Generating low-poly abstractions

Crystal J. Qian ’17, David Dobkin (Advisor)

Princeton University, 2016

Abstract

Polygonal renderings approximate graphics and figures for purposes such as 3D modeling and 2D object detection. Techniques in rendering polygonal meshes, if implemented with an interface for artistic manipulation, can vastly improve current “low-poly” art generation techniques. We introduce an algorithmic approach to 2D mesh generation as well as a tool for interaction with resultant meshes for the purpose of creating art.

Figure 1: *Results of our proposed algorithm on a selection of images.*

1. Introduction

Polygonal approximations are used in 3D modelling to increase frame rates and in computer vision for object detection; in the field of art, “low-poly” image renderings have become a popular form of abstraction.

Tools exist for alternate forms of image representation; popular software includes Adobe Photoshop for pixel-based representations and Adobe Illustrator for vector-based representations. However, there are few effective tools for artists to work with polygonal representations and quickly prototype "low-poly" renderings. We introduce the following algorithm and software as a proof of concept for such a tool.

2. Related Work

Manual approaches: The widely-accepted industry standard for low-poly art generation is a combination of Adobe Suite tools (Photoshop, Illustrator).

Artists using this method triangulate a reference image by hand in Photoshop, then use pen tools in Illustrator to construct a mesh from the triangulation. The mesh is colored by using the eyedropper tool to manually select colors at the approximate center of each face.

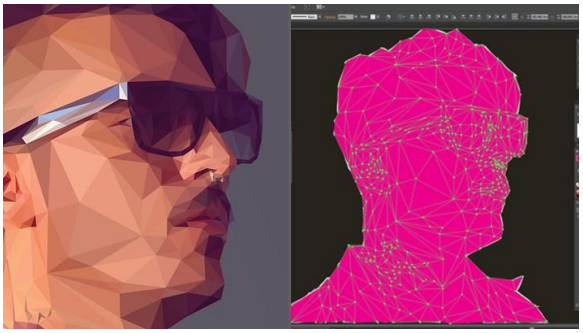

Figure 2: *A step in B. Bitencourt’s tutorial on manual low-poly rendering.*

However, this process is extremely time-consuming. In fig. 2, the artist spent over four hours prototyping one image. Although Bitencourt emphasizes the importance of creating low-poly art by hand as "[the] brain is better than any script or automated process at determining the contours of the face," [1] we suggest that a well-designed system that effectively automates triangulations and allows for artistic direction will vastly speed up and improve the generative process.

Automated approaches: There exists a program, “Triangulation Image Generator,” that generate static (non-editable) triangulations from images. While resultant meshes can be visually pleasing for certain input images, the lack of user control over the triangulations makes the program ineffective as an editing tool. Snorpey’s “Triangulation,” [15] allows for user-controlled parametrization, but doesn’t allow for manual editing.

Figure 3*: A resultant mesh. [2]*

DMesh, a dynamic mesh editor, has the critical feature of allowing manual point editing on existing meshes.

However, the computer vision-based auto point generation algorithm could be improved.

***Figure 4**: A demonstration of dynamic mesh density control in D. Yun's "DMesh" program. [3]*

Research: Current approaches in rendering low-poly meshes for the sake of art have not applied techniques from existing research on constructing meshes out of images for the purpose of object detection and data acquisition [4, 5]; we combine these techniques with methods in adaptive image approximation [6] and edge detection [7, 8, 9] to generate meshes from input images.

3. Problem/Approach

Existing software for artists to create geometric abstractions are too time-consuming and labor intensive. Although user control is critical for creating pleasing aesthetic output, developing an effective image triangulation algorithm would greatly speed up and improve the mesh prototyping process. While programs have been written to solve the image triangulation problem, the lack of user control or weakness of the triangulation algorithm renders them unsuccessful as a means to create visually pleasing meshes.

Our goal is to build a polygonal representation editor tool, which should render aesthetically-pleasing polygonal abstractions, allow for users to refine parameters used in algorithmic low-poly generation, and permit manual editing of resultant triangulations.

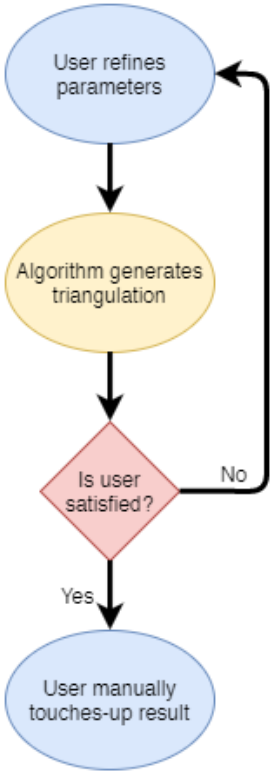

***Figure 5*: Expected user-interaction with our proposed tool.**

To create such a tool, we need to develop an algorithm for parameterized low-poly triangulation (which should be an improvement on existing algorithms), and produce an interface for manual interaction.

4. Implementation

Parametrization: The following are user-defined parameters used in our algorithm.

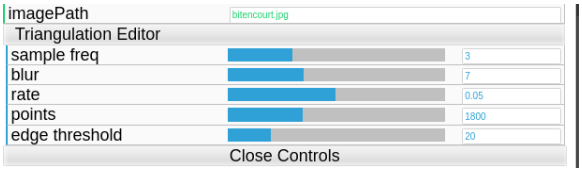

***Figure 6**: A screenshot of our interface for parameter adjustment.*

The user can adjust the values either by using the GUI provided or through a "batch mode" through the webpage URL. The interface portion was inspired by another project with parameterized image filtering. [15]

- **Image path**: the location of the image within the directory
- **Sample frequency** (σ): a value used in our initial edge convolution.
- **Blur** (b): Preprocessed blur value.
- **Rate** (r): Percentage of vertices/total pixel count we want in the output.
- **Points** (n): A cap on the maximum number of vertices desired.
- **Edge threshold** (E): Minimal strength of potential vertices.

Algorithm: Our algorithm must yield aesthetically pleasing approximations. That is, the mesh should preserve key points and edges while keeping redundancies (clustering around key points) at a minimum.

Also, users should be able to interact with parameters of this algorithm in real-time, so rendering meshes from images should be almost instantaneous.

Our current algorithm follows this sequence:

1. Image preprocessing
2. Node detection
3. Triangulation
4. Rendering

Figure 7: *An initial case.*

Preprocessing: Rather than determine the triangulation by sampling points from the base image, we sample points from a "preprocessed" image, where redundancies are removed and edges are highlighted.

"Detection of edges in an image is a very important step towards understanding image features. Edges consist of meaningful features and contain significant information. It… filters out information that may be regarded as less relevant, thus preserving the important structural properties of an image. Most images contain some amount of redundancies that can sometimes be removed when edges are detected…." [10]

We want to preserve significant information in our abstraction, so we filter the image to clearly detect edges. We implement a liberal interpretation of Canny edge detection as we hope to preserve good detection, good localization, and minimal response. [9] However, this implementation must perform faster than Canny edge detection.

First, we use grayscale data to more correctly gauge values of each pixel, as "edges typically correspond to points in the image where the gray value changes significantly from one pixel to the next." [7] For each pixel with RGB values (r, g, b), we get the luminosity l and set the new RBG to (l, l, l).

$$l = .21r + .72g + .07b$$

Figure 8: *Image data after grayscale effect.*

We blur the image with a box-blur implementation whose speed is independent of radius (user-defined parameter b). We get linear-traversal, which is must faster than rendering with the Gaussian blur proposed in Canny edge detection.

We calculate horizontal blur $h_{i,j}$ for each pixel at i, j by sampling pixel values within b, where $f_{i,x}$ is the pixel value at that point.

$$h_{i,j} = \sum_{x=i-b}^{i+b} \frac{f_{i,x}}{2b}$$

These horizontal blurs are interpolated with values of vertical neighbors to get the resultant total blur, $t_{i,j}$.

$$t_{i,j} = \sum_{y=j-r}^{j+r} \frac{h_{i,j}}{2r}$$

***Figure 9**: Blurred convolution with b=3.*

***Figure 10**: Blurred convolution with b=10.*

We then convolute each pixel with a filter to perform a localized edge detection. The result yields a pixel value $e_{i,j}$ for each pixel at i, j.

$$\begin{bmatrix} +1 & +1 & +1 \\ +1 & -8 & +1 \\ +1 & +1 & +1 \end{bmatrix}$$

It is worth exploring the effects of various edge detection operators on the resultant triangulation. However, since we continue filtering the image and ultimately remove edge representations anyway to render an abstraction, we only need a rough approximation. This technique is fast and simple.

We allow for varying degrees of non-maximum suppression (dependent on user-defined parameter σ) to suppress gradient values under σ and thin/filter our detected edges to a desired result.

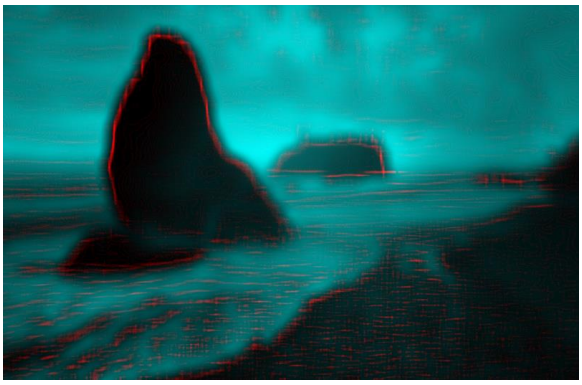

***Figure 11*: Edge detection with $\sigma = 2$**

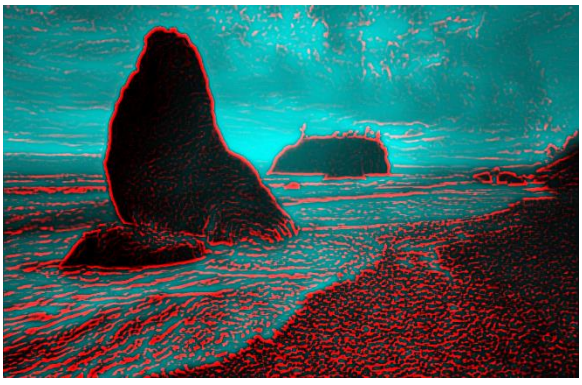

***Figure 12*: Edge detection with $\sigma = 10$**

Node detection: We perform our node detection on the resultant data after preprocessing to get nodes to triangulate.

First, we calculate the edge density for each pixel of the image by sampling a 3x3 window with our desired pixel at the center.

Because the image data itself has been edge detected, the edge density *d* for each pixel at (*i*, j) is simply the average of the pixel values in each window. [12]

$$d_{i,j} = \frac{1}{9} \sum_{x=i-1}^{i+1} \sum_{y=j-1}^{y=j+1} e_{i,j}$$

We check if $d_{i,j} > E$, the user-defined edge threshold parameter. If so, we add the pixel to a list of candidate points L.

Ultimately, we desire to have no more than p nodes in our triangulation, where

$$p = \min(n, r \times w \times h)$$

n and r are the user-defined parameters for points and rate, and w and h are the width and height of the image, respectively.

We randomly select p nodes out of our candidate points list L for our node list.

Triangulation: We define a triangulation T of our image domain Ω as a finite set $\{T\}_{T \in \mathrm{T}}$ of closed triangles $T \in \mathbb{R}^2$ such that the union of triangles in T covers Ω entirely.

$$\Omega = \bigcup_{T \in \mathrm{T}} T$$

Also, the intersection of the interiors of any two distinct triangles $T, T' \in \mathrm{T}$ must be empty. [6]

For $T \neq T'$,

$$\dot{T} \cap \dot{T}' = \emptyset$$

Because we want our triangulation to be made out of individual polygons, we desire our triangulation to be *conforming*; which holds "if any pair of two distinct triangles in T intersect at most at one common vertex or along one common edge." [6]

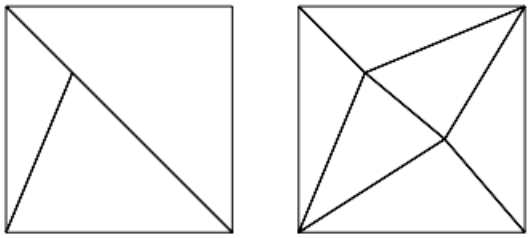

Figure 13: *a non-conforming triangulation (left); a conforming triangulation (right).* [6]

A Delaunay triangulation satisfies this property and avoids constructing long, thin triangles which would be less aesthetically pleasing.

So, we render a Delaunay triangulation D [14] of the nodes previously selected, where D is a conforming triangulation of Ω such that for any triangle in D, its circumcircle does not contain any vertex from D in its interior. [6]

Rendering: We use d3.js [13] and JavaScript for displaying resultant meshes in browser.

d3.js has built-in methods for user-interaction with nodes and links, allowing for simple manual mesh editing.

Specifically, we allow the user to be able to add/remove/drag around vertices. When a vertex is dragged outside the boundaries of its neighboring triangles, the vertices are triangulated once again.

Because we need to quickly and dynamically compute the color of each triangle with user interactions, we sample each triangle at its center of gravity rather than average all pixel values to determine triangle color.

For a triangle with vertices a, b, c, the center of gravity x, y is located at

$$x = \frac{a_x + b_x + c_x}{3}$$

$$y = \frac{a_x + b_y + c_y}{3}$$

Output: After an image is loaded, preprocessed, and triangulated, the output mesh will look similar to this.

***Figure 14**: Result with n = 1700.*

***Figure 15**: Result with n = 2200.*

Does preprocessing improve the aesthetic of the triangulation? More points were added to the mesh to preserve important features in key structures; this is most likely due to the increased noise in the image as a result of not blurring. Whether this is more or less aesthetically pleasing is subjective.

***Figure 16**: No preprocessing.*

In node detection, we randomly sampled *p* points above the edge threshold. Would taking the *p* points with the highest edge densities yield a more beautiful result?

***Figure 17**: Result with n = 1700; points with highest edge densities selected, rather than random selection of points above a threshold.*

Not necessarily. While the results were largely similar, the "best" point approach resulted in more clustering (not clearly shown in the above image). The random sampling approach yields a much cleaner distribution.

5 Evaluation/Results

While our tool does indeed render polygonal abstractions of images, the metric of whether they are "aesthetically pleasing" is purely subjective.

The following experiment aims to determine 1) if the meshes were aesthetically pleasing to a fair sample size, and 2) if preprocessing improved the aesthetic.

Experiment: I sampled twenty subjects, most with academic backgrounds in visual arts or computer science.

I surveyed these subjects twice about three photos. The first time, images were not preprocessed. The second time, images were preprocessed, and the algorithm was refined to address other complaints in the first trial. These test images were not manually edited.

Each subject was shown the original, then the rendering of each image, and asked to give their initial opinions (positive, negative, neutral) and comments on the quality of the triangulation.

City case:

***Figure 18**: City case.*

We chose this image due to the varying noise.

***Figure 19**: City v. 1.*

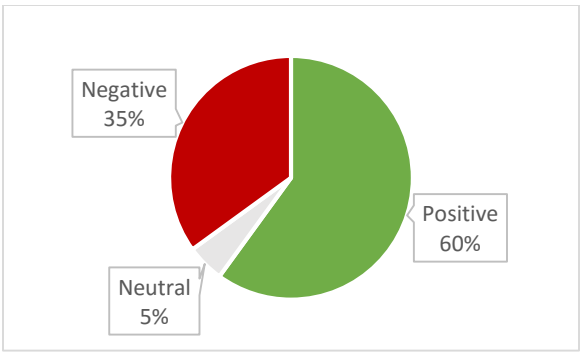

***Figure 20:** Reactions to city, v. 1.*

Reactions to the initial rendering were mostly positive. However, 15% of subjects did not like how small features in the original image (like the water next to the tower) were lost in the abstraction.

***Figure 21:** City v. 2.*

This was the final rendering that was presented to the subjects.

40% of the subjects liked the disparity in the triangle sizes between the city and the sky.

"Selective preservation of hard edges in the original images and the more 'artistic' (less uniform) sizing of the triangles make these better, I think."

– Subject #2

However, 60% of the subjects disliked the larger panels in the background.

"I didn't like the big triangles. I much preferred the finely done sky."

– Subject #5

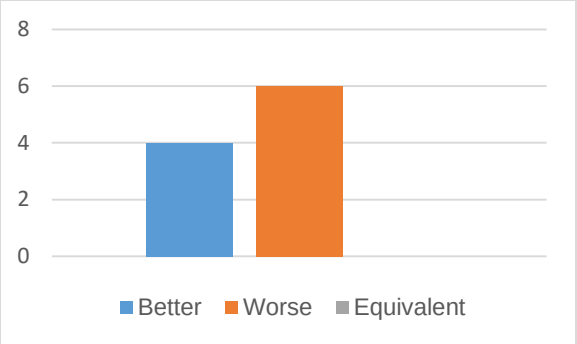

***Figure 22**: How did city v.2 compare to v.1?*

As people preferred v.1 over v.2, we should offer the option of disabling preprocessing for certain images.

Chessboard case:

***Figure 23**: Chessboard case.*

The "chessboard" case was selected due to its distinctive piece shapes and strong edges.

***Figure 24:** Chessboard v. 1.*

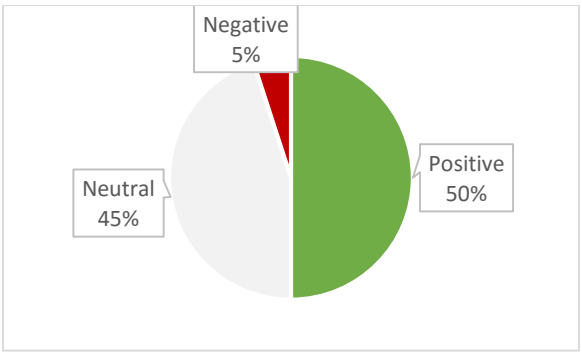

***Figure 25:** Reactions to chessboard, v.1.*

Some liked the jagged edges on the board and the rough abstractions of the pieces.

"I like the jagged lines; it makes the image more interesting, even thought provoking (you expected the grid to be straight lines)."

- Subject #1

Others argued that the jaggedness of the representation was not accurate and therefore displeasing.

"It kind of bothers me that the chess board looks all chipped up. The chess board is supposed to be neat and square."

- Subject #8

30% of responders used a variant of the word "blurry" to describe the effect; this case in particular benefits from edge detection.

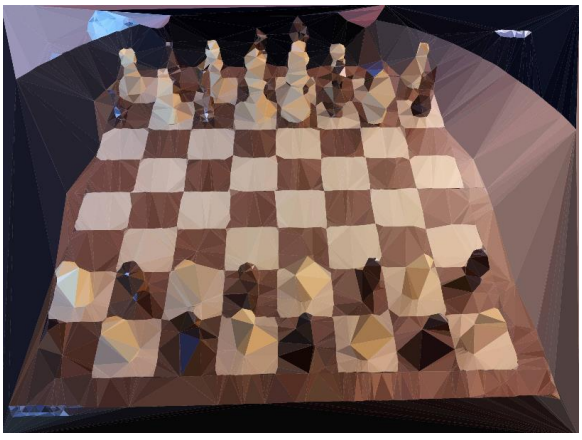

***Figure 24:** Chessboard v. 2.*

Responses to this version were much more positive, with 90% of participants giving the opinion that this chessboard was much better rendered than the original.

"I liked the chessboard because I could make out the details of the individual pieces significantly more."

– Subject #15

"The lines of the chessboard are really clean so it doesn't look like a fuzzy picture anymore, which is awesome!!"

- Subject #4

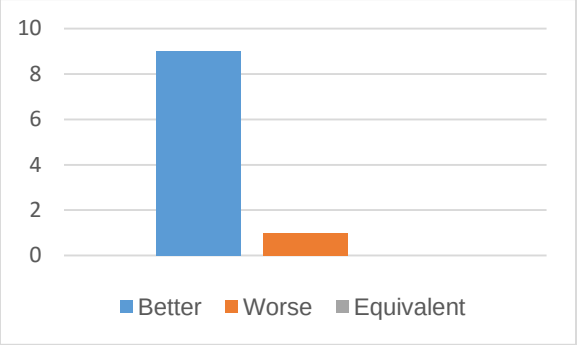

Figure 26: *How did chessboard v.2 compare to v.1?*

The preprocessing and edge convolution made the chessboard case more aesthetically successful, according to the sample.

Portrait case: The portrait was chosen as subjects tend to respond more strongly to human features. Also, the portrait was a bit blurry.

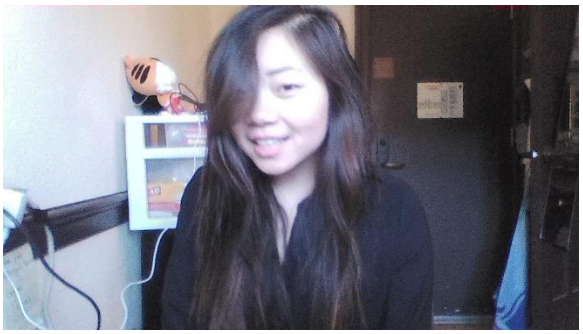

Figure 27*: Portrait case.*

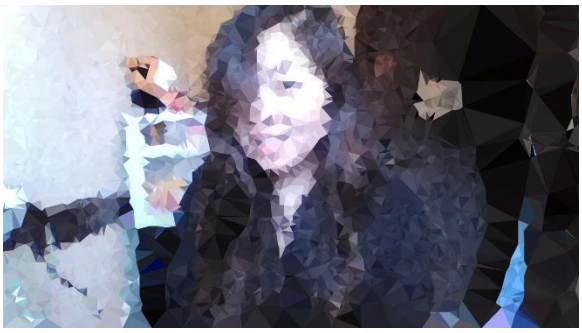

Figure 28: *Portrait v. 1*

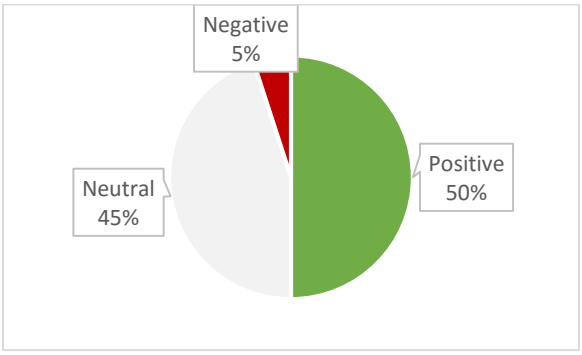

Figure 29: *Reactions to portrait, v1.*

Many people didn't like the lack of feature detection on the face.

"I can still see that it is human, even [Crystal]. But I, as a person, know the important features on a face are around the lips, nose, and eyes. I think there's not enough detail around there, and a lot on the hair. etc."

- Subject #2

Items in the background (shelves, stuffed animal) were abstracted beyond recognition in the triangulations. Some people responded positively to this abstraction.

"The stuffed animal/mascot to the left and the shelves it sits on become pleasantly abstract. Like, you see an orange thing and have to guess at what it could be. The items off to the sides just add interest and fill out the background, but knowing what they are isn't critical to the appreciation of the image."

- Subject #17

Others didn't like this abstraction and recommended removal of the portions altogether from the image.

"The tiger in the back looks like a hand... and so does the sign on your door.... Rendering makes things unrecognizable. I feel like every blob should represent something if it's going to end up in the final [picture], or else it's kind of confusing. "

- Subject #6

"The background is busy, so stuff isn't really recognizable as triangles. It could be less blurred, or you could take a picture with a simpler background."

- Subject #2

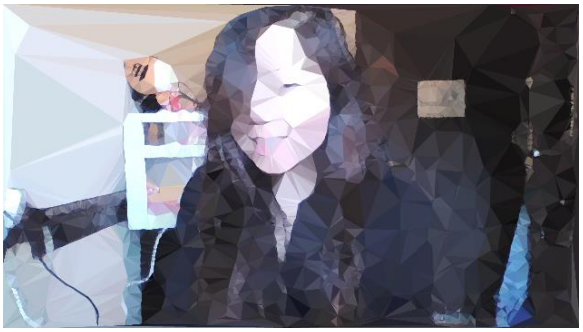

Figure 30: Chessboard, v.2.

Most subjects liked how the new approach to handling the human face.

However, some did not like that abstract objects in the background were no longer left to the imagination. Now that the items had a form, the background was more distracting.

"While the face is slightly more distorted, the background is more distracting in this version."

- Subject #3

"[I don't like this as much], because the door sign on the old [portrait case] looks like it could have been a cat perched on the back of your chair. [It's less ambiguous now.]"

- Subject #14

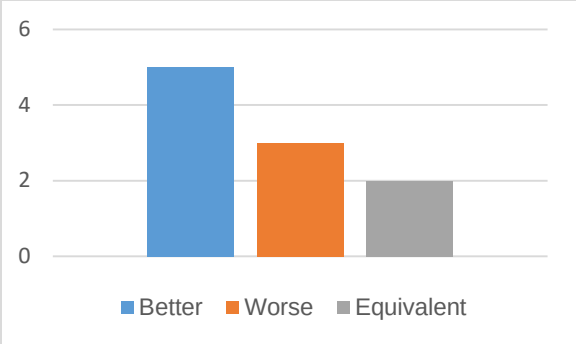

Figure 31*: How did portrait v. 2 compare to v.1?*

Overall feedback: Most subjects responded positively to the raw triangulations.

"The macroscopic structures are well-maintained."

- Subject #19

"The triangulated image gives a good impression of the larger shapes in the image as a whole, but the finer details are lost."

- Subject #2

On the comparison of the two versions:

"If I look back at the previous versions now they look like maybe they were just blurred with a strange triangle shaped blurring, but this wave gives me the impression that the source images have been "reinterpreted" more than just blurred."

- Subject # 15

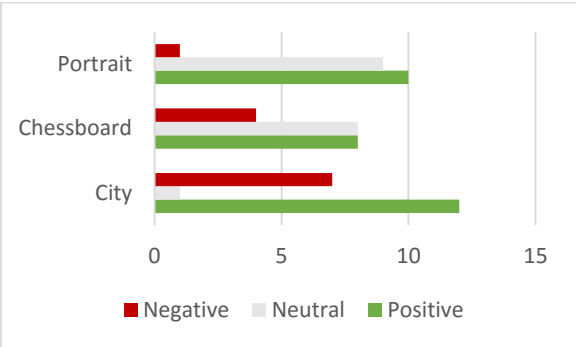

Figure 32: *Overall results of our experiment. Reactions were mostly positive.*

The survey yielded two conclusions:

1) While preprocessing most often improves the overall aesthetic of the rendering, blurring and triangulation along edges doesn't always yield the most aesthetic result.

2) Overall, our algorithm by itself renders generally aesthetically pleasing results (as determined by popular consensus during our sampling.)

Comparative Evaluation: We also wanted to compare results to existing tools previously mentioned.

We compare across the Photoshop standard. Our algorithm used more points and didn't replace the background, but generated a fast prototype in less than .04% of the time it took Bitencourt to render his abstraction. With the user's ability to delete nodes on the rendered mesh, reducing details in desired areas should be an easy task. Additionally, our rendering didn't blank out the background. This shouldn't be too difficult to do manually.

Figure 31: *Manual rendering.*

Figure 32: *Our result.*

To test across algorithmic standards, I plugged a few images into A. Hamamuro's "*Triangulation Image Generator*"**.**

While many cases looked comparatively similar (as his approach also performs some kind of preprocessing before triangulation), some cases stood out that highlighted the importance of allowing for user input. Again, no manual editing of the mesh was done to our output, but parameters were determined by the author.

Figure 33*: Parents case.*

Figure 34: *"Triangulation Generator" output.*

Figure 35*: Our output.*

Reflection: Our tool allows for much faster prototyping of polygonal meshes than existing tools. Additionally, we improved upon a current preprocessing and triangulation algorithm [2, Fig. 34, Fig. 35]; while this is a subjective statement, I believe that our algorithm approximates fine details much better. A survey could be done with the results of the two renderings to confirm this.

Algorithmic results and limitations: We now show some outputs of the algorithm (with parameters specified by the author) and discuss the outputs' aesthetic qualities and limitations.

Figure 36: *IBM case.*

Figure 37*: Our result.*

The two figures looking into the data center are still prominent, despite their shapes having similar luminosity and shape to the objects inside the building. Also, the letters in IBM are clearly displayed.

Figure 38: *Paris case.*

Figure 39: *Our result.*

This photo is similar to the city case from our experiment. Sections behind the tower maintained (the city, buildings in fog, and sky), but the buildings become undecipherable and the clouds in our rendering could easily be mountains. The rounded edges of the clouds lose their distinctive shape; perhaps we should not have preprocessed to edge detect in this case. However, the tower is very well shown, perhaps due to the nice contrast with the fog.

The next photo, *Reflection*, is already a low poly rendering. As expected, our implementation works well on images with clear edges already defined. The lack of noise eliminates strange shapes, and our result looks almost the same as the original.

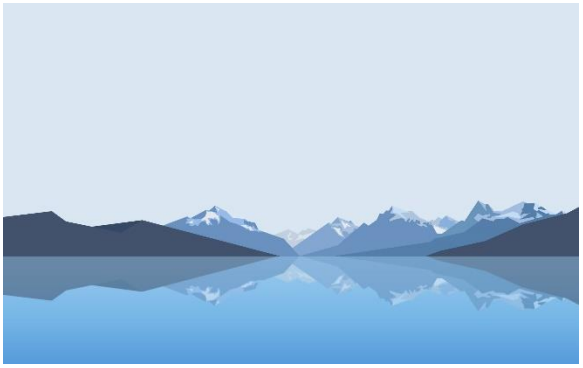

Figure 40: *Mountain case.*

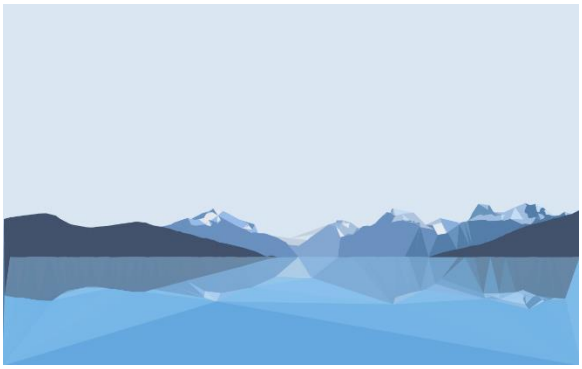

Figure 41: *Our result.*

Figure 42: *New York City case.*

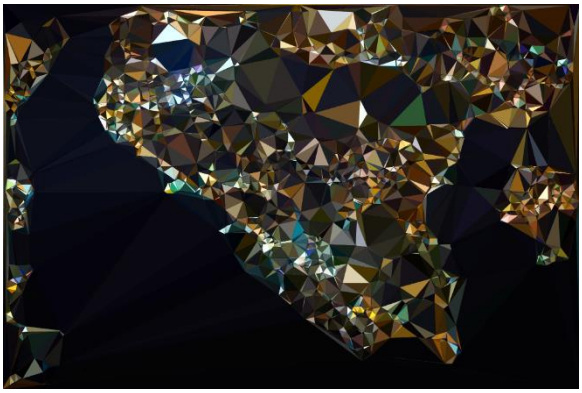

Figure 43*: Our result.*

This photo is unlike the previous in that there is a lot of noise, particularly on the island. Our implementation does not resolve these lights well; or, we'd need more polygons.

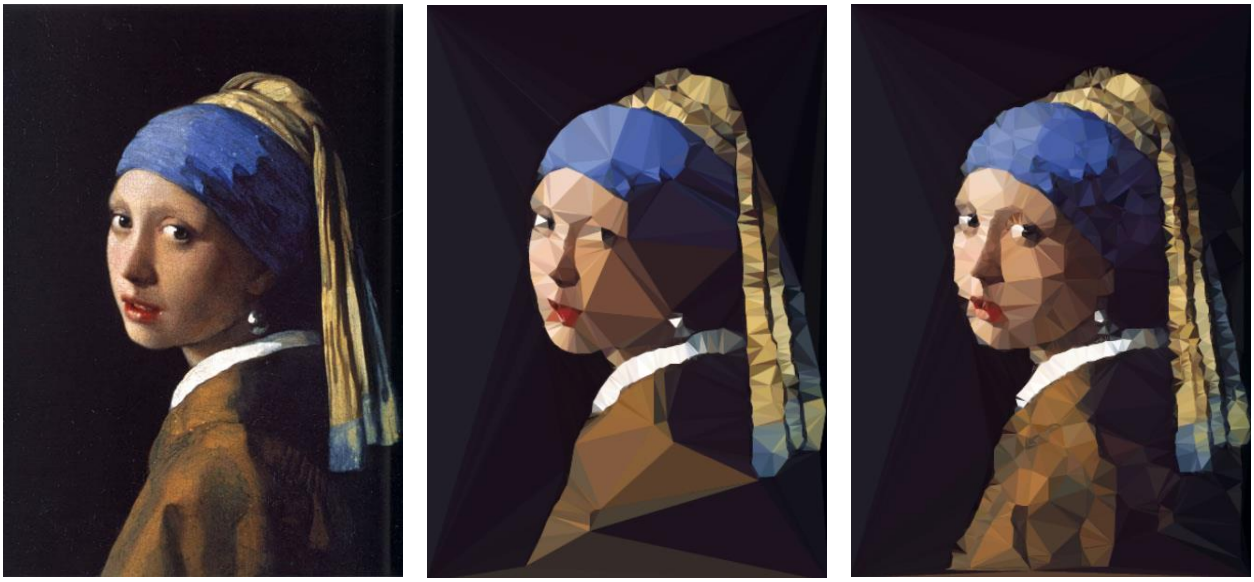

Figure 44: *Girl with the Pearl Earring, Vermeer (left), our result with n = 1200 (center), our result with n = 2000 (right).*

Our result renders the features in the face well; the nose, lips, eyes, and facial structure are distinct. This benefits from the large contrasts in the painting and the clear lines. With the portrait in our test case, perhaps the blurred photo made edge detection weak. In my interpretation, the different triangulations from varying point values changes her expression in each rendering.

Assessment: Overall, the algorithm preserves features in images well, particularly when edges are clearly defined or areas have high contrast. However, areas with lots of noise will lose distinctive features if we use few points in the detection. Whether this is a limitation is subjective.

6 Conclusions

To recap, we developed an algorithm for generating low-poly abstractions from images. The generated abstractions are, overall, regarded as visually pleasing.

We also built a tool that incorporates the fast prototyping of algorithm-based meshes with the aesthetic refinement of the user.

Thus far, most tools for creating art depend entirely on the artist's discretion. If rendering an abstraction of an image, the artist must subjectively determine which parts of the image are important, and which are not.

The magnitude of labor behind creating low-poly art is commendable. When we see a low-poly rendering, we recognize not only the aesthetic structure of the graphic, but also the labor of the artist, who had to render each individual triangle.

With our contribution, such labor has been greatly reduced. This tool encourages the idea of the user and computer working together to generate art. The computer in this case is not just a tool, but also a collaborator.

Additionally, by simplifying the process of generation, we hope to reduce the barriers of entry to the field of low-poly art, provide an easier approach to a challenging and laborious process, and encourage more artists to explore polygonal representation as a basis for aesthetic ingenuity.

7 Future work

The algorithmic approach is adequate for most cases, but struggles with maintaining structures in areas with loud noise (for example, the bottom part of the city example or the *New York City* rendering). Currently, we have to make a tradeoff between allowing very dense meshes in those areas or risking an unrecognizable rendering. This is not necessarily a limitation, but is something worth investigating in more detail. If a graphic designer was given the *New York* photo or any photo with much noise (for example, snow caps or grass), how would he/she produce a low-poly rendering? While our tool does allow an artist to produce an aesthetically-pleasing result with any image, in the worst case completely with manual labor, studying this specific case and adjusting the algorithm to account for loud noise could make our tool even better.

Additionally, there are still features to be implemented in order to create a robust polygonal mesh-editing tool; a feature that could address feedback in the survey is a "layered" feature. Like "layer" features in other editing systems, users could select regions of the mesh to refine or edit rather than have our algorithm be applied to the entire mesh on each iteration. This would allow the user to, for example, throw the background of the *City* image into a layer and increase the number of polygons in that region if he/she did not find that particular section visually pleasing.

We've begun experimenting with wrapping meshes around surfaces and bringing 2D meshes into a 3D space, to further explore the aesthetic of "low-poly" art in a space that would be even more difficult to explore manually.

Figure 45: *Wrapping a planar mesh around a cylinder*

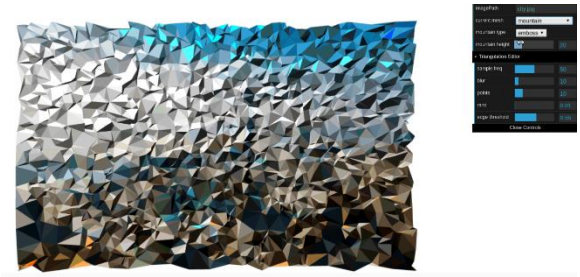

Figure 46: *Crumpling a planar mesh*

We hope that continuing this work in the space of geometric abstraction will yield more visually interesting results, and that this tool will allow more art enthusiasts to join us in this exploration, which is why we're producing a version to release.

8 Honor Statement

I pledge my honor that this paper represents my own work in accordance with University regulations.

Crystal Qian, 4/29/2016

References

Images (and citations), renderings, and supplemental renderings can be viewed in greater detail here: http://imgur.com/a/gMAWZ

[1] B. Bitencourt. (2014) Low-poly self portrait tutorial. Available: https://www.behance.net/gallery/16579635/ low-poly-self-portrait-tutorial

[2] A. Hamamuro. (2012) Triangulation image generator. Available: http://jsdo.it/akm2/xoYx

[3] D. Y. Yun. (2012) Dmesh. Available: dmesh.thedofl.com

[4] G. P. Fickel et al., "Stereo matching and view interpolation based on image domain triangulation," in IEEE Transactions on Image Processing, Vo. 22, No. 9, 2013.

[5] Y. Tao and W. I. Grosky, "Delaunay triangulation for image object indexing: a novel method for shape representation," in In Proceedings of the Seventh Spie Symposium on Storage and Retreival for Image and Video Databases, 1999.

[6] L. Demaret and A. Iske, "Anisotropic triangulation methods in adaptive image approximation," in Approximation Algorithms for Complex Systems, 2010.

[7] O. Vincent and O. Folorunso, "A descriptive algorithm for sobel image edge detection," in Proceedings of Informing Science and IT Education Conference, 2009.

[8] T. P. Sahu and Y. K. Jain, "Improved simplified novel method for edge detection in grayscale images using adaptive thresholding," in Journal of Advances in Computer Networks, Vol. 3, No. 2, June 2015.

[9] R. Maini and D. H. Aggarwal, "Study and comparison of various image edge detection techniques," in International Journal of Image Processing (IJIP), March 2009.

[10] D. Ray, "Edge detection in digital image processing," Rice University, 2013.

[11] M. Klingemann. (2016) Superfast Blur Algorithm, Available: http://incubator.quasimondo.com/processing/superfast_blur.php

[12] S. L. Phung and A. Bouzerdoum, "Detecting People in Images: An Edge Density Approach" in IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2007.

[13] M. Bostock. (2013) Data Driven Documents, Available: https://github.com/mbostock/d3

[14] J. T. L. (2014) Fast Delaunay Triangulation in Javascript, Available: https://github.com/ironwallaby/delaunay

[15] G. Fischer. (2013) Triangulation, Available: https://github.com/snorpey/triangulation