круга, значение функции будет положительным:

$$f_{circl}(x, y)\begin{cases} < 0, & \text{если точка лежит внутри окружности;} \\ = 0, & \text{если точка лежит на окружности;} \\ > 0, & \text{если точка лежит вне окружности.} \end{cases} \tag{4.5}$$

Проверка (4.5) выполняется на каждом шаге выборки для средних положений между пикселями вблизи заданной окружности. Таким образом, функция окружности – это параметр принятия решения в алгоритме средней точки, и для этой функции можно установить операции приращения, как это было сделано для алгоритма построения прямой линии.

Библиотека БГУИР

На рис. 4.3 показана средняя точка между двумя возможными пикселями в точке выборки x_{k+1}.

Рис. 4.3. Визуализация метода средних точек

Допустим, только что поставлена точка в пикселе с координатами (x_k, y_k). Теперь определим, какой из двух пикселей ближе к заданной окружности – пиксель с координатами (x_{k+1}, y_k) или пиксель с координатами (x_{k+1} , $y_k - 1$). Параметром принятия решения будет функция окружности (4.4), которая рассчитывается для средней точки между этими двумя пикселями:

$$p_k = f_{circl}(x_k + 1, y_k - \frac{1}{2}) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2. \qquad (4.6)$$

Если $p_k < 0$, то данная точка лежит внутри окружности, и к заданной окружности будет ближе пиксель, который находится в строке развертки y_k. В противном случае средняя точка находится за пределами окружности или лежит на ней, и выбирается пиксель в строке развертки y_{k-1}.

Последующие параметры принятия решения находятся с помощью операции приращения. Рекурсивное выражение для следующего параметра принятия решения получается вычислением функции окружности в точке выборки x_{k+2}:

$$p_k = f_{circl}(x_k + 1, y_k - \frac{1}{2}) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2$$

или

$$p_{k+1} = p_k + 2(x_k + 1) + (y^2{}_{k+1} - y^2{}_k)^2 - (y_{k+1} - y_k) + 1, \qquad (4.7)$$

где y_{k-1} – это либо y_k, либо y_{k-1}, в зависимости от знака p_k.

Прирост, с помощью которого находится p_{k+1}, равен либо $2 \cdot x_{k+1} + 1$ (если p_k отрицательное), либо $2 \cdot x_{k+1}+1 - 2 \cdot y_{k+1}$. Члены $2 \cdot x_{k+1}$ и $2 \cdot y_{k+1}$ также можно оценить через прирост значения как

$$2x_{k+1} = 2x_k + 2,$$

$$2y_{k+1} = 2y_k - 2.$$

В начальном положении (0, r) эти два параметра имеют значения 0 и $2 \cdot r$ соответственно. Каждое последующее значение $2x_{k+1}$ находится прибавлением двух к предыдущему значению, а каждое последующее значение $2y_{k+1}$ получается вычитанием двух из предыдущего значения.

Чтобы найти начальный параметр принятия решения, вычисляется функция окружности в начальном положении $(x_0, y_0) = (0, r)$:

$$p_0 = f_{circl}(1, r - \frac{1}{2}) = 1 + (r - \frac{1}{2})^2 - r^2 = \frac{5}{4} - r. \quad (4.8)$$

Если радиус r – целое число, то все приросты – целые числа, p_0 можно просто округлить до

$$p_0 = 1 - r.$$

Алгоритм:

1. Вводим радиус r и координаты центра окружности (x_c, y_c), затем задать координаты первой точки на окружности, центр которой находится в начале координат $(x_0, y_0) = (0, r)$.

2. Определяем исходное значение параметра принятия решения:

$$p_0 = 5/4 - r.$$

3. Для каждого значения x_k, начиная с $k = 0$, выполнить следующую проверку:

если $p_k < 0$, то следующая точка на окружности с центром в точке (0, 0) будет (x_{k+1}, y_k) и $p_{k+1} = p_k + 2 \cdot x_{k+1} + 1$;

иначе следующей точкой на окружности будет точка (x_{k+1}, y_{k-1}) и $p_{k+1} = p_k + 2 \cdot x_{k+1} + 1 - 2 \cdot y_{k+1}$, где $2\,x_{k+1} = 2 + 2 \cdot x_k$ и $2 \cdot y_{k+1} = 2 \cdot y_k - 2$.

4. Находим симметричные точки в остальных семи октантах.

5. Перемещаем все рассчитанные пиксели с координатами (x, y) на окружность с центром в точке с координатами (x_c, y_c) и отмечаем точки с координатами $x = x + x_c$, $y = y + y_c$.

6. Повторяем этапы 3–5 до тех пор, пока не получится, что $x \geq y$.

```
void circlePlotPoints (int xc, int yc, int x, int y){
        setPixel(xc + x, yc + y);
        setPixel(xc + x, yc - y);
        setPixel(xc - x, yc + y);
        setPixel(xc - x, yc - y);
        setPixel(xc + y, yc + x);
        setPixel(xc + y, yc - x);
        setPixel(xc - y, yc + x);
        setPixel(xc - y, yc - x);
}
void circleMiddlePoint(int xc, int yc, int r){
        int p = 1 - r;
        int x = 0, y = r;
        while (x <= y)
                {
                circlePlotPoints(xc, yc, x, y);
                x++;
                if (p < 0)
                        p += 2 * x + 1;
                else
```

Библиотека БГУИР

```
                {
                y--;
                p += 2 * (x - y) + 1;
                }
        }
}
```

4.2. Индивидуальные задания

Самостоятельно вывести алгоритм средней точки для построения окружности с координатами центра (x_0, x_1) и радиусом r. Реализовать алгоритм с помощью компьютерных приложений и получить изображение окружности в соответствии с вариантом, предложенным в табл. 4.1.

Таблица 4.1

Варианты индивидуальных заданий к теме 4

№ варианта	Координаты центра, (x_0, x_1)	Радиус, r
1	(0, 9)	13
2	(2, 12)	10
3	(–5, 8)	8
4	(–1, 10)	7
5	(1, 13)	10
6	(–4, 9)	8
7	(–2, 7)	5
8	(4, 14)	13
9	(3, 13)	12
10	(4, 9)	8
11	(–4, 9)	8
12	(2, 11)	10
13	(0, 17)	15
14	(2, 12)	11
15	(0, 10)	9

Библиотека БГУИР

Тема №5. Построение эллипса и фигур, производных из эллипса

Цель работы: изучить растровые алгоритмы построения эллипсов и кривых; написать и отладить программу визуализации эллипса.

5.1. Краткие теоретические сведения

Эллипс – эта вытянутая окружность. Так что его можно описать как модифицированную окружность, радиус которой меняется от максимального значения в одном направлении до минимального значения в перпендикулярном направлении. Прямолинейные отрезки, проведенные внутри эллипса по этим двум перпендикулярным направлениям, называются большой и малой осями эллипса.

Точное определение эллипса можно дать, воспользовавшись расстояниями от любой точки эллипса до двух фиксированных точек, которые называются его фокусами. Сумма этих двух расстояний одинакова для всех точек эллипса. Если расстояние от двух фокусов до любой точки эллипса $P = (x, y)$ обозначить d_1 и d_2, тогда общее уравнение эллипса можно записать как

$$d_1 + d_2 = \text{const}. \quad (5.1)$$

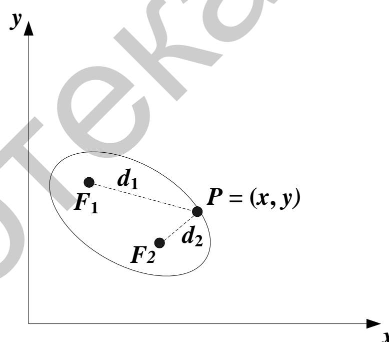


Рис. 5.1. Эллипс

Если выразить расстояния d_1 и d_2 через координаты фокусов $F_1 = (x_1, y_1)$ и $F_2 = (x_2, y_2)$, то получится

$$\sqrt{(x - x1)^2 + (y - y1)^2} + \sqrt{(x - x2)^2 + (y - y2)^2} = \text{const}. \quad (5.2)$$

Возведя это уравнение в квадрат и собрав члены с одинаковыми степенями и снова возведя в квадрат, можно переписать общее уравнение эллипса в виде

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0, \quad (5.3)$$

где коэффициенты A, B, C, D, E и F выражаются через координаты фокусов и длины большой и малой осей эллипса. Большая ось – прямой отрезок, который соединяет одну сторону эллипса с другой и проходит через фокусы. Малой осью

называется отрезок, покрывающий меньшее расстояние, под прямым углом пересекающий большую ось посередине между двумя фокусами (в центре эллипса).

Эллипс можно описать через его относительную ориентацию: ввести два фокуса и точку на границе эллипса. Зная координаты этих трех точек, можно найти константу из (5.2).

После этого значения коэффициентов уравнения (5.3) вычисляются и используются для изображения пикселей, расположенных на эллипсе.

Уравнения эллипса значительно упрощаются, если направления большой и малой осей совпадают с координатами осей.

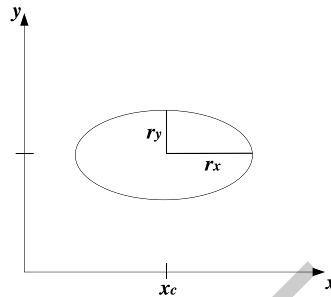


Рис. 5.2. Размещение осей эллипса

Если следовать рис. 5.2, параметр r_x обозначает половину малой оси, а параметр r_y – половину большой оси. Уравнение эллипса с рисунка можно описать через координаты центра эллипса и параметры r_x, r_y:

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1. \tag{5.4}$$

В полярных координатах r и Θ эллипс в стандартном положении можно также описать с помощью параметрического уравнения:

$$\begin{aligned} x &= x_c + r_x \cos\Theta, \\ y &= y_c + r_y \sin\Theta. \end{aligned} \tag{5.5}$$

Угол Θ, называемый углом эксцентриситета эллипса, изменяется по периметру ограничивающей окружности. Если $r_x > r_y$, то радиус ограничивающей окружности равен $r = r_x$. В противном случае радиус будет $r = r_y$.

Как и в алгоритме окружности, количество вычислений можно уменьшить за счет симметрии. Эллипс в стандартном положении симметричен в квадрантах. Можно найти координаты пикселей на дуге эллипса в одном квадранте, а затем найти остальные три точки (рис. 5.3).

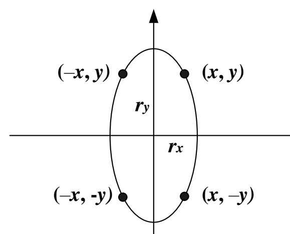


Рис. 5.3. Симметричность отображения точек в эллипсе

Библиотека БГУИР

Алгоритм построения эллипса методом средней точки аналогичен методу для окружности. Если известны параметры r_x, r_y и (x_c, y_c) можно определить координаты точек (x, y) эллипса в стандартном положении с центром в начале координат, после этого сместить все точки так, чтобы получить эллипс с центром в точке (x_c, y_c). Если потребуется изобразить эллипс в нестандартном положении, можно развернуть его вокруг центральной точки, чтобы переориентировать большую и малую оси в нужном направлении.

Метод средней точки для эллипса применяется в обеих частях первого квадранта. Первый квадрант делится на части согласно тангенсу угла наклона эллипса, для которого $r_x < r_y$. При обработке этого квадранта в этой части, где тангенс угла наклона кривой меньше 1, выполняются единичные шаги в направлении x, а затем в той части, где величина тангенса угла наклона больше 1, выполняются единичные шаги в направлении y.

Области 1 и 2 можно обработать разными способами. Можно начать с точки (0, r_y) и двигаться по часовой стрелке по эллиптической траектории в первом квадранте, а затем перейти от единичного шага по x к единичному шагу по y, тогда тангенс угла станет меньше –1.

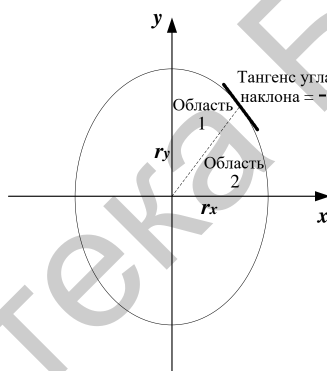


Рис. 5.4. Разделение эллипса на квадранты

Или можно начать с точки (r_x, 0) и выбирать точки в направлении против часов стрелки, перейдя от единичного шага от y к единичному шагу по x, когда тангенс угла наклона станет –1. При наличии параллельных процессоров можно рассчитывать координаты пикселей в двух областях одновременно.

Мы возьмем за начальное положение точку (0, r_y) и будем перемещаться в первом квадранте по часовой стрелке.

Определим функцию эллипса из уравнения (5.4) при (x_c, y_c) = (0, 0):

$$f_{ellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2. \quad (5.6)$$

Эта функция обладает следующими свойствами:

$$f_{ellipse}(x, y) \begin{cases} < 0, \text{ если точка лежит внутри эллипса;} \\ = 0, \text{ если точка лежит на эллипсе;} \\ > 0, \text{ если точка лежит вне эллипса.} \end{cases} \quad (5.7)$$

Эта функция является параметром принятия решения в алгоритме средней точки.

Библиотека БГУИР

Начиная с точки (0, r_y) выполняются единичные шаги в направлении x до тех пор, пока не будет достигнута граница между областями 1 и 2. Затем мы переключается на единичные шаги в направлении y на оставшейся части кривой в первом квадранте. На каждом шаге нужно проверять значение тангенса угла наклона кривой, который находится из уравнения (5.6):

$$\frac{dy}{dx} = -\frac{2{r_y}^2 x}{2{r_x}^2 y}. \quad (5.8)$$

На границе между областью 1 и 2 $dy/dx = -1$ и $2 \cdot r_y^2 \cdot x = 2 \cdot r_x^2 \cdot y$. Следовательно, мы выходим за пределы области 1, когда

$$2{r_y}^2 x \ge 2{r_x}^2 y. \quad (5.9)$$

На рис. 5.5 изображена средняя точка между двумя возможными положениями пикселей, которая определяется при точке выборки x_{k+1} в первой области. Допустим, на предыдущем шаге было выбрано положение (x_k, y_k), тогда найдем следующее положение на эллиптической траектории, вычислив параметр принятия решение в этой средней точке.

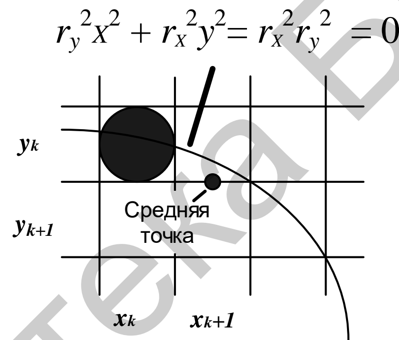


Рис. 5.5. Метод средних для построения эллипса

$$p1_k = fellipse(x_k + 1, y_k - 0{,}5) = {r_y}^2(x_k + 1)^2 + {r_x}^2(y_k - 0{,}5)^2 - {r_x}^2{r_y}^2. \quad (5.10)$$

Если $p1_k < 0$, то средняя точка находится внутри эллипса, тогда пиксель в строке развертки y_k будет ближе к границе эллипса. В противном случае средняя точка лежит за пределами эллипса или на его границе, и выбирается пиксель в строке развертки y_{k-1}.

В следующей точке выборки ($x_{k+1} + 1 = x_k + 2$) параметр принятия решения для области 1 ищется как

$$p1_{k+1} = f_{ellipse}(x_{k+1} + 1, y_{k+1} - 0{,}5) = {r_y}^2(x_{k+1} + 1)^2 + {r_x}^2(y_{k+1} - 0{,}5)^2 - {r_x}^2{r_y}^2$$

или

$$p1_{k+1} = p1_k + 2{r_y}^2(x_k + 1) + {r_y}^2 + {r_x}^2\left[(y_{k+1} - 0{,}5)^2 - (y_k - 0{,}5)^2\right], \quad (5.11)$$

где y_{k+1} равно y_k или y_{k-1} в зависимости от знака $p1_k$.

Параметры принятия решения увеличиваются на следующие величины:

$$\text{прирост} = \begin{cases} 2{r_y}^2 x_{k+1} + {r_y}^2, \text{ если } p1k < 0; \\ 2{r_y}^2 x_{k+1} + {r_y}^2 - 2{r_x}^2 y_{k+1}, \text{ если } p1k \ge 0. \end{cases}$$

Библиотека БГУИР

Приросты для параметров принятия решения вычисляются с использованием только операции сложения и вычисления, как в алгоритме для окружности, поскольку значения членов $2 \cdot r_x^2 \cdot y$ и $2 \cdot r_y^2 \cdot x$ находятся путем сложения приростов. В начальном положении $(0, r_y)$ эти два члена равны

$$2r_y^2 x = 0, \qquad (5.12)$$

$$2r_x^2 y = 2r_x^2 r_y. \qquad (5.13)$$

С увеличением x и y их новые значения находятся путем прибавления $2 \cdot r_y^2$ к текущему значению увеличивающего члена, представленного в уравнении (5.12), и вычитая $2 \cdot r_x^2$ из текущего значения члена, представленного в уравнении (5.13). На каждом шаге сравниваются новые значения приростов, и при выполнении условия (5.9) мы переходим из области 1 в область 2.

В области 1 начальное значение параметра принятия решения находится через функцию эллипса в начальной точке $(x_0, y_0) = (0, r_y)$:

$$p1_0 = f_{ellipse}(1, r_y - 0{,}5) = r_y^2 + r_x^2(r_y - 0{,}5)^2 - r_x^2 r_y^2 = r_y^2 - r_x^2 r_y + 0{,}25 \cdot r_x^2. \qquad (5.14)$$

В области 2 выполняется выборка с единичным интервалом в отрицательном направлении координаты y, а средняя точка на каждом шаге теперь берется между горизонтальными пикселями. Для такой области параметр принятия решения находится как

$$p2_k = f_{ellipse}(x_k + 0{,}5, y_k - 1) = r_y^2(x_k + 0{,}5)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2. \qquad (5.15)$$

Если $p2_k > 0$, то средняя точка находится за пределами эллипса, и выбирается пиксель с координатой x_k. Иначе средняя точка расположена либо внутри, либо на границе эллипса и выбирается пиксель с координатой x_{k+1}.

Чтобы найти взаимосвязь между последовательными параметрами принятия решения в области 2, найдем функцию эллипса для следующего шага $y_{k+1} - 1 = y_k - 2$:

$$p2_{k+1} = f_{ellipse}(x_{k+1} + 0{,}5, y_{k+1} - 1) = r_y^2(x_{k+1} + 0{,}5)^2 + r_x^2((y_k - 1) - 1)^2 - r_x^2 r_y^2 \qquad (5.16)$$

или

$$p2_{k+1} = p2_k - 2r_x^2(y_k - 1) + r_x^2 + r_y^2\left[(x_{k+1} + 0{,}5)^2 - (x_k + 0{,}5)^2\right] \qquad (5.17)$$

при x_{k+1} равным или x_k, или x_{k+1} в зависимости от знака $p2_k$.

При входе в область 2 в качестве начальной точки (x_0, y_0) берется последнее положение в области 1, и тогда начальный параметр принятия решения для области 2 будет равен

$$p2_0 = f_{ellipse}(x_0 + 0{,}5, y_0 - 1) = r_y^2(x_0 + 0{,}5)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2. \qquad (5.18)$$

Чтобы упростить вычисление $p2_0$, положения пикселей можно выбирать в направлении против часовой стрелки, начиная с точки $(r_x, 0)$. Тогда единичные шаги выполняются в положительном направлении оси y до тех пор, пока не будет выбрано последнее положение в области 1.

Этот алгоритм можно применять и для эллипсов с нестандартным положением, воспользовавшись функцией эллипса

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

и вычисляя координаты пикселей на всей траектории.

Библиотека БГУИР

Алгоритм:

1. Вводим r_x, r_y и координаты центра эллипса (x_c, y_c), а затем находим первую точку эллипса с центром в начале координат $(x_0, y_0) = (0, r_y)$.

2. Вычислить начальное значение параметра принятия решения в области 1:

$$p1_0 = f_{ellipse}(1, r_y - 0{,}5) = r_y{}^2 - r_x{}^2 r_y + 0{,}25 \cdot r_x{}^2 .$$

3. Для каждого значения x_k в области 1, начиная с $k = 0$, провести следующую проверку. Если $p1_k < 0$, следующей точкой эллипса с центром (0, 0) будет (x_{k+1}, y_k) и

$$p1_{k+1} = p1_k + 2r_y{}^2 x_{k+1} + r_y{}^2 .$$

В противном случае следующей точкой эллипса будет (x_{k+1}, y_{k-1}) и

$$p1_{k+1} = p1_k + 2r_y{}^2 x_{k+1} - 2r_x{}^2 y_{k+1} + r_y{}^2$$

при

$$2r_y{}^2 x_{k+1} = 2r_y{}^2 x_k + 2r_y{}^2,\ 2r_x{}^2 y_{k+1} = 2r_x{}^2 y_k - 2r_x{}^2 .$$

Этот процесс прекращается, когда $2r_y{}^2 x \ge 2r_x{}^2 y$.

4. Найти начальное значение параметра принятия решения в области 2:

$$p2_0 = r_y{}^2 (x_0 + 0{,}5)^2 + r_x{}^2 (y_0 - 1)^2 - r_x{}^2 r_y{}^2 ,$$

где (x_0, y_0) координаты последней точки области 1.

5. В каждой точке y_k в области 2, начиная с $k = 0$, выполнить проверку: если $p2_k > 0$, то следующей точкой эллипса с центром в точке (0, 0) будет (x_k, y_{k-1}). В противном случае следующей точкой эллипса с центром в точке (0, 0) будет (x_{k+1}, y_{k-1}).

Здесь используются те же расчеты для приростов, что и для области 1. Останавливаемся, когда $y = 0$.

Далее для обеих областей находим симметричные точки в оставшихся трех квадрантах. Также не забываем про смещение эллипса.

Другие кривые

При объектном моделировании, описании траектории в анимации, построении графиков данных и функций, а также в других графических приложениях могут оказаться полезными функции построения различных кривых. Изображения этих кривых можно получить с помощью методов, аналогичных применяющимся для изображения окружностей и эллипсов. Координаты точек на кривой можно найти непосредственно из явного представления функции $y = f(x)$ или из параметрических уравнений. Также можно воспользоваться методом средней точки с нахождением приростов и строить кривую, которая задается в виде неявной функции $f(x, y) = 0$.

Самый простой способ изображения кривой – аппроксимировать ее прямолинейными отрезками. В этом случае полезны параметрические представления,

Библиотека БГУИР

позволяющие получить на траектории кривой точки, находящиеся на одинаковом расстоянии друг от друга. Такие точки можно найти из явного выражения, выбрав независимую переменную в соответствии с тангенсом угла наклона кривой. Там, где величина тангенса угла наклона кривой $y = f(x)$ меньше 1, в качестве независимой переменной выбирается x, и через равные промежутки по этой переменной находятся значения y.

Чтобы получить одинаковые интервалы в той области, где величина тангенса угла наклона больше 1, можно воспользоваться обратной функцией

$$x = f^{-1}(y)$$

и найти значения x через координаты y.

Аппроксимация в виде прямой или кривой линии используется также для построения линейного графика набора дискретных значений. Дискретные точки можно соединить прямолинейными отрезками либо воспользоваться методом наименьших квадратов и аппроксимировать набор данных одной прямой линией. Нелинейный метод наименьших квадратов используется при изображении набора данных с помощью некоторой аппроксимирующей функции, обычно в этой роли выступает полином.

Кроме эллипса и окружности определенной симметрией обладают многие функции, и этим можно воспользоваться для уменьшения объема вычислений при расчете координат пикселей на кривой.

Коническое сечение

В общем случае коническое сечение можно описать с помощью уравнения второго порядка:

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0, \tag{5.19}$$

где параметры A, B, C, D, E, F определяют тип изображаемой кривой. Если этот набор коэффициентов известен, конкретный вид конического сечения, которое получается из этого уравнения, можно определить, вычислив дискриминант $B^2 - 4AC$:

$$B^2 - 4AC \begin{cases} < 0, \text{ эллипс (или окружность);} \\ = 0, \text{ парабола;} \\ > 0, \text{ гипербола.} \end{cases}$$

В некоторых приложениях дуги окружности и эллипсов удобно задавать значениями угловых координат начала и конца дуги. Иногда такие дуги определяются координатами их концов. Дугу можно построить с помощью видоизмененного метода средней точки или аппроксимировать ее набором прямолинейных отрезков (рис. 5.6).

Библиотека БГУИР

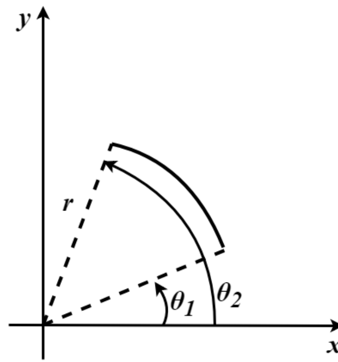


Рис. 5.6. Вид дуги

Эллипсы, параболы и гиперболы оказываются особенно полезными в некоторых анимационных приложениях. Эти кривые описывают движение по орбите и другие движения объектов под действием гравитационных, электромагнитных или ядерных сил. Орбиты планет Солнечной системы, например, можно аппроксимировать эллипсами. А объект, помещенный в однородное гравитационное поле, движется по параболической траектории.

Явный вид уравнения параболической траектории можно записать как

$$y = y_0 + a(x - x_0)^2 + b(x - x_0), \tag{5.20}$$

где константы a, b описывают начальную скорость объекта v_0 и ускорение g, вызванное однородной гравитационной силой (рис. 5.7).

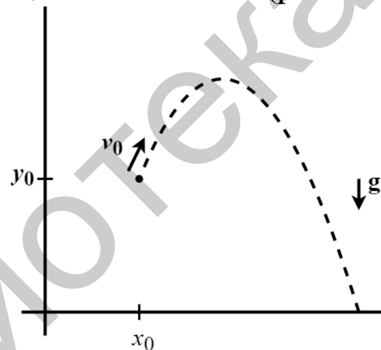


Рис. 5.7. Параболическая траектория

Такое параболическое движение можно также описать с помощью параметрических уравнений, воспользовавшись параметром времени t (с):

$$\begin{aligned} x &= x_0 + v_{x0}t, \\ y &= y_0 + v_{y0}t - gt^2 / 2. \end{aligned} \tag{5.21}$$

Здесь v_{x0} и v_{y0} – компоненты начальной скорости, а g около поверхности Земли составляет приблизительно 9,8 м/с^2. Затем рассчитываются положения объекта на этой параболической траектории через определенные промежутки времени.

Гиперболические кривые могут пригодиться в приложениях, связанных с визуализацией научных исследований. Движение объектов по гиперболической

Библиотека БГУИР

траектории возникает при столкновении заряженных частиц, а также в определенных задачах, связанных с гравитацией. Выбор правой или левой ветки гиперболы зависит от сил, которые рассматриваются в задаче. Стандартное уравнение для гиперболы с центром в начале координат имеет вид

$$\left(\frac{x}{r_x}\right)^2 - \left(\frac{y}{r_y}\right)^2 = 1 \qquad (5.22)$$

при $x \le -r_x$ для левой ветки или $x \ge -r_x$ – для правой. Так как это уравнение отличается от стандартного уравнения эллипса только знаком между членами уравнения, то координаты на гиперболической прямой находятся с помощью видоизмененного алгоритма для эллипса.

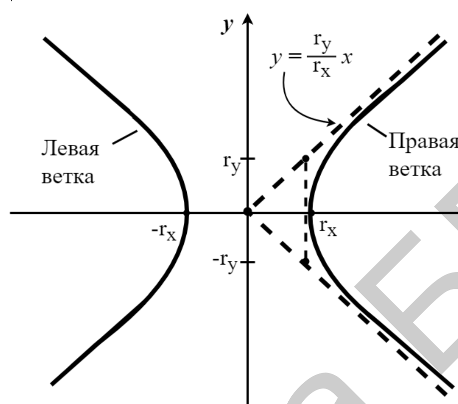


Рис. 5.8. Гиперболическая траектория

Параболы и гиперболы имеют ось симметрии. Например, парабола, которая описывается уравнением (5.5), симметрична относительно оси

$$x = x_0 + v_{x0}v_{y0} / g .$$

Методы, которые используются в алгоритме средней точки эллипса, можно непосредственно применять для поиска точек на одной стороне параболической или гиперболической кривой в двух областях:

1) там, где величина тангенса угла наклона кривой меньше 1;

2) там, где величина тангенса угла наклона кривой больше 1.

Чтобы сделать это, вначале определяется соответствующий вид уравнения конического сечения, а затем выбранная функция используется для вычисления параметров принятия решения в двух областях.

Полиномы и сплайны

Полиномиальная функция n-го порядка по x определяется как

$$y = \sum_{k=0}^{n} a_k x^k = a_0 + a_1 x^1 + ... + a_n x^n , \qquad (5.23)$$

где n – натуральное число, а коэффициенты a_i – константы, причем $a_n \ne 0$.

При $n = 1$ получим прямую линию, при $n = 2$ – кривую второго порядка, при $n = 3$ – кубический полином, при $n = 4$ – кривую четвертого порядка и т. д.

Библиотека БГУИР

Полиномы используются в ряду графических приложений, в том числе при проектировании форм предметов, описании траекторий в анимации, построении графиков данных по дискретным наборам точек.

При проектировании формы предмета или траектории движения, как правило, сначала задаются несколько точек, которые определяют общий вид кривой и представляют собой необходимый контур, а затем по выбранным точкам подбирается подходящий полином. Один из способов подбора кривой – соединить каждую пару заданных точек кубическим участком кривой. Каждый такой участок описывается в параметрической форме следующим образом:

$$x = a_{x0} + a_{x1}u + a_{x2}u^2 + a_{x3}u^3,$$

$$y = a_{y0} + a_{y1}u + a_{y2}u^2 + a_{y3}u^3,$$

где параметр u приобретает значения от 0 до 1.

Значения коэффициентов u в предыдущих уравнениях определяются из граничных условий для участков кривой. Одно из граничных условий заключается в том, что конец одного участка совпадает с началом следующего, а второе условие – совпадение тангенсов угла наклона двух кривых на общей границе, так что в результате получается непрерывная гладкая кривая. Непрерывные кривые, образованные из полиномиальных фрагментов, называются сплайновыми кривыми, или просто *сплайнами*.

Параллельные алгоритмы построения кривых

При построении кривых возможности параллельной обработки используются так же, как и при изображении прямых линий. Можно либо адаптировать для этого последовательный алгоритм, выделив каждому процессору отдельный участок кривой, либо придумать другой способ и соотнести с процессорами отдельные части экрана.

Параллельный метод изображения окружности с помощью средней точки заключается в том, чтобы разделить дугу окружности от 45º до 90º на равные поддуги и сопоставить с каждой из этих частей отдельный процессор. Как и в параллельном алгоритме Брезенхема для прямой линии, для каждого процессора нужно вывести формулу для расчета начального значения y и параметра принятия решения p_k. На каждой поддуге рассчитываются координаты пикселей, а положения точек в остальных октантах можно найти из условия симметрии.

Аналогичным образом в параллельном методе средней точки для эллипса эллиптическая дуга в первом квадранте делится на поддуги, которые распределяются между отдельными процессорами.

Схема разделения экрана для окружностей и эллипсов заключается в том, чтобы каждому процессору поставить в соответствие отдельную строку развертки, которая пересекается с кривой. В этом случае каждый процессор из уравнения окружности и эллипса находит координаты точки пересечения с кривой.

Для изображения эллиптических дуг и других кривых можно просто воспользоваться методом распределения строк развертки. Каждый процессор по

Библиотека БГУИР

уравнению кривой определяет точки пересечения данной кривой с выделенной ему соответственно строкой развертки. Если с процессором соотносятся отдельные пиксели, тогда каждый процессор будет рассчитывать расстояние (или квадрат расстояния) от кривой до выделенного ему пикселя. Если рассчитанное расстояние меньше предопределенного значения, тогда пиксель изображается на экране.

5.2. Индивидуальные задания

Самостоятельно вывести алгоритм средней точки для построения эллипса с координатами центра (x_0, x_1) и большим и малым радиусом r_0 и r_1 соответственно. Реализовать алгоритм с помощью компьютерных приложений и получить изображение окружности в соответствии с вариантом, предложенным в табл. 5.1.

Таблица 5.1

Варианты индивидуальных заданий к теме 5

№ варианта	Координаты центра, (x_0, x_1)	Радиус большой, r_1	Радиус малый, r_0
1	(0, 9)	13	6
2	(2, 12)	10	5
3	(–5, 8)	8	4
4	(–1, 10)	7	3
5	(1, 13)	10	5
6	(–4, 9)	8	4
7	(–2, 7)	5	2
8	(4, 14)	13	6
9	(3, 13)	12	6
10	(4, 9)	8	4
11	(–4, 9)	8	4
12	(2, 11)	10	5
13	(0, 17)	15	8
14	(2, 12)	11	6
15	(0, 10)	9	5

Библиотека БГУИР

Тема №6. Методы определения видимости для каркасных изображений. Закрашивание многоугольников

Цель работы: изучить существующие методы определения выгнутости/выпуклости многоугольников, основные методы определения принадлежности точки фигуре; написать и отладить программу, решающую эти задачи.

6.1. Общие теоретические сведения

При описании элементов рисунка, кроме точек, прямолинейных отрезков и кривых используется еще одна полезная конструкция – область, закрашенная каким-то одним цветом или заполненная определенным узором. Такие элементы рисунка обычно называют цветными фигурами или закрашенными областями. Чаще всего закрашенные фигуры используются для описания поверхностей сплошных объектов, хотя могут пригодиться и в других приложениях. Закрашенные фигуры, как правило, представляют собой плоские поверхности, главным образом, многоугольники. В то же время существует множество форм участков рисунка, которые может потребоваться закрасить с помощью какой-нибудь цветовой опции. Пока будем считать, что закрашивание монохромное.

Хотя форма закрашенной области может быть любой, графические системы, как правило, не поддерживают спецификации произвольных закрашенных фигур. В большей части стандартных процедур требуется, чтобы закрашенная область задавалась в виде многоугольника. Графические функции способны обрабатывать многоугольники намного эффективнее, чем другие виды закрашенных фигур, т. к. границы таких областей описаны линейными уравнениями. Более того, большинство криволинейных поверхностей можно достаточно корректно аппроксимировать набором правильных многоугольников как кривую линию – набором прямолинейных отрезков.

После наложения эффектов освещения и затенения аппроксимированная криволинейная поверхность выглядит вполне реалистично. Аппроксимацию изогнутой поверхности с помощью многоугольных граней иногда называют мозаичным представлением поверхности или аппроксимацией поверхности с помощью сетки многоугольников. На рис. 6.1 изображены боковая и верхняя поверхности цилиндра, аппроксимированные сеткой многоугольников в виде контурной схемы. Изображения таких фигур можно быстро получить в виде каркасных схем, на которых показаны только края многоугольников, по которым получается общее представление о структуре поверхности. После этого каркасную модель можно заштриховать и получить изображение поверхности из материала, выглядящего естественным образом. Объекты, описанные с помощью набора многоугольных участков поверхности, как правило, называют стандартными графическими объектами или просто графическими объектами.

Библиотека БГУИР

В общем случае можно создать закрашенную область с любой границей. Кроме того, некоторые методы можно адаптировать для вывода на экран закрашенных областей с нелинейными границами.

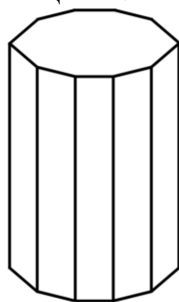


Рис. 6.1. Изогнутая поверхность

Многоугольник – плоская фигура, которая задается с помощью набора из трех и более вершин, последовательно соединенных прямолинейными отрезками, называемыми ребрами. Так, ребра многоугольника не должны иметь общих точек, кроме вершин. Таким образом, все вершины многоугольника должны лежать в одной плоскости, а его края не должны пересекаться.

В приложениях компьютерной графики возможны ситуации, когда не все перечисленные вершины лежат строго в одной плоскости. Это может быть связано с ошибкой округления при вычислении значений, с ошибкой выбора координат вершин или с аппроксимацией криволинейной поверхности набором многоугольных участков. Один из способов исправления этого – разделить заданную сетку многоугольников на треугольники. Но в некоторых случаях могут существовать причины, по которым необходимо сохранить изначальный вид ячеек сетки, поэтому были придуманы методы аппроксимации неплоских многоугольных фигур плоскими объектами.

Внутренний угол – угол внутри границы многоугольника, образованный двумя соседними сторонами.

Если все стороны угла многоугольника меньше или равны 180º, то многоугольник *выпуклый* (рис. 6.2). Эквивалентное определение – весь многоугольник должен лежать по одну сторону от бесконечной прямой, которая является продолжением любой из его сторон. К тому же, если внутри выпуклого многоугольника выбрать любые две точки и соединить их, то отрезок полностью будет принадлежать внутренней поверхности многоугольника.

Многоугольник, который не удовлетворяет предыдущим определениям, называется *вогнутым* (рис. 6.3).

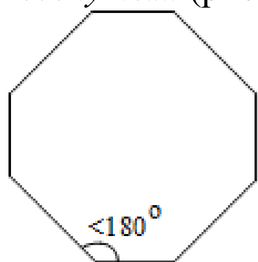


Рис. 6.2.
Выпуклый многоугольник

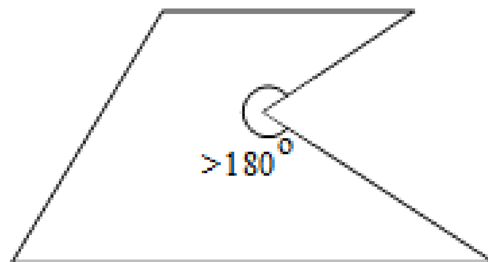


Рис. 6.3.
Вогнутый многоугольник

Библиотека БГУИР

Термином *вырожденный* многоугольник часто описывают набор вершин, которые являются коллинеарными или координаты которых повторяются. Коллинеарные вершины дают прямолинейный отрезок. Вершины с повторяющимися координатами могут дать многоугольную фигуру с лишними линиями, перекрывающимися сторонами, или со сторонами нулевой длины. Иногда термин вырожденный многоугольник еще применяется к списку вершин, который состоит менее чем из трех точек.

Устойчивые графические пакеты могут не воспринимать вырожденные или неплоские наборы вершин. Так, на распознавание требуется дополнительная обработка, поэтому в графических системах выбор ограничивающих условий, как правило, предоставляется на выбор программиста.

Вогнутые многоугольники также создают дополнительные проблемы. Реализация алгоритмов закрашивания и других графических процедур для вогнутых многоугольников намного сложнее, поэтому, как правило, перед обработкой эффективнее разделить вогнутый многоугольник на набор выпуклых.

У вогнутого многоугольника по крайней мере один внутренний угол больше чем 180º. Кроме того, продолжения некоторых сторон вогнутого многоугольника будут пересекать другие ребра, а некоторые пары внутренних точек дадут отрезки, пересекающие границы многоугольника. Любой из этих признаков можно положить в основу алгоритма распознавания вогнутых многоугольников.

Если рассматривать каждую сторону многоугольника как вектор, то в проверке на вогнутость может использоваться векторное произведение двух соседних сторон. Для выпуклого многоугольника такие векторные произведения будут одного знака (положительные или отрицательные). Следовательно, если не все векторные произведения одного знака, то исследуемый треугольник вогнутый (рис. 6.4).

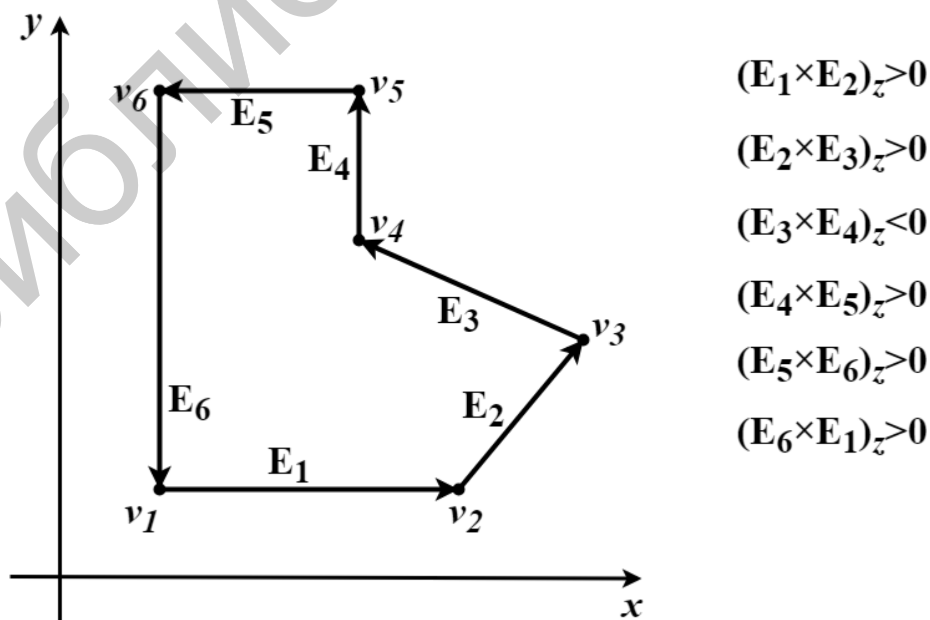


Рис. 6.4. Разложение сторон многоугольника на векторы

Библиотека БГУИР

Вогнутый многоугольник можно разделить на набор выпуклых с помощью векторов сторон и их векторных произведений или с помощью определения, какие вершины находятся по одну сторону от линии, а какие – по другую, используя положения вершин относительно линии продолжения одной из сторон. Для реализации следующих алгоритмов допустим, что все многоугольники находятся в плоскости xy. Разумеется, исходный многоугольник, описанный во внешних координатах, может и не лежать в плоскости xy, но его всегда можно переместить в эту плоскость, воспользовавшись методами преобразования.

Для реализации *векторного метода* разделения вогнутого многоугольника (рис. 6.5) сначала необходимо построить векторы сторон. Если известны координаты двух соседних вершин V_k и V_{k+1}, то вектор стороны между ними определяется следующим образом:

$$E_k = V_{k+1} - V_k.$$

Далее необходимо найти векторные произведения соседних векторов сторон по порядку по всему периметру многоугольника. Если компонент z одних векторных произведений будет положительным, а других – отрицательным, то многоугольник вогнутый. В противном случае многоугольник выпуклый.

Здесь предполагается, что среди вершин нет трех последовательных коллинеарных точек, поскольку в таком случае векторное произведение двух векторов сторон, содержащих эти вершины, будет равно нулю. Если все вершины коллинеарные, то многоугольник вырожден (прямая линия). Векторный метод можно применять, обрабатывая стороны последовательно против часовой стрелки. Если у какого-то векторного произведения появится отрицательный z, то многоугольник – вогнутый, и его можно разделить, продолжив первый вектор стороны из той пары векторов, произведение которых рассматривается.

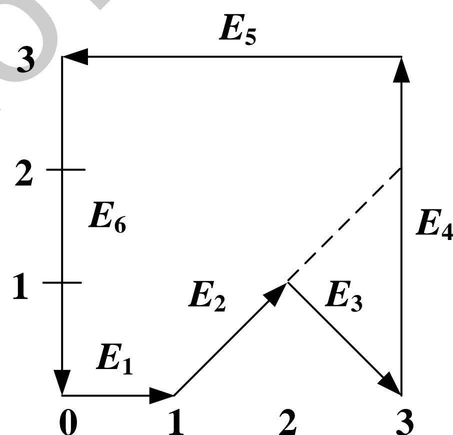


Рис. 6.5. Векторный метод определения вогнутости/выпуклости многоугольников

Кроме того, вогнутый многоугольник можно разделить с помощью *метода вращения* (рис. 6.6). Перемещаясь по сторонам многоугольника в направлении против часовой стрелки, многоугольник разворачивается таким образом,

Библиотека БГУИР

что каждая вершина по очереди попадала в начало координат. Затем многоугольник разворачивается относительно начала координат по часовой стрелке, так, чтобы следующая вершина лежала на оси *x*. Если при этом следующая вершина находится на оси *x*, то многоугольник вогнутый. Этот многоугольник разделяется по оси *x*, и тогда образуются два новых многоугольника. Затем проверка на выпуклость повторяется для двух новых многоугольников. Эти шаги повторяются до тех пор, пока не будут проверены все вершины многоугольника.

После перемещения вершины *V*2 в начало координат и поворота, при котором вершина *V*3 помещается на ось *x*, видим, что вершина *V*4 находится ниже оси *x*. Поэтому далее разделяем многоугольник по линии *V*2*V*3, которая совпадает с осью *x*.

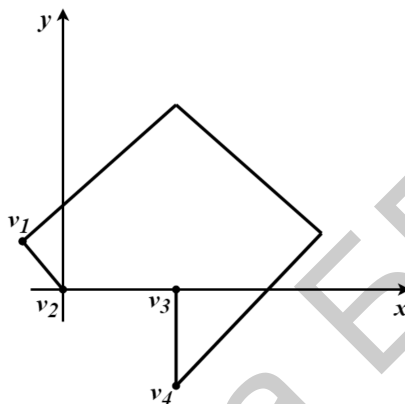


Рис. 6.6. Метод вращения определения вогнутости/выпуклости многоугольников

Если дан список вершин выпуклого многоугольника, его можно преобразовать в набор треугольников. Для этого вначале определяется любая последовательность из трех идущих подряд вершин, которые образуют новый многоугольник (треугольник). После этого из исходного списка вершин удаляется средняя вершина треугольника. Затем эта процедура выполняется с измененным списком вершин, и вырезается еще один треугольник. Формирование треугольников продолжается до тех пор, пока исходный список вершин многоугольника не уменьшится до трех, которые и определяют координаты вершин последнего треугольника этого набора. С помощью этого метода вогнутый многоугольник также можно разделить на набор треугольников, продолжая процесс до тех пор, пока три выбранные вершины на каждом этапе будут образовывать внутренний угол, не превышающий 180º («выпуклый» угол).

В различных графических процессах часто приходится определять внутренние области объектов. Распознавание внутренней области такого простого объекта, как выпуклый многоугольник, окружность или сфера, – процесс несложный. Однако иногда приходится иметь дело с более сложными объектами. В таких ситуациях не всегда понятно, какие участки плоскости *xy* следует отнести к «внутренним», а какие к «внешним». Существует два общепринятых алгоритма определения внутренних областей плоских фигур – правило четного-нечетного и правило ненулевого числа витков.

Библиотека БГУИР

Чтобы воспользоваться *правилом четного-нечетного* (рис. 6.7), сначала мысленно проведем прямую, соединяющую любую точку P с какой-то удаленной точкой за пределами области координат замкнутой ломаной линии. Затем сосчитаем количество точек пересечения этой прямой с прямолинейными участками. Если число отрезков, пересекающих линию, нечетное, тогда P считается внутренней, в противном случае – внешней. Этим можно воспользоваться для заполнения заданным цветом внутренней области между многоугольниками.

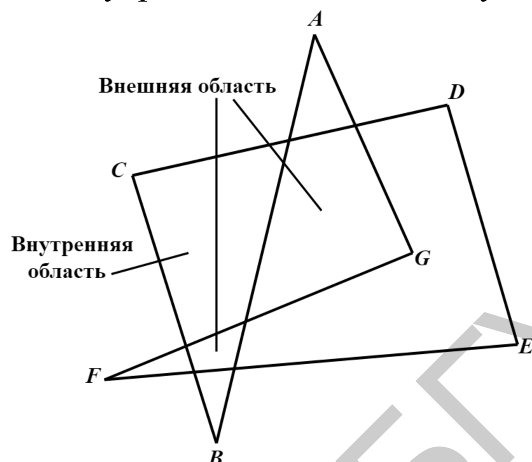


Рис. 6.7. Правило четного-нечетного

Еще один способ определения внутренних областей – *правило ненулевого количества витков* (рис. 6.8), при котором считается, сколько раз граница объекта оборачивается вокруг отдельной точки в направлении против часовой стрелки. Это число называется количеством витков, а внутренние точки двумерного объекта можно определить по тому, имеют ли они ненулевое количество витков. Правило ненулевого количества витков применяется так: сначала количество витков устанавливается в 0, а затем опять представляем себе прямую линию, соединяющую точку P с удаленной точкой за пределами области координат объекта. Выбранная прямая не должна проходить ни через один конец отрезка. При перемещении по этой линии от P до удаленной точки считается количество прямолинейных отрезков, принадлежащих объекту, которые пересекают опорную линию в каждом направлении. При каждом пересечении прямой с отрезком в направлении справа налево к количеству витков прибавляется 1, а при пересечении с отрезком, идущим справа налево, отнимается 1. Конечное значение числа витков после учета всех пересечений с границами определяет относительное положение точки P. Если число витков равно 0, то P – внутренняя.

Чтобы определить направление границы в точке пересечения, можно представить стороны объекта (или отрезки, образующие границу) в виде векторов, а также задать направление опорной линии. После этого для каждой стороны, с которой пересекается опорная линия, вычисляется векторное произведение вектора u, направленного по опорной прямой от точки P к удаленной точке, и вектора стороны объекта E. Допустим, что дан 2D-объект на плоскости xy. Тогда все векторные произведения будут направлены либо по $+z$, либо по $–z$. Если компо-

Библиотека БГУИР

нент z векторного произведения $u \cdot E$ для определенной точки пересечения положителен, то отрезок пересекает прямую справа налево, и к количеству витков прибавляется 1, а противном случае – отнимается 1.

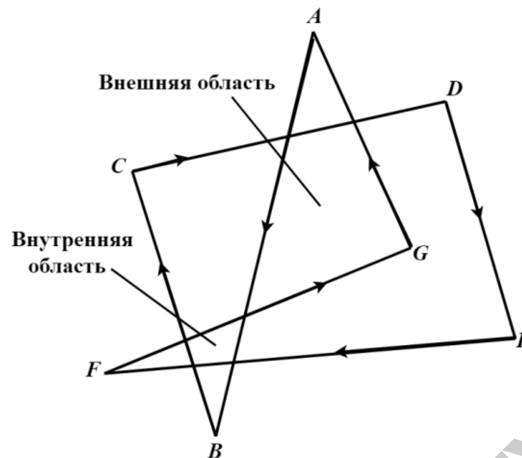


Рис. 6.8. Правило ненулевого количества витков

Еще более простым способом определения направления в точке пересечения с границей является использование не векторного, а скалярного произведения векторов. Для этого нужно задать вектор, перпендикулярный *u* и направленный справа налево, если смотреть вдоль опорной прямой из точки *P* в направлении вектора *u*.

Если компоненты вектора *u* обозначить как (u_x, u_y), компоненты вектора, перпендикулярного к вектору *u*, будут равны ($-u_y$, u_x). Если теперь скалярное произведение этого перпендикулярного вектора и вектора, проложенного по границе, положительное, то направление пересечения – справа налево, и нужно увеличить количество витков.

Правило ненулевого количества витков часто относит к внутренним некоторые области, которые при применении правила четного-нечетного считаются внешними, а в некоторых приложениях этот метод может быть еще более непредсказуем. В общем случае плоские фигуры могут состоять из множества не связанных между собой элементов, и тогда для определения внутренних и внешних областей можно использовать направление, которое задается для каждого набора не связанных между собой границ. К таким объектам относятся символы, вложенные многоугольники, эллипсы. Для кривых линий правило четного-нечетного применяется в форме подсчета числа точек пересечения с кривой. А для правила ненулевого числа витков необходимо найти векторы, направленные по касательной к кривой в точках ее пересечения с опорной прямой, проходящей через точку *P*.

Некоторые разновидности правила ненулевого числа витков позволяют несколько иначе определить внутренние области. Например, точку можно считать внутренней, если количество витков для нее положительно или оно отрицательно. Кроме того, можно воспользоваться любым другим правилом и создать

Библиотека БГУИР

ряд закрашенных фигур. Иногда закрашенная область задается в виде комбинации двух областей с помощью логических операций при использовании одной из разновидностей стандартного алгоритма ненулевого числа витков (рис. 6.9). Согласно этой схеме сначала для каждой из двух областей находятся простые, непересекающиеся границы. Тогда, если считать, что каждая граница обходится против часовой стрелки, объединение двух областей будет состоять из точек с положительным числом витков.

Аналогично область пересечения двух фигур, границы которых обходятся против часовой стрелки, будет содержать только те точки, количество витков которых превышает 1 (рис. 6.10).

Чтобы задать закрашенную область, которая представляет собой разность двух областей ($A - B$), можно обвести область A границей, направленной против часовой стрелки, а область B – по часовой стрелке (рис. 6.11).

Рис. 6.9. Пример закрашивания по правилу ненулевого количества витков

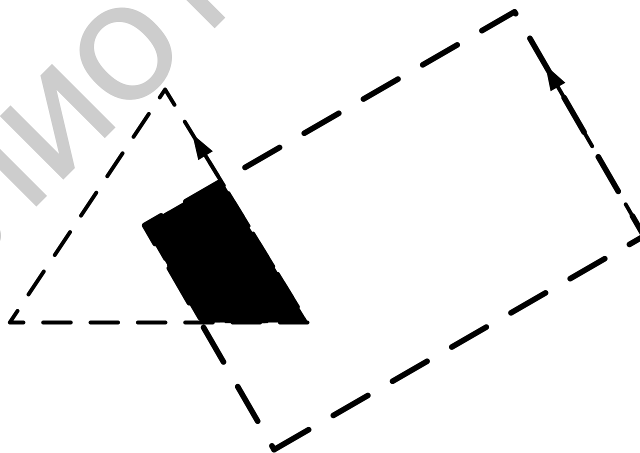


Рис. 6.10. Пример закрашивания пересечения фигур по правилу ненулевого количества витков

Библиотека БГУИР

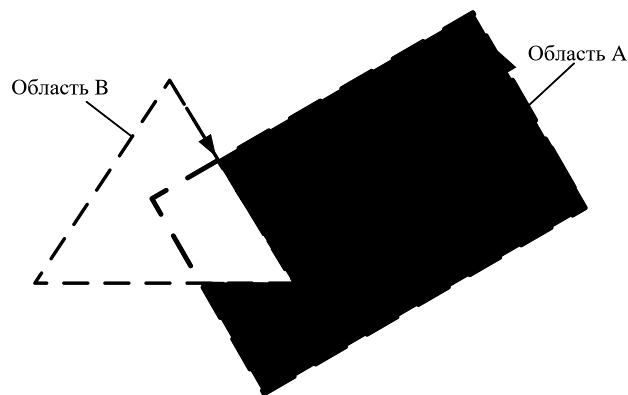


Рис. 6.11. Пример закрашивания разности фигур по правилу ненулевого количества витков

Таблицы многоугольников

Как правило, объекты сцены описываются как наборы многоугольных граней поверхностей (рис. 6.12). Фактически графические пакеты часто предлагают функции определения формы поверхности в виде сетки многоугольных участков. Описание каждого объекта включает информацию о координатах, с помощью которых задается геометрия многоугольных граней и другие параметры поверхностей (цвет, прозрачность, светоотражение). Так как сведения о каждом многоугольнике – входные параметры, данные размещаются в таблицах, которые используются при последующей обработке, выводе на экран и выполнении различных операций с объектами сцены. Эти информационные таблицы многоугольников можно объединить в две группы: геометрические таблицы и таблицы параметров. В геометрических таблицах содержаться координаты вершин и параметры, позволяющие определить пространственную ориентацию многоугольных поверхностей. К информации о параметрах объекта относятся величины, определяющие степень прозрачности объекта, отражающую способность его поверхности и текстурные характеристики.

Геометрическую информацию об объектах сцены удобно распределить по трем спискам: таблица вершин (рис. 6.13), таблица сторон (рис. 6.14), таблица граней поверхности (рис. 6.15).

Координаты всех вершин объекта записываются в таблицу вершин. В таблице сторон содержатся ссылки на таблицу вершин, позволяющие определить вершины, принадлежащие каждой стороне многоугольника. В таблице граней поверхности содержатся ссылки на таблицу сторон, определяющие границы каждого многоугольника. Кроме того, отдельным объектам и составляющим их многоугольным граням можно присвоить специальные указатели на объект или грань, что облегчает обращение к ним.

Запись геометрической информации в виде таблиц обеспечивает удобный способ обращения к отдельным элементам (вершинам, сторонам и граням) каждого объекта. Кроме того, так можно эффективно изображать объекты, используя для определения границ многоугольника информацию из таблицы сторон.

Библиотека БГУИР

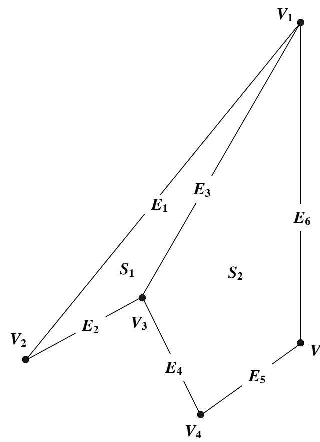


Рис. 6.12. Геометрический объект сцены

Таблица вершин		
V_1	:	x_1, y_1, z_1
V_2	:	x_2, y_2, z_2
V_3	:	x_3, y_3, z_3
V_4	:	x_4, y_4, z_4
V_5	:	x_5, y_5, z_5

Рис. 6.13.
Таблица вершин

Таблица сторон		
E_1	:	V_1, V_2
E_2	:	V_2, V_3
E_3	:	V_3, V_1
E_4	:	V_3, V_4
E_5	:	V_4, V_5
E_6	:	V_5, V_1

Рис. 6.14.
Таблица сторон

Таблица граней поверхности		
S_1	:	E_1, E_2, E_3
S_2	:	E_3, E_6, E_5, E_6

Рис. 6.15.
Таблица граней
поверхности

Другой вариант расположения данных – использовать только таблицу вершин и таблицу граней. Но такая схема менее удобна, а при выводе на экран каркасной модели некоторые стороны могут изображаться дважды. Еще один вариант – использовать только таблицу граней поверхности, но при этом дублируется информация о координатах, поскольку для каждой многоугольной грани в явном виде записываются значения координат всех вершин. Кроме того, взаимосвязь между сторонами и гранями пришлось бы восстанавливать по списку вершин в таблице поверхностных граней.

Чтобы ускорить извлечение информации, информационные таблицы дополняются некоторыми данными. Например, можно расширить таблицу сторон, включив в нее указатели на таблицу поверхностных граней, чтобы определить общие стороны многоугольников.

Библиотека БГУИР

Это особенно удобно для процедур визуализации, в которых затенения поверхностей при переходе через ребро от одного многоугольника к другому должно изменяться плавно. Аналогично для более быстрого доступа к данным можно расширить таблицу вершин, дополнив ее ссылками на соответствующие стороны.

К дополнительной графической информации, которая обычно хранится в информационных таблицах, относятся тангенс угла наклона каждой стороны и координатные границы сторон многоугольников, многоугольных граней и каждого объекта сцены в целом. Так как вводятся координаты вершин, можно рассчитать тангенс угла наклона каждой стороны, а также просмотреть все значения координат и определить минимальные и максимальные значения x, y, z для отдельных линий и многоугольников. Информация о тангенсе угла наклона и ограничивающем многоугольнике необходима для последующей обработки, такой как визуализация поверхностей, идентификация видимых участков поверхности.

Так как таблицы геометрических данных для сложных объектов и сцен могут содержать очень большие списки вершин и сторон, важно проверить, чтобы все данные не противоречили друг другу и были полными. Когда задаются определения вершины, стороны или грани, особенно в интерактивных приложениях, возможны определенные ошибки при вводе данных, которые приведут к искаженному изображению объекта. Чем больше информации содержится в виде информационных таблиц, тем легче проверять ее на наличие ошибок. Следовательно, проверку ошибок легче выполнять, если используются три информационные таблицы (вершин, сторон, граней), поскольку такая схема предполагает наиболее полную информацию. Некоторые из проверок, которые могут выполнять графические пакеты:

1) каждая из вершин должна значиться крайней точкой по меньшей мере двух сторон;

2) каждая сторона должна быть частью как минимум одного многоугольника;

3) все многоугольники замкнуты;

4) у каждого многоугольника должна быть хотя бы одна общая сторона с другими многоугольниками.

Передние и задние грани многоугольника

Обычно многоугольные поверхности, с которыми приходится иметь дело в графических приложениях, представляют собой внешнюю оболочку объекта, необходимо уметь различать две стороны каждой поверхности. Та сторона, которая направлена внутрь объекта, называется задней, а видимая – передней. Определение положения точек в пространстве относительно передней и задней сторон многоугольника является основной задачей многих графических алгоритмов, например, алгоритма определения видимых частей объекта. Каждый многоугольник лежит в бесконечной плоскости, разделяющей пространство на две части. Говорят, любая точка, не принадлежащая этой плоскости, которая видна с лицевой стороны поверхности многоугольника, находится перед плоскостью

Библиотека БГУИР

(снаружи), следовательно, за пределами объекта. Любая точка, которая видна с обратной стороны многоугольника, находится за плоскостью (внутри). Точка, которая находится за всеми плоскостями поверхностей многоугольника, находится внутри объекта. Важно помнить, что классификация «внутри-снаружи» производится относительно плоскости, содержащей многоугольник.

Чтобы определить положения точек в пространстве относительно многоугольных граней объекта, воспользуемся уравнением плоскости. Для любой точки (x, y, z), не принадлежащей плоскости с параметрами A, B, C, D, мы имеем

$$Ax + By + Cz + D \neq 0.$$

То есть по знаку полученного значения выражения мы можем определить, находится ли данная точка за поверхностью многоугольника, лежащего в этой плоскости, или перед ней:

$$Ax + By + Cz + D \begin{cases} < 0, \text{ точка находится за плоскостью;} \\ > 0, \text{ точка находится перед плоскостью.} \end{cases}$$

Эти проверки с помощью неравенств справедливы для правосторонних декартовых систем координат, в которых параметра плоскости A, B, C, D вычисляются с помощью координат, выбранных строго против часовой стрелки, если смотреть на поверхность в направлении от передней стороны к задней (рис. 6.16).

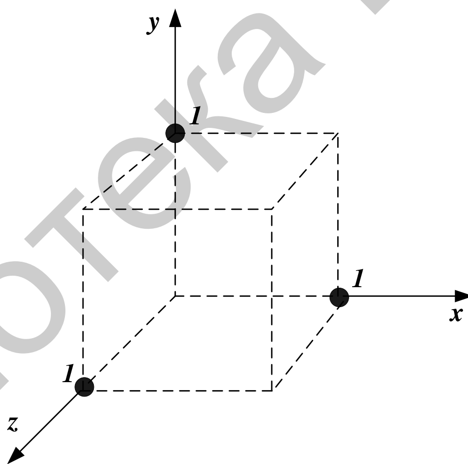


Рис. 6.16. Ориентация фигуры в пространстве

Ориентацию многоугольной поверхности в пространстве можно описать с помощью вектора нормали к плоскости, содержащей этот многоугольник (рис. 6.17). Этот вектор нормали к поверхности перпендикулярен плоскости, а его декартовы координаты – (A, B, C), где параметры A, B, C – коэффициенты плоскости. Вектор нормали указывает направление от внутренней стороны плоскости к наружной, т. е. от задней стороны многоугольника к передней.

Компоненты вектора также можно найти, вычисляя векторное произведение. Предположим, что дана грань поверхности, имеющая вид выпуклого многоугольника, и правосторонняя декартова система координат. Снова выберем любые три вершины V_1, V_2, V_3, взятые против часовой стрелки, если смотреть в направлении снаружи внутрь объекта. Сформируем два вектора: один – от точки

Библиотека БГУИР

V_1 до точки V_2, а второй – от точки V_1 до точки V_3, и найдем N как векторное произведение этих двух векторов:

$$N = (V_2 - V_1) \cdot (V_3 - V_1).$$

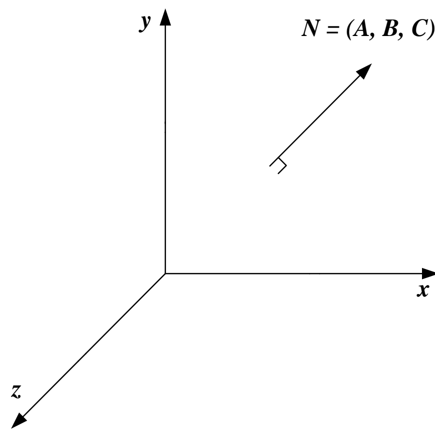


Рис. 6.17. Вектор нормали к плоскости многоугольника

В результате получим значения параметров A, B, C. После этого можно найти значение параметра D, подставив полученные величины и координаты одной из вершин многоугольника в уравнение плоскости и решив его относительно D. Уравнение плоскости можно представить в векторной форме через нормаль N и положение любой точки P на этой плоскости:

$$N \cdot P = -D.$$

Для выпуклого многоугольника параметры плоскости можно найти и с помощью векторного произведения векторов двух соседних сторон. Для вогнутого многоугольника можно таким образом выбрать три вершины, чтобы два названных вектора образовывали угол меньше 180º. В противном случае можно взять вектор, обратный по знаку их векторному произведению, что даст вектор нормали для поверхности многоугольника с правильным направлением.

6.2. Индивидуальные задания

Получить изображение закрашенного многоугольника в соответствии с вариантом, предложенным в табл. 6.1 (формирование фигуры получается обходом вершин по часовой стрелке). Определить выгнутость-выпуклость полученной фигуры.

Библиотека БГУИР

Таблица 6.1

Варианты индивидуальных заданий к теме 6

№ варианта	(x_0, y_0)	(x_1, y_1)	(x_2, y_2)	(x_3, y_3)	(x_4, y_4)
1	(10, 10)	(50, 10)	(50, 30)	(40, 10)	(30, 20)
2	(0, 20)	(20, 20)	(30, 40)	(30, 10)	(20, 0)
3	(0, 0)	(20, 20)	(20, 40)	(30, 20)	(20, 0)
4	(10, 10)	(40, 15)	(50, 35)	(40, 10)	(25, 20)
5	(10, 10)	(50, 10)	(45, 30)	(40, 10)	(30, 20)
6	(0, 0)	(20, 20)	(20, 40)	(30, 30)	(20, 0)
7	(25, 10)	(50, 10)	(55, 30)	(40, 10)	(15, 20)
8	(0, 20)	(20, 20)	(30, 40)	(25, 10)	(20, 0)
9	(10, 20)	(20, 20)	(30, 40)	(30, 10)	(20, 0)
10	(0, 10)	(50, 10)	(45, 25)	(40, 10)	(30, 5)
11	(0,10)	(20, 20)	(20, 40)	(30, 20)	(20, 0)
12	(10, 0)	(60, 15)	(40, 30)	(45, 5)	(30, 20)
13	(10, 10)	(50, 10)	(70, 30)	(40, 10)	(30, 20)
14	(10, 20)	(20, 20)	(30, 40)	(40, 20)	(20, 0)
15	(0, 0)	(20, 20)	(20, 40)	(30, 20)	(20, 10)

Библиотека БГУИР

Тема №7. Атрибуты примитивов

Цель работы: изучить основные атрибуты графических примитивов; написать и отладить программу, реализующую эти возможности.

7.1. Теоретические сведения

В общем случае параметр, который влияет на способ изображения, называется *параметром атрибута*. Такие параметры атрибута, как цвет или размер, определяют фундаментальные характеристики примитива. Существует особый атрибут – видимость или регистрируемость определенного объекта в программе, связанной с интерактивным выбором объектов.

Рассмотрим атрибуты, которые предназначены для управления основными свойствами изображения графических примитивов.

Например, необходимо задать способы прорисовки линии: сплошная или пунктирная, широкая или тонкая, красная или пурпурная.

Один из способов – ввести в состав графического пакета опции атрибутов и таким образом дополнить список параметров, связанных с каждой функцией графического примитива, чтобы в нем содержались соответствующие значения атрибутов. Функция построения прямой линии, например, может содержать дополнительные параметры, определяющие цвет, ширину и другие свойства прямой.

Еще один подход – это сохранять системный список текущих значений атрибутов. В таких случаях в графический пакет включают отдельные функции для ввода текущих значений в список атрибутов. Чтобы создать примитив, система проверяет важные атрибуты и вызывает стандартную процедуру изображения этого примитива, используя при этом текущие значения атрибутов. В некоторых графических пакетах применяется комбинация методов ввода значений атрибутов, а в других библиотеках, в том числе *OpenGL*, атрибуты присваиваются с помощью отдельных функций, которые служат для обновления системного списка атрибутов.

Графическая система, в которой сохраняется список текущих значений атрибутов и других параметров, называется системой состояний или аппаратом состояний.

Атрибуты результирующих примитивов и некоторые другие параметры, такие как текущее положение буфера кадра, называются переменными состояниями или параметрами состояния. При присвоении значения одному или нескольким таким параметрам система вводится в определенное состояние, в котором остается до тех пор, пока значение параметров состояния не изменится.

Значения атрибутов и настройки других параметров задаются с помощью отдельных функций, которые описывают текущее состояние пакета *OpenGL*. К параметрам состояния в *OpenGL* относятся цвет и другие атрибуты примитивов,

Библиотека БГУИР

текущий матричный режим, элементы массива моделирования изображения, текущее положение в буфере кадра и параметры эффектов освещения сцены. Все параметры в *OpenGL* имеют свои значения по умолчанию, которые действуют до тех пор, пока не будут заданы новые значения. В любой момент можно потребовать от системы определить текущее значение любого параметра состояния.

Мы рассмотрим только настройки атрибутов результирующих примитивов.

Все графические примитивы в *OpenGL* изображаются с атрибутами из списка текущего состояния. Изменение одного или нескольких атрибутов повлияет только на те примитивы, которые будут задаваться после изменения состояния *OpenGL*.

Атрибуты примитивов, которые были описаны до смены состояния системы, не изменятся. Таким образом, можно изобразить зеленую линию, поменять текущий цвет на красный и нарисовать еще один отрезок. В этом случае получится изображение и зеленой линии, и красного отрезка. Кроме того, некоторые значения параметров состояния *OpenGL* могут задаваться внутри пары функций *glBegin*/*glEnd* вместе со значениями координат, так что настройки параметров могут меняться при переходе от одной точки к другой.

Основной атрибут всех примитивов – это цвет. Пользователю могут предлагаться различные цветовые опции в зависимости от возможностей и целей конкретной системы. Цветовые опции могут задаваться численно, выбираться из меню или с помощью ползунков. Относительно монитора эти коды цвета затем преобразуются в настройки уровня интенсивности электронного луча.

В цветных растровых системах количество возможных вариантов цветов зависит от объема памяти, который предоставляется для каждого пикселя в буфере кадра. Кроме того, информация о цвете может записываться в буфер кадра двумя способами: можно записывать коды *RGB*-цветов непосредственно в буфер кадра или же составлять отдельную таблицу из кодов цвета и использовать координаты пикселей для записи значений индексов, указывающих на содержимое цветовой таблицы. При прямой схеме записи, когда бы ни задавался определенный цветовой код в программе-приложении, информация об этом цвете будет помещаться в буфер кадра во всех положениях пикселей, составляющих примитив, который должен изображаться в этом цвете. Минимальное количество цветов при такой схеме можно получить при 3 битах памяти на один пиксель.

Каждое из трех положений битов используется для управления уровнем интенсивности соответствующей электронной пушки *RGB*-монитора.

Крайний слева бит служит для управления красной пушкой, средний бит управляет зеленой пушкой, а крайний справа бит – синей. Добавление большего числа битов на один пиксель увеличивает возможное количество доступных цветов. При 6 битах на один пиксель для каждой пушки можно использовать 2 бита. Это позволяет устанавливать четыре различных уровня интенсивности для каждой из трех цветовых пушек и обеспечивать для каждого пикселя на экране 64 цветовые опции. При увеличении количества предлагаемых опций увеличивается также и объем памяти, необходимый для буфера кадра. При разрешении

Библиотека БГУИР

1024×1024 пикселей для буфера кадра в полноцветной (24 бита на пиксель) *BOB*-системе необходимо 3 Мбайта памяти.

Цветовые таблицы – это альтернативный способ расширения возможностей пользователя, касающихся передачи цвета, при котором не требуется буфер кадра большого объема. Когда-то это было очень важным условием, но сегодня стоимость аппаратных средств резко снизилась, и широкие возможности передачи цветовой гаммы стали общедоступными даже в ранних моделях персональных компьютерных систем.

Иногда таблицы цветов называют таблицами видеопоиска. Теперь в качестве элементов цветовой таблицы используются значения, записанные в буфере кадра. Допустим, существует таблица цветов, где каждый пиксель может соотноситься с любым из 256 элементов таблицы, а каждая позиция таблицы состоит из 24 бит, которые описывают цвет в системе *RGB*.

Системы, в которых используется эта специальная таблица поиска, позволяют пользователю выбирать любые 256 цветов из палитры, содержащей почти 17 миллионов различных цветов. По сравнению с полноцветной системой количество цветов, которые можно изобразить одновременно, мало, но необходимый объем памяти буфера кадра также снижен до 1 Мбайта.

Иногда для работы со специализированными приложениями, в которых фигурирует закрашивание (например, схемы устранения неровности контура), создаются множественные таблицы цветов, также они используются в системах, имеющих более одного устройства цветного вывода.

Таблицы цветов могут оказаться полезными в ряде приложений, они могут предложить «разумное» количество одновременно изображаемых цветов, не требуя для этого большого объема буфера кадра. В большинстве приложений для одного рисунка достаточно 256 или 512 различных цветов. Кроме того, вход таблицы можно изменить в любое время, что позволяет пользователю легко экспериментировать с различными цветовыми комбинациями при проектировании, составлении сцены или построении графика, не изменяя структуру графических данных.

При изменении значения цвета в цветовой таблице все пиксели с этим цветовым индексом немедленно изменяют свой цвет на новый. Без цветовой таблицы цвет пикселя можно изменить, только записав новое цветовое значение в это же положение буфера кадра. Аналогично в приложениях, связанных с визуализацией данных, в буфер кадра можно записывать значения некоторой физической величины, например, энергии, и пользоваться поисковой таблицей для экспериментов с различными комбинациями цветов, не изменяя при этом значения пикселей.

В приложениях, связанных с визуализацией и обработкой изображений, таблицы цветов – это удобное средство для задания цветовых пороговых значений таким образом, чтобы все значения пикселей, которые превышают заданное пороговое значение (или меньше него), окрашивались в один и тот же цвет. По этим причинам в некоторых системах предлагаются обе возможности записи информации о цвете. Тогда пользователь может выбирать, использовать ли ему

Библиотека БГУИР

цветовые таблицы или записывать цветовые коды непосредственно в буфер кадра.

Сегодня возможность передачи цвета является стандартной возможностью систем компьютерной графики, для описания оттенков серого цвета (шкалы яркости или шкалы полутонов) используются цветовые функции системы *RGB*. Когда описание *RGB*-цвета содержит равное количество красного, зеленого и синего цветов, в результате получается некий оттенок серого цвета. Значения цветовых компонентов, близкие к 0, дают темно-серый цвет, а более высокие значения, близкие к 1.0, дают светло-серый цвет. Применение шкалы яркости позволяет улучшить качество черно-белых фотографий и создать визуальные эффекты.

Кроме системы *RGB* в различных приложениях компьютерной графики возможны и другие трехкомпонентные способы описания цвета. Например, цвет печати на принтере описывается с помощью голубого, пурпурного и желтого цветовых компонентов, а в цветных интерфейсах при выборе цвета иногда используется светлый или темный оттенок. Кроме того, цвет (как и свет вообще) – это сложный объект, и для описания различных свойств источников света и эффектов освещения в различных областях оптики, радиометрии и психологии было придумано много терминов и понятий.

С физической точки зрения цвет можно описать как электромагнитное излучение в определенном частотном диапазоне и с определенным распределением энергии, но существуют также характеристики, описывающие особенности нашего восприятия цвета. Таким образом, мы пользуемся физическим термином «интенсивность» для измерения количества световой энергии, излучаемой в определенном направлении за период времени, а психологический термин «яркость» используется для характеристики воспринимаемой глазом светимости.

Функция определения цветового режима изображения *RGB* записывается как

glutInitDisplayMode (GLUT_SINGLE I GLUT_RGB);

Первый параметр из списка аргументов сообщает, что для буфера кадра используется один буфер, а второй параметр устанавливает режим *RGB* (или *RGBA*) (цветовой режимом по умолчанию). Для выбора этого режима можно воспользоваться также операторами *GLUT_RGB* или *GLUT_RGBA*. Если требуется описать цвет с помощью элементов таблицы цветов, нужно заменить константу *OpenGL GLUT_RGB* на *GLUT_INDEX*.

Большинство настроек цветов примитивов *OpenGL* выполняется в режиме *RGB*, который, по сути, не отличается от режима *RGBA*. Единственное различие между режимами *RGB* и *RGBA* состоит в том, используется ли для смешивания цветов значение альфа. Когда задается определенный набор цветовых значений примитивов, определяется цветовое состояние пакета *OpenGL*. Текущий цвет будет применяться ко всем задаваемым после этого примитивам до тех пор, пока настройки цветов не изменятся. Новое описание цвета повлияет только на те объекты, которые будут описываться после изменения цвета.

Библиотека БГУИР

В режиме *RGB* задаются значения красного, зеленого и синего цветовых компонентов. Четвертый цветовой параметр, коэффициент альфа не обязателен, а четырехмерное описание цвета называется *RGBA*-цветом.

Этот четвертый цветовой параметр можно использовать для смешивания цветов при наложении объектов друг на друга. Одна из важных областей применения параметра смешивания цветов – это моделирование эффектов прозрачности.

Для таких расчетов значение альфа соответствует настройке прозрачности (или непрозрачности). В режиме *RGB* (или *RGBA*) текущие цветовые компоненты выбираются с помощью функции

glColor (colorComponents);*

Индексы аналогичны используемым для функции *glVertex*. Для выбора режима *RGB* или *RGBA* используются индексы 3 или 4 плюс код типа числовых данных и необязательный индекс, обозначающий вектор. Возможны такие индексы, обозначающие тип числовых данных: *b*, *i*, *s*, *f* и *d*, а также числовые значения без знака. Значения цветовых компонентов с плавающей запятой принадлежат диапазону от 0.0 до 1.0. По умолчанию цветовые компоненты для оператора *glColor*, включая значение альфа, – это (1.0, 1.0, 1.0, 1.0), задающие белый цвет *RGB* и значения альфа, –1.0.

В пакете *OpenGL* цвета можно выбирать и для отдельных точек, используя для этого пару *glBegin*/*glEnd*.

Описание цветовых компонентов с помощью целых чисел зависит от возможностей системы. Для полноцветной системы, в которой выделяется 8 бит на пиксель (256 уровней для каждого цветового компонента), целочисленные значения цветовых компонентов принадлежат диапазону от 0 до 255. Тогда соответствующие значения цветовых компонентов с плавающей запятой будут

$$0.0;\ 1.0 / 255.0;\ 2.0 / 255.0;\ \ 255.0 / 255.0 = 1.0.$$

Для полноцветной системы голубой цвет можно задать с помощью целочисленных значений цветовых компонентов так:

glColor3i (0, 255, 255);

Описание цвета в *OpenGL* может также задаваться в индексном цветовом режиме, в котором даются ссылки на значения элементов цветовой таблицы. В этом режиме текущий цвет устанавливается путем задания индекса цветовой таблицы:

glIndex (colorIndex);*

Параметру *colorIndex* присваивается неотрицательное целое значение. Затем это значение индекса записывается в позиции буфера кадра для последовательно описываемых примитивов. Цветовой индекс можно задавать с любым из следующих типов данных: байтовый без знака, целый или с плавающей запятой. Тип данных параметра *colorIndex* обозначается с помощью индекса *ub*, *s*, *i*, *d* или *f*, а количество индексов цветовой таблицы всегда равно какому-либо числу в степени 2, например, 256 или 1024. Количество битов, приходящееся на каждую позицию таблицы, зависит от аппаратных особенностей системы. Всем примитивам, которые будут описываться после этого оператора, будет присваиваться

Библиотека БГУИР

значение цвета, записанное в этом месте цветовой таблицы, пока текущий цвет не изменится.

В корневой библиотеке *OpenGL* нет функций для занесения значений в цветовую поисковую таблицу, поскольку стандартные процедуры для обработки таблиц являются частью системы окон. Кроме того, некоторые системы окон поддерживают несколько таблиц цветов, тогда как в других системах есть только одна, и выбор цветов ограничен.

Существует стандартная процедура библиотеки *GLUT*, которая взаимодействует с системой окон и заносит описания цветов в таблицу в позицию с заданным индексом.

glutSetColor (index, RED, GREEN, BLUE);

Цветовым параметрам *RED*, *GREEN* и *BLUE* присваиваются значения с плавающей запятой в диапазоне от 0.0 до 1.0. Затем этот цвет заносится в таблицу в позицию, заданную с помощью параметра *index*.

Стандартные процедуры для обработки трех других цветовых таблиц предлагаются как дополнение к корневой библиотеке *OpenGL*. Эти стандартные процедуры являются частью набора для создания изображения *OpenGL*. Цветовые значения, которые хранятся в этих таблицах, можно использовать для изменения значений пикселей при их обработке в различных буферах.

Несколько примеров использования этих таблиц – установка эффекта фокусировки камеры, отфильтровка определенных цветов на изображении, увеличение определенных интенсивностей или регулировка уровня яркости, преобразование черно-белых фотографий в цветные и устранение контурных неровностей изображения. Кроме того, эти таблицы можно использовать для изменения цветовых моделей, т. е. можно поменять цвета *RGB* на другое описание с помощью трех других «основных» цветов (голубого, пурпурного и желтого).

Отдельная цветовая таблица из набора для создания изображений в *OpenGL* активизируется с помощью функции *glEnable* при использовании одного из трех названий таблиц: *GL_POST_COLOR_MATRIX__COLOR_TABLE*, *GL_COLOR_TABLE* или *GL__POST__CONVOLUTION__COLOR_TABLE*. Затем можно воспользоваться стандартными процедурами из набора для создания изображений и выбрать определенную таблицу цветов, задать значения таблицы цветов, скопировать значения из таблицы или указать, какой компонент цвета пикселя требуется изменить и как именно это нужно сделать.

Во многих приложениях было бы удобно сочетать цвета накладывающийся друг на друга объектов или смешивать цвет объекта с цветом. Так достигаются имитация эффекта мазка кисти, формирование сложного изображения, состоящего из нескольких рисунков, моделирование прозрачности и устранение неровностей контуров объектов сцены. В большей части графических пакетов предлагаются различные способы создания разных эффектов смешивания цветов. Эти процедуры называются функциями смешивания цветов или функциями составления изображений. В пакете *OpenGL* цвета двух объектов можно смешать, сначала загрузив в буфер кадра один объект, а затем объединив цвет второго объекта с цветом из буфера кадра.

Библиотека БГУИР

Текущий цвет в буфере кадра называется в *OpenGL* цветом назначения, а цвет второго объекта – цветом источника. Смешивание можно выполнять только в режиме *RGB* или *RGBA*. Чтобы применить смешивание цветов в приложении, сначала нужно активизировать эту возможность пакета *OpenGL* с помощью такой функции:

glEnable (GL_BLEND);

Чтобы отключить стандартные процедуры смешивания цветов, в *OpenGL* используется функция

glDisable (GL_BLEND);

Если возможность смешивания цветов не активизирована, цвет объекта просто заменит цвет, записанный в буфере кадра в положении этого объекта.

Цвета можно смешивать несколькими различными способами, в зависимости от эффекта, которого требуется достичь, а различные цветовые эффекты возникают при задании двух наборов коэффициентов смешивания.

Один набор коэффициентов смешивания задается для текущего объекта в буфере кадра («объект назначения»), а второй набор коэффициентов – для нового объекта («источник»). Новый смешанный цвет, который затем загружается в буфер кадра, находится как

$$(S_r R_s + D_r R_d, S_g G_s + D_g G_d, S_b B_s + D_b B_d, S_a A_s + D_a A_d),$$

где цветовые компоненты *RGBA* источника – (R_s, G_s, B_s, A_s), цветовые компоненты назначения – (R_d, G_d, B_d, A_d) ,коэффициенты смешивания источника – (S_r, S_g, S_b, S_a), коэффициенты смешивания назначения – (D_r, D_g, D_b, D_a).

Значения коэффициентов смешивания выбираются с помощью следующей функции *OpenGL*:

glBlendFunc (sFactor, dFactor);

Каждому из параметров *sFactor* и *dFactor*, т. е. коэффициентам источника и назначения, присваиваются символьные константы *OpenGL*, задающие предопределенный набор из четырех коэффициентов смешивания.

Например, константа *GL_ZERO* дает коэффициенты смешивания (0.0, 0.0, 0.0, 0.0), а константа *GL_ONE* дает набор (1.0, 1.0, 1.0, 1.0). Можно присвоить всем четырем коэффициентам смешивания либо значение альфа-назначения, либо значение альфа-источника, что делается с помощью константы *GL_DST_ALPHA* или *GL_SRC_ALPHA*.

К числу остальных констант *OpenGL*, с помощью которых задаются коэффициенты смешивания, относятся *GL_ONE_MINUS_DST_ALPHA*, *GL_ONE_MINUS_SRC_ALPHA*, *GL_DST__COLOR* и *GL_SRC_COLOR*. Эти коэффициенты смешивания часто используются для моделирования прозрачности. По умолчанию параметру *sFactor* присваивается значение *GL_ONE*, а значение параметра *dFactor* по умолчанию равно *GL_ZERO*. Следовательно, по умолчанию значения коэффициентов смешивания приводят к тому, что новые цветовые значения заменяют текущие значения в буфере кадра.

Библиотека БГУИР

В библиотеку *GLUT* пакета *OpenGL* включены дополнительные функции, в частности, процедура установки смешанного цвета и процедура задания уравнения смешивания.

Цветовые значения для описания сцены можно также задавать вместе с координатными значениями в массиве вершин. Это можно сделать либо в режиме *RGB*, либо в индексном цветовом режиме. Как и для массива вершин, сначала нужно активизировать возможность создания цветового массива *OpenGL*:

glEnableClientState (GL_COLOR__ARRAY);

Затем для цветового режима *RGB* задается положение и формат цветовых компонентов с помощью функции

glColorPointer (nColorComponents, dataType, offset, colorArray);

Параметру *nColorComponents* присваивается значение 3 или 4 в зависимости от того, заносятся ли в массив *colorArray* цветовые компоненты *RGB* или *RGBA*. Символьная константа *OpenGL GL_INT* или *GL_FLOAT* присваивается параметру *dataType* и указывает на тип данных цветовых значений. Для отдельного цветового массива параметру *offset* можно присвоить значение 0. Однако, если в одном массиве комбинируется информация о цвете с информацией о вершинах, значением параметра *offset* должно быть количество битов между каждым набором цветовых компонентов в массиве.

```
typedef GLint vertex3 [3], color3 [3];
vertex3 pt [8] = 0, 0, 0, 0, 1, 0, 1, 0, 0,
1, l, 0, 0, 0, 1, 0, l, 1, l, 0, 1, 1, 1, 1
color3 hue [8] = 1, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0,1,1, 0, 0,1, 0, 0, 0, 0,1, 0, 0,1;
glEnableClientState (GL_VERTEX_ARRAY);
glEnableClientState (GL_COLOR_ARRAY);
glVertexPointer (3, GL_INT, 0, pt);
glColorPointer (3, GL_INT, O, hue);
static GLint hueAndPt [ ] =
1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1;
glVertexPointer (3, GL_INT, 6*sizeof(GLint), hueAndPt [3]);
glColorPointer (3, GL_INT, 6*sizeof(GLint), hueAndPt [0]);
```

Первые три элемента этого массива задают цветовое значение *RGB*, следующие три значения – набор координат вершины (*x*, *y*, *z*), и такой порядок сохраняется до последнего описания цвета и вершины. Параметр *offset* устанавливается равным количеству байтов между соседними цветовыми значениями или значениями координат вершин, которое в обоих случаях равно 6* *sizeof* (*GLint*).

Библиотека БГУИР

Цветовые значения начинаются с первого элемента объединенного массива, который равен *hueAndPt* [0], а значения координат вершин начинаются с четвертого элемента, который равен *hueAndPt* [3].

Поскольку сцена, как правило, состоит из нескольких объектов, у каждого из которых есть множество плоских поверхностей, в пакете *OpenGL* предлагается функция, определяющая сразу все массивы вершин и цветов, а также другие виды информации. Если заменить цветовые значения и координаты вершин из вышеприведенного примера на значения с плавающей запятой, то можно воспользоваться этой функцией:

glInterleavedArrays (GL_C3F_V3F, O, hueAndPt);

Первый параметр – это константа *OpenGL*, которая указывает на то, что и цвет (*C*), и координаты вершин (*V*) описываются с помощью трехкомпонентных значений с плавающей запятой. Элементы массива *hueAndPt* должны объединяться с цветовым значением для каждой вершины из списка перечисленных перед этим координат. Кроме того, эта функция автоматически активизирует как цветовые массивы, так и массивы координат.

В индексном цветовом режиме массив цветовых индексов задается с помощью функции

glIndexPointer (type, stride, colorIndex);

Цветовые индексы перечисляются в массиве *colorIndex*, а параметры *type* и *stride* такие же, как и в функции *glColorPointer*. Параметр *size* не нужен, поскольку индексы цветовой таблицы описываются с помощью одного значения.

В индексном цветовом режиме массив цветовых индексов задается с помощью функции

glIndexPointer (type, stride, colorIndex);

Цветовые индексы перечисляются в массиве *colorIndex*, а параметры *type* и *stride* такие же, как и в функции *glColorPointer*. Параметр *size* не нужен, поскольку индексы цветовой таблицы описываются с помощью одного значения.

Функция, которая служит для выбора цветовых компонентов *RGB* для окна изображения,

glClearColor (red, green, blue, alpha);

Каждому цветовому компоненту в этом обозначении (красному, зеленому и синему), а также параметру альфа присваивается значение с плавающей запятой в диапазоне от 0.0 до 1.0. По умолчанию значения всех четырех параметров равны 0.0, что дает черный цвет. Если каждому цветовому компоненту присвоить значение 1.0, то цветом чистого окна изображения будет белый. Оттенки серого цвета можно получить при одинаковых значениях всех цветовых компонентов в диапазоне от 0.0 до 1.0. Четвертый параметр *alpha* представляет собой опцию для смешивания предыдущего цвета с текущим цветом. Смешивание происходит только в том случае, если активизирована возможность смешивания *OpenGL*; смешивание цветов невозможно, если значения задаются в виде цветовой таблицы.

В пакете *OpenGL* существует несколько буферов цвета, которые можно использовать в качестве текущего буфера регенерации при изображении сцены,

Библиотека БГУИР

а функция *glClearColor* служит для спецификации цвета во всех цветовых буферах. После этого к цветовым буферам применяется операция закрашивания цветом чистого окна изображения, что выполняется с помощью команды

glClear (GL_COLOR_BUFFER_BIT);

С помощью функции *glClear* можно также задать исходные значения других буферов, существующих в *OpenGL*:

1) буфер накопления, в котором хранится информация о смешанных цветах;

2) буфер глубины, содержащий значения глубины (расстояния до точки наблюдения) объектов сцены;

3) буфер шаблонов, в котором хранятся данные, определяющие границы рисунка.

В индексном цветовом режиме для задания цвета окна изображения используется следующая функция (вместо *glClearColor*):

glClearIndex (index);

После выполнения этой команды цвету фона присваивается цвет, который записан в позиции *index* в цветовой таблице. При вызове функции *glClear* (*GL_COLOR_BUFFER_BIT*) окно изображается именно в этом цвете.

В библиотеке *OpenGL* существует множество других функций для задания цвета, подходящих для выполнения различных заданий: изменения цветовых моделей, настройки эффектов освещения сцены, описания эффектов камеры и закрашивания поверхностей объекта.

В основном задаются два атрибута точек: цвет и размер. В системе состояний цвет и размер изображаемой точки определяются по текущим значениям, записанным в списке атрибутов. Цветовые компоненты задаются с помощью значений *RGB* или как элементы цветовой таблицы. В растровых системах размер точки – это целое число, кратное размеру пикселя, так что большие точки изображаются как квадратные блоки пикселей.

Прямолинейный отрезок можно изобразить с помощью трех основных атрибутов: цвета, ширины и стиля. Цвет прямой линии, как правило, задается с помощью той же самой функции, что и для всех графических примитивов, тогда как ширина и стиль линии выбираются с помощью отдельных функций для прямых линий. Кроме того, прямые линии могут изображаться с использованием таких дополнительных эффектов, как мазки кисти или пера.

Реализация опций ширины линии зависит от возможностей устройства вывода. Широкие линии на мониторе могут изображаться как расположенные рядом друг с другом параллельные линии.

В растровых системах линии стандартной ширины строятся путем использования одного пикселя в каждой точке выборки, как в алгоритме Брезенхема. Более широкие линии изображаются кратными положительному целому числу стандартных линий, для чего добавляются дополнительные пиксели на соседних параллельных линиях. Если тангенс угла наклона прямой линии меньше или равен 1, стандартную процедуру для построения прямой линии можно изме-

Библиотека БГУИР

нить таким образом, чтобы широкие линии строились как изображения небольших вертикальных полос пикселей в каждом столбце (координата *x*) на траектории прямой. Количество пикселей, которое следует изображать в каждом столбце, устанавливается равным целому значению ширины линии.

Если тангенс угла наклона прямой линии превышает 1, широкие линии можно изображать с помощью горизонтальных полос, поочередно прибавляя пиксели справа и слева от траектории прямой.

Чтобы реализовать эту процедуру, следует сравнить величины горизонтальных и вертикальных расстояний (∂x и ∂y) между концами отрезка прямой. Если *fabs* (∂x) $\geq$ *fabs* (∂y), то дополнительные пиксели изображаются в столбцах. В противном случае дополнительные пиксели изображаются в строках.

Несмотря на то что широкие линии можно быстро построить, откладывая вертикальные или горизонтальные полосы пикселей, ширина изображаемой линии, измеренная перпендикулярно к направлению прямой, зависит от тангенса угла ее наклона.

Линия с углом наклона 45° будет в 1/*sqrt*(2) раз тоньше по сравнению с горизонтальной или вертикальной прямой, построенной с помощью полос пикселей такой же длины.

Еще одна проблема, связанная с выбором ширины с помощью горизонтальных или вертикальных полос пикселей, состоит в том, что этот метод дает линии, концы которых вертикальны или горизонтальны, независимо от угла наклона самой прямой.

Это особенно заметно при использовании очень широких линий.

Форму концов линий можно подправить, чтобы они выглядели лучше, прибавив к ним наконечники.

Один из видов наконечников для прямой линии – это стыковое перекрытие, для которого характерны прямоугольные края, перпендикулярные к направлению прямой. Если тангенс угла наклона заданной прямой равен m, то тангенс угла наклона прямоугольных концов широкой линии будет $-1/m$. Тогда на каждом конце заданной прямой каждая из составляющих ее параллельных линий будет изображаться между двумя перпендикулярными линиями.

Еще один наконечник прямой линии — это овальное перекрытие, которое получается путем прибавления закрашенного полукруга к каждому стыковому перекрытию. Центры этих полукругов находятся в середине широкой линии, а их диаметр равен ширине линии.

Третий вид наконечников – это проекционное прямоугольное перекрытие. В этом случае прямая просто продолжается, и к ней прибавляются стыковые перекрытия, которые расположены на расстоянии половины ширины линии от заданных концов отрезка.

К числу других способов создания широких линий можно отнести изображение прямой линии в виде закрашенного прямоугольника или создание линии с помощью выбранной модели пера или кисти.

Библиотека БГУИР

Чтобы получить описание границ прямой в виде прямоугольника, рассчитываются координаты вершин прямоугольника, находящихся на перпендикулярах к направлению прямой, чтобы координаты вершин прямоугольника были удалены от точки конца заданной прямой на расстояние, равное половине ширины прямой. К закрашенному прямоугольнику можно добавить овальные перекрытия или увеличить длину прямой таким образом, чтобы получились проекционные прямоугольные перекрытия.

Для создания широких ломаных линий необходимы дополнительные действия. В общем случае с помощью рассмотренных методов построения отдельного отрезка прямой нельзя построить серию прямолинейных отрезков, непрерывно соединяющихся друг с другом. При изображении широких ломаных линий с помощью горизонтальных и вертикальных полос пикселей, например, остаются промежутки на границах между двумя отрезками с различными углами наклона, особенно в местах перехода от горизонтальных полос пикселей к вертикальным. Широкую ломаную линию, части которой непрерывно соединяются друг с другом, можно построить за счет дополнительной обработки концов отдельных отрезков.

К возможным вариантам выбора атрибута стиля прямой линии можно отнести сплошные линии и различные виды пунктира. Алгоритм построения прямой линии можно изменить таким образом, чтобы с его помощью можно было создавать линии, задавая длину изображаемых сплошных участков, расположенных на траектории прямой линии, и расстояние между ними. Во многих графических пакетах можно выбирать как длину самих штрихов пунктира, так и расстояние между ними.

В растровых алгоритмах изображения прямых линий атрибуты стиля линии изображаются путем нанесения полос пикселей. Для создания штрихованных, пунктирных и штрихпунктирных узоров процедура построения прямой линии выдает участки следующих друг за другом пикселей, расположенных на прямой линии, перескакивая через несколько промежуточных пикселей между сплошными полосами.

Количество пикселей, определяющее длину полосы и расстояние между отдельными полосами, задается с помощью пиксельной маски, которая представляет собой узор из двоичных чисел, сообщающий, какие точки, лежащие на прямой, следует изображать. Линейная маска 11111000, например, может быть использована для изображения пунктирной линии с длиной штриха в пять пикселей и расстоянием между штрихами в три пикселя. Пиксели, координатам которых соответствуют биты со значением 1, изображаются в текущем цвете, а пиксели, координатам которых соответствуют биты со значением 0, изображаются цветом фона.

Параметры атрибутов кривых линий не отличаются от параметров отрезков прямых. Изображаемые кривые могут быть различных цветов, ширин, могут иметь различные штрихпунктирные узоры, доступны также и возможности ри-

Библиотека БГУИР

сования пером или кистью. Методы адаптации алгоритмов для построения кривых линий и возможности выбора атрибутов такие же, как и при построении прямых линий.

Растровые кривые различной ширины можно изображать с помощью метода вертикальных или горизонтальных полос пикселей. Там, где величина тангенса угла наклона кривой меньше или равна 1.0, откладываются вертикальные полосы; там, где эта величина превышает 1.0, откладываются горизонтальные полосы.

Еще один способ изображения широких кривых – закрасить область между двумя параллельными кривыми, расстояние между которыми равно требуемой ширине линии. Это можно сделать, воспользовавшись заданной кривой в качестве одной границы и проведя вторую границу либо внутри, либо снаружи исходной траектории кривой. Однако при таком подходе исходное положение кривой смещается либо внутрь, либо наружу в зависимости от того, какое направление выбрано для второй границы. Исходное положение кривой можно сохранить, задав две границы кривой на расстоянии, равном половине ширины этой кривой, в каждую сторону от заданной траектории кривой. Хотя данный метод позволяет точно изобразить широкие окружности, площадь других широких кривых в общем случае он передает только приблизительно. Например, фокусы внешней и внутренней границ широкого эллипса, построенного таким способом, не совпадают.

Пиксельные маски, которые применяются для реализации прямых линий, можно использовать и в растровых алгоритмах для построения кривых и создавать с их помощью штрихованные или пунктирные узоры. Кроме того, в алгоритмах построения прямолинейных отрезков с помощью пиксельных масок изображают штрихи и промежутки между ними, имеющие различную длину вне зависимости от наклона кривой. Если требуется изобразить штрихи одинаковой длины, нужно при перемещении по окружности подбирать количество пикселей в каждом штрихе. Чтобы получить штрихи одинаковой длины, не применяя пиксельную маску с равными полосами, можно наносить пиксели через одинаковые угловые расстояния.

Цвет изображаемой точки зависит от текущих цветовых значений в списке состояний. А задается цвет либо с помощью функции *glColor*, либо с помощью функции *glIndex*.

Размер точки в *OpenGL* задается командой

glPointSize (size);

Точка изображается как квадратный блок пикселей. Параметру *size* присваивается положительное значение с плавающей запятой, которое округляется до целого числа (если только не нужно устранять неровности краев точки). Параметром *size* определяется количество горизонтальных и вертикальных пикселей, составляющих изображение точки. Таким образом, при размере точки 1.0 изображается один пиксель, а при размере точки 2.0 – массив пикселей 2×2. Если

Библиотека БГУИР

активизирована способность пакета *OpenGL* сглаживать неровности краев объектов, то размер изображаемого блока пикселей будет соответствующим образом меняться. По умолчанию размер точки равен 1.0.

Функции атрибутов могут перечисляться внутри пары *glBegin*/*glEnd* и вне ее.

Например, следующий фрагмент программы изображает три точки разного цвета и размера. Первая точка имеет стандартный размер и красный цвет, вторая – двойной размер и зеленый цвет, третья – синяя тройного размера.

```
glColor3f (1.0, 0.0, 0.0);
glBegin (GLflPOINTS);
glVertex2i (50, 100);
glPointSize (2.0);
glColor3f (0.0, 1.0, 0.0);
glVertex2i (75, 150);
glPointSize (3.0);
glColor3f (0.0, 0.0, 1.0);
glVertex2i (100, 200);
glEnd ( );
```

В OpenGL ширина линии задается с помощью функции

glLineWidth (width);

Параметру *width* присваивается значение с плавающей запятой, которое затем округляется до ближайшего неотрицательного целого числа. Если входное значение округляется до 0.0, изображаемая линия будет иметь стандартную ширину 1.0 (по умолчанию). Однако, если к данной линии применить операцию сглаживания, ее края изменятся, чтобы компенсировать растровый «эффект зубцов», и тогда возможны дробные значения ширины. Отметим, что одни версии функции поддерживают только ограниченное число ширин, а другие не поддерживают ширины, отличные от 1.0. Между собой сравниваются величины горизонтального и вертикального расстояний между концами прямолинейного отрезка $(\partial x, \partial y)$, и определяется, как следует изображать широкую линию – с помощью вертикальных или горизонтальных полос пикселей.

По умолчанию прямолинейный отрезок изображается в виде сплошной линии. В то же время можно изображать штрихованные, пунктирные линии, а также линии, состоящие из различных комбинаций точек и штрихов. Кроме того, можно варьировать длину штрихов и расстояние между точками или штрихами. Текущий стиль изображения линий задается с помощью следующей функции *OpenGL*:

glLineStipple (repeatFactor, pattern);

Параметр *pattern* (шестнадцатеричное целое число) указывает, как должна изображаться та или иная линия. В этом шаблоне 1 означает положение пикселя «включено», а 0 – «выключено».

Библиотека БГУИР

Данный шаблон применяется к пикселям, расположенным на прямой, начиная с младших разрядов шаблона. По умолчанию шаблон имеет вид *OxFFFF* (в каждом положении стоит значение 1), что соответствует сплошной линии. Целочисленный параметр *repeatFactor* сообщает, сколько раз должен повторяться каждый разряд в шаблоне перед применением следующего разряда. По умолчанию число повторов равно 1.

Для ломаной линии заданный узор стиля линии не возобновляется в начале каждого отрезка. Он непрерывно продолжается на протяжении всех отрезков, начинаясь в первой точке ломаной линии и заканчиваясь в последней точке последнего отрезка данной серии.

Перед изображением линии с помощью текущего шаблона нужно активизировать возможность *OpenGL* построения линий различного стиля. Для этого в программу вводится следующая команда:

glEnable (GL_LINE_STIPPLE);

Если мы забудем включить эту функцию активизации, то будут изображаться только сплошные линии; т. е. для изображения прямолинейных отрезков используется шаблон по умолчанию *OxFFFF*. В любой момент возможность построения линий различного стиля можно отключить, воспользовавшись такой функцией:

glDisable (GL_LINE_STIPPLE);

Указанная функция меняет текущий шаблон стиля линии на значение, используемое по умолчанию (сплошные линии):

```
typedef struct { float x, y;} wcPtZD;
wcPtZD dataPts [5];
void linePlot (wcPt2D dataPts [5])
{
            int k;
            glBegin (GL_LINE__STRIP)
             for (k = 0; k < 5; k++)
                    glVertex2f (dataPts[k].x,dataPts [k].y);
             glFlush ( );
            glEnd ( );
}
/*Обращение к процедуре изображения координатных осей.*/
glEnable (GL_LINE_STIPPLE);
/*Вводится первый набор значений данных (x, y).*/
glLineStipple (1, ox1C47);
/*Строится штрихпунктирная ломаная линия стандартной ширины.*/
linePlot (dataPts);
/*Вводится второй набор значений данных (x, y).*/
glLineStipple (1, 0x00FF);
/*Строится штрихованная ломаная линия двойной ширины.*/
glLineWidth (2.0); linePlot (dataPts);
```


Библиотека БГУИР

```
/*Вводится третий набор значений данных (к, у).*/
glLineStipple (1, 0x0101);
/*Строится пунктирная ломаная линия тройной ширины.*/
glLineWidth (3.0); linePlot (dataPts);
glDzzisable (GL_LINE_STIPPLE);
```

OpenGL позволяет не только описывать ширину, стиль и цвет всей линии, но и задавать линии с градацией цветовых оттенков. Например, цвет сплошной линии можно изменять, присваивая при определении прямой линии различные цветовые значения разным концам отрезка. В следующем фрагменте программы эта возможность иллюстрируется следующим образом: на одном конце линии задается синий цвет, а на другом – красный. Затем сплошная линия изображается с помощью линейной интерполяции кодов заданных концов линии:

```
glShadeModel (GL_SMOOTH);
glBegin (GL_LINES);
      glColor3f (0.0, 0.0, 1.0);
      glVertex2i (50, 50);
      glColor3f (1.0, 0.0, 0.0);
      glVertex2i (250, 250);
glEnd ();
```

Аргументом функции *glShadeModel* может быть *GL_FLAT*. В этом случае прямолинейный отрезок изображается одним цветом – цветом второго конца (250, 250). Следовательно, получается красная линия.

В действительности аргумент *GL_SMOOTH* задается по умолчанию, поэтому отрезок с непрерывной градацией цвета получается даже тогда, когда в программу эта функция не включена.

Чтобы получить другие эффекты, соседние линии изображаются с помощью различных цветов и шаблонов. Кроме того, можно воспользоваться возможностью смешивания цветов, накладывая друг на друга линии и другие объекты с различными значениями альфа.

Мазки кисти или другие художественные эффекты моделируются с помощью пиксельного массива и смешивания цветов. Кроме того, пиксельный массив можно перемещать интерактивно и таким образом строить прямолинейные отрезки. Отдельным пикселям из пиксельного массива присваиваются различные значения альфа, в результате чего линии выглядят как нарисованные с помощью кисти или пера.

В большинстве графических пакетов закрашенные фигуры ограничиваются многоугольниками, которые можно описать с помощью линейных уравнений. Возможно и более жесткое условие – все закрашенные фигуры должны быть выпуклыми многоугольниками (чтобы строки развертки не пересекались более чем с двумя сторонами границ). Однако в общем случае закрашивать можно лю-

Библиотека БГУИР

бые заданные участки, в том числе окружности, эллипсы и другие объекты с криволинейными границами. Кроме того, такие приложения, как программы рисования, предлагают опции для закрашивания областей произвольной формы.

Существует две процедуры закрашивания участков рисунка в растровых системах, если описание закрашенной области переведено в координаты пикселей. В одной процедуре вначале определяются интервалы перекрытия для строк развертки, которые пересекают эту область. Затем пикселям, попадающим в эти интервалы перекрытия, присваиваются значения, соответствующие узору заполнения. Другой способ заполнения участков рисунка – начать с заданной точки, находящейся внутри этой области, и «зарисовывать пространство» наружу от этой точки, пока не дойдем до того места, где выполняются заданные граничные условия. Метод строк развертки, как правило, применяется для таких простых фигур, как окружности или области с ломаными границами. Также этот метод используется в универсальных графических пакетах. Алгоритмы закрашивания, в которых применяется начальная внутренняя точка, полезны при закрашивании участков с более сложными границами, а также в интерактивных системах рисования.

Основной атрибут закрашенных фигур, предлагаемый в универсальной графической библиотеке, – это стиль изображения внутренней области. Данную область можно заполнить одним цветом или заданным узором, или же оставить ее «чистой», показав только границы фигуры. Кроме того, можно закрашивать выбранные участки сцены с помощью различных видов кистей, комбинаций смешивания цветов или текстур. К числу других опций относятся описания границ закрашенных фигур. Границы многоугольников можно показать другим цветом, они могут иметь разную ширину и выполняться разным стилем. Кроме того, можно выбирать различные атрибуты изображения передней и задней сторон многоугольной области.

Узоры заполнения могут задаваться в форме прямоугольных цветовых массивов, содержащих различные цвета для различных элементов массива, или битовых массивов, в которых указывается, какие относительные положения следует изображать одним выбранным цветом. Массив, с помощью которого описывается узор заполнения, называется маской, и он применяется к области изображения. В некоторых графических системах возможен выбор произвольного исходного положения, с которого маска многократно повторяется в горизонтальном и вертикальном направлениях до тех пор, пока вся область изображения не будет заполнена неперекрывающимися копиями узора.

В тех местах, где на заданные закрашенные участки накладывается узор, из массива узора определяется, какие пиксели следует изображать определенным цветом. Данный процесс заполнения области с помощью прямоугольного шаблона называется клеточным заполнением, а прямоугольный узор заполнения иногда называют ячейкой узора. Система может содержать предопределенные узоры заполнения.

Библиотека БГУИР

Чтобы заполнить объект узором, нужно определить места, где узор накладывается на строки развертки, которые пересекают закрашенную область. Начиная с заданного стартового положения, прямоугольные шаблоны накладываются вертикально по строкам развертки и горизонтально по положениям пикселей в строках развертки.

Каждое повторение массива узора выполняется через промежутки, зависящие от ширины и высоты маски. В тех местах, где узор накладывается на закрашенную область, цвет пикселей устанавливается согласно значениям, записанным в маске.

Чтобы заполнить отдельные области штриховкой, нужно изобразить набор прямолинейных отрезков, после чего получается либо одинарная, либо перекрестная штриховка. Расстояние между линиями штриховки и их наклон можно задавать как параметры таблицы штриховки. Кроме того, узор штриховки можно задать в виде массива узора – набора диагональных прямых линий.

Опорная точка (xp, yp) стартового положения узора заполнения выбирается в любом удобном месте (внутри или за пределами закрашиваемой области). Например, в качестве опорной точки можно выбрать вершину многоугольника или же задать эту точку в левом нижнем углу рабочей области (ограничивающего окна), которая определяется координатными границами данной фигуры. Для упрощения выбора опорных координат в некоторых пакетах в качестве стартового положения всегда выбирается начало координат окна изображения. Более того, постоянное расположение точки (xp, yp) в начале координат упрощает операции клеточного представления, когда каждый элемент узора переносится в отдельный пиксель. Например, если строки в массиве узора нумеруются снизу вверх, начиная со значения 1, то пикселю с экранными координатами (x, y) присваивается цветовой код точки узора ($y \bmod ny + 1$, $x \bmod nx + 1$).

Здесь через ny и nx обозначено количество строк и количество столбцов массива узора. В то же время размещение стартового положения узора в начале координат более эффективно при заполнении узором фона экрана, а не отдельных закрашенных областей. Между соседними или перекрывающимися областями, заполненными одинаковым узором, не будет видна граница, разделяющая эти участки. К тому же перемещение и повторное закрашивание объекта тем же самым узором может привести к смещению соответствующих значений пикселей внутри объекта.

Движущийся объект будет казаться прозрачным телом, передвигающимся относительно стационарного узора фона, а не перемещающимся вместе с узором, которым заполнена его внутренняя область.

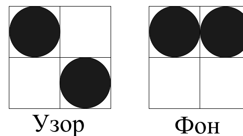
Некоторые методы закрашивания с использованием возможности смешивания цветов называются алгоритмами мягкого или бледного окрашивания. Одна из областей применения этих методов заполнения – смягчение цвета заполнения на границах объектов, размытых при сглаживании краев. Другая область применения – возможность повторного закрашивания цветного участка, который изначально был закрашен с помощью полупрозрачной кисти (в таком случае те-

Библиотека БГУИР

кущим цветом будет смесь цвета кисти и цвета фона «за» этим участком). В любом случае хотелось бы, чтобы новый цвет заполнения изменялся в пределах данной области точно так же, как и текущий цвет заполнения.

Существует несколько способов сочетания узора заполнения объектов с цветом фона.

Объединять узор с цветом фона можно с помощью коэффициента прозрачности, показывающего, насколько фон должен смешиваться с цветом объекта. Кроме того, можно воспользоваться простыми логическими операциями или операциями замены.



7.2. Индивидуальные задания

Изучив предложенные в теоретическом описании атрибуты примитивов, получить линии с заданными параметрами, приведенными в табл. 7.1 с индивидуальными заданиями.

Таблица 7.1

Варианты индивидуальных заданий к теме 7

№ варианта	Начальная точка, (x_0, y_0)	Конечная точка, (x_1, y_1)	Ширина линии	Начальный цвет	Конечный цвет
1	(10, 10)	(50, 10)	2	Красный	Желтый
2	(0, 20)	(20, 20)	4	Синий	Зеленый
3	(0, 0)	(20, 20)	6	Зеленый	Оранжевый
4	(10, 10)	(40, 15)	3	Желтый	Красный
5	(10, 10)	(50, 10)	7	Оранжевый	Голубой
6	(0, 0)	(20, 20)	4	Голубой	Фиолетовый
7	(25, 10)	(50, 10)	8	Зеленый	Синий
8	(0, 20)	(20, 20)	14	Желтый	Зеленый
9	(10, 20)	(20, 20)	9	Оранжевый	Красный
10	(0, 10)	(50, 10)	6	Синий	Оранжевый
11	(0, 10)	(20, 20)	3	Красный	Голубой
12	(10, 0)	(60, 15)	5	Желтый	Синий
13	(10, 10)	(50, 10)	5	Оранжевый	Зеленый
14	(10, 20)	(20, 20)	10	Красный	Голубой
15	(0, 0)	(20, 20)	4	Синий	Желтый

Библиотека БГУИР

Список использованных источников

1. Торн, А. Основы анимации в *Unity* / А. Торн. – М. : ДМК-Пресс, 2016. – 176 с.

2. Вольф, Д. *OpenGL* 4. Язык шейдеров. Книга рецептов / Д. Вольф. – М. : ДМК-Пресс, 2015. – 368 с.

3. Прикладные пакеты векторной графики [+ электр. вариант] : учеб.-метод. пособие / О. С. Киселевский [и др.]. – Минск : БГУИР , 2016. – 96 с.

4. Торн, А. Искусство создания сценариев в *Unity* / А. Торн. – М. : Профессиональная книга, 2015. – 368 с.

5. Гурский, Ю. А. *Photoshop CS*3 / Ю. А. Гурский, А. В. Жвалевский. – СПб. : Питер, 2007. – 208 с.

6. Миронов, Д. Компьютерная графика в дизайне: учебник для вузов / Д. Миронов. – М. : Питер, 2008. – 720 с.

7. Жвалевский, А. В. *CorelDRAW X*3 / А. В. Жвалевский, Д. Донцов. – СПб. : Питер, 2007. – 144 с.

8. Сиденко, Л. А. Компьютерная графика и геометрическое моделирование: учеб. пособие / Л. А. Сиденко. – СПб. : Питер, 2009. – 224 с.

9. Ламмерс, К. Шейдеры и эффекты в *Unity*. Книга рецептов / К. Ламмерс. – М. : Профессиональная книга, 2016. – 274 с.

10. Пахомова, А. В. Колористика. Цветовая композиция. Практикум / А. В. Пахомова, Н. В. Брызгов. – Издательство В. Шевчук, 2011. – 232 с.

11. Гуриков, С. Р. Введение в программирование на языке *Visual C#* : учеб. пособие / С. Р. Гуриков. – М. : Форум НИЦ ИНФРА-М, 2013. – 448 с.

12. Казанский, А. А. Объектно-ориентированное программирование на *Visual Basic* 2010 и *Visual C#* 2010 в среде разработки *Microsoft Visual Studio* / А. А. Казанский. – М. : МГСУ, 2012. – 424 с.

13. Райт, Р. *OpenGL*. Суперкнига. – 3-е изд. / Р. Райт, Б. Липчак. – М. : Издательский дом «Вильямс», 2006. – 1040 с.

14. Панкратова, Т. *Photoshop* 7: учеб. курс / Т. Панкратова. – М : Питер, 2002. – 524 с.

Библиотека БГУИР

Св. план 2018, поз. 20

Учебное издание

Рак Татьяна Александровна
Шатилова Ольга Олеговна
Нестеренков Сергей Николаевич

ДВУМЕРНАЯ ВИЗУАЛИЗАЦИЯ. ОСНОВЫ РАБОТЫ С OpenGL

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. В. Иванюшина*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 25.09.2018. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 5,0. Уч.-изд. л. 5,0. Тираж 50 экз. Заказ 236.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6

Библиотека БГУИР