A New Vision For Artistic Segmentation

Manav Rathod
University of California, Berkeley
manav.rathod@berkeley.edu

Jaiveer Singh
University of California, Berkeley
j.singh@berkeley.edu

Abstract

In this paper, we motivate the need for a new class of segmentation algorithms that partition images in ways that are artistically driven and aesthetically pleasing. Inspired by past work in computational parquetry, we present a novel algorithm to create a spatial feature-driven image segmentation that appeals to certain artistic forms and complements textural feature-based segmentation and matching. We hope that this contribution will motivate other researchers to explore how computer vision techniques can be leveraged to create art that transcends the digital into the physical.

1. Introduction

Recently, Iseringhausen et. al. [5] developed a pipeline for computational parquetry, the task of fabricating physical representations of images from patches of wood (Figure 1). Their work presents a greedy-inspired, tractable algorithm that makes the challenge of matching image patches to wood samples computationally feasible. To demonstrate the effectiveness of their algorithm, the authors introduced a separate, human-guided algorithm to segment images into the grid-like patch pattern of Figure 1.

While the initial results are promising in a proof-of-concept sense, we believe that there is potential to dramatically improve the patch generation process. First, the patch boundaries do not automatically conform to any spatial features in the image. Since the input wood material generally makes only smooth transitions in intensity, this non-edge aligned segmentation imposes a natural limit on how sharp any edge in the output rendering can appear. While Iseringhausen et. al. do include a mechanism for a human operator to manually adjust patch boundaries, we feel that this process is critical to the production of good results and thus should be fully automated.

Second, the semi-automatic process begins with a rigid grid, which constrains the segmentation results to relatively uniform-looking outputs and limits the overall artistic effect. Iseringhausen et. al. demonstrate how a completely custom segmentation would compare against their combination of algorithm and human input (Figure 2). However, they present no method for generating results closer to that custom segmentation.

In this project, we develop a brand-new patch generation process that automatically produces varied, aesthetically pleasing patches from an input image.

This line of work is similar to the familiar task of superpixel segmentation, for which there exist a variety of methods that have remarkable performance. However, a key distinction in this task is a need to be able to break down contiguous regions with similar in texture and style into smaller parts, a step that is necessary to make the wood texture correspondence search and physical wood cutting processes feasible.

Thus, we present a novel, efficient superpixel segmentation technique that produces a unique artistic effect and allows for tractable texture matching and fabrication.

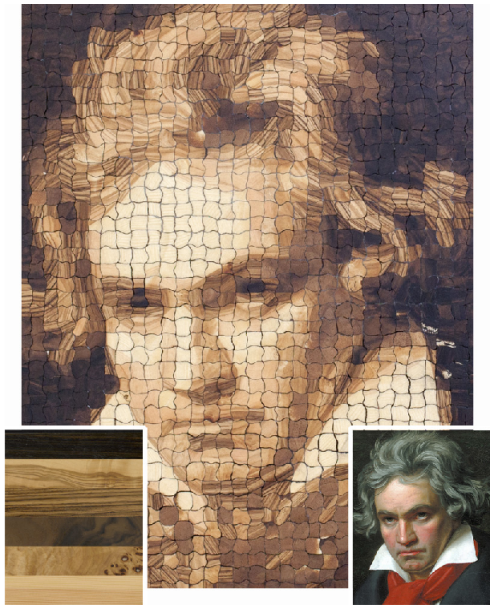

Figure 1. Image taken from [5] demonstrating outputs of computational parquetry.

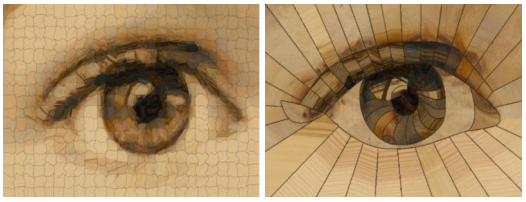

Figure 2. Images taken from [5] demonstrating varying artistic styles for segmentation. Left is a result of their semi-automated method. The right is a custom hand-designed segmentation.

2. Related Works

Image segmentation has been well-studied for several years, and so there exist a variety of different solutions for specific sub-problems in the field. Semantic segmentation aims to find all different classes of objects in the image. Instance segmentation goes a step further, assigning different labels to different objects belonging to the same class. Superpixel segmentation is another related field that group pixels into more meaningful perceptual regions, similar to semantic segmentation, but without classifying individual regions [6, 8]. Our objective in this paper is most similar to superpixel segmentation. In this section, we will describe the existing methods for superpixel segmentation, starting with the methodology used by Iseringhausen et. al. and then into a broader overview of the field.

2.1. Iseringhausen et. al Method

Iseringhausen et. al present a default superpixel segmentation that simply splits the image using a plain Cartesian grid. The authors note the limitation of such an approach and provide a mechanism to semi-automatically adjust that grid to fit to manually-selected features in the image, which is an a priori image-space grid morphing method. This applies an edge-preserving rolling guidance filter [12] to remove smaller image structure and then skeletonize the image by using the Canny edge detector [3]. To capture the higher frequencies, they use the bilateral filter [10] on the image and then once again extract edges with the Canny edge detector. Each edge image is masked individually and then the results are combined with a max operation. This final edge image defines a scalar potential field, which can be used to solve an energy minimization problem to snap the vertices of the grid to the edges in the final image. However, this is only semi-automatic as the user must manually select the regions of interest to apply filters and enable edge adjustment.

2.2. Superpixel Segmentation

The notion of superpixels was first introduced by Ren and Malik and has since gained considerable attention due to the more meaningful entities produced by this method compared against other image processing techniques. Stutz et. al [8] provide a comprehensive overview of this field, categorizing algorithms into one of the following classes: Watershed-based, Graph-based, Density-based, Contour evolution, Path-based, Clustering-based, Energy optimization, and Wavelet-based. Ren and Malik initially presented a graph-based approach that used a linear classifier to discriminate between good and bad segments, and then performed a random search following a Monte Carlo Markov Chain (MCMC) paradigm. This approach was functional yet slow, taking up to 30 minutes to process a 240x160 pixel image. More recent algorithms have significantly sped up computation to the order of tens of milliseconds while maintaining coherent segmentations. However, prior work in this area have only focused on segmentation as a precursor to further image processing algorithms, not as an artistic end goal in and of itself. Thus, many of these segmentations remain largely grid-like and are not aesthetically pleasing. Our goal in this paper is to take inspiration from these approaches but further optimize them for a visually-satisfying effect.

3. Methodology

Our algorithm is most similar to path-based algorithms, with a contour-informed methodology to inform the initial pixels that seed the segmentation. In this section, we will outline the steps of our algorithm.

3.1. Contour Detection

The first part of our algorithm takes the input image and detects all of the contours in the image. This is a critical deviation from the standard methods of segmentation, since we immediately locate the important objects and features in the image instead of beginning with a uniform grid prior.

To accomplish this, we read in the image as gray-scale and apply a thresholding operation to binarize the image; all pixel values less than or equal to half-brightness are set to 0, while values in the brighter half of the range are uniformly set to the maximum. Increasing the contrast in this way has experimentally produced the best contour results. We use the algorithm for contour detection developed by [9], which is conveniently implemented by the OpenCV library and available out-of-the-box [1].

3.2. Point Selection

The next stage of our algorithm is to select specific points to anchor the segmentation. Ideally, these points should be well-distributed throughout the image, to ensure that patches are never too big or too small. Our point selection process is further broken up into two main steps:

3.2.1 Contour Traversal

We first take advantage of the contours we detected in the previous step of the algorithm. A key insight of our work is that aligning patch cut boundaries with the edges inherent to the input image is essential for a sharp output. In order for the edges of the patches to line up in this way, it is necessary for the vertices of the patches to themselves also lie on the same contours, hence motivating the following approach.

At the outset, in order to reduce the noisiness in the output selection, we filter out any contours that have less than CONTOUR_THRESHOLD points. Additionally, we also treat the borders of the image as straight-line contours, preserve a sharpness along the image's perimeter and contributing to an overall framing effect in the output.

In order to step along a contour, we model it as a cubic spline using the SciPy library [11]. The spline is modeled as a parametric function, enabling us to easily identify pixel coordinate outputs given a value for the input parameter t.

We iterate from $t = 0$ to the length of the contour with a small ϵ step. At each step, we compute the pixel coordinates using the spline, rounding to the nearest integer values. Each of these rounded points is stored in order to enable pixel-accurate cuts between the two endpoints.

Additionally, one point in every STEP_SIZE steps is considered for a special set of control points across the entire image, whose relevance is subsequently discussed in section 3.3.2. Rather than blindly accepting all candidate control points, we instead verify that each point selected is sufficiently far from other existing points (MIN_DIST) and from the borders of the image (EDGE_TOLERANCE). EDGE_TOLERANCE is critical to avoid very narrow patches. If the candidate point passes both of these checks, it will be added to our final set of control points.

3.2.2 Harris Corner Selection

One drawback of the point selection algorithm stated thus far is that it requires manual tuning of the STEP_SIZE parameter; while useful as an extra degree of artistic freedom, this parameter is difficult to directly interpret. To remove the dependency on an arbitrary parameter, we employed Harris Corner Detection [4] with Adaptive Non-Maximal Suppression [2].

We apply the corner detection algorithm on a simplified image that contains only the original image's contours on a uniform background, with the goal of selecting only the most relevant points that mark sharp turns in the contour shape. However, simply using corner detection returns a very large set of points that also might be in very close proximity, which manifests in extremely small patches that are difficult to fabricate. To address this, we used Adaptive Non-Maximal Suppression to filter only for points that had high response value and were reasonably distanced from their neighbors. This does not provide a blanket guarantee that all points will meet the MIN_DIST requirement, so we still process each point in sequence and incrementally verify the constraint. A comparison of the outputs using the STEP_SIZE parameter and using Corner Detection will be presented in Section 4.2.

3.2.3 Random Point Selection

Thus far, the points contained in our control set will provide an excellent segmentation around the main features of the image, but entirely omit the large regions of the image that are essentially featureless. Matching and fabricating monolithic patches of this size is difficult, and so there is a need to automatically break up these large regions. Importantly, since these regions are mostly uniform, a random sampling of points is likely sufficient; there are no tricky features that will make patch matching difficult.

However, a naive, uniform random sampling runs the risk of essentially devolving to the grid-like segmentation our method aims to avoid. We observe that the typical image has its subject in the center of the frame, and so selecting points closer to the center will best capture the distribution of details. The natural choice is thus a Gaussian distribution for random sampling, where the mean is at the center of the image and the standard deviation is preserved as a parameter for the individual artist (GAUSSIAN_STD). We sample NUM_RANDOM_ATTEMPTS_THRESHOLD times, but verify each point with the same distance criteria defined earlier to avoid undesirable clusters.

A comparison between these sampling techniques will be presented in Section 4.1.

3.3. Segmentation

Now that we have a set of control points that are well-spread throughout the image and aligned to the main features of the image, we can begin the actual segmentation process.

3.3.1 Triangulation

We adopt a very straightforward approach by simply defining a Delaunay triangulation over the set of all points we have selected. We believe this to be an effective approach since it produces a well-distributed pattern of shapes given the representative points.

Furthermore, we are drawn to triangulation in particular because of the artistic and stylistic properties of triangles. Psychological studies in artistic perception show that humans look for familiar visual patterns that are explicitly organized [7]. This is directly achieved via triangulation, since the tiling of triangles is a strongly geometric pattern that most people have seen. Triangles are also the primitive

of choice in many computer graphics applications, providing the flexibility that creates a more dynamic and aesthetically pleasing output which is closer in spirit to the original custom segmentation of Figure 2.

3.3.2 Contour Fitting

Since triangles are bounded by straight line segments as edges, much of the information of the contours from the original image is lost in the initial triangulation. To address this, we replace some triangle edges with the actual contour segment between the two vertices, leveraging the saved pixel arrays from an earlier step.

We first filter for only the edges in the triangulation whose endpoints are both originally on the same contour that determine all of the points across all of the triangle edges that have contour points between them, which we stored as we traversed the contours in 3.2.1. A second check involves verifying that the new contour does not intersect or even approach any of the other edges in the triangulation. Intersections are clearly problematic in that they violate the mutually-exclusive nature of patches, but even close approaches to other edges would require extremely precise and narrow cuts that are essentially impossible with a conventional manufacturing process.

Generally, we expect a small yet clearly identifiable number of contour replacements during this step. The typical result is that the finished segmentation contains a clearly visible silhouette of the main subject.

4. Results and Discussion

Figure 3 demonstrates the various step of our algorithm on a classic image of Alan Turing. We are extremely pleased with the results, since the final output captures the dynamics and features of the input to an impressive degree, even with a relatively bland image of wood in Figure 9.

We also employ our algorithm on various other inputs as seen in Figure 7 and 8. The wood sample used for all of our experiments can be found in Figure 9.

In the rest of this section we will explore the effects of changing the various parameters we outlined in our methodology on the overall output.

4.1. Uniform vs Gaussian Distribution

In this section, we will compare how the outputs change when using a uniform vs. Gaussian distribution to randomly sample points as explained in 3.2.3. Comparing Figure 3 to Figure 4 and Figure 5 to Figure 6, we see that using a uniform sampling strategy results in a much more fine grained segmentation as there are more points used for the triangulation. The reason for this difference in points is that Gaussian sampling focuses mostly on points near the center instead of the surrounding background. There are two main results due to this difference.

First, having more points gives us a smoother background image, as smaller patches from the wood can be pieced together to match the texture of the background more precisely. The artistic value of this can be debated; the smoother background results in a polished aesthetic, but loses some of the wooden finish. We see this difference as a feature of the algorithm, as it gives some freedom to the artist to decide on to what degree the input material should represent itself in the final output.

Second and more practically, having more triangles in our segmentation strictly increases the likelihood of an acceptable match being found for each patch. However, this comes at the cost of a longer runtime for the matching process. This tradeoff between time and material constraints can also be leveraged by the artist.

4.2. Step Selection vs Harris Corners

In this section, we will compare using Harris Corner detection to naively selecting points according to the STEP_SIZE parameter. Comparing Figure 3 to 5 and Figure 4 to 6, we see that using Harris corners as points does an excellent job of capturing the outlines of the head. Even in the case of uniform sampling, where there are significantly more points, one can appreciate the clearer outline of the head when we use Harris corners.

We believe the main reason for this improvement is the constant nature of the STEP_SIZE parameter. Depending on how tightly wound a contour is, simply increasing the step size is not a good way of producing points that are more spatially distributed; consider an S-shape as an extreme example.

The difficulty in intuitively understanding this parameter leads us to prefer the Harris corner method for general use. Harris corners are effective in finding a good balance of useful control points and then adaptively suppressing points that are too close together.

5. Fun with Triangles

Beyond using our novel segmentation as step towards improved texture matching for computational parquetry, we can also use this segmentation as an artistic piece in and of itself. The triangle pattern of this segmentation is quite aesthetically pleasing, and it evokes a style of art similar to the work of Gregory Dubus (10). Inspired by his work, we adapted our method to automatically generate portraits in his style.

To do this, we use a simple flood-fill algorithm on the segmented triangulation to color in all of the patches in alternating shades of grey, as shown in Figure 11. Though the colors of this image are essentially random, the head of Alan

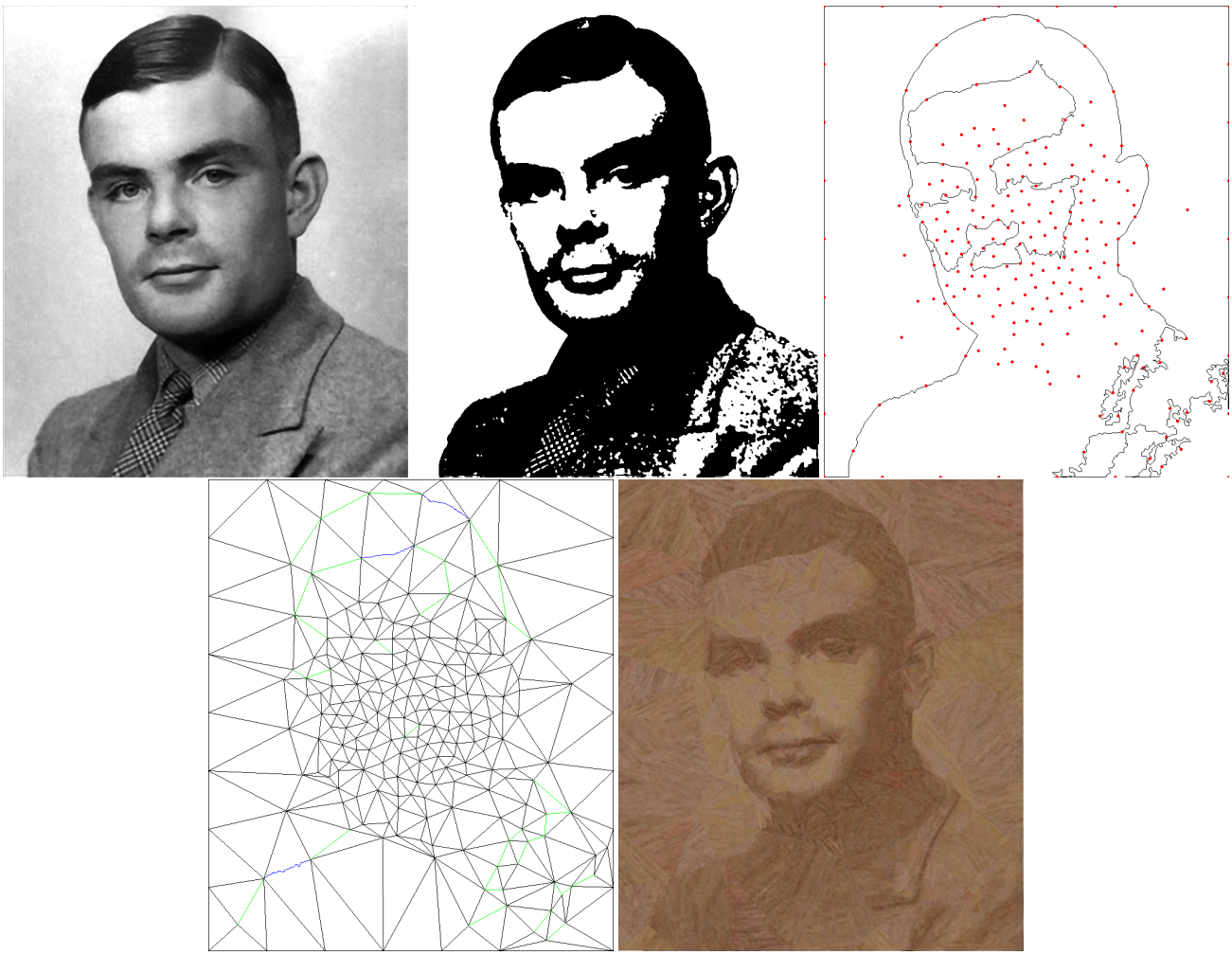

Figure 3. Overall pipeline of our algorithm. We start with the input image, then threshold it in order to create a binary image. We then take the binary image to find contours. We sample points along the contours as well as random points in the image. Then, we use these points to form a triangulation. The green edges represent straight edges that were chosen over contours and blue edges represent instances where we replaced straight edges with the existing contour. Finally, we take these patches and use the algorithm presented in Iseringhausen et. al to match the patches to wood textures. Here we used Gaussian sampling and traversed the contour for points without using Harris Corner detection.

Turing is definitively perceptible in our output. The gradients of gray across triangles also creates a dynamic contrast that keeps the image visually engaging to the viewer. The geometric familiarity of triangles also produces an aesthetically appealing effect.

We believe that the remarkable degree to which the head of the portrait emerges from the background is a testament to the importance of appropriately selecting patch boundaries, and more broadly a verification of the value of our contribution.

6. Conclusion and Future Work

In this paper, we motivate the need for a new class of segmentation algorithms that focus on artistic appeal by considering the final segmentation as an end product instead of merely an intermediate. To accomplish this goal, we designed a novel algorithm for superpixel segmentation that is rooted directly in the main spatial features of the image. We believe our results are already at an aesthetically pleasing level, but we believe that there is much room for future work.

One drawback of our current work is the lack of control of the final number of patches. Currently, this metric is a complex factor of the type of random point selection

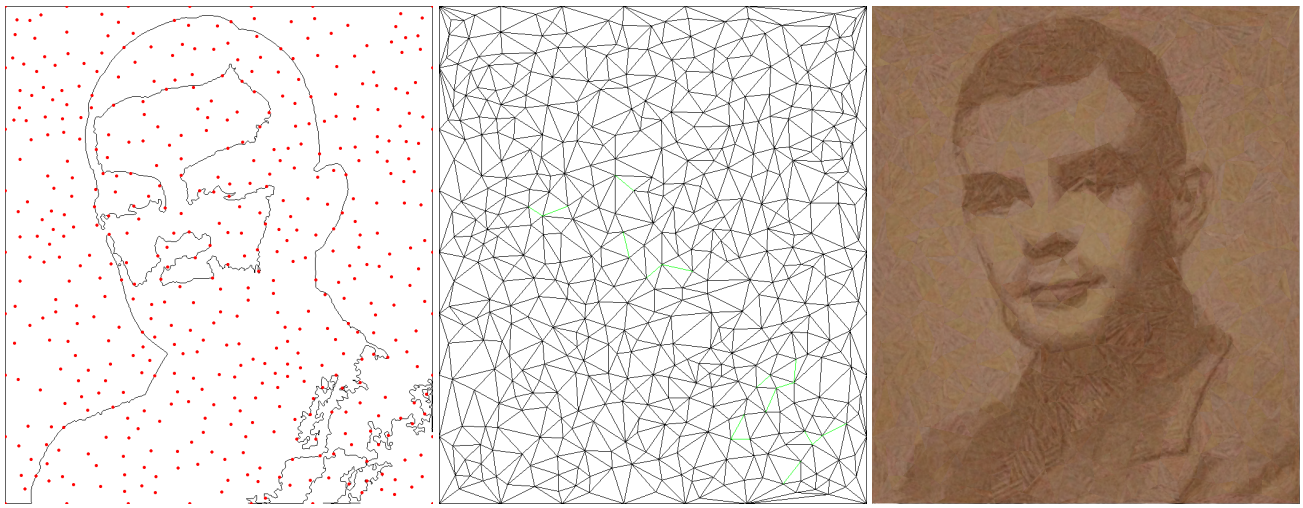

Figure 4. In this experiment, we run our algorithm using uniform sampling without Harris Corner Detection.

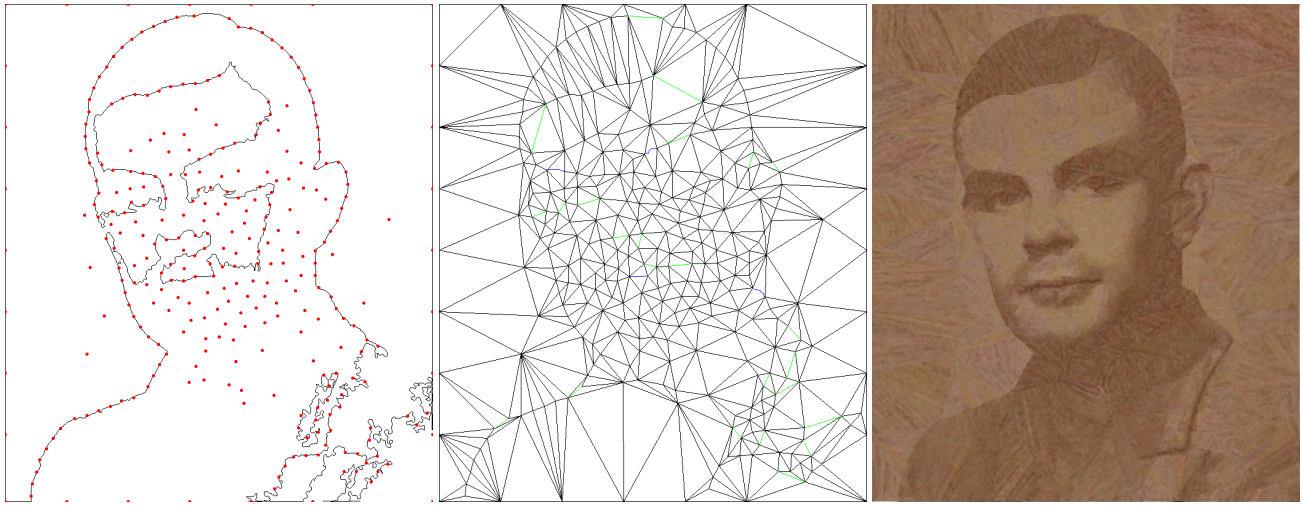

Figure 5. In this experiment, we run our algorithm with Gaussian sampling and Harris Corner Detection.

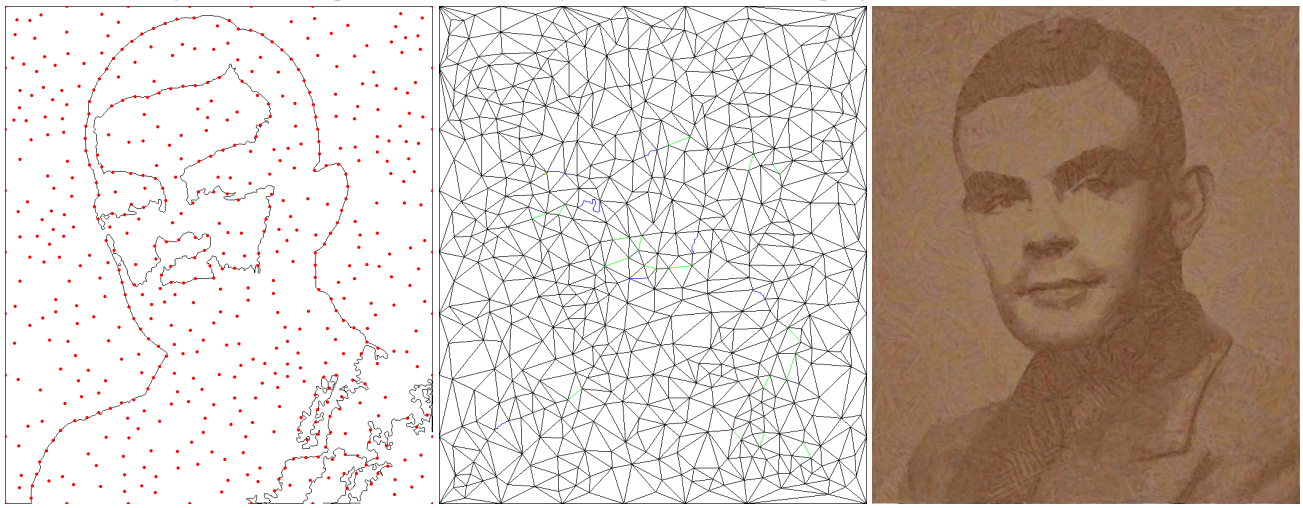

Figure 6. In this experiment, we run our algorithm with uniform sampling and Harris Corner Detection.

Figure 7

Figure 8

Figure 9. Wood texture used for experiments.

Figure 10. Resiste by Gregory Dubus. An abstract self-portrait.

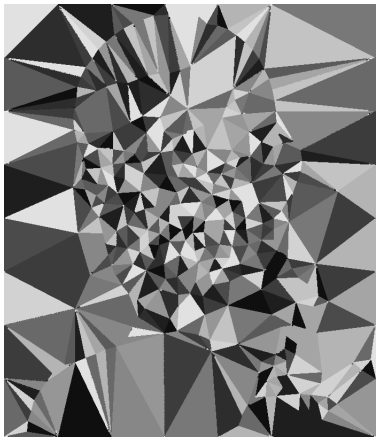

Figure 11. Portrait of Alan Turing using the approach described in section 5.

sampling distribution, the use of Harris corners, and the MIN_DIST parameter. However, from a manufacturing perspective, it is often useful to directly ask for a fixed number of patches. Thus, future work could explore designing an augmentation to our approach that merges or splits patches to achieve a desired target, leveraging textural similarity between adjacent regions.

Furthermore, more sophisticated methods for feature detection could help improve point selection, especially in more complex images with many more background objects. Contour detection is good at capturing the borders of well-defined portraits of prominent objects, but may miss more subtle details or smaller objects, like the cloud in Figure 8. Thus, potentially applying state-of-the-art object detectors to find all objects in an image and outlining those specifically could be an interesting way to generalize this approach for multi-object cases.

Lastly, we think much more work can be done on the triangle coloring approach we demonstrated in this paper. For example, employing a more sophisticated color theory perspective to add more diversity in the colors of the triangles might be an appealing option. Further research in how to use color in this type of approach could lead to many more interesting results.

We are excited about the new possibilities that exist when we recenter the efforts of computer vision to prioritize the development of artistic outputs, and as we explore new ways to bring these digital outputs into the physical world. As researchers in the field, we are eager to see what artists can do with new tools like ours.

Acknowledgements

The authors would like to extend a special thanks to our professors Alexei (Alyosha) Efros and Angjoo Kanazawa for helping us develop the broad appreciation for computer vision that made this project possible.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 2

[2] Matthew A. Brown, Richard Szeliski, and Simon A. J. Winder. Multi-image matching using multi-scale oriented patches. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:510–517 vol. 1, 2005. 3

[3] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8:679–698, 1986. 2

[4] Christopher G. Harris and M. J. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988. 3

[5] Julian Iseringhausen, Michael Weinmann, Weizhen Huang, and Matthias B. Hullin. Computational parquetry. *ACM Transactions on Graphics (TOG)*, 39:1 – 14, 2020. 1, 2

[6] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 10–17 vol.1, 2003. 2

[7] Robert L Solso. The psychology of art and the evolution of the conscious brain. 2003. 3

[8] David Stutz, Alexander Hermans, and B. Leibe. Superpixels: An evaluation of the state-of-the-art. *ArXiv*, abs/1612.01601, 2018. 2

[9] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Comput. Vis. Graph. Image Process.*, 30:32–46, 1985. 2

[10] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, 1998. 2

[11] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey,

İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 3

[12] Qi Zhang, Xiaoyong Shen, Li Xu, and Jiaya Jia. Rolling guidance filter. In *ECCV*, 2014. 2