Fire

COLLABORATORS

	TITLE : Fire	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 24, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Fire

1.1 main

How to code youre own "Fire" Routines
By Phil Carlisle (aka Zoombapup // CodeX) 1994.
Email: pc@espr.demon.co.uk

Hiya Puppies!

Hopefully this information file will give you all the information you need to code youre own fire routines, seen in many demo's and also to actually take it all further and develop youre own effects..

Ok, so lets get on....

Setting Up

First thing we need to do is set up two arrays, the size of the arrays depends on the many things, screen mode, speed of computer etc, its not really important, just that they should both be the same size... I'll use 320x200 (64000 byte) arrays for this text, because that happens to be the size needed for using a whole screen in vga mode 13h.

The next thing we need to do is set a gradient palette, this can be smoothly gradiated through ANY colours, but for the purpose of this text lets assume the maximum value is a white/yellow and the bottom value is black, and it grades through red in the middle.

Ok, we have two arrays, lets call them startbuffer and screenbuffer, so we know whats going on. Firstly, we need to setup an initial value for the start buffer... so what we need is a random function, that returns a value between 0 and 199 (because our screen is 200 bytes wide) this will give us the initial values for our random "hotspots" so we do this as many times as we think is needed, and set all our bottom line values of the start buffer to the maximum colour value. (so we have the last 300 bytes of the start buffer set randomly with our maximum colour, usually if we use a full palette this would be 255 but it can be anything that is within our palette range.)

Ok, thats set the bottom line up.. so now we need to add the effect, for this we need to copy the start buffer, modify it and save it to the

screenbuffer, we do this by averaging the pixels (this is in effect what each byte in our array represents) surrounding our target....

It helps to think of these operations in X,Y co-ordinates....

Lets try a little diagram for a single pixel.....

```
    This is the startbuffer             This is our screenbuffer


    0,00,10,20,30,4 etc...

    1,01,11,21,31,4 etc..            X,Y

    2,02,12,22,32,4 etc..
```

Here we're going to calulate the value for X,Y (notice I didnt start at 0,0 for calculating our new pixel values?? thats because we need to average the 8 surrounding pixels to get out new value.. and the pixels around the edges wouldn't have 8 pixels surrounding them), so what we need to do to get the value for X,Y is to average the values for all the surrounding pixels... that means adding 0,0 0,1 0,2 + 1,0 1,2 + 2,0 2,1 2,2 and then dividing the total by 8 (the number of pixels we've takes our averages from), but there's two problems still facing us..

1) The fire stays on the bottom line....
2) Its slow....
3) The fire colours dont fade...

Ok, so first thing, we need to get the fire moving! :) this is really VERY easy. All we need to do is to take our average values from the pixel value BELOW the pixel we are calculating for, this in effect, moves the lines of the new array up one pixel... so for example our old X,Y value we were calculating for was 1,1 so now we just calculate for 2,1 and put the calculated value in the pixel at 1,1 instead.. easy..

The second problem can be approached in a few ways.. first and easiest is to actually calculate less pixels in our averaging.. so instead of the 8 surrounding pixels we calculate for example, 2 pixels, the one above and the one below our target pixel (and divide by 2 instead of 8) this saves a lot of time, another approach is to use a screen mode, where you can set 4 pixels at a time, or set up the screen so that you can use smaller arrays (jare's original used something like 80X50 mode) which in effect reduces to 1/4 the number of pixels needed to be calculated.

The third problem is just a matter of decrementing the calculated value that we get after averaging by 1 (or whatever) and storing that value.

Last but not least, we need to think about what else can be done... well, you can try setting a different palette, you can also try setting the pixel value we calculated from to another place, so say, instead of calculating from one pixel below our target pixel, you use one pixel below and 3 to the right of our target... FUN! :))

Well, I hope I didnt confuse you all too much, if you need anything

clearing up about this, then email me at pc@espr.demon.co.uk ok?

The pascal source code is in :Source/Article/Fire.pas.