Exploring new Coloring Methods for Image Triangulations

Daan Goossens[1] , **Amal D. Parakkat**[1] , **Elmar Eisemann**[1]
[1]TU Delft

Abstract

Minimalistic, low-poly images have gotten more popular in recent years. But making these low-poly images by hand can be time intensive. For this reason, a lot of research has gone into how to triangulate an image (semi-)automatically. In previous works, only constant color and bilinear interpolation have been used to color the triangles in the triangulation. In this paper, different coloring methods have been investigated and showcased on a fixed mesh. They show some promising first results, but these methods can still be improved upon. When that happens, these coloring methods, combined with previous works on how to optimally triangulate, may result in more accurate or visually pleasing low-poly images.

1 Introduction

Abstract and minimal art styles are getting more popular lately, where for example company logos get simplified each generation. Low-poly images, images that consist of few colored polygons, most notably triangles, are also gaining popularity. An example of a low-poly image can be seen in Figure 1. These art styles can easily be converted to a vector graphics format, because they mostly consist of simple shapes. This has the advantage that it can be scaled without loss of clarity or detail. This is important, because screen resolution are increasing constantly. The simplicity also has the advantage that it is easier to edit as there are less control points.

The low-poly style originated from early video games, where 3D scenes had to be limited in the number of triangles, so that games could run in real-time with a reasonable frame rate on the hardware available at that time. Nowadays, the low-poly style has become more of an artistic choice than the result of hardware constraints, as current computer hardware can render scenes with millions of triangles in real-time. The low-poly style resurfaced in 2D art and video games in recent years. In the 2D art medium, the main focus is to make the low-poly images as visually pleasing as possible by retaining the structure of the image with the minimum amount of triangles necessary. According to "Artistic Low Poly rendering for images"[1], handmade low-poly art, usually achieves this by having big triangles in the background and smaller and more densely packed triangles at places where the main subject of the image is. These images mostly use fat triangles, because that is more visually pleasing than thinner triangles.

Figure 1: Example of low-poly art. Source[1]

Making these low-poly images can be seen as two actions, namely making the mesh, which is a collection of triangles which are in some way connected together, and coloring the triangles in the mesh. As making a mesh, which can be colored to well represent an image, can be time intensive when done by hand, a lot of research has already been done to make this process automatic or semi-automatic. This will be discussed in more detail in Section 2. These works only make use of constant coloring of the triangles and sometimes bilinear interpolation. Therefore, in this paper, new coloring methods will be explored, and how they compare to the already widely used constant color and bilinear interpolation methods will be discussed.

The paper is structured as follows. Section 2 will discuss the research that has already been done on this subject. After which, Section 3 will give a minimal explanation of some important terminology used throughout the paper. The setup to make these low-poly images, and the different

[1]https://pixabay.com/nl/illustrations/ijsvogel-laag-poly-lowpoly-tekening-1458734/

coloring methods that are explored, will be discussed in Section 4. Section 5 will show the resulting images that are generated with these new coloring methods and how they compare against the widely used constant color and linear interpolation methods. After which, in Section 6 the significance of these results will be discussed. The concluding remarks and possible future work can be found in Section 7.

2 Related Work

There has already been some research done in the area of generating low-poly images. I will first discuss some academic research and then some non-academic programs that can be found on GitHub[2]. Results sampled from these works can be found in Figure 2.

2.1 Academic Research

In "Artistic Low Poly rendering for images"[1] an algorithm is introduced to automatically make low-poly art. They use edge detection for vertex placement and centroidal Voronoi tesselation steered by an image saliency map to have big triangles in the background and a more dense triangulation in the foreground elements.

"Pic2Geom"[2] also uses edge detection and a saliency mask for vertex placement. Since visual saliency does not work well on faces, they used face detection to extract facial features to better triangulate faces.

"Low-poly image stylization"[3] uses abstraction and segmentation of the image to place the vertices and then uses a Delaunay triangulation to produce the low-poly artwork.

"Low-Poly Style Image and Video Processing"[4] implements a faster algorithm to make low-poly art by writing a GLSL program that can be run on the GPU. With this faster implementation, they can render low-poly videos in a reasonable amount of time. The resulting low-poly video has some jittering problems, which they solved by using the vertex placement along the edges of image features of the previous frame, which have a chance to be preserved in the next frame.

In "Stylized Image Triangulation"[5] instead of using edge detection to place vertices, they have an error function that returns how close the colored triangulation is to the original image. They then use gradient descent to move the vertices to places where the Delaunay triangulation minimizes a given error metric over the whole image. This method produces results which get very close to the original image, in contrast to other papers, which focus more on how to make a visually pleasing low-poly images. They also implemented low-poly video, where for each frame they use the vertex placement of the previous frame as an initial starting point to calculate gradient descent on.

In "Triwild"[6] they use triangles that can curve to fit edges, which results in fewer triangles to triangulate curvy features with better accuracy. This has also proven useful in simulation work cases.

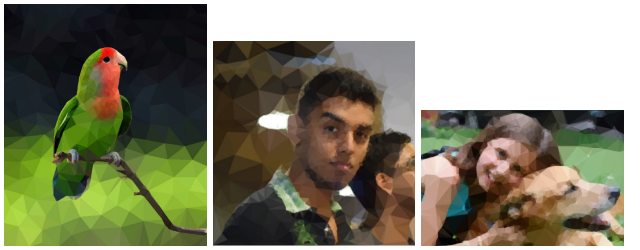
[2]https://github.com/

2.2 Non-academic Projects

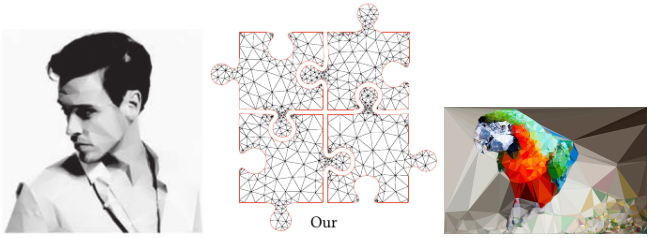
"Triangle"[7] uses edge detection to place vertices and then computes a Delaunay triangulation. The images look nice, but it sometimes has problems with thin triangles (see Figure 2f near the top left), which do not look as visually pleasing as other works.

"Triangula"[8] uses a genetic algorithm to decide where to place the vertices of the triangulation. It results in a triangulation that very closely represents the original image. The algorithm does tend to produce some thin triangles (see Figure 2g at the back wings) and might take a few minutes to complete triangulating a single image.

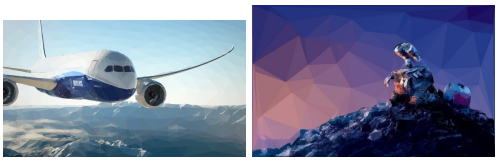
"lowpolify"[9] is very similar in approach to pic2geom, in that they both try to better triangulate faces when they are detected in images. Furthermore, it uses edge detection to place vertices after which Delaunay triangulation is computed, and the triangles are then filled in with a constant color.



(a) Artistic Low Poly rendering for images (b) Pic2Geom (c) Low-poly image stylization



(d) Stylized Image Triangulation (e) Triwild (f) Triangle



(g) Triangula (h) lowpolify

Figure 2: Example results from related work

2.3 Concluding Thoughts

It can be seen that a lot of previous work already went into how to triangulate an image, and the results already look visually pleasing. In the previous works studied, mostly constant color was used to color the triangles by either selecting the average color, median color or the color in the middle of the triangle. Sometimes bilinear interpolation was also used. For this reason, the question if better results can be achieved by using different coloring techniques, will be investigated.

3 Background

Here, some definitions will be given for some recurring terminology used in the paper.

3.1 Visual saliency

Saliency defines how much something stands out from its close neighbors. With images, that applies to pixels, whereas, visual saliency defines how much something stands out in terms of human perception. For example, in Figure 1 the bird itself stands out visually as that is the main focus of the picture, whereas the background is comparatively less important. An example of a visual saliency map can be found in Figure 14b.

3.2 Delaunay Triangulation

A Delaunay triangulation produces a triangulation for a given set of vertices, with the property that for each triangle, the circle through the three vertices has no other vertices inside it (see Figure 3). This maximizes the minimum angle over all triangles in the triangulation, which results in triangles that are as fat as possible. These fat triangles mostly look more visually pleasing in low-poly art, which is why they are used a lot in existing low-poly image generators.

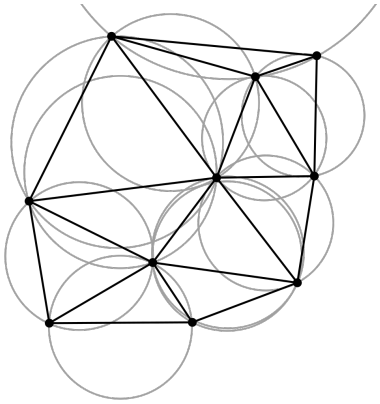


Figure 3: Visual definition of Delaunay triangulation. Source[3]

3.3 Barycentric Coordinates

Barycentric coordinates will be used for interpolating color values inside a triangle. A general definition can be seen in Figure 4, where vectors P, A, B, C can be of any dimension, but in this paper only the 2D case will be used. The barycentric coordinates also have some other convenient properties, namely for a point to be inside the triangle, the weights defined in Figure 4 need to be non-negative and sum to 1. If this is not the case, the point is not inside the triangle.

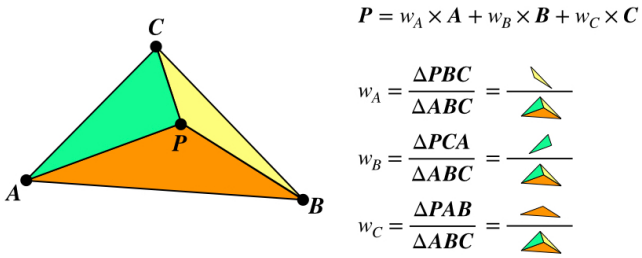


Figure 4: Barycentric coordinates definition. Source[4]

3.4 Bézier Triangles

A Bézier triangle is a triangle that can deform and interpolate between different control points on its surface. There are multiple degrees of Bézier triangles from degree n=1, which corresponds to linear interpolation (see blue triangle in Figure 5), to larger n. The number of control points of a Bézier triangle dependents on the degree n, which can be found with the formula $(n+1)(n+2)/2$. The general Bézier triangle is formulated in Equation 1, where (s,t,u) are the barycentric coordinates, the fractional part are the coefficients of each control point (see Figure 5), and (α,β,γ) describes the position of the control points.

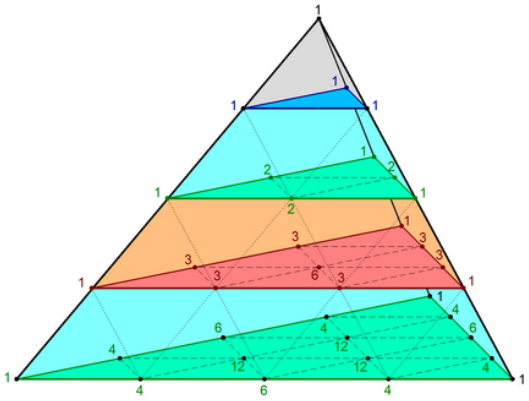


Figure 5: Pascal's pyramid for multiple degrees of n. Blue triangle n=1, red triangle n=3. The numbers on the triangle are the coefficients for each control point. Source[5]

$$\sum_{\substack{i+j+k=n \\ i,j,k\geq 0}} \frac{n!}{i!j!k!} s^i t^j u^k \alpha^i \beta^j \gamma^k \tag{1}$$

4 Method

Two different types of coloring methods were implemented, namely methods that split triangles up into two parts that can be colored individually, and interpolation methods, which try to get as close as possible to the underlying image. The splitting methods were implemented to try out if sharp outlines

[3] https://en.wikipedia.org/wiki/Delaunay_triangulation#/media/File:Delaunay_circumcircles_vectorial.svg

[4] http://wanochoi.com/?p=4170

[5] https://en.wikipedia.org/wiki/Pascal%27s_pyramid#/media/File:Pascalsche.Pyramide.png

on objects can be realized even with a suboptimal triangulation. The interpolation methods were implemented to build on some previous work, which try to get as close as possible to the original image, which can be improved by these more accurate interpolation techniques. To demonstrate the different coloring methods, a program has been made that takes as input an image and produces a low-poly image using these new coloring methods. The current implementation only accepts square images. This decision was made for simplicity, but it could easily be extended to accept any image size. As this paper is focused on the coloring methods, a fixed triangulation will be used. The implementation function as follows. First, an input image will be provided. Then a visual saliency map and edge map will be calculated from the image, with the use of the OpenCV library[6]. From these intermediate steps, the selected coloring method, and additional parameters, the low-poly image will be calculated. This standard pipeline is shown in Figure 6.

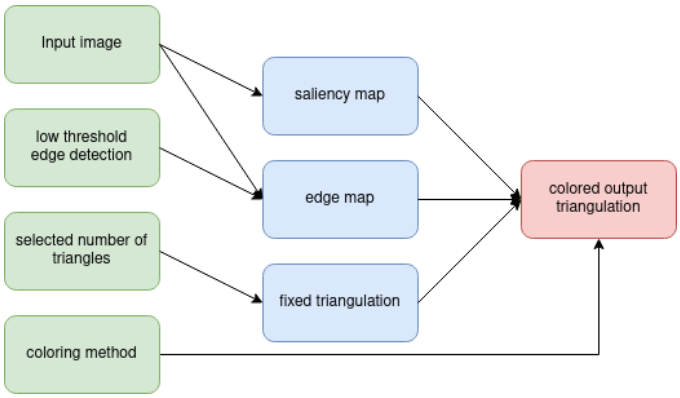


Figure 6: The standard pipeline for generating the low-poly images

4.1 Constant Color

Constant coloring can be described by the Equation 2.

$$f(t) = c \tag{2}$$

Where t defines the triangle in the mesh and c is basically a constant color for each triangle separately. As discussed in Section 2, many strategies exist to color with constant color. Two methods were implemented, namely taking the average color of all pixels inside the triangle and using visual saliency to get the color for each triangle.

The visual saliency option is implemented to test if it can also have a role in coloring, instead of only using it for placing the vertices for the triangulation. To do this, Equation 3 will be minimized over all pixels in the image. The equation basically describes a weighted error, where the weights are the saliency values. This calculates the absolute difference between the input and output pixel color at location (x, y), multiplied by its saliency value, which denotes how visually important that pixel is. Since the saliency value is in the range of [0, 1], a small bias is added, so that no color values are totally discarded. To minimize this equation, for each triangle the weighted average is taken over all pixels inside the triangle, where the weights are the saliency values of the corresponding pixels.

$$\begin{aligned} E(x,y) = &|image(x,y) - chosen\_color(x,y)| \\ &\cdot (saliency\_map(x,y) + b) \end{aligned} \tag{3}$$

4.2 Bilinear interpolation

For bilinear interpolation in this context, the barycentric coordinates, explained in Section 3.3, have to be used to color the triangles. Bilinear interpolation using barycentric coordinates is formulated in Equation 4. Where (s, t, u) represents the barycentric coordinates and (c_1, c_2, c_3) represent the colors of each vertex, which is sampled from the original image at these vertex coordinates.

$$f(s,t,u) = s \cdot c_1 + t \cdot c_2 + u \cdot c_3 \tag{4}$$

4.3 Linear Split

The Linear Split method can be described for each triangle by Equation 5.

$$f(x,y,a,b) = \begin{cases} coloring\_method1 & \text{if } a \cdot x + b >= y \\ coloring\_method2 & \text{else} \end{cases} \tag{5}$$

Where (x, y) are the Cartesian coordinates of the pixel and (a, b) are the parameters for the best fit line. This basically says that each triangle can be split up into two parts, with the use of a straight line. Where the part above the line is colored by coloring_method1 and the part below the line is colored by coloring_method2. The line basically functions as a mask that differentiates two parts of each triangle. Three ways were considered on how to get the parameters (a, b) of the best fit line.

1. Use an error function for some chosen coloring methods and some random starting parameters (a, b). Then use gradient descent on these parameters to find a local minimum for the chosen error function.
2. For each triangle use 2-means clustering to split the triangles pixels into two clusters, where pixels with colors close to each other are clustered together. Then find the line that splits the clusters up as much as possible.
3. Get the edge map of the image by first applying a bilateral filter to filter out the textures and then using Canny edge detection to find the edges. After which for each triangle the best fit line is calculated through the found edge points in the triangle. If there are not enough edge points inside a triangle, then use only one coloring method for it.

In the end, option 3 was chosen for its simplicity and because the other two options probably suffer from being computationally more demanding.

4.4 Quadratic Split

Quadratic split is the same as the linear split, but instead of fitting a line to the edge points, it fits a quadratic equation to

[6]https://github.com/opencv/opencv

it. This coloring method is formulated for each triangle by Equation 6.

$$f(x, y, a, b, c) = \begin{cases} coloring\_method1 & \text{if } a \cdot x^2 + b \cdot x \\ & \quad + c >= y \\ coloring\_method2 & \text{else} \end{cases} \tag{6}$$

This is implemented the same way as linear split (Section 4.3), in the way that it gets an edge map of the image and then finds the best fit to the found edge points in the triangle. The only difference is that it finds the best fit approximation for a quadratic equation instead of a linear one. In principle any degree of polynomial can be used for the best fit, but as a quadratic function can already approximate most curves well, making the degree of the polynomial higher will have diminishing results.

4.5 Interpolation with Bézier Triangles

All degrees of interpolation will be done with the Bézier triangle model (see Section 3.4), which can be formulated by Equation 1 for each triangle. The colors of the control points are optimized as a best fit problem, where the data points are the pixels inside the corresponding triangle. For each of those pixels, its color value and barycentric coordinates (see Section 3.3) are collected. This data is then fitted with the help of the gsl library[7] to the Bézier triangle model for the chosen degree of interpolation. Here only degree n=1 through n=4 will be looked into (n=1 for bilinear interpolation, n=2 for biquadratic interpolation, n=3 for bicubic interpolation, and n=4 for biquartic interpolation). Degree n=5 and up is not looked into as the number of control points goes up quadratically as n goes up, and the returns of how good the fit is, becomes less significant. For all degrees of interpolation, the best fit is gotten for every triangle individually, so no control points are shared between triangles. Note that, the bilinear interpolation here differs from the one discussed in Section 4.2, because in that method it does not find the best fit, as the colors of the control points are selected directly from the original image at those locations.

5 Results

All results are generated on four input images, which can be seen in Figure 14a. Some algorithms also need saliency maps (see Figure 14b) or edge maps (see Figure 15a). The edge maps were generated using user input by changing the lower threshold parameter of the Canny edge detector. This was done by eye to get the best possible results. Also keep in mind that the low-poly images are generated on a fixed mesh of either 28×28 or 52×52 triangles, so they may not look as visually pleasing as the results from previous works (see Figure 2). For this reason, the coloring methods that are used in those papers, are also implemented on the same fixed mesh to get a fairer comparison. These baseline coloring methods are constant color (see Figure 14c and 14d) and bilinear interpolation (see Figure 14e). One thing the constant color method can still be improved upon, is the zigzag artifact near the edges. This effect can be seen in Figure 7. According to "Artistic Low Poly rendering for images"[1], this occurs when a triangle covers two parts of an image with large difference in luminance, which they solved by first sorting all pixels inside the triangle by luminance and then take the average of the median values. This can also be solved by having a better triangulation, where one triangle does not cover two areas with large differences in luminance. Only a subset of the resulting images will be shown in this section. All resulting images can be found in Appendix B.

[7] https://www.gnu.org/software/gsl/doc/html/lls.html

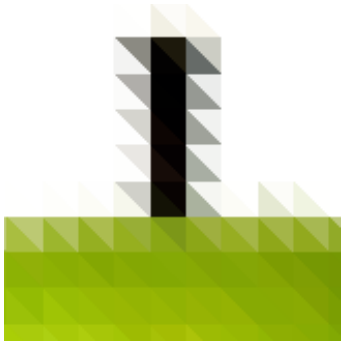
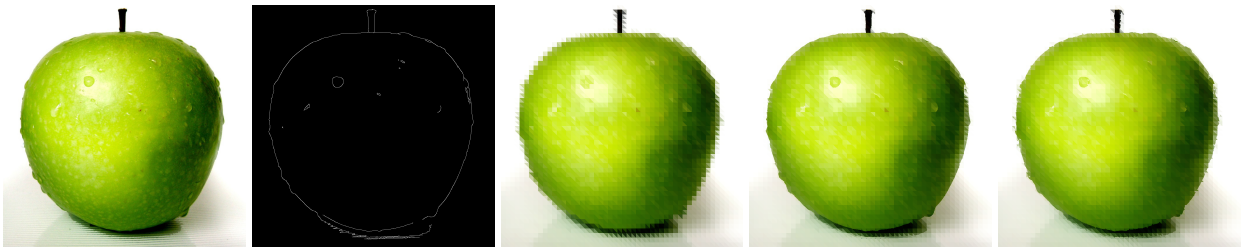


Figure 7: Example of the zigzag artifact. Zoomed in on the apple stem of the image produced with a 52×52 grid with constant color and no saliency

5.1 Visual Saliency

Figure 8: Constant color computed on a 28×28 grid, left=no saliency; right=saliency

In Figure 14c and 14d two constant coloring methods are compared. One using the average color over all pixels inside the triangle and the other using the weighted average of all those pixels, where the weights are the corresponding saliency values. The results are not super impressive, as they all look almost identical. This is also confirmed by the MSE (see Figure 13) between the two methods, as they are very close to each other for all input images. The only observation that can be made is that the images made using visual saliency pop out more, and the images made with the other method look a little more monotone. This makes sense as salient objects in the image have higher saliency values/weights, so these objects will be more defined. This effect can be seen in Figure 8 near the edges of the feathers. This effect is only barely noticeable, and was only noticed when flipping between the two methods.



(a) (b) (c) (d) (e)

Figure 9: Example of the splitting methods, where the images are produced on a 52×52 grid; (a) input image; (b) produced edge map; (c) constant color; (d) linear split with constant color; (e) quadratic split with constant color

5.2 Linear Split

(a) (b)

Figure 10: Example of the splitting method when the edge map is dense. (a) edge map; (b) linear split method with constant color on a 52×52 grid

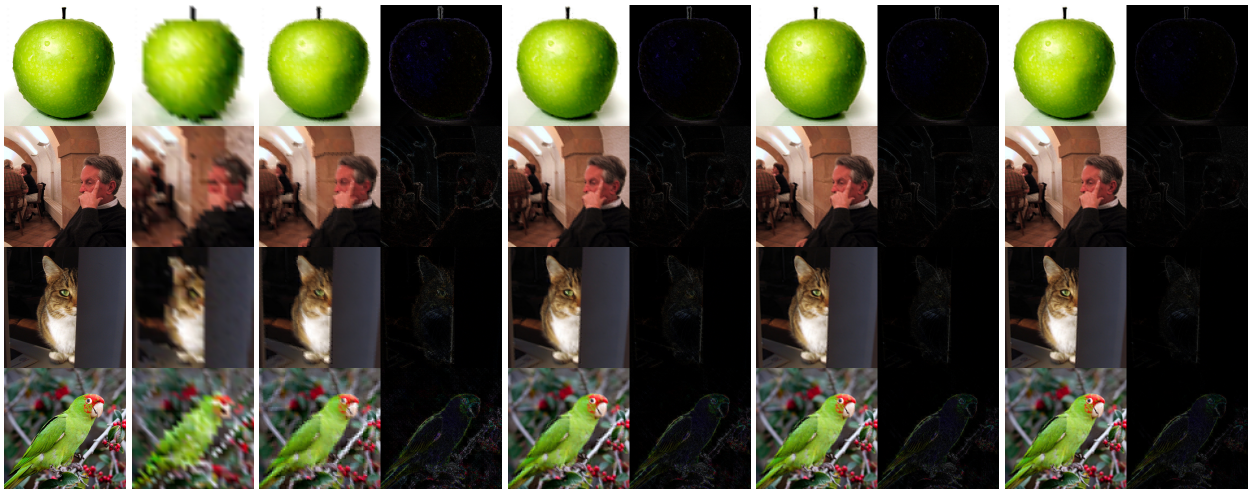
The results of the linear split method with constant color is shown in Figure 15b and 15d, with the corresponding absolute difference with the reference image. The absolute difference can help tell, at which parts of the image, the coloring methods struggles with accurately representing the reference image. Looking at the results, it can be noted that the method is excellent at approximating images which have a clear foreground element and not too much going on in the image, as can be seen in the apple image where the outline is pretty well approximated (see Figure 9d). It struggles more with images where there is a lot of detail, for example with the cats head and the lower body of the parrot (see Figure 10b). This problem can be traced back to the edge maps (see Figure 10a and 15a), where the edge map of the apple is simple and the edge map of the parrot and the cat can in some places be very dense. This denseness of the edge map is the main shortcoming of this method. If a triangle covers two or more distinct edges that go in different directions or an edge that splits up into two directions, it will have trouble accurately approximating a line that does not give weird artifacts in the image. This problem can be solved in multiple ways:

1. Having the edge map generation be more fine-tuned, so that it will not produce such highly packed edge maps.
2. Making the triangles cover a smaller area, which results in more triangles. This way, each triangle is less likely to have the scenario where it covers multiple distinct edges.
3. Optimizing how the mesh is triangulated, so that each triangle does not cover multiple distinct edges in the edge map.

The best solution is a combination of option 1 and 3, seeing as the results of this method are highly dependent on the edge maps and the main point of these low-poly images is to have a minimal number of triangles. From the MSE errors of this method and the constant color method (see Figure 13), it can be concluded that this method gets closer to the target image. On the surface-level, the difference in MSE does not look that impressive, but take into account that only a few triangles are split up into two parts (only when a triangle covers an edge) and the rest are filled in with the normal constant color (see Figure 9c and 9d). Also comparing the images next to each other, shows that this method does make the image look better when the edge map does not produce weird artifact, because of the problems described above.

5.3 Quadratic Split

The results of the quadratic split method can be seen in Figure 15c and 15e. The results are close to that of linear split method, which can also be confirmed by the MSE in Figure 13. From which, it can also be concluded that it performs slightly worse than the linear split method. In theory, this should be the other way around, as this method has more degrees of freedom and should be able to better approximate the image. The reason for this is, because this method suffers from some artifacts. This happens when the quadratic equation approximates a parabola and splits the triangle up into three parts instead of two. This can be seen at the edges or stem of the apple (see Figure 9e). This artifact can be solved by changing the model to fit the data to, as the quadratic formula inherently forms parabolas. For example, a spline with three control points, where two lie on different edges of the triangle and the last one defines the curvature of the spline, could solve this problem. When this artifact is solved, this method should perform better than the linear split method, but the smaller the triangles are, the less noticeable the difference becomes, as the approximated curvature is barely noticeable on smaller triangles. But on larger triangles this should perform noticeably better than the linear split method. Apart from this artifact, this method suffers from the same problems described in Section 5.2 and can also be improved upon in the same ways.



(a) (b) (c) (d) (e) (f)

Figure 11: (a) input images; (c/d/e/f) generated on a 28×28 mesh; left=resulting images, right=absolute difference with target image; (b) bilinear interpolation (without fitting); (c) bilinear interpolation (fitted to data) (n=1); (d) biquadratic interpolation (n=2); (e) bicubic interpolation (n=3); (f) biquartic interpolation (n=4)

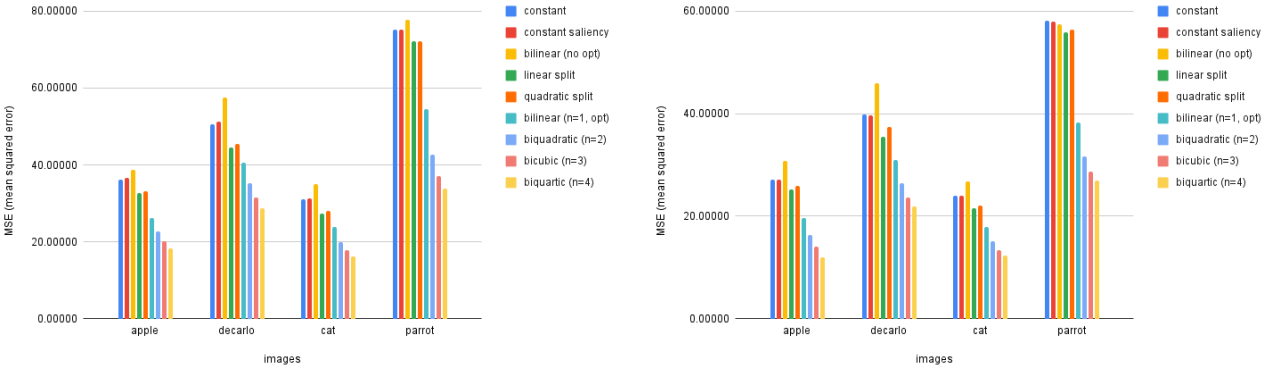
5.4 Interpolation with Bézier Triangles

The results of the interpolation method for different degrees of n on a 28×28 grid can be found in Figure 11. The results on the 52x52 grid can also be seen in Figure 16. Comparing the bilinear interpolation by selecting the color values at the vertex locations (see Figure 11b) and the bilinear interpolation by getting the color values of the control points by fitting it to the pixel data inside the triangle (see Figure 11c), already produces a much better result. This is also confirmed by the MSE found in Figure 13. For higher degrees of interpolation, the resulting image get closer to the original image, but the amount it gets closer by decreases. In contrast to that, the number of control points needed for higher dimensional interpolation increases quadratically, which results in diminishing returns for the data used compared to how close it gets to the image. So there is an optimum to be found for this. There are still three main problems present, namely:

1. The interpolation has trouble with sharp differences in color. For example in the absolute difference at the stem of the apple, which is still a good approximation, but it cannot do sharp edges, unless it is triangulated well or uses the bent triangles of "Triwild"[6] or maybe combined with the split methods discussed earlier (see Section 4.3 and 4.4). The opposite can be seen in the background of the parrot where it has a really easy time approximating the background even with a lower degree of interpolation or lesser number of triangles.
2. It cannot approximate large amounts of detail like the feathers of the parrot. So maybe that can be added back as a post-processing step, like in for example "Stylized Image Triangulation" [5] where line patterns and crosshatching or other styles were added back after triangulation.
3. The triangles will sometimes become visible in areas where the interpolation has trouble approximating the underlying image. This can be seen in for example the feathers of the parrot (see Figure 12). This happens when the control points on the vertices and edges of a triangle will differ noticeably between neighboring triangles. To mitigate this problem, a best fit can be done where the control points that are on the edges and vertices are shared between neighboring triangles. This does produce the problem that this will be a more difficult and slower best fit problem to solve, as not every triangle is done individually, but they are all done at once. Another solution is to have a better triangulation where every triangle does not cover an area with a color distribution that is hard to approximate or by using a higher degree of interpolation.

Figure 12: Example of the triangle artifact on the interpolation with Bézier triangles coloring method

Overall, this method is the most promising as it produces accurate results even with a fixed grid. If the problems described above are addressed in future work and the results are combined with a state-of-the-art triangulation technique, it will produce even more accurate results.



(a) (b)

Figure 13: the MSE of the resulting coloring method compared to the target image. (a) is rendered on a grid of 28×28 triangles and (b) is rendered on a grid of 52×52 triangles.

6 Discussion

Current research has been focused on how to triangulate an image well for mostly constant coloring. The coloring methods showcased in the previous section, show that using other coloring methods might be useful, but they still are in the early stages and further research might be necessary to improve on this. When this happens, the combination of these new coloring methods with previous research on how to optimally triangulate, might result in low-poly images which get closer to the reference image than previous methods, while using the same amount of triangles. If these techniques can get to a point where it can get very close to the original image, it might also find a use in image compression (especially the higher degree interpolation methods), as these low-poly images do not take much data to be stored, especially when using a fixed grid.

7 Conclusions and Future Work

In this paper, new triangle coloring techniques were explored for image triangulation. They were implemented on a fixed mesh and compared to commonly used coloring methods. These new coloring methods show some promise compared to these widely used coloring methods, but they can still be improved upon. Once that happens, combining these new coloring methods with existing research on how to triangulate images, might result in a more accurate or visually pleasing low-poly images.

There are a lot of opportunities to expand on these new coloring methods. Improved edge maps or saliency maps will produce better output images for the linear and quadratic split methods. Or figuring out how to fix the parabola artifacts of the quadratic split by fitting the pixel data to a spline model where two control points are on the different edges of the triangle and the other one controls the curvature of the spline.

Research improving these edge and saliency maps with the help of user feedback might also be interesting to explore.

More interesting further research might involve formulating an optimal triangulation for these new coloring methods, as these methods have some new constraints on what will give a good output image. For example, making sure that a triangle does not cover multiple distinct edges for the linear and quadratic split methods.

Something else that might be interesting to explore later on, when these techniques improve, is looking into how image triangulation with these new coloring methods fare as a compression method. Here the optimum degree of Bézier triangles and the number of triangles, can be explored, as both of these actions result in better approximations and take more data to store. Can different orders of Bézier triangles be mixed, by for example using a lower degree of interpolation on the background elements and a higher degree of interpolation on the foreground elements.

Taking inspiration from “Pic2Geom”[2], figuring out how extracted facial features can be used with these new coloring methods to improve the output image and how that compares against the result of this paper, might also be interesting to explore.

References

[1] M. Gai and G. Wang, “Artistic Low Poly rendering for images,” *VISUAL COMPUTER*, vol. 32, no. 4, pp. 491–500, Apr. 2016.

[2] R. Ng, L. Wong, and J. See, “Pic2geom: A fast rendering algorithm for low-poly geometric art,” in *ADVANCES IN MULTIMEDIA INFORMATION PROCESSING (PCM)*, 2017, pp. 368–377.

[3] T. Uasmith, T. Pukkaman, and P. Sripian, “Low-Poly Image Stylization,” *JOURNAL FOR GEOMETRY AND GRAPHICS*, vol. 21, no. 1, pp. 131–139, 2017.

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

[5] K. Lawonn and T. Guenther, "Stylized Image Triangulation," *COMPUTER GRAPHICS FORUM*, vol. 38, no. 1, pp. 221–234, Feb. 2019.

[6] Y. Hu, T. Schneider, X. Gao, Q. Zhou, A. Jacobson, D. Zorin, and D. Panozzo, "TriWild: Robust Triangulation with Curve Constraints," *ACM TRANSACTIONS ON GRAPHICS*, vol. 38, no. 4, June 2019.

[7] E. Simo, "Triangle," https://github.com/esimov/triangle, 2021.

[8] RH12503, "triangula," https://github.com/RH12503/triangula, 2021.

[9] ghostwritenr, "lowpolify," https://github.com/ghostwriternr/lowpolify, 2018.

[10] D. DeCarlo and A. Santella, "Stylization and abstraction of photographs," *ACM TRANSACTIONS ON GRAPHICS*, vol. 21, no. 3, pp. 769–776, July 2002.

A Responsible Research

Here, some ethical research practices will be discussed. Firstly, not all results will be positive, and in this paper there will be a transparent discussion on what works and what the limitations of the techniques used are. Secondly, the number of data points, namely the images, used are limited. These images were not cherry-picked and to ensure that, the source code to reproduce the results will be made available. This also ensures that the next research project building upon this work, integrating it in their own work or comparing it against their own work, can easily do so. The reader is also encouraged to run the code on their own images!

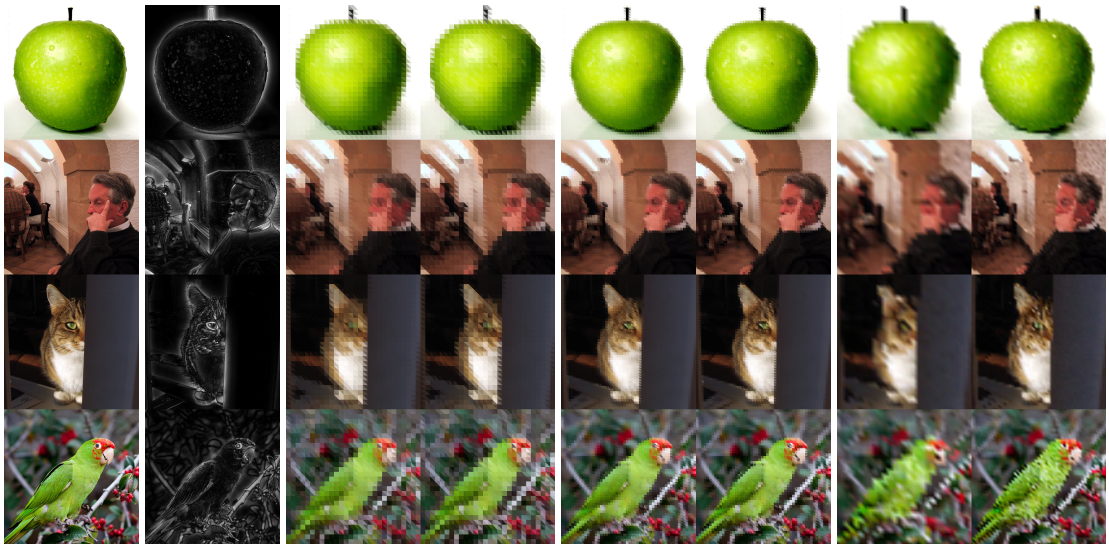
B All Image Results

[8] https://www.freeimages.com/nl/photo/tasty-serie-3-1329678

[9] "Stylization and Abstraction of Photographs"[10]

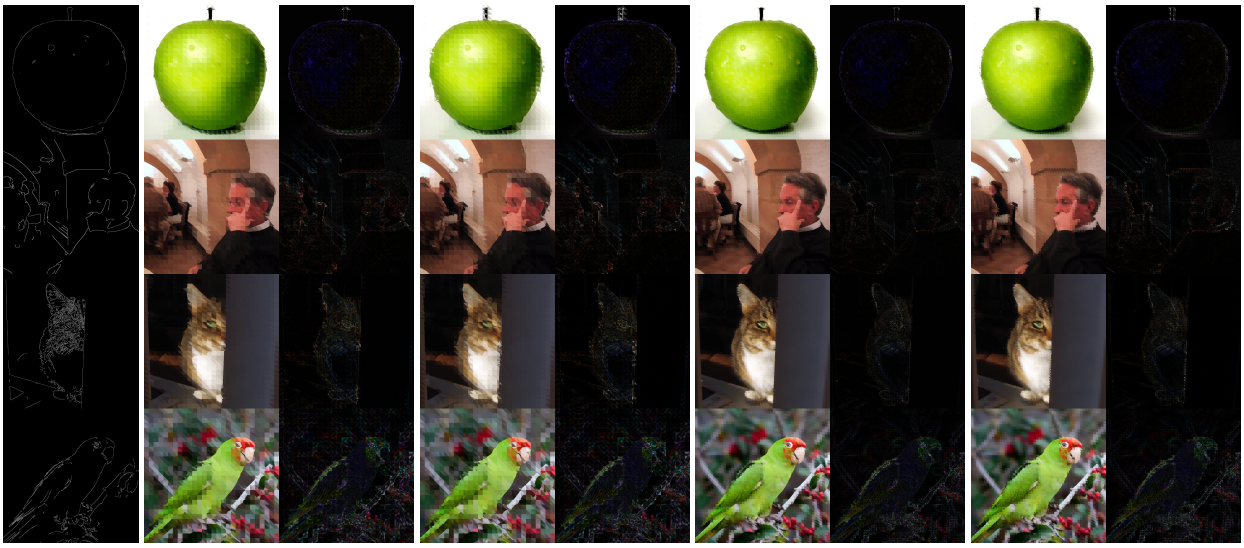
[10] https://www.flickr.com/photos/60168589@N00/414012553

[11] https://www.flickr.com/photos/51035555243@N01/4423082467



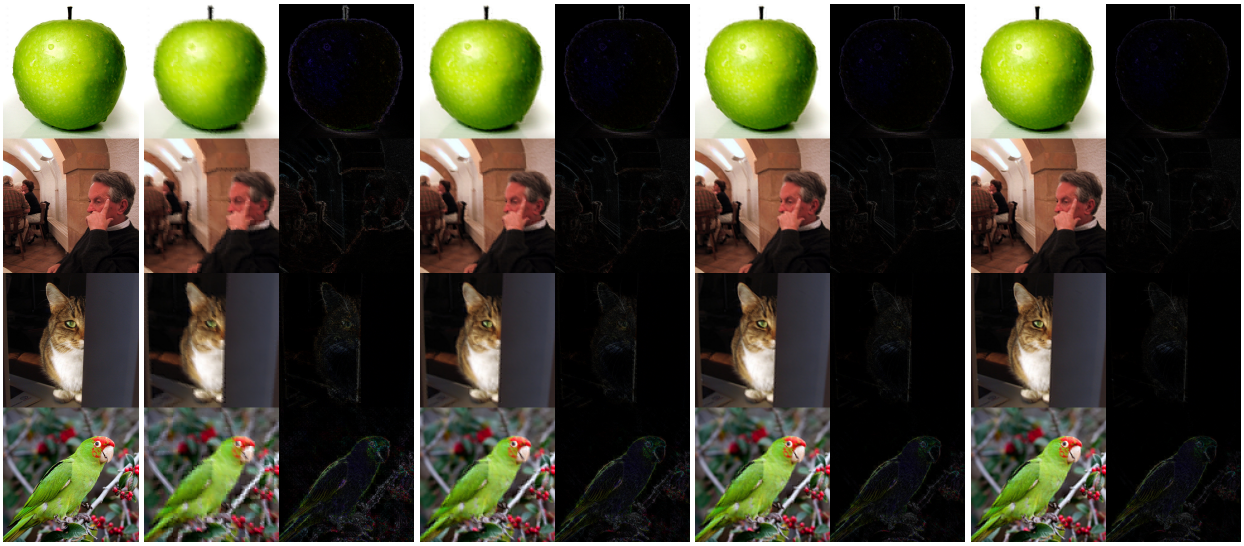
(a) (b) (c) (d) (e)

Figure 14: (a) input images[8][9][10][11]; (b) visual saliency maps; (c) 28×28 grid of triangles, left=without saliency, right=with saliency; (d) 52×52 grid of triangles, left=without saliency, right=with saliency; (e) bilinear interpolation, left=28×28, right=52×52

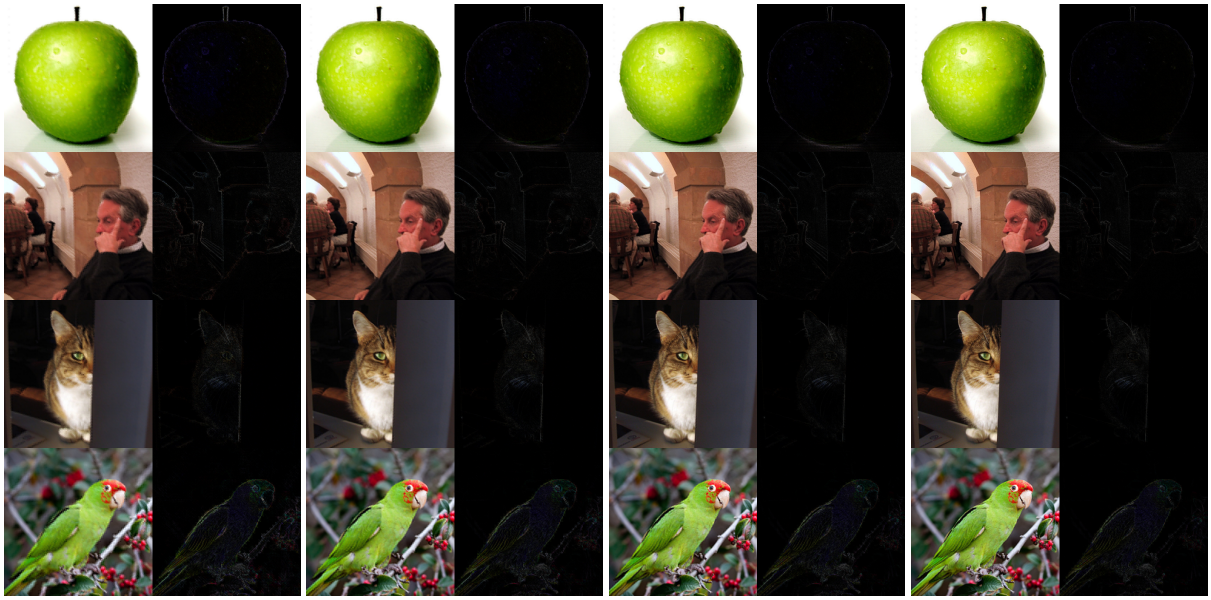


(a) (b) (c) (d) (e)

Figure 15: (a) edge maps used in these coloring methods (The edge maps are generated with threshold edge detection set to 59, 59, 21, 110 from top to bottom respectively); (b/c/d/e) left=resulting images, right=absolute difference with target image; (b) linear split (28×28); (c) quadratic split (28×28); (d) linear split(52×52); (e) quadratic split (52×52)



(a) (b) (c) (d) (e)



(f) (g) (h) (i)

Figure 16: (a) input images; (b/c/d/e/f/g/h/i) left=resulting images, right=absolute difference with target image; (b) bilinear interpolation (fitted to data) (n=1, 28×28); (c) biquadratic interpolation (n=2, 28×28); (d) bicubic interpolation (n=3, 28×28); (e) biquartic interpolation (n=4, 28×28); (f) bilinear interpolation (fitted to data) (n=1, 52×52); (g) biquadratic interpolation (n=2, 52×52); (h) bicubic interpolation (n=3, 52×52); (i) biquartic interpolation (n=4, 52×52)