MOCHA

by Wes Cherry (wes@scumby.com)

MOCHA is a program which generates images by evaluating a bene for each pixel of the image. The benes, called benes (gene/bean, pronounced bean) typically involve x and y, where x and y are the coordinates of the pixel being computed. The computed result of the bene is stored as a pixel value at position x,y. For example, the bene

y

will generate a vertical wash of grays going from black at the top to white at the bottom. (by default 0=black and 255=white).

Benes can also involve the variable t. In this case the image is evaluated successively creating an animated "movie". For example, the bene

y+t

will cause the wash of grays to scroll upwards.

Source Images

You can also load a source image (File/Open Source Image...). Source bitmaps can be accessed using the array operator. Thus to display the source bitmap as is, the bene [x,y] would display it as is. Similarly, [y,x] will rotate it. Images of peoples faces work particularly well with some array operator benes.

Breeding

When in Breed Mode, Mocha will also automatically alter the bene you enter, creating twelve variations (children) of your bene. The parent bene is displayed in the center of the Mocha window with it's twelve children around it. You can choose a variation you like by clicking on it with the mouse. This makes it the next parent, generating new children. After many generations you can "breed" some interesting benes. Many of the benes in this file were generating by breeding.

Palettes

You can also specify the palette that is used for displaying the image. The pixel value computed with the benes described above specifies an index of 0 to 255 into the palette. The Red, Green and Blue components of the palette are specified separately using assignments to the variables r,g and b. When palette benes are evaluated x is used to indicate the current palette index being computed. Thus the default palette of a gray wash is specified via

```
r=x
g=x
b=x
```

If you wanted a blue wash you would specify the palette with

```
r=0
g=0
b=x
```

You may also use the Hue, Saturation, Lightness (HSL) color model to specify color. It is often much easier to create a good palette using HSL. H specifies the hue (0..255 map to Red-Yellow-Green-Cyan-Blue-Magenta). S is color saturation (0..255) and L is lightness (0..255 map to black-white). A nice rainbow swash can be specified

by:

h=x
s=255
l=128

Palettes may also be specified using assignments to p. Use the pal(x, offs) function when using p. The first argument to pal specifies the index into an array of predefined palettes (currently about 30). offs specifies the offset whithin the palette (0..255). Thus

p=pal(13,t)

will rotate palette 13.

Since Mocha has to evaluate the bene for each pixel in the image Mocha is very computationally intensive. To run well it should be run on at least a 486 or better computer. Benes involving t in particular look much better on a fast 486 or a Pentium computer.

Syntax

Benes are given in normal infix form.

Operators

+ : Plus
- : Minus (binary & unary)
* : Mult
/ : Div
% : Mod
^ : Power
< : Less Than
<= : Less Than or Equal
> : Greater Than
>= : Greater Than or Equal
<> : Not Equal
== : Equal
?: : Ternary operator (x ? y : z <= evaluates to y if x != 0 else evaluates to z)
[,] : Array operator -- indexes into the source bitmap at pixel [x,y]. Source bitmap
: must first be loaded via File/Open Source Bitmap

Variables

x : x position of current pixel
y : y position of current pixel
t : the current frame number
xm : the current mouse x position
ym : the current mouse y position

Constants

X : the X extent of the image
Y : the Y extent of the image

Functions

sin(x) : period =256, range = 0..255
cos(x) : period =256, range = 0..255
tan(x) : period =256, range = 0..255^2
asin(x) : inverse sin of x
acos(x) : inverse cos of x
atan(x) : inverse tan of x
dsin(x) : decayed sinusoidal
theta(x,y) : returns angle (0..255) of vector from center of image to x,y
rad(x,y) : returns radius of vector from center of image to x,y
saw(x) : sawtooth function, period = 128, range = 0..255
tri(x) : triangle function, period = 256, range = 0..255
decay(x) : decay function = int(exp(-i/64)*255)
exp(x) : = exp(x/10)
abs(x) : abs value of x
rand(x) : range = 0..x
sqrt(x) : square root of x
peg(x) : pegs x to 0..255 (best applied to the result of an bene)
frac(x) : fractal wave function
snd(x) : sound waveform at x (0..255)
sndl(x) : left channel of sound waveform at x (0..255)
sndr(x) : right channel of soudn waveform at x (0..255)
vol(x) : sound volume at x (0..255)
fft(x) : fast fourier transform of sound waveform (0..255)
pal(x,offs) : specifies which built in palette to use. See palette documentation.

Any of these functions may be plotted by using an bene of the form:

fn(x)==255-y ? 255 : 0

thus to plot a sine wave you would use the bene:

sin(x)==255-y ? 255 : 0

Commands

File

Enter Bene Expression... : Allows you to manually enter an bene and specify it's width, height and Frames Per Second.
Evaluate... : Starts evaluating the current bene.
Open Bene Expression : Open a saved bene expression stored in mocha.moc.
Save Bene Expression : Save a bene expression to the mocha.moc file.
Open Source Image... : Opens a source image to operate on with the [,] operator.
Save Image... : Saves the currently displayed image as a BMP or GIF.
Save Animated GIF/AVI : Writes an animated GIF or AVI file.
Disassemble... : Writes the current compiled bene to mocha.asm (for those who are curious)

Edit

Undo : Backs up one generation
Copy : Copies the current image and it's bene to the clipboard
Paste : Pastes the bene or image from the clipboard

Options

Tile : Tiles the image in the mocha window. Resize the mocha window to see this.

Stretch to Window : Stretches the image to the mocha window.
Background : Sends Mocha to the background, tiling and full screen.
Full Screen : Sizes Mocha window to cover the entire screen.
Breed Mode : Displays parent and 12 mutated children
Animate All in Breed : Animates all images when in breed mode.
Mutation Rate High : Sets the mutation rate high -- this can make some twisted children
Mutation Rate Normal : Sets the mutation rate to a normal value.
Mutation Rate Low : Sets the mutation rate to a low value. Useful for fine-tuning a bene.
Master... : Makes this instance of Mocha the master. Pressing Ctrl-Enter will send the current expression to the current slave. Master slave mode is great for performances because it allows you to compose a cool image while the slave is displaying the last cool image you have created.
Slave... : Makes this instance of Mocha the slave.
Send To Slave : Sends the current bene to the slave. Available only when in Master mode.
Display Frame Rate : Displays animation rate in number of frames per second

Scale

Full->1/8 : Scales the image down. Useful on slow machines or when using complex expressions.

Help

Help... : This thing.
About : Vanity box.

Shortcuts

Keyboard

Space : Enter Bene Expression
Enter : Begin Evaluating Current Bene
Ctrl-S : Save Bene Expression
Ctrl-O : Open Bene Expression
Ctrl-A : Toggles Cycle All when in Breed Mode
Ctrl-F : Toggles Full Screen Mode
Ctrl-T : Toggles Tile
Ctrl-B : Toggles Breed Mode
Ctrl-W : Toggles Stretch To Window
Ctrl-1 : Toggles Display Frame Rate
Esc : Halt Bene Evaluation
Ctrl-Enter : Send Image to Slave
Up : Increases the scale
Down : Decreases the scale
Left : Undo
Right : Mutate current Bene

Mouse Shortcuts

Left Mouse Button : Mutates current Bene
Right Mouse Button : Undo/Mutate again.

Sample Bene Expressions

Here are some sample benes. There are also some good sample expressions included in the default mocha.moc file. Choos File/Open Bene Expression. You may select these expressions in Help and copy them to the clipboard. Activate Mocha and paste them in with Edit/Paste.

Simple benes to try
x+y
sin(x)
sin(2*x)
sin(x+y)
theta(x,y)

Egg Carton
sin(x+sin(y+sin(sin((sin(x*2)+sin(y*2))/2))))
sin(x*t+sin(y+sin((sin(x*2)+sin(y*2))/2)))
sin(sin(x*t-sin(y+sin((sin(x*2)+sin(y*2)-y-6)%8-2)-x)))

Marching Melting Mountains
sin(cos(6*y+sin(x+cos(x+4*t)))+6*x)
sin(cos(6*y+sin(x+cos(x+sin(x+4*t))))+6*x)

Oscillating Too Dark Park
sin((((x-X/2)^2+sin((t-20)*x)*12+(y-Y/2)^2+sin(2*y)*((sin(t)-128)))/100)

Fractal Wipeout
sin(x*sin(y)*100/cos(t))

Jazz
sin(x*(t-100)/10+sin(x*t+sin(y+sin((x*2)+sin(y*2))/2))

Fraktal Supernova
r=x
g=x
b=0
cos(abs(rad(x,y)%tri((x*tri(t)+5-8+21)*y)))

No Name
r=0
g=(x*7)
b=(x*42)
tan(x)/-4/sin(saw(saw((cos(y+t)+(x--15))/-16+sin(t))))

flying saucer
sin(x+y%rad(x,y*t/4))

Cursor position benes:

sin((((x-X/2)^2+(y-Y/2)^2+sin(2*y)*(xm-64)*5+cos(5*x)*(ym-64)*5))/100+t*40)
frames=1000

Image manglement benes

The following benes require you to first generate an image from this bene and then use File/Save to save it and then File/Open source bitmap. Many of these images tile well and make good windows desktop bitmaps.

(sin(x+cos(y))+cos(y+sin(x)))/2
creates mocha.bmp

Using mocha.bmp for bitmap:
[[x,y],[y,x]]
[[[x,y],x],y][[sin(x),cos(y)],[sin(y),cos(x)]]

[sin([cos(x),sin(y)]),cos([cos(y),sin(x)])]
[[[sin(y+t*5),(x+cos(x+t*4))%255],y],x]
[[[sin(y+t*5),(x+cos(x+sin(t*4)))%255],x],y]
[sin(t+6+sin(x)+cos(y)),cos(t*4+y+sin(x+t))]
[sin(t+6+sin([x,y]*t)/cos(t+[y,x])+cos([y,sin(x)])),cos(t*4+y+sin(x+t))]
[[[x,y+cos(t*3)],x],sin(y+t*5)]
[[peg(sin(y+t)+cos(x+t)),peg(cos(y+t)+sin(x-t))],sin(x)]
[[peg(sin(y+t)+cos(x+t)),peg(cos(y+t)+sin(x-t))],peg(exp(x+t*4)/sin(y))] (lava gears)

General warp (tileable at 256x256)
[sin(x)*cos(x+y)/sin(y),cos(y+x)]

Blowup!
[x+(cos(x)-128)/(20-t),y+(sin(y)-128)/(20-t)]
frames=18

Cyclic Inflation
[x+(sin(x)-128)*16/(cos(t*4)+32),y+(cos(y)-128)*16/(cos(t*4)+32)]
frames=1000

The Stimpy Transform (best w/ people faces)
[x+(sin(x*t)-128)/(t*4),y+(sin(y*t)-128)/(t*4)]
frames=20

Mouse Controlled Woogilating
[x+(sin(x+t*xm)-128)/8,y+(sin(y+t*ym)-128)/8]
frames=infinite

Wiped Monty
[abs(x+(exp(x+1-y*2+t*8)-128)/16)%X,abs(y+(cos(x+(y*2)+(t*8))-128)/6)%Y]

Perpetual Inflation
[abs(x+(exp((15+1-y*2)*t*8+y)-128)/16)%X,abs((cos(x+(y*2)+t*8)-128)/-5+y-3)%Y]

Funhouse Mirrors
[sin(x+exp((21-y*2)*t*8-y)*128/10)%X,abs((cos(x+(y*2)+t*8)-128)/11+y-3)%Y]

Moire Yer Face
[abs(x+(exp((15*15-y*2-t)*8+6+y)*1-141)/25)%X,abs((cos((x+x)*(y*2)+t*8)-129+t)/--10+y-3)%Y]

Gritty Zitties
[abs(x+(exp(14+y)*1-121)/13)%X,abs((cos((x+x)*y*2*t*8)-114+t+x)/-10+y-22)%Y]

Rotate Bitmap To View
[x+(y*4/t),y-(x*4/t)]

Monet
[x-t/2+rand(t),y-t/2+rand(t)]

Other Interesting Good Parents
t*exp(tan(x+y-t-t)/35*(7+x)-1-t+t-128-7)+17+x
exp(sin(5-t))*(peg(cos(y))-cos(x))
[16%X/0,abs(y+sin((x*(y-1)/-18+9)/(t-16))-128-t+27)*7%Y]
peg(cos((6+y)%sin((x/t*14+y)/3))%(6*x))
abs((((cos(8-x-y-4)%x+y)/7+t)/37-6)%x*y*-1-24+x)

abs(((cos(8-x-y-4)%x+y)/7*t/37-6)%x*y*-1-24+x)

Great Backgrounds

These guys tile well (Options/Background), Try them at 1/4 or 1/8 scale.

r=x
g=0
b=x
theta(cos(x),sin(y))+6*t+x+y
set width = height = 256

(theta(cos(x)-t,sin(y))-x+t)*(-16+t)-x+t+y
(((theta(5*5-6-1+x-t,(3+y-2)*t+18)--20-t)*-53-5)*4*-6*t+3+y+y--2+19-6-t+t)*-4-x+t-7+y

pinwheels
sin(theta(cos(x),saw(y))--16*t)-x+y
sin(theta(cos(x),saw(y))+16*t)-5+y

Palette functions

((x-X/2)^2+(y-Y/2)^2)/144
r=255-sin(x+t)
g=cos(x+t/3)
b=sin(255-x+t/23)
Frames: 10000

(sin(x+cos(y+cos(t)))+cos(y+sin(x+sin(t))))/3
r=cos(x+t)
b=sin(x+t*3)
g=cos(x+t*5)
Frames: 10000
make it small

r=sin(x+t*5/3)
g=cos(x+t*31/29)
b=sin(x*cos(t)/8+t)
sin((y-saw(x+y-5+t+8)/x-128)/16)%Y+tri(x%(sin(((x+y)+1)*(t+-9))-128+x-8)+8)-X

Some notes on code generation:

In principle, it's pretty simple, the optimizations have gummed it up a bit though.

Benes are parsed into a abstract syntax tree (AST). The code generator then traverses the AST tree matching twigs of the tree with templates for code. For example, consider the bene x+y. This will make a tree of the form:

```
    +
   / \
  x   y
```

This matches a code template for a plus w/ the left child being an bene and the right child an bene:

```
{ntPlus, ttExn, ttExn, {asm code}}
```

the asm code looks something like this (the result of a code hunk is always returned in EAX):

```
{opEvLeft, pushAx, opEvRight, popBx, AddAxBx}.
```

the opEvLeft and EvRight instruct the code generator to gen code for the left and right nodes. each of the variables will emit code like:

```
{ntVar, ttNil, ttNil, {MovAxGlobal}
```

so the final code is something like:

```
    mov         eax,x
    push        eax
    mov         eax,y
    pop         ebx
    add         eax,ebx
```

This is the simple case. There's other code fragments for twigs which can generate better code. The real code that matches the above x+y example is actually:

```
{ntPlus, ttExn, ttVar, {opEvLeft, opAddAxVarRight}}
```

And, since x is referenced so often I store it in ESI, it's actually:

```
{ntPlus, ttExn, ttVarSI, {opEvLeft, opAddAxSi}}.
```

These are optimizations -- the only code fragments that are required are for the {nt, ttExn, ttExn} as well as the code fragment to load factors (variables, constants) into EAX.

The optimizer does other opts such as register allocation rather than pushing temp values on the stack, strength reduction, and tree rotation about commutative operators to minimize temp saves.

There's code fragments for all operators, some of the functions are inlined as well.

Why Mocha? Well, Mocha was dreamt up over a series of caffeine binges with my friend Chris Pirih. He wrote some of the image management functions a long time ago but denies any responsibility for all this.