Fall 2017

Voronoi Error Optimization via Iterative and Global Error Reduction

Bryan Hoyle *CS 633 Computational Geometry*

Abstract

Voronoi diagrams are often used for various artistic applications. These include things like stippling[4] or tiling[2]. This paper introduces and compares two novel uses of voronoi for aesthetics: iterative and global error reduction. These processes attempts to minimize perceptual error given fixed count voronoi decompositions with one color per cell. This potentially has applications beyond just aesthetics: since one can easily compress a diagram by storing the positions using delta encodings and a single color per region, it could be potentially useful for low-bandwidth placeholder images or other places where space is a concern and artistic approximations are acceptable.

1 Motivation and Introduction

The code for this project can be found at
https://github.com/triclops200/voronoi-pictures

The motivation for this project comes from a dream in which the author saw an image that was similar to a voronoi decomposition such that areas with high detail had more voronoi regions. When the author awoke, he implemented the iterative version of the algorithm and got a decomposed image. This image, which looks similar to the dream image, is shown in figure 1.

Figure 1: Image similar to what was seen in the author's dream. The iterative technique was used to create this.

This image was created using an iterative error reduction technique. Regions of high perceptual error are specifically targeted to add more cells and this process iterates until there are no more

cells to distribute. This is a local optimization, however, and therefore the topic of this project was to attempt to use global optimization to improve the final results.

2 Methods Used and Studied

There were two methods used for this comparison, an iterative error reduction approach and a global optimization approach.

2.1 Iterative error reduction

The iterative error reduction algorithm is described in algorithm 1.

Data: img, iterationMax, $K \leftarrow$ cells per iteration
Result: A voronoi decomposition
$0 \rightarrow$ n;
randomVoronoiDiagram(numCells=K) $\rightarrow$ D;
while $n <$ *iterationMax* **do**
  selectHighestErrorCell(img,D) $\rightarrow$ cell;
  splitCell(cell, numCells=K) $\cup$ D $\rightarrow$ D;
  recomputeVoronoiDiagram(D);
end
writeFinalImage(img, D);

Algorithm 1: Iterative Voronoi Error Reduction. Cells are colored based on average image color within the region. Error is CIELab distance. splitCell returns K points positioned within the cell passed in. A typical K is five.

This algorithm can be computed efficiently by taking advantage of two tricks. The first is that, since none of the other points move, any pixel will only be closest to its current parent or one of the K newest points. Secondly, due to the locus property of voronoi[1], we can strictly bound the region to update based on the dimensions of the currently selected cell. Thus, the iteration time gets faster over time as the regions become smaller, and the bulk of the time in computing any decomposition is in the first few percent of the iterations.

2.2 Global error reduction

The global error reduction used a standard genetic algorithm with tournament select[3]. An individual consisted of a voronoi diagram, and was scored by overall perceptual error between the diagram and the underlying image. Crossover consisted of trading voronoi points between two diagrams. Several different individual mutation strategies were tried: completely randomizing an individual, randomizing points, adding Gaussian noise to points, and using a hybrid iterative approach where random cells would be removed and the iterative algorithm would be run to add the same number of cells back.

The actual implementation of the fitness function was the slow part: it has to compute the full voronoi diagram and the cell each pixel belongs to, compute the average color of the cell, and,

finally, compute the cumulative error of all of the pixels. Some of these tasks were parallelizable, such as the construction of the k-d tree, iterating over the pixels for steps 1 and 3, and iterating over the cells for step 2. Also, since the fitnesses of each individual were independent, these could be calculated in parallel as well. The library lparallel was used, which allows for higher-level parallelization primitives and functions using greenlets multiplexed over threads to allow for less time spent context switching.

3 Results/Findings

The iterative method ran much faster. In one example, it took about 17 seconds to run 1000 iterations with 5 cells added per iteration and achieved a global error that was 92% (thus, better) of the error of the genetic algorithm version which took 13 hours to run utilizing 1500% CPU (or 75% of the 20 cores it was given) and close to 80GB of ram. This kind of difference in performance both speed-wise and error-wise was consistent across images. However, the visual results are interesting to compare against. Figures 2 and 3 show the results of the aforementioned runs. Note how

Figure 2: Voronoi image decomposition using the iterative error reduction approach

the global error technique spread its cells out more evenly across the image, while the iterative technique focused on regions of high contrast, like branches. The global technique also seemed to line up cells in such a way that the edges of the cells would correspond to edges in the image. Thus, while the iterative image scores better overall, the second image subjectively looks better. This

Figure 3: Voronoi image decomposition using the global error reduction approach

suggests two things: first, the summed pixel difference in CIELab space is not a perfect indicator of visual similarity, and secondly, it is possible that humans[1] prefer error distributed more evenly across images than have regions of high and low error.

The differences between the iterative and global optimization techniques mean that the techniques are better suited for various types of images. The iterative technique performs subjectively better in images where regions of high contrast are the most important, such as images like fireworks (shown in figure 4) or starfields (shown in figure 5). This makes sense, as most of the image is similarly colored, thus the iterative technique uses its cells in regions that are more visually important, leaving large sections mostly empty of control points. However, at least subjectively, images with more or less distributed interesting regions, the genetic algorithm tends to do better.

[1] At least the author

Figure 4: Firework image decomposed using the two methods. Iterative is the left image, genetic is the right

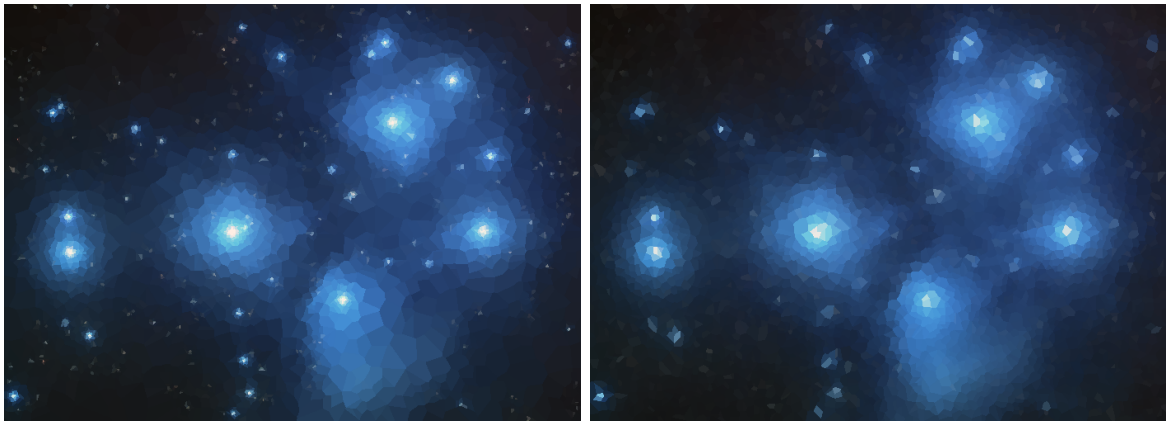

Figure 5: Starfield image decomposed using the two methods. Iterative is the left image, genetic is the right. Notice the extra stars in the iterative technique.

4 Future Work

Planned work for this project initially considered using CMA-ES for the evolutionary algorithm after getting initially promising results with the genetic algorithm. The representation of the individual in CMA-ES would be a vector of floats of x and y values of the position of the cells, which is an easy enough representation to implement. However, the complexity of CMA-ES itself made it difficult to implement from scratch in common lisp within the time allotted. If the initial program was in Java or C++, where there are strong numeric libraries, the problems would be lessened. The math is tricky to get right, and there is no strong scientific computing stack in common lisp, thus a lot of it would have to be implemented by hand, which would be finicky work. A good discussion of the algorithm is in section 9.2.3 in Essentials of Metaheuristics by Sean Luke, which demonstrates its complexity in detail.

Other future work would include looking at better global error functions, which could improve both techniques, as well as GPU implementations of the voronoi decomposition, which could potentially greatly speed up fitness functions for any evolutionary algorithm implementation.

5 Conclusions

Despite the fact that the iterative algorithm worked better for these experiments, the genetic algorithm is actually a global algorithm; given enough time, it would eventually converge to the best possible solution. However, on practical timescales, that is not the case for most images, and, in the amount of time it takes to work even reasonably well, the iterative method generally uses cells more effectively. If the goal is only aesthetics, the iterative method can just be given more cells to work with, and the computation time makes the iterative version, even with its flaws, more useful to work with. This is also true for placeholder images with a fixed number of cells, as these images can be generated automatically with a relatively inexpensive computer using the iterative method but would require a server farm for just a few images using the genetic algorithm. However, the current results show promise, and with future work into algorithms like CMA-ES and optimization of the code using GPUs and lower level languages, there could be potential for much better results.

References

[1] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Introduction*. Springer, 2008. 2.1

[2] Craig S Kaplan. Voronoi art http://www.cgl.uwaterloo.ca/csk/projects/voronoi/. (document)

[3] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/. 2.2

[4] Adrian Secord. Weighted voronoi stippling. NPAR, 2002. (document)