Analysis of the Monte Carlo Method of Image Generation

Chris Rackauckas
Oberlin College

April 2, 2012

Abstract

For my project I decided to invent a Monte Carlo method of image generation. It reads in the image and uses the color values to generate a probability array. Using a psudorandom number generate with this matrix, we create an image that is reminicent of the image we generated the probability array from. An interesting detail about the non-commutativity of this process to the black and white conversion is explored.

1 Introduction

For my project I decided I wanted to in some way develop a program that could generate pointilist-like images. Remembering how Monte Carlo methods of using random points can be used to estimate the area of a circle, I decided to try to use this method of putting random points to generate the image I wanted (sounds pointilist to me!). Thus I created a program that follows these basic steps. First it reads in an image and generates a pixel array. In order to do this in a simple manner, I used the Processing library on Java so that the code for this was one line. From there I created a probability matrix that gave a value from 0 to 1 according to what the brightness of the pixel was (for each color). Thus at pixel i I would have that the probability matrix P[] would have a value at i be (P_i). Thus I set the $P[i] = b$ where b is the brightness of pixel i. Thus the result is that that the $P[]$ is thus a

matrix of values whose probabilities at a given i is equal to the brightness at a given i.

With this matrix I then generate the picture by taking random numbers. A random number is generated as in the uniform distribution $[0, n]$ where n is the number of pixels in the image. I use the random number by partitioning the number into segements using the integer value and taking the decimal value as the random number. Thus for the number $X = x_1.x_2x_3...$, I would set the value $y = x_2x_3...$ to be the random number in the interval $[x_[1]-1, x_1]$ and compare that to $P[x_1]$. If $P[x_1] > y$, then I would increment the brightness at pixel x_i in my drawing photo, else I would do nothing.

This may be easier to understand with an example. Say we have a 10 pixel picture and the random number I receive is 4.36. I would then look at $P[4] > .36$ and if this is true then I would increment the brightness of pixel 4 in my drawing photo, else I would not do a change.

Proof of Correctness It may be simple to see that this should draw the picture when using a large amount of random numbers and small increment values. This is because the probability of incrementing the brightness at any given pixel is directly related to the brightness of the original image's pixel and thus we would expect the distribution of the brightness changes to match the distribution of the original photo and thus draw the photo. We can also prove this more mathematically. Take a random number X from a uniform distribution [0,n]. Let b be the brightness of pixel i. Notice that the probability $X \in [i-1, i-1+b]$ is simply the integral from $i-1$ to $i-1+b$ of the probability density function of X which is $\frac{b}{n}$. Thus given n random numbers, the probability of incrementing i is b. Since this is true for any i after n random numbers the probability of incrementing the brightness of any point is simply the brightness of the point in the original picture.

This shows us that the distribution will be the same as in the original photo. However, to receive the original photo we must also control the brightness. Notice that if we take almost an infinitely many random numbers for an n pixel Mona Lisa and increment the brightness in any increment case by some finite number we will receive a white picture! Thus we must control the end brightness of the photo by setting the number of random numbers that will be chosen. To do so, assume that we want the expected brightness of the photo to be equal to the brightness of the original photo. Assume that you increment the brightness each time by the increment value k. Thus

the expected brightness of the photo is the expected number of increments times the increment value. Notice that the expected number of increments is equal to the probability of incrementing times the number of random numbers taken. The probability of incrementing is simply the average of the probability matrix $P[]$. Denote this average value as a If we take the number of random numbers to be m, the expected number of increments is am. Thus the expected change in brightness is simply kam. Given that the brightness of the original picture, B, would be a known value, a would be a known value, and k would be a parameter we can set, we can solve for $m = \frac{B}{ka}$. Thus if we take m random numbers we will have the expected brightness of the drawn photo equal to the brightness of the original photo.

With the expected brightness being the brightness of the original photo and the distribution of the increments being equal to the distribution of the brightness of the original photo, we can conclude that this process will indeed generate an image like the original photo

2 Non-Communitivity With Black And White Conversions

Knowing that this process will generate the image, I decided to make a program that would use this processs to generate images. One startling thing that I recognized was that there was two different ways that we could implement color. One way is that we could take a random number and then look at the probablility matricies for red, blue, and green all using this random number and run this process, or grab a different random number for each color. Notice that we can see that the expected brightness has no difference! However, the distribution of the points drawn will change since the colors would be correllated. This is easy to see since in the first case a high random number would give a good chance of placing a red, blue, and a green at a given point where as it would only effect one color in the second case. Thus we would expect the brightness to be more "clumped" in the first case then the second case. Thus the contrast of the photo generated using the first method will be different than the contrast of the photo generated using the second method.

This may seem trivial since obviously there is going to be a correlation of the brightness's when we use the same random number for multiple cases,

but this gives a more startling result. Notice that if we do a black and white conversion and then create the probability matrix, we correlate the color values in a mannor like the first case. Thus if we do a black and white conversion and then draw Monte Carlo, we would expect a higher contrast than if we were to draw Monte Carlo and convert that photo to black and white! Thus we see that doing a black and white conversion is not commutative with the Monte Carlo drawing process.

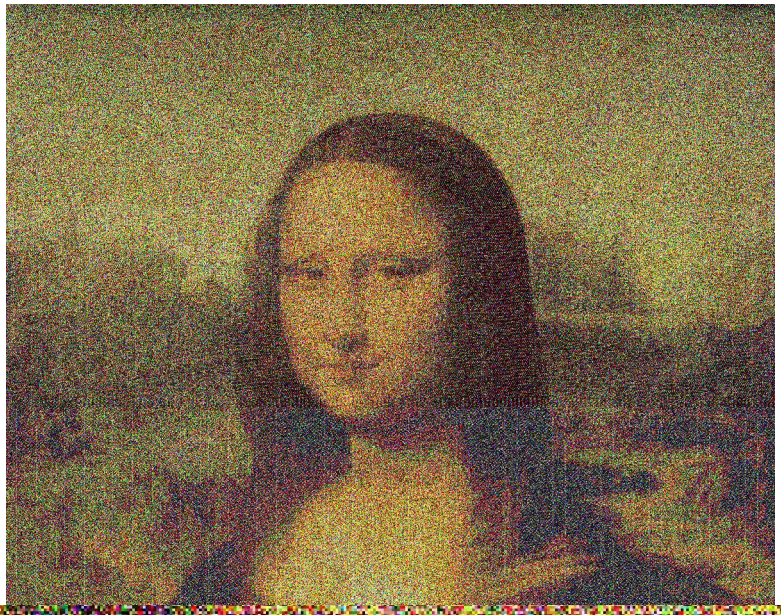
To give more concrete results, I created a method that would run a statistical test to see if we could reject the null hypothesis that the mean values of the contrasts were the same. Not surprisingly, we were able to reject the null hypothesis. Thus in both theory and practice we have shown this holds,

3 Results

Due to the fact that the point class of Processing is currently broken, this cannot be currently extended to doing point drawings instead of pixel increments. However, this can be extended to elipse drawing. A way to do this would be to create a probability matrix that takes a square of 9 pixels as the probablility value and instead of incrementing we would draw a random elipse that fits within those pixels that has some alpha value. This in trun should draw the photo using a Monte Carlo method that would look more pointilistic than this version.

4 Conclusions

After successfully making a program that implements this technique and testing its properties, I have confirmed that the theoretical analysis is indeed true. Now it's picture making time!



Probabilistically
Generated
Mona Lisa
With 9 Colors