CSCI 1430 Final Project Report: Generating Low-Poly Representations of Images Using K-Means Clustering and Edge Detection

Polygonal Non-Generosity: Yifan Ruan and David Young
Brown University

Abstract

Low poly is a popular term in the domains of computer animation, modeling, etc. However, low poly, when used to refer to 2D images, also has its applications, especially to artists who specialize in the low poly aesthetic. Low-poly representations of images are often generated by first applying some type of image segmentation followed by triangulation/polygonization and shading. In our project, we explore the use of k-means clustering to segment an image before triangulation.

1. Introduction

In our project, we try to generate low-poly representations of images with modest computational resources while minimizing the amount of information lost from the original image.

One approach for generating low-poly representations with limited compute is edge detection followed by triangulation on points selected from the edges. However, edge detection only encodes information about the magnitude of difference in color between areas of the image.

We expand on this method by encoding information about the position in the image, the specific color values, and the intensity for each pixel. Using k-means clustering on these features, we hope to achieve better semantic encoding prior to triangulation, thereby minimizing information loss.

If our method is successful, it will allow everyone–especially low-poly artists who may lack the resources to utilize compute-intensive deep-learning approaches to image segmentation–access to a tool to generate low-poly mockups of images that better represent the original image compared to mockups generated by simple edge detection followed by triangulation.

2. Related Work

2.1. Citations

Our work was inspired by previous work done by Crystal Qian while she was a student at Princeton University [2] and by a paper by Zhang et al. [3].

Both of these polygonization implementations were examples of edge detection followed by a Delaunay triangulation. We extended upon these implementations by using k-means clustering to encode additional information about the original image prior to triangulation.

We briefly reference EfficientPS [1], a state of the art in panoptic segmentation, when discussing societal implications.

2.2. Software used

All of the code for the project was written in Python. Notable libraries used:

- **NumPy** – array manipulation
- **scikit-learn** – clustering
- **scikit-image** – image manipulation
- **matplotlib** – visualization
- **SciPy** – Delaunay triangulation

3. Method

Information loss is unavoidable when converting an image to a low-poly representation. Our goal is to minimize the amount of information loss through k-means clustering with custom features.

3.1. Feature extraction

The first step in our polygonization procedure was the extraction of features from the image. We chose to use the following features for each pixel:

1. x position of pixel

2. y position of pixel
3. Value of first color channel
4. Value of second color channel
5. Value of third color channel
6. Intensity of pixel

We encode this information in a vector of length 6 for each pixel in the image. We weight color, intensity, and position in our encoding scheme with scale factors. These scale factors drastically affect the quality of the clustering. For the majority of the figures in this report, we used a color weight of 1.5, an intensity weight of 1, and a position weight of 2.

3.2. Clustering

We then perform k-means clustering on these encoded features. More specifically, we use mini-batch k-means clustering from the scikit-learn Python library. We chose to use mini-batch k-means clustering rather than normal k-means clustering because the former drastically reduces runtimes while not causing any noticeable decrease in cluster quality.

After clustering, we reshape into an image segmented by color using the calculated cluster centers and labels. Before triangulation, we find a subset of pixels to use as vertices for the triangulation. This step can be divided into three steps.

First, we perform edge detection on the segmented image by applying a Sobel filter. We then take 1% of the points from the edges and add them to a set of points to triangulate. Finally, we add evenly spaced points along each border of the image to ensure the triangulation covers as much of the image as possible.

3.3. Triangulation

We then perform triangulation on the segmented image. We use a Delaunay triangulation for the triangulation task, as it is both efficient for large amounts of points (linearithmic runtime) and it has the desirable characteristic of maximizing the minimum angle of triangles. This second characteristic means that it avoids triangles with extremely acute angles (sliver triangles), which would not be visually appealing in most low-poly mockups. We use the scipy library function `scipy.spatial.Delaunay` for the triangulation.

3.4. Visualization

We visualize the results of the triangulation by shading in each triangle with the average color value across all contained pixels.

Image Dimensions	Pixels	Runtime (s)
400×400	160000	0.329
800×600	480000	0.668
1024×768	786432	0.935
1920×1080	2073600	2.391
3840×2160	8294400	8.717

Table 1: Average runtime of polygonization procedure in seconds across ten runs for images of different sizes. Tests performed on a desktop computer with an Intel i9-10900k chip without GPU acceleration.

4. Results

While it is difficult to objectively compare the visual appeal of images, our triangulation scheme managed to produce low-poly representations that were able to capture even relatively small features in images (Figure 4), showing that a significant amount of information from the original image is preserved in the polygonization.

Additionally, our polygonization procedure runs in a reasonable amount of time on a personal computer (Table 1), making it accessible to the general public and people who do not have access to significant computational resources. For smaller image sizes, our procedure nears real-time performance. For images up to 4k, the average runtime in our tests was under ten seconds, which is more than sufficient for generating mockups.

4.1. Effect of number of clusters on quality

We found that the number of clusters has a substantial effect on the quality of the low-poly representation produced. The number of clusters needed depends on the diversity of color in the image. For most natural images we tested, we found that our default of ten clusters was sufficient to represent most details in the image. For images where fine detail must be preserved, such as pictures of people, it may be necessary to increase this parameter in order to minimize information loss.

4.2. Effect of number of vertices on quality

Similar to the number of clusters, we found that the number of vertices necessary for achieving a good low-poly representation scales with the level of detail in the picture. Our default value is 1% of the pixels in the result of applying a Sobel filter to our segmented image. We found that increasing this value is helpful when dealing with low-resolution images. This is because the lower resolution images will have fewer total feature points. Perhaps an approach we could implement in the future is to offer the option of using a max number of vertices instead of a percentage.

4.3. Comparison to similar implementations

Compared with Qian's paper [2], which implemented a similar idea, ours preserves more information about the different objects in the image (Figure 4). The roads and individual buildings preserve their shape better in ours than in theirs, where it is harder to make out the original shapes. We believe this is because our feature points are selected from the result of our kmeans step, which will create clusters along the original shapes, like buildings and roads. This results in more feature points along the boundaries of objects which creates triangles that fall between two boundaries better. You can see this again in the Manhattan image, where our result preserves some structure while theirs looks more cluttered. They mentioned this in their future work section, so we are happy that we were able to (in our opinion) improve upon that.

The tradeoff for doing this is that our approach will lose detail within clusters that our kmeans algorithm finds. This is most apparent on images of easily recognizable shapes with complex "insides" such as human faces, since the kmeans algorithm will remove more detail than simply blurring since we don't select feature points from within clusters.

Another benefit we have is more tunable parameters: number of clusters, percent of feature points to use, and perhaps even weights for the kmeans algorithm. This offers more flexibility to tackle a wider range of images. For instance, more complex images generally look better with a higher number of clusters. Higher number of vertices will preserve much more detail than lower, and looks quite different stylistically, as demonstrated in the example with the sunset. This also allows us to tackle more complex pictures such as the one of Manhattan by increasing the number of clusters and vertices.

The natural tradeoff for this robustness and flexibility is that our default parameters will not be as consistent on various types of images

4.4. Societal Discussion

Please respond to the following questions. Different projects will have different scales and qualities of impact; we ask you to think creatively and consider the broader implications of your project rather than just the more narrow current technical capability. Responses should together take up roughly one page in your final report.

1. Describe the socio-historical context of your project to identify three broad societal factors that could affect your data, goal, and/or hypothesis. These factors might include current or historical policies, events, social conditions, and larger societal systems. Cite at least one outside source.

   (a) If the low-poly aesthetic goes out of fashion in the future, our project would certainly lose some of its appeal. On the other hand, if low-poly media suddenly gains popularity, there may be more incentive to improve the performance of our algorithm. Additionally, if people decide they want to use something like our algorithm to create art for a game or animation, we would have more factors to think about beyond runtime and aesthetic. The target input images would change depending on what input images a potential creator would want to use. Would it be hand-drawn, rendered, or be a real photograph? Would they primarily be interested in characters sprites, landscapes and backgrounds, or items? We would need to test the algorithm on more types of inputs and potentially find which set of hyperparameters works well on each. Another potential future use is to perform polygonization on a video in real time. Our current algorithm would not be feasible, and we'd need to make optimizations such as preserving cluster information between frames and perhaps improving our triangulation speed.

   (b) One major goal in our project was to create a tool that is accessible to the average person. To that end, we chose to use a more naive algorithm like k-means over more advanced panoptic segmentation techniques that would undoubtedly yield a better result [1]. If, in the future, GPU accelerated hardware became the norm, our goal would then be to increase the quality of the low-poly representation even if more computational resources are required. At the end of the day, as long as our procedure is accessible, we will have achieved our goal.

   (c) With the proliferation of smartphones in the past decade, mobile app design has become increasingly relevant and important. The low-poly aesthetic has been heavily adopted by mobile apps in particular. In the context of this boom in mobile app development, our project has increased relevance, as low-poly artwork is quite tedious to create by hand, and app designers would value a product that prioritizes the quality of the low-poly representation produced over the runtime of the procedure.

2. Who are the major stakeholders in this project? What is your relationship to these stakeholders?
   Stakeholders are those who may be affected by or have an effect on your project topic. Some examples of stakeholders are a particular demographic group, residents of a particular geographic area, and people experiencing or at risk for a particular problem.
   Consider the following questions to help identify stake-

Figure 1: A comparison of our polygonization against Qian's
Top left: Original *Top right:* Qian's result *Bottom:* Our result

holders:

- Who does this project topic currently affect?
- Who might be harmed by your research findings?
- Who might benefit from your research findings?

Artists, particularly those working with low-poly aesthetics, are one of the major stakeholders in our project. They stand to benefit from our findings, as our project provides an accessible tool for generating mockups of low-poly art.

At the same time, artists also stand to be harmed by our findings. Generators such as ours for low-poly art can easily be used to modify an artist's work such that it is difficult to recognize, faciliating copyright infringement.

3. Research or journalism on your broader project topic may have already been conducted. What was the societal impact of existing research? Discuss the implication of this research on your project and consider the following questions to help identify at least one implication. It may affect:

    - How you should frame your goal,
    - How you should design your algorithm,
    - How you should analyze your data,
    - How you should interpret your findings, and

Figure 2: A comparison between our result and theirs *Left:* Ours *Right:* Qian's

- How you should present your results.

A framework similar to Qian's was first introduced in 2015 [3]. One implication they raised that we did not think of was that low-poly can be used as a way of compressing images. This makes sense, as the goal in generating good low-poly representations is the same as the goal in compressing images: to encode as much information as possible in a smaller/simpler representation.

To this end, one goal we will focus on is to increase the quality of the semantic encoding while maintaining computational costs that are accessible to the average person.

4. How could an individual or particular community's civil rights or civil liberties (such as privacy) be affected by your project?

   As discussed above, generators such as ours opens up an avenue for malicious actors to trivially modify and pass off artists' work as their own. Thus, our work, while beneficial to artists, has, at the same time, the potential to facilitate copyright infringement against these same artists.

5. If you are using data, what kind of biases might this data contain? Do any of these represent underlying historical or societal biases? How can this bias be mitigated? Consider the following questions to help you:

   - Were the systems and processes used to collect the data biased against any groups?
   - Is the data being used in a manner agreed to by the individuals who provided the data?

   We are not using any datasets for our project. All figures in our project were produced with arbitrarily chosen images from the Internet. As a result, the images we used can have any number of biases inherent in them.

   However, since our polygonization procedure does not attempt true semantic segmentation, the risk of introducing additional bias from polygonization is quite low. That is, our procedure, given a picture of a cityscape, for example, does not actually recognize cars, people, and other objects. Rather, it groups together pixels based on color, position, and intensity.

   One possible source of bias is that our procedure considers color and intensity as main factors in clustering. Thus, given a picture of a person, if the background is too similar in color or intensity to the person's skin tone, the polygonization procedure could fail to distinguish between the two.

   One way to mitigate this bias is to introduce additional features in the encoding. For example, we could use Gabor filters to extract texture information from the original image. This would allow us to better distinguish between the foreground and background without relying so heavily on color and intensity.

5. Conclusion

In our project, we developed an efficient, lightweight framework for polygonizing images that preserves more information than a typical triangulation after edge detection would. This matters because it is an improvement

5.1. Future work

We would like to improve our implementation's performance on faces and recognizable shapes like animals or people. One potential method we would like to explore is to choose points from within clusters and not only on the boundaries between clusters. We believe that this would help reduce the loss of detail from our k-means algorithm.

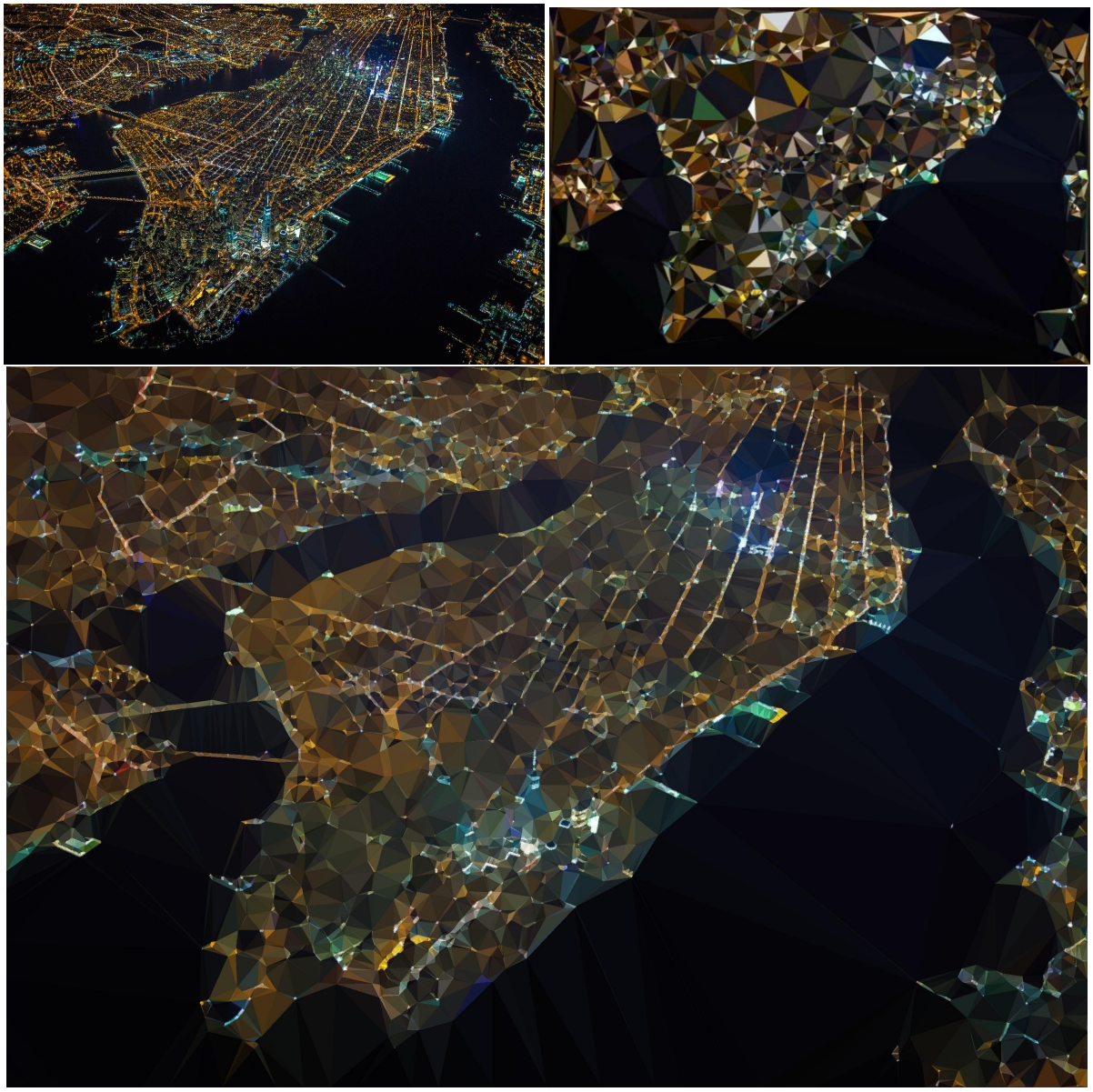

Figure 3: A comparison of our polygonization against Qian's
Top left: Original *Top right:* Qian's result *Bottom:* Our result

Another big area to explore is to improve our k-means algorithm to improve the quality of our clustering segmentation. Adjusting the weights for position, intensity, and color affect the results heavily. Additionally, we could add entirely new parameters such as texture, which would have its own weight in the k-means algorithm.

A triangulation algorithm that could handle adding one cluster at a time could further improve the preservation of shapes in the final results. We ultimately ran out of time to implement this, but found a potentially useful library here.

References

[1] Rohit Mohan and Abhinav Valada. Efficientps: Efficient panoptic segmentation. *CoRR*, abs/2004.02307, 2020. 1, 3

[2] Crystal Qian. Generating low-poly abstractions. 2016. 1, 3

[3] Wenli Zhang, Shuangjiu Xiao, and Xin Shi. Low-poly style image and video processing. In *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 97–100, 2015. 1, 5

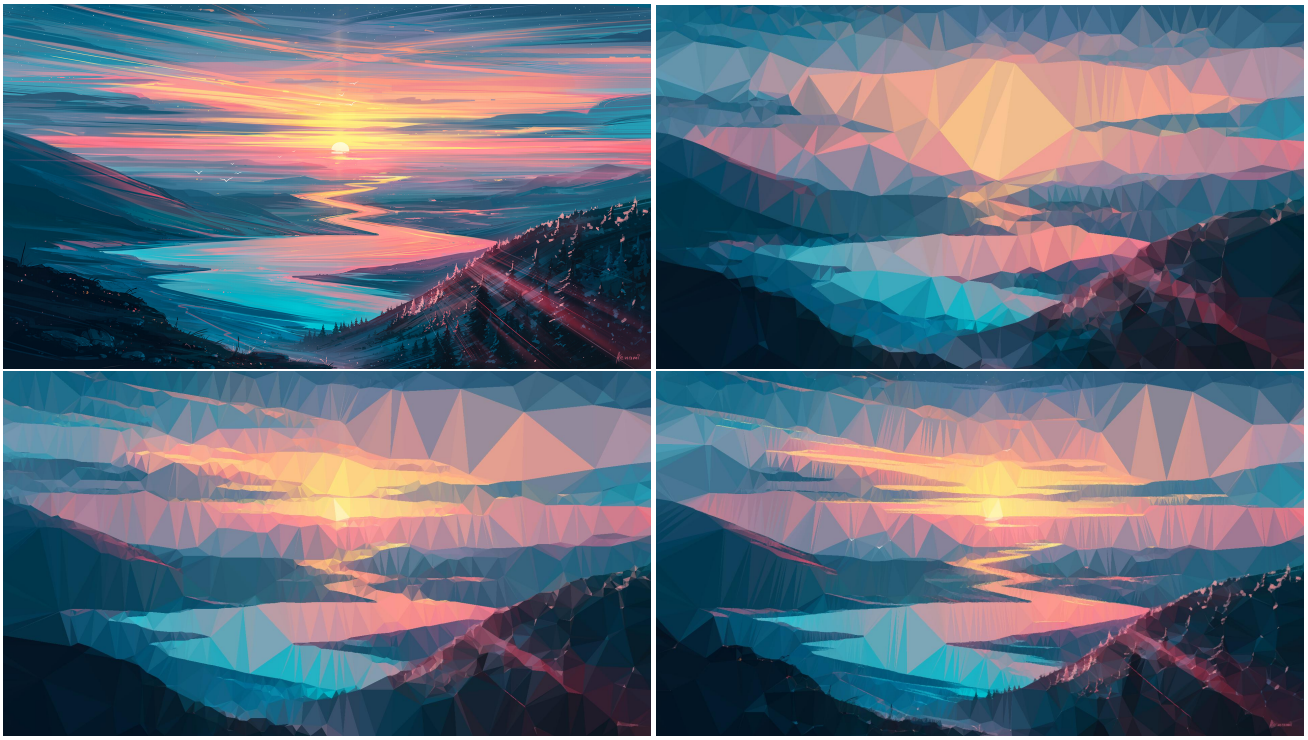

Figure 4: This sunset image showcases how different parameters change the output image. *Top left:* input *Top right:* Default parameters *Bottom left:* Doubled number of clusters and vertices *Bottom right:* doubled number of clusters and 20 times as many vertices used

Appendix

Team contributions

Yifan Ruan Came up with the idea of using KMeans to help produce more meaningful feature points, and contributed the first iteration of code for the KMeans algorithm using the six dimensions. Generated most of the result images that we have in this paper and found the images used in Qian's paper. Wrote code for the final step where we fill in the triangles with the average of the colors within.

David Young Refactored Yifan's clustering code and integrated into a polygonization routine. Handled parsing and other setup in `main.py`. Contributed to report and poster.