DRAFT - Simulating Video Feedback

Jason Rampe jasonrampe@softology.com.au
16th July 2007

This paper is still a work in progress and is presented as a general overview of my attempts to simulate video feedback.

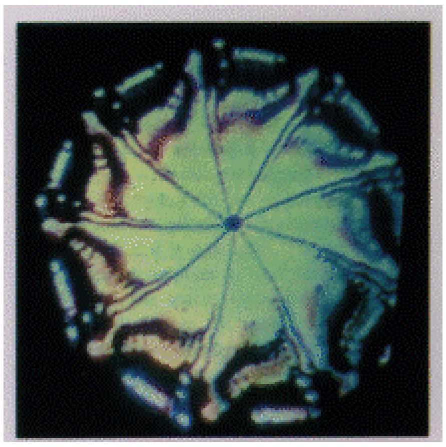
Abstract

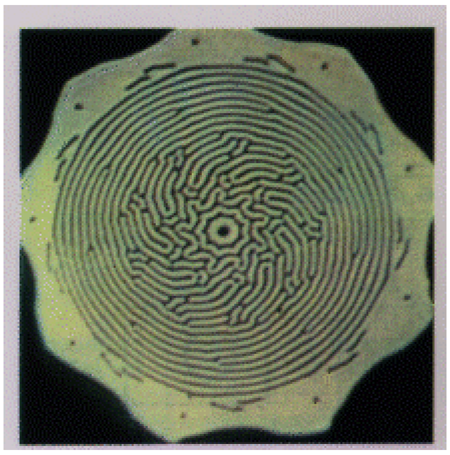
This paper describes the phenomena of video feedback and introduces new methods on how to attempt to simulate the complexities of real video feedback in software.

Video feedback explained

Video feedback involves pointing a camera at a screen that is displaying the output from the camera. This process results in an endless looping of the signal that can create interesting patterns. Video feedback provides a readily available experimental system to study complex spatial and temporal dynamics [1] and non-linear dynamics. This paper describes methods to simulate video feedback in software.

James Crutchfield created the following two images of real video feedback for his 1984 paper [1].





Problems with physical video feedback

When experimenting with real video feedback and trying to get impressive results there are a few issues to deal with.

1. Changing the TV and camera controls needs to done slowly and in small increments. Otherwise the display will drastically change and a potentially interesting set up can be lost.
2. It is not possible to reproduce the results of a video feedback run.

The simplest example of a real video feedback set up would have the following controls;

1. TV with brightness and color controls.
2. Camera with zoom, position and rotation.

This scenario has a nine dimensional space (TV brightness, TV contrast, Camera zoom, Camera X position, Camera Y position, Camera Z position, Camera X rotation, Camera Y rotation and Camera Z rotation) to find pleasing results within. If you consider the variations of the nine variables then there is an almost an infinite combination of settings. There is only a small subset of all the possible settings that will lead to a visually pleasing feedback display.

Video feedback simulation

Rather than relying on mathematically based formulas to simulate feedback, the methods described in this paper are based on a series of image manipulation routines.

The results from my attempts at simulating video feedback have been promising so far and do result in a system that does show the non-linear dynamics of feedback can be simulated.

Basic overview of the simulation process

Start with a bitmap that represents the screen the camera is pointing at. Apply a series of bitmap convolution and manipulation routines to simulate screen and camera controls. The bitmap is then zoomed and rotated through an array of further bitmaps to simulate the repeating images within the feedback loop. All of the bitmap layers are then merged back into the initial bitmap and the process repeats.

This process is similar to an advanced cellular automaton that has many more variables involved when calculating the next cell value.

Complexity of simulation

Attempting to simulate the basics of a camera aimed at a TV did not achieve video feedback like results. The resulting simulations were boring or died out too quickly. It was found that as many variables and controls as possible are required to have results that are visually similar to real video feedback.

Simulation Controls

When running a video feedback simulation it is crucial to have control of the virtual screen and camera controls to adjust the image. Having control parameters that can be adjusted during the simulation process helps finding results matching real video feedback. It was also found that having scrollbars to modify the settings (ie changing rotation within the range of 0 to 360 degrees) quickly helped to get into the ballpark of a good result, but it was required to have further buttons to +/- the values within smaller amounts, even as smaller as +/- 0.001 on some settings makes a difference in fine tuning the settings.

Here are the simulation controls and how they affect the results;

Screen Controls

Brightness – Simulates a TV brightness control by decreasing or increasing the RGB pixel intensities of the bitmap as required.
Color – Simulates a TV color control to decrease the color intensity of the image. This is done by converting the image RGB values to YUV and then decreasing the U and V values to reduce color within the image, but still retain the same monochromatic intensity within the image.
Contrast – Adjusts the contrast of the image on screen.
Noise – Injects randomness into the image each frame. The simulation adds random variations to the RGB components of the bitmap. For grey noise the RGB values are all adjusted by the same amount. For color noise the RGB components are randomly adjusted to allow shifts in color. This helps to introduce color into simulations that would normally have monochromatic results.
Interlacing – On a physical TV screen, every displayed frame is combined from two separate fields. This is simulated by combining two simulation steps into a single bitmap using alternating even and odd lines.
Feedback layers – Controls the number of bitmaps each frame is zoomed and rotated through. It is impossible to simulate the real world infinite levels of feedback, so a number of layers are chosen. More layers give a more realistic result, but slow the simulation down.

Camera Controls

Rotation – The amount the camera is rotated around its optical axis. Rotating the on screen image through the feedback layer bitmaps simulates this.
Zoom – Allows the virtual camera to zoom in and out towards the virtual screen. The simulation uses basic bitmap resizing algorithms to resize the layer bitmaps to simulate this effect. There are also options for the zoom resampling algorithm to use based on well-known examples (Box, Bell, Hermite, etc).
Blur – Simulates a camera's focus control. The simulation does this by gaussian blurring the on screen image.
Sharpen – Applies a sharpening algorithm to the image. This is done by blurring the current image (using a 3x3 non-weighted convolution blur rather than Gaussian) and then subtracting the blurred image from the current frame. Using a different algorithm for the sharpening allows it to be used in conjunction with a blur and not result in the two cancelling each other out.
Invert – Inverts the image the camera records. Simulated by inverting the image each frame (ie the RGB components are changed to 255-R, 255-G and 255-B)

Mouse Injections

Allows a bitmap to be inserted into the simulation while running by clicking and dragging the mouse over the virtual screen. Without any additional injections into the simulations, most will die out to a blank screen.

Bitmap name – Allows any bitmap file to be injected into the running simulation. Small sized bitmaps should be used to not overpower the existing screen image.
Auto-injection – Automatically injects the selected bitmap into the simulation every x frames. Injected images are centered on screen.
X and Y offset – How much off center are the injected images injected. When the zoom is set to <1 a largely offset image will spiral inwards to the screen origin.

Other Controls

Order of effects – Determines the order that the image processing effects are applied each frame. Changing the order gives more control over the simulation.

Image Processing

These settings do not have any real world comparison but they can lead to interesting feedback results.

Mirror image – Simulates putting a mirror between the camera and screen. Options for multiple mirrors and angle of the mirrors help to simulate the more traditional fractal looking video feedback real world examples.
Seamless tile – No real world comparison. Wraps the image around itself at the edges.
Warp image – Distorts each frame using pinch, punch and other common image processing methods.

<u>Interesting results from the current simulation</u>

Without requiring any cellular automata or reaction-diffusion code within the simulation loops the results do show features that do look very much like both CA and RD results.

Once a relatively stable state is found then the auto-injected image can be turned off and the simulation will continue to run on its own, even after changing the parameters like rotation, brightness, contrast and zoom levels. Depending on the settings a stable pulsating simulation run can continue almost indefinitely without further operator influence.

Contrast makes a big difference. Without contrast, the images tend to blur out as repeated blurring will diffuse the image pixel values until they darken to black.

Sharpening also makes a huge difference. Especially blurring the image and then sharpening (due to the different algorithms used when blurring and sharpening the two do not cancel each other out). This led to results that had the "fingering/striping" images that real video feedback shows.

Simulated interlacing. To simulate interlacing every two frames are saved and then using the odd lines of the first frame and the even line of the second frame creates every second frame. This is similar to every one frame of a TV image being made up of two fields (odd and even lines combined). This leads to the problem of the resulting simulation running twice as fast because two frames are combined into each single frame. A better option would be to combine each frame with the previous frame.

Injecting noise. Very important. Each pixel RGB value is altered slightly (by a configurable percentage) to simulate noise fluctuations in the simulation. Values between 10% and 30% are enough in most cases. Also the random seed for the pseudo-random generator function should be set to the same value before each simulation run to ensure future runs display the same output.

Introducing a frame of static. When simulating video feedback injecting a frame of static can really help keep the simulation alive. This can also be compared to in real world video feedback when you quickly turn on and off a light in the room to get a dead feedback running again.

Ideas for further simulation improvements

Genetic mutation. Rather than endlessly randomly changing parameters and hoping to find a pleasing result genetic mutation can be used to mutate an interesting example into a better result. This allows a series of video feedback simulation examples to be slightly tweaked until the perfect result is reached. When mutating a source feedback each parameter is changed by a small amount to tweak the settings slightly but not change the result too drastically. The mutation amount can be configured from a tiny percentage to a large amount.

Another option is to perform a smart search for possible interesting results. Poor settings lead to a blacked out result (ie after a number of repetitions the display dies to a black result) or a fully intense static (ie after a number of repetitions the virtual TV results in a screen full of static). The ideal result of a visually pleasing simulation should be between a black out and a white out. From experiments the result of a potentially pleasing result should be between 30% and 70% overall image brightness (ie total the entire image pixel brightness). For the results within the OK range they can be automatically saved for future checking. This also helps reduce the enormous search space that manual tweaking would need to find appropriate simulation parameters.

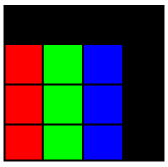
Cellular automata and reaction-diffusion algorithms. During each frame the pixel values could be processed with a CA or RD algorithm for a specified number of steps. Even though the current results do show CA and RD like results using CA and RD algorithms on the frames could lead to more interesting results.

Automated adjustments. Algorithms to know when to increase/decrease brightness or other settings. This could keep a simulation running longer without user intervention. Initial tests based on automatically modifying brightness alone did not help and the simulation flashed back and forth between too dark and too bright. Automatic adjustments need more settings included. Possibly training a neural network to adjust settings would be beneficial.

Better control of camera rotations on all axis. At this stage the simulation software does not allow the camera to be pointed towards the screen at an angle. One idea was to project each frame onto an OpenGL quad that is rotated in 3D space to simulate the various angles the camera could be pointed at the screen. The conversion of the screen image to 3D and back to 2D destroyed the image due to the OpenGL texturing losing detail. A preferred method may be to use a virtual 3D affine or projective transformation routine to allow better control over the way each frame is resampled.

Ability to change settings automatically as simulation runs. This results in smoother changes to simulation parameters compared to large jumps in settings when manually adjusting settings. So far automatically changing the camera angle and angles of the mirrors shows interesting results.

Oversampling. Each frame pixel is expanded 4 x 4 times as much to simulate a real TV screen RGB pixels. Each of the source pixel RGB values are striped down rows of 3 pixels by disregarding the other color components. Eg the red pixel column value is calculated by taking the R value from the pixel RGB value while setting the G and B values to 0.

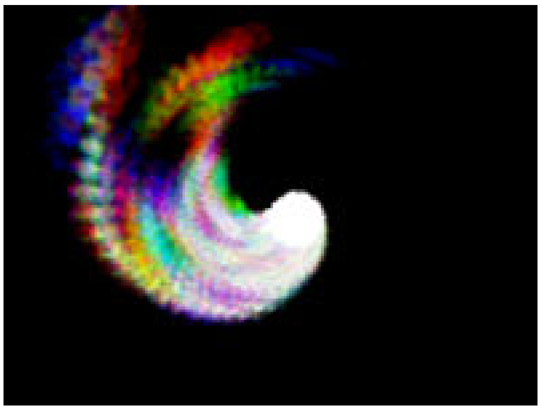


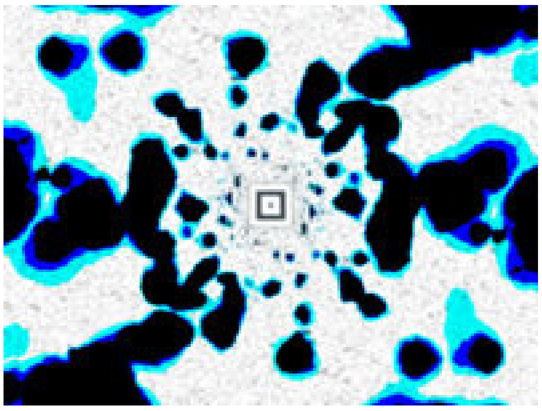
The over sampled image is then fed through the image processing routines and then resized to the current simulation display size. The individual RGB pixels could be leaked into neighbouring cells to simulate the way a real TV pixel values spread out. Initial tests of over sampling show that including the black pixels to simulate raster lines results in the downsized image being too dark. Even when expanding each pixel by 3x3 still leads to the downsized image being too dark. This could be enhanced by adding a constant to the other values (ie R remains the same, but G and B are set to 128). The downside to this is that it results in a frame that has increased brightness so when the frame is reduced to the non expanded image the frame is too dark.

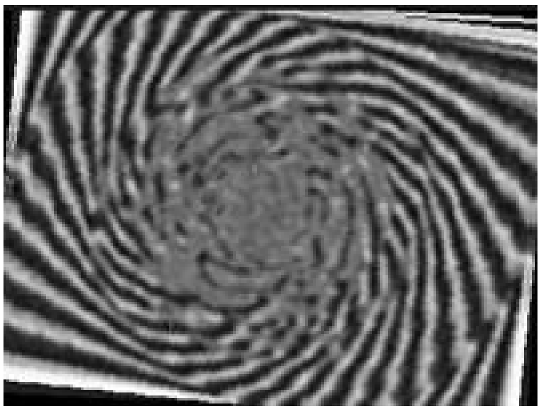
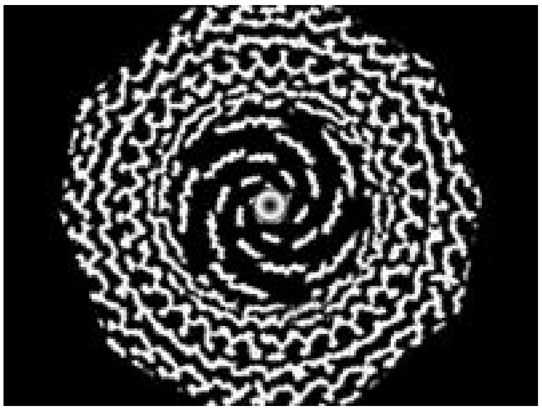
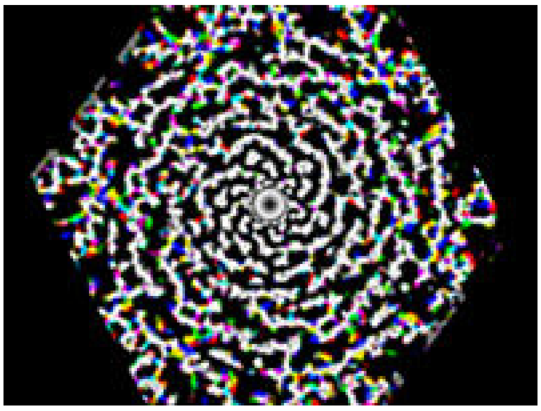
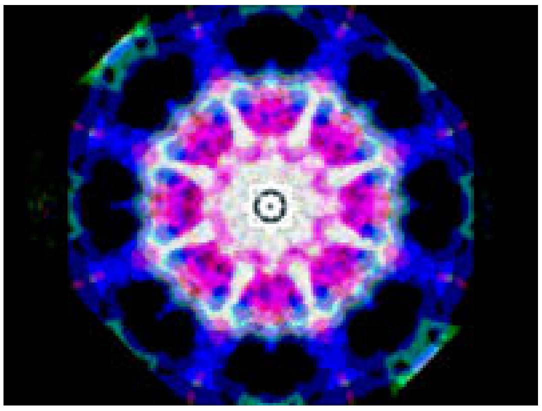
Music based on the image results. Initial tests generating music based on the overall brightness of each frame resulted in boring results. Future music attempts could be based on other values from the simulation.

<u>Example results from current simulation</u>

The following images are example screenshots taken from Visions Of Chaos during feedback simulations in progress.





Conclusion

Simulating video feedback is a possibility. The more variables that are introduced into the simulation result in a more realistic result, but also extend the possible multi-dimensional parameter space to find the pleasing results within. Many of the current results are very accurate to their real world counterparts. There are more enhancements that can be made in the future to the simulation to achieve other realistic results.

Simulation availability

The feedback simulation software is now included as part of the Windows software Visions Of Chaos available at http://softology.com.au/voc.htm.

See http://softology.com.au/videofeedback/videofeedback.htm for further video feedback related information.

Thanks

Thanks must go to Michael Rampe whose videos of real video feedback inspired me to start attempting simulation of the process in software. During the many revisions and rewrites of the simulation software he has always had invaluable ideas that have helped the software evolve to its current level.

References

[1] J. P. Crutchfield: Space-time dynamics in video feedback, Physica 10 D (1984).