EG UK Theory and Practice of Computer Graphics (2008)
Ik Soo Lim, Wen Tang (Editors)

# A Fast Algorithm for Painterly Rendering on Mobile Devices

R. Mukundan, C. Han

Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand.

**Abstract**

*With the rapid growth of mobile graphics applications, non-photorealistic rendering algorithms developed particularly for devices with limited processor capabilities have become important in the areas of games design and augmented reality. This paper presents a fast painterly rendering algorithm suitable for implementation on mobile phones. Connected components in an image are identified and stored in an index buffer, using a sequential scan. Most of the subsequent processing is done only on this index buffer that contains one integer value per pixel. The proposed method does not use recursive procedures, complex floating-point computations, or texture processing functions. The painterly rendered effect is produced by suitably modifying the boundary of connected components and highlighting edges using entries from the index buffer. The paper presents the theoretical framework for the algorithm, implementation aspects and results.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation

## 1. Introduction

Non-photorealistic rendering (NPR) styles have been extensively used in games, cartoon animation, and presentation graphics. Painterly rendering algorithms form a subclass of NPR algorithms, where stylistic depictions of two-dimensional images are produced using simulated brush strokes [Her98], [HJO01] or sketches [GG01]. Painterly effects can also be generated using particle systems [Mei96]. With recent advances in the areas of graphics hardware specifications and shader programming, impressive artistic effects could be generated using the GPU [VBT*06].

Painterly rendering algorithms typically use shape feature descriptors to identify, transform and map brush stroke parameters based on regions of similar color [KS05], [ON05]. Recursive connected component labeling methods can be used to obtain regions of nearly the same color value. Shape parameters can them be extracted from these regions using global feature descriptors like moment functions. Such moment based algorithms have been recently proposed and successfully used for painterly rendering [NV02], [SY00]. Elegant methods for automated painterly rendering have also been developed using computer vision techniques for digitally simulating brush strokes [GCS02]. Another computer vision based approach that uses eye-tracking data for painterly rendering can be found in [SD02].

NPR algorithms find interesting applications on mobile devices such as painterly rendering of captured images, mobile games, puzzles and artistic rendering of three-dimensional objects. Another emerging application area for NPR algorithms is mobile Augmented Reality [HS04]. The increasing importance of NPR in the field of mobile graphics motivates us to develop fast and efficient methods suitable for processors with limited power and memory. This paper presents a method that is derived from the moment-based algorithm using connected components [OMB06]. It replaces complex data structures and procedures with simple functions that can be easily evaluated on a mobile phone. For example, recursive functions for connected component labeling have been replaced with iterative procedures involving a sequential scan. The computations of moment functions, dither images and stroke density are not performed. Instead, region

boundaries are modified using a smoothing function, and edges enhanced using a weighted combination of color values on either side of the edge. Most of the operations are performed on an index buffer containing only integers. The method in [OMB06] uses geometric moments as shape descriptors for mapping brush stroke images. The proposed method, however, aims to produce artistic styles using a series of simple operations on an index buffer, without generating additional brush stroke images. There are three main parts to the proposed algorithm:

1. Connected component labeling
2. Boundary extraction and smoothing
3. Region and edge enhancement

The following sections (Sections 2-4) describe the above methods in detail. Some implementation aspects are discussed in Section 5. Concluding remarks and an outline of possible future extensions are given in Section 6.

## 2. Sequential Connected Component Labeling

A majority of painterly rendering algorithms work on regions of nearly the same color, much the same way as an artist would paint an image by first identifying brush stroke regions of same color. Region based painterly rendering algorithms can be found in [GC02], [Her98], [SY00] and [OMB06]. Several image processing techniques, from simple to complex, can be used for region segmentation.

As a first step towards developing a simplified algorithm for mobile devices, we eliminate recursive structures that take up both computational time and stack space. In the sequential method, connected components are identified with the help of an index buffer that stores thresholded color indices obtained in a single top-to-bottom, left-to-right scan of the image. A color table containing the correspondence between indices, color values, and the total number of pixels containing each index is also maintained (Figure. 1). The iterative algorithm [JKS95] uses a sequential scan visiting each pixel only once, and comparing its color value with those stored against its neighbors indices as shown in Figure 2. The order in which this comparison is made affects the values stored in both the index buffer and the color table.

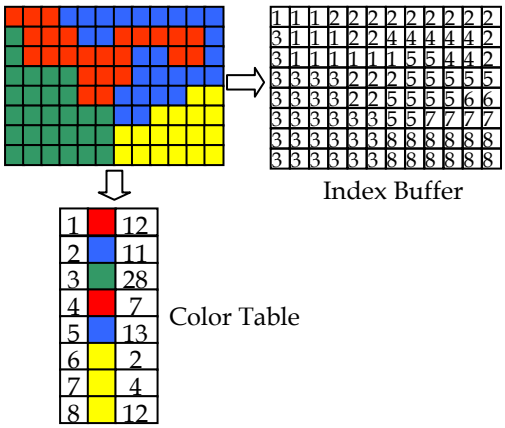



**Figure 1:** *An example showing the index buffer and color table produced by the sequential scan algorithm.*

It may be noted that the order of comparison also affects the way in which a single connected component is split into multiple components where either the *x*-monotone or *y*-monotone properties are not satisfied.

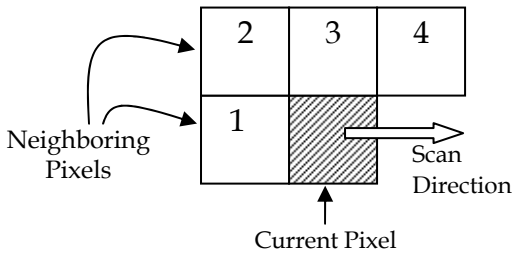


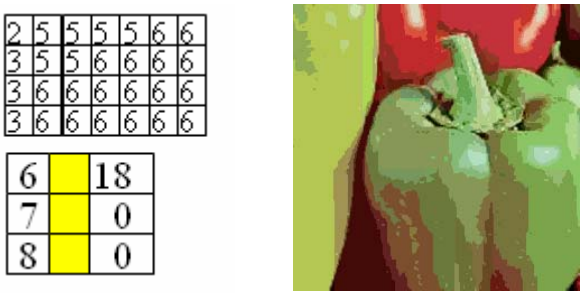

**Figure 2:** *Order in which neighboring pixels are compared with the color value of the current pixel.*

For all practical purposes, the sequential algorithm gives good results, except for the fact that there may exist several indices in the color table with the same color. This is not a major issue, as the edge generated between the corresponding connected components of the same color that border each other, will not be visible. However, an undesirable artifact produced by the sequential scanning algorithm is a set of distinctly visible horizontal streaks of colors, as can be seen in Figure 3.

(a) Original Image (b) Post-sequential scan

**Figure 3:** *Sequential scan produces horizontal streaks*

An effective solution to this problem can be obtained by performing a "reverse scan" immediately after completing each row of the image. Moving from right to left, the index of each pixel is compared with that in the previous row, and the larger value is replaced with the smaller if the color values match. The total number of pixels stored against the corresponding indices in the color table is also simultaneously updated. Thus with the modified algorithm, each row is scanned twice (once in each direction), updating both the index buffer and the color table. However, the reverse scan can be performed using only the index buffer and the color table. The improvement in the result can be clearly seen in Figure 4.




(a) Modified index buffer (b) Modified image

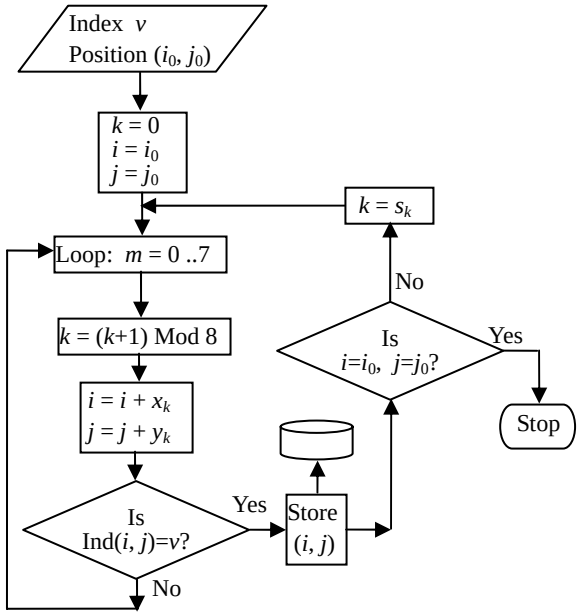
**Figure 4:** *Effects produced using reverse scan*

The bottom-right portion of the index buffer in Figure 1 shows three horizontal segments corresponding to color yellow, that were produced by the sequential scan algorithm. The reverse scan rectifies this problem, and the modified portions of the index buffer and color table are shown in Figure 4(a). The modified version of Figure 3(b) after performing reverse scan on the image, is shown in Figure 4(b).

## 3. Boundary Extraction

The index buffer and the color table together form a convenient data structure for fast boundary extraction and the determination of region size. Since the index buffer contains only integers, the boundary between two regions can be easily identified using a sequential boundary following algorithm outlined below.

For each index value $v$, a sequential scan of the index buffer returns the first pixel position $(i_0 , j_0)$ where the index is found. Since this is clearly a boundary pixel for the region of index $v$, we can start an iterative boundary following algorithm from this point, each time searching for the next boundary pixel. The boundary pixels are then stored in an array for smoothing, edge processing and enhancement. A flow diagram of the boundary following algorithm that traces the boundary of a region in an anti-clockwise sense, is given in Figure 5.




| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $x_k$ | –1 | –1 | 0 | 1 | 1 | 1 | 0 | –1 |
| $y_k$ | 0 | –1 | –1 | –1 | 0 | 1 | 1 | 1 |
| $s_k$ | 6 | 6 | 0 | 0 | 2 | 2 | 4 | 4 |

**Figure 5:** *Boundary following algorithm*

The boundary following algorithm shown above (Figure 5) uses an 8-connected neighborhood for searching for the next boundary pixel, given the current pixel on the boundary. In the figure, $k$ represents one of the eight directions, and $(x_k, y_k)$ represents the offset vectors from the current pixel that defines the current direction (Figure 6). The variable $s_k$ represents the direction in which the search starts, depending on the current direction $k$.

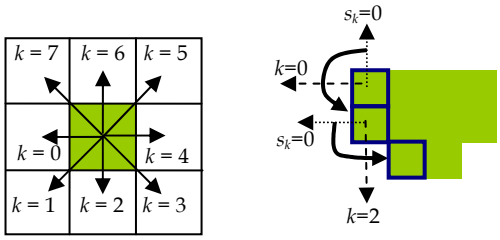



**Figure 6:** *Search directions relative to the current pixel*

For the first identified boundary pixel $(i_0 , j_0)$, the value of $k$ is set to 0, so that the search for the next boundary pixel in the neighbourhood of $(i_0 , j_0)$ starts from $(i_0 , j_0-1)$ in the anti-clockwise direction. This process of sequentially identifying boundary pixels is illustrated in Figure 6. Figure 7 shows the test image with the boundary points marked for each connected component.

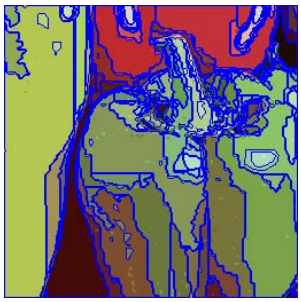


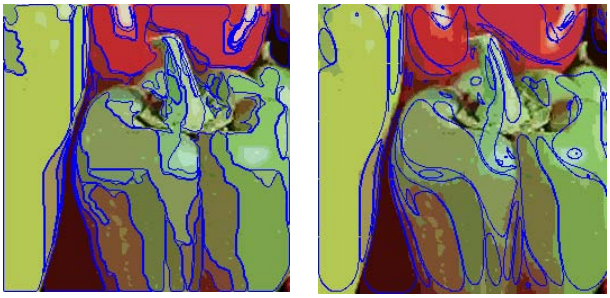
**Figure 7:** *Boundary extraction using index buffer*

The stored boundary points $(i_p , j_p)$, $p$=0..$N_e$–1, can be further processed using a linear running average operator to get a smooth boundary ($N_e$ denotes the total number of boundary pixels for the current region). An $n$-point averaging scheme transforms a boundary point $(i_p , j_p)$ using the formula

$$i'_p = \frac{1}{n}\left( \sum_{k=0}^{n/2-1} i_{p-k} + \sum_{k=1}^{n/2} i_{p+k} \right) \qquad (1)$$

$$j'_p = \frac{1}{n}\left( \sum_{k=0}^{n/2-1} j_{p-k} + \sum_{k=1}^{n/2} j_{p+k} \right) \qquad (2)$$

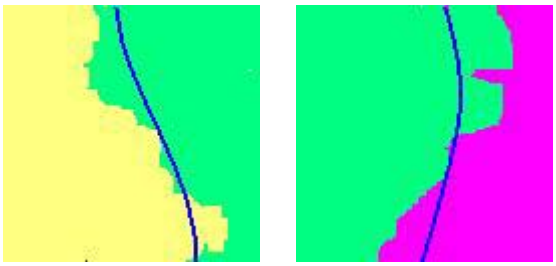
Results of applying the above transformation to boundary pixels for $n$ =16 and $n$ = 32 are shown in Figure 8. Choosing a power-of-2 value for $n$ helps in replacing the division in Equations (1), (2) by a bit-shift operation.



(a) $n$ = 8 (b) $n$ = 32

**Figure 8:** *Boundary smoothing using n neigbouring points*

The color values inside a region will have to be adjusted near the boundary, to match the smoothed edge. This requirement for region adjustment arises when $n$ is large, and can also be seen clearly in Figure 8(b). Two cases to be considered are shown in Figure 9, with the processed region represented by green color, and the smoothed edge in color blue. Since the boundary is always traversed in the anti-clockwise sense, a downward direction in the boundary indicates a left edge, (Figure 9a) while an upward direction indicates a right edge (Figure 9b). Using this classification of edges, we easily adjust the indices of the surrounding pixels to snap the region to the smoothed edge.



(a) Left edge (b) Right edge

**Figure 9:** *Region mismatch with respect to smoothed edge*

The corresponding pseudo codes of the algorithm are given in Figures 10, 11 respectively. In both these pseudo-codes, the current region index is assumed to be $v$ , and the current pixel position $(i, j)$. In order to minimise the amount of redundant

comparisons at this stage, it is preferable to preprocess the points on the smoothed edge, and eliminate duplicate boundary points created as a result of averaging and truncation.

```
IF (edge==LEFT) {
 IF ( ind(i–1, j) == v){
   Scan towards left on the same row
   and get index k such that
   ind(k, j)≠ v
   FOR m = k+1..i–1:
       set ind(m,j)=ind(k,j)
 }
 ELSE IF ( ind(i+1, j) ≠ v){
   Scan towards right on the same row
   and get index k such that
   ind(k, j)== v
   FOR m = i+1..k–1: set ind(m,j)=v
 }
}
```

**Figure 10:** *Adjustment of region indices around left edge*

```
IF (edge==RIGHT) {
 IF ( ind(i+1, j) == v){
   Scan towards right on the same row
   and get index k such that
   ind(k, j)≠ v
   FOR m = i+1..k–1:
       set ind(m,j)=ind(k,j)
 }
  ELSE IF ( ind(i–1, j) ≠ v){
   Scan towards left on the same row
   and get index k such that
   ind(k, j)== v
   FOR m = k+1..i–1: set ind(m,j)=v
 }
}
```

**Figure 11:** *Adjustment of region indices around right edge*

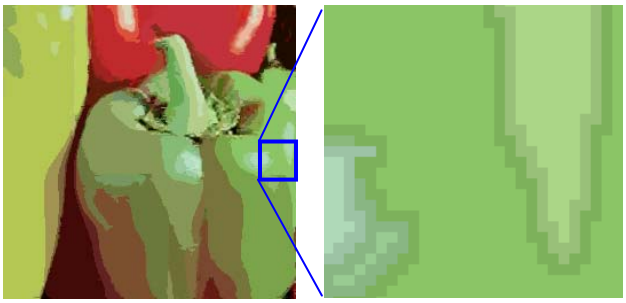
In both the above cases, the total number of pixels stored against the modified indices is also simultaneously updated in the color table. The result of this operation on the original image is shown in Figure 12 below.

**Figure 12:** *Adjustment of color indices surrounding a smoothed edge*

## 4. Region and Edge Enhancement

Depending on the color threshold used in the connected component labeling algorithm, an image can contain several small regions or specks that need to be merged with the surrounding region. Several small patches of different colors can be seen in Figure 12. Region enhancement involves the process of removing these regions, by contracting the boundary to one pixel. In each iteration, the index of a boundary pixel is replaced with that of its neighbor in the index buffer, and the boundary updated. A region can be identified as "small" by checking the color table for the total number of pixels in that region.

Unlike previous operations, edge enhancement is done while generating the output image. For each pixel that is rendered, its index is compared with its neighbors in the index buffer. If its index is different from any of its neighbor's indices, it can be classified as an edge pixel. The color of an edge pixel is darkened by multiplying its red, green and blue components by a value between 0.9 and 1.0. Figure 13 shows the results of region and edge enhancement operations.



(a) Region enhancement (b) Edge enhancement

**Figure 13:** *Effects produced by region and edge enhancements*

The test image after both region and edge enhancements is shown in Figure 14. The stylistic effects generated by the painterly rendering algorithm consisting of all the procedures described above can be seen by comparing Figure 3(a) with Figure 14.

**Figure 14:** *The final result of the NPR algorithm*

**Figure 15:** *Implemented version of the algorithm with user interfaces for image selection and display, on Java WTK emulator.*

## 5. Implementation Aspects

The algorithms described in this paper can be easily implemented in either C++ or Java using mobile graphics APIs, and do not require complex data structures or additional libraries for matrix/texture operations. Our implementation uses the interactive graphics 3D API (M3G: JSR184) developed for J2ME [SM07]. The whole project was developed using NetBeans Interactive Development Environment and the Sun Java Wireless Toolkit (WTK) [SDN07]. Most mobile devices available today support MIDP/M3G environment, and the theoretical framework presented in the paper can be implemented as a MIDlet that can be deployed on a mobile phone. The Java WTK also supports smartphone emulators using which the developed MIDlets can be tested on a desktop computer. Figure 15 shows the implementation of the algorithm presented in the paper along with the necessary user interfaces for image selection and display.

The Mobile Media API (MMAPI - JSR 135) [SDN05] extends the functionality of the J2ME platform, and allows access to native multimedia services available on a mobile device. An NPR application as the one presented in this paper will need to read and write data to phone memory, and this process of data I/O can become complicated due to security constraints imposed by the underlying operating system. The MM-API can be used to capture an image from the built-in camera of a cell phone. The output image will need to be stored in the phone memory for further editing and viewing.

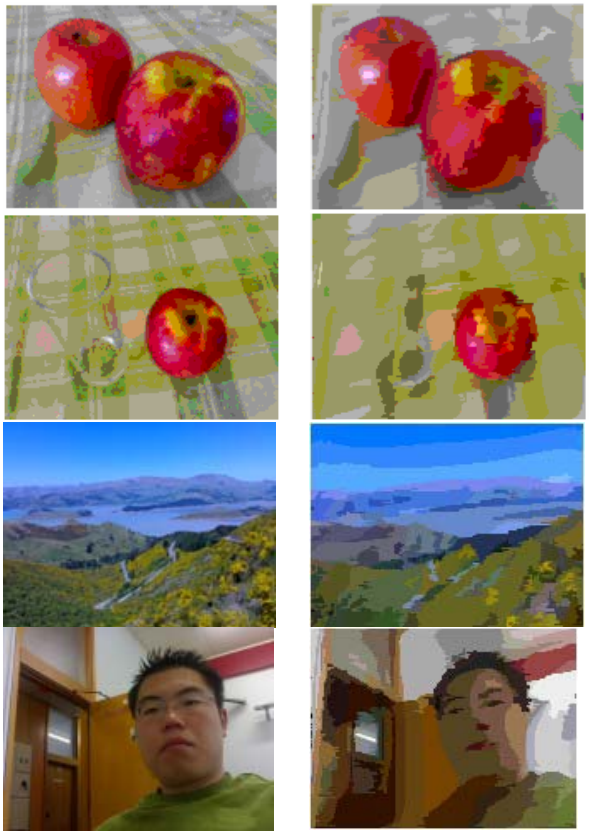
Figure 16 (on next page) shows some of the images generated using the proposed method for painterly rendering. As can be seen from these examples, general scenery and objects can be processed with satisfactory results using the proposed algorithm. The transparency of objects cannot be preserved under painterly rendering, but some of the gross level shape characteristics can be made visible. Subtle variation of colour tones across a region can sometimes lead to undesirable artefacts (eg. face image) and also merging of an object's color with its background.

## 6. Concluding Remarks

This paper has presented a low-complexity painterly rendering algorithm suitable for implementation on mobile devices. A sequential connected component labelling algorithm is used to represent regions of similar color in an index buffer and a color table. All subsequent processing is done on the index buffer to minimise computational complexity. The proposed method does not use recursive structures, complex matrices, large number of floating point operations, or texture mapping functions. These features make the algorithm suitable for implementation on mobile devices with limited processor power and memory.

Possible extensions of the work to improve rendering quality and speed are discussed below. We used a simple Manhattan distance measure in the connected component algorithm. This could possibly be replaced with a difference of hue values. Region enhancements could be tried using

morphological operators. We used a boundary contraction algorithm as it could be very easily implemented using the computed boundary pixels. In some cases it may be preferable to retain small color regions in the image, such as specular highlights and reflections. Instead of completely getting rid of texture mapping functions, it may be possible to use procedural textures to generate effects of brush strokes. Edge enhancements could be made using second order edge operators instead of a simple intensity scale factor.



(a) Source images (b) Output images

**Figure 16:** *Test images used for experimental analysis*

## 7. Acknowlegements

This research work was supported by UOC College of Engineering Strategic Fund (2007). The authors are grateful to the reviewers of this paper for their valuable comments and suggestions.

## References

[GCS02] GOOCH B., COOMBE G., SHIRLEY P.: Artistic vision: Painterly rendering using computer vision techniques. In *Proc. Second Intl. Symposium on Non-Photorealistic Animation and Rendering NPAR02*, (2002), pp. 83-91.

[GG01] GOOCH B., GOOCH A.: *Non-Photorealistic Rendering*, A.K.Peters Ltd., 2001.

[HS04] HALLER M., SPERL D.: Real-time painterly rendering for MR applications. In *Proc. Intl Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* (2004), pp. 30—38.

[Her98] ] HERTZMANN A.: Painterly rendering with curved brush strokes of multiple sizes. In *Proc. SIGGRAPH'98*, (1998), pp. 453-460.

[HJO*01] HERTZMANN A., JACOBS C.E., OLIVER B., SALESIN D.H.: Image analogies. In *Proc. SIGGRAPH'01* (2001).

[JKS95] JAIN R., KASTURI R., SCHUNCK B.G., *Machine Vision*, McGraw-Hill, 1995.

[KS05] KOVACS L., SZIRANYI T.: Painterly rendering by automatic feature extraction. In *Proc. Joint Hungarian-Austrian Conference on Image Processing and Pattern Recognition* (2005), pp. 287-295.

[Mei96] MEIER B. J.: Painterly rendering for animation. In *Proc. Siggraph'96* (1996), pp. 477-484.

[NV02] NEHAB D., VELHO L.: Multiscale moment based painterly rendering. In *Proc. Brazilian Symposium on Computer Graphics and Image Processing SIBGRAPI* (2002), pp. 244-251.

[OMB06] OBAID M., MUKUNDAN R., BELL T.: Enhancement of moment based painterly rendering using connected components. In *Proc. Intl. Conf. on Computer Graphics, Imaging and Visualisation CGIV'06* (2006), pp. 378-383.

[ON05] OH C.S., NAM Y.H.: Oriental color-ink model based painterly rendering for real-time application. In *Lecture Notes in Computer Science 3768* (2005), pp. 970-980.

[SD02] SANTELLA A., DECARLO D.: Abstracted painterly renderings using eye-tracking data. In *Proc. Second Intl. Symposium on Non-Photorealistic Animation and Rendering NPAR02*, (2002), pp. 75-82.

[SDN05] SUN DEVELOPER NETWORK: Mobile Media API (MMAPI); JSR 135, http://java.sun.com/products/mmapi/ (2005).

[SDN07] SUN DEVELOPER NETWORK: Sun Java Wireless Toolkit for CLDC, http://java.sun.com/products/sjwtoolkit/ (2007).

[SM07] SUN MICROSYSTEMS: JSR 184: Mobile 3D Graphics API for J2ME. jcp.org/jsr/detail/184.jsp (2007).

[SY00] SHIAISHI M., YAMAGUCHI Y.: An algorithm for automatic painterly rendering based on local source image information. In *Proc. Intnl. Symp. On Non-Photorealistic Animation and Rendering NPAR* (2000), pp. 53-58.

[VBT*06] VANDERHAEGHE D., BARLA P., THOLLOT J., SILLION F. A.: Dynamic drawing algorithm for interactive painterly rendering. In *Proc. SIGGRAPH '06* (2006), pp. 100.