Exploring new Coloring Methods for Image Triangulations

Daan Goossens – dgoossens@student.tudelft.nl
Dr. Amal D. Parakkat – A.D.Parakkat@tudelft.nl
Prof. Dr. Elmar Eisemann – E.Eisemann@tudelft.nl

1. Introduction

- Low-poly images are images that consist of few colored polygons, most notably triangles.
- It originates from early video games, when hardware resources were limited.
- In the last few years, this style is gaining more popularity in video games and 2D art, but now its more of an artistic choice.

Figure 1: low-poly art example source: https://pixabay.com/nl/illustrations/ijsvogel-laag-poly-lowpoly-tekening-1458734/

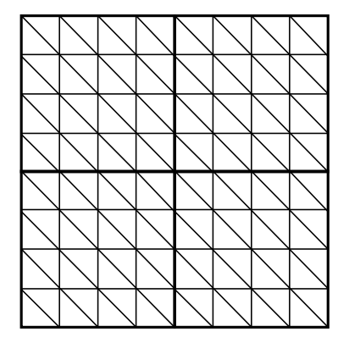
2. Objective

- There have been mainly 2 reasons to triangulate an image in the literature, namely.
  - Making the low-poly images as visually pleasing as possible.[1][2][3]
  - Getting the low-poly image as close to the original image as possible.[3][4]
- Research on automating these low-poly images has mainly been focused on how to optimally create a triangle mesh, which can be colored well to represent the image.
- Most of the literature use constant color to color in the triangles, and some also use bilinear interpolation.
- **Objective: explore new coloring methods for image triangulations, and figure out how they compare against the widely used constant color and bilinear interpolation methods.**

3. Method

Setup

- Problem simplified by:
  - Only accepting square images
  - Using a fixed mesh like shown on the right



Baseline Coloring Methods

- *Constant color*
  - $f(t) = c$
  - Take the average color of all the pixels in each triangle.
- *Bilinear interpolation (without optimization)*
  - $f(s,t,u) = s \cdot c1 + t \cdot c2 + u \cdot c3$
  - Get the colors of the control points by sampling them from the original image at those locations.

New Coloring Methods

- *Constant color with visual saliency*
  - Minimize error function: $E(x,y) = |image(x,y) - chosen\_color(x,y)| \cdot (saliency(x,y) + b)$
  - Saliency value between [0,1], so small bias needs to be added to not discard any pixels.
  - Takes the weighted average over the whole image, where the weights are the saliency values.
- *Linear split*
  - $f(x,y,a,b) = \begin{cases} coloring\_method1 & if\ a \cdot x + b \geq y \\ coloring\_method2 & else \end{cases}$
  - Make an edge map by first applying a biliteral filter and then doing Canny edge detection on it.
  - For every triangle that covers an edges in the edge map, find the best fit line to split the triangle in two. Otherwise use one coloring method for the whole triangle.
- *Quadratic split*
  - $f(x,y,a,b,c) = \begin{cases} coloring\_method1 & if\ a \cdot x^2 + b \cdot x + c \geq y \\ coloring\_method2 & else \end{cases}$
  - For every triangle that covers an edges in the edge map, find the best fit parameters for the quadratic equation to split the triangle in two. Otherwise use one coloring method for the whole triangle.
- *Interpolation methods with Bézier triangles*
  - $f(s,t,u) = \sum_{\substack{i+j+k=n \\ i,j,k \geq 0}} \frac{n!}{i!j!k!} s^i t^j u^k \alpha^i \beta^j \gamma^k$
  - Only degrees n=1 (bilinear interpolation) through n=4 (biquartic interpolation) are looked into.
  - For every triangle get the pixel color and barycentric coordinates from all pixels within it. Then find the Bézier triangle of a certain degree n that best fits that data.

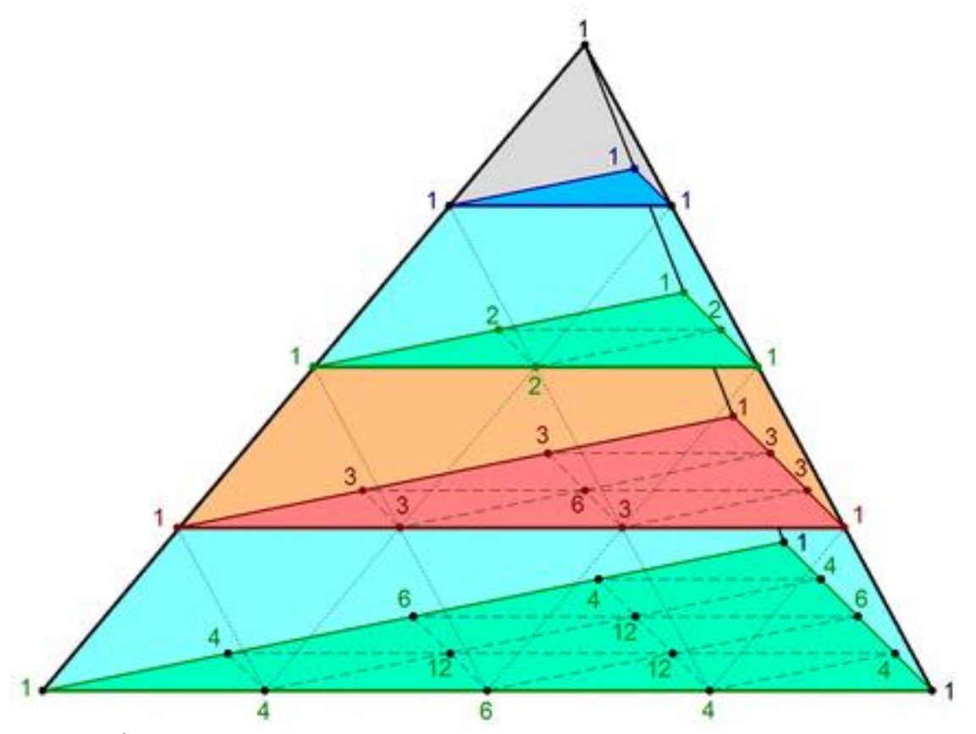


Figure 2: Pascal's triangle showing the control points and their corresponding coefficients for different degrees of Bézier triangles (n=1 bilinear interpolation is colored in blue)

4. Results

- *Constant color with visual saliency*
  - Difference barely noticeable.
  - Using visual saliency makes the image slightly pop.
- *Linear/quadratic split*
  - When the edge map is simple, this method can approximate the edges pretty well with this technique.
  - Dense edge maps can produce artifacts.
  - Quadratic split should in theory produce more accurate results, but in the current state it suffers from artifacts , namely it can split triangles up into three section by approximating to a parabola inside the triangle.
  - Now only constant color is used, but this technique can be used as a mask with different coloring methods.
- *Interpolation methods with Bézier triangles*
  - The bilinear interpolation (n=1) already produces better results than the baseline bilinear interpolation.
  - For higher degrees of interpolation, it gets closer to the input image, but that difference decreases with higher n.
  - The number of control points increases quadratically with the degree n. Combining that with the previous point, results in diminishing returns for higher degrees of interpolation.
  - Has sometimes problems with triangles being noticable when the interpolation has a hard time approximating the data points inside the triangle.
  - This happens when the control points at the vertices and edges between triangles don't line up.
  - This problem can be solved by either using a higher degree of interpolation or a better triangulation (so each triangle covers a area it can approximate well).

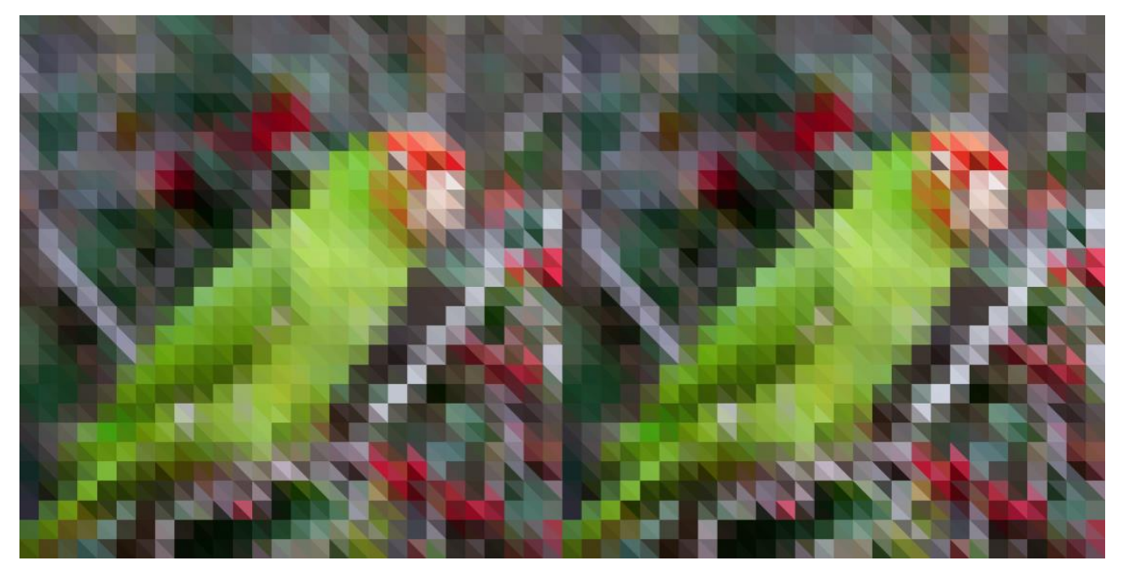


Figure 3: left=constant color; right=constant color with visual saliency

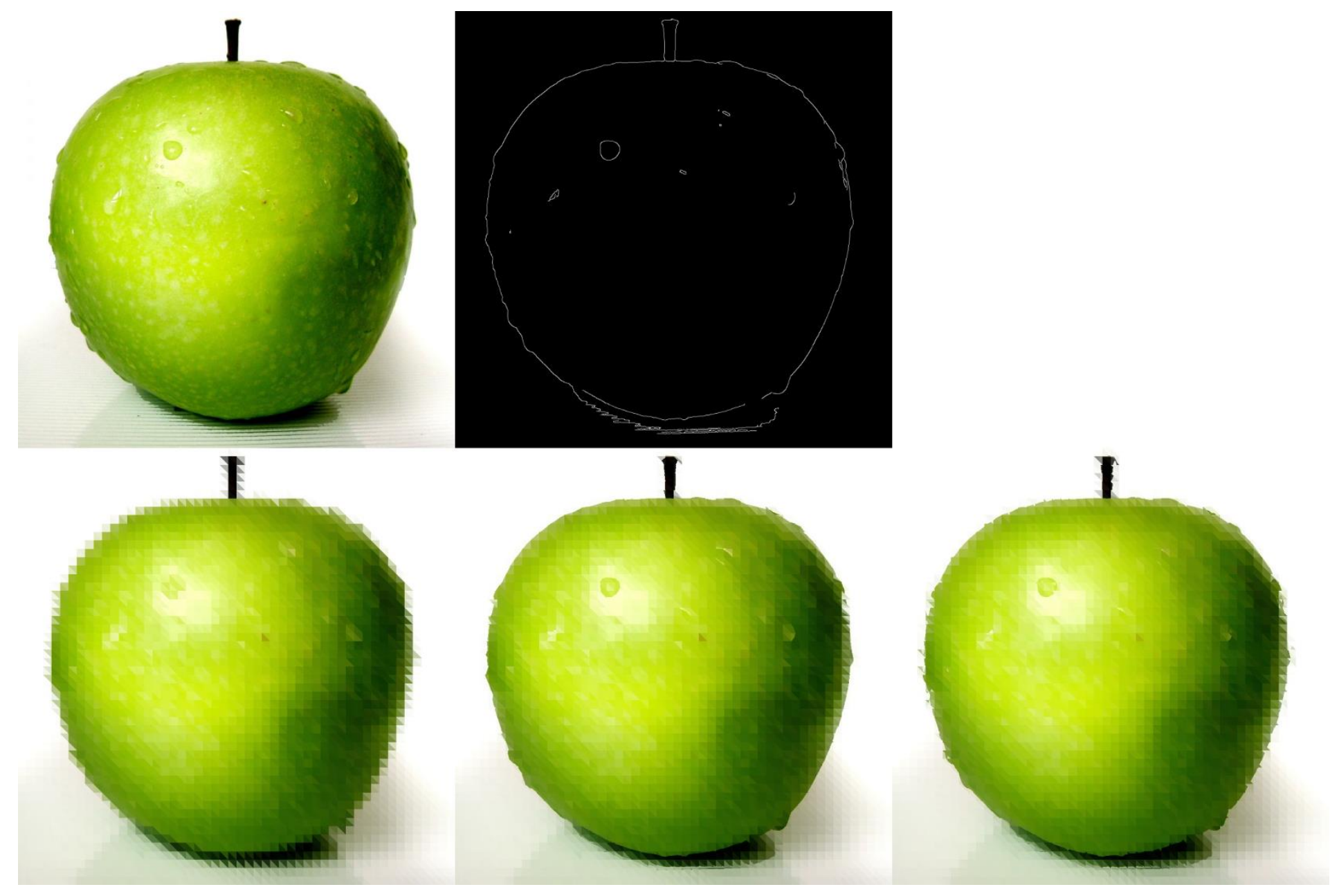


Figure 4: from left to right top to bottom; input image, edge map, constant color, linear split, quadratic split

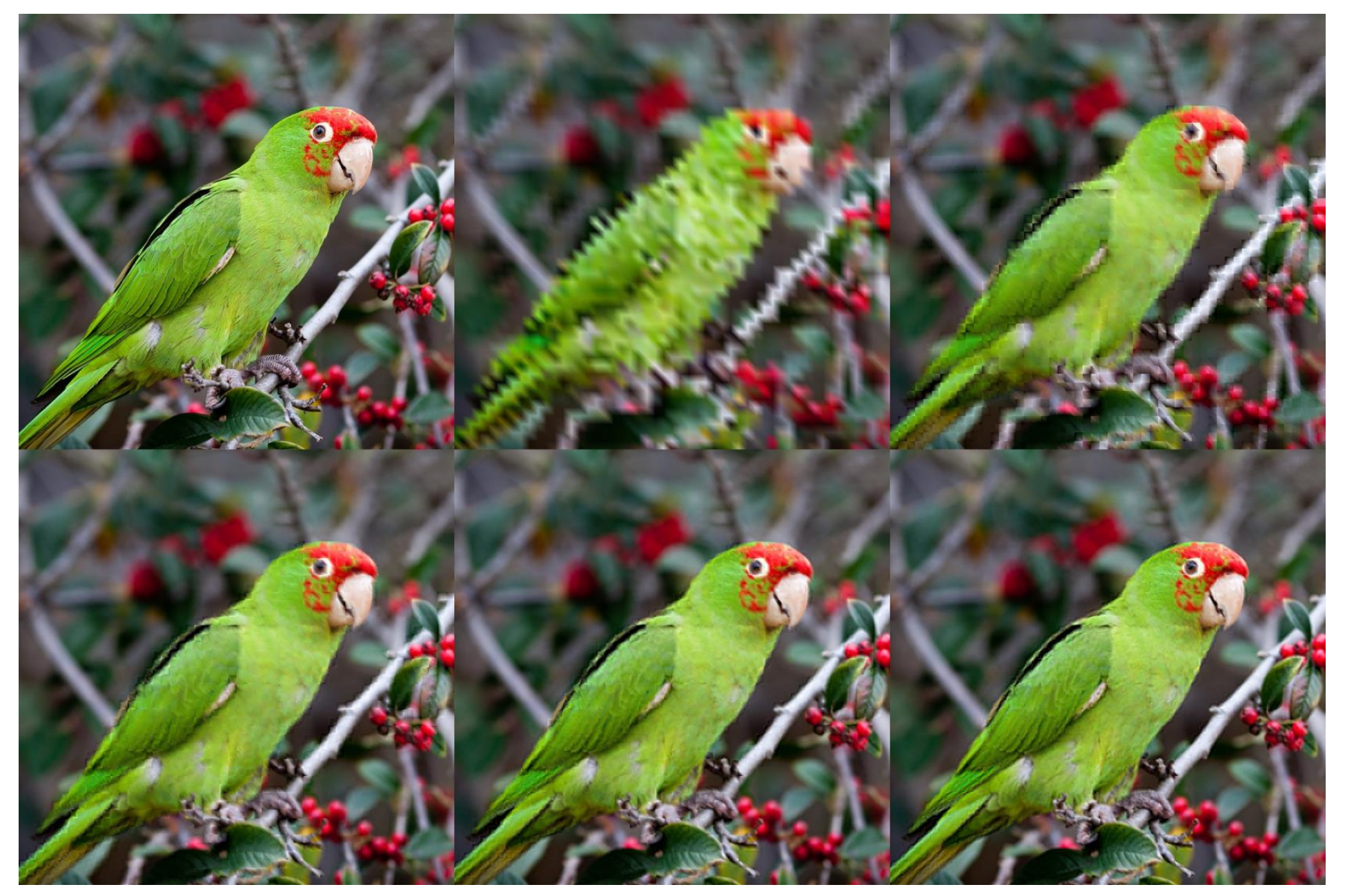


Figure 5: from left to right top to bottom; input image, baseline bilinear interpolation, optimized bilinear interpolation, biquadratic interpolation, bicubic interpolation, biqaurtic interpolation

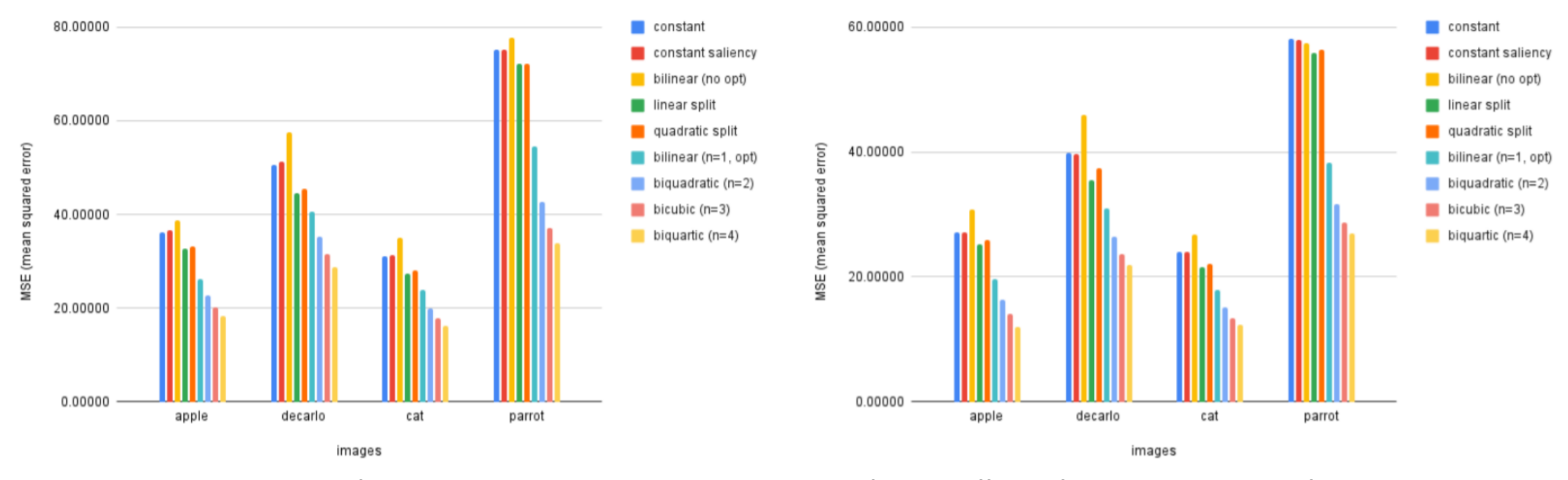


Figure 6: mean squared error for all the coloring methods and input images for two different fixed triangulations; left=28x28, right=52x52

5. Discussion and Conclusion

- The new coloring methods show some promise, but there is still a lot to improve upon.
- When the interpolation methods get more refined, they might find a use in compression.
- Combining these new techniques with existing triangulation methods will produce more accurate or visually pleasing results.

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 INFORMATIONPROCESSING (PCM), 2017, pp. 368–377
[3] K. Lawonn and T. Guenther, "Stylized Image Triangulation" COMPUTER GRAPHICS FORUM, vol. 38, no. 1, pp. 221–234, Feb. 2019.
[4] RH12503, "triangula" https://github.com/RH12503/triangula, 2021