Low-Poly Style Image and Video Processing

Wenli Zhang [1], Shuangjiu Xiao [1], Xin Shi [1]
[1] Shanghai Jiao Tong University, 800 Dongchuan RD. Minhang District, Shanghai, China
lilyperfect@sjtu.edu.cn, xsjiu99@cs.sjtu.edu.cn, shinshi@sjtu.edu.cn

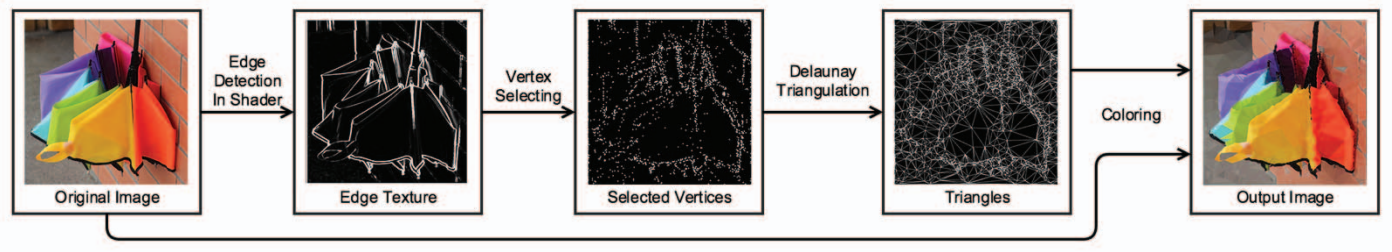


Figure 1. Main framework involves with edge detection, vertex selecting, triangulation, and coloring.

Abstract - **Low-poly has lately become a trending style in flat designs, Web designs, illustrations, and etc., in which only a small amount of triangles are used to provide an abstract and artistic effect. However, creating designs in low-poly style manually is obviously a tiring job. In this paper, we propose a real-time triangulation method to synthesize images and videos automatically into low-poly style. In order to keep edge and color information out of limited triangles, vertices on the edges detected in the image have a greater probability to be selected to compose triangles. For video low-poly stylization, an anti-jittering method is proposed to eliminate the abrupt changes in position and color of the triangles between adjacent frames. We use OpenGL Shading Language (GLSL) to speed up the calculation in GPU. We compare images generated with our method with other algorithms and those drawn manually by artists. Results show that our method provides an elegant and artistic effect for low-poly in real-time.**

Keywords **– Image Processing; Video Processing; NPR; Low-Poly**

I. INTRODUCTION

Back to the days when the performance of computers was quite limited, low-poly is only a compromise to frame-rate in 3D modeling and rendering rather than an artistic rendering style. Nowadays, it has become a trend for artists to draw illustrations in low-poly style, which are generally composed with triangles of different colors and sizes.

Although low-poly image processing is a quite new research topic, non-photorealistic rendering in similar styles has been studied for years. Gerstner et al. [1] proposed an effective method to generate pixel art that transforms high resolution images into low resolution. DeCarlo et al. [2] and Wen et al. [3] generated images in a sketch style with bold edges and large areas of constant color. Winnemöller et al. [4] used bilateral filter to produce abstract images. Chen et al. [5] proposed method to generate stylish images with polygons by Voronoi Tessellation, whose result looks quite similar to those of low-poly style. The purpose of these non-photorealistic rendering are similar, to render the images into a more abstract result without losing main information like shape and color.

As for low-poly image rendering, the main difficulty lies in preserving shape and color information with limited triangles. With only a significantly limited amount of triangles and colors, the task of processing images into low-poly style becomes mostly the task of choosing vertices and colors cautiously.

Recently, Gai et al. [6] proposed a method to select vertices using the Voronoi diagram iteration guided by a feature flow field. Their generated results are impressive but the pipeline is quite time consuming and the processing time for an image of size 512×512 is about 4 seconds.

In this paper, we present a real-time method to generate images and videos into aesthetically pleasing low-poly style that reserves main information in the original image in real-time. We provide a competitive result with much shorter processing time, which can be used in real-time applications.

As Figure 1 shows, our method is composed of three stages, vertex selecting, triangulation, and coloring. In vertex selecting stage, Sobel edge-detecting [7] shader is used to locate edges in the image. Pixels on the edges have a greater probability to be selected so that edges are more likely to be preserved when composing triangles. In triangulation stage, we then use Delaunay triangulation [8] to composite the selected vertices to form triangles, which maximizes the minimum angle of all the angles of the triangles in the triangulation. In coloring stage, each triangle is filled with the color at its center of mass in the input image.

For video processing, however, if we apply this algorithm directly, there would be an obvious jittering artifact. This is because vertices are selected quite randomly in each frame, so that triangles may probably have abrupt changes in position and color between two consecutive frames. We improved our method by providing a higher possibility for vertices on the edge that were selected in the last frame to be selected in this frame.

Our main contribution lies in proposing a low-poly style processing method accelerated in GPU, and eliminating jittering artifact in video processing. We compared the output

of our method with Naïve random algorithm and low-poly images drawn by artists, and result shows that our method has a compelling artistic effect and reserves most information out of the original images.

II. FRAMEWORK

A. Vertex Selecting

Images in low-poly style are composed by triangles of different colors and sizes. In order to preserve main information with limited triangles, we need to find an efficient way to choose triangles.

(a) (b) (c)

Figure 2. Selecting more vertices on the edges is a more efficient way to reserve information from original. (a) Original image of a colorful umbrella. (b) Low-poly image generated by our method with 1000 vertices. (c) Low-poly image generated by naïve random algorithm with 1000 vertices.

Figure 2 shows two rendering result with different vertices selecting algorithm with the same vertex amount. Figure 2(c) uses a naïve random policy, in which vertices are selected fully randomly. As we can see, edges in the original image may probably not remain in the result and the umbrella becomes hardly recognizable. After some attempts, we find it efficient to choose vertices more densely in the detail areas while sparsely in the large areas with similar colors. And Figure 2(b) shows the result with vertices of the same amount chosen with our method, in which vertices on the edges have a higher probability to be selected.

We use edge detection method to represent the density of information in the original image, and pixels on the edges are expected to contain more detail information, and thus are more likely to be selected later. In order to speed up the calculation in GPU, we use a Sobel edge detecting shader to render the edge of the original image into a texture. The color of the texture G is set to be gradient magnitude of the horizontal and vertical convolution result G_x and G_y using a 3×3 kernel Sobel operator [7].

$$G = \sqrt{G_x^2 + G_y^2} \tag{1}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A \tag{2}$$

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \tag{3}$$

where A is the original image.

Figure 3 shows a rendered texture by Sobel edge shader. In this edge texture, pixels that are more likely to be edges are presented with lighter pixel colors, and they have a higher probability to be selected as vertices since vertices on the edges are expected to contain more details. On the other hand, if only pixels on the edges may be selected, the outcome may be too abrupt and abstract. So we also select a small amount of vertices that are not on the edges to make sure the large areas are also well presented in the output.

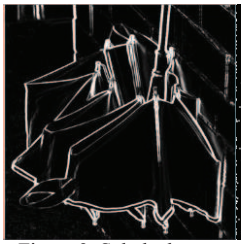


Figure 3. Sobel edge shader.

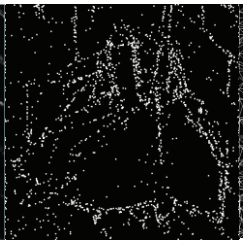


Figure 4. Selected vertices.

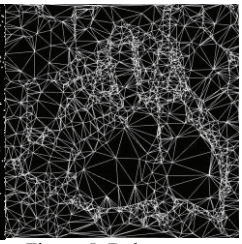


Figure 5. Delaunay triangulation result.

To select n vertices from an image with N pixels, the overall probability for a pixel to be selected as a vertex is $P(x) = \frac{n}{N}$.

Let $E(x) \in \{0, 1\}$ denote whether pixel x on the edge of the original image. $E(x) = 1$ means pixel x is an edge pixel whose color at the same position in the edge texture is larger than a threshold t. We set the probability to be a constant value α for an edge pixel x selected as a vertex.

$$P(x|E(x) = 1) = \frac{\alpha \cdot n}{n_e} \tag{4}$$

where n_e is the number of vertices on the edge.

So, the probability for a non-edge pixel to be selected is

$$P(x|E(x) = 0) = \frac{(1 - \alpha) \cdot n}{N - n_e} \tag{5}$$

In order to make sure details are represented with more triangles, while preventing large areas from being too abstract, we set α to be 0.9 in our case. Selected vertices are shown in Figure 4 in white color.

B. Triangulation

In order to make sure the rendering result is aesthetically good, we need to avoid triangles from being too acute.

Delaunay Triangulations are widely used in many diverse applications. While there are numerous triangulation algorithms, the geometry properties of Delaunay Triangulation tend to avoid skinny triangles [8, 9]. The output of Delaunay triangulation algorithm is a list of triangles as shown in Figure 5.

C. Coloring

Since the number of triangles in low-poly style images is rather limited, choosing a suitable color for each triangle is also a very important procedure. What makes it difficult is that we need the coloring stage to be efficient enough for real-time processing, and on the other hand, it should also depict the original image.

Thus, we use the color in the original image at the center of mass of each triangle to be the triangle's color.

For each triangle $T_{a,b,c}$ generated in Triangulation Stage, which is formed with vertices a, b, and c, its color $C(T_{a,b,c})$ is set to be the color at the center of mass of the $T_{a,b,c}$ in input image.

$$C(T_{a,b,c}) = I(\frac{x_a + x_b + x_c}{3}, \frac{y_a + y_b + y_c}{3}) \quad (6)$$

where $I(x, y)$ is the color at the position (x, y) in the original image, x_a, x_b, x_c and y_a, y_b, y_c are horizontal and vertical positions of vertices a, b, and c.

This is a $O(N)$ algorithm, where N is the number of triangles. And in practice, it is also a very effective algorithm that provides compelling output.

III. VIDEO PROCESSING

For low-poly style video processing, the main difficulty lies in designing a real-time algorithm, as well as solving the jittering artifact. We use shader programs to speed up the calculation in GPU.

(a) (b) (c)

Figure 6. Vertices are selected rather randomly, so there may be abrupt changes in triangles' positions and colors in adjacent frames. (a) ~ (c) are three frames of a video with the same content.

Jittering artifact is a phenomenon happens when the triangles seem to change in a large extent between two successive frames, as shown in Figure 6. This is caused by randomly selecting vertices in each frame. So the main idea of our improved algorithm is to prefer selecting those vertices on the edges of this frame that was selected in the previous frame. We randomly observe a pixel, if it is on the edge of this frame and was select in last frame, then, the probability for it to be selected as a vertex is$1 - \beta$, where β is the elimination rate.

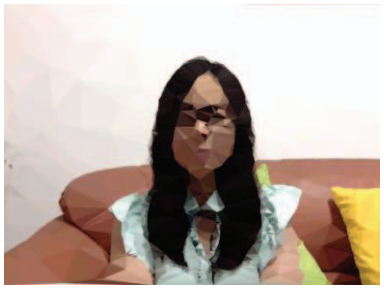


Figure 7. Unsatisfying result when there is no elimination rate, where most of the vertices are selected on the unchanged edges like those of the sofa.

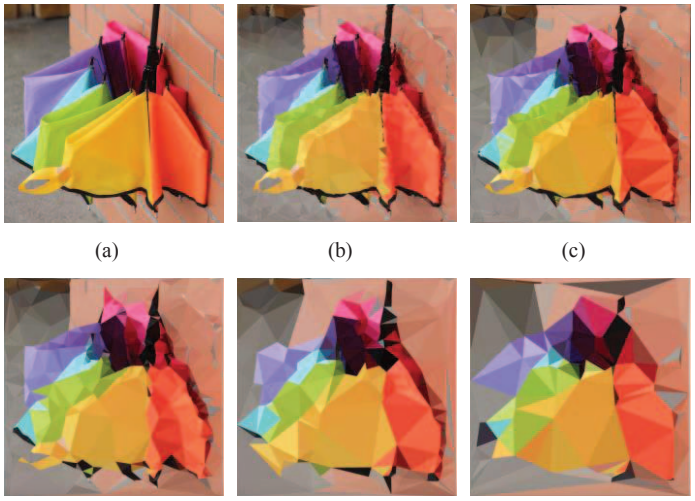
Here, β is set to be a positive value less than 1 to make sure there's also an eliminating chance for the vertices selected. Otherwise, more and more pixels in the unchanged area will be selected and thus leave no more extra vertices for the changed parts. Figure 6 shows an example when β is set to be 0. As we can see, after 10 seconds, the unchanged background takes most of the vertices and the changing foreground is thus made too abstract, which is not an ideal result we want.

After some attempts, we set β to be 0.1 and the jittering artifact has been resolved effectively while the overall rendering result is also aesthetically acceptable and main information in the original image has been reserved.

IV. EVALUATION

A. Comparison with different vertex amount

As shown in Figure 8, low poly images are made more abstract and stylistic with a smaller amount of vertices. But if the number of vertex is set to be too small, the figure in the image may become unrecognizable.



(a) (b) (c)

(d) (e) (f)

Figure 8. Low-poly rendering results with different vertex amount. (a) Original image. (b) ~ (f) are result of 2000, 1000, 500, 200, and 100 vertices.

B. Comparison with other algorithms

We evaluate our method with TRIGRAFF [10] and that proposed by Gai et al. [6]. Figure 9 shows part of the results of an original image at size 558×800.

Degenerated triangles as in Figure 9(b) are largely avoided in our method in Figure 9(d). And in other aspects, we have a generally similar result.

Figure 9(c) provides a better result than our method especially on the smoothness of the edges. But their processing time of this image is 4.791 seconds, while ours is only 0.247 seconds. So our method has a better potential to be used in real-time applications.

C. Comparison with low-poly image drawn by artists

We also compare our method by a low-poly image drawn manually by an artist [11], as shown in Figure 10. The low-poly image drawn by the artist has a better potential to locate edges, especially around the tiger's eyes. And the artist modified the colors to enhance the luminance for better visual effect.

(a) Original Image

(b) Result by TRIGRAFF [10]

(c) Result by Gai, et al [6]

(d) Result by Our Method

Figure 9. Low-poly rendering result comparison with different methods. Figure (a) ~ (c) are provided by [6].

(a) (b) (c)

Figure 10. Low-poly rendering result comparison with image drawn by an artist. (a) Original image (2) Low-poly image generated by our method (3) Low-poly image drawn by an artist from [11].

On the other hand, our method also provides a competitive result that well illustrates the tiger in the original input. And more importantly, it takes less than a second for our method to generate the low-poly image, while it may take an artist more than an hour to do such a job.

D. Video low-poly processing

As for video processing, we use Webcam to get video stream and test with our improved Low-poly video algorithm. Result shows that our method resolves jittering artifact efficiently, although not yet quite thoroughly. On the other hand, our eliminating algorithm effectively updates vertices in unchanged background to prevent it from falling into too much detail.

E. Speed evaluation

We implemented our method with WebGL and edge-detecting algorithm was written in WebGL shader to accelerate calculation in GPU. We test images and videos on a computer with Intel(R) Core(TM) i7-3770 CPU at 3.4GHz, and a 8GB RAM, and graphics card of NVIDIA GeForce GTX 690.

In practice, we usually set the number of selected vertices to be around 1000, and the processing time is short enough for real-time applications, as shown in Table I.

TABLE I. PROCESSING TIME (IN SECONDS) OF DIFFERENT IMAGE SIZE AND NUMBER OF SELECTED VERTICES.

Image width and height (pixels)	**Number of selected vertices**			
	4000	***2000***	***1000***	***500***
1024	0.155	0.134	0.102	0.097
512	0.125	0.08	0.076	0.051
256	0.1	0.072	0.057	0.042

V. CONCLUSION

In this paper, we present an automatic method to generate images and videos into aesthetically pleasing low-poly style that reserves main information in the original image in real-time. As for video processing, we improved our method by providing a higher possibility for vertices on the edge that were selected in the last frame to be selected in this frame. And the elimination rate prevents the unchanged background from falling into too much detail. Our method effectively reserves edge and color information in the original image and has an artistic result.

In the future, we plan to improve our vertex-selecting algorithm to better preserve edge information in the original images and improve our video processing method to eliminate the jittering artifact more effectively. We should also reduce processing time to provide a better real-time experience. Besides, we may also consider low-poly as a method to compress images.

REFERENCE

[1] Gerstner et al., "Pixelated image abstraction." in *Proc. of the Symposium on Non-Photorealistic Animation and Rendering*, 2012, pp. 29-36.

[2] DeCarlo et al., "Stylization and abstraction of photographs." in *ACM Transactions on Graphics (TOG)*, 2002, vol. 21, no. 3, pp. 769-776.

[3] Wen et al., "Color sketch generation." in *Proc. of the 4th international Symp. on Non-photorealistic animation and rendering*, 2006, pp. 47-54.

[4] Winnemöller et al., "Real-time video abstraction." in *ACM Transactions On Graphics (TOG)*, 2006, vol. 25, no. 3, pp. 1221-1226.

[5] Chen et al., "Approximation by piecewise polynomials on voronoi tessellation" in *Graphical Models*, 2014, no. 5, pp. 522-531.

[6] Gai et al., "Artistic Low Poly rendering for images." in *The Visual Computer*, 2015, pp. 1-10.

[7] Kanopoulos et al., "*Design of an image edge detection filter using the Sobel operator.*" in *Solid-State Circuits, IEEE Journal of 23*, 1988, no. 2, pp. 358-367.

[8] B. Delaunay., "Sur la sphere vide. A la memoire de George Voronoi." in Bulletin de l'Académie des Sciences de l'URSS, 1934, no. 6, pp. 793-800.

[9] P. Bourke, "An algorithm for interpolating irregularly-spaced data with applications in terrain modelling." in *Pan Pacific Computer Conference*, Beijing, China. 1989, vol. 6.

[10] K. Okamoto. (2015, Aug. 22). *Art camera trigraff* [online]. Available: https://itunes.apple.com/cn/app/art-cameratrigraff/id646603902.

[11] (2015, Aug. 22) *How To Create Geometric Low Poly Art The Easy Way* [online]. Available: http://blog.spoongraphics.co.uk/tutorials/create-geometric-low-poly-art-easy-way.