V International Workshop on Computer Graphics and Geometric Modeling 2006 (CGGM'2006)
University of Reading, Reading, UK (May 28-31 2006)

Statistical Based Vectorization for Standard Vector Graphics: Supplementary Material

S. Battiato, G. M. Farinella, G. Puglisi

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

Contouring

To obtain a vectorial representation we have to find the border of segmented regions. This could be done more easily if we consider 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.

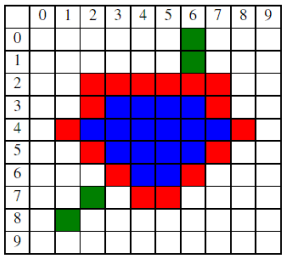

Figure 1: *An example of differend kind of pixels: internal (blue), close (red) and open (green).*

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).

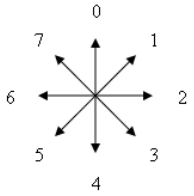

Figure 2: *Possible directions of segments that approximate the boundaries of regions.*

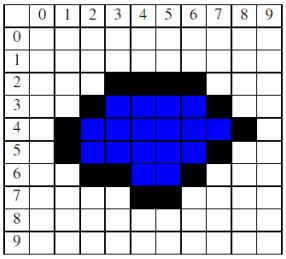

Figure 3: *An example of region with simple boundaries.*

Close Curve

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 creates the segments necessary for a vectorial description.

There are a series of complex configuration:

- regions with internal curve;
- ambiguity in the choice of next `currentPixel`;
- several *close curve* belonging to the same region;

To properly consider these cases it is needed to modify the algorithm described above in the following way:

```
while(closePixelsList contains some pixel);{
 currentCurve = createNewCurve();
 initialPixel = findFirstPixel();
 curveType = getCurveType(initialPixel);
 currentPixel = findNextPixel(initialPixel, curveType);
 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);
   addInCurve(currentCurve, segment);
   segment = createNewSegment(oldCurrentPixel, direction);
  }
 }
 setFinalCoordinate(segment, currentPixel);
 addInCurve(curve, segment);
 insertInCurvesList(currentCurve);
}
```

Some functions have been modified and other are new, in particular:

- `createNewCurve()`: it creates a new curve;
- `addInCurve(currentCurve,segment)`: it inserts in `currentCurve` the segment `segment`;
- `insertInCurvesList(currentCurve)`: it inserts `currentCurve` in the list of curves that describes the boundary of regions;
- `getCurveType(initialPixel)`: analyzing neighbours of `initialPixel` it returns the type of curve (internal or external);
- `findNextPixel(initialPixel,curveType)`: it looks for the next pixel in the neighbourhood following a clockwise or counter clockwise direction based on `curveType` value. If there is an ambiguity the pixel on the left is chosen. Moreover each pixel have a multiplicity equals to the number of regions they are near to. When a pixel is chosen its multiplicity is decremented by one and when it becomes 0 that pixel is deleted from the list of *close pixel*.

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.

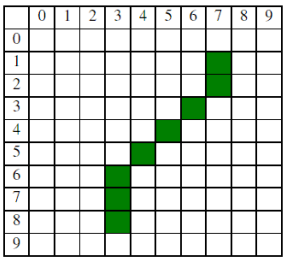

Figure 4: *An example of simple open curve.*

For simple configurations (fig. 4) we could use the following algorithm:

```
initialPixel = findFirstPixel();
currentPixel = findNextPixel(initialPixel);
direction = findDirection(initialPixel, currentPixel);
segment = createNewSegment(initialPixel, direction);
while (it is possible to find new nearby pixels){
 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);
```

It is very similar to the one described for the *close curve*, but the following function has a different behaviour:

- `findFirstPixel()`: it choices a pixel with only a neighbour as initial pixel. Moreover when a pixel is chosen it is deleted from the list of *open pixels*.

For complex configuration is needed a more sophisticated algorithm:

```
while(openPixelsList contains some pixels){
 currentCurve = createNewCurve();
 initialPixel = findFirstPixel();
 currentPixel = findNextPixel(initialPixel);
 direction = findDirection(initialPixel, currentPixel);
 segment = createNewSegment(initialPixel, direction);
 while (secondList is not empty or it's the first
execution){
  if(there isn't new currentPixel and it isn't the first
execution){
   initialPixel = getPixelFromSecondList();
   currentPixel = findNextPixel(initialPixel);
   direction = findDirection(initialPixel, currentPixel);
   segment = createNewSegment(oldInitialPixel, direction);
  }
  while (it's possible to find a new currentPixel){
   oldCurrentPixel = currentPixel;
   oldDirection = direction;
   currentPixel = findNextPixel(currentPixel);
   direction = findDirection(oldCurrentPixel, currentPixel);
   if (direction != oldDirection){
    setFinalCoordinate(segment, oldCurrentPixel);
    addInCurve(currentCurve, segment);
    segment =createNewSegment(oldCurrentPixel, direction);
   }
  }
  setFinalCoordinate(segment, currentPixel)
  addInCurve(currentCurve, segment);
 }
 insertInCurvesList(currentCurve);
}
```

Some functions have been modified and other are new, in particular:

- `findNextPixel(initialPixel)`: it is similar to the precedent version, but it also adds some pixel in the `secondList` every time there is an ambiguity in the choice of the next pixel.
- `getPixelFromSecondList()`: it takes a pixel from the `secondList`.