AN INTERACTIVE SYSTEM FOR LOW-POLY ILLUSTRATION GENERATION FROM IMAGES USING ADAPTIVE THINNING

Yiting Ma, Xuejin Chen, Yu Bai

University of Science and Technology of China
CAS Key Laboratory of Technology in Geo-spatial Information Processing and Application System
mayt0829@mail.ustc.edu.cn, xjchen99@ustc.edu.cn, bby0623@mail.ustc.edu.cn

ABSTRACT

Low-poly style illustrations, which have 3D abstract appearance, have become a popular stylish recently. Most previous methods require special knowledges in 3D modeling and need tedious interactions. We present an interactive system for non-expert users to easily manipulate the low-poly style illustration. Our system consists of two parts: vertex sampling and mesh rendering. In the vertex sampling stage, we extract a set of candidate points from the image and rank them according to their importance of structure preserving using adaptive thinning. Based on the pre-ranked point list, the user can select an arbitrary number of vertices for the triangle mesh construction. In the mesh rendering stage, we optimize triangle colors to create stereo-looking low-polys. We also provide three tools for exible modication of vertex numbers, color contrast, and local region emphasis. The experiment results demonstrate that our system outperforms state-of-the-art method via simple user interactions.

Index Terms— Non-photorealistic rendering, low poly, adaptive thinning, interactive system

1. INTRODUCTION

Non-photorealistic rendering (NPR), which aims to create the stylized images, is a traditional research area in computer graphics. In early years, researchers developed stroke-based systems [1, 2]. Later, it became popular to render different artistic styles, such as Impasto drawing [3], hand-painting [4, 5] and stained glass styles [6, 7]. Recently, novel and interesting renderings are favored, such as the Arcimboldo-like collage [8] and shape abstract arts [9].

Low-poly style illustrations, which consist of triangle meshes and have 3D abstract appearance, were used in early 3D modeling with low-detailed and faceted style when computers had difficulties to handle complex meshes. With the development of simplified visual design, the low-poly style is re-emerging and has become a popular stylish in the non-photorealistic rendering area. Fig. 1 shows a low-poly self portrait created by a graphic designer, Breno Bitencourt [10]. We can see that the artist keeps more details in some areas like the eye area, the fingers and less details in the cloth region. As a new and cool form of non-photorealistic rendering, however, it is non-trivial to create a visually pleasing version from a photograph, even for expert users.

Fig. 1: A low-poly artwork created by Breno Bitencourt [10].

There are many existing software packages to allow users to manually create low-poly illustrations, such as Triangulation, Adobe Photoshop or Illustrator. However, these softwares require users to manually select pixels in an image for creating a triangle mesh. This tedious and skilled work is quite unfriendly for non-expert users. DMesh [11] and TRIGRAFF [12] are two mobile Apps to help users generate low-poly illustrations from images efficiently. The triangle mesh is automatically generated. The former allows the user to control the number of mesh points and manually edit point positions, while the later provides four levels of detail for user to choose. Though they provide simple interfaces for non-expert users, the image content is not well preserved.

We thank the reviewers for their generous reviews that shaped the direction of the paper. This work was supported by the National Key Research & Development Plan of China under Grant No. 2016YFB1001402 and the National Natural Science Foundation of China under No. 61472377, 61632006, 61331017.

978-1-5090-6067-2/17/$31.00

Recently, a few research has been conducted on the automatic generation of low polys from images. Gai and Wang [13] presented an automatic approach to generate low poly renderings from images using the Voronoi diagram iteration guided by a feature flow field. However, the number of points is not allowed to modify. Moreover, this method is sensitive about the quality of the extracted feature edges and produces too many triangles if the input image is full of delicate details. Zhang et al. [14] presented an algorithm to generate low-poly images and videos by giving each point a sampling probability to keep edges. However, the sampled points usually deviate from object boundaries so that the image structure is not well-preserved. Moreover, neither of them supports user interactions. As an artistic creation process, user interaction with powerful and flexible tools is important for generating diverse results and inspiring user's creativities.

In this paper, we present an efficient algorithm and an interactive system for generating low polys from images. To preserve image structure, we use an over-segmentation method to extract the structural information. While plenty of vertices are extracted from the over-segmented region corners, we rank them using the adaptive thinning [15] according to their saliency on changing image structure. Based on the ranked vertex set, our system is able to construct triangle mesh of arbitrary number of points in real time.

The adaptive thinning algorithm [15] is a greedy point removal scheme for scattered data sets, where the points are recursively removed according to some data-dependent criterion. This is in contrast to non-adaptive thinning methods, where the criterion only depends on the geometry of the given planar point set [16]. Adaptive thinning algorithm is widely used in image compression [17, 18, 15] by simplifying the triangulation of an image. This method is also used to generate the skeleton of images [19], aiming to represent an image with a lower number of pixels. In comparison, we use the adaptive thinning algorithm to rank the sampled points in one-pass to support the interactive selection of vertex numbers.

Not only an automatically generated low poly with default parameters is provided, three flexible interaction tools are also provided to allow users to continuously modify the number of mesh points, to paint an image region for emphasizing specific details, and to modify the low-poly contrast by solving a Poisson equation. We demonstrate that our method could automatically generate a variety of visually pleasing low-poly illustrations from images and allow users to produce more diverse results with our simple and efficient interaction tools.

In general, our contribution is two-fold:

- We design an efficient interactive system with multiple easy-to-use tools which is friendly for users to generate visually pleasing low-poly illustrations.
- We propose an effective algorithm to generate colored triangle meshes of arbitrary number of points by pre-ranking the initial point set using an adaptive thinning method. Image structure is preserved during the ranking process by measuring the structure change caused by moving a point.

2. OVERVIEW

Given an input image I which contains N pixels $\{p_1, ..., p_N\}$, our problem is to generate a low poly $\mathcal{L}$, which is a triangle mesh $\mathcal{M}$ with a unique color in each triangle. A good low poly should have a similar structure with the original image in an abstract level and an artistic looking. Given the desired number of vertices K, we select K pixels from the input image as vertices and construct a triangle mesh so that the edges should follow image boundaries.

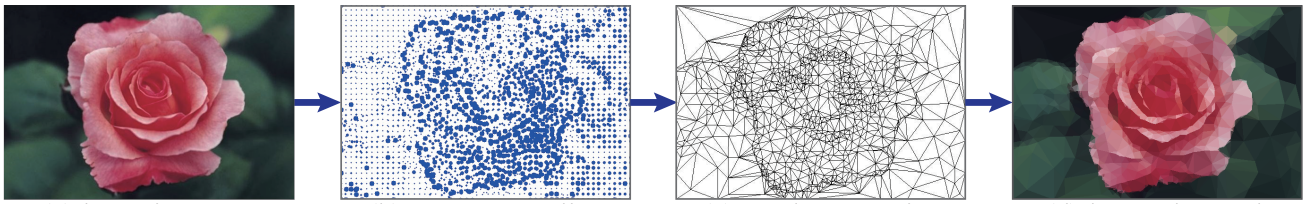
Our algorithm consists of two main parts, as Fig. 2 shows. Firstly, we sample sufficient important points in the input image using an over-segmentation algorithm, and rank them according to their importance of preserving the image appearance using the adaptive thinning algorithm, as described in Sec. 3.1. Then a triangle mesh is constructed from the first K points in the ranked list using the Delaunay triangulation based on the desired number of points K. Finally, we assign each triangle with a color to generate an artistic rendering, as described in Sec. 4. Based on our algorithm, we provide three flexible tools for users to interactively change the number of points, paint local regions to add more details, and adjust the appearance contrast, as described in Sec. 5.

3. VERTEX SAMPLING

In order to allow users specify arbitrary number of vertices, we do not run the entire sampling process at each time since there are many important points must be sampled every time to preserve the image structure. To save the computation cost, we first sample sufficient points from the image and sort them according to their importance for structure preserving. Once the number of points K is given, we can simply pick the first K points from the ranked point set and construct a mesh using the Delaunay triangulation.

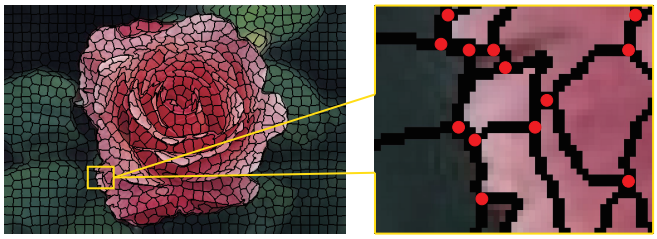
3.1. Vertex candidates

In order to preserve the structures of the input image and sample points on object boundaries, we first over-segment the image into superpixels. We choose the intersections of several adjacent superpixels as the initial set of vertex candidates $P_{subset} = p_1, ..., p_M$, where $K < M < N$. There are many over-segmentation methods in the literature. We choose the SLIC method [20] because of its high boundary recall and efficiency. Initially, to keep as much structure as possible, we set a significantly large number $N_{init} = 3000$ of superpixels. Fig. 3 shows the over-segmented superpixels and the initial point set. We can see that the selected points are located on object boundaries and tend to generate an evenly distributed



(a) input image (b) vertex sampling (c) mesh generation (d) low poly result

Fig. 2: Algorithm overview. Given an input image (a), we first sample pixels and rank them according to their importance of preserving the image structure (b). We can see that the points on the edges are much more important. Then a Delaunay triangulation is constructed from the first K points according to user's selection (c). Finally, we render the color of each triangle to generate a low-poly illustration with a artistic looking.



(a) image over-segmentation (b) vertex candidates

Fig. 3: (a) The over-segmentation result SLIC [20]. (b) Superpixel corners are selected and vertex candidates.

mesh. In the following process, our system will rank the candidate pixels efficiently.

3.2. Vertex ranking

After the over-segmentation, the number of sampled points is much larger needed for generating a low poly. Given a desired number K of vertices in the final low poly, we would select K points from the N_{init} points to construct the triangle mesh. However, if the user wants to set any number from a wide range of K, it is not necessary to run the whole process for each number. Therefore, we propose a ranking algorithm using adaptive thinning scheme [15], which recursively removes less important points according to some data-dependent criterion. The M points are ranked just once. After that, the mesh can be constructed from the first K points according to the user's selection. This is different from the previous two methods which fixed the number of points at the first step and once the number changed, they need to run the entire process again.

The data-dependent criterion we use to evaluate a point is the error between the triangle mesh $\mathcal{M}$ after removing a point and the input image. When a point is removed, the resulted triangulation colors will change. The point is more important if the resulted color differs more from the image. The color $\mathbf{c}_i$ of each triangle T_i in $\mathcal{M}$ is computed as the average of the median part of colors in the triangle to avoid the zigzag artifact, same as [13]. Given a finite points set P in a 2D plane, the *Delaunay triangulation* is a triangulation $\mathcal{D}(P)$ which satisfies that no point of P is inside the circumcircle of any triangle in $\mathcal{D}(P)$. It maximizes the minimum angle of all angles of triangles in $\mathcal{D}(P)$ and is regarded as the most uniform triangulation from P. For how to compute the Delaunay triangulation, we refer to the textbook [21].

If k points P_k are selected, we can construct a Delaunay triangulation $\mathcal{D}(P_k)$ and compute the average color of each triangle to generate a mesh $\mathcal{M}_k$ to approximate the input image I. The color mesh $\mathcal{M}(P_k)$ can be treated as an approximation function to the input image I with k sampled points. The approximation error of $\mathcal{L}_{P_k}$ to the image I is the sum of the color difference at all pixels $\mathbf{x}$ in the image

$$\eta(\mathcal{M}(P_k); I) = \sum_{\mathbf{x} \in I} |\mathcal{M}_{\mathbf{x}}(P_k) - I_{\mathbf{x}}|. \tag{1}$$

If a point p in P_k is removed, a new Delaunay triangulation $\mathcal{D}(P_k - p)$ is constructed and the triangle colors are computed to generate another color mesh $\mathcal{M}(P_k - p)$. We can compute the approximation error of $\mathcal{M}(P_k - p)$ as $\eta(\mathcal{M}(P_k - p; I))$. Due to the Delaunay property, the update of $\mathcal{M}(P_k)$ to $\mathcal{M}(P_k - p)$ is local, which means we only need to run the re-triangulation on the local region of the point p and its adjacent triangles, and the local approximation error is

$$\eta(p; X) = \sum_{\mathbf{x} \in \mathcal{N}(p)} \mathcal{M}_{\mathbf{x}}(P_k - p) - I_{\mathbf{x}}, \tag{2}$$

where $\mathcal{N}(q)$ means the local region composed of all the triangles of which p is a triangle vertex, as shown in Fig. 4.

According to the approximation error of each point, we can recursively remove the point that has the minimum approximation error at each time. Therefore, given the M sampled points P_M initially from the over-segmentation results, we first generate a color mesh $\mathcal{M}(P_M)$ using the Delaunay triangulation. Then we compute the local approximation error of each point in P_M as Eq. 2. We choose the point p^* has minimum approximation error and push it to a queue. Once p^* is removed from P_M, we can obtain a new Delaunay triangulation and a color mesh by locally updating its neighbor points. The approximation errors of its neighbor points are then updated for the next pass of point removing.

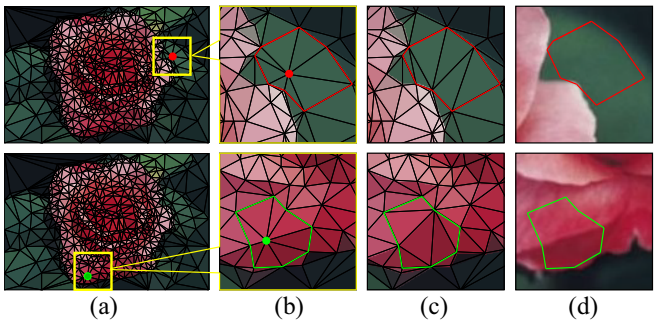


Fig. 4: Local update of the Delaunay triangulation when two points are removed (top and bottom). (a) From $K = 500$ points, a low poly is generated by a Delaunay triangulation with average color in each triangle. (b) Once a point is removed, the re-triangulation is only performed in a local region (c). Compared to the original image in the updated local patch (d), the color difference of the updated region (c) is used as the criterion to rank the importance of the point to be removed. For a vertex in a flat region (the top row), the appearance does not change a lot. While for a vertex in a region of rich-structure (the bottom row), the appearance after the vertex is removed differs more from the image, so that the green vertex is much more important than the red one.

After we rank the M candidate points using adaptive thinning, the user could specify an arbitrary number $K \in (100, M]$ of points so that our system directly picks the first K points in the queue and renders a low poly.

4. MESH RENDERING

After generating the triangle mesh, the next step is to colorize the mesh to generate a visual pleasing low poly. A simple way is to assign each triangle with the average color. However, low-poly style illustrations typically present the visually 3D effect. The 3D feeling is reflected by the scene geometry which conveys by the surface shading. To achieve this effect, we enhance the contrast between adjacent triangles by setting a minimum brightness difference L_{min}.

Given a triangle mesh $\mathcal{M}$, the average color $\mathbf{c}_i$ of triangle T_i is computed as the average of the median part of colors in the triangle to avoid the zigzag artifact, same as [13]. For each pair of adjacent triangles T_i and T_j, we emphasize the lightness difference $d_{i,j}$ as

$$d_{i,j} = \max(|L_i - L_j|, L_{min}), \tag{3}$$

where L_i and L_j are the lightness components in the LAB color space of T_i and T_j respectively.

Then for all the triangles in the mesh $\mathcal{M}$, we compute the enhanced lightnesses L'_i by solving an linear equation system

$$\begin{cases} L'_i - L'_j = d_{i,j}, & \forall \quad \mathcal{N}(T_i, T_j) \\ L'_i - L_i = 0, & \forall T_i \in \mathcal{M} \end{cases} \tag{4}$$

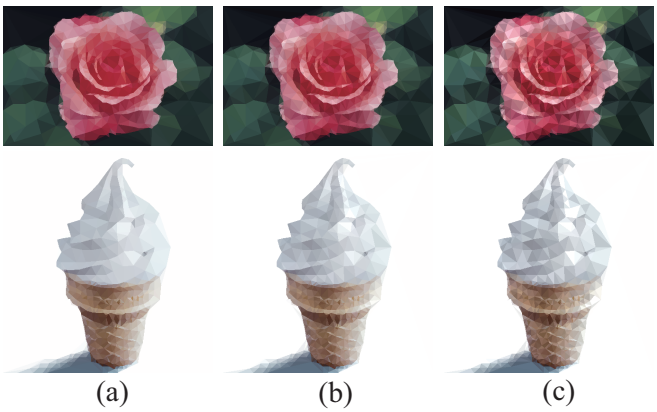


Fig. 5: Comparing with low polys of the same triangle mesh with different color contrasts. (a) Each triangle is assigned with the average pixel color in it. (b)(c) The low poly is enhanced by different minimum contrast.

The result is shown in Fig. 5. We can see that result after rendering have larger standard deviation in the color space. We also provide a slide bar allowing user to easily control the minimum contrast in real-time. As a result, various rendering effects can be generated for users to explore and select.

5. USER INTERACTION AND RESULTS

Besides of providing the user with an automatically generated low poly with default parameters, we also develop an interface to flexibly (a) modify the overall level of image details with a slider controlling the number of vertices, (b) to change the contrast with a slider controlling the minimum contrast L_{min}, and (c) to emphasize a local region with a brush.

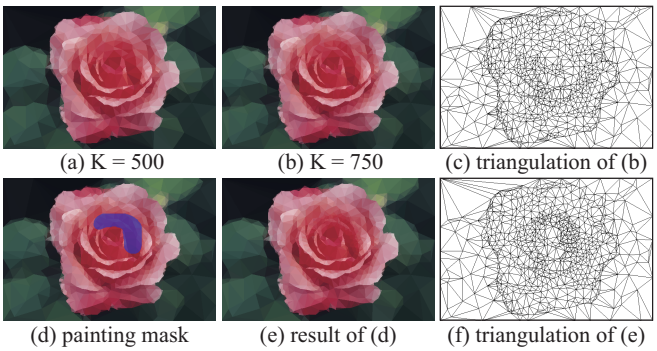


Fig. 6: More results generated by interactively editing the vertex number or painting local regions. Compared with less vertices (a), more details are kept with more vertices. While the same number of vertices is used, the user can paint a local region to emphasize the details (d). With $K = 750$, more triangles are used in the painted region (f) compared to (c).

Vertex Number The number of sampled vertices obviously

affects the final quality of the generated low polys. A fixed number is computed according to the image size in [13]. However, it may work disappointedly when the image have complex textures. Changing the vertex number is needed if the user want to explore more results. We provide a slider for users to continuously modify the vertex number and explore various results in real-time. As Fig. 6 (a)(b) show the generated low polys with $K = 500$ and $K = 750$ respectively. We can see that more details are preserved with more vertices.

Weight Map Different parts in a picture are of different semantic importance. For example, in a face picture like Fig. 1, the artist spends more triangles to emphasize the details in the eye region. Therefore, we provide a tool for users to interactively paint a region to emphasize its details. Since we sample sufficient vertices and compute the vertex importance in one-pass, as described in Sec. 3, we simply keep all the sampled vertices in this area and remove the least important points. Fig. 6 compares the generated low polys with and without user interaction. With the same number of vertices $K = 750$, more triangles are constructed in the painted region Fig. 6(e)(f) compared to Fig. 6 (b)(c).

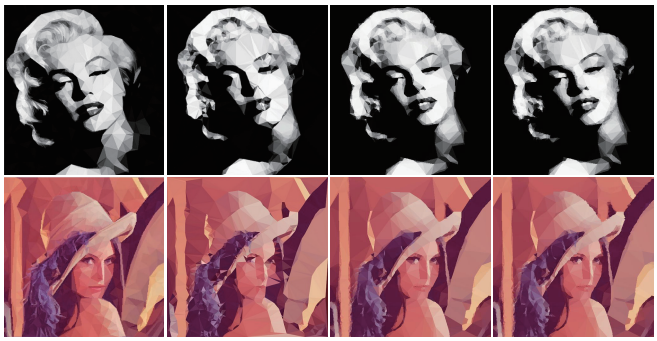


Fig. 7: Compared to two methods, (a) results by [13] and (b) results by TRIGRAFF [12]. Since they don't report the vertex number, we provide two results automatically generated with different vertex number to approximate their results.

We compare our results with the method [13] and the software TRIGRAFF in Fig. 7. It is obvious to see that TRIGRAFF generates lots of distortions in the generated low polys, while our results well preserve the image content in different level of details. Compared with Gai's algorithm [13], our algorithm tends to generate more regular triangles while keeping the image structure. While their method is controlled by a feature flow field, their results tends to follow the flow, such as the hair, and the hat decoration of Lena. In comparison, our results have a polygon style similar with the artist's work. More results generated for a wide range of scenes, including natural scene, objects, faces, and cartoons, are shown in Fig. 8. We can see that our algorithm generated visually pleasing low poly illustrations for various scenes.

To evaluate the efficiency of our method, we list the relevant statistics for running time of each step with an Intel i7 CPU with 3.4GHz and 16GB of RAM in Table 1. The vertex ranking stage using adaptive thinning averagely takes 3.5 second for $M = 2000$ initial points. The remaining part including mesh construction and color rendering runs in real time, thus our system allows users to interactively modify the number of points, triangle contrast, and edit local regions to generate multiple low polys from one single image.

Table 1: Experiment statistics for our examples. The time costs (in seconds) for three main stages in our system include the over-segmentation stage t_{slic}, the vertex ranking stage t_{rank}, the mesh construction and rendering $t_{rendering}$.

Examples	Image size	t_{slic}	t_{rank}	$t_{rendering}$
Flower (Fig. 6)	614×410	0.156	3.099	0.059
Ice-cream (Fig. 5)	444×507	0.136	3.129	0.044
Lena (Fig. 7)	512×512	0.187	3.363	0.062
Hamburger (Fig. 8)	717×537	0.239	3.959	0.086

6. CONCLUSION

We introduce an interactive system to generate low-poly style illustrations from images. By sampling sufficient points by over-segmenting the input image, we employ an adaptive thinning scheme to pre-rank the initial points according to their importance of preserving the image structure. With the one-pass point ranking, our system allows interactive editing by changing the number of vertices, the color contrast, and level of details in local regions. With our system, users could easily explore different low polys and choose desired results. One direction in the future is to combine 3D shape priors and object semantics to generate more 3D-looking low polys.

7. REFERENCES

[1] P. Haeberli, "Paint by numbers: Abstract image representations," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. 1990, SIGGRAPH '90, pp. 207–214, ACM.

[2] M. P. Salisbury, S. E. Anderson, R. Barze, and D. H. Salesin, "Interacting pen and ink illustration," *ACM SIGGRAPH Annual Conference*, pp. 101–108, 1994.

[3] J. P. COLLOMOSSE and P. M. HALL, "Salience-adaptive painterly rendering using genetic search," *International Journal on Artificial Intelligence Tools*, vol. 15, no. 4, pp. 551–575, 2006.

[4] J. P. Collomosse and P. M. Hall, "Painterly rendering using image salience," in *Proceedings 20th Eurographics UK Conference*, 2002, pp. 122–128.

Fig. 8: More results generated using our system for a wide range of scenes.

[5] B. Gooch, G. Coombe, and P. Shirley, "Artistic vision: painterly rendering using computer vision techniques," in *Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*. 2002, NPAR '02, pp. 83–90, ACM.

[6] D. Mould, "A stained glass image filter," in *Proceedings of the 14th Eurographics Workshop on Rendering*. 2003, EGRW '03, pp. 20–25, Eurographics Association.

[7] D. Ashlock, B. Karthikeyan, and K. M. Bryden, "Non-photorealistic rendering of images as evolutionary stained glass," in *2006 IEEE International Conference on Evolutionary Computation*, 2006, pp. 2087–2094.

[8] H. Huang, L. Zhang, and H. Zhang, "Arcimboldo-like collage using internet images," *ACM Transactions on Graphics*, vol. 30, no. 6, pp. 155:1–155:8, Dec. 2011.

[9] Y. Song, D. Pickup, C. Li, P. Rosin, and P. Hall, "Abstract art by shape classification," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 8, pp. 1252–1263, Aug. 2013.

[10] Breno Bitencourt, "Low poly artwork," `https://dribbble.com/shots/1757287-Low-poly-sir`.

[11] "Dmesh," `http://dmesh.thedofl.com/`.

[12] Apple iTunes, "Trigraff," `https://itunes.apple.com/cn/app/art-camera-trigraff/id646603902`.

[13] M. Gai and G. Wang, "Artistic low poly rendering for images," *The Visual Computer: International Journal of Computer Graphics*, vol. 32, no. 4, pp. 491–500, Apr. 2016.

[14] W. Zhang, S. Xiao, and X. Shi, "Low-poly style image and video processing," in *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, Sept 2015, pp. 97–100.

[15] L. Demaret, N. Dyn, M. S. Floater, and A. Iske, *Adaptive Thinning for Terrain Modelling and Image Compression*, pp. 319–338, Springer Berlin Heidelberg, 2005.

[16] M. S. Floater and A. Iske, "Thinning algorithms for scattered data interpolation," *BIT Numerical Mathematics*, vol. 38, no. 4, pp. 705–720, 1998.

[17] L. Demaret, N. Dyn, and A. Iske, "Image compression by linear splines over adaptive triangulations," *Signal Processing*, vol. 86, no. 7, pp. 1604–1616, July 2006.

[18] L. Demaret and A. Iske, "Advances in digital image compression by adaptive thinning," *Annals of the Mcfa*, vol. 3, pp. 105–109, 2003.

[19] M. Sarkis and K. Diepold, "Content adaptive mesh representation of images using binary space partitions," *IEEE Transactions on Image Processing*, vol. 18, no. 5, pp. 1069–1079, May 2009.

[20] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Ssstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, Nov 2012.

[21] F. P. Preparata and M. I. Shamos, *Computational geometry*, Springer-Verlag, 1985.