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

Кафедра интеллектуальных информационных технологий

С. А. Самодумкин, М. Д. Степанова, Д. Г. Колб

ПРАКТИКУМ ПО КОМПЬЮТЕРНОЙ ГРАФИКЕ

В 3-х частях

Часть 2

Под редакцией В. В. Голенкова

Рекомендовано УМО по образованию
в области информатики и радиоэлектроники
в качестве учебно-методического пособия
для специальности 1-40 03 01 «Искусственный интеллект»

Минск БГУИР 2014

УДК 004.92(076.5)
ББК 32.973.26-018.2я73
С17

Р е ц е н з е н т ы:

кафедра информационно-вычислительных систем учреждения образования «Военная академия Республики Беларусь» (протокол №2 от 1 октября 2012 г.);

главный научный сотрудник государственного научного учреждения «Объединенный институт проблем информатики НАН Беларуси», доктор технических наук, профессор В. В. Старовойтов

Самодумкин, С. А.

С17 Практикум по компьютерной графике. В 3 ч. Ч. 2 : учеб.-метод. пособие / С. А. Самодумкин, М. Д. Степанова, Д. Г. Колб ; под ред. В. В. Голенкова. – Минск : БГУИР, 2014. – 80 с. : ил.

ISBN 978-985-543-043-9 (ч. 2).

Содержатся теоретические сведения, упражнения для самостоятельной работы и задания для выполнения лабораторных занятий по дисциплине «Графический интерфейс интеллектуальных систем». Рассмотрены полигональные алгоритмы компьютерной графики.

Часть 1-я издана в БГУИР в 2007 г.

УДК 004.92(076.5)
ББК 32.973.26-018.2я73

ISBN 978-985-543-043-9 (ч. 2)
ISBN 978-985-488-173-7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Обработка информации – одна из самых важных функций компьютера. В компьютерной среде информация может быть разделена на графические изображения, лингвистическую и звуковую информацию. Что касается обработки информации, связанной с изображениями, то здесь можно выделить три основных направления: компьютерная графика (КГ), обработка изображений (ОИ) и распознавание образов.

Задачей **компьютерной графики** является формирование изображений, или визуализация. Формирование изображения выполняется в три этапа: построение модели объекта (описание его геометрических элементов и их связей); подготовка модели к визуализации (выполнение геометрических преобразований, удаление невидимых линий); визуализация (отсечение окном, формирование растрового представления, вывод на терминал). Примером может служить область машиностроительного черчения, получившая свое развитие в системах автоматизированного проектирования (САПР). В таких системах часто необходимо выполнять сечения для произвольных тел.

Обработка изображений – это преобразование изображений. То есть и входными данными, и результатом является изображение. Примером обработки изображений могут служить: фильтрация, сглаживание, подавление шумов, повышение контраста, четкости, коррекция цветов и т. д. В качестве исходных материалов для обработки информации могут служить космические снимки, получаемые в цифровом виде; цифровые карты; отсканированные изображения и т. п. Задачей обработки изображений может быть как улучшение в зависимости от определенного критерия, так и специальное преобразование, кардинально изменяющее изображения. Например, для наложения космического снимка на карту необходимо его преобразовать в ортофотоплан, поскольку космоснимки выполняются в центральной проекции, а карта – в ортогональной. Оказывается, из снимка нельзя получить план аффинными преобразованиями, а только проективными, причем необходимы дополнительные данные о рельефе. В рассмотренном примере обработка изображений является промежуточным этапом для дальнейшего распознавания изображения. В данном случае перед распознаванием часто необходимо выделять контуры, создавать бинарное изображение, разделять по цветам.

При **распознавании изображений** основная задача – получение описания изображенных объектов. Цель распознавания может формулироваться по-разному: выделение отдельных элементов (например условных знаков на карте), классификация изображения в целом (например проверка, изображен ли определенный объект на карте). Задача распознавания является обратной по отношению к визуализации.

Пособие состоит из отдельных тем, соответствующих лабораторным работам. Каждая тема сопровождается необходимыми теоретическими сведениями и примерами работы алгоритмов компьютерной графики. В конце каждой темы приведены задание на выполнение лабораторной работы, а также упражнения, которые могут быть полезны студентам при подготовке к защите лабораторных работ и сдаче экзамена по дисциплине. Неотъемлемой частью практикума является виртуальная лаборатория, используемая при проведении лабораторных работ, целью которой является визуальное сопровождение пошаговой работы алгоритмов компьютерной графики. Библиотека алгоритмов размещена на сервере кафедры интеллектуальных информационных технологий БГУИР.

В процессе лабораторной работы студентам необходимо:

1) изучить теоретическую часть (тема данного практикума, конспект лекций, дополнительная литература);

2) осуществить программную реализацию задания средствами алгоритмического языка высокого уровня;

3) подготовить и защитить отчет по лабораторной работе.

В отчете необходимо отразить цель лабораторной работы; привести формальную запись используемых в работе алгоритмов; распечатать тексты алгоритмов на языке программирования; дать результаты пошагового выполнения алгоритмов в виде таблицы; привести алгоритмическую оценку реализованных алгоритмов и сформулировать выводы по работе.

Авторы глубоко признательны рецензентам – коллективу кафедры информационно-вычислительных систем учреждения образования «Военная академия Республики Беларусь», канд. техн. наук, доценту В. М. Берикбаеву; главному научному сотруднику Объединенного института проблем информатики НАН Беларуси, д-ру техн. наук, профессору В. В. Старовойтову.

1. ГЕОМЕТРИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ

1.1. Аффинные преобразования на плоскости

Преобразование объектов изображений и изображений в целом – одна из наиболее часто встречаемых на практике задач в компьютерной графике. Осуществлять такие преобразования приходится и в системах автоматизированного проектирования, и в индустрии компьютерных игр, и в графических редакторах. Широкое применение геометрические преобразования имеют в геоинформационных системах, так, для преобразования карт из исходной координатной системы в заданную систему координат используются ортогональные преобразования, а при наложении на карту аэрокосмоснимков, полученных в центральной проекции, используются перспективные преобразования.

Геометрическое преобразование – это взаимно однозначное отображение прямой, плоскости или пространства на себя.

Геометрические преобразования по свойствам делятся на следующие виды:

– изометрия (сохраняются расстояния); примерами таких преобразований являются композиции переносов и поворотов;

– подобие (сохраняются углы); примером таких преобразований является однородное масштабирование;

– аффинные преобразования (сохраняется параллельность линий); примерами таких преобразований являются сдвиг, масштабирование;

– линейные (проективные) преобразования (прямые переходят в прямые); примером таких преобразований является перспектива;

– нелинейные преобразования.

В свою очередь преобразования могут комбинироваться, тем самым создаются сложные преобразования.

На рис. 1.1 показаны отношения между типами геометрических преобразований. Из него следует, что наиболее общей разновидностью преобразований являются ***аффинные преобразования***.

Рис. 1.1. Типы геометрических преобразований

В общем случае геометрические преобразования по размерности делятся на следующие виды:

– двухмерные (2D-преобразования);

– трехмерные (3D-преобразования).

Аффинные преобразования – это точечные взаимно однозначные отображения плоскости (пространства) на себя, при которых прямые переходят в прямые.

Рассмотрим основные свойства аффинных преобразований.

1. Аффинные преобразования сохраняют прямые линии. Прямая линия отображается в прямую линию, треугольник отображается в треугольник, а многоугольник отображается в многоугольник (рис. 1.2). Чтобы реализовать аффинные преобразования для линий и многоугольников, графическому механизму достаточно вычислить отображения их вершин, а затем провести линии или соединительные отрезки.

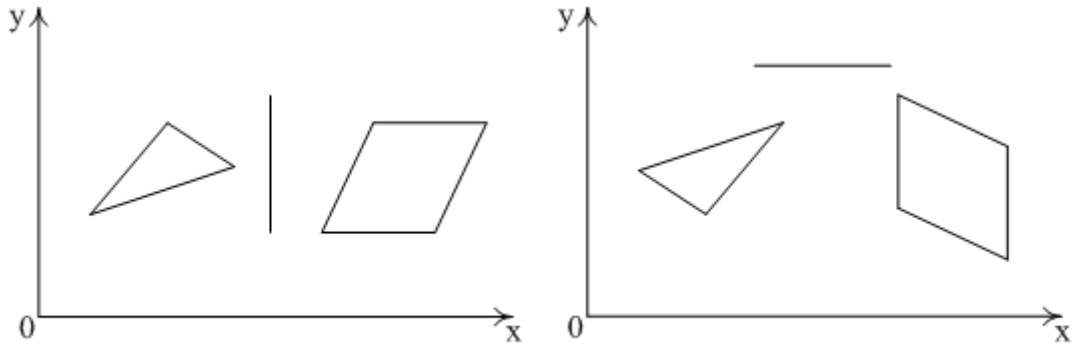


Рис. 1.2. Свойство прямых линий аффинных преобразований

2. Аффинные преобразования сохраняют параллельность (параллельные линии отображаются в параллельные линии) (рис. 1.3). Таким образом, параллелограмм отображается в параллелограмм, хотя прямоугольники не всегда отображаются в прямоугольники, а квадраты необязательно отображаются в квадраты.

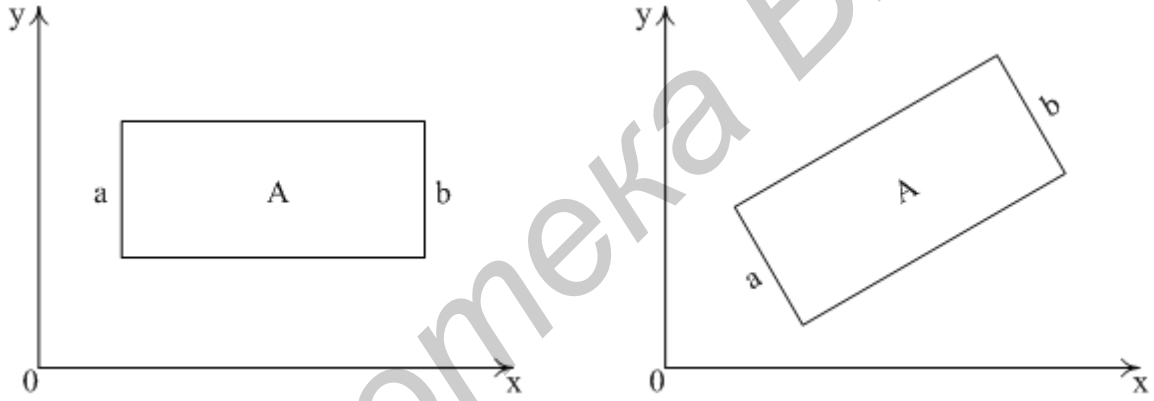


Рис. 1.3. Свойство параллельности аффинных преобразований

3. Аффинные преобразования сохраняют эллипсы. Эллипс в результате аффинного преобразования всегда отображается в эллипс, хотя окружность не всегда отображается в окружность.

4. Аффинные преобразования сохраняют кривые Безье.

5. Аффинное преобразование однозначно определяется тремя вершинами неколлинеарных векторов p, q, r в исходной системе координат и тремя вершинами неколлинеарных векторов p', q', r' в результирующей системе координат. Другими словами, существует только одно аффинное преобразование, которое отображает p, q, r соответственно в p', q', r'.

1.2. Двухмерные преобразования

Для двухмерных преобразований над объектами определены аффинные преобразования сдвига (перемещения), масштабирования (скалирования), отражения (относительно осей абсцисс и ординат) и поворот объекта на заданный угол.

Допустим, на плоскости введена прямолинейная координатная система. Тогда каждой точке M ставится в соответствие упорядоченная пара чисел (x, y) ее координат. Вводя на плоскости еще одну прямолинейную систему координат, мы ставим в соответствие той же точке M другую пару чисел – (x', y').

Переход от одной прямолинейной координатной системы на плоскости к другой описывается следующими соотношениями:

$$\begin{cases} x' = \alpha x + \beta y + \lambda, \\ y' = \gamma x + \delta y + \mu, \end{cases} \tag{1.1}$$

где $\alpha, \beta, \delta, \lambda, \mu$ – произвольные числа, связанные неравенством $\begin{vmatrix} \alpha & \beta \\ \gamma & \delta \end{vmatrix} \neq 0$, т. е.

$$\alpha\delta - \beta\gamma \neq 0.$$

В аффинных преобразованиях плоскости особую роль играют несколько важных частных случаев, имеющих хорошо прослеживаемые геометрические характеристики. При исследовании геометрического смысла числовых коэффициентов в выражении (1.1) для этих случаев нам удобно считать, что заданная система координат является *прямоугольной декартовой*.

Рассмотрим базовые преобразования.

1. Поворот вокруг начальной точки на угол φ (рис. 1.4):

$$\begin{cases} x' = x\cos\varphi - y\sin\varphi, \\ y' = x\sin\varphi + y\cos\varphi. \end{cases} \tag{1.2}$$

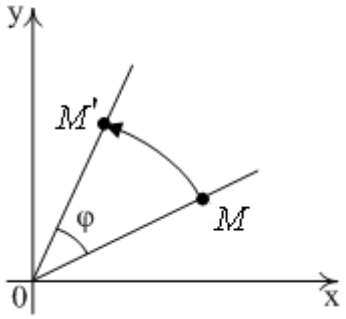


Рис. 1.4. Поворот

2. Масштабирование (растяжение, сжатие) вдоль координатных осей (рис. 1.5). Растяжение обеспечивается при условии, что $\alpha > 1, \delta > 1$, а сжатие $-\ \alpha < 1, \delta < 1$.

$$\begin{cases} x' = \alpha x, \\ y' = \delta y; \end{cases} \qquad (1.3)$$

$\alpha > 0, \delta > 0.$

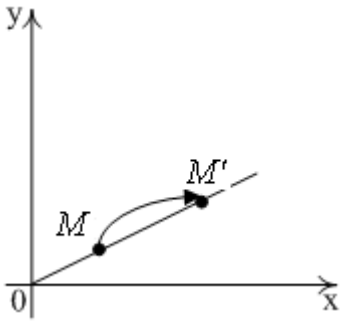


Рис. 1.5. Масштабирование

3. Отражение:

относительно оси абсцисс (рис. 1.6)

$$\begin{cases} x' = x, \\ y' = -y. \end{cases} \qquad (1.4)$$

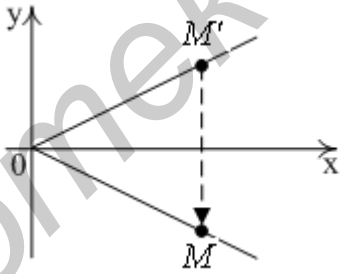


Рис. 1.6. Отражение относительно оси абсцисс

относительно оси ординат (рис. 1.7)

$$\begin{cases} x' = -x, \\ y' = y. \end{cases} \qquad (1.5)$$

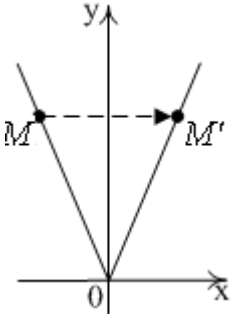


Рис. 1.7. Отражение относительно оси ординат

4. Перенос ((λ, μ) – координаты вектора переноса *MM'*) (рис. 1.8):

$$\begin{cases} x' = x + \lambda, \\ y' = y + \mu. \end{cases} \quad (1.6)$$

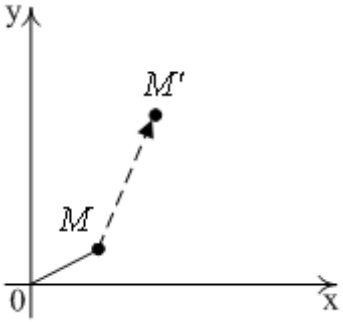


Рис. 1.8. Перенос

Отметим еще одно ***важное свойство аффинных преобразований на плоскости***: *любое* выражение вида (1.1), т. е. *любое преобразование*, можно описать при помощи отображений, задаваемых формулами (1.2) – (1.6).

Для эффективного использования этих известных формул в задачах компьютерной графики более удобной является их матричная запись. Матрицы, соответствующие случаям (1.2) – (1.5), имеют соответственно следующий вид:

$$\begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}, \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Однако для решения рассматриваемых далее задач желательно охватить матричным подходом все пять указанных выше простейших преобразований (в том числе и перенос), а значит, и общее аффинное преобразование. Этого можно достичь, используя математический аппарат ***однородных координат***.

Однородные координаты точки

Пусть M – произвольная точка плоскости с координатами x и y, вычисленными относительно заданной прямолинейной координатной системы.

Однородными координатами этой точки называется любая тройка чисел $x_1, x_2, x_3 \neq 0$, связанных с заданными числами x и y следующими соотношениями: $\frac{x_1}{x_3} = x$, $\frac{x_2}{x_3} = y$.

При решении задач компьютерной графики однородные координаты вводятся следующим образом: произвольной точке $M(x, y)$ плоскости ставится в соответствие точка в пространстве (рис. 1.9).

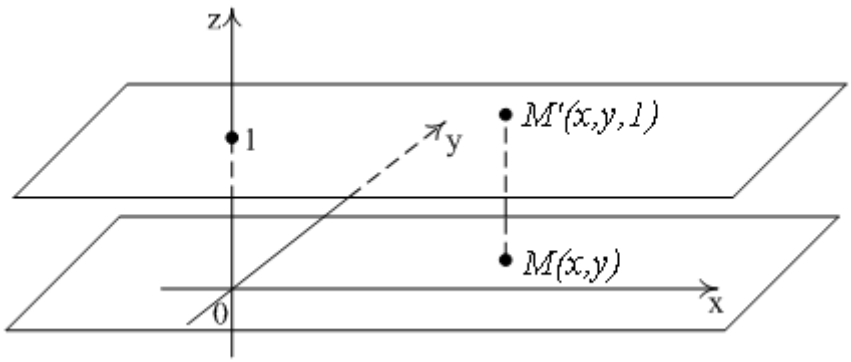


Рис. 1.9. Представление точки в однородных координатах

Заметим, что произвольная точка на прямой, соединяющей начало координат с точкой $M_0(x, y, 1)$, может быть задана тройкой чисел вида (hx, hy, h).

Будем считать, что $h \neq 0$.

Вектор $[hx, hy, h]$ является направляющим вектором прямой, соединяющей начало координат с точкой $M_0(x, y, 1)$. Эта прямая пересекает плоскость $z = 1$ в точке $(x, y, 1)$, которая однозначно определяет точку (x, y) координатной плоскости xy.

Тем самым между произвольной точкой с координатами (x, y) и множеством троек чисел вида $(hx, hy, h), h \neq 0$ устанавливается взаимно однозначное соответствие, позволяющее считать числа hx, hy, h новыми координатами этой точки.

Двухмерный вектор $[x, y]$ в однородных координатах записывается в виде $[wx, wy, w]$, где $w \neq 0$. Число w называется *масштабным множителем*.

Для того чтобы из вектора, записанного в однородных координатах, получить вектор в обычных координатах, необходимо разделить первые две координаты на третью: $[wx/x, wy/w, w/w] \rightarrow [x, y, 1]$.

В общем случае осуществляется переход от n-мерного пространства к $(n + 1)$-мерному. Это преобразование не единственное. Обратное преобразование называется *проекцией однородных координат*.

Некоторые точки, неопределенные в n-мерном пространстве, становятся вполне определенными при переходе к однородным координатам.

Например, однородный вектор $[0, 0, 1, 0]$ в трехмерном пространстве соответствует бесконечно удаленной точке $z = \infty$. Поскольку в однородных

координатах эту точку можно представить в виде $(0, 0, 1, \varepsilon)$ при $\varepsilon \to 0$, то в трехмерном пространстве это соответствует точке $(0, 0, 1/\varepsilon)$.

Применение однородных координат оказывается удобным во многих аспектах задач компьютерной графики:

– представление координат в целых числах (для h=1 точку с координатами $(0.5, 0.1, 1.5)$ представить нельзя, однако при h=10 для той же точки имеем координаты $(5, 1, 15)$);

– предотвращение арифметического переполнения (для точки с координатами $(80000, 40000, 1000)$ можно взять h=0,001, в результате получим $(80, 40, 1)$);

– удобство в применении к геометрическим преобразованиям.

При помощи троек однородных координат и матриц третьего порядка можно описать *любое аффинное преобразование плоскости*. Полагая h=1, соотношение (1.1) примет вид

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \beta & \lambda \\ \gamma & \delta & \mu \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \qquad (1.7)$$

1.4. Двухмерные преобразования в однородных координатах

Элементы произвольной матрицы аффинного преобразования не несут в себе явно выраженного геометрического смысла. Поэтому, чтобы реализовать то или иное *отображение*, т. е. ***найти элементы соответствующей матрицы по заданному геометрическому описанию***, необходимы специальные приемы. Обычно построение этой матрицы в соответствии со сложностью рассматриваемой задачи и с описанными выше частными случаями разбивают на несколько этапов. На каждом этапе подбирается матрица, соответствующая тому или иному из выделенных выше случаев (1.2 – 1.6), обладающих хорошо выраженными геометрическими свойствами.

Запишем матричное представление операций преобразования для однородных координат:

1. Преобразование поворота в однородных координатах:

$$R(Q) = \begin{bmatrix} \cos Q & \sin Q & 0 \\ -\sin Q & \cos Q & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad (1.8)$$

где Q – угол поворота.

2. Преобразование масштабирования в однородных координатах:

$$S(Sx, Sy) = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (1.9)$$

где *Sx*, *Sy* – коэффициенты масштабирования по осям *X* и *Y* соответственно.

3. Преобразование перемещения в однородных координатах:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix}, \quad (1.10)$$

где *Dx*, *Dy* – смещение по осям *X* и *Y* соответственно.

4. Преобразование отражения в однородных координатах:

$$M(x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \text{отражение относительно оси абсцисс;} \quad (1.11)$$

$$M(y) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \text{отражение относительно оси ординат.} \quad (1.12)$$

Запишем матричное представление *обратных операций* преобразования для однородных координат:

1. Обращение поворота выполняется поворотом на тот же угол в противоположную сторону ($\varphi = -Q$). Например двухмерному повороту на угол Q вокруг начала координат соответствует следующая матрица обратного преобразования:

$$R(Q)^{-1} = R(-Q) = \begin{bmatrix} \cos Q & -\sin Q & 0 \\ \sin Q & \cos Q & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (1.13)$$

Отрицательные значения углов поворота дают вращение по часовой стрелке, поэтому при умножении матрицы поворота на обратную получается единичная матрица. Заметим, что матрица, обратная к матрице поворота R, равна транспонированной матрице ($R(Q)^{-1} = R^T$).

2. Матрица обратного преобразования масштабирования получается заменой параметров масштабирования обратными величинами. При двухмерном масштабировании с параметрами *Sx* и *Sy* относительно начала координат обратная матрица записывается следующим образом:

$$S(Sx,Sy)^{-1} = \begin{bmatrix} \frac{1}{Sx} & 0 & 0 \\ 0 & \frac{1}{Sy} & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (1.14)$$

Обратная матрица генерирует противоположное преобразование масштабирования, поэтому умножение любой матрицы масштабирования на обратную дает единичную матрицу.

3. В матрице перемещения расстояния, на которые оно выполнялось, берутся с противоположными знаками. Следовательно, если двухмерное перемещение выполнялось на расстояния *Dx* и *Dy*, обратная матрица имеет следующий вид:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Dx & -Dy & 1 \end{bmatrix}. \quad (1.15)$$

1.5. Двухмерные сложные преобразования

Используя матричные представления, последовательность преобразований можно задать в виде матрицы сложного преобразования, вычислив произведение отдельных преобразований. Поскольку точка представляется однородным вектором-строкой, на вектор-строку нужно *перемножить слева направо матрицы, представляющие требуемые преобразования*. Кроме того, поскольку обычно одна последовательность преобразований применяется ко многим точкам, эффективнее вначале перемножить матрицы преобразования и получить матрицу композиции преобразования.

Таким образом, если требуется применить два преобразования к точке P, новое положение точки будет вычислено следующим образом:

$$P' = P \cdot M_1 \cdot M_2 = P \cdot M.$$

Точка преобразуется с помощью комплексной матрицы M, а не с использованием сначала преобразования M_1, а затем преобразования M_2.

1.5.1. *Сложный двухмерный поворот*

Два последовательных поворота, примененные к точке P, дают следующее положение:

$$P' = R(Q_2) \cdot \{R(Q_1) \cdot P\} = P \cdot \{R(Q_2) \cdot R(Q_1)\}.$$

Перемножая две матрицы поворота, можно убедиться, что два последовательных поворота являются аддитивными:

$$R(Q_2) \cdot R(Q_1) = R(Q_1 + Q_2),$$

так что координаты конечной точки можно вычислить с помощью матрицы сложного поворота:

$$P' = R(Q_1 + Q_2) \cdot P.$$

1.5.2. *Сложное двухмерное масштабирование*

Умножение матриц двух последовательных операций масштабирования дает следующую сложную матрицу масштабирования:

$$\begin{bmatrix} S_{2x} & 0 & 0 \\ 0 & S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_{1x} & 0 & 0 \\ 0 & S_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} S_{1x} \cdot S_{2x} & 0 & 0 \\ 0 & S_{1y} \cdot S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

В этом случае суммарная матрица указывает на то, что последовательные операции масштабирования являются *мультипликативными*. Например, если дважды утроить размер объекта, в результате объект увеличится в 9 раз.

1.5.3. *Сложное двухмерное перемещение*

Если два последовательных вектора перемещения $[D_{1x,}, D_{1y}]$ и $[D_{2x,}, D_{2y}]$ применяются к двухмерной точке P, окончательное положение P' вычисляется следующим образом:

$$P' = T(D_{2x}, D_{2y}) \cdot \{T(D_{1x}, D_{1y}) \cdot P\} = P \cdot \{T(D_{2x}, D_{2y}) \cdot T(D_{1x}, D_{1y})\},$$

где P и P' представляются в однородных координатах трехэлементными векторами-столбцами.

Данный результат можно проверить, вычислив матричное произведение двух соответствующих матриц. Кроме того, матрица суммарного преобразования, определенного данной последовательностью перемещений, будет иметь следующий вид:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_{2x} & D_{2y} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_{1x} & D_{1y} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_{1x} + D_{2x} & D_{1y} + D_{2y} & 1 \end{bmatrix}.$$

1.5.4. *Произвольный двухмерный поворот вокруг оси*

Если в графическом пакете имеется только функция поворота вокруг начала координат, двухмерный поворот вокруг любой другой оси $(\boldsymbol{x_r}, \boldsymbol{y_r})$ можно

получить, выполнив следующую последовательность операций перемещения – поворота – перемещения (рис. 1.10).

1. Переместить объект так, чтобы положение центра вращения совместилось с началом координат.

2. Повернуть объект вокруг начала координат.

3. Переместить объект так, чтобы центр вращения вернулся в исходное положение.

Матрица итогового преобразования для данной последовательности действий получается как результат умножения матриц:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -D_x & -D_y & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos Q & \sin Q & 0 \\ -\sin Q & \cos Q & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos Q & \sin Q & 0 \\ -\sin Q & \cos Q & 0 \\ D_x(1-\cos Q) + D_y \sin Q & D_y(1-\cos Q) - D_x \sin Q & 1 \end{bmatrix}.$$

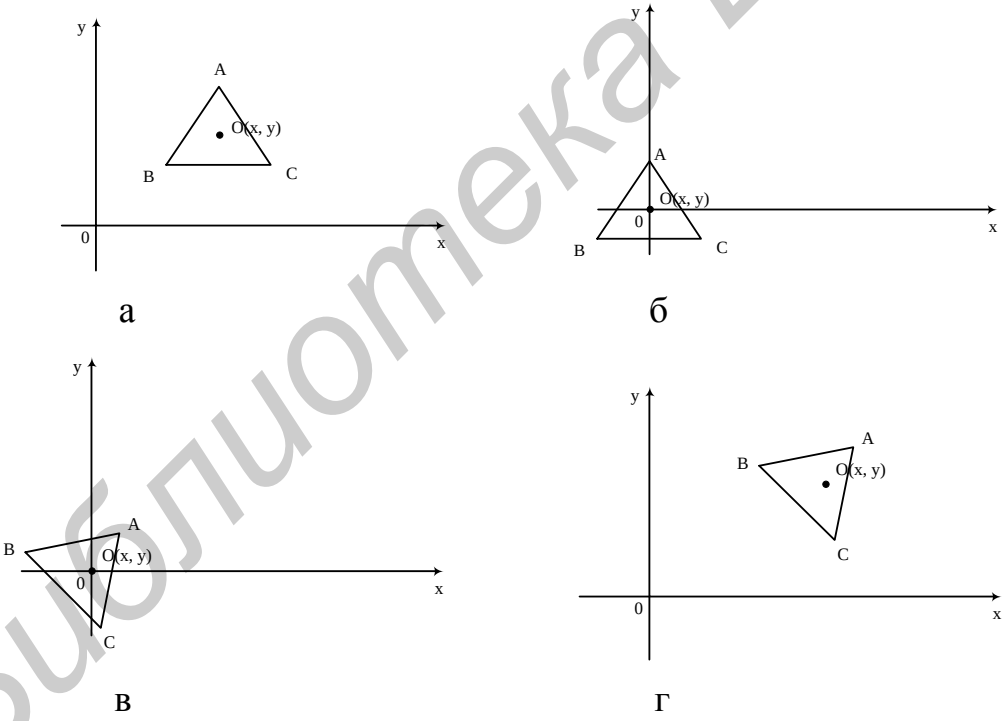


Рис. 1.10. Последовательность преобразований при повороте объекта вокруг заданной точки с использованием матрицы поворота, представленной в выражении (1.2):
а – исходное положение объекта и центра вращения; б – такое перемещение объекта, чтобы положение центра вращения совместилось с началом координат; в – поворот вокруг начала координат; г – такое перемещение объекта, чтобы центр вращения вернулся в исходное положение

Пример. Задана фигура – прямоугольник *ABCD*. Получить матрицу преобразования в прямоугольник $A'B'C'D'$ (рис. 1.11).

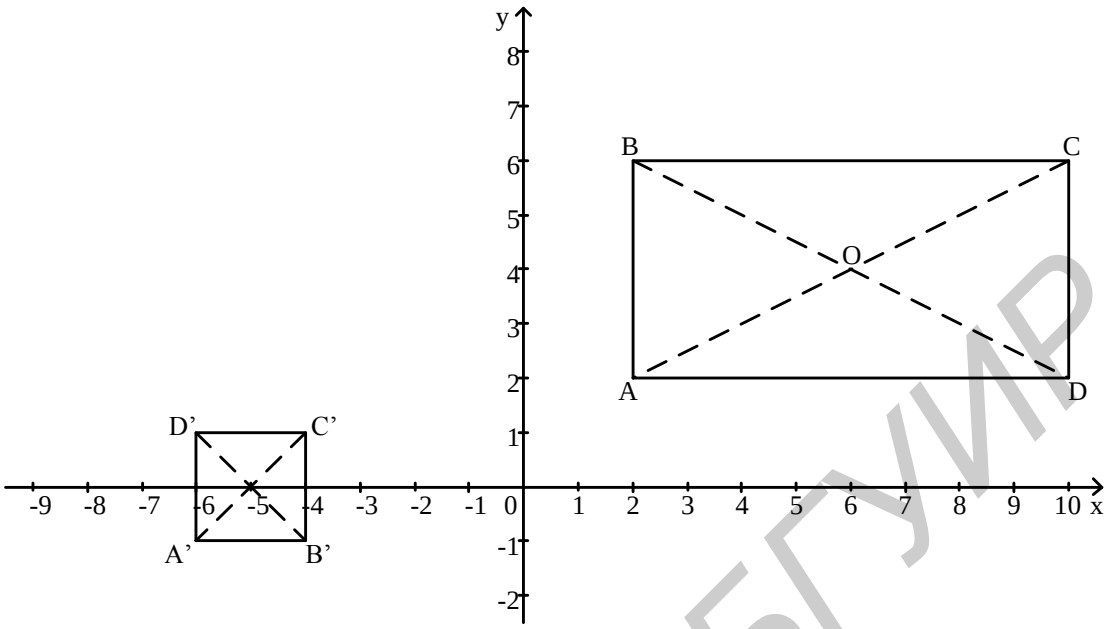


Рис. 1.11. Четырехугольники *ABCD* и искомый $A'B'C'D'$

Запишем матрицу исходной фигуры в однородных координатах:

$$Q = \begin{bmatrix} 2 & 2 & 1 \\ 2 & 6 & 1 \\ 10 & 6 & 1 \\ 10 & 2 & 1 \end{bmatrix}.$$

Для получения матрицы преобразования и искомого прямоугольника *A’B’C’D’* необходимо провести сложное двухмерное преобразование.

1. Перемещение центра в точку (0,0).
2. Масштабирование по *x* в ¼ раза и по *y* в ½ раза.
3. Поворот против часовой стрелки на 90°.
4. Отражение относительно оси ординат.
5. Перемещение влево на 5.

Запишем последовательность преобразований с математическим описанием действий и получением новой фигуры на каждом шаге.

1. Перемещение центра фигуры в точку начала координат (рис. 1.12):

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -6 & -4 & 1 \end{bmatrix}; \; M = \begin{bmatrix} 2 & 2 & 1 \\ 2 & 6 & 1 \\ 10 & 6 & 1 \\ 10 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -6 & -4 & 1 \end{bmatrix} = \begin{bmatrix} -4 & -2 & 1 \\ -4 & 2 & 1 \\ 4 & 2 & 1 \\ 4 & -2 & 1 \end{bmatrix}.$$

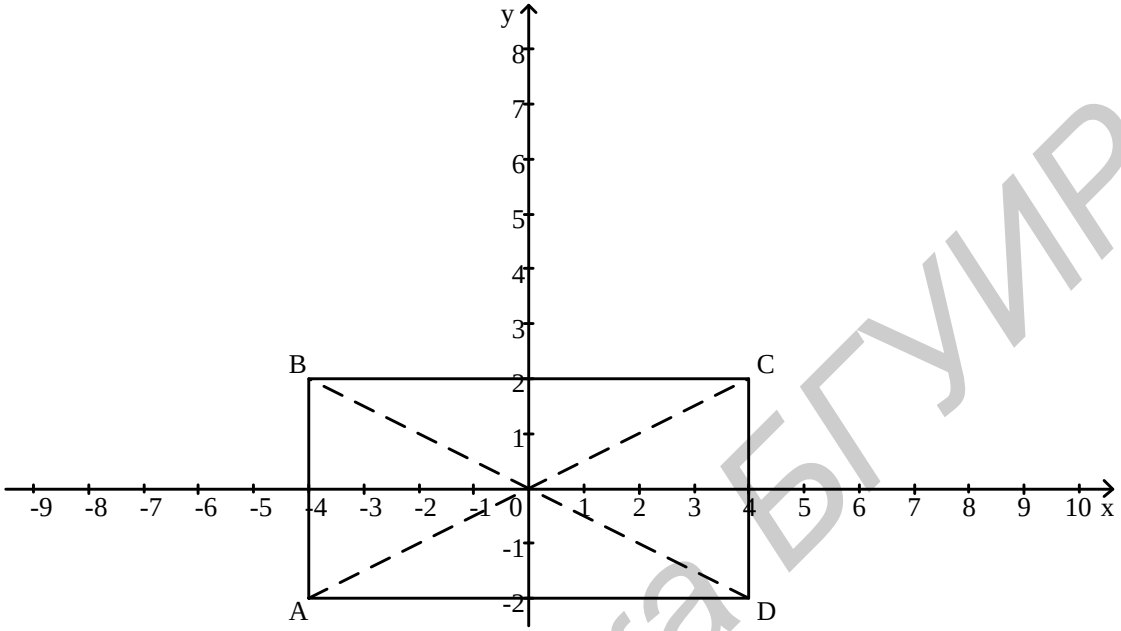


Рис. 1.12. Графическое представление переноса в начало координат

2. Масштабирование по *x* в ¼ раза и по *y* в ½ раза (рис. 1.13):

$$T = \begin{bmatrix} 0{,}25 & 0 & 0 \\ 0 & 0{,}5 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \; M = \begin{bmatrix} -4 & -2 & 1 \\ -4 & 2 & 1 \\ 4 & 2 & 1 \\ 4 & -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0{,}25 & 0 & 0 \\ 0 & 0{,}5 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix}.$$

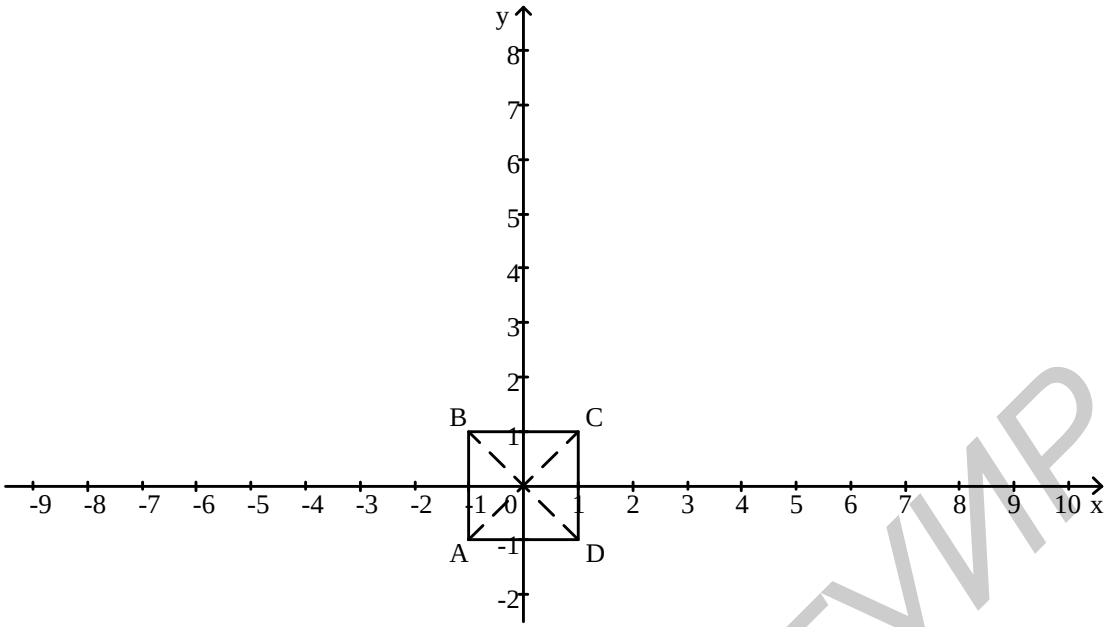


Рис. 1.13. Матричное и графическое представление масштабирования по x в ¼ раза и по y в ½ раза

3. Поворот против часовой стрелки на 90° (рис. 1.14):

$$T = \begin{bmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2} & 0 \\ -\sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

$$M = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

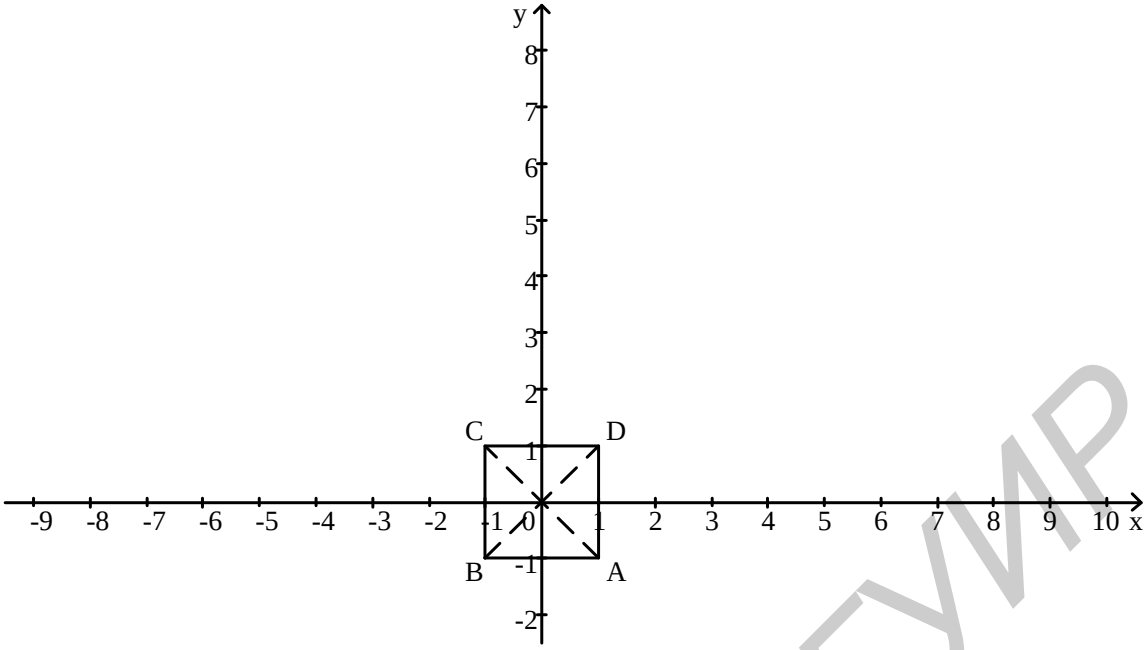


Рис. 1.14. Графическое представление поворота против часовой стрелки на 90°

4. Отражение относительно оси ординат (рис. 1.15):

$$T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \; M = \begin{bmatrix} 1 & -1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}.$$

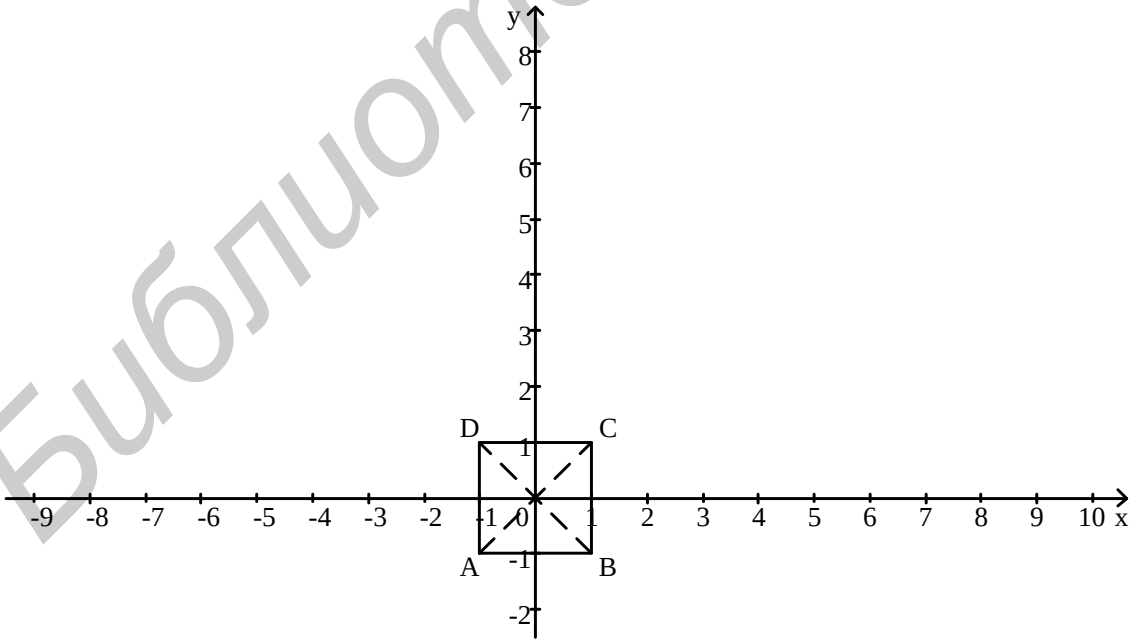


Рис. 1.15. Графическое представление отражения относительно оси ординат

5. Перемещение влево на 5 (рис. 1.16):

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{bmatrix}. \; M = \begin{bmatrix} -1 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -6 & -1 & 1 \\ -4 & -1 & 1 \\ -4 & 1 & 1 \\ -6 & 1 & 1 \end{bmatrix}.$$

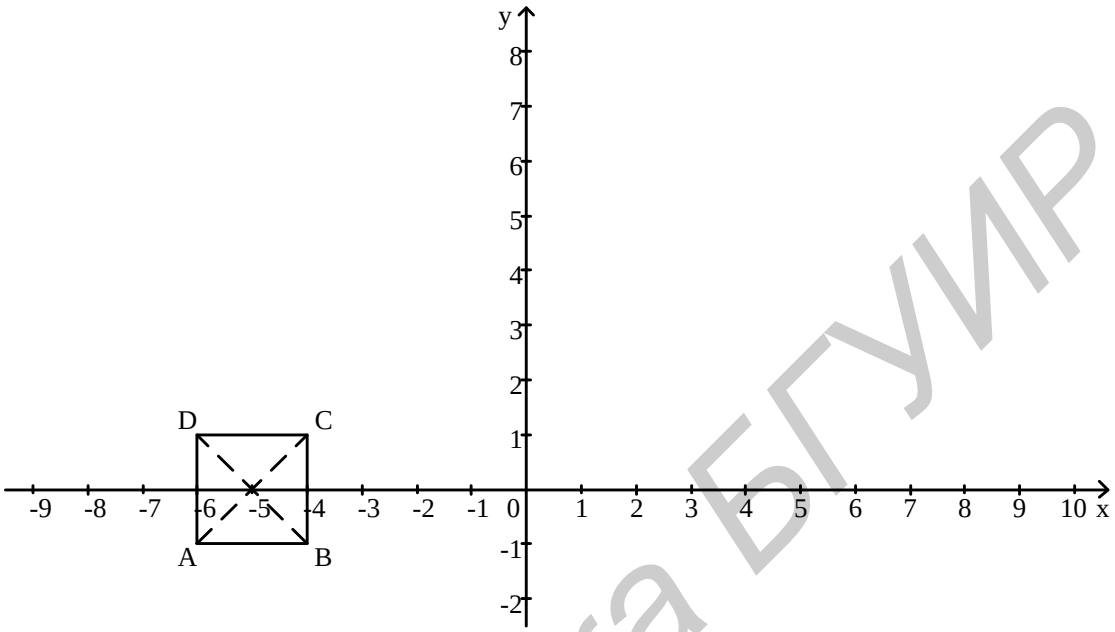


Рис. 1.16. Графическое представление перемещения влево на 5

Таким образом, после пяти шагов простых преобразований получим матрицу искомого четырехугольника $A'B'C'D'$:

$$M = \begin{bmatrix} -6 & -1 & 1 \\ -4 & -1 & 1 \\ -4 & 1 & 1 \\ -6 & 1 & 1 \end{bmatrix}.$$

Данный результат может быть получен путем умножения матрицы исходной фигуры *ABCD* на матрицу композиции преобразования. При этом матрица композиции преобразования получается путем последовательного перемножения матриц преобразования на каждом шаге.

В рассматриваемом примере матрица композиции преобразования получается следующим образом:

$$K = T(-6;-4)\cdot S(1/4;1/2)\cdot R(90°)\cdot M(y)\cdot T(-5;0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -6 & -4 & 1 \end{bmatrix}\cdot\begin{bmatrix} 1/4 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}\times$$

$$\times\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0.25 & 0 \\ 0.5 & 0 & 0 \\ -7 & -1.5 & 1 \end{bmatrix}.$$

Умножая матрицу исходной фигуры *ABCD* на матрицу композиции преобразования *K*, получим искомую фигуру $A'B'C'D'$:

$$\begin{bmatrix} 2 & 2 & 1 \\ 2 & 6 & 1 \\ 10 & 6 & 1 \\ 10 & 2 & 1 \end{bmatrix}\cdot\begin{bmatrix} 0 & 0.25 & 0 \\ 0.5 & 0 & 0 \\ -7 & -1.5 & 1 \end{bmatrix} = \begin{bmatrix} -6 & -1 & 1 \\ -4 & -1 & 1 \\ -4 & 1 & 1 \\ -6 & 1 & 1 \end{bmatrix}.$$

1.5.5. *Свойства произведения матриц*

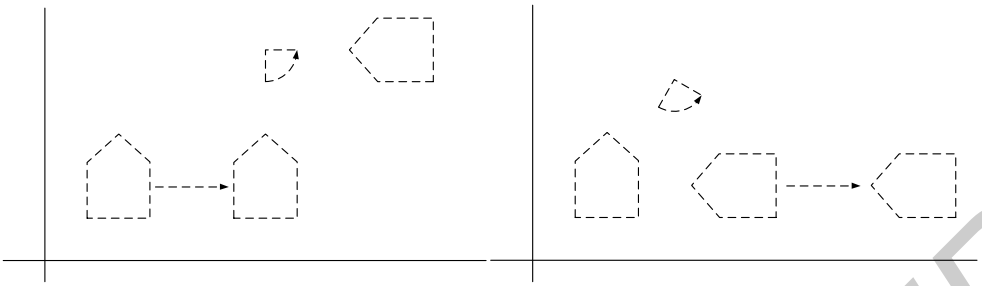
Умножение матриц ассоциативно. Для любых трех матриц M_1, M_2 и M_3 в произведении $M_1 \cdot M_2 \cdot M_3$ все равно, какое умножение выполнять первым – M_3 и M_2 или M_2 и M_1:

$$M_1 \cdot M_2 \cdot M_3 = (M_3 \cdot M_2)\cdot M_1 = M_3 \cdot (M_2 \cdot M_1).$$

Следовательно, в зависимости от порядка, в котором заданы преобразования, сложную матрицу можно строить, перемножая матрицы либо слева направо (умножение слева), либо справа налево (умножение справа). Некоторые графические пакеты требуют, чтобы преобразования задавались в том порядке, в котором их нужно применять. В этом случае вначале нужно активизировать преобразование M_1, затем M_2 и M_3. При вызове каждой последующей процедуры преобразования соответствующая матрица присоединяется слева от предыдущего матричного произведения. В то же время в других графических системах матрицы умножаются справа, так что данная последовательность преобразований будет вызываться в обратном порядке: последнее активизированное преобразование (в приведенном примере это M_1) применяется первым, а первое вызванное преобразование (в рассматриваемом случае – M_3) будет применено последним.

В то же время произведение преобразований может не быть коммутативным. Матричное произведение $M_2 \cdot M_1$ в общем случае не равно $M_1 \cdot M_2$. Это означает, что, если требуется транслировать и повернуть объект, нужно акку-

ратно указать порядок, в котором вычисляется матрица суммарного преобразования (рис. 1.17).



а б

Рис. 1.17. Влияние обращения порядка выполнения ряда преобразований на положение преобразованного объекта:
а – объект транслируется в положительном направлении оси *x*, затем поворачивается против часовой стрелки на угол 45°; б – объект поворачивается на 45° против часовой стрелки, а затем транслируется в положительном направлении оси *x*

В некоторых частных случаях (например, когда подряд выполняется несколько преобразований одного типа) умножение матриц преобразований коммутативно. Например, два последовательных поворота можно выполнить в любом порядке, при этом конечное положение объекта не изменится. Данное свойство коммутативности также справедливо для двух последовательных трансляций или изменений масштаба.

Другой коммутирующей парой операций является поворот и пропорциональное масштабирование ($S_x = S_y$).

1.6. Преобразования в трехмерных координатах

Методы геометрических преобразований в трех измерениях являются развитием двухмерных методов, в которые добавлены соображения, касающиеся координаты z. Перемещение объекта теперь выполняется с помощью заданного трехмерного вектора перемещения, который определяет, насколько нужно переместить объект по каждой из трех координат. Аналогично, чтобы масштабировать объект, выбирают масштабный коэффициент для каждой из трех декартовых координат. Однако расширение методов двухмерных поворотов в три измерения не так прямолинейно. При обсуждении двухмерных поворотов на плоскости xy требовалось рассмотреть только повороты вокруг осей, перпен-

дикулярных плоскости xy. В трехмерном пространстве можно выбирать любую пространственную ориентацию осей поворота.

Некоторые графические пакеты могут обрабатывать трехмерные повороты как совокупность трех поворотов – по одному на каждую декартову ось. В качестве альтернативного решения можно задать общие уравнения поворота, дающие ориентацию оси поворота и требуемого угла поворота.

Подобно тому, как двухмерные преобразования описываются матрицами размером 3×3, трехмерные преобразования могут быть представлены матрицами размером 4×4. Тогда трехмерная точка (x, y, z) записывается в однородных координатах как (wx, wy, wz, w), где $w \neq 0$. Для получения декартовых координат надо первые три однородные координаты разделить на w:

$$P(X, Y, Z) = p(x, y, z) = p(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}).$$

Трехмерное положение, выраженное в однородных координатах, представляется четырехэлементным вектором-столбцом. Кроме того (как и в двух измерениях), любая последовательность преобразований представляется одной матрицей, сформированной последовательной сверткой матриц отдельных преобразований.

1.6.1. *Трехмерное перемещение*

Точка $P(x, y, z)$ трехмерного пространства перемещается в положение $P'(x', y', z')$ путем прибавления расстояний D_x, D_y и D_z к декартовым координатам точки P:

$$x = x' + D_x \quad y = y' + D_y \quad z = z' + D_z \ .$$

Данную операцию трехмерной трансляции можно выразить в матричной форме, как в уравнении (1.10) для двухмерной ситуации. Однако теперь точки P и P' представляются в однородных координатах четырехэлементными векторами-столбцами, и оператор трансляции T– это матрица 4 x 4:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D_x & D_y & D_z & 1 \end{bmatrix}. \qquad (1.16)$$

Чтобы транслировать объект в трех измерениях, нужно преобразовать все координаты объекта, затем восстановить объект в новом положении. Если объект представлен как набор многоугольных поверхностей, транслируется каждая

вершина каждой поверхности, и в конечных положениях строятся многоугольные грани.

Матрица, обратная матрице трехмерного перемещения, получается с использованием тех же процедур, что применялись при двухмерном перемещении. Следовательно, изменяется знак расстояний трансляции D_x, D_y и D_z. В результате получается перемещение в противоположном направлении, причем произведение матрицы перемещения и обратной к ней равно единичной матрице.

1.6.2. *Трехмерный поворот*

Объект можно поворачивать вокруг любой оси в пространстве, но легче всего обрабатывать повороты вокруг осей, параллельных декартовым осям. Кроме того, можно использовать комбинации поворотов координатных осей (дополненные соответствующим переносом) для того, чтобы задать повороты вокруг любой другой линии пространства. Таким образом, вначале рассмотрим операции, фигурирующие при поворотах вокруг координатных осей, а затем проанализируем расчеты, необходимые для других осей поворотов.

По договоренности положительные углы поворота дают повороты против часовой стрелки вокруг координатной оси при условии, что мы смотрим в отрицательном направлении координатной оси.

Уравнения двухмерного поворота вокруг оси z распространяются на три измерения:

$$\begin{cases} x' = x\cos Q - y\sin Q, \\ y' = x\sin Q + y\cos Q, \\ z' = z. \end{cases} \tag{1.17}$$

Параметр Q задает угол поворота вокруг оси z, а значения координаты z при этом преобразовании не меняются. Уравнения трехмерного поворота вокруг оси z следующим образом записываются через однородные координаты:

$$R_z(Q) = \begin{bmatrix} \cos Q & \sin Q & 0 & 0 \\ -\sin Q & \cos Q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.18}$$

Уравнения преобразования для поворотов вокруг двух других координатных осей можно получить с помощью цикличной перестановки параметров x, y и z в уравнениях (1.17):

$$x \to y \to z \to x. \tag{1.19}$$

Подставляя перестановки (1.19) в уравнения (1.17), получаем выражения для поворота вокруг оси x:

$$\begin{cases} y' = y\cos Q - z\sin Q, \\ y' = y\sin Q + z\cos Q, \\ x' = x \end{cases} \quad (1.20)$$

или в матричной форме

$$R_x(Q) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos Q & \sin Q & 0 \\ 0 & -\sin Q & \cos Q & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.21)$$

Циклическая перестановка координат в уравнениях (1.20) дает уравнения преобразования для поворота вокруг оси y:

$$\begin{cases} z' = z\cos Q - x\sin Q, \\ x' = z\sin Q + x\cos Q, \\ y' = y \end{cases} \quad (1.22)$$

или в матричной форме

$$R_y(Q) = \begin{bmatrix} \cos Q & 0 & -\sin Q & 0 \\ 0 & 1 & 0 & 0 \\ \sin Q & 0 & \cos Q & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.23)$$

Матрица, обратная матрице трехмерного поворота, получается аналогично матрице обратного поворота в двух измерениях. Нужно заменить угол Q углом $-Q$. Отрицательные значения углов поворота вращают тело по часовой стрелке, и при умножении матрицы поворота на обратную ей получается единичная матрица. Поскольку при изменении знака угла поворота меняется только значение функции синус, обратную матрицу можно также получить, поменяв строки и столбцы. Следовательно, матрицу, обратную любой матрице поворота R, можно найти, вычислив транспонированную матрицу ($R^{-1} = R^T$).

1.6.3. *Трехмерное масштабирование*

Матричное выражение трехмерного масштабирования точки относительно начала координат является простым расширением двухмерного масштабирования. В матрицу преобразования включается параметр, отвечающий за масштабирование по оси z.

Явные выражения для преобразования масштабирования относительно начала координат записываются так:

$$x' = x \cdot S_x y' = y \cdot S_y z' = z \cdot S_z, \qquad (1.24)$$

где масштабным коэффициентам S_x, S_y и S_z присваиваются любые положительные значения.

Матричная запись трехмерного масштабирования

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (1.25)$$

Масштабирование объекта с помощью преобразования (1.25) меняет положение объекта относительно начала координат. При коэффициенте масштабирования, превышающем 1, точка удаляется от начала координат в соответствующем направлении. Если же коэффициент меньше 1, точка приближается к началу координат по этой координате. Кроме того, если не все коэффициенты масштабирования равны, меняются относительные размеры объекта. Чтобы сохранить исходную форму объекта, используется пропорциональное масштабирование: $S_x = S_y = S_z$.

1.6.4. *Трехмерный поворот вокруг произвольной оси*

Когда объект нужно повернуть вокруг оси, не параллельной ни одной из координатных осей, нужны дополнительные преобразования. В этом случае требуется поворот, выравнивающий ось вращения с выбранной координатной осью, а также возврат оси поворота к исходной ориентации. Для данной спецификации оси вращения и угла поворота требуемый поворот можно выполнить за пять шагов (рис. 1.18).

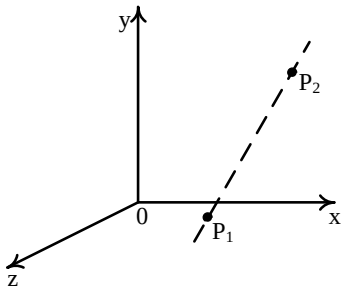
1. Переместить объект так, чтобы ось вращения прошла через начало координат.

2. Так повернуть объект, чтобы ось вращения совместилась с одной из координатных осей.

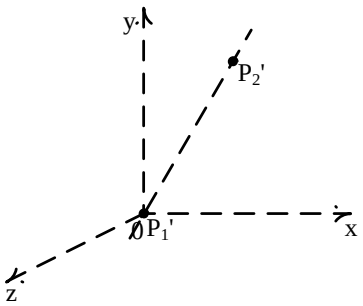
3. Выполнить заданный поворот вокруг выбранной координатной оси.

4. Применить обратные повороты для возврата оси вращения к исходной ориентации.

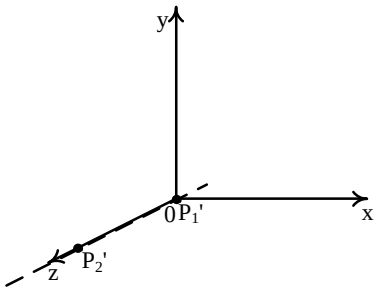
5. Применить обратное перемещение, чтобы вернуть ось вращения к исходному пространственному положению.



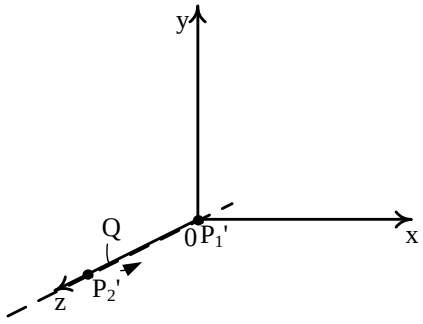
Исходное положение



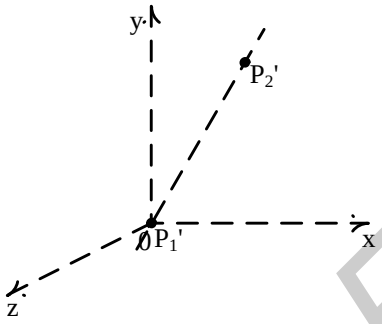
Этап 1: переместить P_1 в начало координат



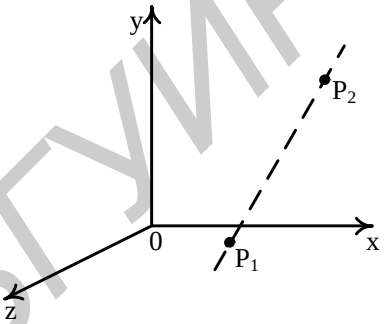
Этап 2: повернуть P_2 на ось z



Этап 3: повернуть объект вокруг оси z



Этап 4: повернуть ось в начальное положение



Этап 5: переместить ось в исходное положение

Рис. 1.18. Пять этапов преобразования, при выполнении которых находится сложная матрица поворота вокруг произвольной оси. Ось вращения проектируется на ось z

Предполагается, что ось вращения определяется двумя точками (см. рис. 1.18) и что направление поворота – против часовой стрелки, если смотреть по оси в направлении от P_2 к P_1.

Компоненты вектора оси вращения вычисляются следующим образом:

$$V = P_2 - P_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1).$$

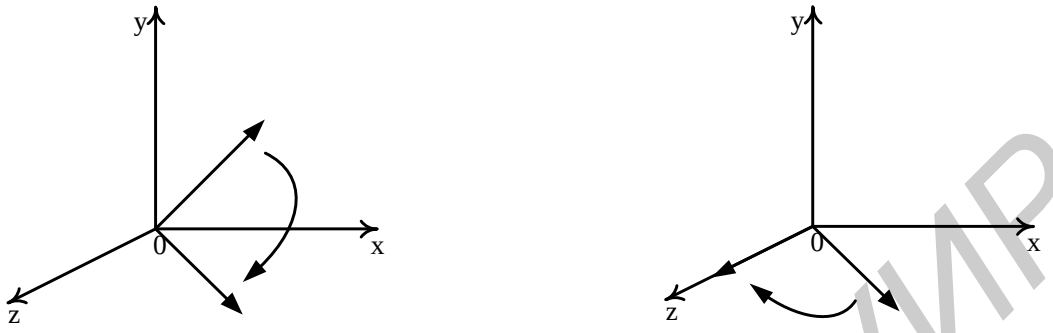
Компоненты a, b и c – направляющие косинусы оси вращения:

$$a = \frac{x_2 - x_1}{|V|} \quad b = \frac{y_2 - y_1}{|V|} \quad c = \frac{z_2 - z_1}{|V|}.$$

Первый шаг при выполнении поворота – задать матрицу перемещения в начало координат:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix}. \qquad (1.26)$$

Далее нужно сформулировать преобразования, которые совместят ось вращения с осью z. Вначале выполним поворот вокруг оси x, а затем вокруг оси y. Поворот вокруг оси x переводит вектор и на плоскость xz, а поворот вокруг оси y совмещает и с осью z (рис. 1.19).



а б

Рис. 1.19. Схематическое представление этапов поворота вектора оси вращения для совмещения с осью z:
а – поворот вокруг оси x; б – поворот вокруг оси y

Чтобы записать матрицу преобразования, характеризующую поворот вокруг оси x, нужно определить синус и косинус угла поворота α, необходимого для перевода вектора и на плоскость xz.

Из выражений скалярных и векторных произведений векторов находим:

$\sin\alpha = \frac{b}{d};\ \cos\alpha = \frac{c}{d};\ d = \sqrt{b^2 + c^2}.$

Матрица поворота вектора вокруг оси x с последующим вращением в плоскости xz:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.27)$$

Матрица преобразования, характеризующая поворот вектора вокруг оси y, равна

$$R_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.28)$$

С помощью матриц (1.25) – (1.27) выравниваем ось вращения по положительному направлению оси z. Теперь поворот на заданный угол Q можно выполнить вокруг оси z:

$$R_z(Q) = \begin{bmatrix} \cos Q & \sin Q & 0 & 0 \\ -\sin Q & \cos Q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Чтобы завершить требуемый поворот вокруг данной оси, нужно вернуть ось вращения в исходное положение. Для этого применяются преобразования, обратные (1.26) – (1.28).

1.7. Проективные преобразования

В общем случае проекции преобразуют точки, заданные в системе координат размерностью n, в точки системы координат размерностью, меньшей чем n. В рассматриваемом случае с помощью проецирования три измерения отображаются в два. Проекция трехмерного объекта (представленного в виде совокупности точек) строится при помощи прямых проецирующих лучей, которые называются *проекторами* и которые выходят из центра проекции, проходят через каждую точку объекта и, пересекая картинную плоскость, образуют проекцию. На рис. 1.20 и 1.21 показаны две различные проекции одного и того же отрезка, а также проекторы, проходящие через его конечные точки. *Проекция отрезка сама является отрезком*, так что достаточно спроецировать одни лишь конечные точки.

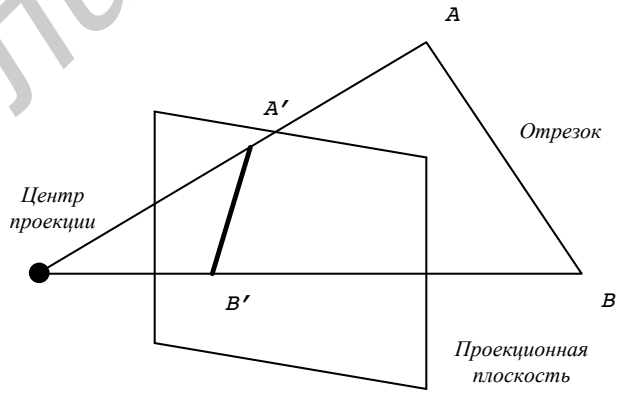


Рис. 1.20. Отрезок AB и его центральная проекция $A'B'$

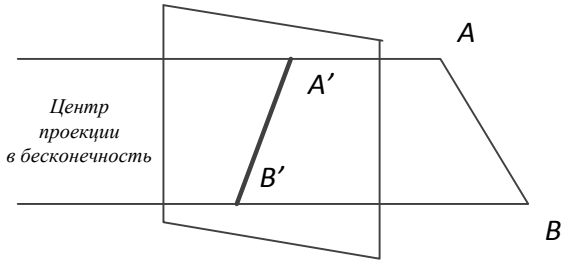


Рис. 1.21. Отрезок AB и его параллельная проекция $A'B'$.
Проекторы AA' и BB' параллельны

Определенный таким образом класс проекций известен под названием плоских геометрических проекций, поскольку проецирование в этом случае производится на плоскость, а не на искривленную поверхность и в качестве проекторов используются прямые, а не кривые линии. В отличие от них *многие картографические проекции являются либо неплоскими, либо негеометрическими*.

Плоские геометрические проекции, которые в дальнейшем будем называть просто *проекциями*, можно подразделить на два основных класса: ***центральные (перспективные)*** и ***параллельные***.

Различия между способами проецирования.

1. В *параллельных* проекциях центры проекции располагаются в бесконечности, а проекторы параллельны.

2. В *перспективной* проекции любое множество линий непараллельных плоскости проекции пересекаются в бесконечно удаленной точке (БУТ).

3. Прямые, параллельные осям координат, пересекаются в главных БУТ (ГБУТ).

Перспективные проекции делят *по числу ГБУТ*:

1. Одна ГБУТ. Используется, если наблюдатель находится на поверхности земли, пример (с одной бесконечно удаленной точкой) можно видеть на рисунке:

2. Две ГБУТ. Этот случай используется для того, чтобы показать, что наблюдатель находится высоко над землей.

3. Три ГБУТ. Практически не используется, т. к. не дает большего реализма, чем первые два случая.

Простейшей является *параллельная прямоугольная проекция*. В ней совместно изображаются виды сверху, спереди и сбоку. Эти проекции часто используются в черчении. В зависимости от соотношения *между направлениями*

проецирования и нормалью к проекционной плоскости параллельные проекции подразделяются на:

– *ортографические*, или *ортогональные* (направления совпадают);

– *косоугольные* (направления не совпадают).

В зависимости от *положения осей системы координат объекта относительно проекционной плоскости* ортографические проекции делятся на:

– *аксонометрические* (оси системы координат составляют разные углы с проекционной плоскостью);

– *изометрические* (оси системы координат составляют одинаковые углы с проекционной плоскостью).

Центральная перспективная проекция приводит к визуальному эффекту, подобному тому, который дает зрительная система человека. При этом наблюдается *эффект перспективного укорачивания*, когда размер проекции объекта изменяется обратно пропорционально расстоянию от центра проекции до объекта. В параллельных проекциях отсутствует перспективное укорачивание, за счет чего изображение получается менее реалистичным, и *параллельные прямые всегда остаются параллельными*.

На рис. 1.22 приведена классификация плоских геометрических проекций.

Рис. 1.22. Классификация плоских геометрических проекций

Каждую из проекций можно описать матрицей размером 4×4. Этот способ оказывается удобным, поскольку появляется возможность объединить матрицу

проецирования с матрицей преобразования, представив в результате две операции (преобразование и проецирование) в виде одной матрицы.

Важным свойством центральной проекции является то, что отрезки прямых после перспективного преобразования переходят в отрезки прямых на проекционной плоскости. Это позволяет проецировать, т. е. производить вычисления только для конечных точек отрезков, а затем соединять проекции точек линиями уже на проекционной плоскости.

Имеется некоторый объект (точка $P(x, y, z)$) с начальными координатами, смотрящими на объект, и необходимо получить проекцию объекта на экранную плоскость (рис. 1.23).

Из подобия треугольников можно выразить:

$$\frac{x'}{d} = \frac{x}{z} \Rightarrow x' = \frac{x}{z/d},$$

$$\frac{y'}{d} = \frac{y}{z} \Rightarrow y' = \frac{y}{z/d},$$

где d – расстояние до плоскости проекции.

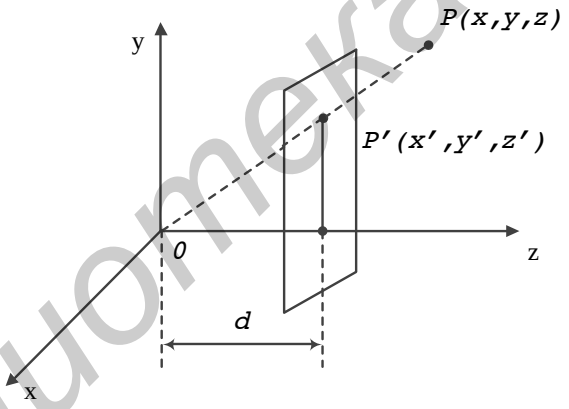


Рис. 1.23. Представление перспективной проекции

Все перспективные преобразования можно представить в виде преобразований в однородных координатах, описываемых матрицами 4×4:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.29)$$

Тогда координаты проекции получаются перемножением старых координат фигуры на матрицу перспективных преобразований:

$$[x \quad y \quad z \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[x \quad y \quad z \quad \frac{1}{d}\right] =$$

$$= \left[\frac{x}{z/d} \quad \frac{y}{z/d} \quad d \quad 1/d\right]. \qquad (1.30)$$

1.8. Вычисление бесконечно удаленных точек

Особенность перспективной проекции состоит в том, что параллельные прямые могут сходиться в бесконечно удаленной точке (БУТ).

Для вычисления однородных координат БУТ и координат ее проекции рассмотрим:

1. Прямую L, проходящую через точку $P(x_1, y_1, z_1)$, направление которой задает вектор $V(l, m, n)$.

2. Уравнение прямой имеет вид $\frac{x-x_1}{l} = \frac{y-y_1}{m} = \frac{z-z_1}{n}$.

3. Положим $L \neq 0$. Тогда имеют место выражения

$$y = \frac{m}{l} \cdot (x - x_1) + y_1,$$

$$y = \frac{n}{l} \cdot (x - x_1) + z_1.$$

4. Точки прямой образуют множество вида

$$\left[x;\ \frac{m}{l}(x - x_1) + y_1;\ \frac{n}{l}(x - x_1) + z_1;\ 1\right].$$

5. Разделим все 4 координаты на x и возьмем предел. Получим

$$\lim_{x\to\infty} \left[x;\ \frac{m}{l}\left(1 - \frac{x_1}{x}\right) + \frac{y_1}{x};\ \frac{n}{l}\left(1 - \frac{x_1}{x}\right) + \frac{z_1}{x};\ \frac{1}{x}\right] = \left[1;\ \frac{m}{l};\ \frac{n}{l}; 0\right]. \qquad (1.31)$$

Получили однородные координаты БУТ. В декартовы координаты перейти невозможно (т. к. есть 0), но зато можно получить проекцию.

Для получения координат проекции БУТ воспользуемся перспективным преобразованием

$$[l \quad m \quad n \quad 0] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix} = [l \quad m \quad n \quad n/d] = \left[\frac{ld}{n} \quad \frac{md}{n} \quad d \quad 1\right]. \qquad (1.32)$$

Произвольное перспективное преобразование будет задаваться матрицей следующего вида:

$$P = \begin{bmatrix} 1 & 0 & 0 & 1/d_x \\ 0 & 1 & 0 & 1/d_y \\ 0 & 0 & 1 & 1/d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.33)$$

Тогда проекция произвольной БУТ будет иметь следующие координаты:

$$[l \quad m \quad n \quad 0] \cdot P = \left[l \quad m \quad n \quad \frac{1}{d_x} + \frac{m}{d_y} + \frac{n}{d_z}\right]. \quad (1.34)$$

Данные математические выкладки продемонстрированы на рис. 1.24.

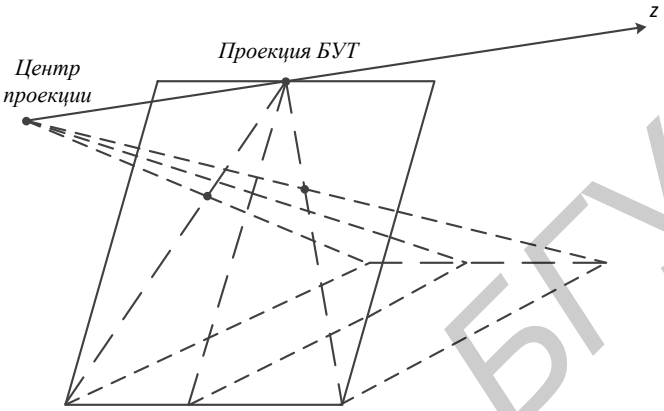


Рис. 1.24. Представление перспективной проекции

Координаты проекции главных точек схода после вычислений имеют вид:

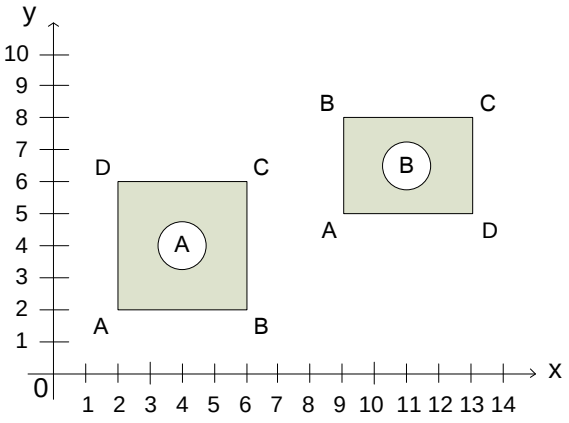
ГБУТ по оси X: $(l, m, n) = (1,0,0)[dx, 0,0,1]$.

ГБУТ по оси Y: $(l, m, n) = (0,1,0)[0, dy, 0,1]$.

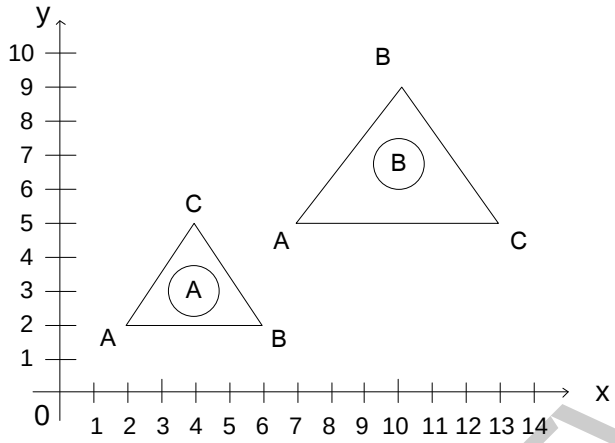
ГБУТ по оси Z: $(l, m, n) = (0,0,1)[0,0, dz, 1]$.

Упражнения для самостоятельной работы

Задание 1. Записать последовательность преобразований, приводящую фигуру *A* в фигуру *B*. Получить матрицу композиции преобразований и вектор данных фигуры *B* ([*B*]=[*A*]**K*, где *K* – матрица композиции преобразования).



Задание 2. Записать последовательность преобразований, приводящую фигуру *A* в фигуру *B*. Получить матрицу композиции преобразований и вектор данных фигуры *B* ([*B*]=[*A*]***K*, где *K* – матрица композиции преобразования).



Задание к лабораторной работе

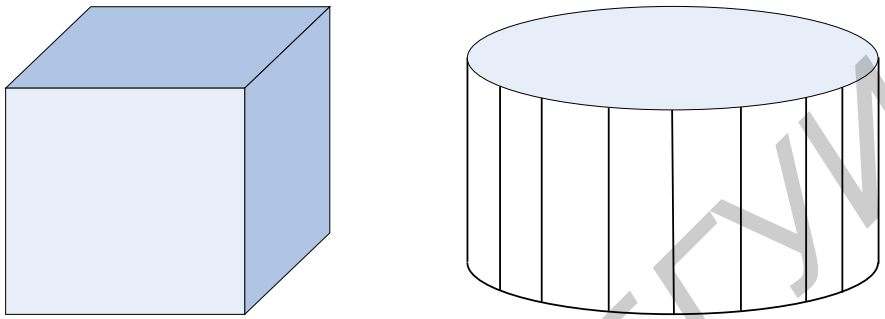
Разработать графическую программу, выполняющую следующие геометрические преобразования над трехмерным объектом: перемещение, поворот, скалирование, отображение, перспектива. В программе должно быть предусмотрено считывание координат 3D объекта из текстового файла, обработка клавиатуры и выполнение геометрических преобразований в зависимости от нажатых клавиш. Все преобразования следует производить с использованием матричного аппарата и представления координат в однородных координатах.

2. ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ПОЛИГОНОВ

В задачах компьютерной графики особое внимание уделяется алгоритмам обработки многоугольников, или полигонов. Это связано в первую очередь с тем, что графические функции способны обрабатывать многоугольники намного эффективнее, чем другие фигуры, поскольку границы многоугольников описываются линейными уравнениями. Более того, большинство криволинейных поверхностей можно достаточно корректно аппроксимировать набором выпуклых многоугольников. В последнем случае после наложения эффектов освещения и затемнения аппроксимированная поверхность выглядит достаточно реалистично. Аппроксимацию изогнутой поверхности с использованием граней, представленных в виде многоугольников, называют мозаичным представлением поверхности, или аппроксимацией поверхности с помощью сетки много-

угольников. Объекты, описанные с помощью набора многоугольников, называют стандартными графическими объектами (или просто графическими объектами), а модели этих объектов представляют собой контурные схемы и называются каркасными проволочными моделями объектов.

На рис. 2.1 показаны геометрические тела, аппроксимированные сеткой многоугольников в виде контурной схемы.



а б

Рис. 2.1. Каркасная модель видимых частей геометрических тел:
а – куб; б – цилиндр

Полигон* (*многоугольник*) –** замкнутая кривая на плоскости, образуемая отрезками прямых линий. Отрезки называются ***ребрами или ***сторонами*** полигона. Концевые точки отрезков, совпадающие для двух соседних ребрах, называются ***вершинами.*** Количество вершин или сторон, содержащихся в полигоне, определяет его размер. Полигон размером n будем называть n-гоном, или n-угольником.

Полигон называется ***простым***, если он не пересекает самого себя. Простой полигон охватывает непрерывную область плоскости, которая образует ***внутреннюю*** часть полигона. Неограниченная область, окружающая простой полигон, образует ***внешнюю*** часть, а набор точек, лежащих на самом полигоне, образует его границу. Такую границу будем называть ***оболочкой*** и чать $CH(Q)$, где Q – множество вершин многоугольника, принадлежащих границе полигона.

Вершины упорядочены циклически вдоль границы полигона. Две вершины, являющиеся концевыми точками одного и того же ребра, являются соседями и про них также говорят, что они ***смежные***.

Внутренним углом полигона называется угол внутри границы полигона, образованный двумя соседними его сторонами.

Вершины полигона подразделяются на ***выпуклые*** и ***вогнутые***. Вершина называется выпуклой, если внутренний угол при этой вершине меньше или равен 180°. В противном случае вершина считается вогнутой. Если все внутренние углы полигона меньше или равны 180°, то такой полигон – ***выпуклый***. Полигон, не являющийся выпуклым, называется ***вогнутым***.

Отрезок прямой линии между двумя не соседними вершинами называется ***диагональю***. Диагональ называется ***хордой*** (или внутренней диагональю), если она лежит внутри полигона и не затрагивает его внешнюю область.

Добавление к полигону хорды делит его на два меньших полигона.

На рис. 2.2 проиллюстрированы некоторые определения, относящиеся к полигонам.

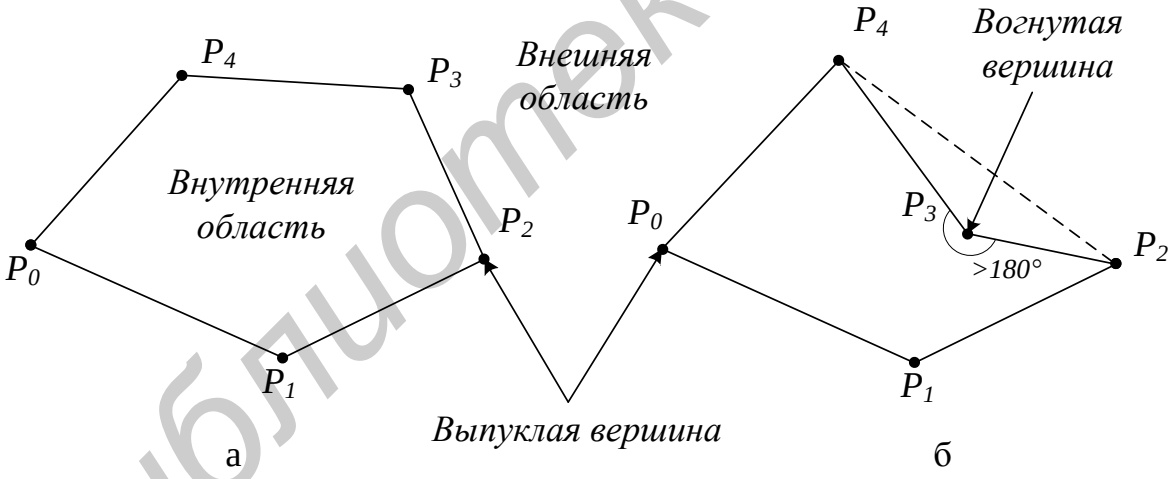


Рис. 2.2. Многоугольники:
а – выпуклый многоугольник и его выпуклая оболочка $CH(P_0, P_1, P_2, P_3, P_4)$; б – вогнутый многоугольник и его оболочка $CH(P_0, P_1, P_2, P_3, P_4)$

Отметим, что выпуклой оболочкой для вогнутого полигона на рис. 2.2, б является граница $CH(P_0, P_1, P_2, P_3, P_4)$.

Точку и отрезок прямой линии иногда удобно рассматривать как вырожденные полигоны. 1-гон состоит из единственной вершины и имеет единствен-

ное ребро нулевой длины, соединяющее вершину саму с собой. 2-гон содержит две вершины и два совпадающих ребра, соединяющих две вершины. Одним из преимуществ использования вырожденных полигонов является упрощение формирования полигона: начиная с 1-гона, вносится вторая вершина для образования 2-гона, добавление последующих вершин приводит к формированию обычных полигонов размерностью 3 и более. Таким образом, начальные этапы формирования полигона ничем не отличаются от последующих.

2.1. Основные задачи над полигонами

В методах компьютерной графики применяются, как правило, только выпуклые полигоны, т. к. реализация алгоритмов закрашивания и других графических процедур для вогнутых полигонов намного сложнее. Поэтому перед обработкой вогнутый полигон разделяют на набор выпуклых полигонов. Отметим, что, как и другие алгоритмы предварительной обработки полигонов, функции разделения вогнутого полигона часто не входят в графическую библиотеку. Следовательно, для многих таких библиотек, в том числе и библиотеки OpenGL, необходимо, чтобы все закрашиваемые полигоны были выпуклыми. Кроме того, в некоторых системах допускается только закрашивание треугольных областей, что значительно упрощает многие алгоритмы создания изображений, т. к. треугольник всегда является выпуклым полигоном.

К *основным задачам*, решаемым над полигонами, относятся:

1. Проверка полигона на выпуклость.
2. Определение принадлежности точки полигону.
3. Построение выпуклой оболочки.
4. Заполнение многоугольников.
5. Операции над многоугольниками (объединение, пересечение, деление и др.).
6. Триангуляция и построение диаграммы Вороного.

2.2. Алгоритм проверки полигона на выпуклость и нахождение его внутренних нормалей

Факт выпуклости полигона можно установить путем определения знаков векторных произведений его смежных сторон, а результатом векторного произведения является вектор, перпендикулярный плоскости полигона.

Векторное произведение двух плоских векторов V_1 и V_2 равно

$$V_1 \times V_2 = k \cdot \begin{vmatrix} V_{x1} & V_{y1} \\ V_{x2} & V_{y2} \end{vmatrix} = k \cdot \left(V_{x1}V_{y2} - V_{y1}V_{x2}\right), \qquad (2.1)$$

где k – единичный вектор, перпендикулярный плоскости, несущей векторы-сомножители.

В качестве исходных данных алгоритма проверки полигона на выпуклость является множество вершин полигона $Q_i, i = \overline{0, n-1}$, *где* n *– количество вершин полигона.*

Алгоритм проверки полигона на выпуклость

Шаг 1. Для каждой вершины полигона V_i (рис. 2.3) определить знак векторного произведения сторон, смежных вершине

$$\overline{V_iV_{i+1}} \times \overline{V_{i+1}V_{i+2}} = k \begin{vmatrix} x_{i+1} - x_i & y_{i+1} - y_i \\ x_{i+2} - x_{i+1} & y_{i+2} - y_{i+1} \end{vmatrix}.$$

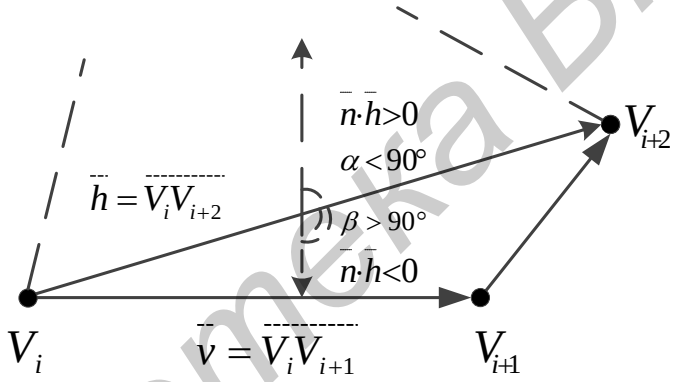


Рис. 2.3. Определение векторного произведения и нормалей к стороне $\bar{v}$

Шаг 2. Анализ знаков векторного произведения:

– все знаки равны нулю – полигон вырождается в *отрезок*;

– есть как положительные, так и отрицательные знаки – полигон *вогнутый*;

– все знаки положительные – полигон *выпуклый*, а внутренние нормали ориентированы *влево* от его контура;

– все знаки отрицательные – полигон *выпуклый*, а внутренние нормали ориентированы *вправо* от его контура.

Конец алгоритма.

Во многих задачах требуется нахождение векторов, перпендикулярных плоскости полигона (нормаль к сторонам). Пусть нам известен некоторый вектор $\bar{V}(v_x, v_y)$, началом и концом которого являются вершины одной из сторон полигона. Вектор $\bar{n}(n_x, n_y)$ будет перпендикулярен вектору $\bar{V}$, если выполняется равенство $\bar{n} \cdot \bar{V} = 0$. Используя свойство скалярного произведения векторов, будем иметь:

$$\bar{n}(n_x, n_y) \cdot \bar{V}(v_x, v_y) = n_x v_x + n_y v_y = 0,$$

$$n_x v_x = -n_y v_y.$$

Поскольку нас интересует только направление нормали положим $n_y = v_x$ без потери общности. Следовательно, $n_x = -v_y$, а вектор $\bar{n} = -v_y i_y + v_x j$.

Таким образом, вектор, перпендикулярный стороне полигона, вычисляется как

$$\bar{n} = [-v_y, v_x]. \qquad (2.2)$$

Факт направленности вектора $\overline{n_v}$ внутрь или вне полигона можно определить следующим образом.

Пусть известны три неколлинеарные вершины полигона V_i, V_{i+1}, V_{i+2} (см. рис. 2.3). Вектор стороны полигона образован как разность векторов пары смежных его вершин $\overline{V_i V_{i+1}} = [x_{i+1} - x_i, y_{i+1} - y_i]$, тогда вектор $n_{V_i V_{i+1}} = [-(y_{i+1} - y_i), (x_{i+1} - x_i)]$ перпендикулярен стороне $V_i V_{i+1}$ полигона, и известна хорда $\overline{V_i V_{i+2}}$, которая имеет координаты $[x_{i+2} - x_i, y_{i+2} - y_i]$.

Знак скалярного произведения $\overline{n_{V_i V_{i+1}}} \cdot \overline{V_i V_{i+2}}$ указывает на ориентацию вектора $\overline{n_{V_i V_{i+1}}}$ относительно контура полигона:

$$sign\left(\overline{n_{V_i V_{i+1}}} \cdot \overline{V_i V_{i+2}}\right) = \begin{cases} 1, \overline{n_{V_i V_{i+1}}} - \text{ориентирован внутрь полигона} \\ -1, \overline{n_{V_i V_{i+1}}} - \text{ ориентирован вне полигона} \end{cases} \qquad (2.3)$$

На рис. 2.3 приведена часть полигона, указана сторона $\bar{v}$, хорда $\bar{h}$, острый угол α, если $\bar{n}$ ориентирован внутрь полигона, и тупой угол β, если $\bar{n}$ ориентирован вне полигона.

Предполагается, что V_i, V_{i+1}, V_{i+2} – три последовательные неколлинеарные вершины (не лежат на одной прямой) выпуклого полигона.

$\overline{V_i V_{i+1}}$ – вектор стороны полигона.

$\overline{V_iV_{i+2}}$ – хорда полигона, проведенная из инцидентной заданной стороне $\overline{V_iV_{i+1}}$ полигона вершины.

$\overline{n_{V_iV_{i+1}}}$ – вектор, перпендикулярный стороне полигона.

$sign$ – функция, возвращающая $-1,0,1$ для отрицательного, нулевого и положительного аргумента соответственно.

Алгоритм нахождения внутренней нормали к стороне полигона

Шаг 1. Определить значения координат вектора стороны $\overline{V_iV_{i+1}}$, началом и концом которого являются соответственно вершины V_i и V_{i+1} полигона. $\overline{V_iV_{i+1}} = [x_{i+1} - x_i, y_{i+1} - y_i]$.

Шаг 2. Определить значения координат хорды $\overline{V_iV_{i+2}}$, началом которой является вершина V_i полигона, а концом – вершина полигона, следующая за второй вершиной, инцидентной заданной стороне $\overline{V_iV_{i+1}}$ полигона $\overline{V_iV_{i+2}} = [x_{i+2} - x_i, y_{i+2} - y_i]$.

Шаг 3. Найти вектор $\overline{n_{V_iV_{i+1}}}$, перпендикулярный заданной стороне $\overline{V_iV_{i+1}}$ полигона $n_{V_iV_{i+1}} = [-(y_{i+1} - y_i), (x_{i+1} - x_i)]$.

Шаг 4. Найти знак скалярного произведения векторов, определенных на шаге 2 и 3:

$$sign\left(\overline{n_{V_iV_{i+1}}} \cdot \overline{V_iV_{i+2}}\right) = sign(-(y_{i+1} - y_i) \cdot (x_{i+2} - x_i) + (x_{i+1} - x_i) \cdot (y_{i+2} - y_i)) =$$

$$= \begin{cases} 1, \overline{n_{V_iV_{i+1}}} - \text{вектор, перпендикулярный стороне,} \\ \quad \overline{V_iV_{i+1}} \text{ и ориентированный внутрь полигона,} \\ -1, \overline{n_{V_iV_{i+1}}} - \text{вектор, перпендикулярный стороне,} \\ \quad \overline{V_iV_{i+1}} \text{ и ориентированный вне полигона.} \end{cases}$$

Для ориентации вектора внутрь полигона необходимо умножить его на –1:

$(-1)\overline{n_{V_iV_{i+1}}} = [(y_{i+1} - y_i), -(x_{i+1} - x_i)]$.

Конец алгоритма.

Рассмотренный подход к нахождению внутренних нормалей возможен только для выпуклых полигонов, поскольку невозможно гарантировать, чтобы вектор $\overline{V_iV_{i+2}}$ являлся хордой полигона.

Пример. Заданы два полигона A (рис. 2.4, а) и B (рис. 2.4, б). Проверить полигоны на выпуклость. Для выпуклого полигона найти внутренние нормали к сторонам.

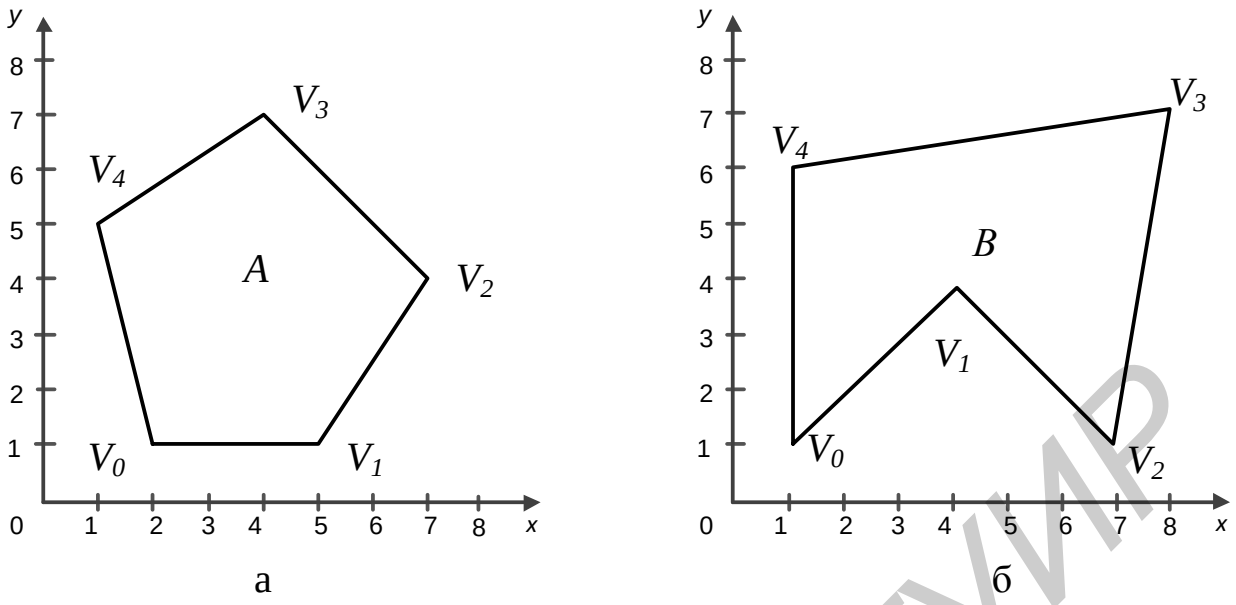


Рис. 2.4. Определение факта выпуклости полигонов

1. Выпишем вершины полигона A:

$V_0(2,1), V_1(5,1), V_2(7,4), V_3(4,7), V_4(1,5)$.

Выпишем вершины полигона B:

$V_0(1,1), V_1(4,4), V_2(7,1), V_3(8,7), V_4(1,6)$.

2. Найдем векторы, задающие каждую из сторон полигонов:

$V_{ij} = [x_j\text{-}x_i.\ y_j\text{-}y_i]$.

Для полигона A: $V_{01} = [3,0], V_{12} = [2,3], V_{23} = [-3,3], V_{34} = [-3,-2], V_{40} = [1,-4]$

Для полигона B: $V_{01} = [3,\ 3],\ V_{12} = [3,-3], V_{23} = [1,\ 6], V_{34} = [-7,-1], V_{40} = [0,-5]$.

3. Вычислим векторные произведения смежных сторон в соответствии с выражением (2.1)

Для полигона A:

$V_{01} \times V_{12} = 9k.$

$V_{12} \times V_{23} = 15k.$

$V_{23} \times V_{34} = 15k.$

$V_{34} \times V_{40} = 14k.$

$V_{40} \times V_{01} = 12k.$

Знаки всех векторных произведений положительные, следовательно, полигон A – выпуклый.

Для полигона B:

$V_{01} \times V_{12} = (-18)k.$

$V_{12} \times V_{23} = 21k.$

$V_{23} \times V_{34} = 41k.$

$V_{34} \times V_{40} = 35k.$

$V_{40} \times V_{01} = 15k.$

Векторное произведение $V_{01} \times V_{12}$ имеет отрицательный знак, следовательно, полигон B – вогнутый.

4. Найдем векторы, перпендикулярные каждой из сторон полигона, в соответствии с выражением (2.2).

Для полигона A:

$\overline{n_{V_oV_1}} = [0, 3].$

$\overline{n_{V_3V_4}} = [2, -3].$

$\overline{n_{V_1V_2}} = [-3,\ 2].$

$\overline{n_{V_4V_0}} = [4, 1].$

$\overline{n_{V_2V_3}} = [-3, 3].$

Для полигона B:

$\overline{n_{V_oV_1}} = [-3, 3].$

$\overline{n_{V_3V_4}} = [1, -7].$

$\overline{n_{V_1V_2}} = [3, 3].$

$\overline{n_{V_4V_0}} = [5, 0].$

$\overline{n_{V_2V_3}} = [-6, 1].$

5. Определим ориентацию векторов, вычисленных на 4 шаге, относительно контура полигонов A и B. На данном шаге для каждой из сторон полигона найдем хорду полигона и знаки скалярных произведений по формуле (2.3). Данные вычислений приведены в табл. 2.1.

Таблица 2.1

Данные вычислений для полигонов *A* и *B*

Сторона полигона	Вектор, перпендикулярный стороне полигона	Хорда	Скалярное произведение вектора и хорды	Внутренняя нормаль
Данные вычислений для полигона A				
V_0V_1	$\overline{n_{V_oV_1}} = [0,3]$	$V_0V_2 = [5,3]$	$\overline{n_{V_oV_1}} \cdot \overline{V_0V_2} = 9$	$\bar{n} = [0,3]$
V_1V_2	$\overline{n_{V_1V_2}} = [-3,2]$	$V_1V_3 = [-1,6]$	$\overline{n_{V_1V_2}} \cdot V_1V_3 = 15$	$\bar{n} = [-3,2]$
V_2V_3	$\overline{n_{V_2V_3}} = [-3,-3]$	$V_2V_4 = [-6,1]$	$\overline{n_{V_2V_3}} \cdot V_2V_4 = 15$	$\bar{n} = [-3,-3]$
V_3V_4	$\overline{n_{V_3V_4}} = [2,-3]$	$V_3V_0 = [-2,-6]$	$\overline{n_{V_3V_4}} \cdot V_3V_0 = 14$	$\bar{n} = [2,-3]$
V_4V_0	$\overline{n_{V_4V_0}} = [4,1]$	$V_4V_1 = [4,-4]$	$\overline{n_{V_4V_0}} \cdot V_4V_1 = 12$	$\bar{n} = [4,1]$
Данные вычислений для полигона B				
V_0V_1	$\overline{n_{V_oV_1}} = [-3,3]$	$V_0V_2 = [6,0]$	$\overline{n_{V_oV_1}} \cdot \overline{V_0V_2} = -18$	$\bar{n} = [3,-3]$
V_1V_2	$\overline{n_{V_1V_2}} = [3,3]$	$V_1V_3 = [4,3]$	$\overline{n_{V_1V_2}} \cdot V_1V_3 = 21$	$\bar{n} = [3,3]$
V_2V_3	$\overline{n_{V_2V_3}} = [-6,1]$	$V_2V_4 = [-6,5]$	$\overline{n_{V_2V_3}} \cdot V_2V_4 = 41$	$\bar{n} = [-6,1]$
V_3V_4	$\overline{n_{V_3V_4}} = [1,-7]$	$V_3V_0 = [-7,-6]$	$\overline{n_{V_3V_4}} \cdot V_3V_0 = 35$	$\bar{n} = [1,-7]$
V_4V_0	$\overline{n_{V_4V_0}} = [5,0]$	$V_4V_1 = [3,-2]$	$\overline{n_{V_4V_0}} \cdot V_4V_1 = 15$	$\bar{n} = [5,0]$

2.3. Разбиение вогнутых полигонов

Вогнутый многоугольник можно разделить на набор выпуклых с помощью векторов сторон и их векторных произведений или, используя положение вершин относительно линии продолжения одной из сторон, определить, какие вершины находятся по одну сторону от этой линии, а какие – по другую. Для реализации данного алгоритма допустим, что многоугольник расположен в плоскости *xy*.

Для реализации *векторного метода* разделения вогнутого многоугольника сначала необходимо построить векторы сторон. Нахождение векторов сторон рассмотрено в подразд. 2.2.

Алгоритм разбиения вогнутого полигона

Шаг 1. Определить векторные произведения соседних векторов сторон по порядку по всему периметру многоугольника.

Шаг 2. Если знак одного из векторных произведений отрицателен – многоугольник вогнутый. Его необходимо разделить, продолжив первый вектор стороны из той пары векторов, произведение которых отрицательно, до его пересечения с одной из сторон многоугольника.

Конец алгоритма.

Пример. Для вогнутого полигона *B* (см. рис. 2.4, б) построим множество выпуклых.

Из имеющихся векторных произведений пар векторов, образующих соседние стороны, выберем пару векторов, произведение которых отрицательно – $V_{01} \times V_{12}$. Для разбиения вогнутого полигона необходимо продлить первый вектор из пары до пересечения с одной из сторон полигона в точке *M* (рис. 2.5).

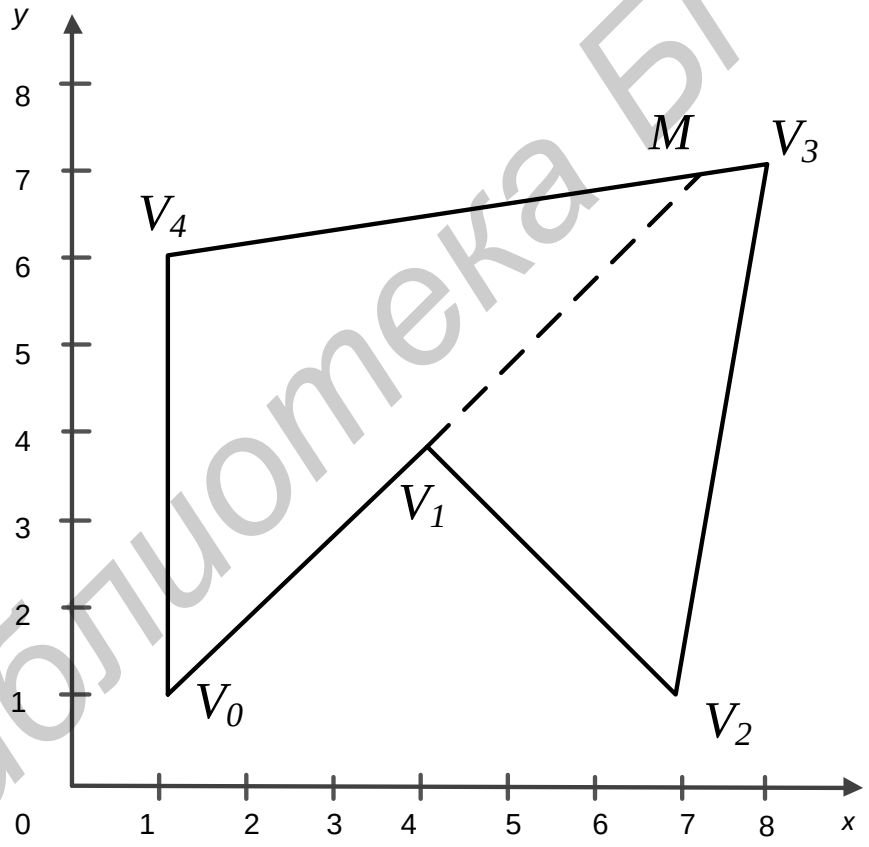


Рис. 2.5. Разбиение вогнутых полигонов

Найдем сторону полигона, пересекаемую вектором V_1M. Найдем уравнение прямой, содержащей в себе вектор V_1M. Затем классифицируем вершины полигона по признаку расположения (левее либо правее) относительно найденной прямой.

Находим уравнение прямой, содержащей вектор V_1M:

$\frac{x-1}{4-1} = \frac{y-1}{4-1}$;

$y - x = 0$ – уравнение прямой.

Подставляя значения координат вершин, проверяемых на положение относительно прямой, в уравнение прямой, разобьем вершины следующим образом:

$$\begin{cases} \text{ax} + \text{by} + \text{c} = 0 - \text{вершина лежит на прямой}, \\ \text{ax} + \text{by} + \text{c} > 0 - \text{вершина лежит правее прямой}, \\ \text{ax} + \text{by} + \text{c} < 0 - \text{вершина лежит левее прямой}. \end{cases}$$

Получим: вершины V_2 и V_3 лежат справа от прямой, вершины V_4 и V_0 – слева.

Следовательно, точка пересечения вектора V_1M и полигона будет лежать на стороне V_3V_4. Чтобы найти точку пересечения вектора и стороны V_1V_4 решим систему уравнений

$$\begin{cases} 7y - x - 41 = 0, \\ y = x. \end{cases}$$

Точка пересечения *M* (6.83, 6.83).

2.4. Алгоритмы построения выпуклой оболочки. Метод обхода Грэхема

Множество различных задач вычислительной геометрии связано с построением выпуклой оболочки. В настоящий момент эта задача хорошо исследована и имеет широкое применение в распознавании образов и обработке изображений.

Понятие выпуклой оболочки множества точек Q, обозначаемое $CH(Q)$, является естественным и простым. В соответствии с определением это наименьшее выпуклое множество, содержащее Q. Чтобы наглядно представить это понятие в случае, когда Q – конечное множество точек на плоскости, предположим, что это множество охвачено большой растянутой резиновой лентой. Когда лента освобождается, она принимает форму выпуклой оболочки.

Однако, несмотря на свою простоту, данное определение выпуклой оболочки не конструктивно, поэтому далее будут рассмотрены эффективные алгоритмы для построения выпуклой оболочки.

Сформулируем задачу. Задано конечное множество точек Q на плоскости. Требуется построить их выпуклую оболочку $CH(Q)$.

В методе обхода Грэхема задача о выпуклой оболочке решается с помощью списковой структуры S, организованной по принципу стека LIFO (Last Input, First Output – Последним пришел, Первым вышел), сформированного из точек-кандидатов. Все точки входного множества Q заносятся в стек, а потом точки, не являющиеся вершинами $CH(Q)$, со временем удаляются из него. По завершении работы алгоритма в стеке S остаются только вершины оболочки $CH(Q)$ в порядке их обхода против часовой стрелки, если просматривать их в стеке снизу вверх.

В методе обхода Грэхема выпуклая оболочка конечного набора точек находится в ***два этапа***. На первом этапе ***предварительной сортировки*** алгоритм выбирает экстремальную точку $p_0 \in Q$ с минимальной *y*-координатой. Остальные точки сортируются в порядке возрастания полярного угла относительно точки p_0. Если две точки имеют одинаковый полярный угол, то точка, расположенная ближе к точке p_0, должна помещаться в список раньше.

Вычислить данный угол можно с использованием *углового коэффициента прямой* – тангенса угла между отрезком и положительным направлением оси x $\operatorname{tg}\varphi = \frac{y_j - y_i}{x_j - x_i}$, где (x_i, y_i)–координаты точки, относительно которой вычисляется угол, а (x_j, y_j) – координаты точки-кандидата.

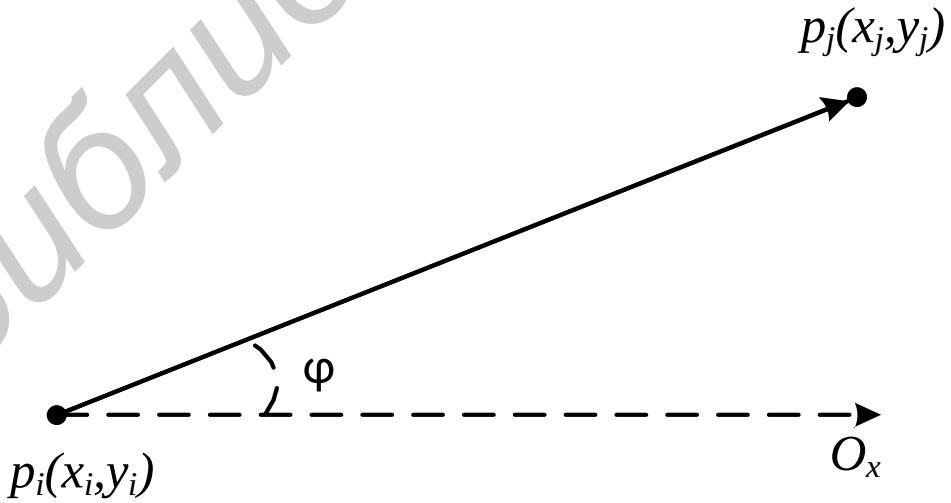


Рис. 2.6. Определение полярного угла точки p_j относительно p_i

Таким образом, *полярный угол точки* $p_j(x_j, y_j)$ относительно точки $p_i(x_i, y_i)$ равен $\text{arctg}\left(\frac{y_j - y_i}{x_j - x_i}\right)$.

На втором этапе построения выпуклой оболочки алгоритм выполняет *пошаговую обработку отсортированных точек*, формируя последовательность ребер, которые в конце образуют выпуклую оболочку $CH(Q)$.

Для определения того, какая именно точка должна быть включена в границу выпуклой оболочки после точки p_i, используется тот факт, что при обходе границы выпуклой оболочки в направлении против часовой стрелки каждая ее вершина должна соответствовать повороту влево.

В качестве входных данных алгоритм использует множество точек P_i, $i = \overline{0, n-1}$ на плоскости.

Алгоритм построения выпуклой оболочки методом обхода Грэхема

Шаг 1. Поиск экстремальной точки p_0.

Шаг 2. Сортировка остальных точек по их полярному углу относительно точки p_0.

Шаг 3. Инициализация текущей оболочки.

Шаг 4. Формирование текущей оболочки, пока она не станет равной выпуклой оболочке для всех точек множества P.

Шаг 5. Преобразование текущей оболочки в объект типа полигон.

Конец алгоритма.

Шаг 1 и шаг 2 алгоритма соответствуют первому этапу, в результате создается массив отсортированных точек по их полярному углу относительно экстремальной точки p_0.

На 3-м и 4-м шагах в списковую структуру итеративно добавляются вершины искомой выпуклой оболочки, что соответствует второму этапу. Завершается работа алгоритма 5-м шагом – преобразованием выпуклой оболочки в полигон.

Пример. Задано множество точек {(21, 14), (11, 8), (21, 18), (24, 23), (15, 20), (11, 14), (8, 23)}. Построить выпуклую оболочку$CH(Q)$.

1. Поиск экстремальной точки p_0, которой является точка с минимальным значением *y*-координаты, а если есть несколько точек с минимальной *y*-координатой, то выбираем из них точку с минимальной *x*-координатой: $p_0 = (11{,}8)$.

2. Сортируем точки по значению полярного угла относительно точки p_0 с тем условием, что если у двух точек одинаковое значение полярного угла, то раньше должна идти точка, находящаяся ближе к p_0.

Формула для нахождения полярного угла точки $p_j(x_j, y_j)$ относительно точки $p_i(x_i, y_i)$: $\operatorname{arctg}\left(\frac{y_j - y_i}{x_j - x_i}\right)$. *Отсортированные точки по полярному углу:* {(21, 14), (21, 18), (24, 23), (15, 20), (11, 14), (8, 23)}.

3. Инициализация текущей оболочки. Заносим точку p_0 и первую из отсортированных точек в стек:

{(11, 8), (21, 14)}.

4. Формирование текущей оболочки, пока она не станет равной выпуклой оболочке для всех точек множества *P*. Последовательно перебираем все отсортированные точки. Проверяем, в какой полуплоскости относительно двух последних точек лежит текущая точка, если слева, то заносим ее в стек. Иначе удаляем последнюю точку из стека и повторяем данный шаг.

Проверить, в какой полуплоскости лежит точка можно путем нахождения векторного произведения e. Если e больше 0, то точка в правой полуплоскости.

Формула для нахождения векторного произведения векторов ab и ac:

$$e = u_x * v_y - v_x * u_y,$$

где $u = \{b_x - a_x, b_y - a_y\}$, $v = \{c_x - a_x, c_y - a_y\}$.

5. Преобразование текущей оболочки в объект типа полигон (табл. 2.2).

Таблица 2.2

Метод обхода Грэхема

№	Текущая точка	e	Стек	Комментарий
1	(21, 18)	40	{(11, 8), (21, 14),(21, 18)}	Левый поворот
2	(24, 23)	–12	{(11, 8), (21, 14)}	Правый поворот. Удаляем последнюю точку
3	(24, 23)	72	{(11, 8), (21, 14),(24, 23)}	Левый поворот
4	(15, 20)	72	{(11, 8), (21, 14), (24, 23),(15, 20)}	Левый поворот
5	(11, 14)	42	{(11, 8), (21, 14), (24, 23), (15, 20),(11, 14)}	Левый поворот
6	(8, 23)	–54	{(11, 8), (21, 14), (24, 23), (15, 20)}	Правый поворот
7	(8, 23)	–48	{(11, 8), (21, 14), (24, 23)}	Правый поворот
8	(8, 23)	144	{(11, 8), (21, 14), (24, 23), (8, 23)}	Левый поворот. Так как текущая точка последняя, завершаем алгоритм

2.5. Алгоритмы построения выпуклой оболочки. Метод Джарвиса

При построении выпуклой оболочки множества точек Q путем обхода по Джарвису используется метод, известный как *заворачивание подарка*.

Интуитивно обход по Джарвису моделирует обертывание плотного куска бумаги вокруг множества Q. Начнем с того, что прикрепим конец бумаги к нижайшей точке множества, т. е. к той же точке p_0, с которой начинается сканирование по Грэхему. Эта точка является вершиной выпуклой оболочки. Натянем бумагу вправо, чтобы она не провисала, после чего переместим ее вверх до

тех пор, пока она не коснется какой-то точки. Эта точка также должна быть вершиной выпуклой оболочки. Сохраняя бумагу натянутой, продолжим наматывать ее на множество вершин, пока не вернемся к исходной точке p_0.

Алгоритм Джарвиса использует тот факт, что *отрезок l, определяемый двумя точками, является ребром выпуклой оболочки тогда и только тогда, когда все другие точки заданного множества лежат на l или с одной стороны от него.*

Алгоритм построения выпуклой оболочки методом Джарвиса

Входные данные: Множество точек P на плоскости.

Шаг 1. Поиск экстремальной точки p_0 (аналогично алгоритму Грэхема).

Шаг 2. Построение правой цепи: помещать в стек точку p_{i+1}, имеющую наименьший полярный угол относительно точки p_i, пока не будет достигнута наивысшая точка p_k (имеющая максимальную *y*-координату).

Шаг 3. Построение левой цепи: необходимо изменить на противоположные направления осей координат и иметь дело теперь с полярными углами, наименьшими относительно *отрицательного* направления оси x. Начиная с p_k, продолжить помещение в стек вершин, чей полярный угол наименьший относительно предыдущей. Продолжать, пока не будет достигнута исходная точка p_0.

Шаг 4. Преобразование текущей оболочки в объект типа полигон.

Конец алгоритма.

Пример. Построение выпуклой оболочки методом Джарвиса. Дано множество точек P: {(12, 4), (7, 10), (10, 20), (12, 12), (16, 12), (21, 16), (21,12)}.

Поиск экстремальной точки p_0, которой является точка с минимальным значением *y*-координаты, а если есть несколько точек с минимальной -координатой, то выбираем из них точку с минимальной *x*-координатой:

$$p_0 = 12, y = 4.$$

Построение правой цепи: ищем точку с минимальным значением полярного угла относительно точки последней занесенной в стек. Если найденная точка является точкой p_0, то завершаем алгоритм.

Формула для нахождения полярного угла точки $p_j(x_j, y_j)$ относительно точки $p_i(x_i, y_i)$: $\text{arctg}\left(\frac{y_j - y_i}{x_j - x_i}\right)$.

Построение левой цепи: если последняя занесенная в стек точка не является максимальной, то выполняем шаг 2. Иначе поворачиваем координатные оси в противоположное направление и повторяем шаг 2.

Преобразование текущей оболочки в объект типа полигон (табл. 2.3)

Таблица 2.3

Метод Джарвиса

№	Верхняя точка стека	Выбранная точка	Угол	Стек	Комментарий
1	(12, 4)	(21, 1)	41.63	{(12, 4), (21, 12)}	
2	(21, 12)	(21, 16)	90.0	{(12, 4), (21, 12),(21, 16)}	
3	(21, 16)	(10, 20)	160.0	{(12, 4), (21, 12), (21, 16), (10, 20)}	Меняем направление осей
4	(10, 20)	(7, 10)	73.3	{(12, 4), (21, 12), (21, 16), (10, 20),(7, 10)}	
5	(7, 10)	(12, 4)	129.8	{(12, 4), (21, 12), (21, 16), (10, 20), (7, 10)}	Выбранная точка совпала с p_0. Конец алгоритма

2.6. Нахождение точки пересечения отрезка со стороной полигона

Часто встречаемой на практике задачей является нахождение точки пересечения отрезка со стороной полигона.

Пусть задан отрезок P_1P_2 параметрическим уравнением прямой $P(t) = P_2 + (P_2 - P_1)t, 0 \le t \le 1, P_1, P_2$ – координаты концов отрезка. Известен также вектор $\overline{n_\alpha}$, перпендикулярный стороне a_i полигона S, ориентированный внутрь полигона (нахождение такого вектора рассмотрено в подразд. 2.2) и известна граничная точка F, инцидентная стороне a_i (рис. 2.7).

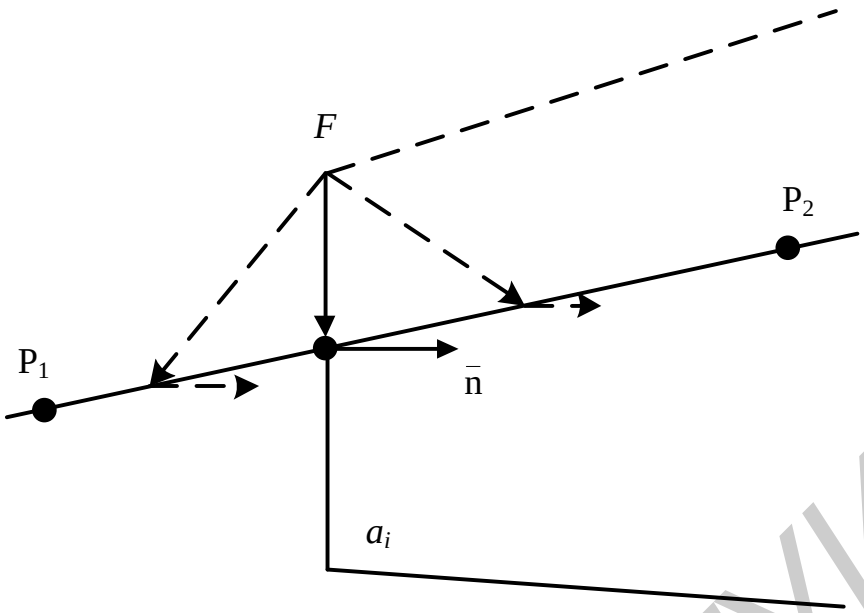


Рис. 2.7. Нахождение точки пересечения со стороной полигона

Скалярное произведение вектора нормали $\overline{n_\alpha}$ на вектор, начинающийся в произвольной точке отрезка и заканчивающийся в другой точке, лежащей на пересекаемой стороне:

$$\overline{n_\alpha} \cdot [P(t) - F] = \begin{cases} > 0 \text{ точка отрезка внутри полигона,} \\ = 0 \quad \text{точка отрезка на стороне,} \\ < 0 \quad \text{точка отрезка вне полигона.} \end{cases} \qquad (2.4)$$

Подстановка параметрического уравнения прямой в выражение (2.4) дает условие пересечения отрезка с границей области:

$\overline{n_\alpha} \cdot [P_1 + (P_2 - P_1)t - F] = 0,$

$\overline{n_\alpha} \cdot [P_1 - F] + \overline{n_\alpha} \cdot [P_2 - P_1]t = 0.$

Заметим, что вектор $P_2 - P_1$ определяет ориентацию отрезка, а вектор $P_1 - F$ пропорционален расстоянию от первого конца отрезка до избранной граничной точки. Обозначим через $D = P_2 - P_1$ директрису или ориентацию отрезка, через $W = P_1 - F$ – некий весовой множитель. Тогда уравнение можно записать так:

$t \cdot (\overline{n_\alpha} \cdot D) + \overline{n_\alpha} \cdot W = 0.$

Решая последнее уравнение относительно t, получим

$t = \frac{\overline{n_\alpha} \cdot W}{\overline{n_\alpha} \cdot D}, D \neq 0.$

Здесь $\overline{n_\alpha} \cdot D$ может быть равным нулю только при $D = 0$, а также параллельности D избранной границе, а это означает, что $P_2 = P_1$ – вырождение отрезка в точку.

$$\overline{n_\alpha} \cdot W = \begin{cases} > 0 \text{ точка отрезка внутри полигона,} \\ = 0 \quad \text{точка отрезка на стороне,} \\ < 0 \quad \text{точка отрезка вне полигона.} \end{cases}$$

2.7. Определение принадлежности точки полигону

Одной из классических задач вычислительной геометрии является определение принадлежности точки полигону.

Под *принадлежностью точки полигону* понимается принадлежность точки *внутренней части полигона или его границе*. В частности, *все вершины полигона принадлежат полигону*.

Для реализации алгоритма предложен *метод трассировки луча*.

Предположим, что нам необходимо определить принадлежность точки α полигону P. Для этого из некоторой удаленной точки проведем прямую линию в точку α. На этом пути может встретиться нуль или несколько пересечений границы полигона: при первом пересечении мы входим внутрь полигона, при втором – выходим из него, при третьем пересечении снова входим внутрь и так до тех пор, пока не достигнем точки α.

Таким образом, каждое нечетное пересечение означает попадание внутрь полигона P, а каждое четное – выход из него. Если мы попадаем в точку α с нечетным числом пересечений границы полигона, то точка α лежит внутри полигона, а если получается четное число пересечений, то точка находится вне полигона P. Например, на рис. 2.8 луч r_α пересекает границу только однажды, поскольку единица – число нечетное, то точка a находится внутри полигона. Относительно точки b мы придем к заключению, что она лежит вне полигона, поскольку луч r_b пересекает границу четное число раз (дважды).

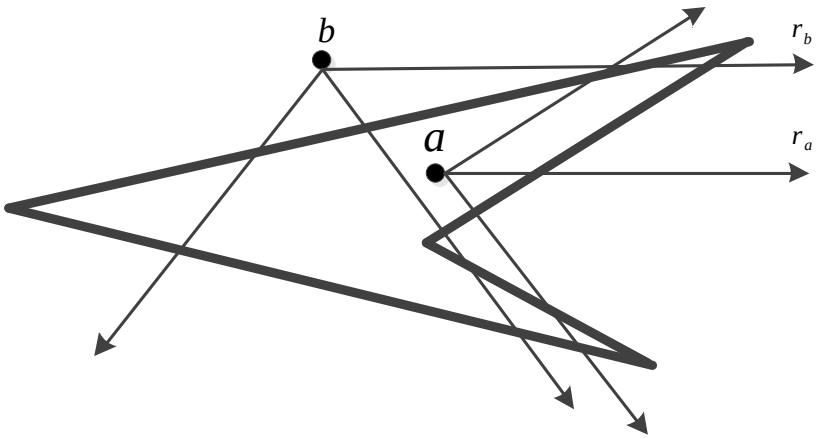


Рис. 2.8. Каждый луч, исходящий из точки α, пересекает границу нечетное число раз, а каждый луч, исходящий из точки b, имеет четное число пересечений с границей полигона

Построение алгоритма на основе трассировки луча базируется на двух особенностях:

1. Для решения задачи подходит любой луч, начинающийся в анализируемой точке a (см. рис. 2.8). Поэтому для простоты выбирается правый *горизонтальный луч* r_α, начинающийся в точке a и направленный вправо параллельно положительной полуоси O_x.

2. Порядок расположения ребер, пересекаемых лучом r_a, безразличен, имеет значение лишь *четность их общего числа*. Следовательно, для алгоритма достаточно определить все пересечения ребер в *любом порядке*, определяя четность по мере продвижения. Простейшим решением будет обход границы полигона с переключением бита четности при каждом обнаружении ребра, пересекаемого лучом r_α.

Относительно горизонтального луча r_α, направленного вправо, будем различать три типа ребер полигона:

– *касательное ребро*, содержащее точку a;

– *пересекающее ребро*, не содержащее точку a;

– *безразличное ребро*, не имеющее пересечения с лучом r_α (рис. 2.9)

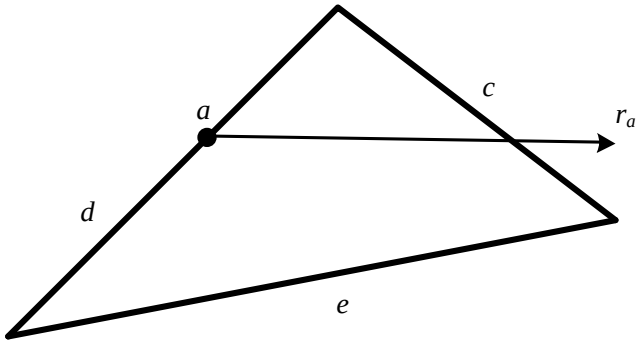


Рис. 2.9. Ребро c является пересекающим ребром, ребро d – касательное, ребро e – безразличное

Алгоритм определения принадлежности точки полигону

Входные данные: Точка a, произвольный полигон P.

Шаг 1. Обход границы многоугольника P. Для каждого ребра полигона определить его тип:

а) метод должен правильно обрабатывать особые ситуации, возникающие при прохождении луча r_α точно через вершины (рис. 2.10). В этих случаях устанавливаются особые правила переключения переменной, определяющей признак пересечения ребра лучом;

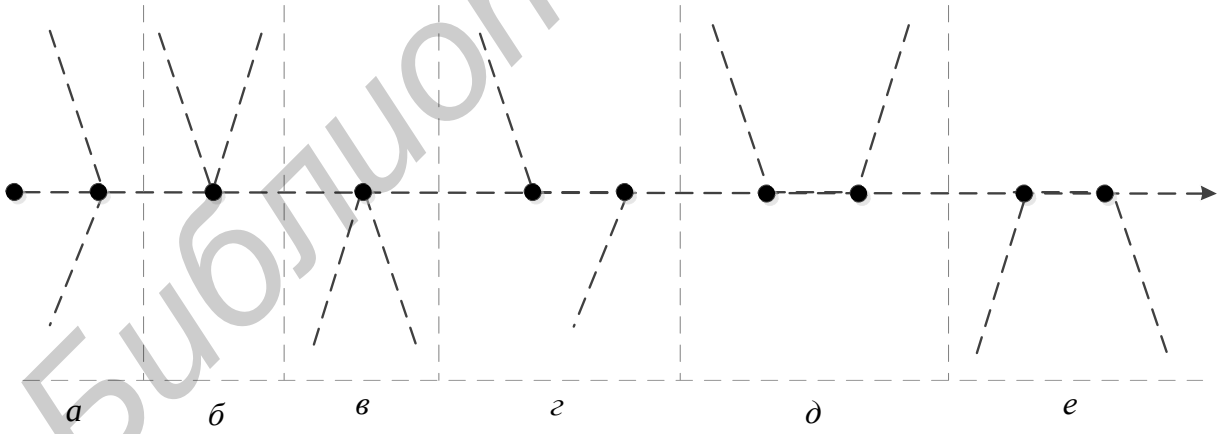


Рис. 2.10. Специальные случаи:
а, г – переменная переключается однажды; б, д – переменная не переключается;
в, е – переменная переключается дважды и ситуация остается без изменения

б) классифицировать ребро:

– *касательное ребро*, если оно содержит точку a;

– *пересекающее ребро*, если выполняются условия:

– ребро не горизонтальное;

– луч r_α пересекает ребро в некоторой точке, отличающейся от его нижней конечной точки;

– *безразличное*, если оно не касательное и не пересекающее.

Шаг 2. Определить четность переменной признака пересечения ребра лучом:

– если переменная *четная*: точка *не принадлежит* полигону;

– если переменная *нечетная*: точка *принадлежит* полигону.

Конец алгоритма.

Пример. Определить принадлежность точки C (2, 5) полигону B (см. рис. 2.4, б).

Через точку C проведем прямую l, параллельную оси x. Требуется определить количество пересечений прямой со сторонами полигона. Если количество четное – точка принадлежит полигону, иначе – точка лежит вне полигона.

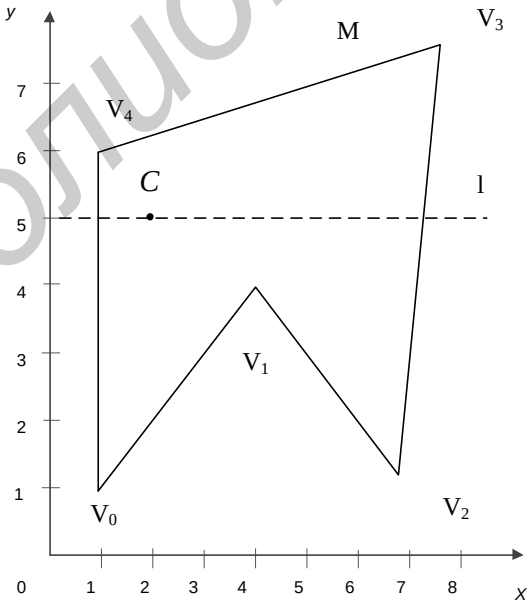


Рис. 2.11. Определение принадлежности точки полигону

Ограничим прямую l точками $P_1(1, 5)$ и $P_2(8, 5)$ – минимальная и максимальная координаты полигона B по x:

$P_a = P_1 + u_a(P_2 - P_1)$ – уравнение прямой l,

$P_b = V_0 + u_b(V_1 - V_0)$ – уравнение прямой, содержащей вектор $\overline{V_0V_1}$.

Решение относительно точки $P_a = P_b$ дает два уравнения на координаты (u_a и u_b).

Решая относительно u_a и u_b, имеем:

$$u_a = \frac{(x_4-x_3)(y_1-y_3)-(y_4-y_3)(x_1-x_3)}{(y_4-y_3)(x_2-x_1)-(x_4-x_3)(y_2-y_1)},$$

$$u_b = \frac{(x_2-x_1)(y_1-y_3)-(y_2-y_1)(x_1-x_3)}{(y_4-y_3)(x_2-x_1)-(x_4-x_3)(y_2-y_1)}.$$

Для того чтобы узнать, пересекаются ли отрезки, нужно проверить, лежат ли значения u_a и u_b на промежутке [0, 1]. Если хотя бы одна из этих переменных $0 \le u_i \le 1$, то соответствующий отрезок содержит точку пересечения.

Для стороны V_0V_1:

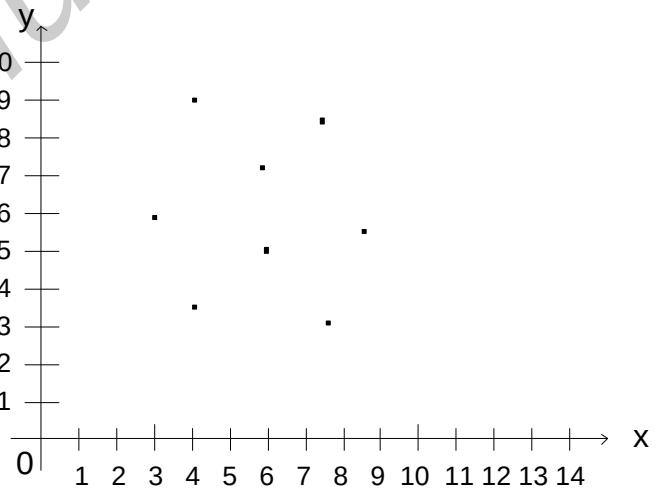
u_a= –1,33,

u_b= 1,33.

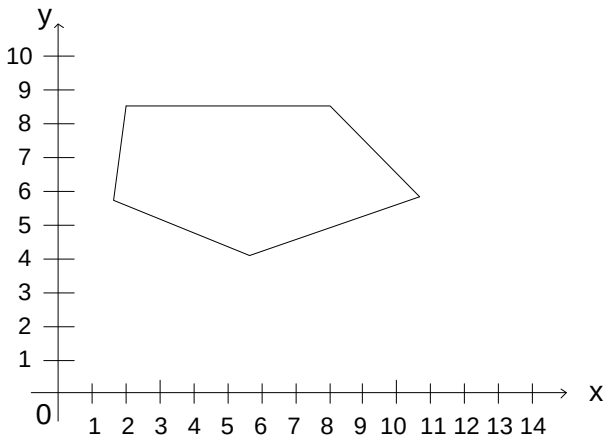
Следовательно, прямая l со стороной V_0V_1 пересечения не имеет. Далее требуется найти u_a и u_b для каждой из сторон полигона и определить положение точки C относительно полигона B.

Упражнения для самостоятельной работы

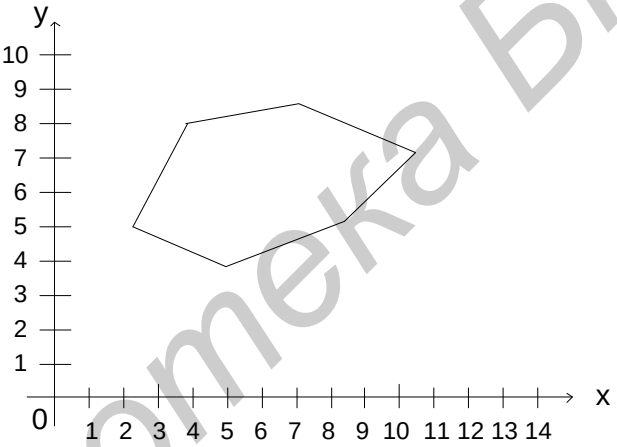
Задание 1. Произвести построение выпуклого полигона для заданного набора точек по алгоритму Грэхема и алгоритму Джарвиса.



Задание 2. Доказать, что показанный полигон является выпуклым и определить внутренние нормали для каждой из сторон.



Задание 3. Доказать, что показанный полигон является выпуклым и определить внутренние нормали для каждой из сторон.



Задание к лабораторной работе

Разработать элементарный графический редактор, реализующий построение полигонов. Реализованная программа должна уметь проверять полигон на выпуклось, находить его внутренние нормали. Программа должна выполнять построения выпуклых оболочек методом обхода Грэхема и методом Джарвиса. Выбор метода задается из пункта меню и должен быть доступен через панель инструментов «Построение полигонов». Графический редактор должен позволять рисовать линии первого порядка (лабораторная работа №1) и определять точки пересечения отрезка со стороной полигона, также программа должна определять принадлежность введенной точки полигону.

3. ЗАПОЛНЕНИЕ ПОЛИГОНОВ

Особенностью растрового графического устройства является возможность представления сплошных областей. Генерацию сплошных областей из простых описаний ребер или вершин будем называть ***растровой разверткой сплошных областей***, заполнением полигонов. Для этого можно использовать несколько методов, которые подразделяются на три группы методов:

– растровая развертка;

– затравочное заполнение;

– комбинированные методы, сочетающие преимущества методов растровой развертки и затравочного заполнения.

В методах растровой развертки пытаются определить в порядке сканирования строк, лежит ли точка внутри полигона или контура. Эти алгоритмы обычно идут от «верха» полигона к «низу».

Многие замкнутые контуры являются простыми полигонами. Если контур состоит из кривых линий, то его можно аппроксимировать подходящим многоугольником или многоугольниками. Простейший метод заполнения многоугольника состоит в проверке на принадлежность внутренности многоугольника каждого пиксела в растре. Этот метод слишком расточителен.

Можно разработать более эффективный метод, используя факт, что соседние пикселы, вероятно, имеют одинаковые характеристики (кроме пикселов граничных ребер). Это свойство называется пространственной когерентностью.

3.1. Алгоритмы растровой развертки

Данные алгоритмы используют свойство пространственной когерентности. Характеристики пикселов на сканирующей строке изменяются только там, где ребро многоугольника пересекает строку. Эти пересечения делят сканирующую строку на области.

Пример. Рассмотрим растровую развертку сплошной области, представленной на рис. 3.1.

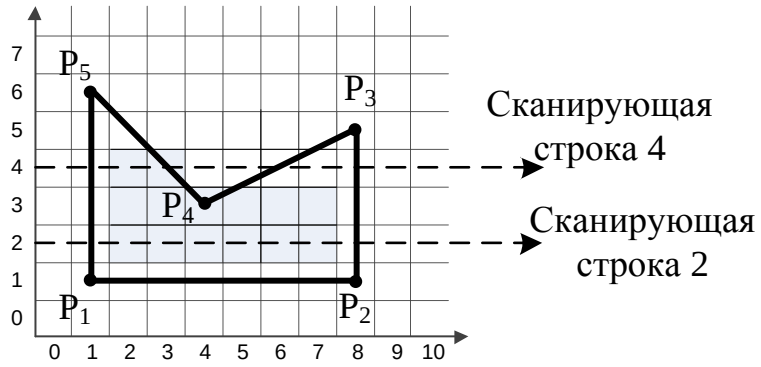


Рис. 3.1. Растровая развертка сплошной области

Для сканирующей строки 2 получаем три области (табл. 3.1).

Таблица 3.1

Области пересечения полигона со сканирующей строкой 2

Область	Расположение
$x < 1$	вне полигона
$1 \le x \le 8$	внутри полигона
$x > 8$	вне полигона

Для сканирующей строки 4 получаем пять областей (табл. 3.2):

Таблица 3.2

Области пересечения полигона со сканирующей строкой 4

Область	Расположение
$x < 1$	вне полигона
$1 \le x \le 3$	внутри полигона
$3 \le x \le 6$	вне полигона
$6 \le x \le 8$	внутри полигона
$x > 8$	вне полигона

Точки пересечения полигона со сканирующими строками не задаются в фиксированном порядке. Например, если многоугольник задается списком вершин, а список ребер последовательными парами вершин $P_1P_2, P_2P_3, P_3P_4, P_4P_5, P_5P_1$, то для строки 4 будут найдены следующие точки пересечения с ребрами многоугольника: 8, 6, 3, 1. Эти точки надо отсортировать в возрастающем порядке по x, т. е. получить 1, 3, 6, 8.

При определении интенсивности, цвета и оттенка пикселов на сканирующей строке рассматриваются пары отсортированных точек пересечения. Для каждого интервала, задаваемого парой пересечений, используется интенсивность или цвет заполняемого многоугольника. Для интервалов вне многоугольника используется цвет фона либо они вообще не отображаются.

Горизонтальные ребра не могут пересекать сканирующую строку, поэтому игнорируются. Дополнительная трудность возникает при пересечении сканирующей строки и многоугольника точно по вершине, как показано на рис. 3.2.

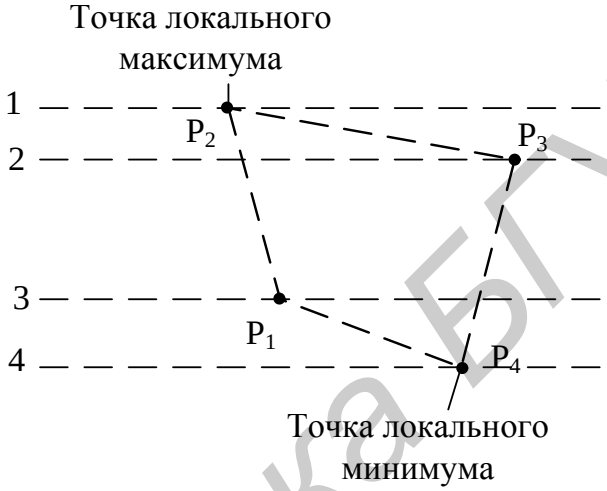


Рис. 3.2. Пересечение сканирующей строкой вершин полигона

Третья строка пересекается ребрами P_1P_2, P_1P_4, P_3P_4, в результате получается нечетное количество пересечений. Чтобы задать интервал необходимо слияние точек в вершине P_1 в одну точку. Для строки 1 слияние точек производить не нужно, т. к. в результате слияния останется одна точка и невозможно будет построить интервал.

Правильный результат можно получить, учитывая точку пересечения в вершине *два раза*, если она является точкой локального экстремума, и учитывать ее *один раз* в противном случае.

Алгоритм определения локальных экстремумов выпуклого полигона

Входные данные: множество вершин P полигона.

Шаг 1. Цикл по всем вершинам полигона, p_i – текущая вершина.

Шаг 2. Определить координаты концевых точек двух ребер, соединенных в вершине p_i.

Шаг 3. Если у обеих концевых точек координата y больше, чем у вершины p_i, значит, p_i – точка локального минимума, если у обеих координата y меньше, значит, p_i – точка локального максимума. Иначе точка p_i не является ни точкой локального максимума, ни точкой локального минимума.

Конец алгоритма.

Для рис. 3.2 точка P_2 – точка локального максимума, точка P_4 – точка локального минимума. Следовательно, в точках P_2 и P_4 учитываются два пересечения со сканирующими строками, а в P_1 и P_3 – одно.

3.2. Алгоритм растровой развертки с упорядоченным списком ребер

Используя описанные выше методы, можно разработать эффективные алгоритмы растровой развертки сплошных областей, называемые алгоритмами с упорядоченным списком ребер. Они зависят от сортировки в порядке сканирования точек пересечений ребер полигона со сканирующими строками. Эффективность этих алгоритмов зависит от эффективности сортировки.

Алгоритм растровой развертки с упорядоченным списком ребер

Шаг 1. Для каждого ребра многоугольника найти точки пересечения со сканирующими строками (можно использовать алгоритм Брезенхема). Занести каждое пересечение в общий список.

Шаг 2. Отсортировать полученный список по следующему пу: (x_1, y_1) предшествуют (x_2, y_2), если $y_1 < y_2$ или $y_1 = y_2$ и $x_1 \leq x_2$.

Шаг 3. Выделить из списков пары пересечений, т. е. получить интервалы для закраски.

Конец алгоритма.

Пример. Произвести закраску области, представленной на рис. 3.3, методом растровой развертки с упорядоченным списком ребер.

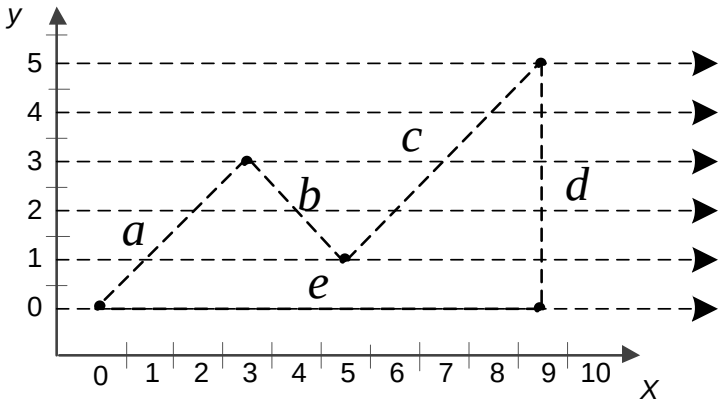


Рис. 3.3. Растровая развертка с упорядоченным списком ребер

1. Найдем точки пересечения ребер со сканирующими строками:

Ребро a: $(0, 0)^{min}$ (1, 1) (2, 2) $(3, 3)^{max}$.

Ребро b: $(5, 1)^{min}$ (4, 2) $(3, 3)^{max}$.

Ребро c: $(5, 1)^{min}$ (6, 2) (7, 3) (8, 4) $(9, 5)^{max}$.

Ребро d: $(9, 0)^{min}$(9, 1) (9, 2) (9, 3)(9, 4) $(9, 5)^{max}$.

Ребро e: не рассматривается.

2.Сортировка полученного списка:

(0, 0) (9, 0) (1, 1) (5, 1) (5, 1) (9, 1) (2, 2) (4, 2) (6, 2) (9, 2) (3, 3) (3, 3) (7, 3) (9, 3) (8, 4) (9, 4) (9, 5) (9, 5).

3. Выделение интервалов для закраски:

[(0, 0)(9, 0)] [(1, 1) (5, 1)] [(5, 1)(9, 1)] [(2, 2) (4, 2)] [(6, 2) (9, 2)] [(3, 3) (3, 3)] [(7, 3)(9, 3)] [(8, 4)(9, 4)][(9, 5) (9, 5)].

3.3. Алгоритм растровой развертки с упорядоченным списком ребер, использующий список активных ребер

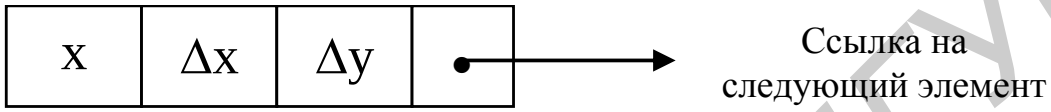
В алгоритме растровой развертки с упорядоченным списком ребер генерируется большой список, который необходимо полностью отсортировать. Алгоритм можно улучшить, если повысить эффективность этой сортировки, которая достигается благодаря использованию связного списка – добавочной структуры данных. Предварительное вычисление пересечения каждой сканирующей строки каждым ребром многоугольника требует больших вычислительных затрат и значительных объемов памяти. Введя список активных ребер (САР), можно со-

кратить потребность в памяти и вычислять пересечения со сканирующими строками в пошаговом режиме.

Алгоритм растровой развертки с упорядоченным списком ребер, использующий список активных ребер

Шаг 1. Подготовка данных: для каждого ребра многоугольника определяется наивысшая по оси *y* сканирующая строка, пересекающая это ребро. Ребро заносится в *y*-группу, соответствующую этой сканирующей строке. Внутри *y*-группы ребра отсортированы по возрастанию координаты x.

Описание ребра в *y*-группе представляется в следующем виде:



Информационные поля элемента списка *y*-группы:

x – координата пересечения со сканирующей строкой;

Δx – шаг приращения по *x* от одной сканирующей строки к другой;

Δy – число сканирующих строк, пересекаемых ребром многоугольника.

Шаг 2. Формирование изображения начинается с максимальной по y сканирующей строки. САР для текущей сканирующей строки формируется добавлением информации из *y*-группы, соответствующей этой строке.

2а. Для текущей сканирующей строки проверить соответствующую *y*-группу на наличие новых ребер. Если такие есть, то добавить их в САР.

2б. Отсортировать координаты *x* точек пересечения САР в порядке возрастания.

2в. Выделить пары точек (интервалы для закраски), активизировать на сканирующей строке группы пикселей.

2г. Для каждого ребра из САР уменьшить Δy на 1. Если для какого-то отрезка получилось $\Delta y \leq 0$, то исключить его из списка активных ребер.

2д. Вычислить новое значение координат x точек пересечения: $x_{\text{нов}} = x_{\text{стар}} + + \Delta x$.

2е. Перейти к следующей сканирующей строке (см. п. 2а).

Конец алгоритма.

Пример. Произвести закраску области, представленной на рис. 3.3, методом растровой развертки с упорядоченным списком ребер, использующим список активных ребер.

Сформируем массив *y*-групп сканирующих строк и отсортируем ребра по возрастанию координаты *x* (рис. 3.4).

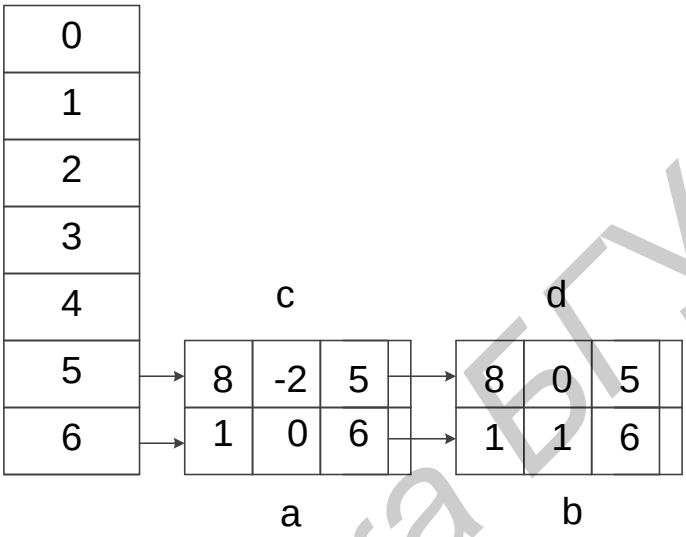


Рис.3.4. Массив сканирующих строк

Поскольку алгоритм обрабатывает сканирующие строки, начиная с максимальной, из рис. 3.5 видно, что для ребра *b* шаг приращения Δx положителен, а для ребра *c* – отрицателен.

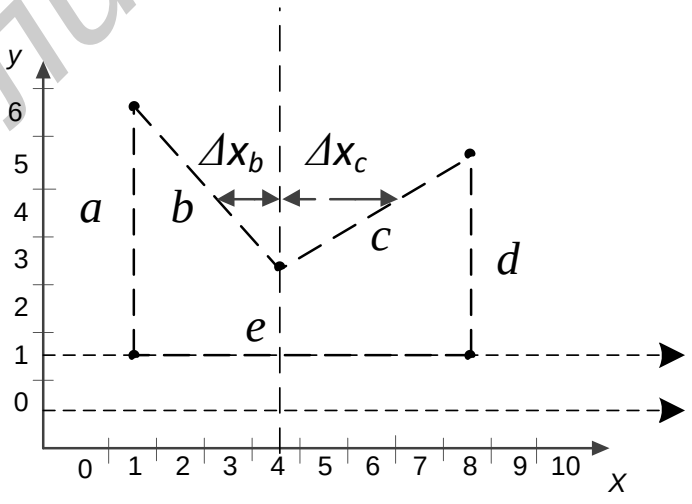


Рис. 3.5. Шаги приращения для ребер *b* и *c* при сканировании сверху вниз

Сформируем список активных ребер для каждой сканирующей строки (табл. 3.3).

Таблица 3.3

Результаты работы алгоритма растровой развертки с упорядоченным списком ребер, использующего САР

Сканирующая строка	Число отрезков	Отрезки в САР											
		a			*b*			*c*			*d*		
		x	Δ*x*	Δ*y*	*x*	Δ*x*	Δ*y*	*x*	Δ*x*	Δ*y*	*x*	Δ*x*	Δ*y*
7	0	-	-	-	-	-	-	-	-	-	-	-	-
6	2	**1**	0	**6**	**1**	1	**6**	-	-	-	-	-	-
5	4	**1**	0	**5**	**2**	1	**5**	**8**	-2	**5**	**8**	0	**5**
4	4	**1**	0	**4**	**3**	1	**4**	**6**	-2	**4**	**8**	0	**5**
3	4	**1**	0	**3**	**4**	1	**5**	**4**	-2	**3**	**8**	0	**3**
2	2	**1**	0	**2**	-	-	-	-	-	-	**8**	0	**2**
1	2	**1**	0	**1**	-	-	-	-	-	-	**8**	0	**1**
0	0	-	-	-	-	-	-	-	-	-	-	-	-

Таким образом, пройдя по всем сканирующим строкам и выделив пары интервалов, можно получить следующие результаты для закраски:

[(1, 6) (1, 6)]; [(1, 5) (2, 5)]; [(8, 5) (8, 5)]; [(1, 4) (3, 4)]; [(6, 4) (8, 4)]; [(1, 3) (4, 3)]; [(4, 3) (8, 3)]; [(1, 2) (8, 2)]; [(1, 1) (8, 1)].

3.4. Алгоритмы заполнения с затравкой

В методах затравочного заполнения предполагается, что известна некоторая точка (затравка) внутри замкнутого контура. В алгоритмах ищут точки, соседние с затравочной точкой и расположенные внутри контура. Если соседняя точка расположена не внутри контура, то значит, обнаружена граница контура. Если же точка расположена внутри контура, то она становится затравочной точкой и поиск продолжается рекурсивно. Подобные алгоритмы применимы только к растровым устройствам.

Данному классу алгоритмов можно сопоставить физическую интерпретацию, а именно представить, что в затравочной точке помещен источник, зали-

вающий всю область определенным цветом. Поэтому часто такие алгоритмы называют алгоритмами заливки.

В них предполагается, что известен хотя бы один пиксел из внутренней области многоугольника. Алгоритм пытается найти и закрасить все другие пикселы, принадлежащие внутренней области.

Области делятся на:

– внутренне-определенные – все пикселы, *принадлежащие внутренней части*, имеют один и тот же цвет или интенсивность;

– гранично-определенные – все пикселы *на границе* области имеют выделенное значение или цвет (рис. 3.6).

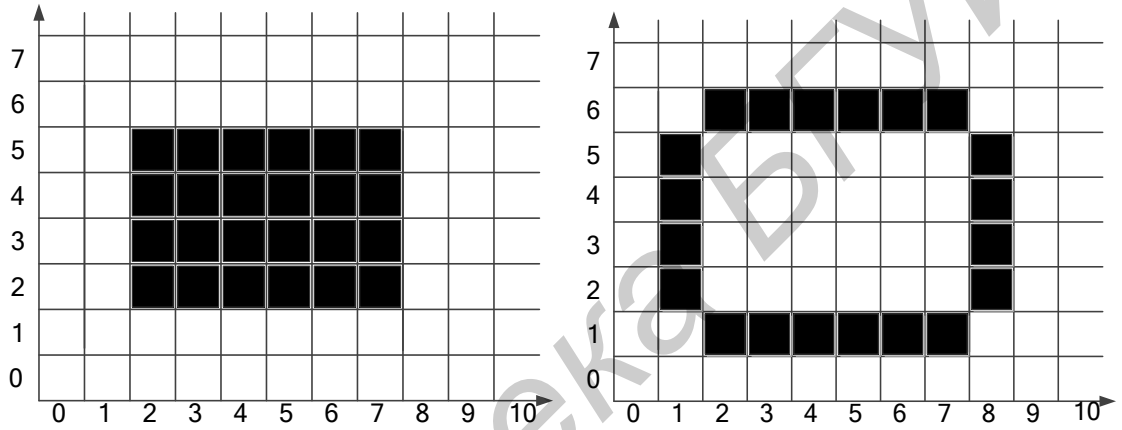
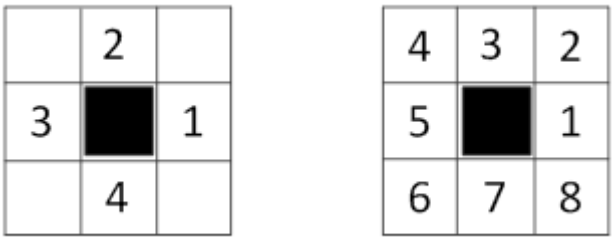


Рис. 3.6. Внутренне-определенная и гранично-определенная области

Внутренне-определенные и гранично-определенные области могут быть:

– 4-связные – любой пиксел в области можно достичь с помощью комбинаций движений только в 4-х направлениях: налево, направо, вверх, вниз;

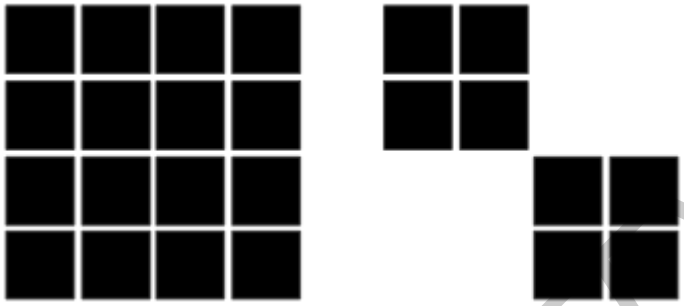
– 8-связные – любой пиксел в области можно достичь с помощью комбинаций движений в 8-и направлениях (добавляются еще и диагональные направления) (рис. 3.7).



а б

Рис. 3.7. 4-связные (а) и 8-связные (б) области

Алгоритм заполнения 8-связной области заполнит 4-связную, однако обратное неверно. На рис. 3.7 показаны простые примеры 4- и 8-связных областей. Хотя каждая из подобластей 8-связной области на рис. 3.7, б является 4-связной, для перехода из одной области в другую требуется 8-связный алгоритм. Однако в ситуации, когда требуется заполнить разными цветами две отдельные 4-связные области, использование 8-связного алгоритма вызовет неправильное заполнение обеих областей одним и тем же цветом.



а б

Рис. 3.8. Примеры 4-связных (а) и 8-связных (б) областей

3.5. Простой алгоритм заполнения с затравкой

Используя стек, можно разработать простой алгоритм заполнения граничноопределенной области. Стек – это просто массив или другая структура данных, в которую можно последовательно помещать значения и из которой их можно последовательно извлекать. Когда новые значения добавляются в стек, все остальные значения опускаются вниз на один уровень.

Алгоритм простой заполнения с затравкой

Шаг 1. Поместить затравочный пиксел в стек.

Пока стек не пуст.

Шаг 2. Извлечь пиксел из стека.

Шаг 3. Закрасить его.

Шаг 4. Для каждого из соседних к текущему 4-связных пикселов проверить: является ли он граничным пикселом или не закрашен ли он уже. Проигнорировать пиксел в любом из этих случаев. В противном случае поместить в стек.

Конец алгоритма.

Пример. В качестве примера применения алгоритма рассмотрим полигонную область, представленную на рис. 3.9.

Пусть затравочный пиксел – (3, 5). Область заполняется пиксел за пикселом в порядке, указанном на рис. 3.9. Пикселы, помеченные *, закрашиваются повторно, т. к. при работе алгоритма некоторые пикселы помещаются в затравочный стек многократно.

После извлечения из стека затравочного пиксела (3, 5) закрашиваем его. Проверяем каждый из соседних по отношению к нему 4-связных пикселов. Если пиксел не является граничным и не закрашен, то помещаем его в стек. Проверка 4-связных пикселов осуществляется в порядке, указанном на рис. 3.10.

Для первого затравочного пиксела (3, 5) такими пикселами будут: (3, 6), (2, 5), (3, 4) и (4, 5). На следующем шаге извлекаем из вершины стека пиксел (3, 6), закрашиваем его. 4-связными к нему будут пикселы: (3, 7), (2, 6). Помещаем их в стек.

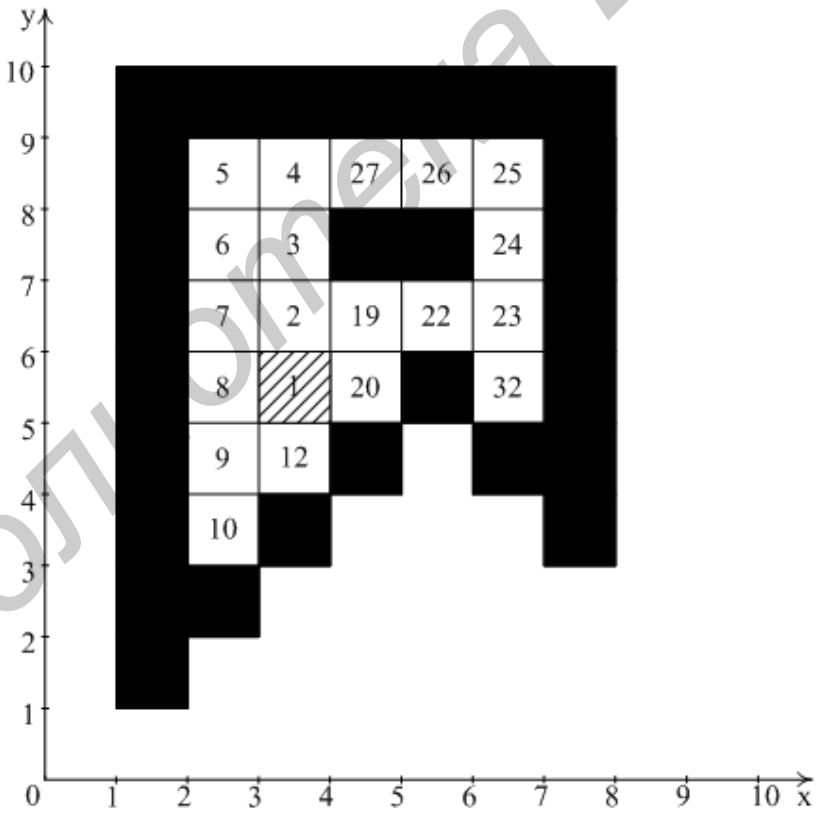


Рис. 3.9. Простой алгоритм заполнения с затравкой

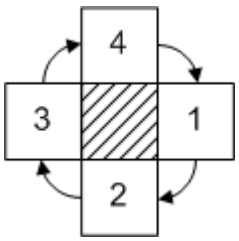


Рис. 3.10. Порядок проверки 4-связных пикселов

Полный список шагов и результат на каждом из них представлен в табл. 3.4. Алгоритм заканчивает свою работу, когда в стеке не остается ни одного пиксела.

Таблица 3.4

Состояние стека вершин на каждом из шагов алгоритма заполнения с затравкой

Шаг	**Закрашиваемый пиксел**	**Стек затравочных пикселов**
1	(3,5)	
2	(3,6)	(2,5) (3,4) (4,5)
3	(3,7)	(2,6) (4,6) (2,5) (3,4) (4,5)
4	(3,8)	(2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
5	(2,8)	(2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
6	(2,7)	(2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
7	(2,6)	(2,5) (2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
8	(2,5)	(2,4) (2,5) (2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
9	(2,4)	(2,3) (3,4) (2,4) (2,5) (2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
10	(2,3)	(2,3) (3,4) (2,4) (2,5) (2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
11	(2,3)*	(3,4) (2,4) (2,5) (2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
12	(3,4)	(2,4) (2,5) (2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
13	(2,4)*	(2,5) (2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
14	(2,5)*	(2,6) (2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
15	(2,6)*	(2,7) (2,7) (2,6) (4,6) (2,5) (3,4) (4,5)

Окончание табл. 3.4

Шаг	Закрашиваемый пиксел	Стек затравочных пикселов
16	(2,7)*	(2,7) (2,6) (4,6) (2,5) (3,4) (4,5)
17	(2,7)*	(2,6) (4,6) (2,5) (3,4) (4,5)
18	(2,6)*	(4,6) (2,5) (3,4) (4,5)
19	(4,6)	(4,5) (5,6) (2,5) (3,4) (4,5)
20	(4,5)	(4,5) (5,6) (2,5) (3,4) (4,5)
21	(4,5)*	(5,6) (2,5) (3,4) (4,5)
22	(5,6)	(6,6) (2,5) (3,4) (4,5)
23	(6,6)	(6,7), (6,5) (6,6) (2,5) (3,4) (4,5)
24	(6,7)	(6,8) (6,7) (6,5) (6,6) (2,5) (3,4) (4,5)
25	(6,8)	(5,8) (6,8) (6,7) (6,5) (6,6) (2,5) (3,4) (4,5)
26	(5,8)	(4,8) (5,8) (6,8) (6,7) (6,5) (6,6) (2,5) (3,4) (4,5)
27	(4,8)	(4,8) (5,8) (6,8) (6,7) (6,5) (6,6) (2,5) (3,4) (4,5)
28	(4,8)*	(5,8) (6,8) (6,7) (6,5) (6,6) (2,5) (3,4) (4,5)
29	(5,8)*	(6,8) (6,7) (6,5) (6,6) (2,5) (3,4) (4,5)
30	(6,8)*	(6,7) (6,5) (6,6) (2,5) (3,4) (4,5)
31	(6,7)*	(6,5) (6,6) (2,5) (3,4) (4,5)
32	(6,5)	(6,6) (2,5) (3,4) (4,5)
33	(6,6)*	(2,5) (3,4) (4,5)
34	(2,5)*	(3,4) (4,5)
35	(3,4)*	(4,5)
36	(4,5)*	

3.6. Комбинированный метод. Построчный алгоритм заполнения с затравкой

Недостатком группы алгоритмов развертки, использующих построчное заполнение, является то, что в процессе их выполнения необходимо использовать сортировку.

В алгоритмах затравочного заполнения стек может быть довольно большим, а также содержать дублирующуюся информацию.

Возможно применение комбинированных методов, использующих преимущества затравочных и алгоритмов построчного заполнения.

В построчном алгоритме заполнения с затравкой стек минимизируется за счет хранения только затравочного пиксела для любого непрерывного интервала на сканирующей строке. Непрерывный интервал – это группа примыкающих друг к другу пикселов (ограниченная уже заполненными или граничными пикселами). Данный алгоритм применим к гранично-определенным 4-связным областям, которые могут быть как выпуклыми, так и не выпуклыми, а также могут содержать дыры.

Алгоритм построчный заполнения с затравкой

Шаг 1. В стек помещается затравочный пиксел.

Пока стек не пуст.

Шаг 2. Из стека извлекается очередной затравочный пиксел.

Шаг 3. Слева и справа от пиксела заполняется интервал, пока не встретится граница.

Шаг 4. Запоминаются крайний левый $X_{лев}$ и крайний правый $X_{прав}$ пикселы интервала.

Шаг 5. В диапазоне $X_{лев} < x < X_{прав}$ проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется, есть ли еще в них незаполненные пикселы. Если такие пикселы есть (т. е. не все пикселы граничные, или уже заполненные), то в указанном диапазоне крайний правый пиксел отмечается как затравочный и помещается в стек.

Конец алгоритма.

Пример. В качестве примера применения алгоритма рассмотрим полигонную область, представленную на рис. 3.11.

Пусть затравочный пиксел – (8, 6). Он извлекается из стека затравочных пикселов, слева и справа от него заполняется интервал, пока не встретится граница. Таким интервалом в данном примере будут являться пикселы [(6, 6) (9, 6)].

Далее в этом интервале осуществляется проверка строк $y = 5$ и $y = 7$. Поскольку в них содержатся незаполненные пикселы, в затравочный стек помещаются 2 крайних пиксела интервалов из строк 5 и 7: (9, 5) и (9, 7).

Состояние стека на каждом из шагов показано в табл. 3.5.

На 6-м шаге в стек помещаются 3 пиксела: пиксел (4, 6) из интервала [(2, 6) (4, 6)] строки $y = 6$, а также пикселы (9, 8) и (6, 8) из интервалов [(9, 9) (9, 9)] и [(2, 8) (6, 8)] строки $y = 8$ соответственно.

Обратите внимание, что пиксел (6, 8) находится в стеке дважды, это произошло из-за особенностей области, которую мы заполняем, а именно группы пикселов: (7, 8), (7, 9), (8, 9) и (8, 8).

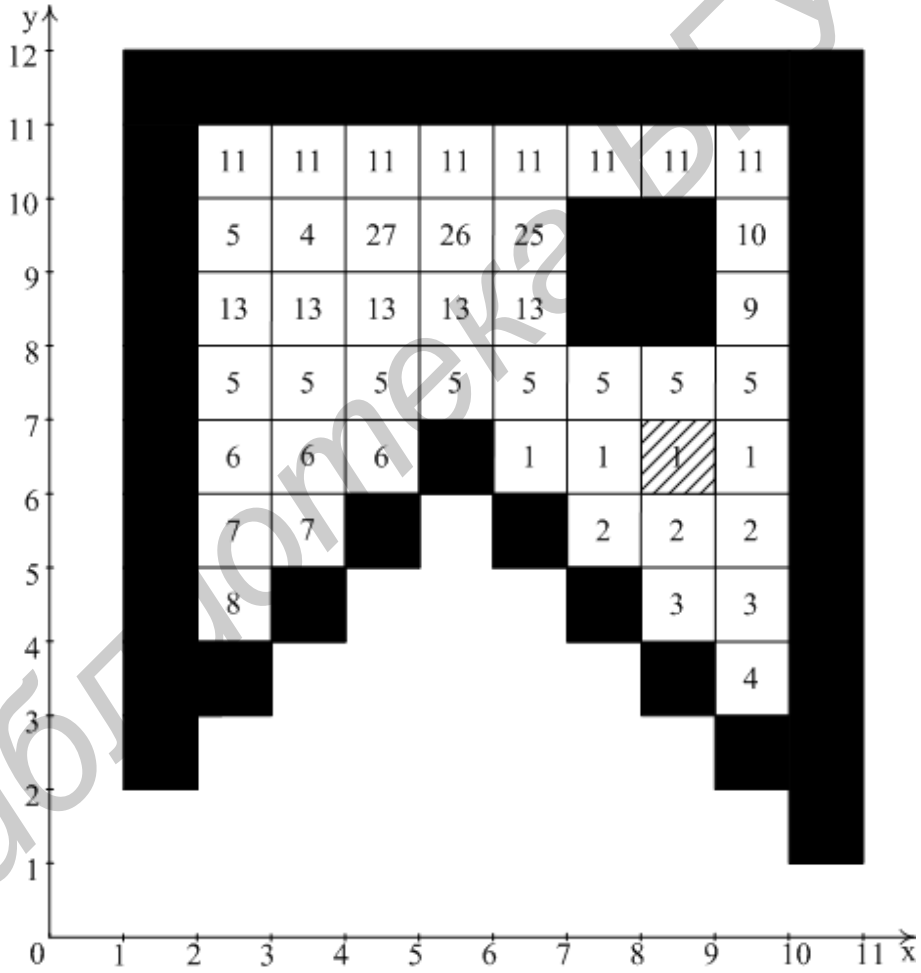


Рис. 3.11. Комбинированный метод

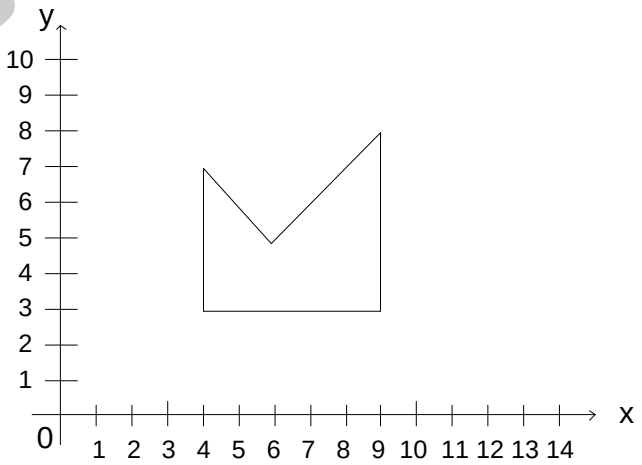
Таблица 3.7

Состояние стека затравочных пикселов и интервалы закраски на каждом из шагов построчного алгоритма заполнения с затравкой

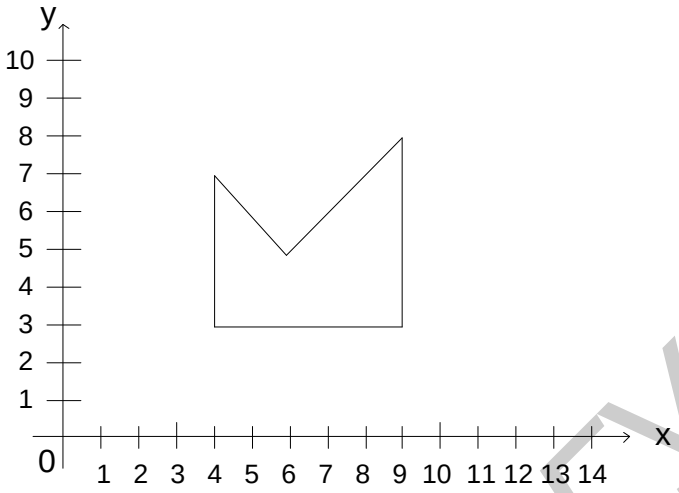
Шаг	**Стек затравочных пикселов**	**Интервал закраски**
1	(8,6)	[(6,6)(9,6)]
2	(9,5), (9,7)	[(7,5)(9,5)]
3	(9,4), (9,7)	[(8,4)(8,5)]
4	(9,3), (9,7)	[(9,3)(9,3)]
5	(9,7)	[(2,7)(9,7)]
6	(4,6), (9,8) ,(6,8)	[(2,6)(4,6)]
7	(3,5), (9,8) ,(6,8)	[(2,5)(3,5)]
8	(2,4), (9,8) ,(6,8)	[(2,4)(2,4)]
9	(9,8) ,(6,8)	[(9,8)(9,8)]
10	(9,9) ,(6,8)	[(9,9)(9,9)]
11	(9,10) ,(6,8)	[(2,10)(9,10)]
12	(6,9) ,(6,8)	[(2,9)(6,9)]
13	(6,8) ,(6,8)	[(2,8)(6,8)]
14	(6,8)	-

Упражнения для самостоятельной работы

Задание 1. Произвести закраску представленной области методом растровой развертки с упорядоченным списком ребер. Записать интервалы для закраски.



Задание 2. Произвести закраску представленной области методом заполнения с затравкой. Записать содержимое стека затравочных пикселов на каждой итерации алгоритма. Затравочный пиксел – (4, 5).



Задание к лабораторной работе

Разработать элементарный графический редактор, реализующий построение полигонов и их заполнение, используя алгоритм растровой развертки с упорядоченным списком ребер; алгоритм растровой развертки с упорядоченным списком ребер, использующий список активных ребер; простой алгоритм заполнения с затравкой; построчный алгоритм заполнения с затравкой. Выбор алгоритма задается из пункта меню и доступен через панель инструментов «Алгоритмы заполнения полигонов». В редакторе должен быть предусмотрен режим отладки, где отображается пошаговое решение.

ЛИТЕРАТУРА

1. Аммерал, Л. Машинная графика на персональных компьютерах / Л. Аммерал; пер. с англ. – М. : Сол Систем, 1992. – 232 с.
2. Аммерал, Л. Принципы программирования в машинной графике / Л. Аммерал; пер. с англ. – М. : Сол Систем, 1992. – 224 с.
3. Блинова, Т. А. Компьютерная графика : учеб. пособие / Т. А. Блинова, В. Н. Порев ; под ред. В. Н. Порева. – Киев : Юниор, 2006. – 520 с.
4. Гилой, В. Интерактивная машинная графика : структуры данных, алгоритмы, языки / В. Гилой ; пер. с англ. – М. : Мир, 1981. – 384 с.
5. Кравченя, Э. М. Компьютерная графика : учеб. пособие / Э. М. Кравченя, Т. И. Абрагимович. – Минск : Новое знание, 2006. – 248 с.
6. Павлидис, Т. Алгоритмы машинной графики и обработка изображений / Т. Павлидис ; пер. с англ. – М. : Радио и связь, 1986.– 400 с.
7. Петров, М. Н. Компьютерная графика : учеб. пособие для вузов / М. Н. Петров, В. П. Молочков. – СПб. : Питер, 2002. – 735 с.
8. Поляков, А. Ю. Методы и алгоритмы компьютерной графики в примерах на Visual C++ / А. Ю. Поляков, В. А. Брусенцев. – 2-е изд. – СПб. : BHV-Петербург, 2003. – 545 с.
9. Пореев, В. Н. Компьютерная графика / В. Н. Пореев. – СПб. : БХВ-Петербург, 2002. – 432 с.
10. Рейнбоу, В. Компьютерная графика / В. Рейнбоу ; пер. с англ. – СПб. : Питер, 2003. – 768 с.
11. Роджерс, Д. Алгоритмические основы машинной графики / Д. Роджерс ; пер. с англ. – М. : Мир, 1989. – 512 с.
12. Роджерс, Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс ; пер. с англ. – М. : Мир, 2001. – 604 с.
13. Шикин, А. В. Компьютерная графика. Полигональные модели / А. В. Шикин, А. В. Боресков. – М. : Диалог-МИФИ, 2000. – 464 с.
14. Фень, Юань Программирование графики для Windows / Юань Фень ; пер. с англ. – СПб. : Питер, 2002. – 1072 с.
15. Фоли, Дж. Основы интерактивной машинной графики. В 2 т. / Дж. Фоли, А. Ван Дэм. – М. : Мир, 1985.

Св. план 2010, поз. 36

Учебное издание

Самодумкин Сергей Александрович
Степанова Маргарита Дмитриевна
Колб Дмитрий Григорьевич

ПРАКТИКУМ ПО КОМПЬЮТЕРНОЙ ГРАФИКЕ

В 3-х частях

Часть 2

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

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

Подписано в печать . Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. . Уч.-изд. л. 4,5. Тираж 100 экз. Заказ 429.

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.
2200013, Минск, П. Бровки, 6