The 8th ACM/EG Expressive Symposium
C. Kaplan, A. Forbes, and S. DiVerdi (Editors)

Irregular Pebble Mosaics with Sub-Pebble Detail

Ali Sattari Javid, Lars Doyle, and David Mould

Carleton University, Ottawa, Canada

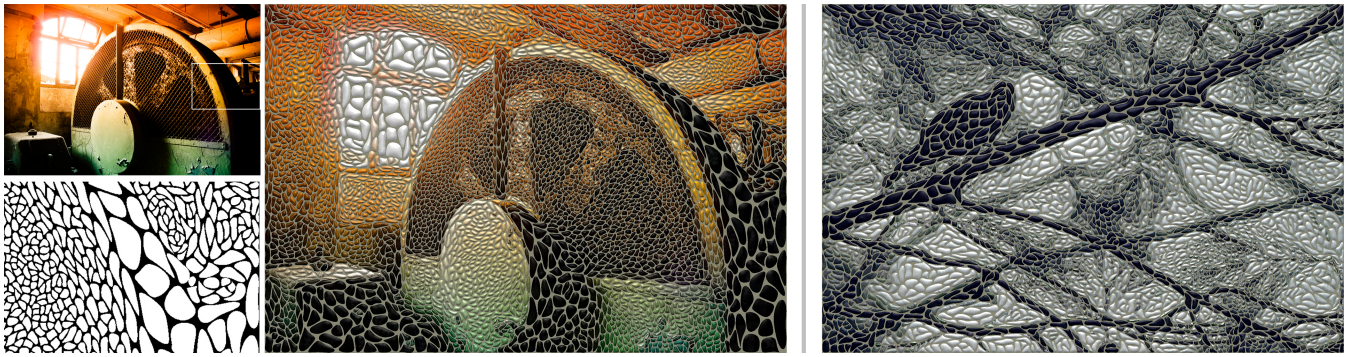

Figure 1: *Our pebble mosaic algorithm renders details through variable tile sizes and textures that hint at the underlying image content. Left: abbreviated algorithm pipeline showing source, pebble layout detail, and final result; right: additional result.*

Abstract

Pebble mosaics convey images through an irregular tiling of rounded pebbles. Past work used relatively uniform tile sizes. We show how to create detailed representations of input photographs in a pebble mosaic style; we first create pebble shapes through a variant of k-means, then compute sub-pebble detail with textured, two-tone pebbles. We use a custom distance function to ensure that pebble sizes adapt to local detail and orient to local feature directions, for an overall effect of high fidelity to the input photograph despite the constraints of the pebble style.

CCS Concepts

• ***Computing methodologies*** → ***Non-photorealistic rendering;*** *Image processing;*

1. Introduction

Pebble mosaics are an artform with roots in the ancient world yet still used today. Unlike tesselated mosaics, characterized by a uniformity and regularity of the tesserae used to construct them, pebble mosaics use rounded stones with potentially a high degree of variability in size and shape.

This paper seeks to create a digital version of a pebble mosaic from an input photograph. Previously existing specialized mosaic creation systems do not yet deal with highly heterogeneous tile creation, and general stylization systems based on neural networks struggle with creating distinct tile boundaries, often yielding blurred or malformed tiles. Two sample results, and a summary of our process, are shown in Figure 1.

Our process has two main phases: pebble creation and pebble detailing. First, we coarsely approximate the image with rounded regions generated with an algorithm akin to k-means. Second, we add color and texture to each pebble to provide image detail at sub-pebble resolution. Like recent work on pebble mosaics [DACMar], this work bridges the gap between realistic and non-photorealistic graphics, computing plausible-looking pebble shapes and textures while abstracting an input photograph. This paper improves on its predecessor in two main ways. First, it achieves greater irregularity in the shape and size of pebbles, automatically adapting to local image detail. Second, it improves on the fidelity of the mosaic to the original image, incorporating sub-pebble detail by using two-tone pebbles and selecting pebble texture with reference to the image content in the pebble region. Both changes dramatically improve the appeal of the final result.

DOI: 10.2312/exp.20191084

2. Previous Work

In their comprehensive survey, Battiato et al. [BDBFG] classify digital mosaic algorithms into two distinct areas: tile mosaics and multi-picture mosaics. Hitherto, these methods have taken separate paths, with different objectives and technical contributions. Simply put, the objective of a tile mosaic is to reproduce a target image as a collection of small, colored tile primitives. In contrast, a multi-picture mosaic employs a dataset of images that, when assembled, each source image closely matches the part of the target that it covers. Our work draws on aspects of both areas and we highlight some of the main contributions of previous work here.

Tile Mosaics. Haeberli's technique [Hae90] of tiling the image plane with a Voronoi diagram has been influential in both tile mosaic algorithms and digital stippling [Sec02]. However, simply coloring the Voronoi regions does not adequately emulate the look of ancient mosaics. Hausner [Hau01] instead sought to align rectangular tiles with image edges. His method used hardware-accelerated Centroidal Voronoi Diagrams that iteratively move away from image edges and align with a direction field. A more precise look can be seen in Elber and Wolberg [EW03] where tiles are arranged along B-spline feature curves. Their method is also able to render spatially varying tile sizes. Di Blasi and Gallo [DBG05] propose cutting tiles that intersect image edges. This method can render finer details and produces more consistent grout spacing with irregular-shaped tile fragments. An alternate approach is taken by Dalal et al. [DKLS06], who use Fast Fourier Transforms to efficiently pack tiles for mosaics, both static and animated. While the above methods are largely automatic, Abdrashitov et al. [AGYS14] present an interactive system that simplifies the digital mosaic creation process.

Multi-picture Mosaics. A popular form of the multi-picture mosaic is the photomosaic, as demonstrated by Silvers [Sil97] in his early work. Some of the common questions that arise in this area of research are: how to partition the image plane, what features and search methods should be used, and should color shifts be allowed on the tile images. Various authors [FR98, DBP06, OK08, PCK09, MM12] have taken different positions on these issues. For example, Finkelstein and Range [FR98] used a regular grid with wavelet-based image matching, and scaled colors to match the source images to the target. Alternately, Pavić et al. [PCK09] search a dataset of over a million images, finding sources that can cover an adaptively tiled target image without color alterations. They use polynomial image descriptors to obtain high accuracy over the entire tile. Another technique is to consider offsets while aligning a source with the target image. Orchard and Kaplan [OK08] use Fourier transforms to accelerate color alignment in irregularly shaped regions. The puzzle image mosaic is a variant in this area. Here, irregular shaped containers are populated with collections of shapes or images without the need to represent interior detail. Kwan et al. [KSH*16] and Saputra et al. [SKA18] demonstrate recent work in this area.

Related to photomosaics is the area of hidden or camouflage images. The objective here is to hide foreground images within the textures of a background image. Tong et al. [TZHM11] align edges from the foreground image with the background and combine them with a Poisson blending approach. Chu et al. [CHM*10] recognized that fast human recognition is aided by texture dissimilarity. They mitigate this effect by exchanging foreground textures with replacements from the background image.

Pebble Mosaics. Our current work extends Doyle et al.'s [DACMar] recent work on pebble mosaics. They use a modified SLIC [ASS*12] to obtain elongated tiles which are then smoothed in the frequency domain by means of a low-pass filter. They construct 3D pebble geometry by solving a Laplace equation on the region between two contours and then apply texture and lighting effects. Their results have excessively uniform pebbles that rely on color rather than shape to convey image content; also, they do not have any mechanism for showing details at a scale smaller than the pebbles.

3. Algorithm

The mosaic generation algorithm consists of two phases. First, we generate the mosaic tiles, computing a content-aware irregular tesselation of the image plane; the resulting tiles are smoothed into more rounded shapes resembling pebbles. This smoothing operation is implemented in the frequency domain and is identical to the one described by Doyle et al. [DACMar]. Second, we add detail to the pebbles, first creating two-tone pebbles that better match the original image content, and also adding texture, where the texture is selected to give the impression of the higher-frequency details of the original image. An outline of our algorithm is illustrated in Figure 2 and we discuss each of these phases in detail below.

3.1. Generating tiles

3.1.1. Distance function

Akin to SLIC, we use an iterative process of computing tiles as Voronoi cells from a site distribution, moving the site to the centroid of its region, and repeating. In SLIC, the distance function is the Euclidean distance in a 5D color+image space. Here, we use a more complex distance function that emphasizes color distance as well as incorporating directly the tile distortion.

Our distance function computes a distance from a site s to a destination d in the image plane. It has three essential elements: spatial distance in the image plane, a color distance computed by integrating color differences along the ray from s to d, and an elongation factor computed by measuring the deviation of the vector $d-s$ from the desired elongation direction. We take as input a 2D elongation map $\vec{E}$; image gradient directions are a good choice for $\vec{E}$, and we use the gradient for all our results, but other fields could be used instead.

We compute a color distance D_c as follows. First, we have the incremental color distance at some point q along the path:

$$\Delta C(s,q) = ||I(s) - I(q)||^{\gamma}, \tag{1}$$

where γ is a constant determining the responsiveness of the distance calculation to color differences; Since color variation is a variable between 0 and 1, the lower values of γ results in more emphasis on color variation. For the results included in this paper, we used $\gamma = 0.55$. Now, at a point q, we also have the maximum color distance

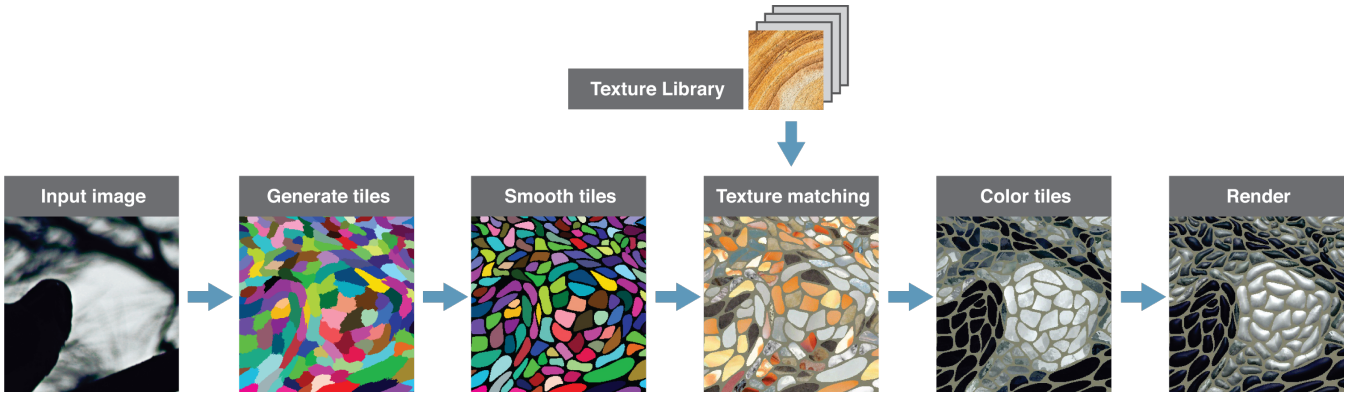

Figure 2: *Algorithm pipeline.*

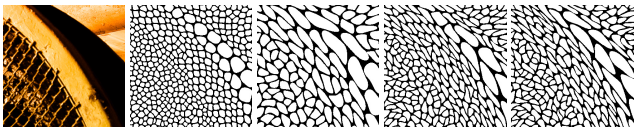

Figure 3: *Tile elongation is controlled by the z component of $\vec{E}$. Here we set $z = 1/m$ for $m = 1,3,5,9$.*

Figure 4: *Effects of changing constant parameters in distance function. From left to right: input, default variables, $\gamma = 0.85$, $z = 3$, $\alpha = 1.5$*

$C_{\max}$ seen so far:

$$C_{\max}(s,q) = \sup\{\Delta C(s,x) | s \le x \le q\} \tag{2}$$

The full color distance is the integral of these maxima, divided by the length of the path:

$$D_c(s,d) = \int_s^d \frac{C_{\max}(s,x)}{||d-s||} dx \tag{3}$$

We take a similar approach to computing the elongation factor D_E; small values of D_E extend the tile, and large values compress it. The incremental elongation factor F is the cross product of the 3D elongation field $\vec{E}$ with the ray direction, giving a value that is smaller when the ray aligns with the field. Note that the input field was 2D; we introduce a z component to $\vec{E}$ to control how pronounced the elongation will be. For the results shown in this paper, we used $z = 1/5$. Larger z produces more compact tiles, with smaller z producing more stretching in the image plane.

$$\Delta F(s,p) = \frac{||\vec{E}(p) \times (p-s)||}{||\vec{E}(p)||.||(p-s)||} \tag{4}$$

An example showing variable tile elongation appears in Figure 3. From left to right we set z to $1/m$ for $m = 1,3,5,9$.

For each point along the ray, we find the maximum so far. The final elongation factor is the integral of these maxima:

$$D_E(s,d) = \int_s^d \frac{\Delta F(s,x)_{\max}}{||d-s||} dx \tag{5}$$

Finally, the full distance is the product of the spatial distance, the color distance, and the elongation factor:

$$D_T(s,d) = ||d-s|| * (\alpha + D_E(s,d)) * D_c(s,d). \tag{6}$$

The constant $\alpha > 0$ used above provides a minimum contribution of spatial distance even when elongation distance is near zero. We typically use $\alpha = 0.15$ for results shown in this paper.

3.1.2. Assignment

We use the above distance function to compute a k-means partition of the image plane. However, the distance function is expensive, and for a large number of regions, some sort of spatial partitioning is required to make the computation tractable.

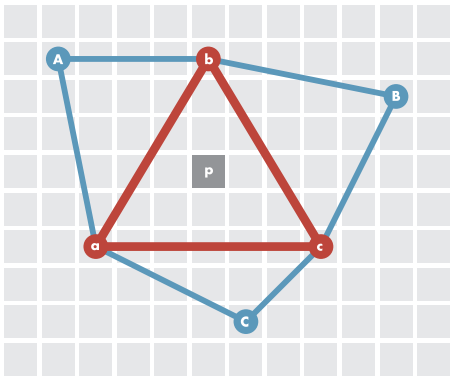

Figure 5: *To determine the region for pixel P, we consider the six sites at the corners of the four surrounding triangles.*

Partitioning with a uniform grid is common, but will not be effective in our case since the region sizes are so variable. Instead, we estimate the maximum region size before beginning: given a Delaunay triangulation over the region centres, a region will be bounded

by the set of triangles surrounding its originating site. See Figure 5; for pixels within a given triangle, we compute site-pixel distances for the sites at the corners of the enclosing triangle (sites *abc*), plus the sites on the face-adjacent triangles (sites *ABC*). This partitioning dramatically reduces the number of distance calculations, at the cost of doing a Delaunay triangulation at each iteration. The Delaunay triangulation also helps us decide where to add and remove regions in the next step.

3.1.3. Adding and Removing regions

As in stippling [Sec02], variable site density hinders convergence speed, and it is not obvious how to get a reasonable starting distribution. Using a uniform initial density could require many iterations to converge. Instead, we start with a larger density of regions than we actually want, and allow regions to merge and split in order to attain the desired local density.

During the assignment step, we also compute a confidence metric T for each pixel p: $T(p) = 1/D_T(p, N_p)$, where $D_T()$ is the distance from p to its nearest site N_p. The pixels with least confidence are furthest from any site and hence are primary candidates for adding new sites. Of course, the confidence field is rather smooth, and the pixel with least confidence will have neighbors with confidence almost as low.

We sort the pixels by confidence and process the least confident pixels, seeking to determine whether they are suitable locations for new sites. For each pixel, we compute the distance to the nearest site; if this distance is above a threshold, a site is added at this pixel's location. We chose a fairly forgiving threshold of half the smallest distance between any two sites; this adds unnecessary sites, but we need not be too strict in this phase, since the subsequent merging step can fix any problems introduced by adding too many new regions.

To resolve site merging, we reuse the Delaunay tessellation over the sites; we consider merging across each edge of the triangulation. Two sites are merged if and only if their separation according to our distance function is below a threshold $D_{\min}$:

$$(D_T(s_1, s_2) + D_T(s_2, s_1))/2 < D_{\min}.$$

Notice that the distance formulation is not symmetric (in general, $D_T(a,b) \neq D_T(b,a)$) so we average the distances in either direction.

This merging process might result in an unwanted reorganization of the regions. For example, if during our checks for merging, we repeatedly contract a region with another region to its right, an area with lower than intended density will remain to the left. We noticed this problem when we used a fixed traversal of the triangulation; if we randomize the order of the checks, the problem disappeared.

3.2. Sub-pebble details

For larger pebble sizes, the pebble layouts alone are not quite capable of conveying the target image. Augmenting the tile image with a layer of sub-pebble detail can help bring back some of this lost information. Inspired by work in hidden images [CHM*10, TZHM11], we seek to hide salient image details within pebble textures. Image content is subtly hinted at while at the same time not so forcefully depicted as to look unnatural.

3.2.1. Texture matching

We match texture swatches from an external library to the underlying target image. A small sample of which are shown in Figure 6. These swatches will later be blended with a characteristic color, or pair of colors, for each tile region. There are two steps in this matching process. First, we query the library for the best texture swatch to use for a given tile region. Then we determine the best offset that will align the chosen swatch with the edge features in the target image.

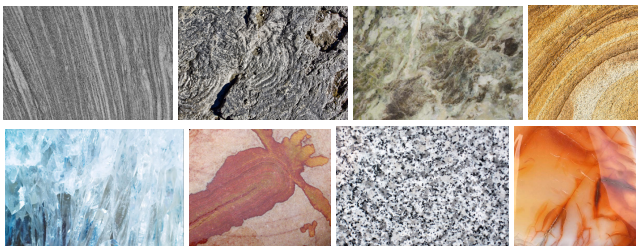

Figure 6: *Example textures used in our method.*

Texture queries are executed with Law's texture descriptors [Law80] which describe four basic filter types:

$$L5 = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} (Level)$$
$$E5 = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \end{bmatrix} (Edge)$$
$$S5 = \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \end{bmatrix} (Spot)$$
$$R5 = \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \end{bmatrix} (Ripple).$$

We use these four types to construct nine 2D filters comprising the combinations L5E5/E5L5, L5R5/R5L5, E5S5/S5E5, S5S5, R5R5, L5S5/S5L5, E5E5, E5R5/R5E5, and S5R5/R5S5 as described in Shapiro and Stockman [SS01]. Filtering the luminosity channel of a texture swatch with the above filters, obtains images F_j for $j =$ 1 to 9. We then construct f, a global texture-energy feature vector, for each swatch as follows:

$$f_j = \frac{1}{|F_j|} \sum_{x \in F_j} |F_j(x)|. \tag{7}$$

Feature vectors for a query are computed similarly from the target image. The exception is that the sum is constrained to locations under a mask representing the pebble region. We carry out texture queries in our small library of 50 samples through linear search while retaining the k-nearest neighbors in a priority queue. For larger libraries, using accelerated data structures for kNN search would reduce runtimes. Finally, one of the k options is selected randomly in order to avoid excessive repetition in uniform areas. We set k to 6 in our examples.

Now that we have found the texture image that will be used for rendering a given pebble, we conduct a second search that aims to align edges from the target and texture images. We obtain an edge map for each library texture by thresholding the gradient magnitude at the 90th percentile of the computed values. The edge map

for the target image is computed with a cropped section equal to the dimensions of the selected texture centered under the pebble. We then use Orchard and Kaplan's spectral method [OK08] for finding the minimum *sum-of-squared-difference (SSD)* between a target and offset source image under a given mask. Formally we minimize:

$$C(a,b) = \sum_{i,j} ||S(i-a, j-b) - T(i,j)||^2 W(i,j), \qquad (8)$$

with respect to the offsets a and b. Here, S and T are the edge maps from the texture and target images, respectively. The mask image, W, is constructed by setting a 1 in areas corresponding to the pebble region and setting a 0 otherwise. This offset search is carried out on four 90°rotations of the texture swatch and selecting the minimum SSD result. This selected texture will not only be used for blending with the pebble colors, but will be used for computing the colors themselves as we show in the next section.

3.2.2. Two-color pebbles

In nature, pebbles often exhibit more than a single color – they are often speckled, or streaked with mineral veins. Mimicking this phenomenon, we compute two-color pebbles by segmenting the underlying image content into two clusters using SLIC [ASS*12] with one small modification: we augment the colors in the target image with a percentage (50% in our examples) of the texture colour. This effectively distorts the boundaries so that they capture some of the texture characteristics. We also note that we histogram equalize both the target image and the texture swatch as a prepossessing normalization step.

Each of the two clusters are then colored with the average of their corresponding regions in the unaltered target image. Finally, the texture swatch is blended with the two-colored pebble region in the luminosity channel. An illustration of this process is given in Figure 7 on a contrived example showing diverse image content within a single region.

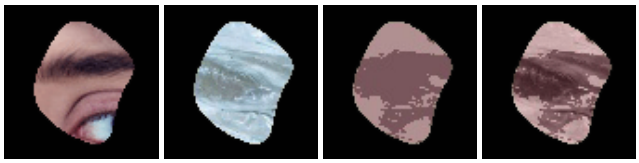

Figure 7: *Sub-pebble detail is suggested by rendering a pebble with two colors. Left to right: target image, aligned texture swatch, two-color clustering, blending with texture swatch.*

3.3. Rendering

Our rendering method is identical to Doyle et al.'s [DACMar]. We set gradient and value constraints on two contour boundaries – the outer pebble boundary and an inner boundary set at 85% of the maximum in the region's distance transform. The pebble height-field is constructed by solving a Laplace equation and the resulting geometry is rendered using Phong shading.

4. Results and Discussion

Example results from our algorithm applied to various subject matter can be seen in Figure 8 with their original sources shown in Figure 15. We recommend looking at these images both close up, to examine individual pebbles and textures, and also at a medium distance, say 70-100 cm from the display.

Our tile-generation method demonstrates the advantages of using variable pebble sizes. It adds smaller pebbles where needed in high-gradient regions and allows less-varied spaces to be filled by larger pebbles. Hence, fine details can be rendered, but there is still an overall reduction in tile count compared to previous methods. The tree on the middle left reveals this quality as the small pebbles that represent the high-frequency leaves give way to the low-frequency tree trunk, which is rendered in much larger pebbles. The advantage of rendering sub-pebble detail in textured two-colored pebbles is shown in the sheep image on the top left. Here, the texture of the sheep's fleece is emphasized in the pebble details. Another example is visible in the leaf image on the middle right. Some of the pointy tips of the leaves are too fine to be segmented into individual pebbles, and instead these small details are rendered with two colors, combining the dark foreground and the bright background within the same tile.

In the portrait on the bottom left, we can observe several qualities of our method at once. Pebble sizes range from large, in the featureless background, to small, in the highly textured areas on the hat. Sub-pebble textures also hint at the woven textures that are present in the original image seen in Figure 15. Hard color breaks are present within select pebbles, particularly on the eye and the hat brim, which help reaffirm edges where the tile segmentation fails to capture them.

Looking at the black and white pebble outlines, and the final rendering, in Figure 1 further confirms how tile size helps the viewer visually segment the image. Previous methods [Hau01, DACMar] relied on tile color and alignment with edges to indicate image content. Here, we show that the extra dimension of size separates the smooth, curved sides of the engine structure from the detailed front-facing area. This distinction is clearly made even without the dimension of color. Earlier methods allowed a user to specify regions that should be rendered with more detail, but here we can get tile size variation with no manual intervention.

4.1. Comparisons with previous work

In Figure 9 we compare our results against Hausner [Hau01] and Doyle et al. [DACMar]. Both the previous methods, as well as our result on the bottom left, use 2000 pebbles. On the bottom right we use 1000. We render our images without lighting effects and use two solid colors on each pebble to enable a more direct comparison. Our method and Doyle et al.'s are both able to characterize finer levels of detail through irregular tile shapes that conform to image boundaries. However, our method also has the advantage that higher-frequency regions, such as the figure's face, are rendered with smaller tiles. Indeed, even with the smaller tile count on the bottom right we can show comparable detail with the larger tile count in Doyle et al.'s result on the top right. Using textured pebbles also contributes to the advantage of our method.

Figure 8: *Some results.*

In Figure 10 we compare our result with Doyle et al.'s. While we use their method to construct the pebble geometry, our irregular segmentation is better able to represent image content. In our result on the right, the uniform background is rendered in large pebbles. This contrasts with higher density on the rough, scaly iguana. Additionally, using variable pebble size has allowed the lizard's pupil to be clearly depicted. In Doyle et al.'s result (center) this detail is lost.

Varying tile size has also been experimented with in previous work – mostly as a result of manual segmentation [Hau01, DACMar]. Some examples are shown in Figure 11. On the top right, Elber and Wolberg [EW03] present a method for increasing tile sizes

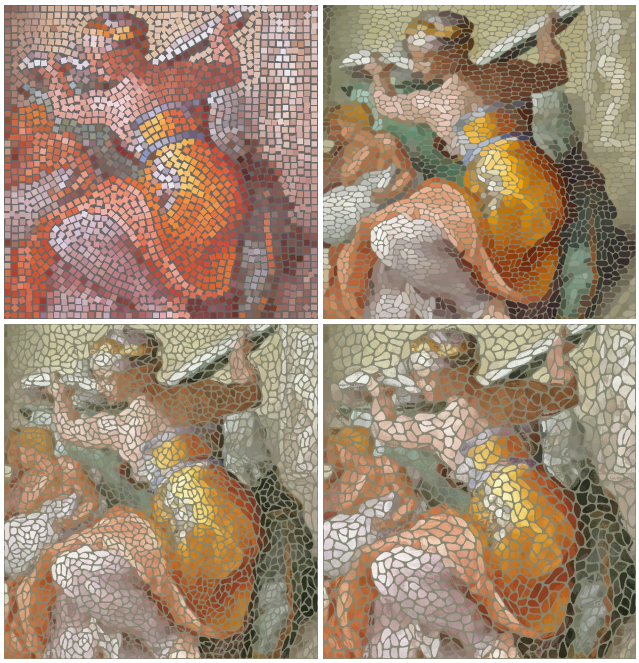

Figure 9: *Comparison with Hausner [Hau01] (top left) and Doyle et al. [DACMar] (top right) using 2000 tiles. Our results are shown using 2000 and 1000 tiles on the bottom left and right.*

Figure 10: *Comparison with Doyle et al. [DACMar]. From left to right: original, Doyle et al., ours.*

as they move away from user-defined feature lines; an abstraction emerges that emphases the user-imposed structure rather than image content. Our result on the bottom shows a greater variability than Hausner's [Hau01] manual segmentation on the top left. Additionally, it is more economical, as small tiles are only used where necessary, such as the regions around the eyes.

Our method shares a goal with Orchard and Kaplan's [OK08] photomosaic paper. Namely, we desire sub-tile details to align with image edges. Orchard and Kaplan search a large dataset of images to find offsets, or rotations, that match a target image. A result is depicted in Figure 12 on the top left where we can see the tile image aligned with the architectural structure. Owing to their large search space they measure runtimes in hours. In our approach, we do not seek such accurate texture matches since they are used principally

Figure 11: *Methods for varying tile size. Top left: Manual segmentation [Hau01]; top right: Elber and Wolberg's method [EW03]; bottom: ours.*

to augment the clustering procedure in Section 3.2.2. Looking at the zoomed section of the cat image on the bottom left, one can see how pebble texture details align with the lower edge of the eye. Our reduced matching requirements enables us to query our texture dataset with low dimensional texture-energy features. The final offset search is only performed on four rotations of a single image, with the entire search process taking 2 minutes on a 1900×1300, 4000 tile image.

4.2. Limitations

Our method has difficulty handling low-contrast images. The image of the statue in Figure 13 is barely recognizable in our rendering. Adding to these difficulties are the monochrome palette and similar texture energy between foreground and background. These qualities prevent any real distinction between pebble colors or sizes.

Inherent in any mosaic style is the tension between scale of image detail and scale of the pebbles. We have to some degree addressed the challenge of showing details smaller than the pebble sizes, but the pebbles are still the most prominent aspect of the result. More investigation into sub-pebble detail is warranted, perhaps using more prominent textures without compromising believability.

Figure 14 contrasts our synthetic pebbles against a real mosaic example. While real pebbles are sometimes slightly concave, our artificial pebbles have far more pronounced concavities. More problematic still are the pointy tips of some of our pebble shapes.

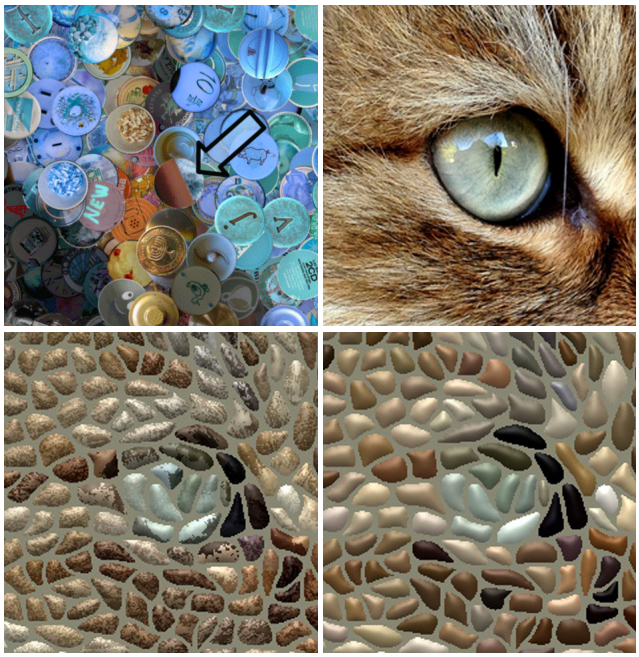

Figure 12: *Sub-tile edge alignment compared with Orchard and Kaplan [OK08] (top left). Our result is a zoomed version of Figure 8, bottom right. On the bottom left we see two-color tiles which render sub-tile edges. The result on the bottom right uses a single tile color. The original zoomed image is shown on the top right.*

Figure 13: *Results on angel image; the forms in the image have become difficult to see.*

Our Fourier-based smoothing method is not able to remove these structures that originate in the initial segmentation. However, it is likely that subsequent morphological operations would be able to remove these artifacts.

We also note that our segmentation process is sensitive to noise. In the landscape on the top right of Figure 8, one would expect the variable cloud texture to be rendered in smaller pebbles than the sky. However, we witness the opposite effect here; low-level noise in the sky, visible when the image is zoomed in, causes our method to split the pebbles. Pre-processing with a smoothing filter would be helpful in these situations.

Figure 14: *Comparison with real pebble samples. Left: synthetic pebbles, right: real pebbles.*

5. Conclusions

In this work, we have presented an automated pebble mosaic image-stylization algorithm. It extends previous work on pebble mosaics in two ways. First, we generate irregular sized tiles that are better able to describe image features. Though an adaptive splitting and merging process, our method places more tiles in higher-energy regions and less where the image content is uniform. Second, we create sub-pebble texture details that are able to suggest content from the underlying target image. We query a dataset of texture swatches and use matching samples to guide a tile coloring process. This is especially helpful for rendering fine features that are too small to be adequately captured by tile boundaries. Our mosaics are more detailed than was possible with previous methods using the same number of tiles.

In the future, we would like to extend this work to animated pebble mosaic movies. This will present new challenges, such as ensuring the coherent movement and shape of tiles between frames. Another area for improvement is in salience estimation for background removal. While photographs provide good a starting point for a mosaic algorithm, real examples display abstract patterns as well as figures. Similarly, we could automatically incorporate non-photographic elements into non-salient image regions. On the other hand, semantic information could also be used to emphasize important areas – either by increasing the pebble count to draw the eye, or to re-enforce boundaries at weak edges.

Acknowledgements

We would like to thank the anonymous reviewers for many insightful comments. Thanks also to members of the Graphics, Imaging and Games Lab for productive comments and discussions. Funding for this work was provided by NSERC, OGS, and by Carleton University.

We used the NPR-general benchmark dataset for several examples and found it helpful in evaluation. We would like to thank the many photographers who provided original images for this paper under a Creative Commons license: Julio Romero (iguana), Eole

Figure 15: *Source images used in this paper.*

Wind (angel), Theen Moy (cat), sicknotepix (toque), Julita B.C. (sheep), Rory MacLeod (portrait), Akulatraxas (engine), Takashi .M (leaves), Lina (bird), Hardik Buddhabhatti (mountain), and Thomas Mues (tree).

References

[AGYS14] ABDRASHITOV R., GUY E., YAO J., SINGH K.: Mosaic: Sketch-based interface for creating digital decorative mosaics. In *Proceedings of the 4th Joint Symposium on Computational Aesthetics, Non-Photorealistic Animation and Rendering, and Sketch-Based Interfaces and Modeling* (New York, NY, USA, 2014), SBIM '14, ACM, pp. 5–10. URL: http://doi.acm.org/10.1145/2630407.2630409, doi:10.1145/2630407.2630409. 2

[ASS*12] ACHANTA R., SHAJI A., SMITH K., LUCCHI A., FUA P., SUSSTRUNK S.: SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 11 (Nov. 2012), 2274–2282. 2, 5

[BDBFG] BATTIATO S., DI BLASI G., FARINELLA G. M., GALLO G.: Digital mosaic frameworks - an overview. *Computer Graphics Forum 26*, 4, 794–812. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01021.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2007.01021.x, doi:10.1111/j.1467-8659.2007.01021.x. 2

[CHM*10] CHU H.-K., HSU W.-H., MITRA N. J., COHEN-OR D., WONG T.-T., LEE T.-Y.: Camouflage images. *ACM Trans. Graph. 29*, 4 (July 2010), 51:1–51:8. URL: http://doi.acm.org/10.1145/1778765.1778788, doi:10.1145/1778765.1778788. 2, 4

[DACMar] DOYLE L., ANDERSON F., CHOY E., MOULD D.: Automated pebble mosaic stylization of images. *Computational Visual Media* (to appear). 1, 2, 5, 6, 7

[DBG05] DI BLASI G., GALLO G.: Artificial mosaics. *The Visual Computer 21*, 6 (Jul 2005), 373–383. URL: https://doi.org/10.1007/s00371-005-0292-4, doi:10.1007/s00371-005-0292-4. 2

[DBP06] DI BLASI GIANPIERO G. G., PETRALIA M. P.: Smart ideas for photomosaic rendering. In *Eurographics Italian Chapter Conference* (2006). 2

[DKLS06] DALAL K., KLEIN A. W., LIU Y., SMITH K.: A spectral approach to NPR packing. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2006), NPAR '06, ACM, pp. 71–78. URL: http://doi.acm.org/10.1145/1124728.1124741, doi:10.1145/1124728.1124741. 2

[EW03] ELBER G., WOLBERG G.: Rendering traditional mosaics. *The Visual Computer 19*, 1 (Mar 2003), 67–78. URL: https://doi.org/10.1007/s00371-002-0175-x, doi:10.1007/s00371-002-0175-x. 2, 6, 7

[FR98] FINKELSTEIN A., RANGE M.: Image mosaics. In *Electronic Publishing, Artistic Imaging, and Digital Typography* (Berlin, Heidelberg, 1998), Hersch R. D., André J., Brown H., (Eds.), Springer Berlin Heidelberg, pp. 11–22. 2

[Hae90] HAEBERLI P.: Paint by numbers: Abstract image representations. *Computer Graphics 24*, 4 (Sept. 1990), 207–214. URL: http://doi.acm.org/10.1145/97880.97902, doi:10.1145/97880.97902. 2

[Hau01] HAUSNER A.: Simulating decorative mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 573–580. URL: http://doi.acm.org/10.1145/383259.383327, doi:10.1145/383259.383327. 2, 5, 6, 7

[KSH*16] KWAN K. C., SINN L. T., HAN C., WONG T.-T., FU C.-W.: Pyramid of arclength descriptor for generating collage of shapes. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 229:1–229:12. URL: http://doi.acm.org/10.1145/2980179.2980234, doi:10.1145/2980179.2980234. 2

[Law80] LAWS K. I.: Rapid texture identification. *Proc.SPIE 0238* (1980), 0238 – 0238 – 6. URL: https://doi.org/10.1117/12.959169, doi:10.1117/12.959169. 4

[MM12] MILLER J., MOULD D.: Accurate and discernible photocollages. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (Goslar Germany, Germany, 2012), CAe '12, Eurographics Association, pp. 115–124. URL: http://dl.acm.org/citation.cfm?id=2328888.2328908. 2

[OK08] ORCHARD J., KAPLAN C. S.: Cut-out image mosaics. In *Proceedings of the 6th International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2008), NPAR '08, ACM, pp. 79–87. URL: http://doi.acm.org/10.1145/1377980.1377997, doi:10.1145/1377980.1377997. 2, 5, 7, 8

[PCK09] PAVIĆ D., CEUMERN U., KOBBELT L.: Gizmos: Genuine image mosaics with adaptive tiling. *Computer Graphics Forum 28*, 8 (2009), 2244–2254. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01437.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01437.x, doi:10.1111/j.1467-8659.2009.01437.x. 2

[Sec02] SECORD A.: Weighted Voronoi stippling. In *Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2002), NPAR '02, ACM, pp. 37–43. URL: http://doi.acm.org/10.1145/508530.508537, doi:10.1145/508530.508537. 2, 4

[Sil97] SILVERS R.: *Photomosaics*. Henry Holt and Co., Inc., New York, NY, USA, 1997. 2

[SKA18] SAPUTRA R., KAPLAN C., ASENTE P.: RepulsionPak: Deformation-driven element packing with repulsion forces. In *Proceedings of Graphics Interface 2018* (2018), GI 2018, Canadian Human-Computer Communications Society, pp. 10–17. doi:10.20380/GI2018.03. 2

[SS01] SHAPIRO L. G., STOCKMAN G. C.: *Computer Vision*. Pearson, 2001. 4

[TZHM11] TONG Q., ZHANG S.-H., HU S.-M., MARTIN R. R.: Hidden images. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2011), NPAR '11, ACM, pp. 27–34. URL: http://doi.acm.org/10.1145/2024676.2024681, doi:10.1145/2024676.2024681. 2, 4