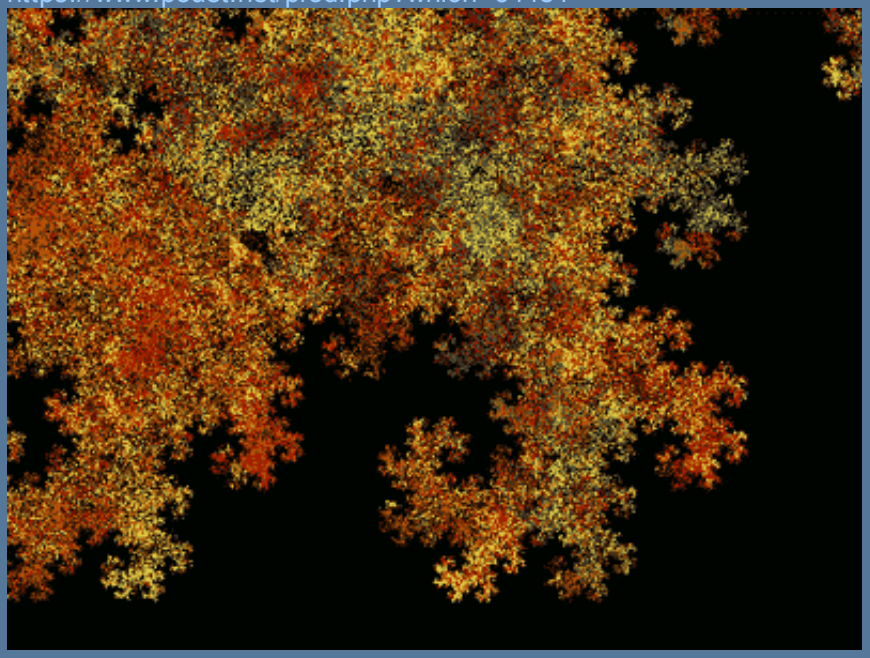
Superfunction Compo - Fire Coral - HellMood / DESiRE

1 message

Héll Mood <helmut.toedtmann@gmail.com> Thu, Sep 10, 2015 at 11:28 PM
To: Héll Mood <helmut.toedtmann@gmail.com>

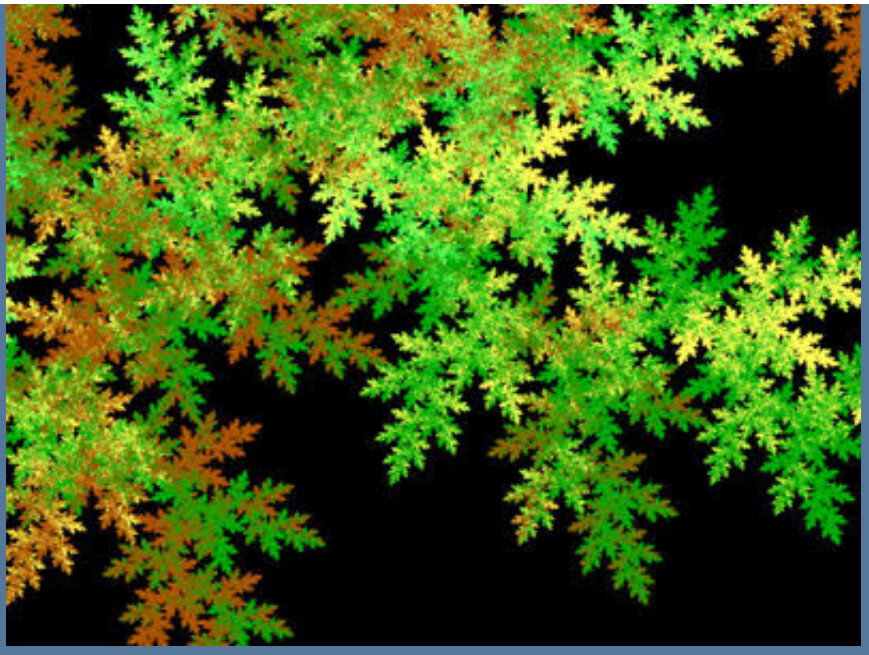
An attempt to explain details and correlations between

"Fire Coral", "Dragon Fade" and "Autumn"
https://www.pouet.net/prod.php?which=64484



https://www.pouet.net/prod.php?which=63522

https://www.pouet.net/prod.php?which=64159



1.) DRAWING TECHNIQUE
We use the "Set Pixel" function of interrupt 0x10
CX = x, DX = y , AL = color, AH = 0x0C (set pixel)
http://en.wikipedia.org/wiki/INT_10H
NOTE : Depending on the Graphics BIOS, you might
see different results when plotting negative
coordinates, or exceeding the legal amount of
colors. Here, one underflow is accepted, the other
is catched with a signed flag branch
"sub cx,dx" + "js Skip" = don't draw on negative x
2.) THE FRACTAL
We use a simple iterated function system (IFS)
consisting of two functions with equal probability.
http://en.wikipedia.org/wiki/Iterated_function_system
We use a discrete, scaled up version to save space.
This also fits the drawing technique in a good way.
Used in sizetros:
Dragon Fade :
https://www.pouet.net/prod.php?which=63522
Autumn :
https://www.pouet.net/prod.php?which=64159
NOTE : Obviously keeping both functions similar
and having coefficients of "0.5" enables binary
commands (arithmetic shift!) and uses little space.

```
IFS1    CX      DX
initial x       y
add dx,cx       x       x+y
sar dx,1        x       (x+y)/2
sub cx,dx       (x-y)/2 (x+y)/2
```

Scaled down, normalized floating point equ.
f(x) = 0.5x - 0.5y f(y) = 0.5x + 0.5y

```
IFS2    CX      DX
initial x       y
add dx,cx       x       x+y
sar dx,1        x       (x+y)/2
sub cx,0x533    x-0x533 (x+y)/2
sub cx,dx       (x-y)/2-0x533   (x+y)/2
```

Scaled down, normalized floating point equ.
g(x) = 0.5x - 0.5y - 1 g(y) = 0.5x + 0.5y
f(x) & g(x) written in coefficients with probability
0.5 -0.5 0.5 0.5 0 0 0.5
0.5 -0.5 0.5 0.5 -1 0 0.5
copy paste to http://cs.lmu.edu/~ray/notes/ifs/
to see what it looks like (and to explore further)
check http://guyanddeb.com/af.html to see, where
the fractal is located on the plane (clipping!)

separate coefficients with commas there, and
supress probability, then hit "Correct Input"
0.5, -0.5, 0.5, 0.5, 0, 0
0.5, -0.5, 0.5, 0.5, -1, 0
Adjust xmin, xmax, ymin, ymax and hit "Draw"
3.) MAKING THE CHAOS PREDICTABLE
Normally, to implement the "chaos game" version
of iterated function systems, you would need a
random number generator. To be precise, you would
need a VERY GOOD one, as carefully pointed out by
Peitgen, Jürgens and Saupe in "Chaos and Fractals:
New Frontiers of Science" (just google for "random
number generator pitfall") Surprisingly enough, if
done right, one doesn't need ANY rng - at all!
The secret lies in converting the chaotic infinite
real number system into a deterministic, finite,
discrete system with a rather short period, somewhat
like a "knights tour" on a chess board. In other words,
we need a function F which we apply N times to a point P
so that
F(F(F(F ... F(P)))) = F^N(P) = P
and so that every pixel on the plane inside the fractal
is actually "visited" once per cycle. It turns out, that
at least one such function exists (several, actually),
characterized by the magic number 0x533, and the branching
criteria "sum of x and y is even" (checking the carry
flag after "add dx,cx" and "sar dx,1") Currently, i
dont't know exactly what "magic numbers" produce stable
plane filling cycles, how they are connected to the
branching criteria, or how to find them, other than by
trying. I'll leave that for later - or for the
mathemagicians amongst us ;)
Link to "random number generator pitfall"
TL/DR : the final function is both : the fractal, as well
as the random number generator it needs =)
NOTE : Every pixel is drawn once per cycle, with the very
same color each cycle, which makes it a rather still image.
The "noise effect" happens due to the drawing technique,
different points of the fractal are projected to the same
screen coordinates.
4.) COLORING
The idea is to code the branching history directly
into the pixel color, so that all pixels which branched
equally in their last 4 iterations are plotted with the
same color. "shl ax,1" keeps all the iteration history,
except one (the oldest), and the free (lowest) bit is
marked according to the latest. "inc ax" is just
a shorter version of the more comprehensive "or ax,1"
This has been used in "Dragon Fade", and generates
very regular color structures. To achieve a mixture
between random and regular structures, we can use the
fact, that we have several ways to initialize the graphic
mode to 0x12 from an initial emtpy AL, "xor" beeing one
of them. "xor al, 0x12" works together with "shl ax,1"
as a minimized xorshift randomizer, which is yet still
dominated by the regularity of the branching history.
"aam ??" is a tuneable coloring parameter.
NOTE : "aam ??" and skipping the drawing of certain
points, while keeping track of iteration history, lifts
the coloring beyond simple predictability. At this point,
i stopped analyzing and was satisfied with the
tuning parameters i found ;)
5.) USER EXIT FUNCTION
Normally, you would use the infamous "in al,0x60" to
check if the user wants to ESCape. Unfortunately, AL
holds the current pixel color all the time, so that
at least two bytes (push ax,pop ax) would be needed
to do it that way. Luckily, MSDOS provides another
function to check the keyboard, modifying the Zero Flag

```
if some key was pressed. And since a simple "mov" does
not alter that flag, we can smoothly weave that into
the code. Still, all together this takes up 5 bytes,
and i am somewhat happy, that there was enough space
to include this.
http://www.ctyme.com/intr/rb-1755.htm
6.) ADDITIONAL NOTES
The fading effect of "Dragon Fade" is actually just
a well tuned coloring function. Since every pixel is
"touched once" per cycle, and colors above 0x80 are
"xor"ed onto the screen, you just need to make sure,
that the color function remains constant for each
pixel (for consecutive iterations).
http://webpages.charter.net/danrollins/techhelp/0127.HTM
To be independent from "magic numbers", you can delay
the iteration history. You basically choose the function
by the branch another pixel took X iterations ago. Simply
"rcr bp,1" or "rcr ebp,1" does the trick (between generating
and checking the carry flag). This advanced selection
is used in "autumn". It is basically needed there, because
the average achieved period is way too low to cover every
pixel of the screen. Just change the delay from 32 to 16 by
replacing "rcr ebp,1" with "rcr bp,1" to see what i mean.
Note, that even if you don't get total garbage with "unlucky"
numbers using this technique, some numbers still do better
than others. But the chance to find one, and the overall
screen coverage is really heavily increased.
Note, that besides clever usage of just one call to
int 0x10 and some flag shenanigans (carry, sign, zero)
no real "dirty" tricks have been used here, like, odd
alignment, self modifying code etc. Checking politely
for a key and quitting takes up 5 bytes, "sub cx,0x533"
occupies a whooping 4 bytes, and in fact the shortest
fading dragon i can come up with, is 23 bytes long.
What i want to point out is, this is nowhere near the
limit, when we think 32b, and i think, it can be MUCH
more impressive when we think 64b (think about a lot
better "autumn").
Greets and Respect go to : rrrola, frag, Baudsurfer,
Sensenstahl, Whizart, g0blinish, Rudi, orbitaldecay,
igor, Drift and all Desire members =)
If you have further questions, don't hesitate to
mail me : storryteller@hotmail.com
```

Code:

```
Start
xor al, 0x12    ; Set 640*480*16 Mode, also Coloring (special)
int 0x10        ; Set Resolution, also : Set Pixel
Skip
add dx,cx       ; Shared IFS
shl ax, 0x01    ; Coloring (base)
sar dx, 0x01    ; Shared IFS
jnc Branch      ; Deterministic alteration
sub cx, 0x533   ; Special IFS (Difference)
inc ax  ; Coloring (base)
Branch
aam 0x0E        ; Coloring (special)
sub cx,dx       ; Shared IFS
js Skip ; Clipping
mov ah, 0x01    ; User Exit Function
int 0x16        ; User Exit Function
mov ah, 0x0C    ; Set Command to "Set Pixel"
jz Start        ; User Exit Function
ret     ; Quit
```