Journal for Geometry and Graphics
Volume 21 (2017), No. 1, 131–139.

Low-Poly Image Stylization

Thitiwudh Uasmith, Tantikorn Pukkaman, Peeraya Sripian

Dept. of Media Technology, King Mongkut's University of Technology Thonburi
126 Pracha Uthit Rd., Bang Mod, Thung Khru, Bangkok 10140, Thailand
emails: tdy2020@gmail.com, pukkarmoo@gmail.com, midorip@gmail.com

Abstract. A low-poly image is a minimalist style of art that is currently widely used. It is an image derived from low-polygon 3D objects with an idea of image non-photorealistic abstraction. The trend of using low-poly images has accelerated rapidly since the introduction of vector images. Because low-poly images are based on vectors, the images are compact, scalable, editable, and easy to animate. Although semi-automatic low-poly image conversion tools exist, the resulting graphic generated from the tool does not resemble the original image in terms of its global structure. This paper presents an application that automatically creates a low-poly image that preserves global details, such as edges. Given a raster image, our algorithm automatically computes the polygons that best approximate the image. Our proposed algorithm is mainly based on K-means segmentation, contour extraction, the Ramer–Douglas–Peucker simplification, and the Delaunay triangulation. The output image is in a vector file format that can be easily further manipulated. For evaluation purposes, we distributed questionnaires to 30 participants that were comprised of 30 low-poly image sets; each set contained a low-poly image generated using our method and one generated using another method. From the experiment, we found that the majority of participants preferred the low-poly images generated using our method.

Key Words: image vectorization, image abstraction, image triangulation, low-poly image

MSC 2010: 68U05, 68U10

1. Introduction

As currently seen in modern graphics and artworks, there is a growing preference for a minimalist style. This trend has accelerated rapidly through the years since the introduction of vector images. Vector graphics use polygons to represent images. Since the image is based on vectors, it is compact, scalable, editable, and easy to animate. The number of polygons per area represents the resolution of the graphic and, thus, is a significant factor for performance optimization. Therefore, artists must compensate for the resolution of a 3D scene with rendering time when creating typical real-time 3D games or real-time 3D rendering. In real-time

ISSN 1433-8157/$ 2.50

manipulation of a 3D scene, fewer triangles in polygons can be used to achieve an acceptable frame rate, which is known as low-poly meshes [1]. A low-poly graphic usually appears to be blocky and lacks detail. It is perceived at its global structure because the image can be distinguished into parts, although it cannot be perceived at its local structure since one may not be able to determine what an image part represents.

A low-poly image has been used in many modern computer graphics and press artworks. With its compactness and resolution independency, it can easily be applied in a press artwork or used for multiple resolution screens, even for very large screens.

Although this type of image is perceived as not realistic since the contours are all hard, straight edges, and the image tends to look lifeless and obviously manufactured, it is currently used to achieve a certain retro style concept that is similar to the pixel art used in classic video games. Hence, low-poly art is widely used by artists who are fond of minimalist art or artwork that can be used for multiple purposes. For example, the *Low Polygonal Studies* series of artwork on Breno BITENCOURT's webpage [2] shows a stunning use of low-poly art to represent overall image structure and main colorization. The artist used *Adobe Photoshop* and *Adobe Illustrator* to draw triangles and painted each triangle by hand with a single color that represented the overall structure of the original image. The whole process typically takes at least four hours as stated in the tutorial on the webpage [3]. This paper explores existing low-poly image generation tools and proposes an algorithm to generate a better low-poly image in terms of resemblance to the original image, especially in the edges.

Our proposed algorithm made the following contributions:

- Fully automatic generation of a low-poly vector image in a scalable vector graphic format from an input of a raster-type image.
- The output image is represented in polygons that constitute sets of triangles. Each polygon is filled with color that is interpolated from the original image pixels that lie within the polygon boundary.

2. Related work

This section explores existing low-poly image generation tools that are available for commercial use. The latter part of the section discusses the current image processing algorithm used to generate low-poly images.

2.1. Existing commercial tools

We explored some semi-automatic low-poly image generation tools that are commercially published as smartphone applications: *DMesh* [5] and *Polygen* [8]. Both applications receive input bitmap images from the photo library of a smartphone or from an image taken with a smartphone equipped with a camera; the image is then converted into a low-poly image. The resolution of the output image can be adjusted by increasing or decreasing the number of generated triangles to achieve a particular objective as determined by the user. The increment of triangles yields a more photorealistic low-poly image. However, the resulting low-poly image generated from *DMesh* or *Polygen* displays jagged edges.

Without the adjustment of vertices by the user, the generated results do not seem to preserve the significant features of the original image, especially around the edges. Figure 1 shows an output image from *DMesh*, *Polygen*, and our method.

Figure 1: A low-poly image generated using the *DMesh* application (left), the *Polygen* application (center), and our method (right).

2.2. Related algorithms

K-means clustering

The K-means algorithm [7] is an unsupervised clustering algorithm that classifies the input data points into multiple classes based on their inherent distance from each other. The algorithm assumes that the data features form a vector space and tries to find the natural clustering within them. The points are clustered around the centroids μ_i, $i = 1, \ldots, k$, that are obtained by minimizing the objective as:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} |x_j - \mu_j|^2 , \tag{1}$$

where there are k clusters S_i, $i = 1, \ldots, k$, and μ_i is the centroid or mean point of all the points $x_j \in S_i$.

Delaunay triangulation

In mathematics and computational geometry, a *Delaunay triangulation* of a set P of points in a plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$. Delaunay triangulations maximize the minimum angle of all angles in the triangles of the triangulation and tend to avoid sliver triangles. The triangulation is named after Boris Delaunay for his work from 1934 [6]. The Delaunay triangulation has been researched and developed for a long time. Accordingly, it has been implemented in various ways. We chose the triangle library for *Python* for a quick and exact generation without large-angle triangles [10, 11]

Ramer–Douglas–Peucker algorithm

The *Ramer–Douglas–Peucker algorithm* (RDP) [9] is an algorithm that is used to reduce the number of points in a curve that is approximated by a series of points. The initial form of the algorithm was independently suggested in 1972 by Urs Ramer and in 1973 by David Douglas and Thomas Peucker. The starting curve is an ordered set of points or lines, and the distance dimension $\epsilon > 0$.

The algorithm recursively divides the line. Initially, it includes all the points between the first and last point, and it automatically marks the first and last points to be kept. It then finds the point that is furthest from the line segment, which is obviously furthest on the curve from the approximating line segment between the end points. If the point is closer than ϵ

to the line segment, then points that are not currently marked to be kept can be discarded without the simplified curve being worse than ϵ.

If the distance between the furthest point and the line segment is greater than ϵ from the approximation, then that point must be kept. The algorithm recursively calls itself with the first point and the worst point, and then with the worst point and the last point, which includes marking the worst point being marked as kept.When the recursion is completed, a new output curve can be generated that consists only of those points that have been marked as kept.

Our method uses the contour approximation routine in the *OpenCV library*, which is based on the RDP algorithm [4].

3. Proposed solution

To generate low-poly images without the inaccuracies mentioned in Section 2.1, we prioritized the image edges and corners. These are significant details that can be used for constructing low-poly images that still resemble the original images.

The proposed algorithm begins with image preprocessing to remove excess noise, while preserving image edges. Next, the image is segmented, and the contours are defined and simplified. The simplification process uses the RDP algorithm, while extracting the image edges and corners, which will be used for triangulation. Finally, we filled each triangle with a specific interpolated color.

3.1. Preprocessing

One of the problems with the existing tools, as stated in Section 2.1, is that the output image cannot accurately preserve significant image details, such as edges and corners. Since a low-poly image represents the input images globally, and not locally, high resolution details, such as the texture of sand, fabrics, and wood patterns, should not be presented in the resulting image. Therefore, prior to the segmentation and triangulation of the image, the input image is smoothed by a bilateral filter that preserves the global edges of the image. The results from this preprocessing step is shown in Figure 2.

Figure 2: Original input image (left) and filtered image (right) where $d = 2$, $\sigma_{color} = 40$, $\sigma_{space} = 20$.

Figure 3: One of the clusters of pixels that are similar in color, marked in white (left, $K = 8$) and its simplified topological form (right, $\epsilon = 3$).

3.2. Segmentation

In the segmentation process, we begin with the K-means image segmentation to divide the image into clusters. Each cluster contains a set of pixels that are similar in RGB color space. The number of clusters is defined by the parameter K, which is used as an adjusting parameter; the resulting image is shown in Figure 3.

These clusters boundaries show the edges and corners of the image, and are converted into a topological form, the contours. Once the contours have been formed, each contour is simplified using the RDP algorithm. Here, the parameter ϵ for the RDP algorithm could

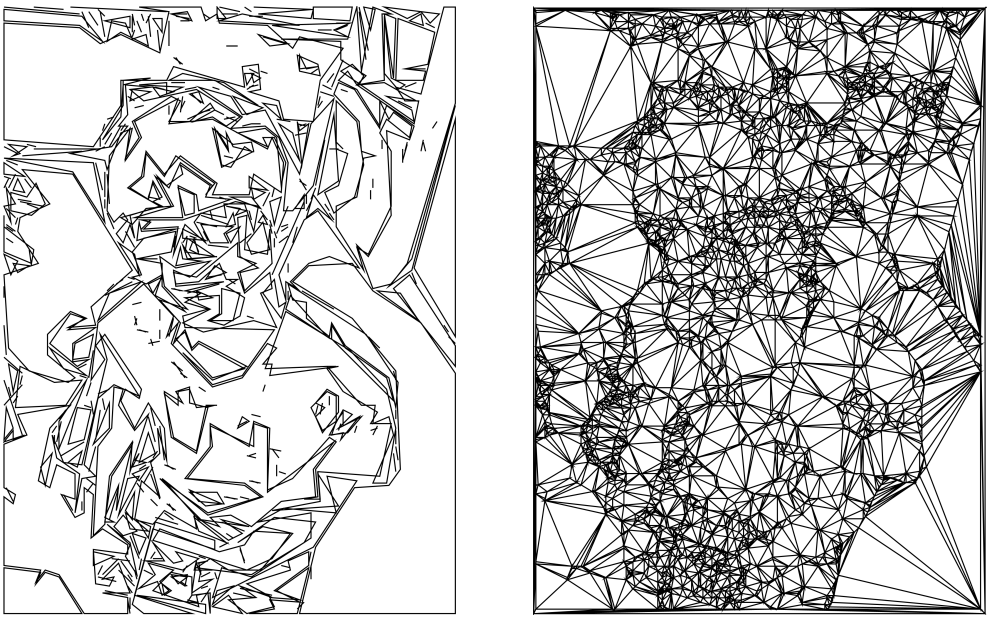

Figure 4: Combined contour (left) and triangulated contour (right).

be adjusted according to the user's purpose. A large ϵ will yield more degree of contour simplification. Figure 3 shows the simplified contour image from the cluster.

3.3. Triangulation

Once they have been simplified by the RDP algorithm, the clusters may not share their boundaries anymore. However, this is not regarded as a problem since the process trickily emphasizes the edges and corners of the image. Therefore, we combine the clusters into one cluster, which is triangulated using the Delaunay triangulation. The combined contour and the triangulated contour are shown in Figure 4.

3.4. Color interpolation

Each triangle is filled with an average color value that is interpolated from the set of pixel values inside the boundary using the triangle rasterization algorithm. [12]

Figure 5: The final image.

4. Experiment and evaluation

To evaluate our proposed algorithm, we performed subjective and numerical evaluations. For our subjective evaluation, we conducted a user test using the blind test procedure. For our numerical evaluation, we measured the mean structural similarity index (MSSIM) of the images using a method suggested by [13].

4.1. Subjective evaluation

We conducted a questionnaire that contained 30 image sets to 30 participants. Each image set contained two low-poly images that were generated from the *Polygen* application and our method. We did not display information about the source of the images so that the participants could not determine which images were from the *Polygen* application and which images were from our method. The participants then had to indicate which image they preferred for each of the 30 image sets.

Subjects

Thirty students from King Mongkut's University of Technology Thonburi, who studied in the field that involves art, computer graphic, or human perception fields, volunteered to participate in the study.

Stimuli

Sixty low-poly images were used in the user test. There were 30 original images; each image was rendered as a low-poly image using two different tools: the *Polygen* application and our algorithm. Therefore, a total of 30 image sets comprised of two low-poly images were used as stimuli. The source images varied in terms of content and features.

Experimental procedure

An online questionnaire create by *Google Form* was used and distributed through online channels, such as email and social networks. Participants were directed to select the image that they preferred, given the original image and the two low-poly images generated by the *Polygen* application and our algorithm. The whole process was a blind test; users were not aware of the methods used to generate the low-poly images.

Experimental results

The user test results revealed that 65% of the participants preferred the low-poly image that was generated using our method. However, the *Polygen*-generated low-poly images were preferred for some types of images.

4.2. Numerical evaluation

We measured the MSSIM of each pair of generated images, by *Polygen* and our algorithm, and the original one. For this measurement, we set $K_1 = 0.01$, $K_2 = 0.03$, $C_3 = \frac{C_2}{2}$ and $\alpha = \beta = \gamma = 1$ under a 11×11 Gaussian weighting function with a standard deviation of 1.5.

Experimental result

A comparison between the images generated by the *Polygen* application and our algorithm showed that for 20 out of 30 image pairs, the MSSIM of our algorithm-generated image was greater than the MSSIM of the *Polygen*-generated image.

4.3. Conclusion

From the experiment, we found that the low-poly images generated by our algorithm were preferred to images generated by the *Polygen* application. However, there were some *Polygen*-generated images that were preferred even though the edges were jagged. Future research on the relationship of low-poly images and image content is needed to provide more information on this finding.

5. Discussion

Our algorithm allows for a fully automatic low-poly image generation that preserves some global features, such as average colors in image segments and the edges of images. One possible error with our generated low-poly image is that the proposed algorithm was not designed to detect human-specific importance in the images. More experiments on suitable parameters may be conducted to achieve better results for each type of image.

As stated in the experimental results, there seemed to be some factors, such as image content and the viewer's personal preferences, that may have affected the final results. This should be studied further in terms of low-poly art appreciation. Also, the image size strongly affects the computational time. We recommend that the image should be resized in proportion with the input parameters to avoid lengthy calculations.

6. Conclusions

In this paper, we presented a better automatic method to produce low-poly vector image from bitmaps. Using this method, unimportant details are filtered out of the input image, and the image is then segmented, simplified, triangulated, and assigned color. Finally, the output image is written in a vector format that can be conveniently further edited for purposes. This can be useful for applications or in modern art, low-polygon style, or on a large press.

Acknowledgements

We thank all the volunteers from King Mongkut's University of Technology Thonburi who participated in the user test. We also thank the anonymous reviewers for their comments during preparation of our abstract.

References

[1] Autodesk: *Autodesk 3ds Max 9 essentials*. Focal Press, 1990–2006.

[2] B. Bitencourt: *Low Poly Studies*. Bēhance, 2016. Retrieved 19 January 2016 from `https://www.behance.net/gallery/14372365/Low-Poly-Studies/`.

[3] B. Bitencourt: *Create a low-poly portrait*. Digital Arts, 2016. Retrieved 10 January 2016 from `http://www.digitalartsonline.co.uk/tutorials/adobe-illustrator/create-low-poly-portrait-photoshop-illustrator/`.

[4] G. Bradski: *The OpenCV Library*. Dr. Dobb's Journal **25**/11, 120–126 (2000).

[5] DMesh: *Triangulation Image Generator*. Retrieved 31 January 2016 from `http://dmesh.thedofl.com/`.

[6] B. Delaunay: *Sur la sphère vide*. Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles **6**, 793–800 (1934).

[7] T. Kanungo, D.M. Mount, N. Netanyahu, C. Piatko, R. Silverman, A.Y. Wu: *An efficient k-means clustering algorithm: Analysis and implementation*. IEEE Trans. Pattern Analysis and Machine Intelligence **24**, 881–892 (2002).

[8] B. Niemtur: *PolyGen – Create Polygon Art*. Retrieved 31 January 2016 from `https://play.google.com/store/apps/details?id=com.bitbotany.polygen/`.

[9] U. Ramer: *An iterative procedure for the polygonal approximation of plane curves*. Comp. Graph. Image Process **1**/3, 244–256 (1972).

[10] J. Shewchuk: *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*. In Ming C. Lin, D. Manocha (eds.): *Applied Computational Geometry: Towards Geometric Engineering*, Lecture Notes in Computer Science, vol. 1148, Springer-Verlag, Berlin 1996, pp. 203–222.

[11] J. Shewchuk: *Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator*. Retrieved 2 February 2016 from `http://www.cs.cmu.edu/~quake/triangle.html/`.

[12] Sunshrine: *Software Rasterization Algorithms for Filling Triangles*. Sunshrine's Homepage. Retrieved 19 April 2017 from `http://www.sunshine2k.de/coding/java/TriangleRasterization/TriangleRasterization.html/`.

[13] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli: *Image Quality Assessment: From Error Visibility to Structural Similarity*. IEEE Trans. Image Process. **13**/4, 600–611 (2004).

Received August 6, 2016; final form May 12, 2017