A collectin of screensavers:

Alarm:
no description

MartinView

MartinView is a port of an X-Windows program called xmartin written by Ed Kubaitis at UIUC which I have included with its man page as a tribute and source. I saw this running on a local machine and said "that's cool, that should be on a NeXT", and where else but on BackSpace.

Why the name MartinView you ask? Well, taken from the xmartin source code:

[The] Hopalong [algorithm] was attributed to Barry Martin of Aston University (Birmingham, England) by A. K. Dewdney in the 9/86 Scientific American. The Ranf
portable random number generator is based on work by D. H. Lehmer, Knuth,
David Sachs(Fermilab) and Curt Canada(NCSA).

Installation Instructions

This directory contains the source for MartinView, a BackSpace screen-saver module that works with BackSpace v3.0. This module needs to be installed so that BackSpace can find it, but by default, make will not install it. The following will install the module in your **~/Library/BackSpaceViews** directory:

```
make install
```

BackSpace will actually find the module if it is in one of the following places:

~/Library/BackSpaceViews,
its "app wrapper" folder
/LocalLibrary/BackSpaceViews

You can specify an alternate installation directory like this:

```
make install INSTALLDIR=/LocalLibrary/BackSpaceViews
```

Inspector Panel Options

The first thing to notice is that I added so many options, that I needed to use a separate panel for the Inspector instead of the default box view. Thanks for this option Sam!
I wanted to be as true to the original xmartin source and offer as many features as possible.

Many features have Auto switch buttons which determine whether their values are determined automatically (in some cases meaning randomly) or whether they are set to a specific user defined value.

The **Function** pop-up button specifies which of the four algorithms is in use. With Auto on, one of the four functions is chosen randomly based on percentages specified in MartinView.m.

The **Points** box determines the **maximum number of in-range points** to use (those that fall within the coordinates of the view and the **maximum total points** (includes all points calculated). Both are necessary to insure proper completion of one set because given the random nature of the parameters you sometimes get results outside the view's range. With Auto turned on, the maximum values are calculated as a percentage of the size of the view.

The **Hopalong Parameters** are specified by three floating point values and six boolean values

basically determining signs. See the -oneStep method in MartinView.m for more specifics on what these values do for each algorithm. With Auto turned on, their values are randomly determined.

You can specify a **seed perturbation** value and how often this perturbation should be applied. With Auto turned on these values are determined randomly. A **magnification factor** can be specified for the entire set and the Auto option always returns a value of 4 for the martin2 function, and 1 for the other functions. An overall **displacement** vector can be specified in the default coordinate system of the view as well.

The **flush buffer interval** specifies the number of pixels to compute before flushing them to the screen. The default flush buffer size is 400 pixels in one call to NXRectFillList, with a maximum value of 900. I think any amount higher than 900 or so may cause a slow down with respect to time/pixel. So the only essential usefulness of this option is to slow down the drawing of the fractal by setting the interval to a low number. The original xmartin code allowed for the creation of the image off-screen and a complete flush to the screen at one time, but that is no fun, and I think this is a good example to show those groups of individuals who say "The NeXT?, but isn't it slow".

Color change interval specifies the interval within which the color should change. With Auto

on, the interval is a certain percentage of the view size.

For those B&W users out there, there is a pop-up button that specifies whether **RGB** colors or shades of **Gray** should be used. With **RandomColor** on, a completely random color or shade of gray is used each time, otherwise a source coded array of 14 colors is used whose order is fixed.

Given the randomness of the parameters, every so often you get a picture that you know just isn't going to turn out to be something spectacular or fill the whole screen. Press the **New One** button and get on with your life! Some of the results are so cool I wanted to do a screen grab, so I added a **Pause** button so that you would have all the time in the world to screen grab what you want.

Finally, if you want to show this off to your friends and/or want to use MartinView as a screen-saver and want spectacular results every time, the **Remember** button and **use file** switch button are for you. The **Remember** button appends all of the current parameter values and Function to a file in your home directory called ~/**.martinView**. Hit the Remember button while one of your favorite parameter sets is running, and when the **use file** switch is on, MartinView will cycle through all of the sets you have stored in the file ~/**.martinView**.
The numbers are stored in the file in a straightforward way and each set begins with the line

"MartinView" so it is easy to edit the file and remove any sets that are getting old.

I have included the file **".martinView"** containing some nice fractal parameter settings I have found. Just copy this file directly into your home directory and choose **use file** to see these settings. Besides storing complete sets in a file, MartinView also stores five of the options in the Defaults database for BackSpace.app. Yes, you could store all of the values, but I am not sure of the usefulness of storing them all. The following parameters are stored in the Defaults database:

Parameters stored in Defaults database:

The Function and whether it is randomly chosen or not (**MartinFunction**)
Maximum total points (**MartinMaxTot**)
Maximum in-range points (**MartinMaxIn**)
The color change interval (**MartinColorInt**)
Whether to use random color choices (**MartinRandomCol**)
Whether to run in Color or B&W (**MartinColorMode**)
Whether to use the file ~/.martinView (**MartinUseFile**)

Thanks to...

Ed Kubaitis, at UIUC for the X-Windows code on xmartin from which my port was made.

and of course, **Sam Streeper** for BackSpace and some useful suggestions for improving this module.

Any suggestions, comments or extremely interesting sets of parameters for MartinView would be greatly appreciated.

Jeffrey Adams
Email: jeffa@wri.com
NeXTmail accepted

MovieShow - A BackSpace module which shows TIFF movies

Notes from sam:

I hacked this up for BackSpace 3.0. I'm aware that the movie selection mechanism is pretty broken; you have to select a tiff file from within a .anim folder, and you may have to fight with the open panel to do this. This could probably be fixed pretty easily if I wasn't so sleepy.

original readme follows:

Hacked by Paul Burchard <burchard@math.utah.edu> from
Bill Carson's SlideShow module (see its README for more history).

NOTES:

This module accepts as a movie any folder containing at least one TIFF file (for example, *.anim movies). The TIFF frames will be displayed in the alphabetical order of their file names. Use the dwrite command to select a movie (as described below).

If you like the enclosed weather movie, you can make your own up-to-date weather movies using WAIStation.app (also available from the FTP archives). Get "weather.src" source from "directory-of-servers.src" by asking about "weather", then ask "weather.src" for "gif". When you

retrieve the GIF files, they'll open in ImageViewer, which lets you save them as TIFF files.

INSTALLATION:

1. Move "MovieShowView.o" into the Backspace application folder, and "Weather.anim" to your Images folder. E.g.:

```
mv MovieShowView.o /LocalApps/Backspace.app
mv Weather.anim ~/Library/Images/Weather.anim
```

You can recompile first if you wish, using the supplied Makefile.

2. Set prefs as desired using the dwrite command. (You need to restart the screensaver for any changes to take effect.) Here are the default values:

```
dwrite MovieShow Movie          ~/Library/Images/Weather.anim
```

```
dwrite MovieShow FrameTime     5
dwrite MovieShow BeginPause    12
dwrite MovieShow EndPause      30
dwrite MovieShow DarkTime      0

dwrite MovieShow Jump          YES
dwrite MovieShow SlideFrames NO
dwrite MovieShow SlidePauses NO

dwrite MovieShow TimeUnit      25
```

FrameTime = all movie frames held this many time units
BeginPause = extra time units to hold first frame
EndPause = extra time units to hold last frame
DarkTime = time units between movie runs, when screen is completely dark

Jump = do you want movie to start at random new location each time it runs?
SlideFrames = do you want movie to slide on screen as it is running?
SlidePauses = do you want picture to slide while movie pauses at begin/end?

TimeUnit = number of program loops to count as one time unit; don't make this
too big or sliding will be jerky

Multi:
no description

Neko:
dito

Planet:
Watch planets wander about space, dance, bump into each other, and explode...

Most of PlanetView was done by Kurt Werle (frsvnsvn!kurt@crash.cts.com). I thought his idea had promise, but it was kind of flickery and slow, so I reworked it into this. I changed it rather extensively, so you probably shouldn't blame Kurt if you don't like it. I removed his inspector kludge and put in a BackSpace 3.0 inspector panel.

I'm sure this thing could be much faster with the right performance tweaks to drawing. I might get around to this someday...

-sam (backspace@next.com)

Regular Polyhedra.

This is yet another BackSpace module. It is somewhat different from any other module I've seen though. It attempts to model the three-d behaviour of the regular polyhedra (tetrahedron, cube, octahedron, dodecahedron,
and icosahedron), in the following manner:

At each vertex of the polyhedron, place a unit mass. Between each pair of vertices string a massless, rigid (i.e. non-bending), velocity damped spring. Fill in a selected few of the faces, to make it look good. Put in a large room. Give a random push in a random direction. Make sure that the room doesn't burn in...

Now, this gave me some real headaches. Be careful about changing the parameters as they are set, especially the use of small masses, large spring constants, and small damping factors: because with any or all of these, the polyhedron can become **very** unstable - and very lacking in anything that could be termed beauty. (Unstable behaviour could also cause random features to manifest themselves; I think I've stopped the nastiest, but you never know). As set though, the polyhedra are fairly stable, and will not behave in an inappropriate fashion.

So, as usual, copy PolyhedronView.o into BackSpace.app, or use make install.

Possible bugs:

Not so much a bug, but could someone tell me what the fastest way to do the drawing I'm doing? It would seem to me that a user path would be slower than a wrap, since the path would never be re-used...

I suspect that the algorithm I have for drawing opaque faces in the appropriate order isn't quite the most optimal.

Despite my best efforts, the dodecahedron is unstable (it looses its natural shape after only a few collisions with the walls). I have suspicions about this one.

The compiler complains a bit. Don't worry. I think everything's ok. I'll fix this sometime.

Possible future extensions:

Notice that if a given solid polyhedron has sufficient symmetry (currently, each face must have the same number of vertices; each vertex in the same number of faces), it can also be handled by the simple expedient of adding the appropriate information to PolyhedronPartView.m (I got the information there mainly from Mathematica). (It had just occurred to me that this is a bit useless - I think that only the regular polyhedra have this property. So, then it becomes the more difficult task of handling polyhedron with different numbers of vertices per face, etc. Oh well).

This is strictly a black, grey and white production. When someone lends me a colour Station for a few days, I'll do something about it. There are some interesting tricks that one could play with the colour of an edge being related to its length. Amongst other things.

Certainly sometime in the future, you'll be able to select the polyhedron, and it's size, and characteristics.

Having more than one at once on the screen (and even bouncing off one another - or at least the 'solid' faces).

If you have suggestions, comments, or bugs, mail me. I have nothing better to do for a month or so.

Simon Marchant
simon@math.berkeley.edu

Swarm:

This is mostly a port of XSwarm (one of the greatest uses I'd found for X :-)
The gist is that there are a bunch of bees (initially 50) chasing one wasp.

The neat extras come into play if you move the cursor to the upper right corner of the window. When you do so, the Preferances panel comes on!

To install - just put the SwarmView.o and the SwarmPrefs.nib file in your BackSpace object directory.

Credit:
This 'program' was originally created by Jeff Butterworth (butterwo@cs.unc.edu) for use with X (if you look about you can probably find the source in the archives). I have liberally hacked his code, and stolen code from other backspace modules (especially LizardSaver).

Disclaimer:
This is my first backspace hack (and my first 'real' NeXT hack (**OH BOY!**)). Yeah, there's room for improvement... Go ahead (and send me some diffs :-)

'Bugs' :-)
This view only behaves well on a full window. I've probably neglected to lockview when getting the cursor coordinates or somesuch.

Comments:
BackSpace is real nifty! The only addition I want is Command-Shift-P to fire up a preferances window for the current 'View'.

Plea:
I love programming. Especially on the NeXT. Hire me. Pay me poor wages. Let me make you money. Make me programm on the NeXT. (buy me the Concept docs while you're at it :-)

Kurt Werle
frsvnsvn!kurt@crash.cts.com

[Regarding modifications by Scott Byer:]

The display code may look cryptic, due to the way it is trying to keep the data, but you can see that it is fast. Encoded user paths are the way to go if at all possible. I haven't spent too much time on this, so there is probably a lot of room for improvement in the math part. It would be interesting to change the math to use 16.16 fixed point numbers.

However, the display part is quite tight at this point.

Tricky things I found out when converting to BS3.0: Remembering to connect the id variable for the inspector panel back to the grouped box that contained the inspector was the hardest. Grouping the inspector pieces and getting it the right size was tricky. Now I just need a fancy icon or something for the backround. I'll play with a 3-D modeller and see what I can get.