Animation Made Easy

by Xavier Pacheco

This article demonstrates how you can achieve simple sprite animation using Delphi and the Object Pascal Language. It also shows how Delphi simplifies what is usually considered a tedious process since Delphi automatically manages device context for you.

The example that I've created illustrates how you would display a background image (the universe) and draw a sprite image (the UFO) at different locations on the background.

The project's source code is shown in Listings 1 and 2: `ANIMATE.DPR` and `UNIT1.PAS`. These files and the required bitmaps will be included on the free disk which will come with Issue 2 of The Delphi Magazine.

This simple animation example uses three Windows .BMP files: `BACK.BMP` to serve as the main form's background, with `AND.BMP` and `OR.BMP` to make up the sprite image – both are 64x32 pixel bitmaps of a UFO.

A `TSprite` class that I have defined contains the sprite's properties that maintain its location on the form and the `Create()` and `Done()` methods.

`TSprite.Create` creates two `TBitMap` classes, `FAndImage` and `FOrImage`, and reads in the two bitmap files using the `TBitMap.LoadFromFile()` method. It then sets its properties `Top`, `Left`, `Width` and `Height` accordingly. `TSprite.Done` frees the memory used by `FAndImage` and `FOrImage`.

The main form has the variables `BackGnd1`, `BackGnd2` of type `TBitMap` and `Sprite` of type `TSprite`. `BackGnd1` is our original bitmap that we use for our background. `BackGnd2` is the copy of `BackGnd1` to which we perform the `BitBlt()`ing of the sprite image.

The reason we do all the drawing to `BackGnd2` instead of the form's canvas is because to achieve animation we must call `BitBlt()` three times: once to erase the sprite on the form's canvas, once to `AND` `FAndImage` to the form's canvas, and once to `OR` `FOrImage` to the form's canvas. All this drawing to the form's canvas results in a horrible flicker when the image is drawn continuously.

By performing the grunt work on `BackGnd2`, we can copy a rectangle surrounding the old sprite location and new sprite location from `BackGnd2` to the form's canvas with one `BitBlt()` call to eliminate flicker. Therefore, the overhead of maintaining a separate copy of the form's canvas is justified.

`FAndImage` (see Figure 1) effectively creates a black hole where the sprite is to be displayed on the background and preserves the background colors where the sprite does not appear by using the `BitBlt()` function with the `SRCAND` operation.

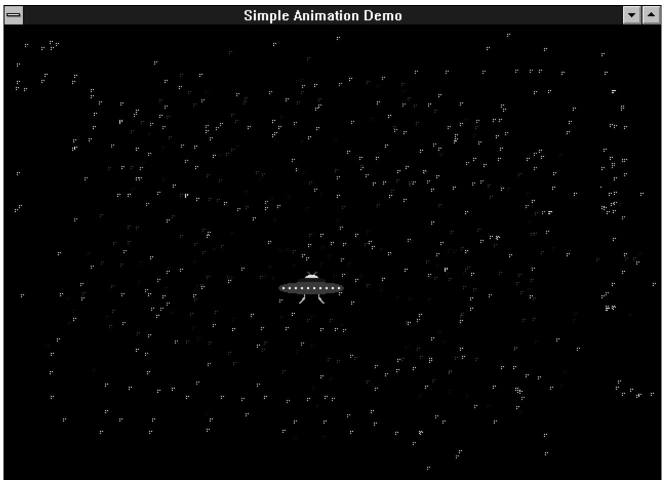
As you can see from the Figure 1, the sprite is shown where the pixel color is black. Since each black pixel has the value 0 and each white pixel has the value 1, when performing an `AND` operation of `FAndImage` to the destination background the resulting color is preserved where `FAndImage`'s color is white. Where `FAndImage` is black, the result is black.

```
BackGround       1001   some color
Image        AND 0000   black
Result           0000   black

BackGround       1001   some color
Image        AND 1111   white
Result           1001   some color
                 (same as Destination)
```

Once I create this black hole, I draw the actual image, still preserving the background's original colors, by `BitBlt()`ing `FOrImage` using the `SRCPAINT` operation.

Notice from Figure 2 that the `FOrImage`'s sprite contains the actual colors while its background is white, or all 1s. You can see from the boolean operation below how `OR`ing the color white to a destination maintains the



The example program running, with the spaceship scooting across a starry sky! It's in full colour of course and this print doesn't do it full justice.

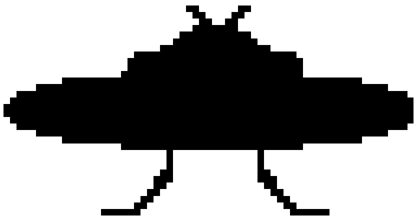


Figure 1

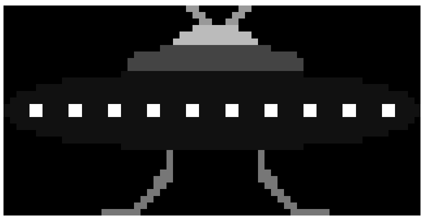


Figure 2

destination's color. Since we are ORing the sprite to an only-black background (our black hole), the sprite's colors are maintained.

```
BackGround      1001  some color
Image        OR 1111  white
Result          1001  some color

BackGround      0000  black
Image        OR 1101  some color
Result          1101  some color
               (same as FOrImage)
```

All the drawing is performed in the TForm1.DrawSprite method. Here, I use some simple logic to keep the sprite within the form's client area.

I then erase the old sprite from BackGnd2, re-draw it in BackGnd2 at the new location, and finally copy a rectangle from BackGnd2 to Form1.canvas to effectively erase and re-position the sprite on Form1's canvas.

TForm1.MyIdleEvent is where TForm1.DrawImage is called. I then assign this method to the Application.OnIdle event handler in TForm1.Create. The method Application.OnIdle, as the name implies, is executed when the application is in Idle.

TForm1.Paint BitBlt()s the original background, BackGnd1, to its canvas.

Notice the TSprite is not a component in and of itself, that is, a descendant of an original Delphi component such as TControl or TGraphicControl.

The reason I did this was because the form repaints itself whenever making changes to any child controls causing a yucky flicker on the screen. Also, the TSprite object was simple enough that I didn't really need any data or methods from an already existing object.

Although this example is very simple, it is possible to extend the functionality of TSprite to be more self contained, such as maintaining it's own direction, drawing itself, and being a non-static image, that is an image that changes as it is moved on the background.

Also, I didn't do anything special in this example to create true bounces – something I can keep for a later project!

Xavier Pacheco is a Consulting Engineer at Borland International. You can reach Xavier on CompuServe at 76711,666 or at xpacheco@wpo.borland.com

Listing 1 ANIMATE.DPR

```
program Animate;
uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Form1};
{$R *.RES}
begin
  Application.CreateForm(TForm1,
    Form1);
  Application.Run;
end.
```

Listing 2 UNIT1.PAS
[Sorry about the small text size, it's the only way we could get it all in I'm afraid, but the code will be on the disk with Issue 2. Editor]

```
unit Unit1;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages,
  Classes, Graphics, Controls,
  Forms, Dialogs, Menus, Stdctrls;
type
  TSprite = class
  private
    FWidth: integer;
    FHeight: integer;
    FLeft: integer;
    FTop: integer;
    FAndImage, FOrImage: TBitMap;
  public
    property Top: Integer read FTop write FTop;
    property Left: Integer read FLeft write FLeft;
    property Width: Integer read FWidth
      write FWidth;
    property Height: Integer read FHeight
      write FHeight;
    constructor Create(AOwner: TComponent);
    destructor Done;
   end;
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    BackGnd1, BackGnd2: TBitMap;
    Sprite: TSprite;
    GoLeft,GoRight,GoUp,GoDown: boolean;
    procedure MyIdleEvent(Sender: TObject;
      var Done: Boolean);
    procedure DrawSprite;
  end;
const
  BackGround = 'BACK.BMP';
var
  Form1: TForm1;
implementation
{$R *.DFM}
constructor TSprite.Create(AOwner: TComponent);
begin
  inherited Create;
  FAndImage := TBitMap.Create;
  FAndImage.LoadFromFile('AND.BMP');
  FOrImage := TBitMap.Create;
  FOrImage.LoadFromFile('OR.BMP');
  Left := 0;
  Top := 0;
  Height := FAndImage.Height;
  Width := FAndImage.Width;
end;
destructor TSprite.Done;
begin
  FAndImage.Free;
  FOrImage.Free;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  BackGnd1 := TBitMap.Create;
  with BackGnd1 do begin
    LoadFromFile(BackGround);
    Parent := nil;
  end;
  BackGnd2 := TBitMap.Create;
  with BackGnd2 do begin
    LoadFromFile(BackGround);
    Parent := nil;
  end;
  Sprite := TSprite.Create(self);
  GoRight := true;
  GoDown := true;
  GoLeft := false;
  GoUp := false;
  Application.OnIdle := MyIdleEvent;
  ClientWidth := BackGnd1.Width;
  ClientHeight := BackGnd1.Height;
end;
procedure TForm1.MyIdleEvent(Sender: TObject;
  var Done: Boolean);
begin
  DrawSprite;
  Done := false;
end;
procedure TForm1.DrawSprite;
var
  OldOrigin: TPoint;
  TempRect: TRect;
begin
    With OldOrigin do begin
      X := Sprite.Left;
      Y := Sprite.Top;
    end;
    with Sprite do begin
      if GoLeft then
        if Left > 0 then
          Left := Left - 1
        else begin
          GoLeft := false;
          GoRight := true;
        end;
      if GoDown then
        if (Top + Height) < self.ClientHeight then
          Top := Top + 1
        else begin
         GoDown := false;
         GoUp := true;
        end;
      if GoUp then
        if  Top > 0 then
          Top := Top - 1
        else begin
         GoUp := false;
         GoDown := true;
        end;
      if GoRight then
      if (Left + Width) < self.ClientWidth then
        Left := Left + 1
      else begin
        GoRight := false;
        GoLeft := true;
      end;
  end;
  {Erase the old sprite in BackGnd2 }
  with OldOrigin do
    BitBlt(BackGnd2.Canvas.Handle, X, Y,
      Sprite.Width, Sprite.Height,
      BackGnd1.Canvas.Handle, X, Y, SrcCopy);
 {Draw the sprite at the new location in BackGnd2}
  with Sprite do begin
    BitBlt(BackGnd2.Canvas.Handle, Left, Top,
      Width, Height, FANDImage.Canvas.Handle,
      0, 0, SRCAND);
    BitBlt(BackGnd2.Canvas.Handle, Left, Top,
      Width, Height, FOrImage.Canvas.Handle,
      0, 0, SRCPAINT);
  end;
  {Copy a rectangle from BackGnd2 to erase and
   reposition the sprite to the form's canvas}
  with OldOrigin do
    BitBlt(Canvas.Handle, X-2, Y-2,
      Sprite.Width+2, Sprite.Height+2,
      BackGnd2.Canvas.Handle, X-2, Y-2, SrcCopy);
end;
procedure TForm1.FormPaint(Sender: TObject);
begin
  BitBlt(Canvas.Handle, 0, 0, ClientWidth,
    ClientHeight, BackGnd1.Canvas.Handle,
    0, 0, SrcCopy);
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  BackGnd1.Free;
  BackGnd2.Free;
  Sprite.Free;
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  DrawSprite;
end;
end.
```