Автор: Андрей
10.12.2009 20:34 - Обновлено 15.12.2009 10:01

Описание программы

Программа, предложенная сегодня вашему вниманию, представляет интерес сразу с двух точек зрения. Во-первых, это просто красивая программа, более или менее достоверно рисующая эффект пламени. Во-вторых, она представляет собой хороший пример *двойной буферизации*, наглядно демонстрируя - для чего эта двойная буферизация нужна и насколько она эффективна. Давайте я расскажу о каждом аспекте поподробнее.

Итак, прежде всего непосредственно про реализацию самого эффекта пламени. Чтобы смоделировать его, я представил себе пламя как объект, состоящий из отдельных искр, которые зарождаясь где-то в нижней части объекта (там, где сгорает нечто горючее), затем, будучи легче воздуха, поднимаются вверх, попутно прогорая и постепенно теряя свою яркость. При этом я считаю, что в момент своего зарождения искра выбрасывается из точки своего образования с достаточно произвольной скоростью как горизонтальной, так и вертикальной (впрочем, только вверх); далее - что в процессе ее движения на искру действуют силы - во-первых, подъемная сила, а значит, она поднимается вверх ускоренно, во-вторых, сила сопротивления воздуха, следовательно, горизонтальная составляющая ее скорости снижается, пока не станет равной нулю. Наконец, я предполагаю, что яркость искры постоянно уменьшается, а когда становится равной нулю - искра прогорела.

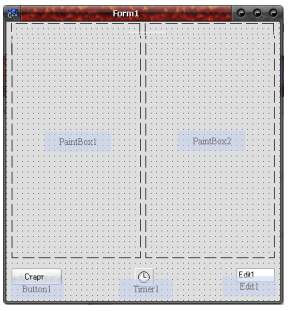
Исходя из этих предположений, я сделал искры классом, создал массив искр, решив, что пламя будет разгораться постепенно - при каждом срабатывании таймера число горящих искр будет увеличиваться, пока не будут гореть сразу все. А когда искра прогорает, она заново "зарождается" где-то внизу (при этом точное место определяется достаточно произвольно).

Что же касается, *двойной буферизации*, то ее идея проста - процесс построения изображения происходит на обычно невидимом графическом объекте (
буфере
), а по завершении этого построения содержимое буфера просто копируется туда, где изображение должно выводиться. Делается это для борьбы с эффектом мерцания в ситуации, когда за один проход изменяются цвета многих пикселей (много объектов меняют свое положение). В случае с C++ Builder'ом это обычно выглядит так: имеется объект типа TBitmap, в Canvas которого рисуется все изображение, а потом готовое изображение рисуется в Canvas формы. В данном конкретном случае вместо канвы невидимого битмапа и канвы формы используется канва двух объектов типа TPaintBox.

С принципиальной точки зрения это не имеет абсолютно никакого значения, но позволяет воочию увидеть - что творится там, где изображение строится в реальном времени (это буфер, его мучения мы обычно не видим), и как все здорово и красиво в конечном изображении, куда копируется уже готовая картинка из буфера.

Используются следующие компоненты: Button, Edit, PaintBox, Timer.

Расположение компонентов на форме



Свойства компонент, измененные по сравнению со стандартными

Button1:
Caption: "Старт"
PaintBox1:
Height: 345
Width: 193
PaintBox2:
Height: 345
Width: 193
Timer1:
Enabled: false
Interval: 10

Код программы

```
    #include <math>

const int N = 5000;

class FirePointClass
  {
  public:
  void Activate();
  void Evolution();
  void Drawing();
```

```
    bool IsBurning();
    private:
    int ColorChange;
    double X, Y;
    double Vx, Vy;
    int FirePointColor;
    };
 void FirePointClass::Activate()
    {
    Randomize();
    X = random(11) - 5; Y = 0;
    Vx = 100.0 - random(200);
    Vy = random(100);
    FirePointColor = 0x000000FF;
    ColorChange = -0x00000001;
    }
 void FirePointClass::Evolution()
    {
    X += 0.01 * Vx;
    Y -= 0.01 * Vy;
    if (abs(Vx) >= 0.1)
        Vx -= 1 * (Vx / fabs(Vx));
    Vy += 0.5;
    FirePointColor += ColorChange;
    }
 void FirePointClass::Drawing()
    {
    int XX = (Form1 -> PaintBox1 -> Width) / 2;
    int YY = (Form1 -> PaintBox1 -> Height) - 20;
    Form1 -> PaintBox1 -> Canvas -> Pixels[XX + X][YY + Y] = TColor(FirePointColor);
    }
 bool FirePointClass::IsBurning()
    {
    return(FirePointColor > 0x00000000);
    }

  FirePointClass FirePoint[N];
 int n;
  //---------------------------------------------------------------------
  void __fastcall TForm1::Button1Click(TObject *Sender)
 {
 n = 0;
 Timer1 -> Enabled = true;
 }
 //---------------------------------------------------------------------
 void __fastcall TForm1::Timer1Timer(TObject *Sender)
```

```
{
TRect Rect = TRect(0, 0, PaintBox1 -> Width, PaintBox1 -> Height);
PaintBox1 -> Canvas -> Brush -> Color = clBlack;
PaintBox1 -> Canvas -> FillRect(Rect);
for (int i = 0; i  CopyRect(Rect, PaintBox1 -> Canvas, Rect);

 if (n                        Edit1 -> Text = n;
}
```

Код программы с комментариями

```
    #include <math> //Подключаем математическую библиотеку

const int N = 5000; //Число "искр"

class FirePointClass //Класс "искр" (конструктор не требуется, т.к. при каждой
//активации "искры" свойствам заново присваиваются новые начальные значения)
  {
  public:
  void Activate(); //Функция, активирующая заново прогоревшую "искру"
  void Evolution(); //Изменения свойств при каждом срабатывании таймера
  void Drawing(); //Функция рисования "искры"
  bool IsBurning(); //Не прогорела ли еще "искра"
  private:
  int ColorChange; //Величина изменения цвета искры за один шаг
  double X, Y; //Координаты
  double Vx, Vy; //Компоненты скорости
  int FirePointColor; //Текущий цвет
  };
void FirePointClass::Activate()
  {
  Randomize(); //Обеспечиваем случайность "случайных чисел"
  X = random(11) - 5; Y = 0; //Задаем начальные координаты
  Vx = 100.0 - random(200); //Задаем начальную скорость
  Vy = random(100);
  FirePointColor = 0x000000FF; //Вначале искра красного цвета
  ColorChange = -0x00000001; //Устанавливаем шаг изменения цвета
  }
void FirePointClass::Evolution()
  {
  X += 0.01 * Vx; //Смещаем искру
  Y -= 0.01 * Vy;
  if (abs(Vx) >= 0.1) //Если горизонтальная скорость больше нуля
     Vx -= 1 * (Vx / fabs(Vx)); //Уменьшаем ее из-за "сопротивления воздуха"
```

```
    Vy += 0.5; //На искру действует "подъемная сила"
    FirePointColor += ColorChange; //Уменьшаем яркость
    }
 void FirePointClass::Drawing()
    {
    int XX = (Form1 -> PaintBox1 -> Width) / 2; //Координаты точки отсчета
    int YY = (Form1 -> PaintBox1 -> Height) - 20;
    Form1 -> PaintBox1 -> Canvas -> Pixels[XX + X][YY + Y] = TColor(FirePointColor);
  //Рисуем искру в виде точки
    }
 bool FirePointClass::IsBurning()
    { //Если цвет искры отличен от черного - значит, она еще горит
    return(FirePointColor > 0x00000000);
    }

  FirePointClass FirePoint[N]; //Объявляем массив искр
 int n; //Число уже горящих искр (для постепенного разгорания пламени)
  //---------------------------------------------------------------------
  void __fastcall TForm1::Button1Click(TObject *Sender)
{
n = 0; //Изначально нет ни одной горящей искры
Timer1 -> Enabled = true; //Запускаем таймер
}
//---------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
TRect Rect = TRect(0, 0, PaintBox1 -> Width, PaintBox1 -> Height);
  //Прямоугольник, закрывающий весь PaintBox
PaintBox1 -> Canvas -> Brush -> Color = clBlack; //Устанавливаем цвет кисти
PaintBox1 -> Canvas -> FillRect(Rect); //И заливаем этим цветом изображение
for (int i = 0; i  CopyRect(Rect, PaintBox1 -> Canvas, Rect);
  //Ключевой момент двойной буферизации - копируем готовое изображение
  //на "основной экран" (здесь - второй PaintBox)

  if (n                    Edit1 -> Text = n; //Выводим количество уже горящих искр
}
```

Советы по улучшению и расширению программы

В принципе данная реализация эффекта пламени в своем методе является вполне полноценной и добавить сюда что-то принципиально новое трудно. Однако, вы можете сделать небольшие дополнения/исправления, например, такого плана: во-первых, можно сделать так, чтобы вектор начальной скорости искры имел всегда одну и ту же длину, но разное направление. В текущей реализации компоненты скорости генерируются

независимо, однако, если случайным образом определять лишь угол вылета (от нуля до двух пи - не забывайте, что аргументами синуса и косинуса являются углы в радианах), картинка станет более достоверной - проверьте!;) Поэкспериментируйте также с интервалом угла разлета и модулем вектора начальной скорости для достижений наилучшего результата.

Также вы можете определять начальные координаты не в пределах небольшого отрезка, как сделано здесь, а смоделировать горение шнура, изогнутого весьма причудливым образом - дугой, кольцом, синусоидой, параболой, синусоидой с затухающей амплитудой... И вообще, вы можете совместить эту программу с программой построения графиков
, генерируя случайным образом t в каком-то интервале, а затем параметрически вычисляя начальные координаты искры и запуская процесс ее горения.

Ну и, разумеется, вы можете и должны использовать двойную буферизацию при выводе графической информации, если предполагается активное обновление изображений. Вы сами видели, что происходит в противном случае!;)