**Введение о системах счета.**

Позиционной системой счисления считается такая система счета, в которой значение числа определяется не только набором знаков, но и взаимным их расположением, поэтому «место» цифр в числе строго определено и называется *разрядом*. Разряды десятеричной системы счисления удостоились чести носить особые титулы: единицы, десятки, сотни и т.д. Основание системы — это значение, определяющее, во сколько раз различаются соседние разряды. А еще от значения основания системы зависит, сколько знаков (цифр) потребуется для записи всего множества чисел. То есть, если в двоичной системе основание системы — 2, то и цифр используется всего две: 0 и 1. А вот в шестнадцатеричной системе цифр, наоборот, не хватает, поэтому используются еще и буквы, но только пять (по количеству недостающих цифр) — ABCDEF.

**Система счисления RGB**

А 256-ричную систему счисления можете себе представить? Наглядный пример ее использования — система кодирования цвета RGB (согласно «принципу старшинства» следовало бы звать ее BGR :-)). И вот там, где число обращается в цвет, где математическая абстракция вспыхивает фейерверком радужных переливов, нас и ждут удивительные изображения, что по красоте и сложности могут конкурировать лишь с фракталами…

RGB-кодирование цвета предполагает, что существуют три цветовых канала, которые определяют яркость пиксела в красном, зеленом и синем цвете. При этом каждый из каналов яркости имеет 255 градаций, но цвет точки на экране традиционно задается одним числом. Так как число 255 в десятеричной системе равно 11 11 11 11 в двоичной, то есть равняется одному байту, следовательно, число, определяющее RGB-цвет, имеет 3 байта длины:

11 11 11 11 11 11 11 11 11 11 11 11

Первый байт указывает на яркость синего канала, второй — зеленого, третий — красного.

То есть цвет точки задается исходя из закономерности:

*С:=256\*256\*В+256\*G+R,*

где B, G, R — переменные, определяющие яркость соответствующих каналов.

А теперь возьмем Delphi, создадим форму, положим на нее PaintBox и зададим ему размеры:

paintbox1.Height:=256;

paintbox1.Width:=256;

И в обработчике события напишем код:

var x,y,c:integer;

begin

c:=0; //начальный цвет точки — 0 (черный)

for y:=0 to paintbox1.Height-1 do begin // для каждого пиксела по вертикали

for x:=0 to paintbox1.Width-1 do begin // и по горизонтали

paintbox1.Canvas.Pixels[x,y]:=c; //раскрасим его текущим цветом

c:=c+1; // и увеличим значение цвета на 1

end;

end;

end;

Посмотрим на полученное изображение ( **Рис. 1**): в первой прорисованной строке пикселов видим переход цвета от черного к красному. Оно и понятно — байт красного канала является младшим: когда он достигнет максимального своего значения (255), зеленый канал (второй байт) получит приращение на один бит, а красный сбросится в 0. Так же и в 10-ричном числе единицы, достигнув девяти, при следующем приращении на 1 увеличивают значение десятков, а сами сбрасываются в 0.



В изображении это будет выглядеть, как «скачок» цвета после плавного перехода: за максимально красным будет вновь следовать почти черный пиксел (который лишь на 1 бит будет «зеленее» черного). И так далее, итого 255 строк — поэтому нарастание яркости зеленого канала мы наблюдаем по вертикали.

Это общие принципы зависимости *Цвет — Числовое значение — Координата*.

Теперь начинаем экспериментировать!

Если интенсивность красного канала нарастает с координатой  *х*, а зеленого — с координатой  *у*, то воспользуемся границами области расчета цвета (0..255) и посмотрим, что будет, если цвет каждого пиксела данной области будет равен, например, удвоенной сумме его координат?

*c:= ((x+y)\*2);*

Получаем нечто, напоминающее красную драпировку ( **Рис. 2**). Здесь «провал» цвета после переполнения младшего (Red) разряда заметен более четко. Поэкспериментируйте, как будет меняться изображение с изменением коэффициента (в данном случае он был равен двум), на который умножается сумма координат…



А если цвет каждого пиксела будет равен произведению координат?

*c:=((x\*y));*

Попробуйте-ка нарисовать такое вручную ( **Рис. 3**)!



Этот фрагмент был получен в пределах *х (0..255)*, *у (0..255)*. А что же там, за пределами этих границ?

Увеличим область просмотра:

paintbox1.Height:=512;

paintbox1.Width:=512;

и посмотрим на тот же узор, но взятый более общим планом ( **Рис. 4**):



Используем закономерность, выявленную в экспериментах с суммой (умножение на некоторое число уменьшает изображение, позволяя заглянуть за его границы, не изменяя области просмотра, деление на константу — увеличивает размер изображения, позволяет рассмотреть мелкие детали).

Это изображение ( **Рис. 5**) получено при просчете формулы



*c:= ((x\*y)\*64);*

Так простая формула произведения двух чисел порождает теоретически бесконечное (а практически — ограниченное диапазоном видимого света и 24 битами, отведенными под его кодирование) самоподобное изображение. Что же говорить о более сложных формулах?

Это изображение ( **Рис. 6**) построено по формуле



*c:=round((x\*x+y\*y)/2);*

А картинка ( **Рис. 7**), построенная по формуле



*c:=round(abs((x\*x-y\*y)/128));*

является 1/8192-й частью **Рис. 8** ( *c:=(abs((x\*x-y\*y)\*64))*), а он, в свою очередь — 1/16-й частью **Рис. 9** ( *c:=(abs((x\*x-y\*y)\*1024))*).

    

А какие шедевры рвутся на ваши экраны из области тригонометрических функций! Правда, в их визуализации есть несколько особенностей.

Поскольку период sin и cos равен *2р*, область визуализации установим так:

paintbox1.Height:=360;

paintbox1.Width:=360;

Еще нам потребуется коэффициент k:=0,0174 (3,14/1800) — для перевода градусов в радианы). И еще придется учесть, что значения тригонометрических функций могут быть отрицательными…

Процедура вывода изображения на основе тригонометрических функций будет иметь вид:

var x,y,c:integer;

k:real;

begin

paintbox1.Height:=360;

paintbox1.Width:=360; // втискиваем градусы в декартовы координаты

k:=0.0174; //коэффициент — пригодится для масштабирования

for y:=0 to paintbox1.Height-1 do begin

for x:=0 to paintbox1.Width-1 do begin

paintbox1.Canvas.Pixels[x,y]:=c;

c:=round(255\*cos(x\*k)+255\* cos (y\*k)); // Задаем теплые оттенки для рисования

if c<0 then c:=255\*255\*255+c; //а если значение цвета отрицательное — то холодные

end;

end;

end;

В данном фрагменте кода изображение строится по формуле, где цвет точки по осям *х*и  *у* меняется в зависимости от косинуса соответствующей координаты. Это для начала, ведь так нагляднее: косинус — функция симметричная.

И вот что получается ( **Рис. 10**)



Масштабирование тригонометрических узоров осуществляется по тем же принципам — умножая коэффициент k на число, уменьшаем изображение; деля коэффициент на некоторое значение, увеличиваем картинку.

Тригонометрические изображения дают больше степеней свободы: помимо изменения формулы и масштаба можно менять коэффициенты цвета, чтобы отчетливее визуализировать «обрывы» цветового градиента и обнаруживать новый уровень узора.

Изображение на **Рис. 11** образуется по формуле:



*c:=round(255\*cos(x\*k)\*sin(y\*k)+255\*cos(y\*k)\*sin(y\*k));*

**Рис. 12** получен по той же формуле, но с увеличенным цветовым коэффициентом:



*c:=round(4096\*cos(x\*k)\*sin(y\*k)+4096\*cos(y\*k)\*sin(y\*k));*

А что получится, если они будут не равны? Экспериментируйте, вас ждет немало сюрпризов.



Хочется отметить еще одну особенность тригонометрических изображений — они периодичны, а значит могут выступать в качестве бесшовных текстур.