Sinus

COLLABORATORS

	TITLE : Sinus	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 22, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sinus

1.1 Screenblankers for Garshneblanker

```
This document describes my screen blankers for the excellent screen
blanking utility Garshneblanker 3.6 (38.8).

 Installation   - How to install them
 The Blankers   - Description of each blanker
 Public Domain  - This is as PD as it can be
 The Author     - Address, Email, ...
 Credits        - People I want to thank

Warning! I was only able to test these blankers using my setup, an
A4000/30. If You use different machines, please send @{ " me " link "Author"} a  ↩
   mail
containing Your configuration (model, ram, graphic chipset/card)
and if the blankers worked with Your setup. Thanks.
```

1.2 installation

```
The archive AgBlankers has created the following directory structure:

AgBlankers/Blankers       contains the blanker executables and
        the files they need.
    .../MC68000     contains everything for the 68000 CPU
    .../MC68020     ... for the 68020
    .../MC68030     ... for the 68030
    .../MC68040     ... for the 68040
       .../Source               contains the blanker sources
       .../Doc                  contains documentation

To install the blankers: Copy everything in Blankers/<Your CPU> to
the drawer containing Your Garshneblanker blankers.

The "Sinus" and "IFS" blankers use floating point arithmetics. However,
FPU binaries are not included with my blankers. If You've got SAS/C 6.x
You should be able to compile the blankers without problems.
```

After You've installed the blankers You can use Garshneblanker's
Prefs window to test them. The default preferences work only with
AGA amigas. If You've got a non-AGA machine You have to change the
DISPLAY setting of every blanker.

Recommended DISPLAY settings:

```
  000,no AGA  0[234]0,no AGA  020, AGA  030, AGA  040, AGA
================================================================================ ←

Cubes   Hires, 16 Hires, 16 HiresNoFlck, 64 HiresNoFlck, 64 HiresNoFlck, 128
IFS HiresIntl, 8  HiresIntl, 8  HiresNoFlck, 8  HiresNoFlck, 8  HiresNoFlck, 8
Sinus HiresIntl, 8    HiresIntl, 16 HiresNoFlck, 32 HiresNoFlck, 64 HiresNoFlck,  ↩
    256
Trek  Hires, 16       Hires, 16 HiresNoFlck, 16 HiresNoFlck, 16 HiresNoFlck, 16
Water   Lores, 8        Lores, 16       HiresNoFlck, 8  HiresNoFlck, 16  ↩
    HiresNoFlck, 16
```

1.3 The new blankers

This package gives You the following new screen blankers (the links
surronded by stars are links to iff screenshots. This feature is only
available in Workbench 3.x. Look at the iff files "by hand" if You've
got 2.x).

```
  *  Sinus  - Draws nice  *figures*  based on a  sinus formula
  *  IFS    - Draws  iterated function systems  like  fire  for  xlock .
               Click  *here*  for some screenshots..
  *  Cubes  - Inspired by the Amigabasic mondrian demo included
               with WB1.x this draws @{ " *cube art* " link "cubes.iff/main" } to  ↩
                  the screen.
  *  Water  - Simulation of @{ " *Waterfalls* " link "water.iff/main" } and such.
  *  Trek   - Simulates importantly blinking @{ " *lamps* " link "trek.iff/main" } ↩
      like in Star Trek.
```

1.4 xlock

xlock is the most commonly used modular screen saving system for UNIX
machines using X11R5.

1.5 fire

Fire is a screen blanker for xlock which draws iterated function systems.
IMHO the best of its kind.

1.6 The SINUS Screenblanker

The Sinus blanker draws nice figures depending on a simple formula .
The appearance of these figures can be controlled by several different
parameters .

This screenblanker was created and implemented by Karlheinz Agsteiner .

1.7 The SINUS Screenblanker - background

The Sinus blanker draws its formulas based on the following sequence
(where needed, I use TeX-like notation)

Let a,b,c,d be values in { min, ..., max };

Then for some real z the functions

```
  x(z) = sin(a * z) + cos(b * z),
  y(z) = sin(c * z) + cos(d * z).
```

define a set of points in |R^2.

The actual plot is produced by an approximation of this set by drawing lines
from (x(z+delta^n),y(z+delta^n)) to (x(z+delta^{n+1}),y(z+delta^{n+1}))
for very small delta values. By setting delta to PI+delta we get more
complicated looking line pictures.

The color of each line depends on its length.

1.8 The parameters controlling sinus

The figures produced by Sinus can be controlled by several parameters
adjustable in the Prefs window:

  * SimplePics - Controls how many simple pictures (consisting of short
        lines instead of long ones giving a more "clean" look)
        should be drawn.
  * Range Param - The range for the four parameters in the computation
        formula. Roughly, a range of n allows the blanker to
        draw n^4 different figures. Larger ranges give more
        figures, but as a rule, the larger the range the more
        complicated/weird/ugly the figures will look.
  * Duration - How many time should Sinus spend with drawing one
        figure.
  * Delay - How long (in seconds) should Sinus wait after having
        completed a figure.
  * Display - Like in the other savers. The best results can be
        achieved by high resolution and many colors. Monochrome
        looks good, too (but monochrome, of course :-))

1.9 The IFS blanker

This screenblanker draws figures that contain random fractals called "iterated function systems". The basic functions drawn by IFS are roughly the same You probably know from the Scrawls blanker. However, IFS uses FFP numbers, and a probability distribution to minimize the probability of recalculating the same point over and over again. Therefore it looks much faster as Scrawls.

The coloring scheme used in IFS is inspired by the fire blanker in xlock which is IMHO the best iterated function systems blanker.

Several parameters control this blanker.

1.10 A probability distribution for area selection

When calculating an iterated function system each function roughly stands for a different area of the figure. If functions are selected with the same probability, the algorithm spends "too much" time on the smaller areas. This can make the figure look a bit more structured but usually it's just unnecessarily slow. "Special" computes the compression of each function and selects each function with a probability proportional to the area the function represents.

1.11 Parameters to manipulate IFS

The iterated function systems produced by IFS can be controlled by several parameters adjustable in the Prefs window:

* Functions - Sets the maximum number of random transformation functions that are used to compute the figures. The larger this number is the more sophisticated figures can be drawn. However, even 3 random functions provide a broad range of different figures while large systems with 8 or more functions often tend to result in clouds. I'd suggest a value of about 4 to 6.
* Iterations - Controls how long IFS computes one figure. Each iteration performs a computation sequence of about 100 points. Default is 300 iterations. Of course, large cloudlike structures are given a longer calculation time as small figures do.
* Area - Can be either special or uniformal. Uniformally drawn figures look a bit more structured in some cases but wastes much time. Generally, special is far better.
* Transpose - If set to "no", the center of each figure is set exactly in the middle of the picture. "Yes" will cause each IFS being displayed with some offset from this center.
* ContProb - The probability to paint the next ifs without clearing the screen first (in percent).

```
  * Decrease    - The number of percents substracted from ContProb
        after each drawing.
        Example: ContProb = 120%, Decrease = 20% will
        clear the screen after a drawing with the following
        probabilities:

          0  120% = always
          1  100% = always
          2   80% = 8 out of 10
          3   60% = 6 out of 10
          4   40% = 4 out of 10
          5   20% = 2 out of 10
          6    0% = never

        Thus the setting 120/20 means "draw at least two
        figures and at most 6".
  * Display     - Like the other blankers. IFS paints each figure
        in the same color so 8 colors should suffice for
        the most cases (except for very large ContProbs).
```

1.12 Basics of iterated function systems

Iterated function systems (IFS) are perhaps the most important kind of Fractals. The basic concept of IFS is self-similarity: A fractal is composed by parts which are similar to the whole fractal.

Technically, such a fractal is defined by a set of functions (where needed, I use TeX-like notation)

$$f_1, \ldots, f_n$$

Each f_i is the composition of a rotation, a scalation and a translation. Thus, every f_i transforms a set of points in $\mathbb{R}^2$ into a similar set of points. Starting by a certain set S_0 we can compute the union

$$S_{i+1} = \bigcup_{i=1}^{n} f_i(S_i)$$

The fractal defined by these functions is defined by

$$\lim_{n \to \infty} S_n$$

Example:

```
  f_1 = "scale by factor 2"
  f_2 = "scale by factor 2, translate by (0.5,0)"
  f_2 = "scale by factor 2, translate by (0,0.5)"
```

will give a the fractal shown @{ " *here* " link "IFSexample.iff/main"}.

1.13 cubes

The cubes blanker roughly draws the same modern art as the AmigaBasic mondrian demo contained in the distribution of Amiga-OS1.x (only much nicer :-)). This blanker doesn't need any parameters except the Display mode.

1.14 The Water blanker

The Water blanker simulates particles and gravitation. The outcome of this simulation is that it displays all kinds of waterfalls. This blanker looks especially good when viewed from >3 meters away.

* parameters
* technical details
* memory requirement

1.15 Parameters You can set, parameters You get

The Water blanker knows of lots of parameters modifying the behavior of each waterfall. Since the basic waterfall is quite a monotonous thing most of these parameters are selected randomly. Only two can be chosen by the user:

* Drops specifies how many drops should build each waterfall. The best value depends heavily on the machine used. On my A4000/30, 500-1000 drops look "fluidly" enough and give a good impression. Notice that every drop requires memory .
* Time specifies how many moves each drop should make until a new waterfall is created. I use about 500 here.
* Display: as usual. This blanker looks best with 8 or 16 colors. Fewer colors spoil the effect that the water gets brighter the longer it falls (looks a bit 3D to me), more colors make the blanker slower.

The following internal parameters that can't be modified by the user specify the look of the waterfall:

* Width: How wide is the waterfall (how far are the single drops spreading)
* Ground: Some waterfalls have a ground from which the drops reflect. This may not be physically too credible, but it looks good. :-)
* Sides: There are left->right, right->left and narrow->wide waterfalls
* initial direction: A waterfall can simply drop water from some height or blow it into the air first.

1.16 Technical Details on the Water blanker

The basic physical model for the water blanker is very simple.
Waterdrops have a certain initial point and speed and accelerate
downwards due to some gravitation. Every point moves independently,
no interaction takes place between them.

But even this simple model is too complicated for a blanker that
should look nice: Either we have to use hand coded fixed point
arithmetics or have a FPU, or we get incredibly slow. Even with
FPU, we're suboptimal. The way out of this mess is - of course -
precomputation.

Before a waterfall is drawn, a number of paths (adjustable at
compile time by the macro VARIATIONS (default: 32) of a certain
length (macro STAGES=128) are precomputed. Thus, each drop can
only move in 30 different ways. These paths are normalized so
that they correspond to a rectangle of at most

  (<width of screen> / 2) x (<height of screen> - 10)

Then, every point is characterized by an initial coordinate
in a 20*10 rectangle, the path it uses and the stage it is
at.

The blanker itself can now calculate each drop by a few
additions and array calculations. Using this method, we get
quite a good speed (about 8700 movements per second in
16 colors on my A4000/30).

Of course, due to all this precalculation Water requires a lot of
 memory .

1.17 Water's memory requirements

The Water blanker precomputes and stores much data in order to gain speed
when drawing the waterfall. Therefore Water is quite a memory intense blanker.
It needs:

Bytes	Size	Description
25600	VARIATIONS * STAGES * 8	All the paths
128	VARIATIONS * 4	additional data

==> 25728 Bytes allocated in any case plus 16 bytes for every point.

This means that a maximum of 4000*16+25728 = 89728 bytes can be
allocated. If any allocation fails the blanker will do nothing.

1.18 The Star Trek blanker

Have You ever wished Your computer could display important lamps

like the ones featured in the "Star Trek" series? Or the ones
that make transputer systems look so important? Now thanks to
the Trek blanker Your Amiga can look important, too!

Trek organizes the display into rectangles (lamps) and displays blinking
rectangles in them. Lamps can be organized in vertical or horizontal
rows, too. Every lamp blinks in a certain rhythm.
Several different blinking styles exist. To make everything look
even more important, the blanker adds 3D-boxes around every lamp
and seperates some lamps (rows of lamps) by horizontal or
vertical lines.

Of course, a blanker as important as this one can be controlled
by loads of parameters .

1.19 Parameters accepted by the Trek blanker

The Trek blanker can look quite differently depending on the following
parameters (modifiable via the Prefs window):

* Rhythms: Controls how many different blanking rhythms should be used in each display. If two lamps use the same rhythm they will go on and off at the same time. Thus, a small number of rhythms will look less chaotic as a large one. Don’t know which setting is optimally important. :)
* Duration: How many simulation cycles should the blanker spend with one set of lamps. Default: 500 cycles
* Delay: How long (in ticks) should the blanker pause after each simulation cycle (default: 3).
* Width, Height: Before actually drawing lamps the Trek blanker splits the screen into rectangles. When a rectangle gets thinner as Width or lower as Height, the blanker will use it as a lamp or a row of lamps. Small values give small lamps, big values big ones.
* Submode: Controls if a "basic rectangle" will contain one lamp or a row of lamps. Rows of lamps look more important, IMHO, so this is the default.
* Background: Controls the background color. Normally, one would use black here. Of course the blanker looks much better with a grey background because the 3D borders really don’t look very good on a black background.
* Free: Controls how many percent of the screen should be left free (ie. not covered by lamps).
* Display: As usual. The Trek blanker doesn’t look good with only two colors. If You use 4 colors You get the normal 3D-look but lamps are all colored in the same color (3 are used for the 3D-look). 16 Colors is best. High resolution is not necessary but recommended.

1.20 Public Domain

Everything in this distribution is PD. You can use the sources/binaries/
documentation in any way You like - run it, modify it, include it in
Your .signature, ...

The only exception to this rule is that if You release a modified version
of my blankers which doesn't add major changes I feel free to make fun
of You in the public (ie. UseNet).

Of course, PD means that I don't take any responsibility to any
harm any file in this distribution will cause to You or other
persons. This may sound like legal rubbish, but in the case of
screen blankers it's really important - I had some guys screen
saver crash while I was optimizing my hard disk. Hours of endless
fun formatting and installing my hard disk were the result.

One last word about shareware: IMHO it's absolutely ridiculous what kinds
of programs get released as shareware today. I've seen

  * memory games (the ones where You have to find two matching cards)
  * chess games which can't play chess at all (but have a vector graphics GUI)
  * othello games
  * WB icons
  * ...

being released as shareware. Most of the time the "reason" given for this
policy was "I've invested so much time there". Come on. The criterion for
releasing a program as shareware has to be the quality of this program.
Why should anybody be willing to pay eg. for a memory game?

1.21 Who wrote these great blankers? :-)

I like to hear from You! If You've got another Amiga model than I do
(I've got an A4000/30) please tell me if my blankers run on Your
configuration. If it doesn't work, it would be nice if You provided
additional information (Your setup, what's happening).

Send bug reports etc. to:

   karlheinz.agsteiner@informatik.tu-chemnitz.de

My WWW page is located at

   http://www.tu-chemnitz.de/~kas/agsteiner.html

or (snail mail):
Karlheinz Agsteiner
Lessingstr. 10
09130 Chemnitz
Germany

1.22 credits

Thanks go to

* Michael D. Bayne for Garshneblanker, the best screen saving system I've seen so far - being able to write blankers without having to care for a Gadtools interface is great!
* SAS for their compiler, especially Doug Walker and Jim Cooper for being present on comp.sys.amiga.programmer without getting tired of answering the same questions over and over again. Too bad Amiga SAS/C is no longer supported.
* The Free Software Foundation for creating THE ONE EDITOR.
* Nicola Salmoria and Philip A. Vedovatti for NewIcons - the best Icon Package around (the icons included in this distribution are drawn by them and will look a bit awkward if NewIcons is not installed.