Volume 0 (1981), Number 0 pp. 1–11 COMPUTER GRAPHICS *forum*

Fast Rendering of Image Mosaics and ASCII Art

Nenad Markuš[1], Marco Fratarcangeli[2], Igor S. Pandžić[1] and Jörgen Ahlberg[3]

[1]University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia
[2]Chalmers University of Technology, Dept. of Applied IT, Kuggen, Lindholmsplatsen 1, SE-412 96 Göteborg, Sweden
[3]Linköping University, Dept. of Electrical Engineering, Information Coding Group, SE-581 83 Linköping, Sweden

nenad.markus@fer.hr, marcof@chalmers.se, igor.pandzic@fer.hr, jorgen.ahlberg@liu.se

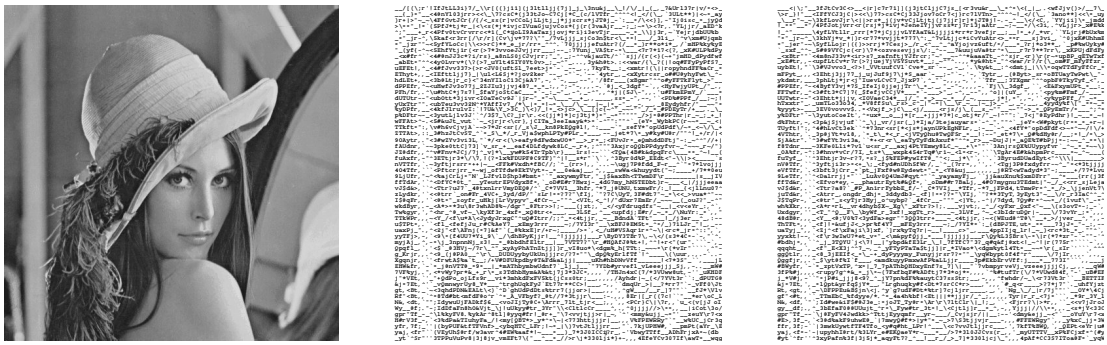

Figure 1: *From left to right, the Lenna image rendered in ASCII art with the SSIM-based mapping [WBSS04] and our decision tree-based approach. Our approach reproduces the same visual results enhancing the performance of two orders of magnitude.*

Abstract

An image mosaic is an assembly of a large number of small images, usually called tiles*, taken from a specific dictionary/codebook. When viewed as a whole, the appearance of a single large image emerges, i.e., each tile approximates a small block of pixels. ASCII art is a related (and older) graphic design technique for producing images from printable characters. Although automatic procedures for both of these visualization schemes have been studied in the past, some are computationally heavy and cannot offer real-time and interactive performance. We propose an algorithm able to reproduce the quality of existing non-photorealistic rendering techniques, in particular ASCII art and image mosaics, obtaining large performance speed-ups. The basic idea is to partition the input image into a rectangular grid and use a decision tree to assign a tile from a predetermined codebook to each cell. Our implementation can process video streams from webcams in real-time and it is suitable for modestly equipped devices. We evaluate our technique by generating the renderings of a variety of images and videos, with good results. The source code of our engine is publicly available.*

Keywords: *ASCII art, image mosaics, decision trees, SSIM*

ACM CSS: *Rendering [Computer Graphics]: Non-photorealistic rendering*

1. Introduction

As pointed out by Tran [Tra99], a good image mosaic is striking because it cleverly puts together otherwise ordinary and unrelated features of the individual tiles into a coherent larger framework. Besides having aesthetic value, image mosaics have been investigated in the context of copyrighted

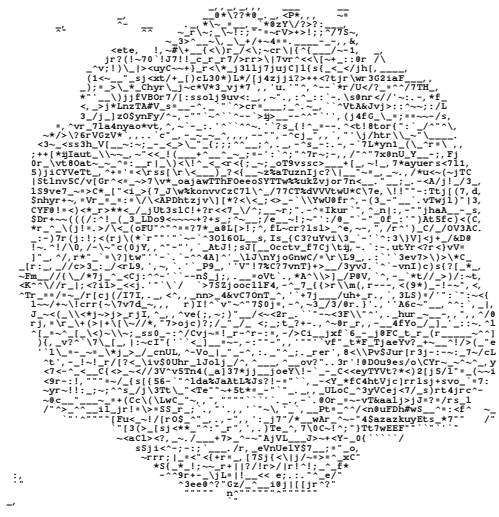

Figure 2: *A flower rendered in ASCII tiles with our method.*

material protection and hiding secret data [BG03]. ASCII art is a similar graphic design technique that aims at producing images from printable characters. It originated in its modern form in times when printers had limited graphical capabilities and transferring images over computer networks was inappropriate due to bandwidth constraints. It has its value as an interesting reminder of graphical expression on early computer interfaces. Today, ASCII art is commonly used in media that cannot display images or mainly uses text. These include e-mails, bulletin boards, discussion forums, internet chats and short message services (SMS).

We introduce a novel technique that significantly increases the speed of ASCII art and image mosaic rendering. This is achieved by addressing the problem as a classification task which is then solved with an optimized binary decision tree. The tree samples pixel intensities at relevant locations in the input image and compares them with a set of thresholds in order to assign an appropriate tile to each image region. The thresholds and pixel sampling locations are set during the tree learning stage. This simple procedure produces visually appealing results, as shown in Figure 1, while being at least an order of magnitude faster than other approaches reported in the literature. The limitation of our method is in rendering with a small number of tiles (low resolution) and/or with a limited codebook. However, these limitations also apply to other approaches. A sample rendering produced with our method can be seen in Figure 2. The runtime part is visually summarized through the example in Figure 3.

2. Related work

Generating ASCII artwork by hand is a laborious task that requires careful placement of individual glyphs. There exist tools which simplify the process [MJN11] or convert the image to ASCII art in a completely automatic way [aal, OR08, TTIN13, XZW10]. The usual way to automatically generate ASCII art is to subdivide the image into a rectangular grid and replace each cell with a font glyph in such a way that the average grey level stays approximately the same. A representative system that uses this technique is AAlib [aal], written in the 1990s. It relies on precomputed lookup tables and is able to produce visually appealing results in real-time. Another interesting approach has been described by O'Grady and Rickard [OR08]. They treat the conversion of binary images to ASCII art as an optimization problem and use an algorithm based on non-negative matrix factorization techniques [LS99] to solve it. The approach has limited applications due to the considerable processing time needed for a single image. Inspired by the techniques for optimizing binary images for printing [PQW*08], Takeuchi et al. [TTIN13] propose a method which approximates the characteristics of the human visual system with a Gaussian filter and uses this metric to drive the optimization procedure which involves placing individual font glyphs. The authors manage to achieve real-time frame rates on images of moderate size by implementing the algorithm on a GPU. Recently, research has focused on structure-based art [XZW10] which represents an object by rendering its rough outline in ASCII characters. Their system achieves impressive results that rival even the artwork created by artists. However, it cannot process images in real-time on modern machines and, thus, has a different area of application than the approach we describe in this paper.

As with ASCII art, image mosaics are usually generated by partitioning the input image in a grid and assigning a tile from a codebook to each cell. (There are also techniques for generating image mosaics with non-rectangular, rotated or slightly deformed tiles (e.g., [KP02] and [OK08]), which is a different problem from the one we address here.) Silvers [Sil97] described the first computer program for automatizing this process. The similarity between a tile and a grid cell is usually calculated by taking L_1, L_2 or some string distance between them [Sil97, Tra99], or by using techniques from content-based image retrieval [FR98, ZNZ03]. Klein et al. [KGFC02] extended the idea to rendering video streams with an assembly of small video tiles. An overview of classical algorithms can be found in [BBFG07]. Blundo and Glandi [BG08] improve on basic techniques by incorporating multiresolution analysis to include the information from surrounding grid cells. The approach produces gentle color changes, smooth shapes and more refined textures. Kang et al. [KSRY13] speed up the basic method based on per-pixel distance calculation [Sil97, Tra99] by first downsampling the tiles/cells to $n \times n$ thumbnails, where n is a small number (3 or 4). The computation time needed to find the best matching tile is significantly reduced. Additionally, the mosaic generation process becomes more robust to small image perturbations and noise. In our preliminary experiments, we ob-

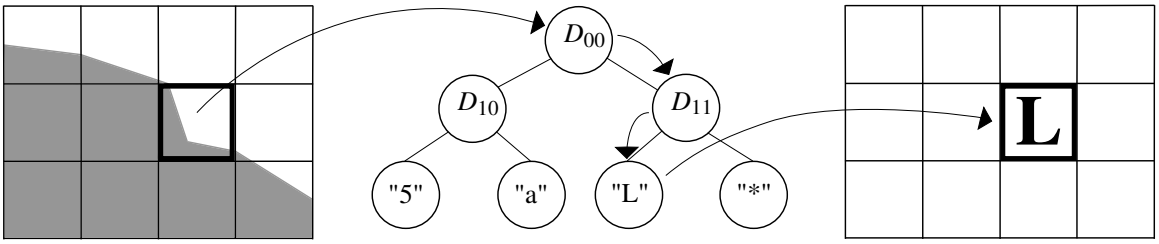

Figure 3: *ASCII art rendering example. The input grayscale image is partitioned into a rectangular grid (left) and each cell is passed through a decision tree (middle) in order to replace it with an ASCII glyph (right). The decisions in internal tree nodes, D_{ij}, are obtained by comparing the intensity of a pixel at a predetermined location with a threshold.*

served that the approach of Kang et al. [KSRY13] provides the best tradeoff between image mosaic quality and speed, thus, we use it as a baseline for comparison. The aforementioned approaches require the search of the whole codebook for the best matching tile, and this demands large computational resources (the search complexity is linear in the size of the codebook). This issue has been addressed by Choi et al. [CJKK13] and Kang et al. [KSRY11]: they report real-time performance by optimizing the tile set and taking advantage of GPU acceleration. Another interesting approach to speed up the database search has been proposed by Blasi and Petralia [BP05]. The idea is to build an *antipole tree* data structure [CFP*05] prior to rendering. This enables fast approximate nearest neighbour search in feature space extracted from grid cells, which is then exploited during rendering. We discuss the main differences with respect to our approach at the end of Section 3.2.

3. Method

We subdivide the image into a rectangular grid and map each cell to a tile image from our codebook (a font glyph, for example). The visual outcome of the process depends on this mapping. In the simplest case, the mapping assigns a tile based on its mean intensity or color value. This can lead to an unnecessary loss of image detail. A more sophisticated approach is to base the mapping on a correlation measure that takes the spatial distribution of pixels into account (i.e., the structure of the underlying region). Our method is independent of the correlation measure and it is useful for improving the speed of the process. We first describe particular correlation measures which will be used in this paper (Section 3.1) and then introduce a machine learning technique to approximate them (Section 3.2).

3.1. The structural similarity mapping

To demonstrate our approach for ASCII rendering of grayscale images, we employ a measure of structural similarity (SSIM) introduced by Wang et al. [WBSS04]. Each region of the image is assigned a font glyph that is most similar to it, as measured by the SSIM index. To obtain the SSIM index between two arrays of pixels (a grid cell and a font glyph in our case), we first compute luminance, contrast and structure similarities:

$$L_{12} = \begin{cases} 1, & \text{if } \mu_1 \text{ and } \mu_2 \text{ are } 0 \\ \frac{2\mu_1\mu_2}{\mu_1^2+\mu_2^2}, & \text{otherwise} \end{cases}$$

$$C_{12} = \begin{cases} 1, & \text{if } \sigma_1 \text{ and } \sigma_2 \text{ are } 0 \\ \frac{2\sigma_1\sigma_2}{\sigma_1^2+\sigma_2^2}, & \text{otherwise} \end{cases}$$

$$S_{12} = \begin{cases} 1, & \text{if } \sigma_1 \text{ is } 0 \text{ or } \sigma_2 \text{ is } 0 \\ \frac{\sigma_{12}^2}{\sigma_1\sigma_2}, & \text{otherwise} \end{cases}$$

where μ_1 and μ_2 are the pixel mean values of the arrays, σ_1^2 and σ_2^2 are their variances, and σ_{12}^2 is their correlation. The SSIM index is computed as

$$\text{SSIM}_{12} = L_{12}^l C_{12}^c S_{12}^s, \tag{1}$$

where exponents l, c and s are parameters used to adjust the relative importance of each component. The downside of this approach is its computational load: it requires a loop over all font glyphs for each grid cell to compute σ_{12}^2.

In the case of color images, we take into account the spatial distribution of color to increase the sharpness of generated image mosaics. This is achieved by subdividing the current image region into $n \times n$ subregions, calculating average color within each of them and concatenating all values into a vector. This vector serves as a region descriptor. The spatial color similarity (SCSIM) between an image region (represented by $\mathbf{r}$) and a tile from a codebook (represented by $\mathbf{t}$) is calculated as

$$\text{SCSIM} = \begin{cases} 1, & \text{if } \mathbf{r} \text{ and } \mathbf{t} \text{ are } \mathbf{0} \\ \frac{2\mathbf{r}^T\mathbf{t}}{\mathbf{r}^T\mathbf{r}+\mathbf{t}^T\mathbf{t}}, & \text{otherwise} \end{cases} \tag{2}$$

We empirically found that this simple similarity measure gives good results in practice. In particular, the generated image mosaics are qualitatively similar to the ones generated with other similar methods [Tra99, ZNZ03].

There are a number of other similarity measures that would also work, like the ones based on color correlograms, commonly used in content-based image retrieval, e.g., [HKM*97]. Once the region color descriptor has been computed, it has to be compared with all the descriptors in the database to find the best matching tile. This is computationally heavy and, thus, not suitable for some applications, like real-time rendering of image mosaics. Approximations are required to make the approach feasible.

We solve this problem by using a machine learning technique which is specifically designed to extract correlations from large quantities of data. We treat the tile assignment step as a classification task which we solve using an optimized binary decision tree. This was inspired by the work done by Rosten et al. [RPD10] in the field of corner detection, which is an important problem in computer vision.

3.2. Approximating the mapping with a decision tree

Decision trees are tools for function approximation [HTF09]. The basic idea is to split recursively the original problem into two simpler ones, each solvable by a model of reduced complexity. The splits are performed at internal nodes of the tree, based on problem-specific binary tests. Leaf nodes contain simple models that approximate the desired output. We treat the tree construction process as a supervised learning problem. We choose the binary tests in internal nodes and output models at leaf nodes based on a finite set of input-output pairs in a training set.

In our case, the internal nodes of the tree employ simple binary tests which are defined for an image region R as

$$\text{bintest}(R;x,y,c,t) = \begin{cases} 0, & R_{x,y,c} \leq t \\ 1, & \text{otherwise} \end{cases}$$

where $R_{x,y,c}$ is the pixel intensity sampled from color channel c at location specified by x and y, and t is the threshold associated with the test. Each leaf node of the tree contains a label that represents a specific tile.

The construction of the tree is supervised. For example, in the case of ASCII rendering, the training data is a set $\{(R_s, g_s) : s = 1,2,\ldots,S\}$ where $g_s \in \{!, ", \ldots, a, b, c, \ldots, A, B, C, \ldots, \sim\}$ is the ground truth glyph label associated with image region R_s. The binary test in each internal node of the tree is selected in a way to minimize the average entropy obtained when the incoming training data is split by the test. The average entropy is computed as

$$H = p_0 H_0 + p_1 H_1, \tag{3}$$

where p_0 and p_1 are fractions of training samples for which the results of a binary test on an associated region were 0 and 1, respectively. Values H_0 and H_1 are the entropies computed from the distributions of ground truth labels within each of these two partitions. As the set of all pixel intensity comparisons is prohibitively large, we generate only a small subset during optimization of each internal node by randomly selecting a number of (x,y,c,t) quadruplets. The test that achieves the smallest entropy according to equation 3 is selected. The training data is recursively clustered in this way until at least one termination condition is met. In our setup, we limit the depth of the tree to reduce training time, runtime processing speed and memory requirements. Each leaf node contains a tile label which was the most frequent of the ones arriving there during the learning process.

For a tree of depth equal to D, the required memory is $\Theta(2^D)$ and the time needed to transform an image region into a tile is $\Theta(D)$.

Our approach shares some similarities with the tree-structured vector quantization technique (TSVQ) [GG92]. As the main goal of TSVQ is signal compression, the codebook is not given in advance like in our case (ASCII glyphs or image tiles) but is constructed during the learning process. Another difference is in the binary tests used in internal nodes of the tree. In the TSVQ case, each internal node stores two prototype vectors with which the incoming sample is compared. The nearest (minimum distortion) one determines in which of the subtrees the search has to continue. A similar structure known as the *antipole tree* [CFP*05] has beed used by Blasi and Petralia [BP05] to speed up the database search when rendering image mosaics. This approach and TSVQ are both based on codebook clustering while our method relies on approximating the similarity measure by learning a discriminative classifier from training data. Thus, neither TSVQ nor [BP05] can be directly applied to ASCII art rendering based on the SSIM mapping (Section 3.1) as these methods require feature vector (descriptor) extraction from each grid cell. This gives our method more flexibility, since similarity measures can be easily constructed from feature vectors whereas the converse is not true.

4. Experimental analysis

To show how our framework performs for grayscale images, we use it to render ASCII art. We use a font with glyph size equal to 10×6 pixels. The implemented renderer replaces image regions with glyphs by finding the best match according to Equation 1. In our experiments, we set the exponents to $l = 10$, $c = 0.1$ and $s = 0.1$. We experimentally found that these adequately weight the contribution of the region structure content when selecting an appropriate glyph and lead to visually appealing results. To show how our framework works for color images, we render image mosaics using 1500 flower tiles (15×15 resolution). Some examples from the flower tile codebook can be seen in Figure 4. Image mosaics are generated by finding the best match for each image region according to Equation 2. We set $n = 3$, i.e., our renderer subdivides each region/tile into 3×3 blocks when

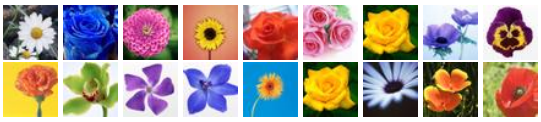

Figure 4: *Some example tiles from the flower codebook used in our experiments.*

computing its descriptor. This value provides a good tradeoff between retrieval speed and spatial color matching accuracy.

As we demand real-time performance on hardware with limited computational resources, we use a decision tree to map a tile to each image region. Learning of the tree is described in Section 4.1. Section 4.2 demonstrates some results on real-world images. Section 4.3 discusses processing speed advantages.

4.1. Learning step

To generate the training set, we render 200 5 Mpixel images of "natural" scenes (flowers, landscapes, sky, etc.) by subdividing each of them into a rectangular grid and assigning each cell a tile image by directly optimizing the criterion defined by Equation 1 or Equation 2, depending on whether or not we are incorporating color information. This process results in tens of millions of region-tile pairs which are then used as training samples to learn the tree.

The parameters of the tree learning process have to be set in advance. We generate 1024 binary tests during the optimization of each internal node by repeated sampling of four integers (x, y, c and t) from an appropriate distribution. The thresholds t are selected uniformly from $\{0,1,2,\ldots,255\}$ as the intensity of each pixel is represented with a single byte. Channel index c is sampled from $\{0,1,2\}$ (this parameter is ignored when dealing with grayscale images). In the case of $w \times h$ pixel tiles in the codebook, x and y are chosen uniformly from $\{0,1,\ldots,w-1\}$ and $\{0,1,\ldots,h-1\}$, respectively. We learn trees of depths equal to 8, 12, 16 and 20 for the purpose of experiments. Learning the tree of depth equal to 16 takes around 1 hour on a modern PC with 4 cores and 8GB of RAM. The tree requires 500kB of storage.

To numerically investigate the accuracy of ASCII art rendering with a decision tree, we transformed a large number of images to ASCII with different approaches and compared the errors induced by the approximation. Cumulative error distributions are displayed in Figure 5. We can see that the trees outperform the luminance-based approach, which serves as a baseline. As the depth d increases, the accuracy approaches the ground truth curve obtained with SSIM. Similar conclusions can be made when rendering image mosaics. We omitted these results for brevity.

Figure 5 shows that the trees indeed manage to approximate

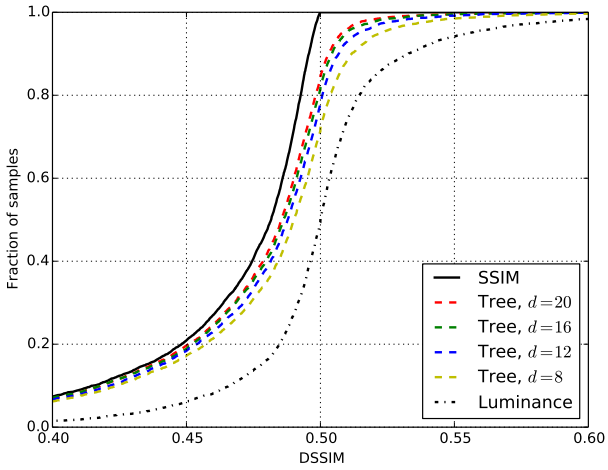

Figure 5: *Cumulative error distributions for different methods on a validation set (10 5 Mpixel images). The error is defined as* $DSSIM = (1 - SSIM)/2$.

Method	Mean score μ	Standard deviation	95% confidence interval for μ	
			$\mu-\delta$	$\mu+\delta$
Luminance-based	4.51	2.10	4.29	4.74
SSIM-based	5.54	2.01	5.32	5.75
Tree-based	5.61	1.95	5.40	5.82

Table 1: *Survey statistics for ASCII art.*

the mapping. However, it is not clear how the presented numerical measure correlates with visual quality. In the next two sections we provide images and processing speed analysis, and discuss the applicability of our method.

4.2. Rendering

We use the tree of depth equal to 16 to display the artwork in the rest of this paper as this has proved to be a good tradeoff between memory usage, processing speed and quality. Figure 6 shows the rendering of a circle in ASCII. When comparing the produced ASCII artwork, we conclude that the tree-based approach manages to reduce aliasing artefacts and preserves more image details than the luminance-based method with dithering, and the quality approaches that of SSIM. These results are consistent with numerical measures from Figure 5. To back our claims, we also conducted a survey in which 50 participants were invited to score the similarity between each of the six source images and their ASCII renderings obtained by three different methods: luminance-, SSIM- and tree-based. (The possible scores were on a 10-point scale; 10 being best.) The detailed results of the survey are given in the supplementary material. Table 1 shows a short summary. We can see from the statistics that SSIM and the tree perform approximately the same while produc-

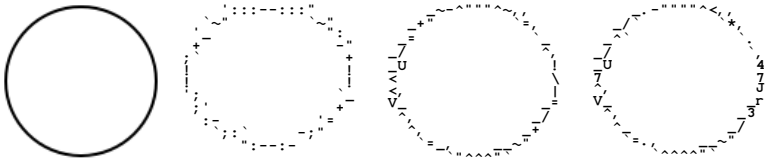

Figure 6: *This figure visually demonstrates the advantage of taking into account the structure of the underlying image regions: A circle (left) has been rendered in ASCII using luminance information (middle left), using the SSIM-based mapping (middle right), and with the tree-based approach (right).*

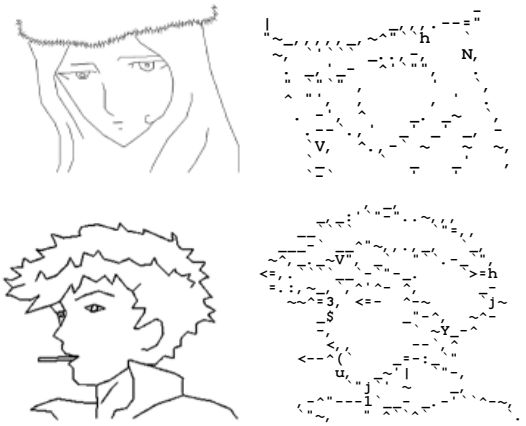

Figure 8: *Low resolution images from [XZW10] rendered in ASCII with our method.*

ing visually more pleasing results than the luminance-based method with dithering.

Figure 7 displays classical works of art in ASCII. We omitted the original images on purpose to give the article an old fashioned feel. These experiments demonstrate that the tree-based approach is suitable for producing high resolution ASCII art; the renderings are sharp and the tone is preserved. However, as pointed out in [XZW10], rendering low text-resolution ASCII art is more difficult, and our method does not perform well in this case. This is illustrated in Figure 8, where due to the limited resolution, the depiction of very fine spatial detail is problematic. We exploit the rendering speed of our method to reduce this drawback by taking human perception into account, similarly to [DER*10]: the input is rendered and displayed multiple times each second with small randomized displacements. Consequently, ASCII images displayed on a computer screen look much better than the ones on paper, as shown in the accompanying video.

Figure 9 compares image mosaics generated with different methods. The visual quality of the tree-based approach mosaic is higher than that of the one generated with average color descriptor ($n = 1$), and is similar to the ground truth

Method	Mean score μ	Standard deviation	95% confidence interval for μ	
			$\mu - \delta$	$\mu + \delta$
$n = 1$	4.89	1.73	4.72	5.06
$n = 3$	6.75	1.57	6.59	6.90
Tree-based	6.24	1.46	6.09	6.38

Table 2: *Survey statistics for image mosaics.*

[KSRY13] ($n = 3$). Table 2 summarizes the scores between seven source images and their corresponding image mosaics rendered with three different methods. We can see that the score of the tree-based method approaches that of its ground truth target ($n = 3$), and is higher than the average color descriptor ($n = 1$). This shows that the learning capacity of the tree is sufficient for our application. The rendering of the Baboon image can be seen in Figures 10. Figures 11 and 12 shows some failure cases. The next section discusses large processing speed gains offered by our method.

4.3. Processing speed analysis

Table 3 presents processing times on various devices required to convert a 640×480 pixel image to ASCII. All computations were performed on a single CPU core available on the device although all methods can be easily parallelized. This can be achieved, for example, by transforming each row of the image to ASCII in a separate thread. The computation times scale linearly with the number of pixels in the input image and the depth of the tree. The tree-based method outperforms the standard approaches because the tree needs to analyze only a relatively small subset of pixels within each image region in order to replace it with a font glyph. The method is more than two times faster than the luminance-based approach which serves as a baseline (represented by AAlib [aal]). Takeuchi et al. [TTIN13] implemented their method on a GPU and achieved real-time processing speeds. However, our method is around 20 times faster while running on a single core of a CPU. Furthermore, our method is orders of magnitude faster than other approaches reported in the literature [XZW10, OR08]. Table 4 shows the processing speed required to convert a 640×480 pixel image into an image mosaic. Note that the difference in processing speed for ASCII art and image mosaic rendering comes (partly)

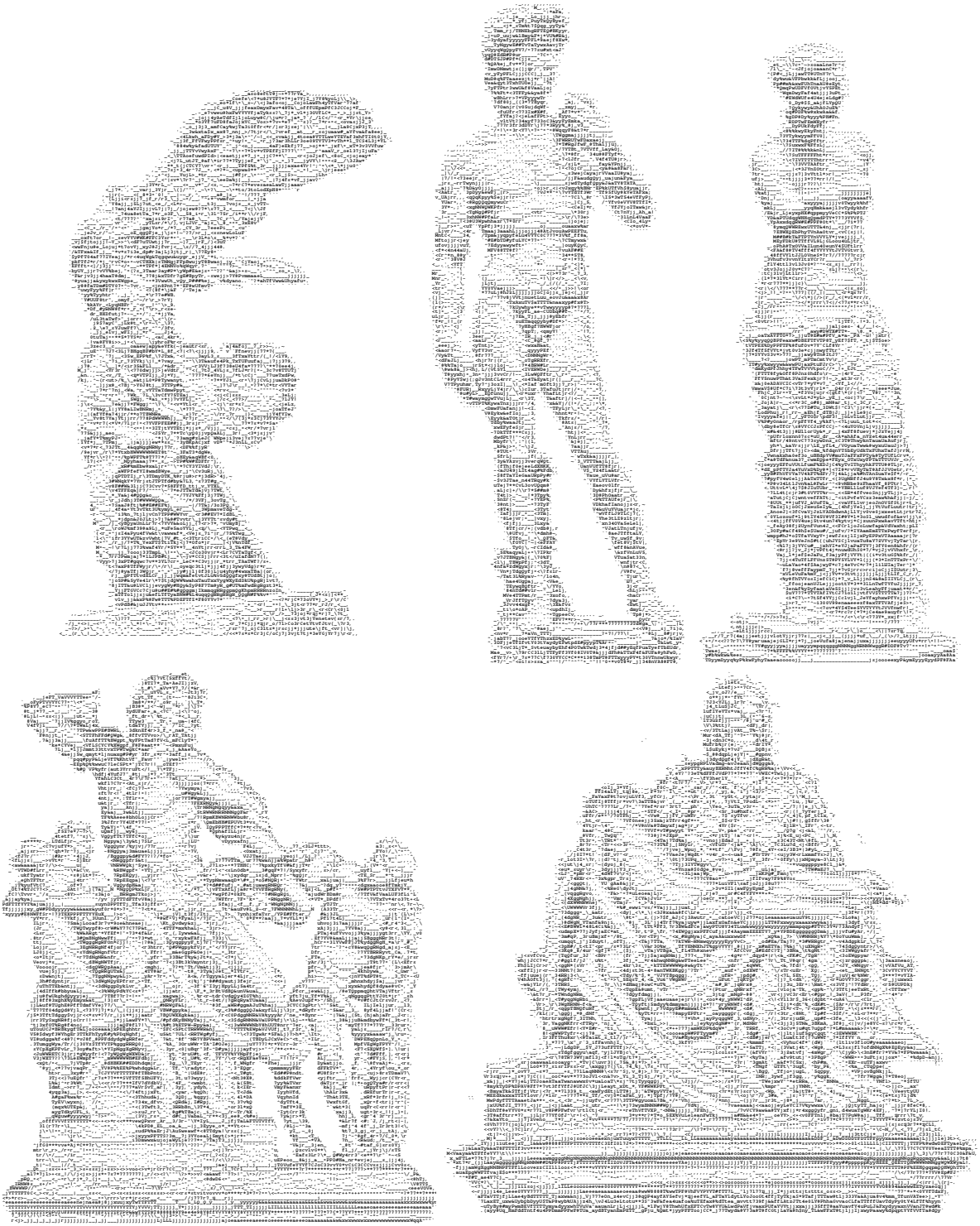

Figure 7: *Famous masterpieces of sculpture rendered in ASCII. Original images are provided in supplementary material.*

Figure 9: *An image of flowers (upper left) rendered with flower tile codebook by setting $n = 1$ (upper right), $n = 3$ (lower left) and the decision tree-based approach (lower right).*

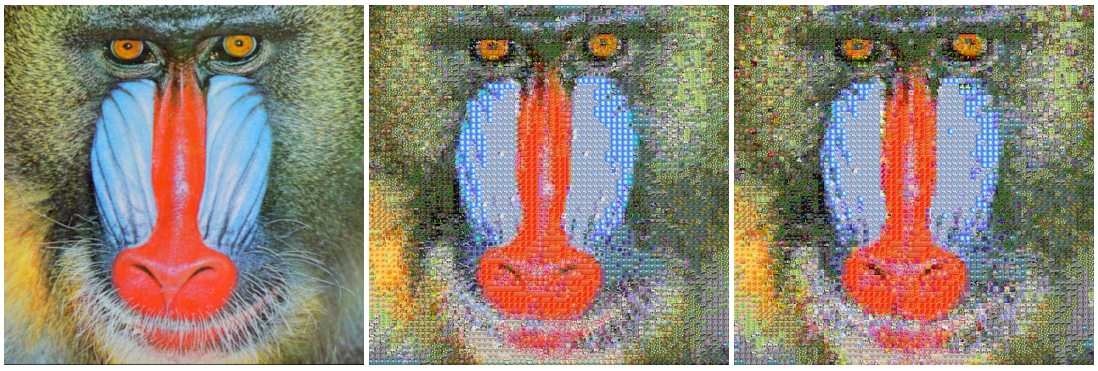

Figure 10: *The Baboon image (left), ground truth rendering with $n = 3$ (middle), and the tree-based one (right).*

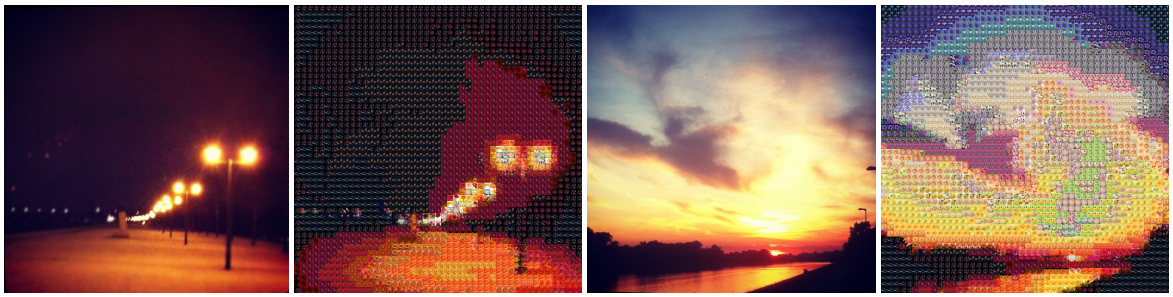

Figure 11: *Rough color transitions may occur due to the limited codebook size.*

Device	CPU	Time [ms]		
		Luminance-based	SSIM-based	Tree-based
Laptop	2.4GHz Core i7-4700MQ	1.34	93.07	0.55
iPhone 5	1.3GHz Apple A6	7.24	342.12	1.42
iPad 2	1GHz ARM Cortex-A9	19.84	484.28	2.54
iPhone 4S	800MHz ARM Cortex-A9	24.76	605.71	3.15

Table 3: *Average times required to transform a* 640×480 *greyscale image into ASCII art using a single core available on the device.*

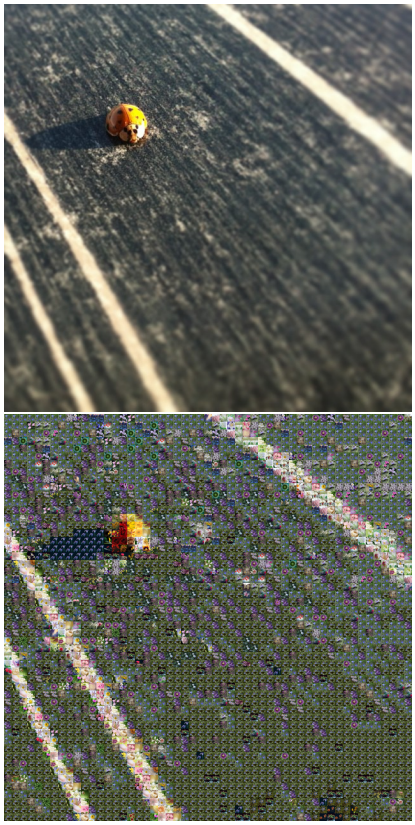

Figure 12: *The depiction of fine details is reduced due to the limited resolution and codebook size.*

Device	Time [ms]		
	n=1	n=3	Tree-based
Laptop	7.42	31.90	0.25
iPhone 5	33.36	252.43	0.58
iPad 2	88.32	679.12	1.01
iPhone 4S	118.12	913.94	1.21

Table 4: *Average times required to transform a* 640×480 *color image into an image mosaic using a single core available on the device.*

from the different tile size. Our method is more than 100 times faster than other approaches [KSRY13, Tra99, ZNZ03] that utilize only the CPU.

5. Possible extensions

When the codebook is very large and high accuracy is needed, a potential strategy to increase the approximation accuracy of our approach is to learn multiple trees with low correlation between them (i.e., a random forest [Bre01]). For example, this can be achieved by learning each tree on a separate training set. During the rendering of a grid cell, each tree outputs a candidate tile from the codebook. Then, these tiles are ranked according to their similarity with the grid cell (using equation 1 or 2), and the best candidate is picked. This procedure is repeated for each cell in the image. The computation time scales linearly with the number of trees. However, we did not find this procedure necessary, as was also suggested by the surveys in tables 1 and 2.

As noted by Tran [Tra99], a low tile variety within a local region makes a mosaic monotonous. This issue is clearly seen in Figure 11 in which neighboring tiles are usually the

same. In case of video streams coming from webcams, this is hardly noticeable because the video frames are noisy and this results in a low probability of two neighbouring tiles being the same. In the general case, this issue can be improved by learning two decision trees on separate codebooks in a preprocessing step. During rendering, each grid cell in the input image is assigned a *black* or *white* label in a checkerboard pattern style (inspired by the duplicate reduction procedure from [KSRY11]). Cells labeled as *black* are processed with the first tree and cells labeled as *white* are processed with the second tree. In this way, neighbouring cells always have different tiles assigned to them (provided that the trees were learned on disjoint codebooks), while the computation time remains the same. Another possibility is to use a single tree that keeps two labels in its terminal nodes (the best and the second best one, assigned in the training process), and alternate between them during rendering. This procedure greatly reduces the probability of having same neighbouring tiles, as illustrated in Figure 13. Of course, modifications are possible which would ensure even greater tile diversity.

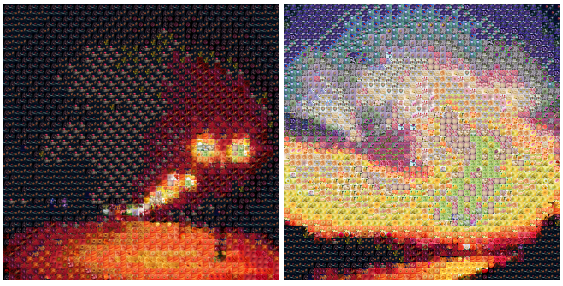

Figure 13: *A modified duplicate reduction procedure from [KSRY11] ensures low probability of same neighbouring tiles in a 4-connected neighbourhood. This ensures greater diversity when the mosaic is seen from a short distance.*

6. Conclusions

The main contribution of our work is a simple and elegant method for rendering ASCII art and image mosaics at very high speed. The method can be easily extended to include other similarity mappings and other tile codebooks than the ones used in this paper. The implemented engine can achieve real-time frame rates on mobile devices. To the best of our knowledge, this has not been achieved before. Our framework can be used as a non-photorealistic rendering method for computer games or as a video display technique on computers without a graphical user interface. We have made the source code publicly available: https://github.com/nenadmarkus/n3ar.

Acknowledgements

This research is partially supported by Visage Technologies AB, Linköping, Sweden, by the Ministry of Science, Education and Sports of the Republic of Croatia, and by the European Union through ACROSS project, 285939 FP7-REGPOT-2011-1.

References

[aal] Aa-lib. http://aa-project.sourceforge.net/aalib/. Accessed on March 2014. 2, 6

[BBFG07] BATTIATO S., BLASI G. D., FARINELLA G. M., GALLO G.: Digital mosaic frameworks – an overview. *Computer Graphics Forum* (2007). 2

[BG03] BLUNDO C., GLANDI C.: Hiding information in image mosaics. *Computer Journal* (2003). 2

[BG08] BLUNDO C., GLANDI C.: Creating image mosaics with a surrounding matching scheme. *Optical Engineering* (2008). 2

[BP05] BLASI G. D., PETRALIA M.: Fast photomosaic. In *Poster proceedings of ACM/WSCG2005* (2005). 3, 4

[Bre01] BREIMAN L.: Random forests. *Machine Learning* (2001). 9

[CFP*05] CANTONE D., FERRO A., PULVIRENTI A., REFORGIATO D., SHASHA D.: Antipole tree indexing to support range search and k-nearest neighbor search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering* (2005). 3, 4

[CJKK13] CHOI Y.-S., JUNG S., KIM J. W., KOO B.-K.: Real-time video photomosaics with optimized image set and gpu. *Journal of Real-Time Image Processing* (2013). 3

[DER*10] DIDYK P., EISEMANN E., RITSCHEL T., MYSZKOWSKI K., SEIDEL H.-P.: Apparent display resolution enhancement for moving images. *Transactions on Graphics (Proc. SIGGRAPH 2010)* (2010). 6

[FR98] FINKELSTEIN A., RANGE M.: Image mosaics. In *Electronic Publishing, Artistic Imaging, and Digital Typography*, Lecture Notes in Computer Science. 1998. 2

[GG92] GERSHO A., GRAY R. M.: *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992. 4

[HKM*97] HUANG J., KUMAR S. R., MITRA M., ZHU W.-J., ZABIH R.: Image indexing using color correlograms. In *CVPR* (1997). 3

[HTF09] HASTIE T., TIBSHIRANI R., FRIEDMAN J.: *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer New York Inc., 2009. 4

[KGFC02] KLEIN A. W., GRANT T., FINKELSTEIN A., COHEN M. F.: Video mosaics. In *Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering* (2002). 2

[KP02] KIM J., PELLACINI F.: Jigsaw image mosaics. *Transactions on Graphics (Proc. SIGGRAPH 2002)* (2002). 2

[KSRY11] KANG D., SEO S., RYOO S., YOON K.: A parallel framework for fast photomosaics. *IEICE Transactions* (2011). 3, 10

[KSRY13] KANG D., SEO S., RYOO S., YOON K.: A study on stackable mosaic generation for mobile devices. *Multimedia Tools and Applications* (2013). 2, 6

[LS99] LEE D. D., SEUNG H. S.: Learning the parts of objects by non-negative matrix factorization. *Nature* (1999). 2

[MJN11] MIYAKE K., JOHAN H., NISHITA T.: An interactive system for structure-based ascii art creation. In *Proc. NICOGRAPH International 2011* (2011). 2

[OK08] ORCHARD J., KAPLAN C. S.: Cut-out image mosaics. In *Proceedings of the 6th International Symposium on Non-photorealistic Animation and Rendering* (2008). 2

[OR08] O'GRADY P. D., RICKARD S. T.: Automatic ascii art conversion of binary images using non-negative constraints. In *Proc. Irish Signals and Systems Conference* (2008). 2, 6

[PQW*08] PANG W.-M., QU Y., WONG T.-T., COHEN-OR D., HENG P.-A.: Structure-aware halftoning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2008)* (2008). 2

[RPD10] ROSTEN E., PORTER R., DRUMMOND T.: Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2010). 4

[Sil97] SILVERS R. S.: Digital composition of a mosaic image, 1997. US Patent 6,137,498. URL: http://www.google.com/patents/US6137498. 2

[Tra99] TRAN N.: Generating photomosaics: an empirical study. In *ACM symposium on Applied computing* (1999). 1, 2, 3, 6, 9

[TTIN13] TAKEUCHI Y., TAKAFUJI D., ITO Y., NAKANO K.: Ascii art generation using the local exhaustive search on the gpu. In *2013 First International Symposium on Computing and Networking* (2013). 2, 6

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing* (2004). 1, 3

[XZW10] XU X., ZHANG L., WONG T.-T.: Structure-based ascii art. *Transactions on Graphics (Proc. SIGGRAPH 2010)* (2010). 2, 6

[ZNZ03] ZHANG Y., NASCIMENTO M. A., ZAIANE O. R.: Building image mosaics: an application of content-based image retrieval. In *International Conference on Multimedia and Expo* (2003). 2, 3, 6