Инструкционная карта № 6

Тема: «Графический редактор»

Графические возможности объекта Image

1. Все, что касается рисования, содержится в свойстве-объекте Canvas.
2. Свойства Canvas, предопределяющие то, как все будет рисоваться: Pen, Brush, Font.
3. Методы Canvas рисования фигур: Rectangle, Ellipse.
4. Методы Canvas рисования линий: MoveTo, LineTo.
5. Метод Canvas копирования картинок: Draw.

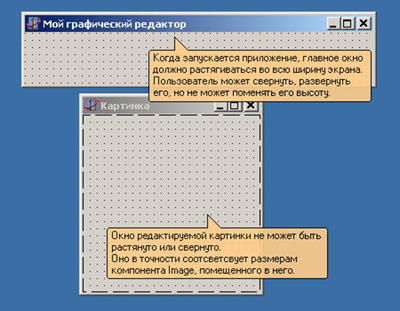
Важнейшим свойством компонента Image является свойство Canvas:TCanvas (холст). Это свойство само является объектом (одни объекты вполне могут быть свойствами других, ведь свойство – по сути, переменная, а в переменной может храниться ссылка на объект). Таким образом, у самого Canvas есть методы и свойства. Это свойства и методы рисования на холсте.

В представленном фрагменте кода фактически задействованы три свойства (тоже являющиеся объектами!):

* Pen: TPen (перо): рисование линий, границ фигур и т.п. производится пером. Важнейшие свойства пера – Color: TColor (цвет), Width: Integer (ширина), Style: TPenStyle (стиль).
* Brush: TBrush (кисть): закраска фигур, заднего фона надписей и т.п. производится кистью. Важнейшие свойства кисти – Color: TColor (цвет), Style: TBrushStyle (стиль).
* Font: TFont (шрифт): надписи на холсте выполняются с учетом значений его свойства Font. При этом задний фон за надписью закрашивается текущим значением кисти. Для того чтобы закраски не было, нужно установить прозрачный стиль кисти: Brush.Style := bsClear, но затем не забудьте сделать кисть непрозрачной: Brush.Style := bsSolid, иначе фигуры, которые вы будете в дальнейшем рисовать, тоже окажутся незакрашенными. Важнейшие свойства шрифта – Name: string (имя), Size: Integer (размер), Color: TColor (цвет). Вы также использовали два метода холста:
* Rectangle(X1, Y1, X2, Y2: Integer) (прямоугольник): рисует прямоугольник, закрашивая его кистью и обводя пером (в нашем случае, для того чтобы рамки не было видно, мы сделали цвет кисти и пера одинаковым). В качестве параметров задаются координаты верхнего левого X1, Y1 и нижнего правого X2, Y2 углов относительно верхнего левого угла холста.
* TextOut(X, Y: Integer; const Text: string) (вывести текст): рисует текст согласно заданному шрифту и закрашивая задний фон согласно заданной кисти (в нашем случае эта закраска незаметна, поскольку текст рисуется на том же фоне, что и текущее значение кисти). В качестве параметров задаются координаты верхнего левого угла текста X, Y относительно верхнего левого угла холста и текст надписи Text.

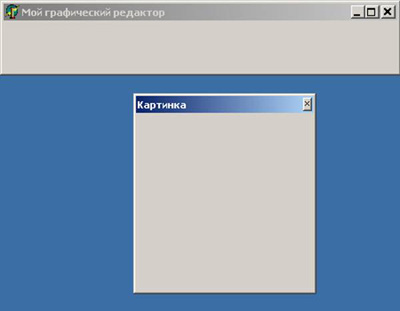
Графический редактор.

Этот практикум посвящен созданию многооконного графического редактора. Вы должны проявить свои знания в области разработки многооконных приложений. У вас в приложении должно быть два окна: одно главное для кнопок управления, и одно вспомогательное, на котором будет располагаться редактируемая картинка.



Указание:

* Используйте свойства Align, Constraints, BorderStyle, AutoSize.



Добейтесь, чтобы при запуске программы у вас получалась подобная картина, – и можете двигаться дальше.

Теперь сделайте так, чтобы в окне Картинка можно было рисовать мышкой.



Указания:

* Используйте методы OnMouseDown, OnMouseMove и OnMouseUp.
* Придется ввести глобальную переменную, определяющую, идет ли рисование.

Если получилось, то можно двигаться дальше.

Добавьте на главную форму компонент Shape и сделайте, чтобы при нажатии на него можно было выбрать цвет, которым вы рисуете.

Указания:

* Используйте компонент диалога выбора цвета.
* Не упустите из виду, что Shape находится на одной форме, а рисование происходит на другой, – неизбежно придется указывать ссылку в uses.



Добавьте на главную форму компонент Shape и сделайте, чтобы при нажатии на него можно было выбрать цвет, которым вы рисуете.

Указания:

* Используйте компонент диалога выбора цвета.
* Не упустите из виду, что Shape находится на одной форме, а рисование происходит на другой, – неизбежно придется указывать ссылку в uses.

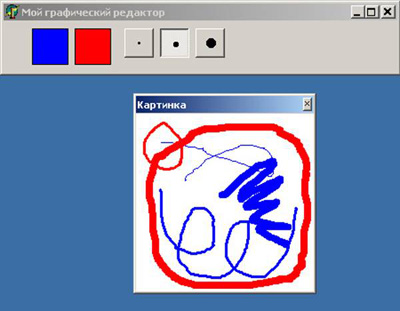


Теперь добавьте три кнопки SpeedButton и сделайте так, чтобы переключая их можно было бы выбирать ширину пера при рисовании.

Указания:

* Вспомните про свойства GroupIndex.
* Кружок на кнопке можно сделать, используя Caption и символ шрифта Wingdings, соответствующий букве «l». А размер кружка пусть определяет Font.Size.

Должно получиться нечто подобное:



На главную форму добавьте кнопку создания нового документа.

Добавьте еще одну форму, она должна вызываться модально при нажатии этой кнопки.

Если пользователь нажал «Новый документ» и правильно ввел размеры картинки, то

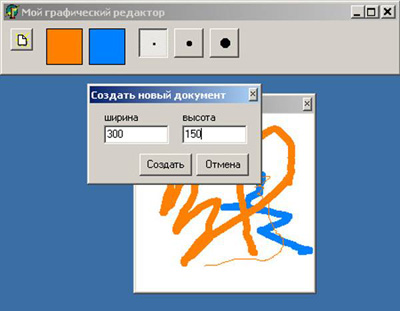
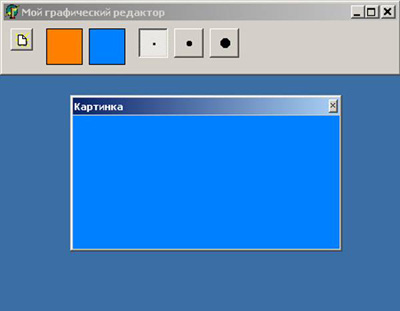


Image формы картинки должен принять указанные размеры и оказаться закрашенным цветом фона.

Указания:

* Для проверки правильности ввода значений можно воспользоваться конструкцией try.. except.
* При изменении размеров Image нужно также изменить значения Image.Picture.Bitmap: если w и h – новые размеры, то Image.Width := w; Image.Picture.Bitmap.Width := w; Image.Height := h; Image.Picture.Bitmap.Height := h;

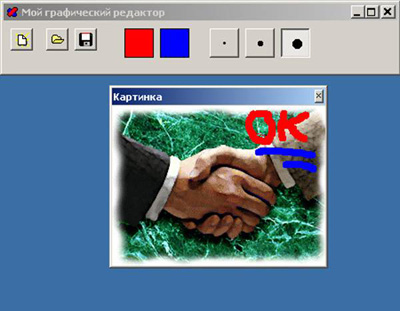
Похоже?



Осталось навести марафет: добавить возможность сохранения и загрузки картинок; если останется время, добавить собственную иконку; запрашивайте при выходе, стоит ли сохранить файл и т.п.

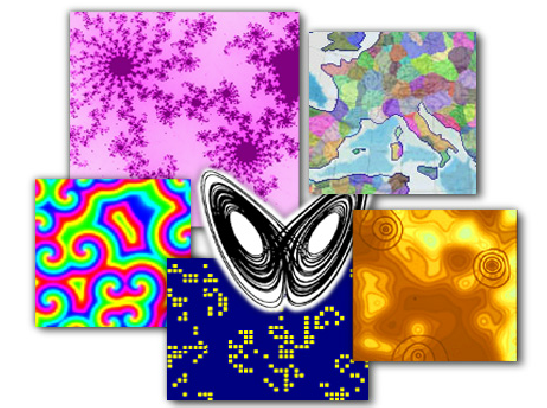
Указания:

* Используйте диалоги сохранения и загрузки.
* Формат загружаемых и сохраняемых картинок – bmp.
* Для сохранения и загрузки используйте методы Image.Picture.LoadFromFile и Image.Picture.SaveToFile, при загрузке не забывайте устанавливать размер Image соответствующим размеру Picture.Bitmap.



Одно из популярных сейчас научных направлений, синергетика, изучает всяческие нелинейные явления и процессы. В ее арсенале имеется множество интересных и поучительных эффектов, которые можно достаточно просто реализовать на компьютере. Приобщимся и мы к великому и создадим программу, рисующую фрактальное множество.

Наш нелинейный мир таит множество загадок. Сталкиваясь с ними, мы постепенно учимся их разгадывать. В последнее время не только ученые-теоретики, но и программисты-практики находят множество проявлений нелинейности. Всем известна, к примеру, игра клеточных автоматов «Жизнь». С ней мы еще встретимся, а нынешний урок посвящен другой разрешенной загадке – фракталам.



Самым известным фрактальным множеством является множество Мандельброта. Бенуа Мандельброт, обнаружив его, фактически заложил основу новой науки.

При всей многогранности и богатстве образов, содержащихся в этом фрактале, для построения его используется очень простое, но, что важно, нелинейное уравнение в комплексных переменных.

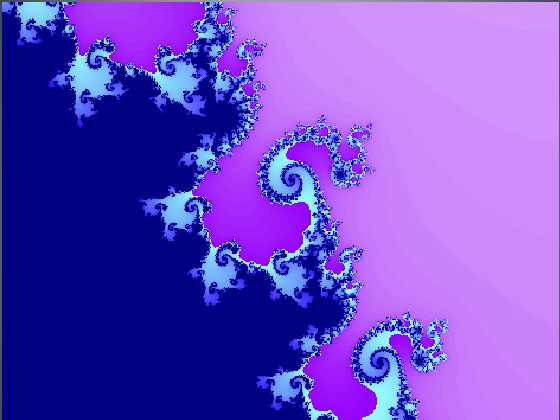
Это уравнение может быть приведено к действительным переменным, если его разбить на вещественную и мнимую часть.

Фрактал строится следующим образом. В комплексном пространстве фиксируется точка с координатами *p* и *q*, начальные *x* и *y* полагаются равными нулю, а затем просчитывается последовательность преобразований.

В результате этих преобразований комплексное число может начать расти, и в этом случае его модуль рано или поздно превысит некоторое критическое значение *L*. В этом случае точка окрашивается в цвет, соответствующий количеству циклов преобразований, прошедших с начала расчета.

Однако не все точки обладают таким свойством. Для некоторых из них модуль комплексного числа никогда не превысит *L*. Если модуль не превысил *L* за определенное время, то преобразования останавливаются, и точка окрашивается в темный цвет.

Такой простой алгоритм приводит к построению потрясающего фрактала. Бесконечно увеличивая его, мы никогда не перестанем сталкиваться с новыми причудливыми картинами.



Алгоритм построения множества Мандельброта

Выполнять для каждой точки выбранного прямоугольника:

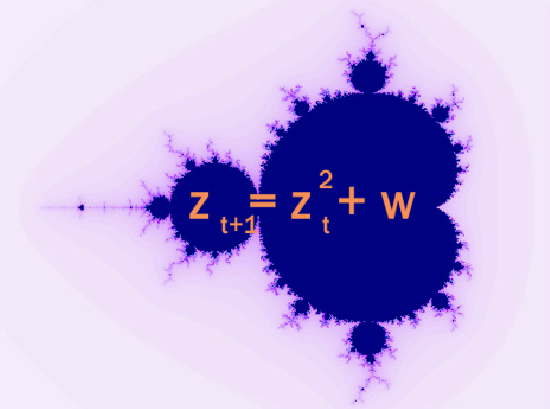
1. Записать координаты точки в p и q, обнулить x и y, обнулить счетчик. Перейти к пункту 2.
2. Увеличить счетчик, реализовать преобразования x и y. Перейти к 3.
3. Если модуль числа превысил L или счетчик превысил определенное время Time, остановить преобразования и перейти к 4, иначе перейти к 2.
4. Если счетчик превысил Time, окрасить точку в черный цвет, иначе (модуль превысил L) окрасить точку в цвет тем более темный, чем больше счетчик в момент выхода. Перейти к следующей точке.

Преобразования в пункте 2 имеют вид:

xt+1 = xt2 – yt2+p

yt+1 = 2xtyt + q

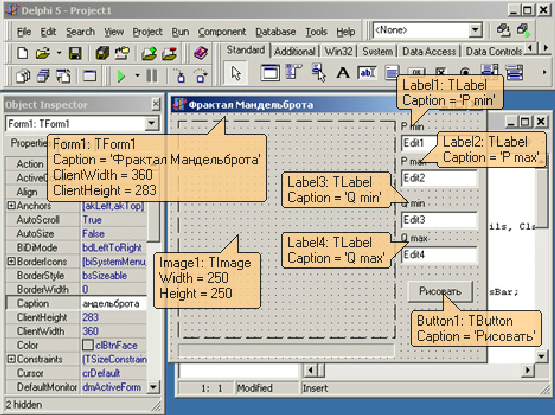
Модуль комплексного числа z = x + iy вычисляется по формуле: |z|2 = x2 + y2



Начните новый проект, поместите на форму Image кнопку Button, четыре поля ввода Edit, четыре надписи Label и линейку прогресса ProgressBar.

Установите свойства компонентов.

Значения этих полей ввода будут задавать прямоугольную часть комплексной плоскости, которая будет отображаться на компоненте Image.



Итак, по нажатию кнопки мы делаем следующее.

Определяем временные и пространственные ограничения.

Считываем данные из полей ввода.

Обеспечиваем цикл по точкам картинки, переводя их координаты в координаты на комплексной плоскости.

Обнуляем *x*, *y* и *t*.

Реализуем преобразования.

Оканчиваем цикл преобразований, когда модуль превысил *L* или время истекло.

В зависимости от условия выхода из цикла, окрашиваем точку.

Меняем значение у линейки прогресса, чтобы пользователь знал, скоро ли картинка обновится.

procedure TForm1.Button1Click(Sender: TObject);  
var i, j, t, Time, Size, L: integer;  
p0, q0, p1, q1, p, q, x, y, xx: double;  
begin  
  Time := 500;  
  Size := 250;  
  L := 100;  
  p0 := StrToFloat(Edit1.Text);  
  p1 := StrToFloat(Edit2.Text);  
  q0 := StrToFloat(Edit3.Text);  
  q1 := StrToFloat(Edit4.Text);  
  for i := 0 to Size - 1 do  
  begin  
    p := p0 + i \* (p1 - p0) / (Size - 1);  
    for j := 0 to Size - 1 do  
    begin  
      q := q0 + j \* (q1 - q0) / (Size - 1);  
      x := 0;  
      y := 0;  
      t := 0;  
      repeat inc(t);  
      xx := x;  
      x := sqr(x) - sqr(y) + p;  
      y := 2 \* xx \* y + q;  
      until (x \* x + y \* y > L) or (t = Time);  
      if t = Time then Image1.Canvas.Pixels[i, j] := clNavy      
      else Image1.Canvas.Pixels[i, j] := RGB(250 - t div 2,  
      250 - t div 2, 250);  
    end;  
    ProgressBar1.Position := round(i \* 100 / Size);  
  end;  
end;

При реализации этого алгоритма мы столкнулись с некоторыми новыми для нас функциями. Остановимся подробнее на следующих моментах:

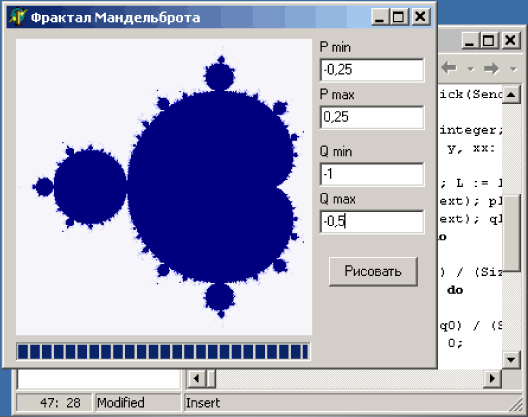
* Функция round служит для перевода чисел с плавающей точкой в целые числа. Делает она это, округляя число до ближайшего целого. Таким образом round(1.4) = 1, a round(8.7) = 9. Мы используем эту функцию для того, чтобы установить свойство Position у полоски прогресса, которое имеет целый тип. Есть и другая функция, переводящая числа с плавающей точкой в целые. Функция trunc округляет до наименьшего целого, то есть trunc(1.4) = 1 и trunc(8.7) = 8.
* функия StrToFloat(const S: string): Extended служит для перевода текстовых строк в действительные числа. Поскольку не всякую строку можно перевести в число, то при вызове этой процедуры нужно быть уверенным, что строка описана верно. Иначе ее вызов приведет к возникновению исключительной ситуации (exception). Об исключительных ситуациях мы поговорим на следующем уроке, пока же, если вдруг возникнет исключительная ситуация, – нажмите Ok и прервите выполнение программы (Ctrl-F2). Обратите внимание, что в строках string десятичная точка должна быть записана в виде запятой: StrToFloat(‘5.5’) вызовет исключительную ситуацию, а StrToFloat(‘5,5’) выдаст число, равное 5.5. Для обратного перевода используется функция FloatToStr(), а для работы с целыми числами используются StrToInt() и IntToStr() соответственно.
* Напоминаем тем, кто работал с Turbo Pascal, что функция inc() увеличивает целое число на единицу, sqr() выдает квадрат действительного числа, а div – это деление целых чисел нацело.
* Введенная нами переменная xx нужна для хранения значения x на прошлом шаге. Если бы ее не было, то y рассчитывался бы с учетом уже измененного x, что некорректно.
* Мы уже знакомы с холстом Canvas, но еще не рассматривали его свойство Pixels[X, Y: Integer]: TColor. В этом доступном для чтения и записи свойстве содержится цвет точки в формате TColor.

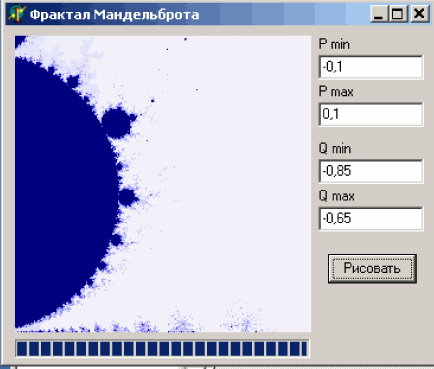
Запустите программу и нажмите кнопку. Вы видите фрактальное множество Мандельброта.

Самое интересное произойдет, когда вы попробуете увеличить какой-нибудь его кусочек.

Для этого задайте границы другими значениями и нажмите кнопку.

При увеличении вы увидите, что картинка не упрощается, а наоборот, оказывается все более сложной, самоповторяющейся. Попробуйте сами проникнуть в глубину фрактала, увидеть его красоту и многогранность.





Увеличивать картинку, задавая каждый раз с клавиатуры четыре числа и нажимая кнопку, не очень удобно. Лучше добиться этого одним нажатием мыши.

Сделаем так, чтобы при нажатии на Image картина увеличивалась в два раза, с центром в точке, на которую мы нажали.

Выделите Image и в Инспекторе Объектов дважды щелкните справа от события OnMouseDown.

В создавшемся обработчике введите код. Здесь мы считываем из полей ввода данные. В зависимости от того, какая кнопка мыши была нажата – левая или правая, мы вычисляем новые границы картинки – увеличенные или уменьшенные.

Эти границы записываем в поля ввода.

Затем вызываем метод Click кнопки Button1. Фактически мы имитируем нажатие на кнопку и, следовательно, вызываем обработчик этого нажатия, перерисовывая картинку.

В результате теперь нам не нужно задавать размеры с клавиатуры, а можно увеличивать и уменьшать картинку, нажимая левую и правую кнопку мыши.

Запустите программу и попробуйте сами.

Как отмечалось, необходимо заботиться о пользователе, тем более что пока вы сами являетесь единственным пользователем программы. Неплохо бы облегчить навигацию по фракталу. Мышь в этом предоставляет нам богатые возможности.

procedure TForm1.Image1MouseDown(Sender: TObject;  
Button: TMouseButton;  
Shift: TShiftState;  
X, Y: Integer);  
var p0, q0, p1, q1, x0, y0, x1, y1: double;  
begin  
  p0 := StrToFloat(Edit1.Text);  
  p1 := StrToFloat(Edit2.Text);  
  q0 := StrToFloat(Edit3.Text);  
  q1 := StrToFloat(Edit4.Text);  
  if Button = mbLeft then  
  begin  
    x0 := p0 + (p1 - p0) \* X / Image1.Width - (p1 - p0) / 4;  
    y0 := q0 + (q1 - q0) \* Y / Image1.Height - (q1 - q0) / 4;  
    x1 := p0 + (p1 - p0) \* X / Image1.Width + (p1 - p0) / 4;  
    y1 := q0 + (q1 - q0) \* Y / Image1.Height + (q1 - q0) / 4;  
    Image1.Repaint;  
  end   
  else  
  begin  
    x0 := p0 + (p1 - p0) \* X / Image1.Width - (p1 - p0);  
    y0 := q0 + (q1 - q0) \* Y / Image1.Height - (q1 - q0);  
    x1 := p0 + (p1 - p0) \* X / Image1.Width + (p1 - p0);  
    y1 := q0 + (q1 - q0) \* Y / Image1.Height + (q1 - q0);  
  end;  
  Edit1.Text := FloatToStr(x0);  
  Edit2.Text := FloatToStr(x1);  
  Edit3.Text := FloatToStr(y0);  
  Edit4.Text := FloatToStr(y1);  
  Button1.Click;  
end;

Метод OnMouseDown вкратце был описан во втором уроке. Здесь вы можете наблюдать, как реально используются его параметры.

Параметр Button содержит информацию о нажатой кнопке. В данном случае мы проверяем, равен ли он mbLeft, то есть была ли нажата левая кнопка. Если да, то, опираясь на параметры X и Y, в которых хранятся координаты нажатия мыши, мы переопределяем границы картинки.

Затем мы записываем новые границы в поля ввода Edit, используя описанную выше в этом уроке функцию FloatToStr.

И, наконец, вызов метода Click кнопки Button1 имитирует нажатие кнопки. Вызывается обработчик Button1.Click(), перерисовывается картинка (при перерисовке берутся уже измененные границы, содержащиеся в полях ввода Edit и, следовательно, картинка рисуется увеличившейся или уменьшившейся).

Интересный факт: начальная картинка на экране вашего монитора – около 10 см в ширину. Если вы будете увеличивать ее, щелкая мышкой, то где-то щелчков через 50 картинка «испортится» – разобьется на прямоугольники. Это связано не с тем, что фрактал исчерпал себя (он бесконечен), а с тем, что компьютер уже не может обеспечить необходимую точность. А во сколько раз при этом увеличилась начальная десятисантиметровая картинка? В это трудно поверить, но ее размер при подобном увеличении составит около 100.000.000.000 км! (расстояние от Земли до Солнца – "всего" 150.000.000 км) Вот вам и фрактал…

Для того чтобы наша программа приобрела еще более привлекательный вид, добавим несколько оформительских штрихов.

Во-первых, установим компоненту Image курсор мыши в виде руки, чтобы пользователю было ясно, что нужно нажимать на картинку.

Во-вторых, при увеличении будем рисовать рамку, обозначая новые границы отображаемой области. Для этого добавим в обработчик нажатия мыши соответствующий код.

В-третьих, предоставим пользователю возможность сохранять полученные картинки. Для этого добавим на форму еще один Edit, еще один Button и настроим их соответствующим образом.

В Edit пользователь должен указать имя файла без расширения, в который будет сохранена картинка в формате bmp. В обработчике нажатия на кнопку вызовем процедуру сохранения картинки.

В-четвертых, заметим, что раскраску фрактала можно производить каким-то другим способом, отличным от того, который предложили мы. Попробуйте на досуге сами поэкспериментировать.

Теперь ваша первая полноценная программа приобрела вполне привлекательный и профессиональный вид. Не забудьте сохранить и продемонстрировать ее вашим родителям, друзьям и знакомым.

При рисовании рамки обратим внимание на то, что у кисти (Brush) стиль устанавливается в значение bsClear. В результате рисуется рамка без закрашивания (с прозрачной закраской).

  Image1.Canvas.Pen.Color := clRed;  
  Image1.Canvas.Brush.Style := bsClear;  
  Image1.Canvas.Rectangle(X - Image1.Width div 4, Y -  
     Image1.Height div 4, X + Image1.Width div 4, Y +  
     Image1.Height div 4);  
  Image1.Repaint;

Необходимость вызова метода Repaint для Image1 связана с тем, что перерисовка происходит, когда обработчик события заканчивает работу. А он закончит работу только тогда, когда отработают все вызываемые внутри него процедуры, и в том числе Button1.Click, которая изменяет картинку. То есть перерисовка Image произойдет уже после всех вычислений, а нам нужно, чтобы рамка была видна во время вычислений. Поэтому необходимо самим перерисовать Image, вызвав метод Repaint.

Сохранение изображения, хранящегося в Image, происходит с помощью вызова метода SaveToFile(const FileName: string), принадлежащего свойству Picture (если помните, в прошлых уроках именно в Picture вы загружали картинки из файла в Image через Инспектор Объекта). С помощью метода SaveToFile картинки сохраняются в файлы в формате bmp. В качестве параметра FileName нужно указать имя файла, например:

SaveToFile(‘фрактал.bmp’)

или

SaveToFile(‘.\Pictures\’ + Edit1.Text + ‘.bmp‘)

Напоследок отметим, что при увеличении фрактала качество изображения (как вы, наверное, заметили) несколько ухудшается. Это происходит из-за того, что ограничитель L и время расчета Time малы. Увеличив Time, можно получить более детальный фрактал. Однако это замедлит расчеты, и тогда нужно будет откорректировать функцию раскраски фрактала – нужно сделать, чтобы компоненты цвета были в диапазоне от 0 до 255, то есть при Time = 1000 нужно будет t делить уже на 4 а не на 2:

RGB(250 – t div 4, 250 – t div 4, 250)

Раскрашивать фрактал можно вообще по другому принципу, например, опираясь не на t, а на компоненты x и y при выходе преобразований из цикла.

То есть вместо

if t = Time then Image1.Canvas.Pixels[i, j] := clNavy  
else Image1.Canvas.Pixels[i, j] := RGB(250 - t div 2, 250 - t div 2, 250);

написать

Image1.Canvas.Pixels[i,j] := $00007f + $000201 \* round(127 \* (x \* x / (x \* x + y \* y)));

Замечание. Цвета, кроме известных вам констант clRed, clBlue и функции RGB, можно задавать в шестнадцатеричном формате: $ff7f00. Первый байт $ff = 255 – компонента синего цвета, второй $7f = 127 – компонента зеленого и третий $00 = 0 – красная компонента.

Можете придумать и свою раскраску фрактала …