Exploring Triangulation

Tim Hung, Miguel Lumapat, Alan Plotko

Binghamton University, CS 455 Term Project

Problem Statement

Triangulation is a process where you take a set of points and create triangles with each triplet of points within the set. This thus allows us to draw triangular sub-regions across a given image. The procedure can be extended to various fields, namely graphics and visualization, but our research particularly focuses on exploring applications of triangulation in areas of re-visualization and compression. For the former, we produce images from other images that contain some unique twist. In this case, re-visualization pertains to matters like generating low-poly art. This art style is composed of images that are triangulated in such a fashion that pixels of similar intensity or hue are enclosed within a triangular sub-region and computed to one average intensity or hue for the entire sub-region. For the latter, we are interested in increasing the ratio of the image quality, and thus the image's specific recognition factor, to file size reduction.

Literature Review

The Delaunay triangulation on a set of points is a triangulation such that the smallest angles in each triangle are maximized. More precisely, we can define the MaxMin angle criterion on a set of triangles. Let $\boldsymbol{I}(\boldsymbol{\Delta^k})$ be a non-decreasing ordering imposed on a set of $\boldsymbol{k}$ triangles.

$$I(\Delta^k) = \left(\beta_1, \beta_2, \cdots, \beta_{|T|}\right), \beta_i \geq \beta_j, i < j$$

where β_i is the smallest interior angle of each of the $\boldsymbol{k}$ triangles. Thus, the MaxMin angle criterion establishes an ordering $J(\Delta^k)$ that is lexicographically larger than any other ordering on the set of triangles. In this case, "lexicographically larger" means that all minimum interior angles in J are greater or equal to that in any I.

The Delaunay triangulation on a set of points is thus an ordering of triangles produced by triangulating the points so that it is lexicographically the largest in accordance with the MaxMin criterion. In some other cases, we can establish a MinMax angle criterion that seeks for a minimum in the largest interior angles. However, we only focus on the MaxMin angle criterion in our research. An alternate definition of the Delaunay triangulation is a triangulation such that all vertices do not lie inside any circumcircle of all triangles produced. Nonetheless, this is only relevant for algorithms that utilize circumcircles to create Delaunay triangulations.

Another existing technique for triangulation is the even distribution method, where we break up an image into equivalent sized triangular sub-regions, each colored with a hue or intensity corresponding to the average hue or intensity of the pixels in that sub-region. Even distribution method is a consistent form of triangulation and can be applied in compression. Compression is widely used to reduce an object's storage requirements while conserving its other factors. In the context of image compression, we consider image quality to be the main factor of conservation. Image compression can be classified into two types: lossless compression

and lossy compression. Lossless compression allows for us to restore the original state of the image data, which is ideal for people who wish to share images with one another. For instance, users of social media networks often find that their high-resolution photos diminish in quality once uploaded. While it is beneficial for companies to cut down on space (considering their scale and the number of photos uploaded daily), it is disadvantageous for them to upset their users with compression that ruins photo quality. We call this the consumer's perspective because photographers share the common concern about maintaining the same quality after compression. Lossy compression, on the contrary, will discard information from the image. Not all information in photos are vital, and some information may prove redundant. When considering all possible scenarios involving images, many do not revolve around the consumer's perspective. We will explore other perspectives where information loss is acceptable beyond human standard; we call this the machine's perspective.

Low-poly art arose from the primitive graphics engines of the 1990s. In a shift from the purely 2D or pseudo-3D design in video games or computer assisted art, graphics programmers began to develop for more advanced 3D renderings. By representing real objects with an approximated polygon mesh of triangles, game developers and computer animators were able to merge the physical world into the digital world. Unfortunately, the processing power of hardware around the turn of the century was not yet capable of rendering both high polygon counts and maintaining a reasonable frame rate. This meant that the tradeoff for an acceptable frame frequency of around 30 frames per second (FPS) was a decrease in the number of polygons used to portray 3D objects. Despite

Figure 1: A render of a Legend of Zelda video game

its reception as a reluctant compromise, these media that utilized low polygon counts steadily gained recognition for their unique digital appearance and artistic merit over the past twenty years. While the personal computers and home consoles of today effortlessly display models of individual resources with polygon counts in the hundreds of thousands, fans of low-poly graphics still enjoy the art style for its charm and nostalgic value.

Transforming Photographs into Low-Poly Art

Nowadays, low-poly artists exercise their craft by utilizing tools such as Adobe Photoshop. Using high resolution photographs as references, the artists meticulously trace over the contours and curves of the image by hand, tessellating the image into hundreds of adjacent triangles. Artists paint beautiful low-poly portraits and landscapes by cleverly placing these regions around contiguous patches of similar colors or textures and then subsequently coloring in the regions with their average color from the original photograph.

Figure 2: a sample of the low-poly art style on a face of a lion

Using techniques from image processing and computational geometry, we aim to automate the arduous hand drawn process of creating a low-poly rendering of a preexisting image, which currently takes artists several tedious hours. To do so, we will examine several methods of triangulating images and evaluate them in search of a good and valuable implementation.

Method

The first method we will consider involves a simplistic brute force approach to triangulation. By iterating over equally sized blocks of an image, we can partition it into an orderly tiled mesh of equally sized triangles. This evenly distributed mosaic is useful because it is easily and quickly generated independently of the details in the source image. We discovered that this form of triangulation is convenient for a form of primitive image compression: this process is detailed in a later section. However, this method is unsuitable for emulating low-poly art as it completely disregards the features of the source and in doing so, results in a loss of both image quality and subjective artistic merit. We must resort to a more advanced method of triangulation upon which we analyze our source image, procedurally generate a set of points to effectively portray important features on the source, and ultimately use Delaunay triangulation to partition the points into triangular regions.

Delaunay triangulation (will be elaborated upon later) triangulates on a set of points. These points will form the vertices of the image's triangular regions, which means that the edges between the vertices should demarcate different areas of interest in the source image, especially changes in color. From this understanding, we aim to generate points on areas of the image where there are sharp changes in intensity, namely the edges of images. A primitive way to prioritize placement for seed points on these edges is as follows:

1. Generate the edge image of the source using the Canny Edge Detector
2. Randomly place seed points along the generated edges
3. Run Delaunay triangulation on those points

This procedure is effective, but we would like to devise a method that does not rely on randomness. Branching off from our idea of using edges to segment our image, we will narrow our search of important features by detecting corners instead.

Formally, a corner can be defined as the intersection of two edges in an image. We would like to find corners and place points on them, as this helps us place points on areas of even higher contrast, extends our previous method, and avoids randomly choosing points on edges. As with edge detection, corner detection employs a myriad of notable and effective algorithms. For our purposes, we will be utilizing the algorithm described by Shi and Tomasi (1994). This algorithm further develops the underlying theory behind the classic Harris & Stephens Corner Detector algorithm (1988) by proposing a new and updated scoring function. Conveniently, OpenCV has a high quality implementation of the Shi-Tomasi corner detection algorithm. Our improved point generation procedure is as follows:

1. Find the best corners of the image using the Shi-Tomasi algorithm
2. Place a specified number of seed points on the best corners
3. Run Delaunay triangulation on those points

Results

These following images depict the seed points generated by our two algorithms on this grayscale image of a scenic riverscape of the Swiss city of Basel.

Figure 3: the original grayscale image of a riverscape in the Swiss city of Basel

First we will examine the results of our random edge points procedure.

Figure 4: the result of generating the seed points on figure 3

In figure 4, you can see that all our seed points were generated on the edges of the image. Notably, the points seem to outline many of the prominent features in the photograph, especially the rooftops and borders between buildings. One immediate downside to this method is seen in the sporadicity of the point placement. Since point placement on edges is purely random, there is no real mechanism to prevent point clustering, so we get many undesirable cliques of points.

Next, we will examine the results of our corner detection point seeding procedure.

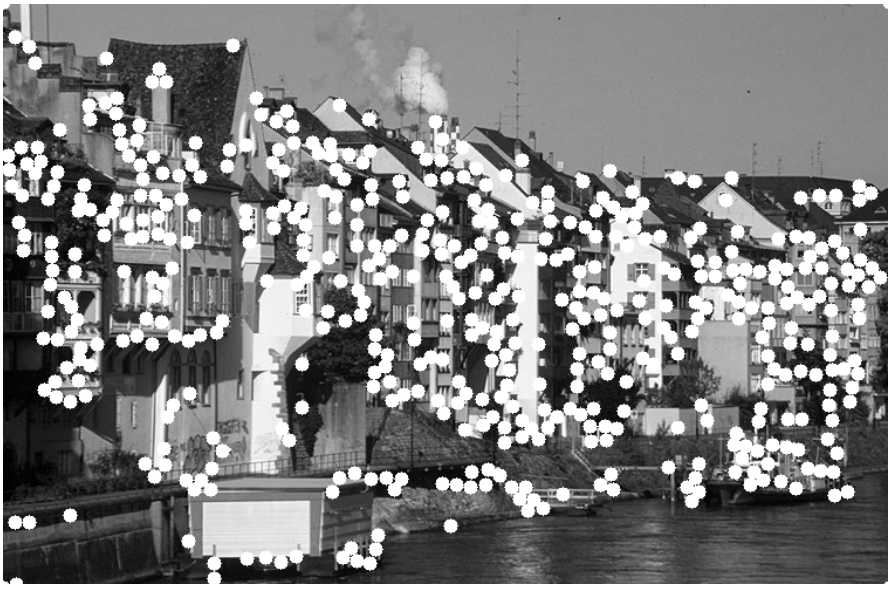


Figure 5: the result of generating the corner points on figure 3

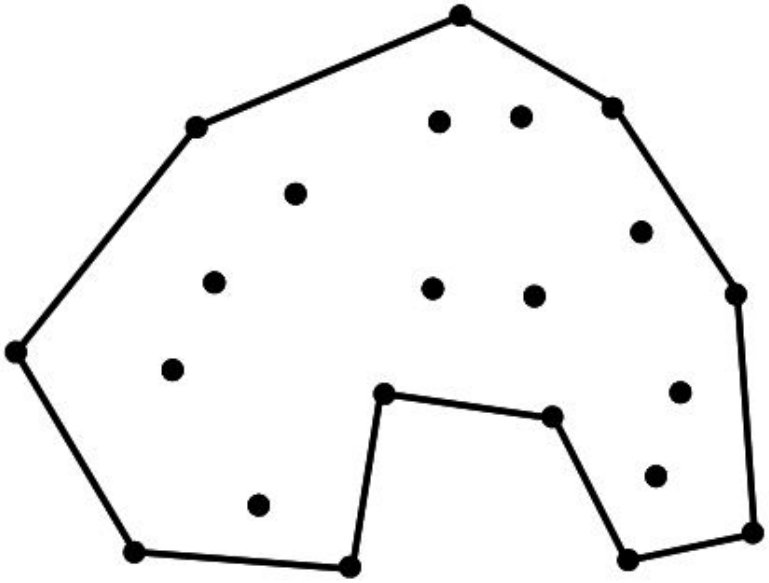
As with the previous method, in figure 5, we also generate all our points on the image's edges, as that is how corners are defined. Again, this demonstrates the tendency for the points to lie along the contours of the image and surround contiguously intense regions. Immediately, we can see an improvement in the clustering of points due to the lack of randomness in point selection. We see that points are chosen at areas with sharp angles and help demarcate the numerous regions of the image. Now that we have explored several ways to prepare an image for triangulation, we will go in depth on how we can tessellate triangles onto these point fields.

Delaunay Triangulation

Method

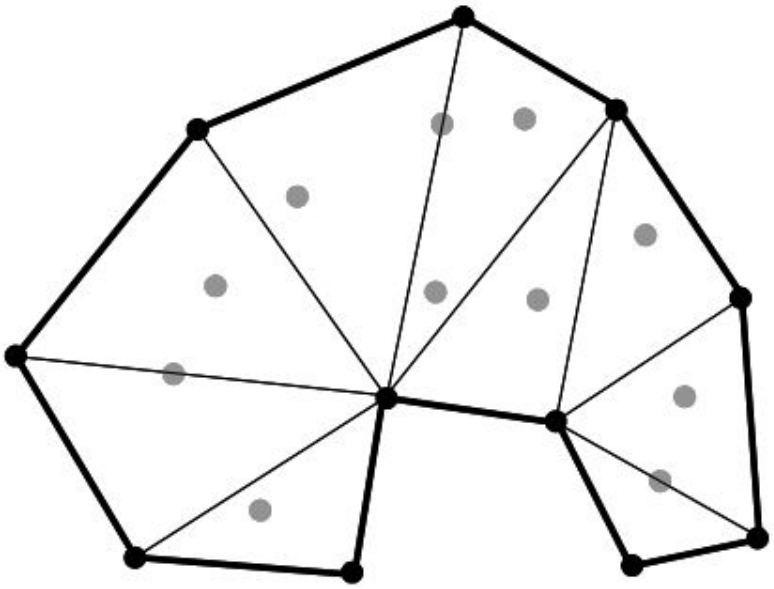
For this project, a slightly different approach to Delaunay triangulation was implemented. However, all approaches revolve around a fundamental set of procedures. Thus, the basic procedure for producing a Delaunay triangulation is the following:

Step 1. First, the convex hull of a set of points is computed. This can be done in any manner (or incorporated with the other steps in the triangulation).



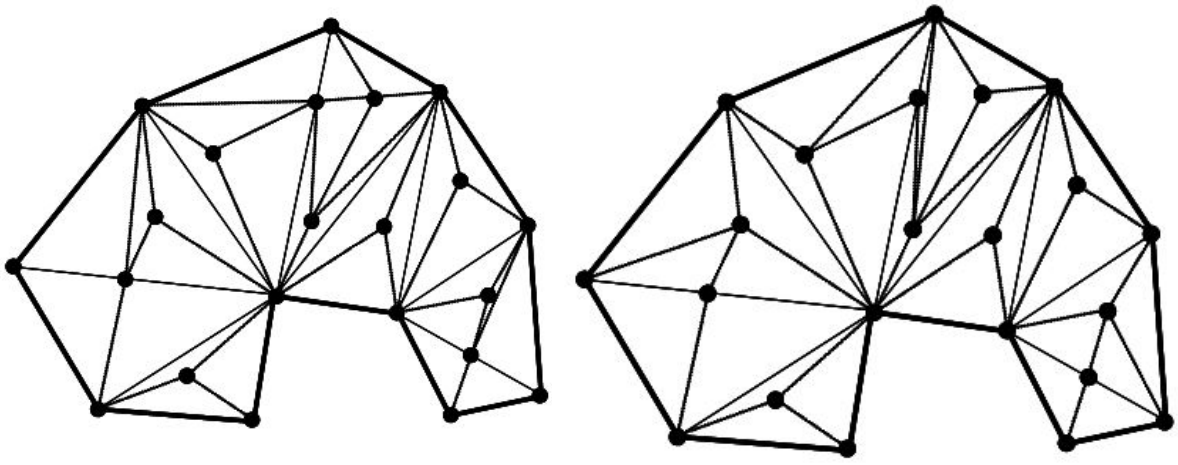
Step 1

Step 2. Then, a triangulation is done on the convex hull points to provide us with an initial triangulation.



Step 2

Step 3. The rest of the interior points are then inserted one-by-one. If a point lies inside an existing triangle, a segment is drawn to each vertex from the point, creating three triangles. If a point lies on a segment, then a segment is drawn to each vertex that subtends the segment containing the point.



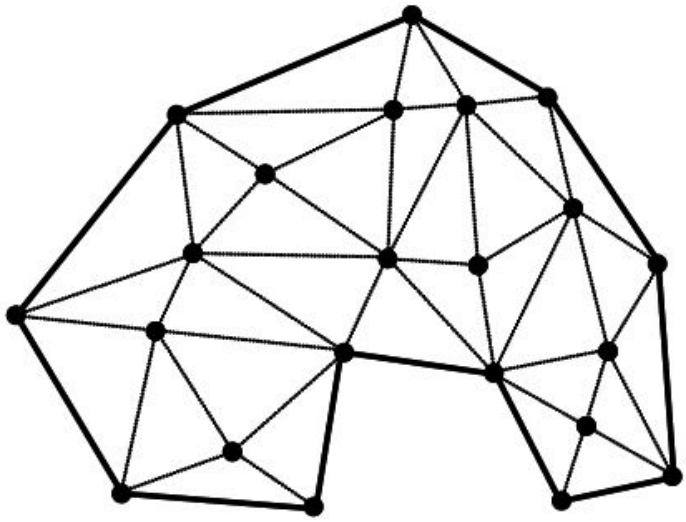
Step 3

At this point, most of the triangles produced by the previous triangulations are in a suboptimal state. That is, the current ordering of smallest interior angles isn't the lexicographical highest.

Step 4. Thus, we perform a local optimization procedure (LOP) which performs two operations:

1. Determine if the smallest interior angles in a triangle can be increased.
2. If it can, perform an edge swap.

This can be best illustrated in a quadrilateral formed by two triangles. A diagonal exists to show that the quadrilateral comprises two triangles. Thus, an edge swap should swap the diagonal, moving it to the other pair of vertices. Refer to the changes in the following diagram for a visual example of the edge swap.



Step 4

LOP is performed until the triangulation cannot be optimized any further. This would then be the Delaunay triangulation being sought after. The algorithm for Delaunay triangulation

utilized in our research is a bit different from the basic procedures outlined above. The algorithm is called radial sweep triangulation, and it utilizes the centroid of a set of points. First, the point closest to the centroid of the set is computed. A segment is then drawn from the centroid to each point. From this, an initial triangulation is formed by connecting each of the points with its neighbors. The convex hull is then established by creating triangles along the boundaries where segments resemble a side of a concave polygon. Finally, LOP is performed which utilizes the circumcircle test outlined in Hjelle and Daehlen's paper. This test includes a series of checks on the sines and cosines of the interior angles in each triangle of a quadrilateral that help determine whether an edge swap should occur. The following is a list of identities used to calculate the sine and cosine:

$$\sin\alpha = (\mathbf{v}_1 \times \mathbf{v}_2)\cdot\mathbf{e}_3 = \frac{(x_3-x_2)(y_1-y_2)-(x_1-x_2)(y_3-y_2)}{\|p_3-p_2\|_2\,\|p_1-p_2\|_2}$$

$$\sin\beta = (\mathbf{v}_3 \times \mathbf{v}_4)\cdot\mathbf{e}_3 = \frac{(x_1-x_4)(y_3-y_4)-(x_3-x_4)(y_1-y_4)}{\|p_1-p_4\|_2\,\|p_3-p_4\|_2}$$

$$\cos\alpha = \mathbf{v}_1\cdot\mathbf{v}_2 = \frac{(x_3-x_2)\,(x_1-x_2)+(y_3-y_2)\,(y_1-y_2)}{\|p_3-p_2\|_2\,\|p_1-p_2\|_2}$$

$$\cos\beta = \mathbf{v}_3\cdot\mathbf{v}_4 = \frac{(x_1-x_4)\,(x_3-x_4)+(y_1-y_4)\,(y_3-y_4)}{\|p_1-p_4\|_2\,\|p_3-p_4\|_2}.$$

Figure 6: a list of identities to compute sine and cosine

These sines and cosines, as previously stated, will serve as the criteria for issuing an edge swap:

Algorithm 3.1 Circumcircle test

```
1. if (cos α < 0   and   cos β < 0)
2.     return FALSE    // swap the edge
3. if (cos α > 0   and   cos β > 0)
4.     return TRUE
5. if (cos α sin β + sin α cos β < 0)
6.     return FALSE      // swap the edge
7. else
8.     return TRUE
```

Figure 7: a simple test of whether edge swapping should occur

Results

The following images show the radial sweep in each of its phases/procedures. Here is the centroid and segments drawn from it to all the other points.

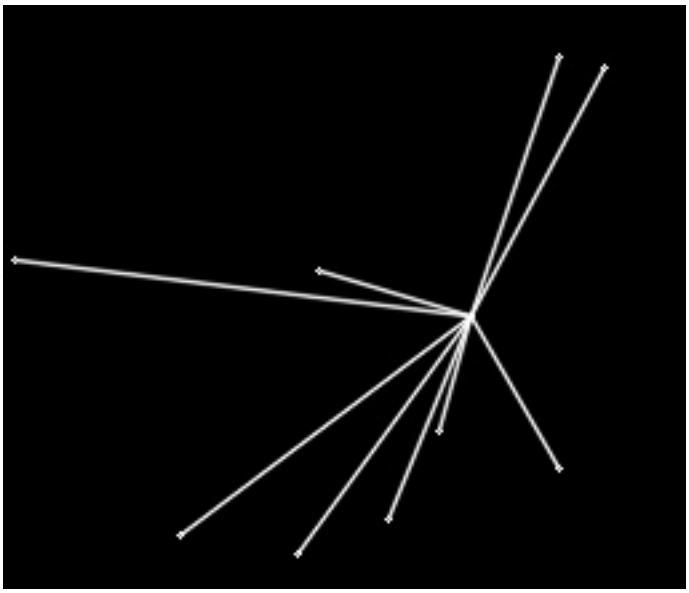


Figure 8: a radial sweep where the centroid is linked to all other points

Then, all the outer points are connected to each other to create an initial triangulation.

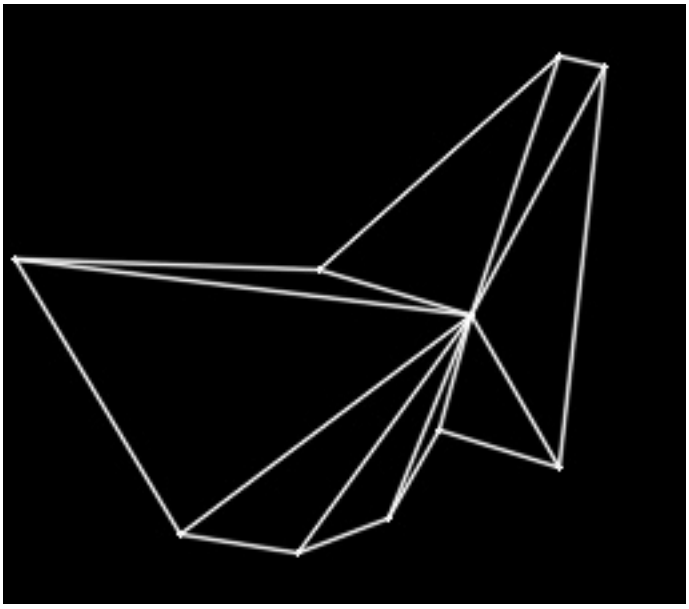


Figure 9: set up for initial triangulation

From this, the convex hull is then completed by creating triangles from outer acute angles formed by previously drawn segments.

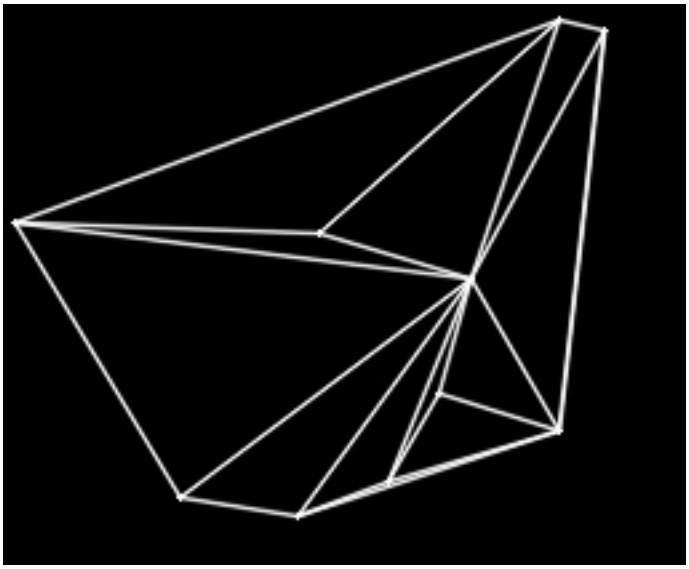


Figure 10: the completed convex hull

Finally, LOP is performed. Unfortunately, the precision loss of floating-point values and order of segment/triangle processing cause the local optimization procedure to loop forever. Thus, LOP was set to run for 100 iterations to capture all optimizations.

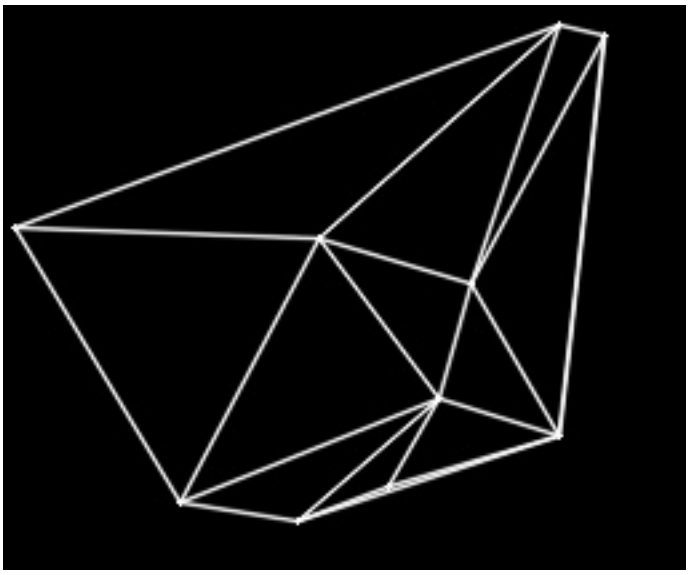


Figure 11: the result after running LOP a fixed number of times

The previous images were a triangulation performed on a set of ten points. The following is thus a triangulation of the points generated by the procedures introduced earlier, in the point generation portion of our research. Figure 12 depicts the riverscape from figure 5 after a triangulation produced by choosing random edges. Figure 13 depicts the riverscape after a triangulation using points produced by Shi-Tomasi.

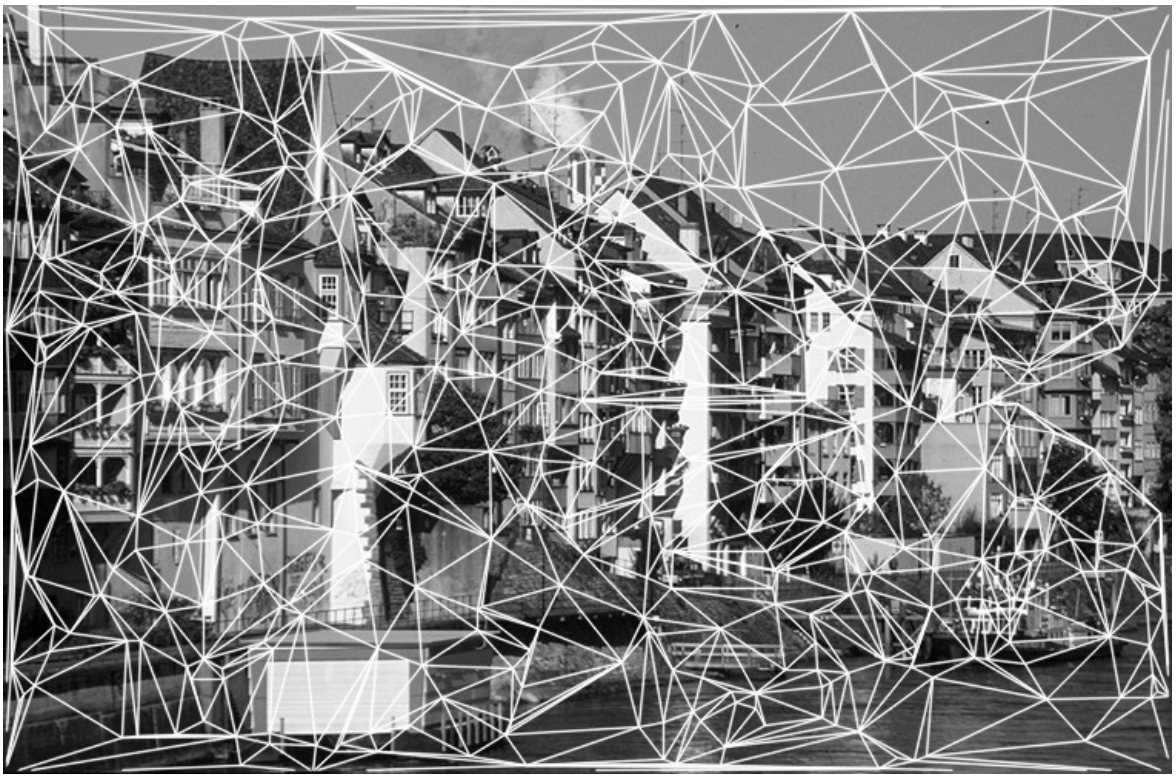


Figure 12: a triangulation on figure 5, produced by choosing random edges

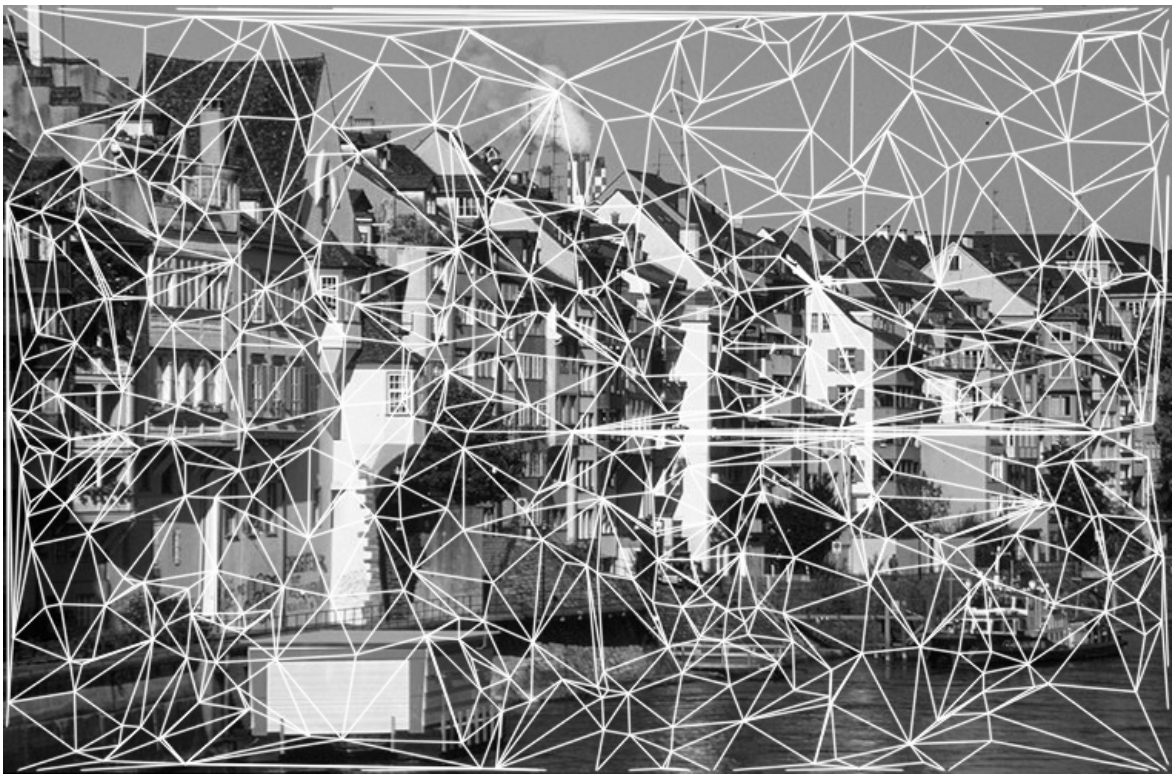


Figure 13: a triangulation on figure 5 with points produced by Shi-Tomasi

Each of the images have a point inserted in each of the four corners to help triangulate all the regions around the image. 500 points were generated for each. Occasionally, a suboptimal configuration of triangles is produced, which causes a dense region of segments or multiple, lengthy scalene triangles to show up. Nonetheless, a stable Delaunay triangulation should show up for the most part.

Using Triangulation for Compression

Method

Regardless of what triangulation method we employ, every pixel is contributing information regarding the image. In a grayscale image, every pixel has an intensity represented by an unsigned character. In a color image, every pixel has three channels: blue (B), green (G), and red (R). We can store these RGB values as unsigned characters as well. A wallpaper of size 1920 x 1080 equates to storing 2,073,600 unsigned characters for a grayscale image and 6,220,800 unsigned characters for a color image. This deliberately assumes that we store this information individually and solely this information. We must also consider other information such as the image dimensions, meta data, and so forth. Existing image formats optimize storing this information in various ways and compression algorithms address different problems to further reduce storage requirements. Our research will address the use of triangulation, namely the even distribution method, as a new approach to reducing storage requirements.

Storing information for every pixel is the purest method of brute force. In our tests, it usually ends up surpassing the original image size. However, in high-resolution photos, we often can zoom in and notice large regions of pixels that have little to no differences. Select any arbitrary pixel in such a region and compare it with its neighbors. The difference in intensity or

color is very minor, but exists to highlight finer details. These details matter to consumers who want to maintain the original image quality; however, we will elect to discard such information where feasible and reasonable. Our case study focuses less on the consumer's perspective where the consumer's purpose is to share content freely among others at the highest quality possible. There is a myriad of other reasons for why we share visual data, but all reasons share one common focus: understanding the image. Consumers want high quality because they want to be able to understand the image—its subject, location, and what it describes. Machines also seek to understand the image for their own programmed purposes. Sample scenarios include neural networks and restrictive environments for transmitting visual data.

In the former scenario, neural networks are created and trained to identify subjects from photos in their original format as well as in reduced forms such as edge information. Microsoft's Kinect device, for example, tracks the motions of players in its field of view. Kinect shouldn't care for the appearance of the players or the finer details and patterns of their shirts. We could discard all this information and focus on the important details—the edges that contribute to the outlines of the subjects and the motions of their bodies (e.g. their arms and legs). Google's "Quick, Draw!" demonstration, which identifies the subject of a person's drawing, also discards a lot of information when it selects the necessary regions to make matches between the submitted drawing and the training images in its data banks. Altogether, machines do not value finer details as much as people do. In the latter example, we may be operating in low bandwidth environments where sending visual information is costly such as with data caps for consumers on mobile devices. For identifying subjects in a generalized fashion, we do not need to transfer every minor detail. Traffic reporting may care more for the presence of many cars than the make

or model of these cars. A security camera may care for the presence of people than it does a specific person when it decides whether to start record information or just proceed in monitoring.

To discard information, we employ the even distribution method in which we split an image into fixed-size triangular sub-regions as opposed to by their edge information and corners. This allows us to generalize the image in a consistent manner. Given the resulting image, the subject should still be distinguishable. The foreground and background should also maintain clarity. The algorithm works by traversing a copy of the original image matrix from left to right, top to bottom. We look at one block at a time where the block is of an initial size chosen by the user. Draw a diagonal across this block from the bottom left corner to the top right corner. We then analyze the two triangular halves. We will calculate the average intensity (if grayscale) or RGB values (if color) and overwrite the matching information in the original, essentially remapping the left and right halves of the sub-region into two single intensity values or colors. We have a single channel (intensity) in grayscale images and three channels (RGB) in color images. We will track the average channel values for use in replacing all pixels within the corresponding triangular half of the sub-region. The larger the block size, the larger the influence of neighboring pixels. We thus observe a correlation in accuracy for maintaining image quality with the initial selected block size for traversal. Accuracy increases as block size decreases. In that case, we may conclude that we'd like to keep the accuracy as high as possible and region size as low as possible; on the contrary, that brings us to our original brute force method where a block size of zero implies we store all information as one-to-one as opposed to one-to-many.

Figure 14 on the right displays the pixel colors in a sample 12 x 12 sub-region. Unless zoomed in appropriately in the original photo, the colors can seem indistinguishable from one another at the standard view. We want to take advantage of this recurring feature in high resolution photos where large patches of pixels contain redundant information or minor details. Thus, we will have the same sub-region featured in figure 14, split into two triangles by the diagonal drawn from the bottom left corner to the top right corner. The result is not too different from where we started: we have the same number of pixels to store. All that changed was the coloring of each pixel. However, this is what will allow us to cut down our storage requirements. We will create a new custom file format where we store only the necessary components to reconstruct what we'll call a "triangulated image". Triangulated images are images that can be broken down into equivalently sized triangles, each colored with a single color. We should only need the image dimensions, block size, and sub-region colors to reconstruct the image. As such, we see a reason to increase the block size as a trade-off for lessening our accuracy: we store much less data. In a 12 x 12 subregion such as figure 14, we have 144 pixels to store. That's two triangular regions of approximately 72 pixels in size each. It is redundant to store the varying minor difference of green 144 times. Instead, we can reduce it to one color per triangle, which equates to 1 or 3 unsigned characters per triangle. The larger our block size, the larger our savings, and the smaller our accuracy. We'll allow the user to control the block size so that they can determine the best trade-off where the subject is still distinguishable and the foreground and background have not mixed too much. This algorithm thus allows us to generalize the image in a consistent manner.

Figure 14: a blown-up view of a sample sub-region

Results

The following image of a parrot in figure 15a is where we acquired the sub-region in figure 14. This is a high-resolution photo of size 3456 x 5184. If it were grayscale, we'd store 17,915,904 pieces of information, likely as unsigned characters. However, we have three channels, so three times as much information for 53,747,712 unsigned characters in total. This is a terrifying amount of data to store. The .jpg format takes 6 MB to store the parrot. We choose a block size where the subject is still distinguishable and then triangulate the image, obtaining figure 15b. Given a block size of *X*, each triangular sub-region costs 1 pixel, for which we store a single unsigned character or the set of RGB values. Our reconstruction or import phase of the program will reconstruct the entire sub-region with that single pixel's information. Our overall ratio in terms of savings per triangular sub-region can be consistently represented as $\frac{X^2-1}{X^2}$.

Given a block size of 50, for instance, that would mean we reduce having to store 2,500 pixels into just the average pixel for each triangular sub-region, which takes up 6 unsigned characters or two vectors in OpenCV. Half of either datatype is dedicated to the RGB value representing one triangle's new color. That's a massive amount of data that we cut down. Our triangulated image alone, when maintaining the .jpg format, brings us down to 2.1 MB and the custom file format that we denote as .cs455 is 7 KB. We store the image dimensions, region size, and the unsigned characters or vectors per sub-region into the file. The table in figure 15c shows our savings across both grayscale and color versions of the original parrot depicted in figure 15a. We also observe the results of triangulation using the even distribution method on a 4K high-resolution photo of a horse galloping in the grasslands. The results are as follows:

Figure 15a: the original color image of a parrot for which we derive the subregion depicted in figure 14

Figure 15b: the triangulated version of the parrot in figure 15a

Color Space Comparisons		
JPG Format	6 MB	Original Image
Triangulated Image	2.1 MB	63.33 % Savings
Custom File Format	7 KB	99.88 % Savings

Figure 15c: the amount of data saved from the original color image is displayed above for .jpg format and the custom .cs455 format

Figure 16a: the original grayscale image of a parrot

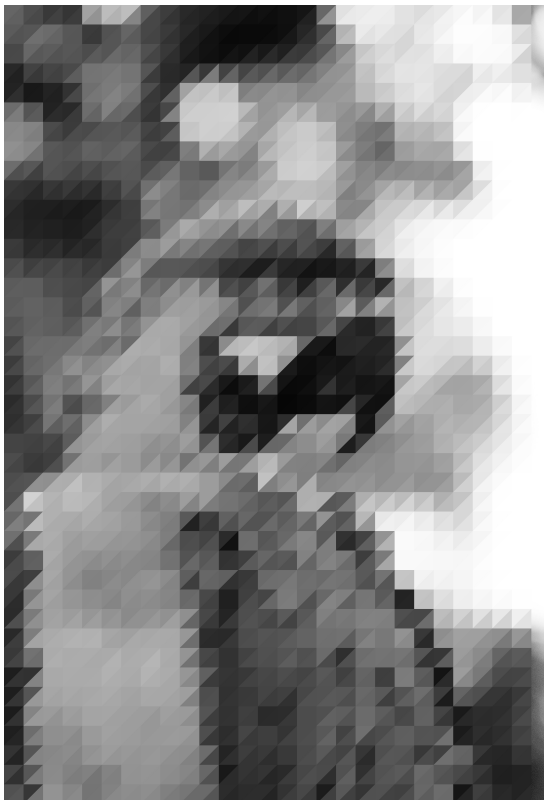


Figure 16b: the triangulated version of the parrot in figure 16a

Grayscale Space Comparisons		
JPG Format	5.6 MB	Original Image
Triangulated Image	1.5 MB	73.21 % Savings
Custom File Format	2 KB	99.96 % Savings

Figure 16c: the amount of data saved from the original grayscale image is displayed above for .jpg format and the custom .cs455 format

Figure 17a: the original color 4K high-resolution photo of a horse galloping in the grasslands

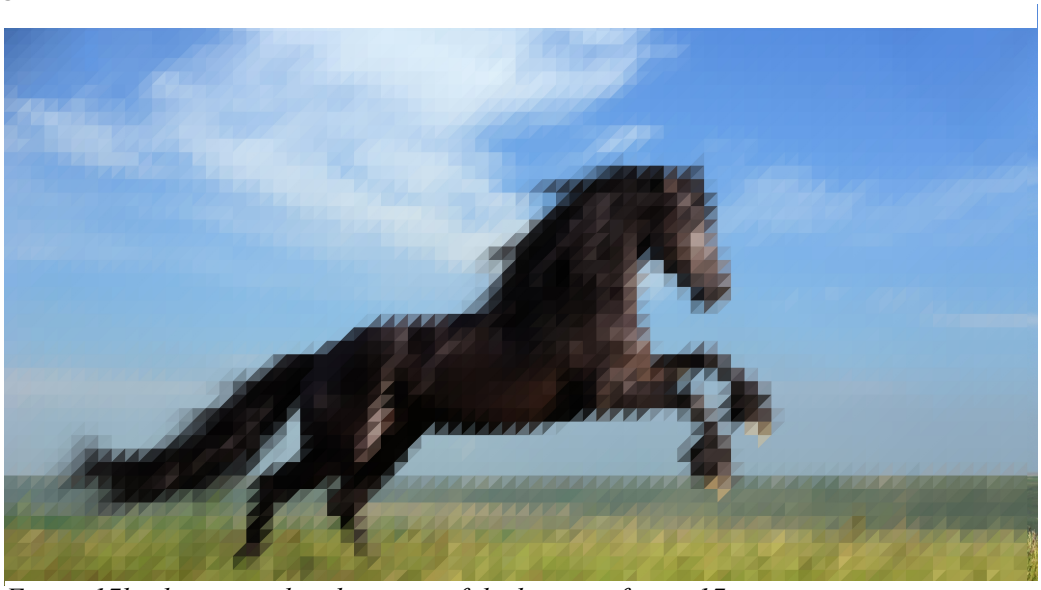


Figure 17b: the triangulated version of the horse in figure 17a

Color Space Comparisons		
JPG Format	2.2 MB	Original Image
Triangulated Image	1.3 MB	48.00 % Savings
Custom File Format	20 KB	99.20 % Savings

Figure 17c: the amount of data saved from the original color image is displayed above for .jpg format and the custom .cs455 format

Figure 18a: the original grayscale 4K high-resolution photo of a horse galloping in the grasslands

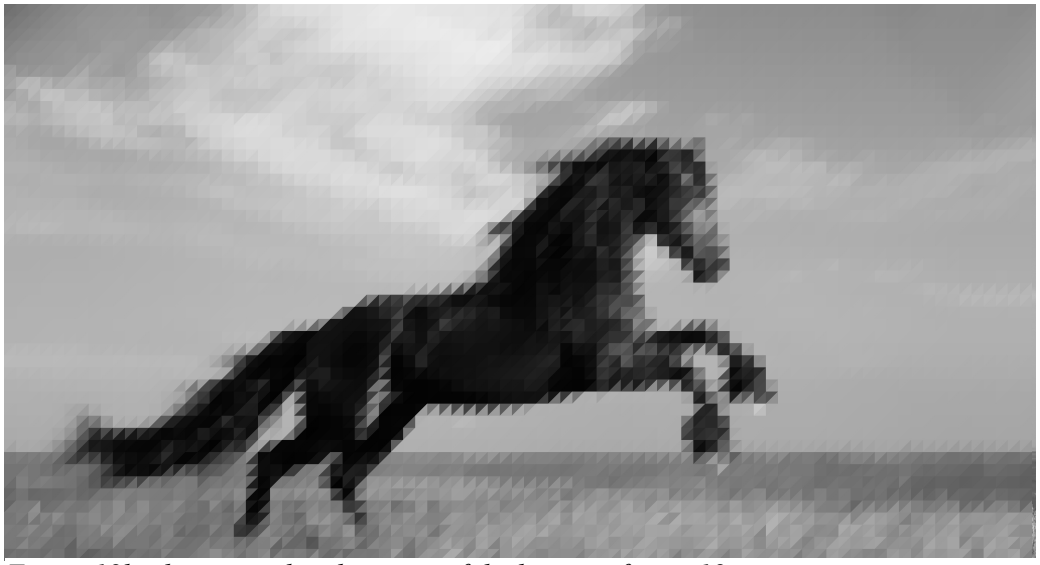


Figure 18b: the triangulated version of the horse in figure 18a

Grayscale Space Comparisons		
JPG Format	2.3 MB	Original Image
Triangulated Image	937 KB	59.26 % Savings
Custom File Format	8 KB	99.65 % Savings

Figure 18c: the amount of data saved from the original grayscale image is displayed above for .jpg format and the custom .cs455 format

Conclusion and Discussion

In this project, we achieved the generation of points of interest on an image, the Delaunay triangulation on those sets of points using radial sweep, and the compression of images via triangulation.

In order to automate the process of low-poly art generation, we examined how artists skillfully triangulate images by hand. By emulating their techniques, we devised and analyzed several procedures on generating a point field that outlines the features of a source image to triangulate upon.

For Delaunay triangulation, radial sweep was implemented and performed on the set of points generated by random edge selection and Shi-Tomasi corner detection. Most triangulations were in accordance with the Delaunay criteria, although suboptimal triangles were occasionally produced. Although this was not implemented in our project, the entire process would be completed by coloring in the triangular regions with their average intensity.

Our results in compression conclude that by discarding the fine details of an image, we can maintain the subject while immensely reducing our file size. The primary reason for these results is attributed to the number of pixels we can replace each sub-region with. Given a sub-region of some arbitrary size, if we deem that size sufficient for identifying the subject, we can consistently draw triangular sub-regions across the image. Its consistent manner allows us to easily reconstruct the image given only the color of the sub-region. By recoloring the sub-regions as part of the triangulation process, we eliminate that entire group of pixels in favor of a single, consistent color (represented by 1 or 3 channels). As such, we can replace nearly all information each region contributes when we write to our custom file format (.cs455). This is highly useful

for higher resolution photos with sub-regions like figure 14 where the varying levels of green do not contribute much as opposed to a single chosen green.

If any work were to continue from this point, a combination of both Delaunay triangulation on selected points of interest on an image and compression would be most likely. This would lead on to improvements or new developments in image compression based on vector graphics. Additionally, different approaches to Delaunay triangulation could be explored for better triangulation turnouts and time/space complexities which would contribute to the overall effectiveness and savings of the compression. Vector quantization would be another route to explore in which we consider how many bits is truly necessary to represent a pixel's worth of information while maintaining our original goals of a recognizable subject.

References

1. Oyvind, H., & Dahlen, M. (2006). Triangulations and Applications. Berlin: Springer-Verlag.
2. Shi, J., & Tomasi, C. (1994). Good features to track. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94.
3. Harris, C., & Stephens, M. (1988). A Combined Corner and Edge Detector. Procedings of the Alvey Vision Conference 1988.