В. О. Дженжер, Л. В. Денисова
Оренбургский государственный педагогический университет, Россия

НАУЧНАЯ ГРАФИКА В PASCALABC.NET: ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ В ПРЯМОУГОЛЬНОЙ ДЕКАРТОВОЙ СИСТЕМЕ КООРДИНАТ

Аннотация
В статье рассмотрен алгоритм построения графиков функций в прямоугольной декартовой системе координат. Изложен способ преобразования координат из естественной системы в экранную. Алгоритм реализован на языке программирования PascalABC.NET. Представлены необходимые сведения о графических средствах языка, приведен код программы с подробными пояснениями. Отличительной особенностью метода является использование наглядных и легко вычисляемых параметров для масштабирования графика.
Ключевые слова: научная графика, компьютерная графика, компьютерное моделирование, программирование.
DOI: 10.32517/2221-1993-2020-19-1-31-39

1. Введение

Если у кого-нибудь сегодня возникнет необходимость построить график некоторой функции, то к его услугам будет множество самых различных средств. Это и электронные таблицы, и огромное количество платных и бесплатных математических пакетов (Matlab, Mathcad, Scilab, Octave, SMath и др.), и удобные графические модули некоторых языков программирования (например, matplotlib в Python), и веб-инструменты (например, WolframAlpha.com). Графики можно построить и средствами PascalABC.NET: есть удобная процедура DrawGraph, которая рисует график функции, заданной в прямоугольной декартовой системе координат. Такие построения стали общедоступными и рутинными.

Мы зададимся простым вопросом: а как именно все эти программы строят графики? Ведь всегда найдется любознательный школьник, который спросит об этом. Вопрос тем более интересный, что многие книги по языкам программирования для школьников либо ничего (или практически ничего) не говорят о графике [1, 8], либо в них данный вопрос рассматривается с точки зрения не очень доступных сегодня языков программирования [2, 4], либо при описании используется ручной подбор коэффициентов для вывода графиков на экран [3]. Довольно полно построение графиков описано в презентации К. Ю. Полякова [5]. К сожалению, автор использует устаревшую версию языка Pascal и практикует введение «магических констант», значения которых подбираются каким-то неясным образом.

Здесь мы хотели бы обсудить **универсальный способ построения графиков функций в прямоугольной декартовой системе координат**. Подобные задачи постоянно возникают при визуализации компьютерных моделей. Причем стандартные средства построения графиков оказываются зачастую не вполне пригодными, если:

- требуется создать анимацию перемещения тела по траектории, когда нам нужно не только увидеть саму линию, но и понять характер движения точки по ней — например, движется точка с ускорением или замедлением;
- нужно разделить в пространстве анимацию движения объекта и построение графика — например, при моделировании колебания грузика на пружине рисовать рядом траекторию движения грузика и график зависимости $x = x(t)$;

Контактная информация

Дженжер Вадим Олегович, канд. физ.-мат. наук, доцент, зав. кафедрой информатики, физики и методики преподавания информатики и физики, Оренбургский государственный педагогический университет, Россия; *адрес:* 460014, Россия, г. Оренбург, ул. Советская, д. 19; *e-mail:* vdjenjer@yandex.ru

Денисова Людмила Викторовна, канд. пед. наук, доцент кафедры информатики, физики и методики преподавания информатики и физики, Оренбургский государственный педагогический университет, Россия; *адрес:* 460014, Россия, г. Оренбург, ул. Советская, д. 19; *e-mail:* lv-denisova@yandex.ru

V. O. Dzhenzher, L. V. Denisova
Orenburg State Pedagogical University

SCIENTIFIC GRAPHICS IN PASCALABC.NET: PLOTTING FUNCTION GRAPHS IN A RECTANGULAR CARTESIAN COORDINATE SYSTEM

Abstract
The article considers the algorithm for plotting functions in a rectangular Cartesian coordinate system. A method is described for converting coordinates from a natural system to a screen system. The algorithm is implemented in the programming language PascalABC.NET. The necessary information about the graphical tools of the language is presented, the program code with detailed explanations is given. A distinctive feature of the method is the use of visual and easily calculated parameters for scaling the graph.
Keywords: scientific graphics, computer graphics, computer modeling, programming.

- вместе с рисованием графика нужно производить дополнительные построения, такие как штриховка областей, — например, закрашивание секторов при проверке второго закона Кеплера.

Таких примеров можно привести достаточное количество. Поэтому на самом деле проблема стоит шире: **как разместить множество точек плоскости, построенных в одной из «естественных» систем координат, на экране компьютера**.

Дальнейшее изложение вполне доступно школьнику, знакомому с преобразованиями графиков функций (сжатием и растяжением, параллельным переносом и преобразованиями симметрии), знающему про синус и косинус, а также владеющему основами программирования на языке PascalABC.NET. Знание объектно-ориентированного программирования не требуется, но приветствуется.

2. Краткое введение в графический модуль GraphWPF

В систему программирования PascalABC.NET входят два модуля для двумерной растровой графики: *GraphABC* и *GraphWPF*. Первый из них в настоящий момент признан устаревшим и больше не поддерживается, хотя включен в стандартную поставку и им по-прежнему можно пользоваться. Модуль *GraphWPF* является более новой разработкой и рекомендуется к использованию для обучения основам компьютерной графики. Остановим наш выбор на нем и рассмотрим кратко только те возможности модуля, которые будем использовать в своих проектах.

Модуль подключается командой:

```
uses GraphWPF;
```

У *GraphWPF* есть одна особенность, которая поначалу может показаться необычной: координаты точек графического окна можно задавать действительными числами, без округления. Это как минимум избавляет от необходимости вставлять повсюду функцию round, что немного сокращает код.

2.1. Графическое окно.

Рисование осуществляется на поверхности объекта-окна, называемого *Window*, у которого есть полезные для нас свойства и методы (табл. 1).

Как это принято в компьютерной графике, начало прямоугольной декартовой системы координат, связанной с графическим окном, расположено в левом верхнем углу окна, ось *OX* направлена вправо, а ось *OY* — вниз. В *GraphWPF* можно изменить эти умолчания при помощи процедуры *SetMathematicCoords*, но здесь мы не будем ее использовать, а рассмотрим подход, который будет работать в общем случае.

2.2. Цвета.

Тип цвета носит имя *Color* (или *GColor*), что является синонимом для *System.Windows.Media.Color* в Microsoft.NET. Цветовые константы определены в классе *Colors* (синоним для *System.Windows.Media.Colors*). Например, красный цвет можно задать так: *colors.Red*. Вообще, запоминать эти константы никому не придет в голову, поскольку в системе программирования работает Intellisence, т. е. технология автоматического дополнения текста, который начал набирать программист. Так, при вводе точки после слова «colors» выпадает подсказка по всем предопределенным цветам (рис. 1).

Таблица 1

Некоторые свойства и методы объекта Window

Имя	Описание и примеры
Свойства	

`Width: real`	Ширина окна в пикселях:
	`Window.Width := 500;`
	`Print(Window.Width);`

`Height: real`	Высота окна в пикселях:
	`Window.Width := 500;`
	`Print(Window.Width);`

`Title: string` (оно же `Caption`)	Заголовок окна: `Window.Title := 'Мой заголовок';`
Методы	
`Maximize`	Развернуть окно на весь экран: `Window.Maximize;`
`Save('ИмяФайла')`	Сохранить изображение окна в графическом файле: `Window.Save('gr.png');`
`Load('ИмяФайла')`	Загрузить изображение из графического файла в окно: `Window.Load('gr.png');`
`Clear`	Очистить окно белым цветом
`Clear(color)`	Очистить окно заданным цветом (про использование цветов сказано ниже)

Рис. 1. Работа Intellisence при выборе цвета

Такая же подсказка появляется всякий раз после ввода точки после имени объекта, класса, типа, модуля и т. д.

При необходимости создать цвет, отсутствующий в стандартном наборе, можно обратиться к функции *RGB*, которая возвращает цвет, построенный из красной, зеленой и синей составляющих, или к функции *ARGB*, которая позволяет задать еще и параметр прозрачности (все параметры целые и должны лежать в диапазоне 0..255). Например:

```
var c := ARGB(100, 230, 44, 155);
```

Имеется функция для генерации случайного цвета — *RandomColor* (она же *clRandom*).

2.3. Объекты *Pen* и *Brush*.

Объект *Pen* (перо) отвечает за рисование линий. Объект *Brush* отвечает за закрашивание областей. Оба объекта связаны с графическим окном: они автоматически создаются при создании графического окна. Нам могут понадобиться следующие свойства этих объектов (табл. 2).

Таблица 2

Некоторые свойства объектов *Pen* и *Brush*

Имя	Описание и примеры
Pen	
`Color: GColor`	Цвет пера. Пример использования: `Pen.Color := colors.DarkSeaGreen;`
`Width: real`	Толщина пера. Определяет толщину линий, которые будут получаться при рисовании этим пером. Измеряется в пикселях. Например: `Pen.Width := 5;`
Brush	
`Color: GColor`	Цвет пера. Пример использования: `Brush.Color := colors.MistyRose;`

2.4. Графические примитивы.

Для наших целей понадобится не очень много средств рисования. Главными будут точка (пиксель) и линия. В некоторых случаях пригодятся окружности и прямоугольники. Представим необходимые инструменты в таблице 3.

Модуль *GraphWPF* чрезвычайно богат возможностями, но их полное описание заняло бы слишком много места. Кроме того, изучать эти инструменты лучше всего в контексте решения каких-нибудь интересных задач. Заметим, что модуль довольно подробно описан во

Таблица 3

Некоторые графические примитивы модуля `GraphWPF`

Синтаксис	Описание
`SetPixel(x, y: real; c: color);`	Закрасить точку (x, y) цветом c
`Line(x1, y1, x2, y2: real);`	Нарисовать отрезок прямой от точки ($x1$, $y1$) до точки ($x2$, $y2$) текущим цветом
`Line(x1, y1, x2, y2: real; c: color);`	Нарисовать отрезок прямой от точки ($x1$, $y1$) до точки ($x2$, $y2$) цветом c
`Circle(x, y, r: real);`	Нарисовать закрашенный круг с центром в точке (x, y) и радиусом r текущими пером и кистью
`Circle(x, y, r: real; c: color);`	Нарисовать закрашенный круг с центром в точке (x, y) и радиусом r текущим пером и заполненный цветом c
`DrawCircle(x, y, r: real);`	Нарисовать окружность с центром в точке (x, y) и радиусом r текущим пером
`DrawCircle(x, y, r: real; c: color);`	Нарисовать окружность с центром в точке (x, y) и радиусом r и цветом c
`Rectangle(x, y, w, h: real);`	Нарисовать закрашенный прямоугольник с координатами вершин (x, y) и ($x + w$, $y + h$) текущими пером и кистью
`Rectangle(x, y, w, h: real; c: color);`	Нарисовать закрашенный прямоугольник с координатами вершин (x, y) и ($x + w$, $y + h$) текущим пером и заполненный цветом c
`DrawRectangle(x, y, w, h: real);`	Нарисовать незакрашенный прямоугольник с координатами вершин (x, y) и ($x + w$, $y + h$) текущим пером
`DrawRectangle(x, y, w, h: real; c: color);`	Нарисовать незакрашенный прямоугольник с координатами вершин (x, y) и ($x + w$, $y + h$) цветом c

встроенной справке и при необходимости всю информацию можно почерпнуть оттуда.

3. Графики в прямоугольной декартовой системе координат

Как уже отмечалось выше, прямоугольные декартовы системы координат, принятые в математике и компьютерной графике, заметно различаются. Но, может быть, для нас эти различия несущественны? Напишем первую программу, чтобы проверить это (листинг 1).

Алгоритм довольно прост. Переменная x изменяется от значения $x = x_0$ до $x = x_k$ с шагом dx. Для каждого такого x вычисляем $y = x^2$, после чего ставим на экране красную точку с координатами $(x; y)$. Результат работы программы представлен на листинге 1, и он не вполне удовлетворителен. Во-первых, оказалась построенной только половина параболы, а именно ее ветвь в первой четверти. Во-вторых, ветви параболы развернуты вниз, в то время как должны быть (при положительном значении коэффициента при x^2) направлены вверх. Наконец, в-третьих, все расстояния на экране измеряются в пикселях, из-за чего парабола выглядит очень мелкой.

Посмотрим, как можно разрешить эти проблемы. Обозначим оси абсцисс и ординат экранной системы координат как U и V соответственно. Координаты точки на экране будем обозначать $(u; v)$. Тогда общая задача, стоящая перед нами, состоит в том, чтобы найти способ преобразования естественных координат $(x; y)$ в $(u; v)$.

3.1. Проблема 1: половина параболы.

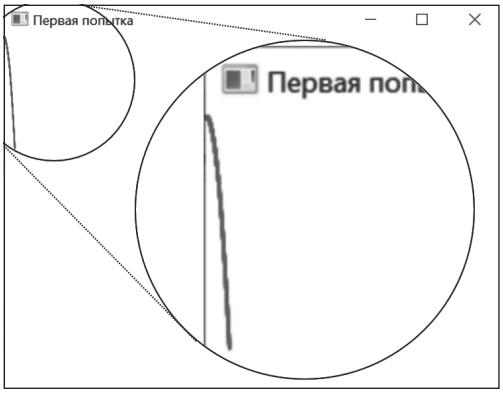
Проблема возникла из-за того, что начало экранной системы координат расположено в левом верхнем углу экрана. Нам хотелось бы, чтобы эта точка располагалась внизу экрана и горизонтально по центру. Назовем точку на экране, в которую хотим перенести начало системы координат, $(H_u; H_v)$. Здесь индексы u, v показывают, что речь идет об экранной системе координат, т. е. эти величины измеряются в пикселях. Далее рассмотрим только те случаи, когда центр системы координат не выходит за пределы графического окна.

Наиболее часто используемое в практике расположение осей OX и OY на экране показано на рисунке 2.

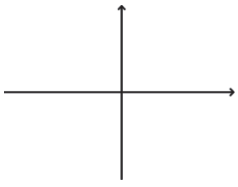
Разумеется, при необходимости можно расположить центр системы координат в любом месте экрана, как удобно в каждом конкретном случае.

Чтобы решить нашу первую проблему, нужно к вычисленным в цикле координатам $(x; y)$ каждый раз прибавлять значения H_u и H_v. Тогда вершина параболы переместится в нужную точку.

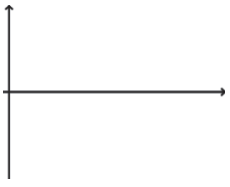
```
uses GraphWPF;
begin
  Window.Title := 'Первая попытка';
  var (x0, xk) := (-10.0, 10.0);
  var dx := 0.01;
  var x := a;
  while x < b do
  begin
    var y := x * x;
    SetPixel(x, y, colors.Red);
    x += dx;
  end;
end.
```



Листинг 1. Программа для рисования параболы (первый вариант) и результат ее выполнения

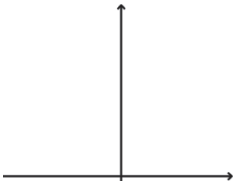


а) Hu = Window.Width / 2
Hv = Window.Height / 2



б) Hu = 0
Hv = Window.Height / 2

в) Hu = 0
Hv = Window.Height



г) Hu = Window.Width / 2
Hv = Window.Height

Рис. 2. Возможные расположения осей OX и OY на экране и соответствующие значения Hu и Hv

3.2. Проблема 2: ветви параболы направлены вниз.
Проблема возникает из-за того, что вертикальная ось экранной системы координат (ось *V*) направлена вниз, тогда как «в тетрадке» она направлена вверх. Очевидно, для исправления ситуации следует развернуть эту ось, для чего соответствующую координату *y* нужно взять с минусом.

3.3. Проблема 3: неудобная система единиц.
Решением является масштабирование графика, которое сводится к замене пикселей на единицы метрической системы: сантиметры, метры и т. п. Сделать это очень просто, умножив соответствующую координату на масштабный коэффициент. Будем называть эти коэффициенты для осей *OX* и *OY* соответственно S_x и S_y. Как определить их значения? Для ответа на этот вопрос соберем все наши рецепты вместе и запишем конечные формулы для преобразования условных сантиметров в пиксели.

$$\begin{cases} u = x \cdot S_x + H_u \\ v = -y \cdot S_y + H_v \end{cases} \quad (1)$$

Здесь *x* и *y* мы вычисляем в цикле, а *u* и *v* — координаты соответствующей точки на экране.

Рассмотрим размерности величин для первого уравнения в (1). Очевидно, что слева стоят пиксели, значит, и справа должны быть пиксели. H_u измеряется в пикселях, значит, и произведение $(x \cdot S_x)$ — тоже в пикселях. Но *x* измеряется в сантиметрах, тогда в чем должно измеряться S_x? Очевидно, в пикселях на сантиметр: $\frac{\text{пиксель}}{\text{см}}$. То есть величина S_x показывает, сколько пикселей укладывается на одном сантиметре оси *OX*. Чтобы вычислить это значение, нужно знать количество пикселей, приходящихся на ширину и на высоту окна (это *Window.Width* и *Window.Height*), а также количество сантиметров, которое несет на себе каждая из осей. Если мы примем, что по оси *OX* график занимает *rx* см, а по оси *OY* — *ry* см, то масштабные коэффициенты можно вычислить по формулам:

$$\begin{cases} S_x = \dfrac{Window.Width}{rx} \\ S_y = \dfrac{Window.Height}{ry} \end{cases} \quad (2)$$

Заметим, что величины *rx* и *ry* имеют совершенно ясный смысл: они определяют, сколько сантиметров занимает график по осям *OX* и *OY на бумаге*. Другими словами, если мы нарисуем график в тетрадке, то *rx* и *ry* — это ширина и высота в сантиметрах минимального прямоугольника со сторонами, параллельными осям системы координат, которым можно накрыть график.

Однако в таком случае масштабные коэффициенты по осям будут различны, а это значит, что единичные отрезки на осях будут иметь разную длину в пикселях. Такое допустимо, если по осям *OX* и *OY* отложены величины разной размерности, например, время по оси *OX* и скорость по оси *OY*. Если же по осям *OX* и *OY* отложены одинаковые величины (например, сантиметры), то мы получим искажение картинки — сжатие по одной из осей. При построении графиков это не всегда уместно. Чтобы такого не произошло, будем пользоваться формулой:

$$\begin{cases} S_x = \dfrac{Window.Width}{rx} \\ S_y = S_x \end{cases} \quad (3)$$

Надо понимать, однако, что если $rx < ry$, то в этом случае мы *обрежем график вдоль оси OY*: он не поместится целиком в квадрат со стороной *rx*. Если хочется все-таки уместить график на экране целиком, следует увеличить *rx*, не изменяя при этом x_0 и x_k.

3.4. Общая схема построения графиков в прямоугольной декартовой системе координат.
Итак, получается следующая общая схема построения графиков в прямоугольной декартовой системе координат:

1. Определить желаемое положение центра системы координат на экране и задать значения H_u и H_v.
2. Определить, одинаковые ли единицы откладываются по осям *OX* и *OY*. Если да, то воспользоваться формулой (3), иначе — формулой (2).
3. Определить, сколько места займет график по осям *OX* и *OY* — это величины *rx* и *ry*. Обычно для прямоугольной декартовой системы координат:
   $rx = x_k - x_0$;
   $ry = y_{\text{наибольшее}} - y_{\text{наименьшее}}$ (на отрезке $[x_0; x_k]$).
   Если мы пользуемся формулой (2), то *ry* можно не определять.
4. Вычислить масштабные коэффициенты S_x и S_y по формулам (2) или (3).
5. Построить график, используя для перевода $(x; y) \to (u; v)$ формулы преобразования системы координат (1).

4. Написание программы

4.1. Общая структура программы

Для работы с экраном удобнее всего было бы создать класс, который содержит описанные ранее переменные и методы и производит расчеты и рисование. Но в этой статье, чтобы не усложнять код и сделать его доступным даже для начинающих, мы воспользуемся процедурным подходом, а все переменные, нужные для расчетов, сделаем глобальными. (Использование глобальных переменных в подпрограммах не приветствуется, но в данном случае нас в некоторой степени извиняет то, что мы будем только считывать значения этих переменных везде, кроме процедуры инициализации.)

Опишем пять процедур:

1) рисование масштабной сетки с заданным шагом по осям (*Grid*);
2) построение осей с привязкой к H_u и H_v (*Axes*);
3) инициализация (*Init*);
4) вывод точки на экран с предварительным переводом из «обычной» системы координат в экранную (*OutPixel*);
5) построение графика (*Plot*).

Рассмотрим их создание на том же примере — построение параболы (листинг 2).

В листинге 2 приведена структура программы. Описаны только заголовки процедур. Рассмотрим сами процедуры, но не в порядке их описания, а в порядке их появления в главной программе.

```
1.  uses GraphWPF;
2.  var dx, x0, xk,            // переменные, относящиеся к поставленной задаче
3.  Hu, Hv, rx, Sx, Sy: real;  // переменные для работы с графикой
4.  procedure Grid(dx, dy: real);
    …
5.  procedure Axes;
    …
6.  procedure Init;
    …
7.  procedure OutPixel(x, y: real; c: color);
    …
8.  procedure Plot;
    …
9.  begin
10.   Init;
11.   Plot;
12. end.
```

Листинг 2. Структура программы

```
1.  procedure Init;
2.  begin
3.    // Задание переменных, используемых в постановке задачи
4.    (x0, xk) := (-3, 3);                  // начальное и конечное значения по X
5.    dx := 0.001;                          // шаг по X
6.    // Задание переменных для работы с графикой
7.    Hu := Window.Width / 2;               // См. Листинг 2(в)
8.    Hv := Window.Height - 1;              // (-1) -- чтобы ось OX не ушла под экран
9.    rx := xk - x0;                        // Область построения графика по OX
10.   // ry не вычисляем, т. к. используем одинаковый масштаб по осям
11.   Sx := Window.Width / rx;              // См. формулу (3)
12.   Sy := Sx;
13.   Window.Title := 'Парабола: версия 2';  // Заголовок окна
14.   Axes;                                 // Построение осей системы координат
15. end;
```

Листинг 3. Текст процедуры Init

4.2. Инициализация данных

Первой выполняется процедура *Init* (листинг 3).

Заметим, что все переменные, которые мы задаем, весьма наглядны: интервал построения графика, шаг при построении, положение центра системы координат на экране, размеры области построения. Масштабные коэффициенты, которые иногда в подобных программах подбирают вручную, у нас вычисляются автоматически.

4.3. Построение осей системы координат и масштабной сетки

В последней строке процедуры *Init* вызывается процедура построения системы координат *Axes* (листинг 4).

Здесь в строках 6–7 рисуются две серые линии: оси системы координат. Линии проходят через начало системы координат — точку (H_u; H_v), поэтому при изменении положения осей на экране никаких изменений в процедуру вносить не придется. В строке 4 вызывается процедура построения масштабной сетки *Grid*. Ее текст приведен в листинге 5.

Процедура получает на вход два параметра *dx* и *dy*, которые задают шаг сетки по соответствующим осям в сантиметрах. Например, в листинге 4 (строка 4) процедура вызвана с параметрами (1, 1). Это означает, что расстояние между двумя соседними линиями сетки равно 1 см. Как мы выяснили выше, на 1 см оси *OX* при-

```
1.  ///Рисует оси ПДСК, проходящие через точку (Hu; Hv)
2.  procedure Axes;
3.  begin
4.    Grid(1, 1);                                  // Сетка с шагом 1 по осям
5.    Pen.Width := 2;                              // Толщина пера = 2 пикселя
6.    Line(0, Hv, Window.Width, Hv, colors.Gray);  // OX
7.    Line(Hu, 0, Hu, Window.Height, colors.Gray); // OY
8.  end;
```

Листинг 4. Текст процедуры Axes

```
1.   ///Рисует сетку с шагом dx см по оси OX и с шагом dy см по оси OY
2.   procedure Grid(dx, dy: real);
3.   begin
4.     var (du, dv) := (dx * Sx, dy * Sy);                  // Шаг сетки по осям (в пикселях)
5.     var (nx, ny) := (round(1 / dx), round(1 / dy));      // Линий на единицу
6.     Pen.Width := 1; // Толщина линий сетки
7.     //Вертикальные линии сетки
8.     var u := Hu;                                         // Начинаем от оси OY ...
9.     var k := 1;                                          // ... и смещаемся влево и вправо на (k *
du)
10.     while (u + k * du).InRange(0, Window.Width) or     // пока хотя бы одна из
11.           (u - k * du).InRange(0, Window.Width) do     // ... линий в окне
12.     begin
13.       Pen.Color := k mod nx > 0 ? colors.Gold : colors.DarkOrange;
14.       line(u + k * du, 0, u + k * du, Window.Height);
15.       line(u - k * du, 0, u - k * du, Window.Height);
16.       k + = 1;
17.     end;
18.     //Горизонтальные линии сетки
19.     var v := Hv;                                         // Начинаем от оси OX ...
20.     k := 1;                                              // ... и смещаемся вверх и вниз на (k * dv)
21.     while (v + k * dv).InRange(0, Window.Height) or    // пока хотя бы одна из
22.           (v - k * dv).InRange(0, Window.Height) do    // ... линий в окне
23.     begin
24.       Pen.Color := k mod ny > 0 ? colors.Gold : colors.DarkOrange;
25.       line(0, v + k * dv, Window.Width, v + k * dv);
26.       line(0, v - k * dv, Window.Width, v - k * dv);
27.       k + = 1;
28.     end;
29.   end;
```

Листинг 5. Текст процедуры Grid

ходится S_x пикселей, и на 1 см оси OY — S_y пикселей. Мы воспользовались этим в строке 4 листинга 5, чтобы вычислить шаг по осям в пикселях.

В строке 5 листинга 5 вычисляются параметры *nx* и *ny*, определяющие количество линий сетки, приходящееся на единичный отрезок. Далее в строке 13 для каждой вертикальной линии с номером *k*, кратным *nx*, задаем более темный цвет; таким образом, линии, проходящие через целое число единиц, будут выделены, что улучшает восприятие графика. Аналогичный код встречается в строке 24 для выбора цвета горизонтальных линий сетки. Заметим, что в строках 13 и 24 мы использовали условную (тернарную) операцию как замену традиционному *if* для сокращения кода.

Далее рисуются вертикальные линии сетки. Для этого мы стартуем (строка 8) с точки H_u (т. е. с точки ноль по оси OX) и движемся симметричными шагами влево и вправо от этой точки: сначала на $\pm du$, потом на $\pm(2 \cdot du)$ и так далее на $\pm(k \cdot du)$. Построение продолжается, пока хотя бы одна из линий (правая или симметричная ей левая) попадает в границы экрана. Принадлежность точки x отрезку $[a; b]$ проверяем при помощи метода (логической функции) $x.InRange(a, b)$, который возвращает истину, если выполняется условие $a \le x \le b$.

Со строки 18 рисуются горизонтальные линии сетки. Алгоритм их построения аналогичен только что рассмотренному.

```
1.   ///Строит график функции в соответствии с инициализацией
2.   procedure Plot;
3.   begin
4.     var x := x0;                    // От x = x0 ...
5.     while x < xk do                 // ... и пока x < xk
6.     begin
7.       var y := x * x;               // Вычисляем значение функции в точке x
8.       OutPixel(x, y, colors.Red);   // Рисуем точку красным цветом
9.       x += dx;                      // Берем следующую точку
10.    end;
11.  end;
```

Листинг 6. Текст процедуры Plot

```
1.  ///Рисует на экране точку цветом col, соответствующую точке (x; y)
2.  procedure OutPixel(x, y: real; col: color);
3.  begin
4.    // Переводим (x; y) -> (u; v) (см. формулу (1))
5.    var u :=  x * Sx + Hu;
6.    var v := -y * Sy + Hv;
7.    SetPixel(u, v, col); // Ставим точку
8.  end;
```

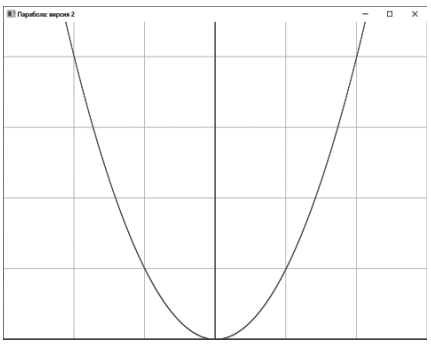
Листинг 7. Текст процедуры OutPixel

4.4. Построение графика

Вернемся к главной программе. После вызова *Init* идет обращение к процедуре *Plot* (листинг 6). Ее текст практически воспроизводит программу листинга 1.

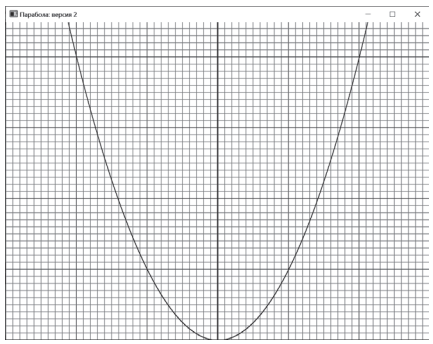
Существенным отличием от первой версии программы является вызов процедуры *OutPixel*, которая выполняет преобразование из «естественной» системы координат в экранную и только потом рисует точку. Текст процедуры приведен в листинге 7. Сами преобразования описываются формулой (1).

На рисунке 3 показаны результаты работы программы для нескольких вариантов настроек. Обратите внимание, что интервал построения графика в каждом из примеров один и тот же: [–3; 3]. Но в случаях *а* и *б* пара-



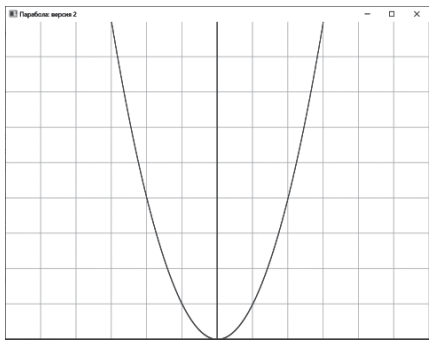
а

```
(x0, xk) := (-3, 3);
rx := xk - x0;
Grid(1, 1);
```



б

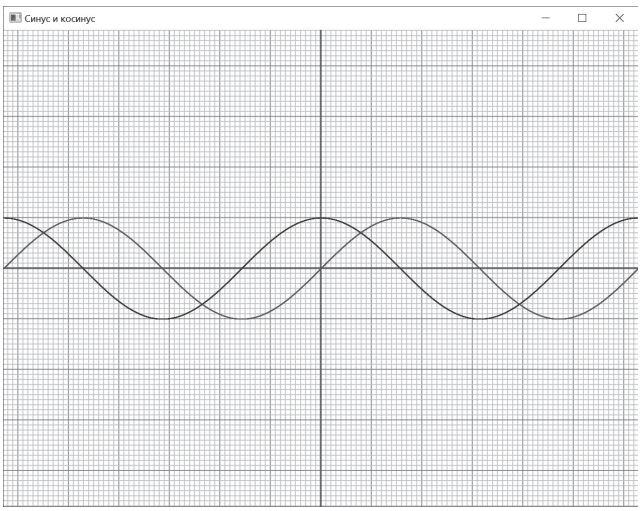
```
(x0, xk) := (-3, 3);
rx := xk - x0;
Grid(0.1, 0.1);
```



в

```
(x0, xk) := (-3, 3);
rx := 2 * (xk - x0);
Grid(1, 1);
```

Рис. 3. Результаты выполнения программы для некоторых настроек




```
(x0, xk) := (-2 * pi, 2 * pi);
Hu := Window.Width / 2
Hv := Window.Height / 2
rx := xk - x0;
Grid(0.1, 0.1);
```

Рис. 4. Построение двух графиков в одной системе координат

```
1.   ///Строит два графика
2.   procedure Plot;
3.   begin
4.     var x := x0;                      // От x = x0 ...
5.     while x < xk do                   // ... и пока x < xk
6.     begin
7.       var y := sin(x);                // Вычисляем значение функции sin в точке x
8.       OutPixel(x, y, colors.Red);     // Рисуем точку красным
9.       y := cos(x);                    // Вычисляем значение функции cos в точке x
10.      OutPixel(x, y, colors.Blue);    // Рисуем точку синим
11.      x += dx;                        // Берем следующую точку
12.    end;
13.  end;
```

Листинг 8. Текст процедуры Plot для построения двух графиков

метр «зума» *rx* = 6, а в случае *в* он в два раза больше, что позволило увидеть верхнюю часть графика, обрезанную на первых двух иллюстрациях.

На рисунке 4 показан пример построения в одном графическом окне сразу двух графиков — синуса и косинуса. Нам потребовалось изменить начальное и конечное значения переменной *x* в процедуре *Init* и добавить в процедуру *Plot* построение второго графика.

Текст измененной процедуры *Plot* приведен в листинге 8.

Как видно, текст процедуры подвергся минимальным изменениям. Мы использовали уже определенные переменные *x* и *y* для вычисления значения второй функции.

5. Заключение

Рассмотренный алгоритм построения графиков функций в прямоугольной декартовой системе координат может использоваться не только по своему прямому назначению. Небольшие модификации алгоритма позволяют визуализировать, например, движение грузика на пружинке или полет тела, брошенного под углом к горизонту. Этот же алгоритм является базовым для построения графиков функций в полярной системе координат, а также функций, заданных параметрически. Овладев им, студент и школьник смогут перейти на новый уровень использования компьютера в моделировании.

Список использованных источников

1. *Грацианова Т. Ю.* Программирование в примерах и задачах. 2-е изд. (эл.). М.: БИНОМ. Лаборатория знаний, 2015. (ВМК МГУ — школе). 354 с.

2. *Комлев Н. Ю.* Полезное программирование. М.: СОЛОН-Пресс, 2016. 256 с.

3. *Майер Р. В.* Компьютерное моделирование: учебно-методическое пособие для студентов педагогических вузов [Электронное учебное издание на компакт-диске]. Глазов: Глазов. гос. пед. ин-т, 2015. https://sites.google.com/site/mayerrv/comp_mod

4. *Могилев А. В., Пак Н. И., Хеннер Е. К.* Информатика: учебное пособие для студентов пед. вузов. 8-е изд. М.: Академия, 2012. 848 с.

5. *Поляков К. Ю.* Основные идеи структурного программирования. http://kpolyakov.spb.ru/download/pasppt3.zip

6. Построение графиков функций на языке PascalABC.NET на GitHub. https://github.com/vdjenjer/Plotting

7. Современное программирование на языке Pascal. http://pascalabc.net/

8. *Ушаков Д., Юркова Т.* Паскаль для школьников. 2-е изд. СПб.: Питер, 2011. 320 с.

НОВОСТИ

Рособрнадзор начнет использовать искусственный интеллект при проверке ЕГЭ

Проверка всех письменных работ ЕГЭ с помощью технологий искусственного интеллекта начнется в штатном режиме уже в 2020 году. Об этом рассказал врио руководителя Рособрнадзора Анзор Музаев.

«Использование программ искусственного интеллекта — распознавание почерков, эту работу мы провели в режиме апробации в прошлом году, с этого года вводим в штатный режим по всем предметам, — заявил он. — Пропускаем работы через программу, она определяет почерк. Порядка 35 работ экспертиза подтвердила, что они выполнены разными участниками экзамена. Подавляющее большинство подмен было в следующей последовательности: на сочинение приходил кто-то другой. А на ЕГЭ мы видели работу настоящего участника».

По словам Анзора Музаева, такая работа позволила из огромного массива данных — 5 млн сканов — выделить единичные нарушения, которые возникли в ходе экзамена.

Что касается технологий распознавания лиц на ЕГЭ и поведенческого анализа с помощью нейросети, то это будет пока тестироваться в режиме апробации, с согласия школьников и родителей.

(По материалам «Российской газеты»)