Eurographics Italian Chapter Conference (2007)
Raffaele De Amicis and Giuseppe Conti (Editors)

Rendering Order Optimization for SVGStat Improvement

S. Battiato, G. Puglisi

Dipartimento di Matematica e Informatica, University of Catania, Italy
Image Processing Laboratory
http://www.dmi.unict.it/~iplab

Abstract
Vector representation of digital images offers a number of advantages over the more common raster representation, such as scalability and resolution independence. Many efforts have been made to deploy scalable raster standards for photographic imagery addressed to portable applications. However, they lack the flessibility and simplicity of vector representation. Vector graphics is a new and little explored alternative to the more common representation. In this paper we present our raster to vector technique, called SVGStat, improved with a new boundaries simplification algorithm.

Categories and Subject Descriptors (according to ACM CCS): Vectorization, Segmentation, SVG.

1. Introduction

Vector representation of digital images offers a number of advantages over the more common raster representation, such as scalability and resolution independence. These features make it amenable for portable applications since it can accomodate for a wide range different displaying conditions, varying in resolution, quality, and level of detail.

Many efforts have been made to deploy scalable raster standards for photographic imagery addressed to portable applications, such as JPEG2K [JPE]. Anyway, since they have been focused on raster images, they lack the flessibility and simplicity of vector representation. On the other hand, while many applications exist to enable artists to build vector images from scratch, converting photographic imagery from raster to vector formats is a relatively new topic.

Recently, SVG (Scalable Vector Graphics), a new vector format for web deployment, has been released ([DHH02], [Qui03]). A number of applications have appeared to convert raster images to vector graphics in the SVG format (Vector Eye [VLDP03], Autotrace [Web02], Kvec [Kuh03], VISTA [PS05]). Anyway, most of these methods are devoted to synthetic images with a small colour palette and strong neat borders between image regions. They often fail to vectorialize photographic images, because they have blurred and fuzzy edges and huge colour palettes. As we already shown in ([BFP06b], [BFP06a]), our technique SVGStat, outperforms other methods both in terms of rendering quality and overall compression rate. Moreover it is based on a single input parameter that makes easy to find the correct trade-off between final perceived quality, scalability and corresponding file size. In this paper we have introduced a boundaries simplification step in the algorithm that permits us to obtain smaller file size without losing in perceived and measured quality.

The rest of the paper is organized as follows. In section 2 we briefly review the main details of SVGStat. The successive section describes the new boundary simplification step whereas section 4 reports some experimental results devoted to compare the performances of the proposed approach with respect to previous version of SVGStat. A final section closes the paper tracking also direction for future works.

2. SVGStat

SVGStat is a raster to vector technique that consists of three main steps:

- image partitioning in polygonal regions using SRM (Statistical Region Merging) [NN04];
- borders tracking of segmented regions;
- regions coding by SVG primitives.

2.1. Image Partitioning by Segmentation

Segmentation is the process of partitioning an image into disjoint and homogeneous regions. A more formal definition can be given in the following way [LM01]: let I denote an image and let H define a certain homogeneity predicate; the segmentation of I is a partition P of I into a set of N R_n, n=1, 2, ..., N, regions such that:

- $\bigcup_{n=1}^{N} R_n = I$ with $R_n \cap R_m = \emptyset$, $n \neq m$;
- $H(R_n) = true \quad \forall n$;
- $H(R_n \cup R_m) = false \quad \forall R_n$ and R_m adjacent.

Recently, thanks to the increasing speed and decreasing cost of computation, many techniques have been developed for segmentation of colour images. In particular we used the Statistical Region Merging ([NN04]) algorithm, a region growing techniques with statistical test for region fusion. This algorithm has several good features: it has a linear complexity and a single input parameter that fixes the "coarseness" of segmentation. SRM algorithm gives, for each segmented region, the list of pixels belonging to it and the related mean colour. We use this output as starting point to create a vectorial representation of image.

2.2. Contouring

To obtain a vectorial representation we have to find the border of segmented regions. This could be done more easily if we considered the pixels belonging to several groups (fig. 1). First of all pixels are divided in:

- *internal pixels*: pixels with all neighbours (in a 4-connexity scheme) belonging to the same region;
- *border pixels*: remaining pixels.

Due to the overall complexity of border regions a further setting into two groups is required:

- *close pixels*: pixels with at least an internal pixel as neighbour (in a 8-connexity scheme);
- *open pixels*: remaining pixels.

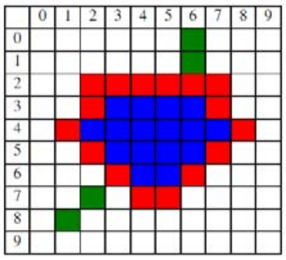


Figure 1: *An example of different kind of pixels: internal (blue), close (red) and open (green).*

After having assigned each pixel to the corresponding category we describe regions in vectorial form. In particular there are two types of curves: *close curves* and *open curves*. In both cases we could approximate their boundaries through segments with eight possible directions (fig. 2).

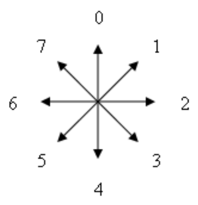


Figure 2: *Possible directions of segments that approximate the boundaries of regions.*

2.2.1. Close Curve

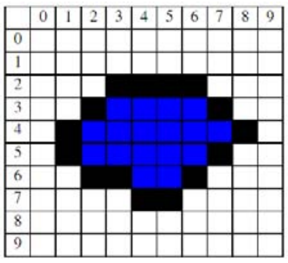


Figure 3: *An example of region with simple boundaries.*

A *close curve* is a curve made up of only *close pixels*. Initially, we consider a simple configuration (fig. 3) to explain how segments could be found from *close pixels* list. The pseudo-code that describes the algorithm is the following:

```
initialPixel = findFirstPixel();
currentPixel = findNextPixel(initialPixel);
direction = findDirection(initialPixel, currentPixel);
segment = createNewSegment(initialPixel, direction);
while (currentPixel != initialPixel){
 oldCurrentPixel = currentPixel;
 oldDirection = Direction;
 currentPixel = findNextPixel(oldCurrentPixel);
 direction = findDirection(oldCurrentPixel, currentPixel);
 if (direction != oldDirection){
  setFinalCoordinate(segment, oldCurrentPixel);
  insertInSegmentsList(segment);
  segment = createNewSegment(oldCurrentPixel, direction);
 }
}
setFinalCoordinate(segment, currentPixel);
insertInSegmentsList(segment);
```

The functions are:

- `findFirstPixel()`: it chooses the top-left pixel as initial pixel.
- `findNextPixel(currentPixel)`: it looks for the next pixel in the neighbourhood following a counter clockwise direction.

- `createNewSegment(oldCurrentPixel, direction)`: it creates a new segment with first coordinate `oldCurrentPixel` and direction `direction`.
- `setFinalCoordinate(segment, oldCurrentPixel)`: it sets the final coordinate of segment `segment` at `oldCurrentPixel`.
- `insertInSegmentList(segment)`: it adds `segment` in the list of segments that describes the *close curve*.

Our algorithm chooses the top-left pixel of curve as initial pixel and, following the boundary in counter clockwise direction, it creates the segments necessary for a vectorial description.

2.2.2. Open Curve

Even if a good segmentation algorithm should create regions with simple boundaries and not ragged this is not always the case. For this reason we have divided border pixels into two groups: *close pixels* and *open pixels*. The last are the pixels devoted to describe the ragged above mentioned.

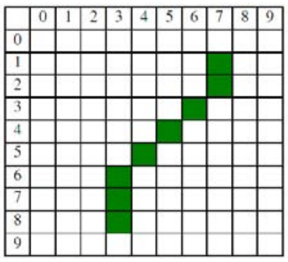


Figure 4: *An example of simple open curve.*

For simple configurations (fig. 4) it is possible make use of a simple contouring technique as above but just considering as starting point only pixels having a single neighbour. Moreover when a pixel is chosen it is deleted form the list of open pixels.

2.3. SVG Generation Code

After we have tracked the boundaries of curves it is necessary to map the data by SVG primitives. In particular a generic *close curve* could be represented in the following way:

```
<path d="M x1,x1 L x2,y2 Lx3,y3 Z" stroke ="#RRGGBB"
fill"#RRGGBB" />
```

where `x1,y1,x2,y2,x3,y3` are the vertexes coordinates and `RR, GG, BB` are respectively the hex representation of red, green, blue mean value of the region that *close curve* belong to.

An *open curve* could be represented in the following way:

```
<path d="M x1,x1 L x2,y2 Lx3,y3" stroke ="#RRGGBB"
fill="none" />
```

Open curves are not filled (`fill = "none"`) and the starting point is not equal to final point (`Z` parameter is absent).

In order to obtain small file size some optimization could be done [Wor03]:

- `path` element permits to eliminate some separator characters, to use relative coordinate (`m, l` command) and `h, v` command for horizontal and vertical lines respectively;
- `<g>` elements is used to properly ensemble common graphic properties of various primitives.

3. Boundaries Simplification

Each identified region has been properly contoured by making use of a list of segments. However, this representation is not optimal in terms of coding efficiency; in fact each border is considered twice. In order to reduce such redundancy the rendering order is used to simplify some boundaries. Just for example considering two nearby regions R_i and R_j, some borders of R_i can be extended under the image region covered by R_j with ordering such that $i<j$. Such simplification strategy is useful to reduce the overall number of points and primitives required by the classical boundary description. Just before boundaries reduction a closing operation is applied just to smooth ragged borders.

In order to obtain boundaries reduction an iteratively three step strategy has been implemented. The first simplification step aims to make convex each involved region. In particular it iterates over all segments setting as "blocked" (no further optimization can be done) those that have nearby regions already drawn. Afterwards it considers couples of not blocked consecutive segments (s_1, s_2) that form a concave region. This method considers the segment s_3 starting from the initial point of s_1 and ending on the final point of s_2; if the triangle formed by (s_1, s_2, s_3) segments can be added to current region without including parts of other regions already drawn, it simplifies the border removing s_1, s_2 and introducing the new segment s_3 (see figure 5).

The second step simplifies an oblique segment by using two segments along the main axes (see figure 6). Like in the first simplification step it is necessary to avoid to include parts of regions already drawn.

The third and final step consists of a simplification of consecutive segments with the same direction. In particular it iterates over all not blocked segments and, detecting consecutive sequences with the same direction, it replaces them with one segment (see figure 7). In the case of segment with opposite direction, this method simplifies these segments by making use of an algebraic sum of their length.

The overall technique can be summarized by the following pseudo code:

```
BorderSimplify(List RegionsList){
 OrderedRegionList = sort(RegionList);
 While(there are regions to process){
  currentRegion = getRegion(OrderedRegionList);
  nearRegionsIdList = computeNearRegionsId(currentRegion);
  count =
computeNumberNearRegionsAlreadyDrawn(nearRegionsIdList);
  if (count ==0){
   simplify = computeAndControlRect(currentRegion);
   if (simplify) simplifyWithRect(currentRegion);
   else {
    iterativelySimplify(currentRegion);
    }
  } else if (count<size(nearRegionsId))
   iterativelySimplify(currentRegion);
  } else {
   //If all nearby regions has been already drawn no
simplification can be made.
 }
}

iterativelySimplify(Region currentRegion){
 simplify = true;
 while(true){
  simplify = firstSimplification(currentRegion);
  if (!simplify) break;
  secondSimplification(currentRegion);
  thirdSimplification(currentRegion);
 }
 thirdSimplification(currentRegion);
}
```

The functions are:

- `sort(RegionList)`: it sorts, in decreasing order, the regions list according to dimension of rectangle containing the regions.
- `getRegion(OrderedRegionList)`: it returns the first element of `OrderedRegionList`;
- `computeNearRegionsId(currentRegion)`: It computes `currentRegion` nearby regions identifiers;
- `computeNumberNearRegionsAlreadyDrawn (nearRegionsIdList)`: it computes the number of regions, near `currentRegion`, already drawn;
- `computeAndControlRect(currentRegion)`: It verifies if `currentRegion` can be replaced with the rectangle containing it. This simplification has not to include visible parts of regions already drawn.
- `simplifyWithRect(currentRegion)`: It simplify `currentRegion` with the rectangle containing it;
- `firstSimplification(currentRegion)`: It implements the first simplification step described above. It returns true only when a simplification has been made.
- `secondSimplification(currentRegion)`: It implements the second simplification step described above.
- `thirdSimplification(currentRegion)`: It implements the third simplification step described above.

As it can be seen in figure 8, regions produced with the boundaries simplification step are simpler than those of the previous version of SVGStat.

4. Experimental Results

In [BFP06a] we have shown that SVGStat, without the optimizations introduced in this work, outperforms other

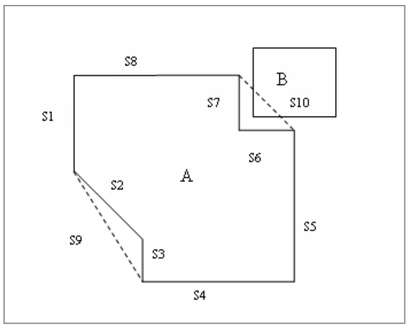


Figure 5: *First simplification step: this method adds the triangle (s2, s3, s9) to region A. Moreover the region B doesn't allow to include the triangle made up of s6, s7, s10.*

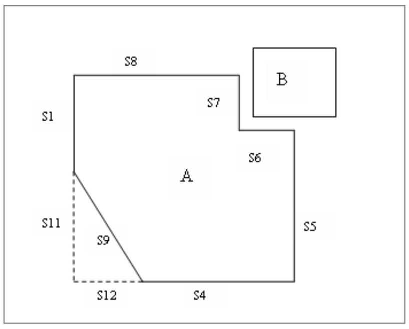


Figure 6: *Second simplification step: this method includes the triangle (s9, s11, s12) into region A.*

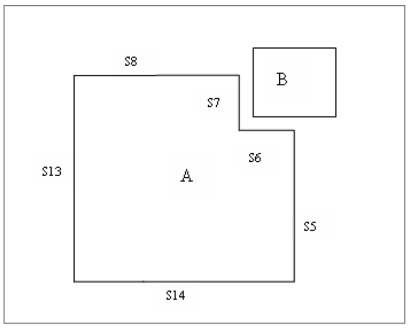


Figure 7: *Example of third simplification step; this method, in this case, simplifies segments s1, s11 and s12, s4 (see figure 6) into s13 and s14 respectively.*

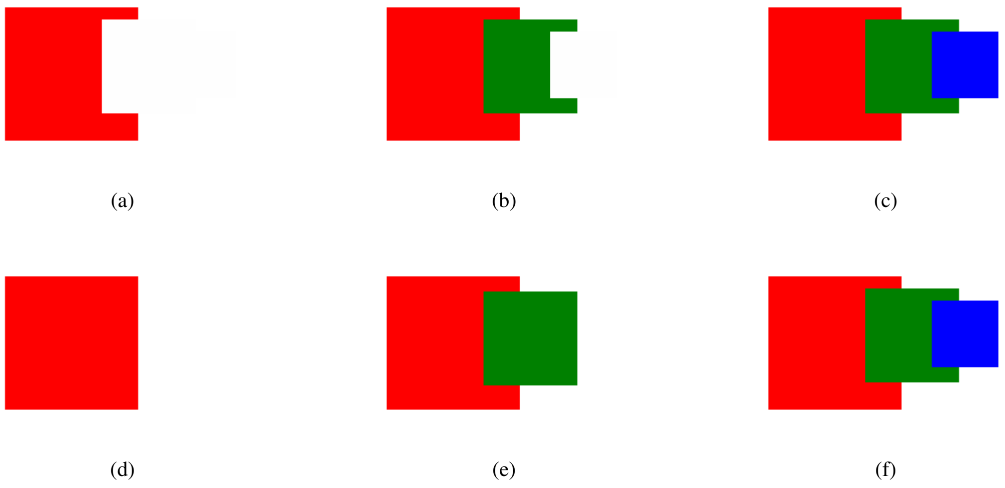


Figure 8: *Comparison between old and new rendering of regions. Regions considered by new algorithm (see d, e, f) are simpler than those of the old algorithm (see a, b, c).*

similar approaches like VectorEye [VLDP03], Autotrace [Web02], VISTA [PS05], SVGenie [BBD*05] and SWaterG [BCDN05] in particular for photographic images. Maintaining the same level of perceived (an measured by PSNR) visual quality, SVGStat [BFP06a] obtains better performances in terms of compression rate. In this paper we have proposed a new step in SVGStat that, simplifying regions boundaries, allows obtaining, without losing in visual quality (see figures 9, 10, 11), greater compression rate. In particular, considering visually agreeable images, we obtain an average gain in file size of 11,5 percent for photographic images (see figure 12) and of 21,9 percent for clip art images (see figure 13). This happens because our boundaries simplification technique is only devoted to close curves and clip arts, due to their simplicity, are described principally by this kind of curves. Moreover as shown in figure 14 the optimized approach outperforms the previous version of SVGStat also varying the quantization parameter. In fact the new SVGStat PSNR-bpp curve is always above the other.

5. Conclusion and Future Works

In this paper we have proposed a new boundaries simplification algorithm in order to improve SVGStat performances. Moreover, we have carried out several experiments showing that, without losing in visual quality of images, the new version of SVGStat obtains better compression rate. Future researches will be devoted to design advanced region merging heuristics by making also use of Bezier curves and filter enhancements.

References

[BBD*05] BATTIATO S., BARBERA G., DI BLASI G., GALLO G., MESSINA G.: Advanced SVG Triangulation Polygonalization of Digital Images. In *Proceeding of SPIE Electronic Imaging-Internet Imaging VI-* (2005), vol. 5670.1.

[BCDN05] BATTIATO S., COSTANZO A., DI BLASI G., NICOTRA S.: SVG Rendering by Watershed Decomposition. In *Proceeding of SPIE Electronic Imaging-Internet Imaging VI-* (2005), vol. 5670.3.

[BFP06a] BATTIATO S., FARINELLA G. M., PUGLISI G.: Statistical Based Vectorization for Standard VectorGraphics. In *Fifth Int.Workshop on Computer Graphics and Geometric Modeling(to appear) LNCS* (2006), vol. 5670.3.

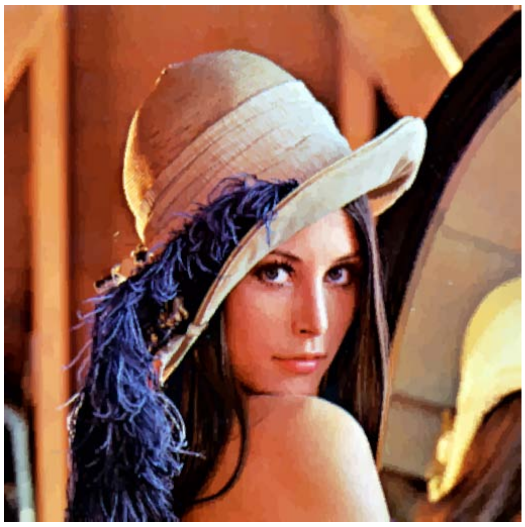
[BFP06b] BATTIATO S., FARINELLA G. M., PUGLISI G.: SVGl Vectorization by Statistical RegionMerging. In *Proocedings of Fouth Conference Eurographics Italian Chapter* (2006), Catania (Italy).

[DHH02] DUCE D., HERMAN I., HOPGOOD B.: Web 2D Graphics File Format. *Computer Graphics forum 21*, 1 (2002), 43–64.

[JPE] : *ISO/IEC JTCI/SC29/WG! N1646: JPEG2000 final committee draft v1.0.* http://www.jpeg.org/jpeg2000/.

[Kuh03] KUHL K.: Kvec 2.99, 2003. Copyright KK-Software, http://www.kvec.de.

[LM01] LUCCHESE L., MITRA S.: Color Image Segmentation: A State-of-the-Art Survey. In *Proc. of the Indian National Science Academy(INSA-A)* (Mar. 2001), vol. 67 A, pp. 207–221.


(a) Lena vectorized by old SVGStat (PSNR=28, bpp=19.72).

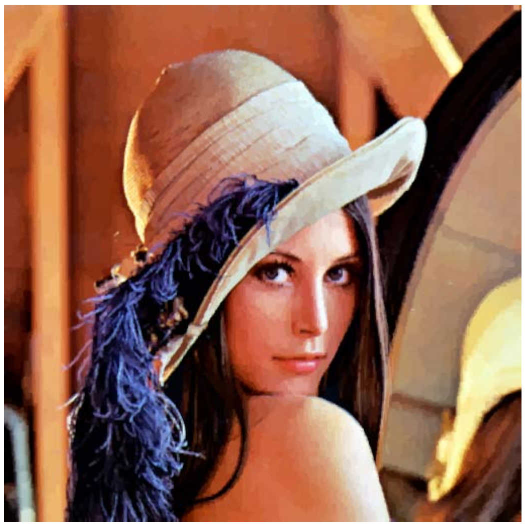

(b) Lena vectorized by new SVGStat (PSNR=27.9, bpp=17.56).

Figure 9: *Visual comparison between image Lena vectorized with our old e new solution. The quality of image produced with our techniques is the same but the optimized approach produces a smaller svg file.*

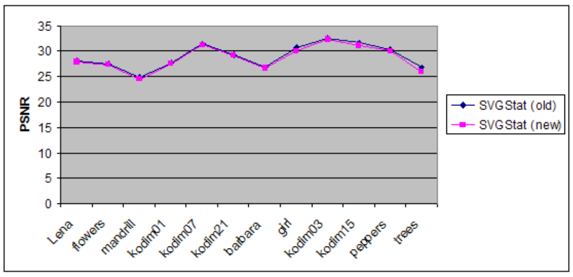


Figure 10: *PSNR comparison between old and new version of SVGStat for photographic images.*

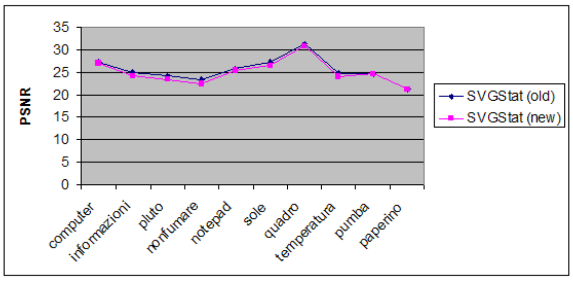


Figure 11: *PSNR comparison between old and new version of SVGStat for clip art images.*

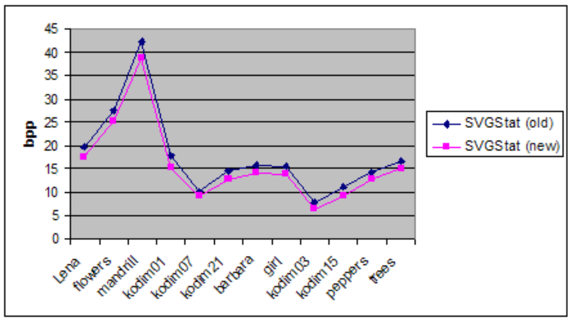


Figure 12: *Bpp comparison between old and new version of SVGStat for photographic images.*

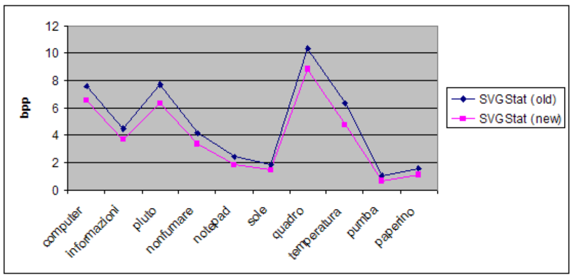


Figure 13: *Bpp comparison between old and new version of SVGStat for clip art images.*

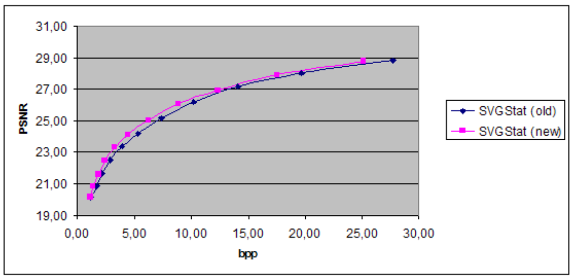


Figure 14: *Relation beetwen PSNR and bbp for Lena image in the new and old version of SVGStat.*

[NN04] NOCK R., NIELSEN F.: Statistical Region Merging. *IEEE Transaction on Pattern Analysis and Machine Intelligence 26*, 11 (NOVEMBER 2004), 1452–1458.

[PS05] PRASAD L., SKOURIKHINE A.: Raster to Vector Conversion of Images for Efficient SVG Representation. In *Proceedings of SVGOpen'05* (August 2005), NL.

[Qui03] QUINT A.: Scalable Vector Graphics. *IEEE Multimedia 3* (2003), 99–101.

[VLDP03] VANTIGHEM C., LAURENT N., DECKEUR D., PLANTINET V.: Vector eye 1.0.7.6, 2003. Copyright SIAME e Celinea, http://www.siame.com, http://www.celinea.com.

[Web02] WEBER M.: Autotrace 0.31, 2002. GNU General Public License, http://www.autotrace.sourceforge.net.

[Wor03] WORLD WIDE WEB CONSORTIUM: *Scalable Vector Graphics (SVG) 1.1 Specification*, 2003. http://www.w3.org/TR/2003/REC-SVG11-20030114/.