

Министерство науки и высшего образования  
Российской Федерации  
Ярославский государственный университет  
им. П. Г. Демидова  
Кафедра математического моделирования

**Алгоритмы компьютерной графики.  
Лабораторные работы**

**Практикум**

Ярославль  
ЯрГУ  
2022

УДК 004.92 (076.5)  
ББК 3973.2-018.3я73  
А 45

*Рекомендовано  
редакционно-издательским советом ЯрГУ в качестве учебного  
издания. План 2022 года.*

Рецензент  
кафедра математического моделирования  
Ярославского государственного университета  
им. П. Г. Демидова

Составитель Н. Д. Елисеева

А 45 **Алгоритмы компьютерной графики. Лабораторные работы** : практикум / сост. Н. Д. Елисеева; Яросл. гос. ун-т им. П. Г. Демидова. — Ярославль, 2022 — 36 с.

Практикум знакомит студентов с такими основными понятиями и алгоритмами компьютерной графики, как алгоритмы растеризации, удаления невидимых линий, заполнения областей и построения выпуклых оболочек.

Предназначен для студентов, изучающих дисциплину «Компьютерная графика».

УДК 004.92 (076.5)  
ББК 3973.2-018.3я73

© ЯрГУ, 2022

# Введение

Задача компьютерной графики — визуализация, то есть создание изображения. Визуализация выполняется, исходя из описания (модели) того, что нужно отображать. Существует много методов и алгоритмов визуализации, которые различаются между собой в зависимости от того, что и как отображать. Например, отображение того, что может быть только в воображении человека — график функций, диаграмма, схема, карта. Или наоборот, имитация трёхмерной реальности — изображения сцен в компьютерных развлечениях, художественных фильмах, тренажёрах, в системах архитектурного проектирования. Важными и связанными между собой факторами здесь являются скорость изменения кадров, насыщенность сцены объектами, качество изображения, учёт особенностей графического устройства.

Наиболее известны два способа визуализации: растровый и векторный. Первый способ ассоциируется с такими графическими устройствами, как дисплей, телевизор, принтер. Второй используется в векторных дисплеях, плоттерах. Мы будем рассматривать растровый.

Растровая визуализация основывается на представлении изображения на экране или бумаге в виде совокупности отдельных точек (пикселей). Вместе пиксели образуют растр. Каждый пиксель может иметь свой цвет. Совокупность пикселей различного цвета образует изображение. В зависимости от расположения пикселей в пространстве различают квадратный, прямоугольный, гексагональный растры или иные типы. Для описания расположения пикселей используют разнообразные системы координат. Общим для всех таких систем является то, что координаты пикселей образуют дискретный ряд значений (необязательно целые числа). Мы будем использовать прямоугольную систему координат, причём каждый пиксель имеет целочисленные координаты.

# 1. Задачи вычислительной геометрии

Первая лабораторная работа посвящена задачам вычислительной геометрии, основным ее объектам (точка, прямая, отрезок, плоскость и т. д.), способам задания этих объектов и алгоритмам для решения геометрических задач.

## 1.1. Лабораторная работа 1

Лабораторная работа состоит из четырёх заданий, латинская буква определяет вариант. В первом и втором заданиях точки заданы на плоскости, а в третьем и четвёртом — в пространстве (если не указано иное).

1. (a) Две точки заданы своими координатами. Составить уравнение прямой, проходящей через эти точки.
- (b) Точки  $A, B, C$  заданы своими координатами. Найти уравнение прямой, проходящей через точку  $C$ , и параллельной прямой  $AB$ .
- (c) Точки  $A, B, C$  заданы своими координатами. Найти уравнение прямой, проходящей через точку  $C$ , и перпендикулярной прямой  $AB$ .
- (d) Определить, принадлежит ли точка прямой. Точка задана координатами, прямая — коэффициентами.
- (e) Три точки заданы своими коэффициентами. Определить, лежат ли они на одной прямой.
- (f) Определить положение двух точек относительно прямой (по одну сторону от прямой, по разные стороны, на прямой). Прямая задана своими коэффициентами, точки — координатами.
- (g) Две прямые заданы своими коэффициентами. Определить их взаимное расположение (пересекаются, не пересекаются, совпадают).
- (h) Точки  $A, B, C, D$  заданы своими координатами. Определить взаимное расположение прямых  $AB$  и  $CD$  (пересекаются, не пересекаются, совпадают).

2. (a) Точки  $A, B, C$  заданы своими координатами. Определить, принадлежит ли точка  $C$  отрезку  $AB$ .
- (b) Точки  $A, B, C$ , лежащие на одной прямой, заданы своими координатами. Определить расположение точки  $C$  относительно отрезка  $AB$  (между точками  $A$  и  $B$ , вне отрезка за точкой  $A$ , вне отрезка за точкой  $B$ ).
- (c) Точки  $A, B, C$ , лежащие на одной прямой, заданы своими координатами. Определить, какая из них лежит между двумя другими.
- (d) Точки  $A, B, C$  заданы своими координатами. Определить, принадлежит ли точка  $C$  лучу  $AB$ .
- (e) Точки  $A, B, C, D$ , лежащие на одной прямой, заданы своими координатами. Определить, пересекаются ли отрезки  $AB$  и  $CD$ .
- (f) Точки  $A, B, C, D$ , лежащие на одной прямой, заданы своими координатами. Определить, пересекаются ли лучи  $AB$  и  $CD$ .
- (g) Точки  $A, B, C, D$ , лежащие на одной прямой, заданы своими координатами. Определить, пересекаются ли луч  $AB$  и отрезок  $CD$ .
- (h) Точки  $A, B, C$ , лежащие на одной прямой, заданы своими координатами. Определить расположение точки  $C$  относительно луча  $AB$ .
3. (a) Точки  $A, B, C$  заданы своими координатами. Определить вид угла  $ABC$  (острый, тупой или прямой).
- (b) Точки  $A, B, C$  заданы своими координатами. Определить вид треугольника  $ABC$  (остроугольный, тупоугольный или прямоугольный).
- (c) Точки  $A, B, C, D$  заданы своими координатами. Перпендикулярны ли прямые  $AB$  и  $CD$ ?
- (d) Точки  $A, B, C$  заданы своими координатами на плоскости. Определить, является ли обход  $ABC$  обходом по часовой стрелке или против.

- (e) Точки  $A, B, C$  заданы своими координатами. Найти площадь треугольника  $ABC$ .
  - (f) Точки  $A, B, C, D$  заданы своими координатами. Выясните, лежит ли точка  $D$  внутри угла  $ABC$ ? Имеется в виду тот из углов  $ABC$ , который замечается лучом при повороте от  $BA$  к  $BC$  против часовой стрелки.
  - (g) Точки  $A, B, C$  заданы своими координатами. Определить вид угла  $ABC$  (острый, тупой или прямой).
  - (h) Точки  $A, B, C$  заданы своими координатами на плоскости. Определить, является ли обход  $ABC$  обходом по часовой стрелке или против.
4. (a) Составить уравнение плоскости, проходящей через три точки  $(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$ .
- (b) Точки  $A, B, C, D$  заданы своими координатами. Найти уравнение плоскости, проходящей через точку  $D$ , и параллельной плоскости  $ABC$ .
  - (c) Точки  $A, B, C, D, E$  заданы своими координатами. Найти уравнение плоскости, проходящей через точки  $D$  и  $E$ , и перпендикулярной плоскости  $ABC$ .
  - (d) Заданы коэффициенты уравнения плоскости и координаты точки. Определить, принадлежит ли точка плоскости.
  - (e) Четыре точки заданы своими координатами. Определить, лежат ли они в одной плоскости.
  - (f) Определить положение двух точек относительно плоскости (по одну сторону от плоскости, по разные стороны, на плоскости). Плоскость задана своими коэффициентами, точки — координатами.
  - (g) Три точки в пространстве заданы своими координатами. Определить, лежат ли они на одной прямой.
  - (h) Определить, принадлежит ли данная точка прямой, заданной двумя плоскостями. Плоскости заданы своими коэффициентами, точка — координатами.

## Комментарии к заданию

1. Каждое задание должно быть оформлено отдельной процедурой или функцией.
2. Входные данные читаются из файла, а результат работы программы выводится в консоль.

## 2. Геометрические преобразования

Для того чтобы представить изображение на экране компьютера, необходим способ описания объектов на плоскости. Под описанием объекта будем понимать информацию о положении каждой точки объекта на плоскости в любой момент времени. Для этого удобно использовать декартову систему координат.

Декартовы координаты точек позволяют описывать статичное положение объектов в пространстве. Однако для проведения каких-либо действий над объектами необходимо иметь дополнительные математические конструкции. В качестве одной из таких конструкций используют радиус-векторы. Радиус-векторы обладают всеми свойствами векторов, но имеют одну особенность: начало радиус-вектора находится в начале координат, а конец радиус-вектора — некоторая точка плоскости.

### Двумерные матричные преобразования

Рассмотрим преобразования координат точек на плоскости. Будем считать, что в результате преобразования точка  $A$  с координатами  $(x, y)$  переходит в точку  $A'$  с координатами  $(x', y')$ .

### Перенос

Операция переноса представляет из себя перемещение точки  $A$  в точку  $A'$ . Математически этот перенос можно описать с помощью вектора переноса  $\overline{AA'}$ . Пусть  $\overline{R} = (R_x, R_y)$  радиус-вектор, соответствующий этому вектору переноса. Тогда переход из точки  $A$  в точку  $A'$  будет соответствовать векторной записи  $\overline{A'} = \overline{A} + \overline{R}$ , т. е.

для переноса точки в новое положение необходимо добавить к ее координатам некоторые числа — координаты вектора переноса.

## Отражение относительно осей

Рассмотрим отражение относительно оси  $OX$ . При таком преобразовании меняется знак ординаты исходной точки, т. е. в матричном виде это отражение выглядит так:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

## Масштабирование

Масштабированием объектов называется растяжение объектов вдоль соответствующих осей координат относительно начала координат. Эта операция применяется к каждой точке объекта путём умножения координат точек на некоторые константы. Пусть  $M_x$ ,  $M_y$  — коэффициенты масштабирования. Тогда для точек  $A$  и  $A'$  операция масштабирования в матричном виде будет выглядеть следующим образом:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} M_x & 0 \\ 0 & M_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

## Поворот

Рассмотрим далее операцию вращения точки на некоторый угол  $\alpha$  относительно начала координат, т. е. радиус-вектор  $\overline{A'}$  получается из радиус-вектора  $\overline{A}$  путём его поворота на угол  $\alpha$  против часовой стрелки. Тогда в матричном виде вращение точки  $A$  на угол  $\alpha$  выглядит следующим образом:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$



## Однородные координаты и матричное представление двумерных преобразований

Среди рассмотренных матричных преобразований только одна — операция переноса — не может быть записана в виде произведения матрицы на двумерный вектор. Вместе с тем легко видеть, что для любой последовательности операций, записанных в матричном виде, можно вычислить результирующую матрицу преобразования, вычислив произведение матриц используемых преобразований.

Очевидно, что удобнее применять результирующую матрицу, вместо того чтобы каждый раз заново вычислять произведение матриц. Однако таким способом нельзя получить результирующую матрицу преобразования, если среди последовательности преобразований присутствует хотя бы один перенос. Было бы удобнее иметь математический аппарат, позволяющий включать в композиции преобразований все из указанных выше операций. При этом получился бы значительный выигрыш в скорости вычислений. Однородные координаты и есть этот математический аппарат.

Двумерный вектор  $(x, y)$  в однородных координатах записывается в виде  $(x, y, 1)$ . Теперь точки двумерного пространства будут описываться трёхэлементными вектор-столбцами, поэтому и матрицы преобразований, на которые будет умножаться вектор точки, будут иметь размеры  $3 \times 3$ .

Запишем матричное преобразование операции переноса для однородных координат:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & R_x \\ 0 & 1 & R_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Запишем матричный вид операции масштабирования.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Для операции поворота матричный вид будет такой:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Легко заметить, что любая последовательность из описанных операций в однородных координатах может быть представлена одной матрицей, которая является произведением матриц данных операций. С вычислительной точки зрения гораздо проще и быстрее применять матрицу уже готового преобразования, вместо того чтобы применять их последовательно одну за другой.

Для примера рассмотрим задачу поворота точки на плоскости относительно некоторой произвольной точки  $B$ . Пока мы умеем поворачивать объекты только вокруг начала координат, поэтому эту задачу можно представить как последовательность элементарных операций:

1. Перенос, при котором точка  $B$  переходит в начало координат.
2. Поворот на заданный угол относительно нового начала координат.
3. Перенос, при котором точка из начала координат возвращается в первоначальное положение точки  $B$ .

## 2.1. Лабораторная работа 2

Лабораторная работа состоит из двух частей. Первая часть посвящена преобразованию геометрических фигур на плоскости, вторая — созданию анимации на основе таких преобразований.

### Первая часть

Реализовать с заданной совокупностью фигур (по вариантам, изображенным на рис. 1) преобразования:

- перенос вдоль оси  $OX$ ,
- перенос вдоль оси  $OY$ ,

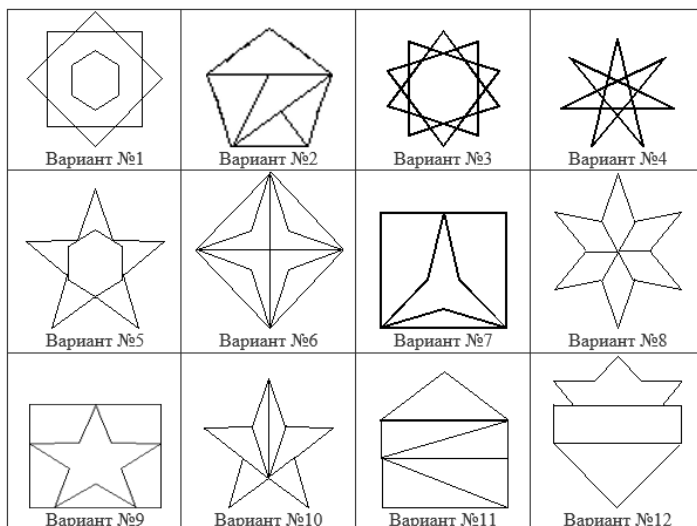


Рис. 1. Варианты первой части лабораторной работы 2

- отражение относительно оси  $OX$ ,
- отражение относительно оси  $OY$ ,
- отражение относительно прямой  $Y = X$ ,
- масштабирование независимо по обеим осям,
- поворот на заданный угол относительно центра координат
- поворот на заданный угол относительно произвольной точки, указываемой в ходе выполнения программы.

## Вторая часть

На основе первой части создать программу, работающую с двухмерными объектами по вариантам.

1. Написать программу, показывающую колесо со спицами, катящееся по наклонной поверхности.

2. Написать программу, имитирующую механические часы.
3. Разработать программу, показывающую полет вращающегося бумеранга.
4. Разработать программу, имитирующую падение листа с дерева.
5. Разработать программу, имитирующую падение снежинок.
6. Написать программу, показывающую падающие фигуры «Тетриса»; при нажатии на клавиши осуществляется поворот фигур и их окончательное падение.
7. Разработать программу, отображающую три взаимосвязанных вращающихся шестерни.
8. Разработать программу, имитирующую игру в настольный теннис (вид сверху).
9. Разработать программу, отображающую пульсирующее сердце.
10. Разработать программу, отображающую летящий самолёт с вращающимся винтом.
11. Написать программу, выводющую на экран шагающего человека.
12. Написать программу, выводющую на экран взлетающую ракету. С удалением от земли ракета уменьшается.

## **Комментарии к заданию**

### **Первая часть**

1. Управление организовать через меню, кнопки и т. д.
2. Предусмотреть восстановление исходной позиции фигур.
3. Предусмотреть применение нескольких последовательных преобразований.

4. Начало координат должно быть расположено в центре окна.
5. Обязательным является использование однородных координат.
6. Метод для умножения матрицы на вектор должен быть оформлен отдельной процедурой.

### **3. Алгоритмы растеризации отрезка и окружности**

Большинство графических устройств являются растровыми, представляя изображение в виде прямоугольной матрицы (сетки, целочисленной решетки) пикселей (растра), и большинство графических библиотек содержат внутри себя достаточное количество простейших растровых алгоритмов.

Сформировать растровое изображение можно по-разному. Для того чтобы создать изображение на растровом дисплее, можно просто скопировать готовый растр в видеопамять. Этот растр может быть получен, например, с помощью сканера или цифрового фотоаппарата. А можно создавать изображение объекта путём последовательного рисования отдельных простых элементов.

Простые элементы, из которых складываются сложные объекты, будем называть графическими примитивами. Их можно встретить повсюду. Например, для построения изображения используется некоторый набор примитивов, которые поддерживаются определенными графическими устройствами вывода. Графические примитивы также можно применять для описания пространственных объектов в базе данных компьютерной системы — модели объектов. Могут использоваться различные множества примитивов для модели и для алгоритмов отображения. Удобно, когда эти множества совпадают, тогда упрощается процесс отображения.

Простейшим и вместе с тем наиболее универсальным растровым графическим примитивом является пиксель. Любое растровое изображение можно нарисовать по пикселям, но это сложно и долго. Необходимы более сложные элементы, для которых рисуются сразу несколько пикселей. В современных графических системах в ка-

честве таких элементов наиболее часто используются графические примитивы — линии и фигуры.

Рассмотрим растровые алгоритмы для отрезков прямой линии. Предположим, что заданы координаты концов отрезка прямой. Для вывода линии необходимо закрасить в определённый цвет все пиксели вдоль линии. Для того чтобы закрасить каждый пиксель, необходимо знать его координаты. Простейший алгоритм, позволяющий это сделать? предполагает использование уравнения прямой, содержащей заданный отрезок и непосредственное вычисление одной из координат на основе другой или вычисление текущей координаты из предыдущей путём ее увеличения на угловой коэффициент. Однако за такой простотой скрывается использование чисел с плавающей точкой и арифметических операций над ними. Кроме погрешности представления самих чисел в памяти компьютера, будет накапливаться ошибка при последовательных вычислениях.

Д. Э. Брезенхэм предложил подход, позволяющий разрабатывать алгоритмы растеризации, основанные на построении циклов вычисления координат. При этом используются только операции сложения и вычитания целых чисел без использования операций умножения и деления. Описание алгоритма растеризации отрезка представлено в [1, пункты 2.3 — 2.5], окружности — в [1, пункт 2.6].

### 3.1. Лабораторная работа 3

#### Первая часть

Реализовать алгоритм Брезенхэма растеризации **отрезка**.

*Входные данные:* целочисленные координаты концов отрезка (в координатах сетки).

*Выходные данные:* изображение сетки, узлы которой являются пикселями; изображение отрезка стандартным методом; результат работы алгоритма в виде отмеченных пикселей.

#### Вторая часть

Реализовать алгоритмы Брезенхэма растеризации **окружности**.

*Входные данные:* радиус и координаты центра окружности.

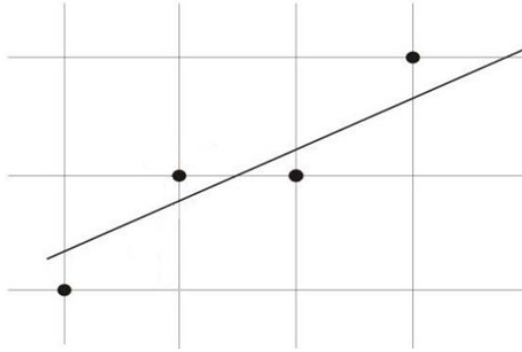


Рис. 2. Алгоритм Брезенхема растеризации отрезка

*Выходные данные:* изображение сетки, узлы которой являются пикселями; изображение окружности стандартным методом; результат работы алгоритма в виде отмеченных пикселей.

## Комментарии к заданию

Все вычисления должны выполняться в целых числах.

Примерный вид работы программы для первой части изображён на рис. 2.

## 4. Алгоритмы отсечения отрезка

На практике графические объекты всегда отображаются на конечном растре, границы которого соответствуют границам экрана. Растеризация на конечном растре требует возможности отсечения растеризуемого объекта относительно границ растра, т. е. удаления частей растеризуемого объекта, лежащих за пределами растра. Выполнение алгоритма растеризации без предварительного отсечения приведёт к ошибке при попытке осуществить изменение цвета пикселя с координатами за пределами растра (это может привести к изменению цвета не того пикселя или даже к системному сбою).

Можно было бы просто проверять, лежит ли пиксель внутри растра непосредственно перед изменением его цвета. К сожалению,

такое решение в большинстве случаев слишком неэффективно. Во-первых, подобная проверка сама по себе чрезвычайно замедлит растеризацию. Во-вторых, в тех случаях, когда значительная часть объекта лежит вне раstra, растеризация для этой области будет выполняться вхолостую и отсечение подобных областей ускорит вычисления.

Мы рассматриваем алгоритмы отсечения отрезка относительно границ прямоугольника или многоугольника (иногда мы будем называть этот прямоугольник или многоугольник «окном»).

Если изображение выходит за пределы экрана, то на части дисплеев увеличивается время построения за счёт того, что изображение строится в «уме». В некоторых дисплеях выход за пределы экрана приводит к искажению картины, так как координаты просто ограничиваются при достижении ими граничных значений, но не выполняется точный расчёт координат пересечения (эффект «стягивания» изображения). Некоторые, в основном простые, дисплеи просто не допускают выхода за пределы экрана. Все это, особенно в связи с широким использованием технологии просмотра окнами, требует выполнения отсечения сцены по границам окна видимости.

Отсекаемые отрезки могут быть трёх классов: целиком видимые, целиком невидимые и пересекающие окно. Очевидно, что целесообразно более рано, без выполнения большого объёма вычислений, принять решение о видимости отрезка целиком или об его отбрасывании. По способу выбора решения об отбрасывании невидимого отрезка целиком или принятия его существуют два основных типа алгоритмов отсечения: алгоритмы, использующие кодирование концов отрезка или всего отрезка, и алгоритмы, использующие параметрическое представление отсекаемых отрезков и окна отсечения. Представители первого типа алгоритмов — алгоритм Сазерленда — Коэна и алгоритм средней точки. Представителем алгоритмов второго типа является алгоритм Цируса — Бека.

Алгоритмы с кодированием применимы для прямоугольного окна, стороны которого параллельны осям координат, в то время как алгоритмы с параметрическим представлением применимы для произвольного окна.

Полное описание алгоритмов Сазерленда — Коэна, средней точки и Цируса — Бека можно найти в [1, пункты 3.2, 3.3, 3.5].



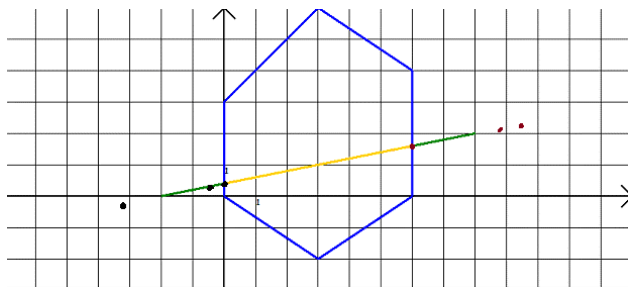


Рис. 3. Алгоритм Цируса — Бека

#### 4.1. Лабораторная работа 4

Лабораторная работа состоит из трёх частей:

1. Реализовать алгоритм Цируса — Бека отсечения отрезка многоугольником.
2. Реализовать алгоритм Сазерленда — Коэна.
3. Реализовать алгоритм средней точки.

Для каждой части:

*Входные данные:* координаты вершин многоугольника (прямоугольника), координаты концов отрезка.

*Выходные данные:* изображение многоугольника (прямоугольника), изображение отрезка, отмечены все потенциальные точки входа и выхода (для первой части), часть отрезка внутри многоугольника (прямоугольника) выделена другим цветом.

#### Комментарии к заданию

1. Входные данные читаются из файла.
2. Начало координат расположено в центре окна программы.

Примерный вид работы программы для первой части изображён на рис. 3

## 5. Алгоритмы заполнения областей

Ранее мы рассмотрели алгоритмы отображения линий на растре. Линии на растре зачастую образуют замкнутые области, которые требуется закрасить (заполнить) тем или иным цветом. Теперь мы рассмотрим алгоритмы, позволяющие выполнить такое заполнение.

Будут рассмотрены два случая. В первом требуется нарисовать на растре многоугольник, заданный своими вершинами (растеризация многоугольника). Во втором, более общем, случае мы будем считать, что растровая область задана цветом своей границы и точкой (пикселем) внутри области. Этот пиксель называется затравочным, а сам метод называется заполнением с затравкой.

### Растеризация многоугольников

Пусть задан многоугольник  $P_1P_2 \dots P_N$  и требуется растеризовать его вместе с внутренними точками. Будем считать, что многоугольник целиком помещается в растровом окне. Для удобства каждое ребро многоугольника будем задавать координатами  $(x_1, y_1)$  и  $(x_2, y_2)$  его концов, так что  $y_1 \leq y_2$ . Условимся также отсчитывать на экране координату  $x$  слева направо, а  $y$  — сверху вниз (таким образом, точка  $(x_1, y_1)$  будет верхним концом ребра, а  $((x_2, y_2)$  — нижним). Большинство алгоритмов заполнения основано на том факте, что любое горизонтальное сечение контура многоугольника состоит из чётного числа точек. Это утверждение неверно в двух случаях (см. рис. 4):

- когда секущая прямая содержит горизонтальное ребро;
- когда она содержит вершину, а оба смежных ребра лежат выше (ниже) ее.

Однако существуют простые способы исключить эти случаи из рассмотрения; они будут рассмотрены позднее. Пока же будем считать, что приведённое утверждение справедливо всегда. Задача заполнения многоугольника, таким образом, сводится к заполнению

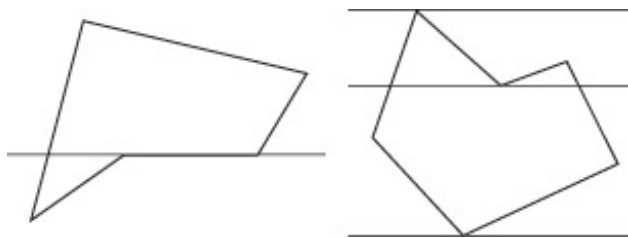


Рис. 4. Исключительные случаи

определённых промежутков между точками сечения, что должно быть проделано для каждого горизонтального сечения многоугольника. Заметим, что при таком подходе никаких ограничений на свойства многоугольника (выпуклость, отсутствие самопересечений и т. д.) не накладывается.

### Алгоритм со списком рёберных точек

Этот алгоритм состоит из трёх основных этапов.

На первом этапе растеризуются все негоризонтальные ребра многоугольника. Все точки помещаются в списки. Для каждой координаты  $y_{min}$ ,  $y_2$ ,  $\dots$   $y_{max}$  сопоставим список  $x$ -координат всех пикселей, закрасенных при растеризации рёбер, которые находятся на горизонтали  $y$  (здесь  $y_{min}$  и  $y_{max}$  — минимальная и максимальная  $y$ -координаты пикселей в растровом изображении многоугольника).

На втором этапе каждый из  $y$ -списков упорядочиваются по возрастанию.

На третьем этапе в каждом из  $y$ -списков заполняются (закрашиваются) все отрезки вида  $[x_{2i-1}, x_{2i}]$ .

### Алгоритм со списком активных рёбер

Попробуем несколько видоизменить предыдущий алгоритм. Вместо того чтобы хранить в памяти точки пересечения контура с каждой строкой раstra, ограничимся лишь одной строкой — текущей. А именно организуем список «активных» рёбер (CAР), в котором будем хранить информацию обо всех рёбрах многоугольника,

ка, пересекаемых текущей строкой. Удобство такого подхода в том, что при переходе к новой строке не требуется полностью переформировывать САР. Достаточно лишь удалить из него «закончившиеся» ребра (т. е. ребра из САР, чей нижний конец оказался выше нового значения  $y$ ) и добавить появившиеся.

Перейдём к формальному описанию алгоритма. Для каждого ребра создадим структуру данных  $\{y = \lceil y \rceil; dx = \frac{x_2 - x_1}{y_2 - y_1}; x = x_1 + dx \cdot (y - y_1)\}$ . Все такие структуры поместим в список ( $y$ -список) и упорядочим его по возрастанию  $y$ . Далее, пока САР не пуст, для текущего значения  $y$

- добавляем новые ребра из  $y$ -списка ( $y$  них  $\text{ребро.y} = y$ ), сохраняя упорядоченность САР по  $x$ ;
- закрашиваем отрезки вида  $[x_{2i-1}, x_{2i}]$  в строке  $y$ ;
- переходим к следующему значению  $y$ ;
- удаляем «закончившиеся» рёбра, в остальных рёбрах обновляем значение  $x$ , увеличивая его на  $dx$  ребра, при этом сохраняя упорядоченность САР по  $x$ .

Преимущество двух приведённых алгоритмов перед последующими состоит в том, что операции вывода на экран (относительно «медленные» во многих системах) для каждого пикселя выполняются не более одного раза. Недостатком является использование динамических структур данных (списков), что сильно усложняет код и требует дополнительной памяти.

Существует класс систем, в которых использование динамических структур данных нежелательно (вследствие ограниченности ресурса памяти или отсутствия удобных средств разработки программ), в то время как замедление работы из-за частого обращения к видеопамати не критично (сюда относятся, например, мобильные телефоны и другие портативные устройства, имеющие графический дисплей). В таких системах эффективным будет использование следующих алгоритмов, оперирующих непосредственно с данными в видеобуфере (т. е. с «содержимым» экрана).

## Алгоритм с операцией XOR

Этот оригинальный алгоритм использует свойства операции XOR (исключающее ИЛИ). Напомним, что XOR — бинарная операция над битами, действующая по правилу:

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Пусть контур многоугольника растеризован и выведен на экран. Тогда его закрашивание сводится к заполнению в каждой строке раstra всех промежутков вида  $[x_{2i-1}, x_{2i}]$ , где через  $x_k$  обозначены  $x$ -координаты «включённых» пикселей в данной строке, упорядоченные по возрастанию.

Через  $I(x, y)$  обозначим состояние пикселя с координатами  $(x, y)$ :  $I(x, y) = 1$ , если пиксель «включён», и  $I(x, y) = 0$  в противном случае. Нетрудно убедиться, что последовательное выполнение операции

$$I(x + 1, y) = I(x, y) \text{ XOR } I(x + 1, y)$$

для  $x = 1, 2, 3, \dots, X - 1$  (где  $X$  — горизонтальный размер раstra) приведёт к требуемому результату с той лишь разницей, что последний пиксель в каждом промежутке закрашен не будет. Эта небольшая неточность в большинстве случаев не критична и визуально незаметна. В противном случае можно после закрашивания повторно вывести на экран контур многоугольника.

Достоинством алгоритмов XOR является их предельная простота. Недостаток — невозможность работы при наличии посторонних изображений на экране.

## Исключительные случаи

Остановимся теперь на способах обработки исключительных случаев, приведённых на рис. 4. При рассмотрении горизонтального ребра достаточно при растеризации этого ребра вывести лишь его концы.

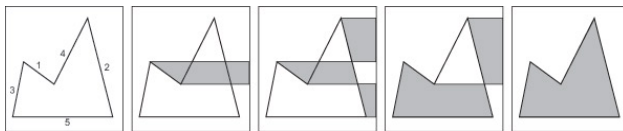


Рис. 5. XOR с перегородкой

Для исключения остальных случаев можно поступить следующим образом. При растеризации каждого ребра многоугольника не будем выводить его нижний конец  $(x_2, y_2)$ , а верхний конец выведем с помощью операции  $I(x_1, y_1) = I(x_2, y_2) \text{ XOR } 1$ . Это приведёт к тому, что верхние (нижние) концы рёбер, попавшие в один и тот же пиксель, не будут выведены, а значит, «одиночные» точки в строках раstra будут исключены.

### Алгоритм с операцией XOR с перегородкой

Алгоритм заключается в инвертировании цвета всех пикселей, расположенных правее  $i$ -го ребра, производимом последовательно для  $i = 1, 2, \dots, N$  (порядок нумерации рёбер не имеет значения). Горизонтальные ребра при этом игнорируются. Как видно из рис. 5, в результате закрашенными окажутся все внутренние пиксели многоугольника, и только они.

К достоинствам данного алгоритма можно отнести его простоту и оригинальность, а также отсутствие дополнительных структур данных. Недостатком является необходимость выполнения большого числа операций с пикселями (до  $N$  операций с каждым пикселем), в том числе и вне многоугольника. В частности, чем больше расстояние между многоугольником и правой границей экрана, тем больше будет совершено «лишних» операций.

От этого недостатка свободна модификация данного алгоритма — «XOR-2 с перегородкой». Идея ее заключается в том, чтобы инвертировать область не между ребром и правой границей экрана, а между ребром и вертикальной прямой («перегородкой»), мысленно проведённой в любом удобном месте, например, пересекающей многоугольник.

## Заполнение с затравкой

Область, подлежащая заполнению, не всегда задаётся в виде многоугольника. В этом разделе мы рассмотрим случай, когда заполняемая область задаётся цветом своей границы. Множество пикселей на растре не определяет область однозначно, поэтому требуется задать координаты «затравочного» пикселя, принадлежащего области.

Алгоритмы, рассматриваемые в этом разделе, используют структуру данных под названием стек. Стек содержит упорядоченный набор элементов и поддерживает две основные операции: добавить элемент и извлечь элемент. Вторая операция возвращает элемент, добавленный последним, и удаляет его из набора элементов. Программно стек может быть реализован на основе одномерного массива.

Простейший алгоритм заполнения с затравкой использует идею обхода раstra в глубину (переходим к рассмотрению соседнего пикселя; если подходящих соседних пикселей больше нет, рассматриваем соседей предыдущего рассмотренного пикселя) или в ширину (сначала рассматриваем соседей текущего пикселя, потом соседей соседей и т. д.).

Данный алгоритм можно модифицировать. Вместо того чтобы на каждой итерации закрашивать один пиксель, можно закрашивать линию. Для этого используется пространственная когерентность:

- пиксели в строке меняются только на границах;
- при перемещении к следующей строке размер заливаемой строки скорее всего или неизменен, или меняется на один пиксель.

Таким образом, на каждый закрашиваемый фрагмент строки в стеке хранятся координаты только одного начального пикселя, что приводит к существенному уменьшению размера стека.

Последовательность работы алгоритма следующая:

1. Затравочный пиксель помещается в стек, затем до исчерпания стека выполняются пункты 2 — 4.

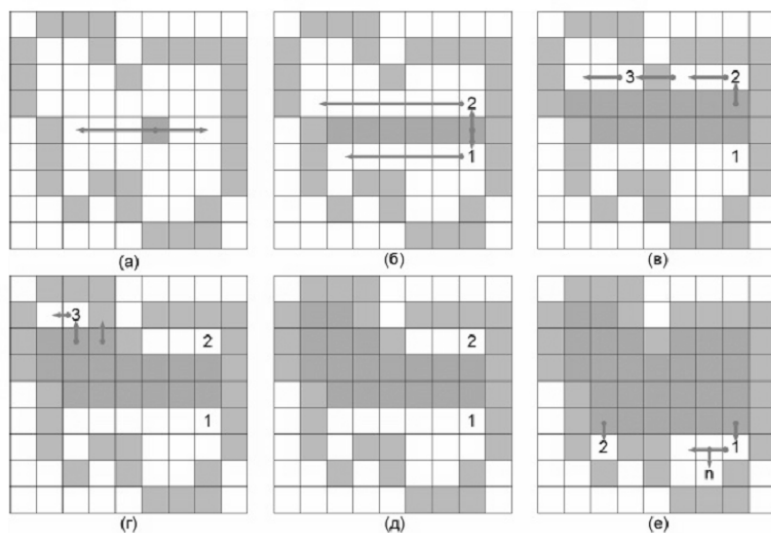


Рис. 6. Заполнение с затравкой по отрезкам

2. Очередной пиксель извлекается из стека и выполняется максимально возможное закрашивание вправо и влево по строке, содержащей этот пиксель, т. е. пока не попадётся граничный пиксель. Пусть это соответственно  $X_L$  и  $X_R$ .
3. Анализируется строка ниже закрашиваемой в пределах от  $X_L$  до  $X_R$ , и в ней находятся крайние правые пиксели всех незакрашенных фрагментов. Их координаты заносятся в стек.
4. То же самое проделывается для строки выше закрашиваемой.

Пример работы алгоритма показан на рис. 6.

Более подробное описание алгоритмов можно найти в [1, пункты 2.17 — 2.24].

## 5.1. Лабораторная работа 5

В этой лабораторной работе необходимо реализовать один из алгоритмов заполнения области:

1. Алгоритм со списком реберных точек.



2. Алгоритм со списком активных ребер.
3. Алгоритм с XOR.
4. Алгоритм XOR с перегородкой.
5. Алгоритм XOR-2 с перегородкой.
6. Алгоритм заполнения с затравкой.
7. Алгоритм заполнения с затравкой по отрезкам.

*Входные данные:* координаты вершин многоугольника и координаты точки (для вариантов 6 и 7).

*Выходные данные:* изображение многоугольника, многоугольник закрашивается по шагам, каждый шаг – несколько шагов алгоритма.

## Комментарии к заданию

Пошаговое построение означает демонстрацию принципа работы алгоритма, т. е. при демонстрации должно быть понятно, какой алгоритм реализован.

## 6. Построение выпуклой оболочки

Задача вычисления (построения) выпуклой оболочки не только является центральной в целом ряде приложений, но и позволяет разрешить ряд вопросов вычислительной геометрии, на первый взгляд не связанных с ней. Построение выпуклой оболочки конечного множества точек, и особенно в случае точек на плоскости, уже довольно широко и глубоко исследовано и имеет приложения, например в распознавании образов, обработке изображений, а также в задаче раскроя и компоновки материала.

Понятие выпуклой оболочки множества точек  $S$  является естественным и простым. В соответствии с определением это наименьшее выпуклое множество, содержащее  $S$ . Чтобы наглядно представить это понятие в случае, когда  $S$  — конечное множество точек на плоскости, предположим, что это множество охвачено большой

растянутой резиновой лентой. Когда лента освобождается, то она принимает форму выпуклой оболочки.

## Алгоритм полного перебора

Сначала рассмотрим алгоритм, использующий простую идею. *Точка  $p$  не является крайней (т. е. принадлежащей границе выпуклой оболочки) плоского выпуклого множества  $S$  только тогда, когда она лежит в некотором треугольнике, вершинами которого являются точки из  $S$ , но сама она не является вершиной этого треугольника.*

Эта теорема даёт идею для алгоритма удаления точек, не являющихся крайними. Имеется  $O(N^3)$  треугольников, определяемых  $N$  точками множества  $S$ . Проверка принадлежности точки заданному треугольнику может быть выполнена за некоторое постоянное число операций, так что за время  $O(N^3)$  можно определить, является ли конкретная точка крайней. Повторение этой процедуры для всех  $N$  точек множества  $S$  потребует времени  $O(N^4)$ .

Для того чтобы найти выпуклую оболочку конечного множества точек, требуется выполнить следующие два шага:

1. Определить крайние точки.
2. Упорядочить эти точки так, чтобы они образовывали выпуклый многоугольник.

Хотя данный алгоритм является чрезвычайно неэффективным, он очень прост в идейном плане и показывает, что крайние точки могут быть определены за конечное число шагов. Алгоритм затрачивает время  $O(N^4)$  только на определение крайних точек, которые должны быть как-то упорядочены, чтобы образовать выпуклую оболочку. Если известны крайние точки некоторого множества, то его выпуклую оболочку можно найти, упорядочив крайние точки в соответствии с полярным углом относительно одной из крайних точек.

## Алгоритм Грэхема

Существуют более эффективные алгоритмы построения выпуклой оболочки. Алгоритм Грэхема имеет сложность  $O(N \log N)$ . Суть алгоритма состоит в однократном просмотре упорядоченной последовательности точек, в процессе которого удаляются внутренние точки. Оставшиеся точки являются вершинами выпуклой оболочки, представленными в требуемом порядке.

1. Находим самую нижнюю из самых левых точек  $Q$  — она гарантированно является крайней.
2. Сортируем точки множества по величине полярного угла в системе координат с центром в  $Q$ . Сортировку точек можно производить по знаку косого произведения  $\overrightarrow{QP_i} \cdot \overrightarrow{QP_{i+1}}$ . В отсортированном массиве точек все указанные произведения должны быть неотрицательны. Точки с равными углами ( $\overrightarrow{QP_i} \cdot \overrightarrow{QP_{i+1}} = 0$ ) должны располагаться в порядке увеличения длин соответствующих векторов  $\overrightarrow{QP_i}$ .
3. Помещаем первые две точки в стек.
4. Последовательно просматриваем остальные точки в порядке сортировки. Пусть  $P_i$  — текущая точка,  $P_{k-1}, P_k$  — две точки в вершине стека. Пока участок ломаной  $P_{k-1}P_kP_i$  не является выпуклым, т. е.  $\overrightarrow{P_{k-1}P_k} \cdot \overrightarrow{P_kP_i} < 0$ , из стека удаляется точка  $P_k$ .
5. Затем  $P_i$  помещается в стек.

Так как любая точка добавляется в стек ровно один раз, то и удаляется она из него не более одного раза, поэтому время просмотра составляет  $O(N)$ .

## Алгоритм Эндрю

Эндрю А. М. предложил несколько модифицировать алгоритм Грэхема. Сложность алгоритма по-прежнему  $O(N \log N)$ , однако алгоритм легче запрограммировать.

Пусть на плоскости задано множество из  $N$  точек. Определим сначала его левую и правую крайние точки  $L$  и  $R$  и построим прямую  $(LR)$ , проходящую через эти точки. Оставшиеся точки разбиваются на два подмножества (нижнее и верхнее) в зависимости от того, по какую сторону от прямой  $LR$  они располагаются — ниже или выше. Для каждого из подмножеств строится огибающая, нижняя и верхняя соответственно, объединение которых дает выпуклую оболочку исходного множества.

Рассмотрим построение верхней оболочки. Точки упорядочиваются по возрастанию абсциссы. К полученной последовательности применяется метод обхода Грэхема.

## Алгоритм Джарвиса

Этот алгоритм иногда называют алгоритмом «заворачивания подарка».

В предыдущих алгоритмах проверялось свойство вершины быть вершиной выпуклой оболочки. Отличие алгоритма Джарвиса от трех предыдущих в том, что он проверяет, является ли отрезок ребром выпуклой оболочки. *Отрезок, соединяющий две точки заданного множества, является ребром выпуклой оболочки этого множества тогда и только тогда, когда все точки множества лежат по одну сторону от прямой, содержащей этот отрезок (либо на самой прямой).*

Сначала нужно определить первую точку, заведомо принадлежащую выпуклой оболочке. В качестве неё выберем самую нижнюю из самых левых точек. Далее каждая следующая точка выпуклой оболочки определяется следующим образом: из всех оставшихся точек выбирается та, что образует минимальный полярный угол в системе координат с центром в последней определённой точке выпуклой оболочки. Таким образом, все точки множества будут находиться слева от отрезка, соединяющего последнюю точку выпуклой оболочки и найденную точку.

Допустим,  $P_i$  —  $i$ -я вершина выпуклой оболочки. Следующую точку  $P_{i+1}$  нужно выбрать таким образом, чтобы косые произведения  $\overline{P_i P_{i+1}} \cdot \overline{P_i M} \geq 0$  для всех точек  $M$  из множества. Поиск точки  $P_{i+1}$  можно осуществлять так. Сначала следующей считает-

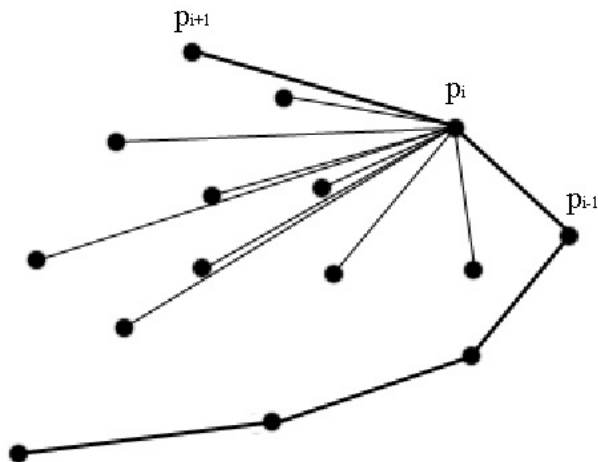


Рис. 7. Алгоритм Джарвиса

ся любая точка. Затем вычисляем значение  $\overline{P_i P_{i+1}} \cdot \overline{P_i M}$ , рассматривая в качестве  $M$  все остальные точки. Если для одной из них указанное выражение меньше нуля, считаем следующей ее и продолжаем проверку остальных точек (аналогично алгоритму поиска минимального элемента в массиве). Если же значение выражения равно нулю, то сравниваем квадраты длин соответствующих векторов. В результате за  $O(N)$  операций очередная вершина выпуклой оболочки будет найдена. Продолжая эту процедуру, мы рано или поздно вернёмся к точке  $P_1$ . Это будет означать, что выпуклая оболочка построена. Процесс выбора следующей точки  $P_{i+1}$  изображён на рис. 7.

Алгоритм имеет сложность  $O(Nh)$ , где  $h$  — количество вершин выпуклой оболочки. Таким образом, в худшем случае (много точек лежит на выпуклой оболочке) алгоритм имеет сложность  $O(N^2)$ . В этом случае алгоритм Джарвиса уступает алгоритму Грэхема. Если же большинство точек лежит внутри выпуклой оболочки, то алгоритм будет иметь линейную сложность.

## Метод «Разделяй и властвуй»

Метод «Разделяй и властвуй» при построении выпуклой оболочки основан, как обычно для этого типа алгоритмов, на идее разбиения вычислительной задачи на подзадачи примерно одинаковой размерности, которые решаются рекурсивно, после чего результаты объединяются:

1. **Разделяй:** имеющийся набор точек делится вертикальной прямой на две равных по количеству точек части.
2. **Властвуй:** находятся выпуклые оболочки обеих частей.
3. **Объединение решений:** получившиеся выпуклые оболочки объединяются.

Рассмотрим шаги более подробно.

На первом шаге требуется разделить имеющееся множество точек на два. Для этого необходимо отсортировать точки по возрастанию абсциссы, а затем вызвать алгоритм рекурсивно для каждой из половин. Рекурсивный вызов будем осуществлять до тех пор, пока не останется одна точка или две.

Теперь рассмотрим процесс объединения двух выпуклых оболочек. Дадим несколько определений.

Прямая называется *касательной* выпуклого многоугольника  $P$ , если она проходит через вершину многоугольника  $P$  и внутренняя часть  $P$  лежит по одну сторону от этой прямой.

Прямая называется *мостиком* двух выпуклых многоугольников  $P$  и  $Q$ , если она является касательной для каждого из многоугольников.

Мостик  $l$  двух выпуклых многоугольников  $P$  и  $Q$  называется *верхним (нижним)*, если оба многоугольника  $P$  и  $Q$  лежат выше (ниже) прямой  $l$ .

Таким образом, для слияния оболочек необходимо

- найти верхний и нижний мостики данных оболочек;
- мостики делят каждую из оболочек на две части;
- для каждой из оболочек выбрать часть, входящую в оболочку объединения;

- объединить найденные части.

Рассмотрим алгоритм нахождения верхнего мостика. Предполагаем, что выпуклые оболочки заданы списком вершин в порядке обхода против часовой стрелки.

1. Установить указатель  $V_l$  на самую правую вершину левого многоугольника, а указатель  $V_r$  — на самую левую вершину правого.
2. Пока поворот от вектора  $\overline{V_l V_r}$  к вектору  $\overline{V_l V_{r+1}}$  осуществляется против часовой стрелки, смещать указатель  $V_r$  на вершину  $V_{r+1}$ .
3. Пока поворот от вектора  $\overline{V_r V_l}$  к вектору  $\overline{V_r V_{l-1}}$  осуществляется по часовой стрелке, смещать указатель  $V_l$  на вершину  $V_{l-1}$ .
4. Если на 2 или 3 шаге указатели смещали, то вернуться к шагу 2.

### 6.1. Лабораторная работа 6

В этой лабораторной работе необходимо реализовать один из алгоритмов построения выпуклой оболочки:

1. Алгоритм полного перебора.
2. Алгоритм Грехэма.
3. Алгоритм Эндрю.
4. Алгоритм Джарвиса.
5. Метод «разделяй и властвуй».

*Входные данные:* координаты точек *Выходные данные:* пошаговое построение выпуклой оболочки заданного набора точек.

### Комментарии к заданию

Пошаговое построение означает демонстрацию принципа работы алгоритма, т. е. при демонстрации должно быть понятно, какой алгоритм реализован.

## Литература

1. Роджерс, Д. Алгоритмические основы машинной графики / Д. Роджерс. — М. : Мир, 1989. — 504 с.
2. Шикин, Е. В. Компьютерная графика. Динамика, реалистические изображения / Е. В. Шикин, А. В. Боресков. — М. : ДИАЛОГ-МИФИ, 1995. — 288 с.



# Оглавление

<b>Введение</b>	<b>3</b>
<b>1. Задачи вычислительной геометрии</b>	<b>4</b>
1.1. Лабораторная работа 1 . . . . .	4
<b>2. Геометрические преобразования</b>	<b>7</b>
2.1. Лабораторная работа 2 . . . . .	10
<b>3. Алгоритмы растеризации отрезка и окружности</b>	<b>13</b>
3.1. Лабораторная работа 3 . . . . .	14
<b>4. Алгоритмы отсечения отрезка</b>	<b>15</b>
4.1. Лабораторная работа 4 . . . . .	17
<b>5. Алгоритмы заполнения областей</b>	<b>18</b>
5.1. Лабораторная работа 5 . . . . .	24
<b>6. Построение выпуклой оболочки</b>	<b>25</b>
6.1. Лабораторная работа 6 . . . . .	31
<b>Литература</b>	<b>32</b>

Учебное издание

Алгоритмы компьютерной графики.  
Лабораторные работы  
Практикум

Составитель  
**Елисеева Надежда Дмитриевна**

Редактор, корректор Л. Н. Селиванова  
Компьютерный набор и вёрстка: Н. Д. Елисеева

Подписано в печать 31.10.2022. Формат 60х84 1/16.  
Усл. печ. л. 2,1. Уч.-изд. л. 1,5. Тираж 2 экз. Заказ .

Оригинал-макет подготовлен  
в редакционно-издательском отделе  
Ярославского государственного университета  
им. П. Г. Демидова

Ярославский государственный университет  
им. П. Г. Демидова  
15003, Ярославль, ул. Советская, 14.



