Summary and Implementation of "A Novel Image Interpolation Technique Based on Fractal Theory"

Joshua Scoggins

November 27, 2011

Contents

What is a Fractal

A Fractal is an infinite image that is generated by something called an interable function system (IFS). Which is a set of affine transformations that are applied to a set of values to generate a new set of values. This function system can be continually applied to these results.

Affine Transformations

An Affine transformation is an operation that defines the rotation, scaling, and translation of a given point. The equation looks like this:

$$k(x, y) = (ax + by + e, cx + dy + f)$$

The Hutchinson Operator

We take a set of affine transformations and apply them to an image using the following function

$$k(C) = \bigcup_{n=1}^{N} k_n(C)$$

Fractal Interpolation

We can perform interpolation using the following formulas:

$$x_{\text{added}} = \frac{(x_i + x_{i+1})}{2} + s * w * \text{rand}(s_0)$$

$$y_{\text{added}} = \frac{(y_i + y_{i+1})}{2} + s * w * \text{rand}(s_1)$$

Fractal Dimension

The fractal dimension is how much space a fractal will consume. Ideally one should take the derivative of the function used to describe the image. However, this is really hard to do so an approximation is computed instead.
If we can get a good fractal dimension then the interpolated points become

$$x_{\text{added}} = \frac{(x_i + x_{i+1})}{2} + f * \text{rand}(s_0)$$

$$y_{\text{added}} = \frac{(y_i + y_{i+1})}{2} + f * \text{rand}(s_1)$$

Approximating the Fractal Dimension: Box Counting

Instead of finding the exact curvature we can find the amount of area that the curve will take up by overlaying them with boxes. We use the following formula to denote this

$$D = \frac{\log(N_r)}{\log(\frac{1}{r})}$$

Where r is the scaling factor and N_r is the number of non-overlapping scaled copies of the fractal curve.

Random Midpoint Displacement Method or the Plasma Fractal

We can't just distribute the pixels of the original image throughout the new image. We need to apply something called random midpoint displacement to properly place pixels in the resultant image. It works by assigning values to the corner of a rectangular area of the result image and continually subdividing those points until the distance between two points is less than a single pixel. Once that happens take the average of the four corners and combine it with the displacement to get the intensity for that pixel. Rinse and Repeat.

Paper Summary

A summary of "A Novel Image Interpolation Technique Based on Fractal Theory"

The K Fractal Dimension

The paper defines a fractal dimension called the k-dimension. It is defined as:

$$k = \frac{\sum_{j=0}^{n-1}\sum_{i=0}^{n-2}(|G_{(i+1,j)} - G_{(i,j)}|) + \sum_{i=0}^{n-1}\sum_{j=0}^{n-2}(|G_{(i,j+1)} - G_{(i,j)}|)}{2 * 255^n}$$

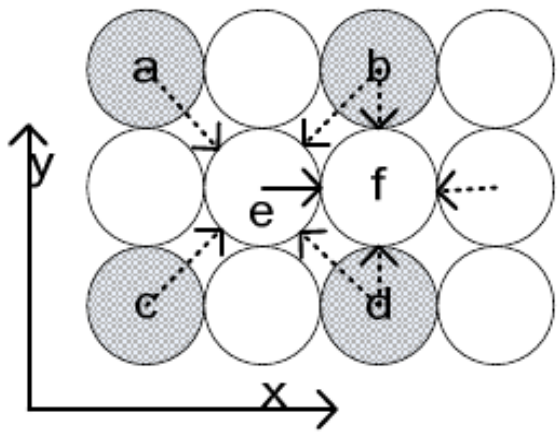
Where n is the size of the overlay mask, G represents pixels in the original image. When $k = 1.0$ then the image is very rough and when $k = 0.0$ the image has no texture.

Decomposing this fractal dimension

This paper was pretty badly written and the most evident place is in explaining why this k-dimension function works. It seems that the k-dimesion works because it is trying to find the amount of "roughness" in a given image section. When $k = 0.0$ the image section is full white, when $k = 1.0$ the image is really rough.

Subdividing the Image using Random Midpoint Displacement on 4x4 blocks

The paper calls denotes that applying Random Midpoint Displacement to 4x4 blocks will ensure image quality and sane running times.
This means that two types of pixels must be interpolated. I have denoted them E and F type pixels based off the following image



Computing E type pixels

The formula to compute E pixel intensity is:

$$G(x,y) = \frac{1}{4}\{G(x-1,y-1)+G(x+1,y)+G(x-1,y+1)+G(x+1,y+1)\}$$

$$+\sqrt{1-2^{2k-2}} * ||\Delta X|| * \text{Gauss} * \sigma$$

Where $||\Delta X||$ is the distance between pixels, *Gauss* is a gaussian random variable, and sigma is the pixel variance.

Computing F type pixels

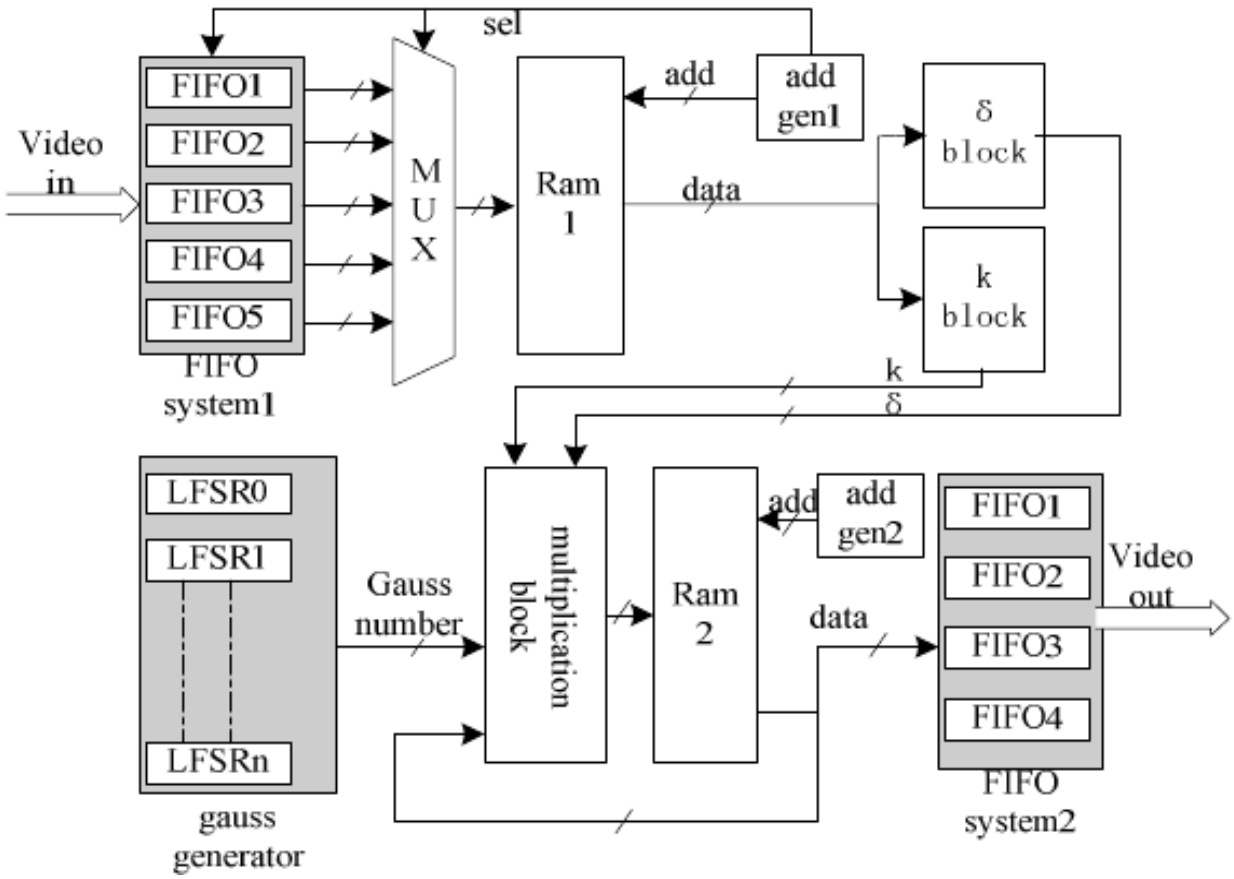
The formula to compute F pixel intensity is:

$$G(x,y) = \frac{1}{4}\{G(x,y-1) + G(x-1,y) + G(x+1,y) + G(x,y+1)\}$$

$$+2^{-\frac{k}{2}} * \sqrt{1-2^{2k-2}} * ||\Delta X|| * \text{Gauss} * \sigma$$

A Hardware Block Diagram

This paper gives a hardware block diagram for the above fractal interpolation algorithm that looks like this



Implementation Details and Results

- Environment Details
- Gaussian Random Variables
- Operating on 2x2 instead of 4x4 blocks
- Post Implementation Work
- True Plugin Architecture

Implementation Details

- Written in C#
- Works in Linux, Windows, and Mac OS X
- Crashes like a champ in Wmii!
- Target platform is a Lenovo T61p
  - 2.5 Ghz Core 2 Duo T9500
  - 8 Gigabytes DDR2 800
  - 500 Gigabyte 5400RPM WD AV-25 with 32-mb cache
  - Nvidia Quadro FX 570M
  - Arch Linux amd64 running wmii
- Tested on Mono 2.10.6 and .NET 4.0
- Found an infinite loop bug with the floor function

Gaussian Random Variables

This was one of the more random, heh, aspect of implementing this paper. It turns out that the following code is a suitable substitute to writing a gaussian random number generator.

```
Random r = new Random(Guid.NewGuid().ToString().GetHashCode())
```

Improvements over the paper

I had to make several trade offs to get a working implementation that I believe increased quality:

- The paper operated on 4x4 blocks. I kept on having crashes so I used 2x2 blocks instead.
- I didn't use f-pixels because the expansion process of 2x2 blocks doesn't need them...

A Wild Image Appears!

Image Comparison: Fractal Versus Nearest Neighbor (720x480 to 4096x3072)

Image Comparison: Fractal Versus Nearest Neighbor (720x480 to 8192x6144)

Post Implementation Work

Implementation is really slow! How can we speed this up?

- Reimplement using 4x4 blocks....Not sure how
- Thread the code (Precomputed k values ahead of expander)
- Only compute values when source pixel isn't already assigned.

Note: I found this after I wrote the paper and turned it in.

Comparison Between Correct and Nearly correct code

Generation	Image Size	Desired Size	Running Time
First	160x160	8192x8192	28 Minutes
Second	160x160	8192x8192	14 Seconds
First	1024x768	8192x6144	6 Minutes
Second	1024x768	8192x6144	9 Seconds

Demo

Conclusion

Fractal Interpolation using the K-Dimension is actually really viable. I wish the paper was a little more specific on how the displacement was acquired