Geometric Algorithms Project: Artstract

Group 6: Steven van den Broek (1324268), Martijn Leus (1315366), Tristan Trouwen (1322591)

Introduction

We developed *Artstract*, a online tool (and toy) that automatically produces abstract art from a given input image. The tool attempts to initially produce an appealing output, and the user can further change the output by tweaking various parameters. The website and algorithms have been implemented from scratch; no external libraries have been used apart from *dat.gui* [1], which provides the graphical user interface for changing parameters. In this report, we describe the process by which the tool produces the art, and the difficulties we encountered during development.

Implemented Algorithms and Data Structures

We first give an overview of how an image is processed, referring to Figure 2. The process starts when the user provides an image; this image is then processed such that the individual pixels are accessible. Not all pixels of the image are used in the rest of the process since there may be millions, making subsequent operations slow. Instead, we employ a quadtree to produce a non-uniform sample of points. These points are then grouped on colour by using the median cut algorithm. For each group of points a Delaunay triangulation is created, after which comparatively large triangles are removed to obtain a weighted alpha shape. The resulting alpha shapes are layered, yielding the final image.

Point Sampling Using a Quadtree

Our goal of non-uniform point sampling of the image is that few points are used to represent an image well. Clearly, there is a trade-off between the number of points used, and the quality of the representation. We use an adapted quadtree [5] to determine which parts of the image contain detail, and sample more points in those parts. The main modification is the criterion for subdividing a node. Namely, a quadtree node containing a set of pixels P is subdivided if the following cost exceeds a certain threshold:

$$cost(P) = c_r V_r(P) + c_g V_g(P) + c_b V_b(P).$$

Above, $V_r(P)$, $V_g(P)$ and $V_b(P)$ are the variances of the red, green and blue channels of the pixels P respectively (with variance we mean the average squared Euclidean distance of a channel value in S to the mean channel value of S). The constants $c_r = 0.2989$, $c_g = 0.5870$ and $c_b = 0.1140$ are adapted from the referenced website [5].

Subdivision of quadtree nodes may continue until a maximum depth is reached, which can be configured by the user. After computing the quadtree, the point sampling takes place. Points are created on the corners and middle of the quadtree cells, in such a way that there are no duplicate points. This is illustrated in Figure 1. The coordinates of the points are rounded such that the points lie on the integer grid of the pixels of the image. This rounding makes it easy to assign each point a color, namely the color of the underlying pixel. Furthermore, the rounding allows further computations to be robust as it may prevent floating point errors.

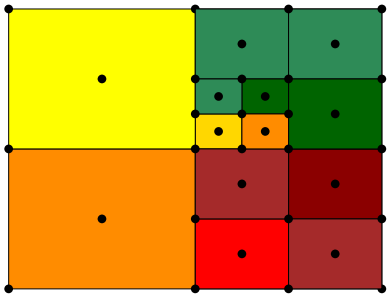


Figure 1: Quadtree point sampling

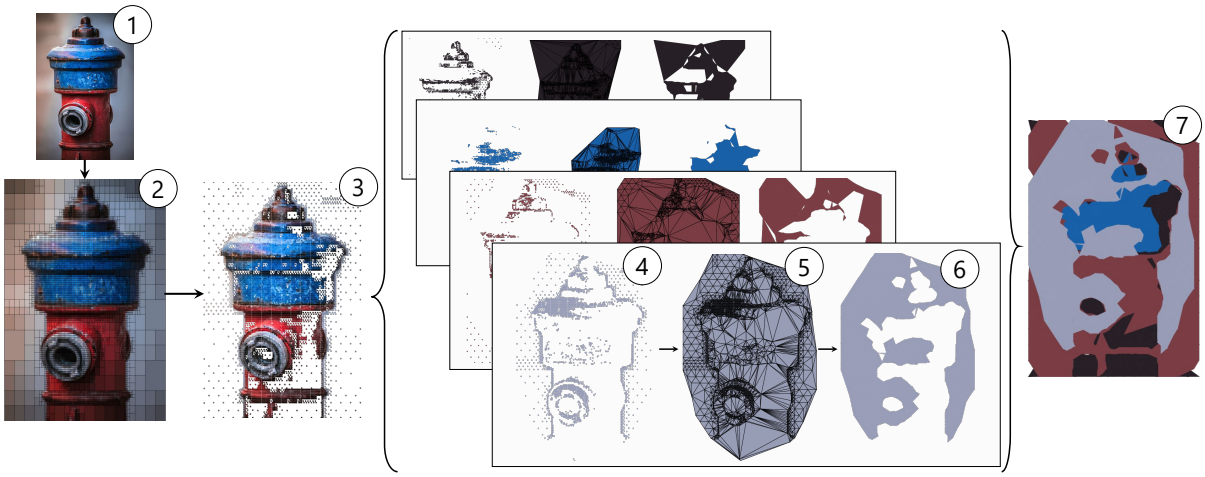


Figure 2: The process of producing an image. Steps: 1: input image; 2: create quadtree; 3: sample points; 4: group points on color; 5: perform Delaunay triangulation; 6: construct alpha shape; 7: layer the alpha shapes.

Color Quantization Using the Median Cut Algorithm

After having sampled a collection of points, we partition them based on their color. The colors of these points are first quantized to produce a palette of 2^k colors, where the positive integer k is a parameter configured by the user. This quantization is done using the median cut algorithm [4]. The points are then partitioned by mapping each point to the closest (Euclidean distance) color in the palette.

Delaunay Triangulation

A Delaunay triangulation can be computed using a randomized incremental construction algorithm [3]. This algorithm creates a bounding triangle around all points, after which all points are added one-by-one to the triangulation. When adding a point, the triangle in which it lies is divided into three triangles. In addition, if a vertex of a triangle is in the circumcircle of a triangle, the triangulation is not a Delaunay triangulation anymore. This is solved by an edge legalization process. A search structure is kept such that finding the triangle in which the point lies can be done in efficiently.

For the tool, we also use the search structures of the Delaunay triangulations for point location. Namely, when a user clicks with the mouse, a search on the mouse position is performed for each polygon, starting with the polygon that is on the top layer. If a triangle is found that is not the bounding triangle and has not been removed in further processing, then the user clicked on the corresponding polygon.

Alpha Shape

An unweighted alpha shape (as introduced in [2]) of a given alpha is computed from a Delaunay triangulation by filtering triangles based on their circumradius: if the circumradius of a triangle is at least alpha it is removed. Because we use non-uniform point sampling, however, this typically does not yield good results when using a constant alpha. In particular, triangles that contain points in large quadtree cells will be removed much more often than triangles with points in small quadtree cells. For this reason, we let alpha depend on the quadtree. Each point stores the length of the diagonal of the quadtree cell it originates from. The alpha that is used for a certain triangle is then a base alpha scaled by the largest diagonal length stored in the three points of the triangle. To obtain

a base alpha, we use a binary search to find an alpha such that all points are part of some triangle. To prevent the alpha from becoming too large to satisfy the condition, an upper bound is set and can be controlled by the user by adjusting the limit alpha parameter.

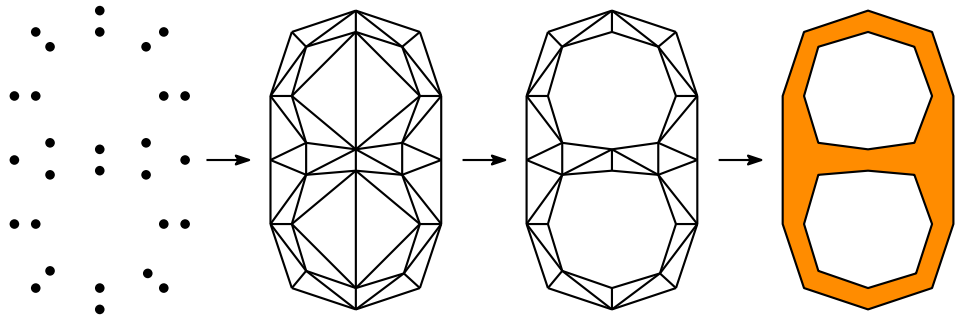
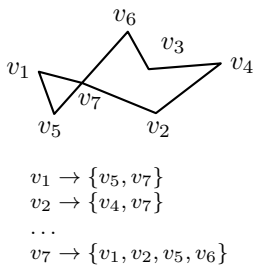


Figure 3: Process of computing an (unweighted) alpha shape

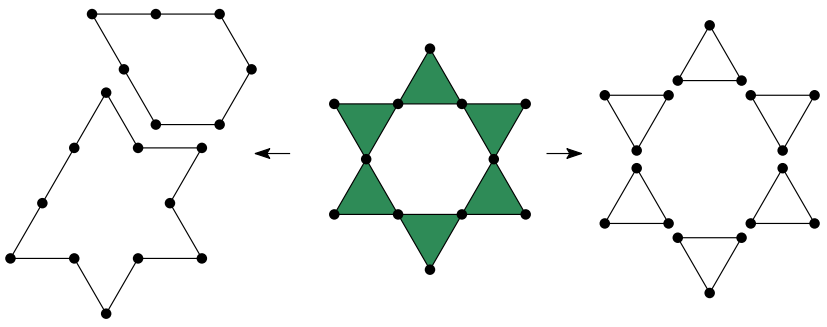
Outline of Alpha Shape

As described above, we obtain the alpha shape by filtering triangles from the Delaunay triangulation, therefore the alpha shape is represented by a collection of triangles. The interior of an alpha shape can then be drawn on the screen by drawing all triangles in the corresponding collection, but to draw the outline of the alpha shape more work needs to be done.

To compute the outline of the alpha shape we first determine the boundary edges. For this, we count for each edge how often it is part of a triangle in the alpha shape: an edge that is counted exactly once lies on the boundary, while edges that are counted twice lie in the interior of the alpha shape. One could stop here, and draw each boundary edge separately, but drawing full polygons gives better results for the corners of the polygon. Therefore we want to determine the polygons that the edges form. For this, we create an adjacency list which maps a vertex to the list of its neighbours as illustrated in Figure 4a. Using this adjacency list, a search for cycles is performed. The search starts at an arbitrary vertex, and each time an edge (v_i, v_j) is traversed vertex v_i is removed from the neighbours of v_j and vice versa. When the vertex we visit is the same as the one where we started, a cycle has been found and the corresponding polygon is output. A new arbitrary start vertex is then chosen from the vertices that have not been visited before. This process continues until every vertex has been visited. Note the set of outline polygons is not necessarily unique, see Figure 4b.



(a) Example adjacency list



(b) A shape that has multiple boundary polygon decompositions. Two example decompositions are shown on the left and right of it.

Encountered Difficulties

In this section we list some difficulties we encountered during the course of this project.

Delaunay Triangulation

- The pixels of an image have only integer coordinates, however JavaScript did introduce floating point errors when we did not explicitly round the coordinates.
- With a deep quadtree, duplicate points arose due to the middle and the corners of a cell being equal when rounded. These duplicate points introduced errors in the Delaunay triangulation (e.g. triangles overlap); this was time-consuming to debug.
- Due to not implementing the Delaunay triangulation with a DCEL, we had difficulties computing adjacent triangles. This was solved by having each node in the search structure point to the nodes of adjacent triangles.
- To output all the triangles of the triangulation we initially recursively searched the search structure to find all leaf nodes. Because of a coding mistake this was initially very slow, often taking minutes. After fixing this mistake, it was still one of the bottlenecks in our calculation. We improved the performance a bit further by using the pointers to adjacent triangles. Firstly a single triangle was found in the search structure, after which we go over all adjacent triangles. This process is repeated until all triangles are output. Asymptotically the two methods are equally efficient, but in practice utilizing the adjacent triangles sped up the computation significantly.

Alpha Shape

- We had some trouble deciding the representation of the alpha shape. Initially, we wanted to have a polygon representation of the alpha shapes, and therefore computed the boundary polygons of the alpha shape. However, because alpha shapes may contain holes, we would need to determine which boundary polygons are holes. A brute force approach to determine this may take quadratic time, therefore we decided to simply keep the alpha shapes as triangles and only use the boundary polygons to drawn an outline.
- The alpha shape is not a very well-known computational geometric object, therefore there was comparatively little information available.

Workload Distribution

1. **Tristan**: Implemented the Delaunay triangulation, and prepared and performed the presentation.
2. **Steven**: Implemented the quadtree and alpha shape, created the UI and glued the different algorithms together to get an output image.
3. **Martijn**: Implemented the median cut algorithm and wrote the introduction of the report; proofreading the report.

In addition, for the report everyone wrote the parts belonging to the algorithms and data structures they implemented.

References

[1] Dataarts. *Dataarts/dat.gui: Dat.gui is a lightweight controller library for JavaScript.* URL: `https://github.com/dataarts/dat.gui`.

[2] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. "On the shape of a set of points in the plane". In: *IEEE Transactions on Information Theory* 29.4 (1983), pp. 551–559. DOI: `10.1109/TIT.1983.1056714`.

[3] L. J. GUIBAS. “Randomized Incremental Construction of Delaunay and Voronoi Diagrams”. In: *Algorithmica* 7 (1992), pp. 381–413. URL: https://ci.nii.ac.jp/naid/80006333550/en/.

[4] Paul Heckbert. “Color Image Quantization for Frame Buffer Display”. In: *ACM SIGGRAPH Computer Graphics* 16.3 (July 1982), pp. 297–307.

[5] Jeffery Russell. *Segmenting images with Quadtrees*. URL: https://jrtechs.net/photography/segmenting-images-with-quadtrees.