Semi-Automatic Image Segmentation: A Bimodal Thresholding Approach

Hanwei Shen
Christopher R. Johnson
E-mail: hwshen@cs.utah.edu and crj@cs.utah.edu

UUCS-94-019

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

January 3, 1995

Abstract

We have developed a semi-automatic image segmentation tool which combines conventional manual segmentation utilities with a novel automatic image segmentation algorithm. Manual segmentation is achieved by dropping control points and fitting cubic splines to these points. Automatic segmentation is achieved by bimodally thresholding local windows of the target image and contour following. By combining these two segmentation methods, a user can obtain accurate boundary descriptions with much less effort.

1 Introduction

Image segmentation, the process of defining boundary domains in 2D images, takes on a very important role in model building applications. Given slices of 2D images, the boundaries of interesting regions must be defined before surface reconstruction, mesh generation, and other modeling operations begins. Although image segmentation and contour/edge detections have been investigated for quite a long time[Rus92] [Sch89] [RK82], till now there is still no algorithm which can automatically find region boundaries perfectly. There are two reasons for this. One is that most of the image segmentation algorithms are still noise sensitive. The other is that most of the segmentation tasks require certain background knowledge about the region of interest.

To ensure the accuracy of the segmentation, many people still manually segment the image, i.e, drop control points by hand and invoke data fitting algorithms to fit curves. Owing to the complexity and the large number of images needed to be processed, this segmentation process has been recognized as one of the most time consuming parts of the whole modeling process.

We have developed a semi-automatic image segmentation tool which combines conventional manual segmentation utilities with a novel automatic image segmentation algorithm. Manual segmentation is achieved by dropping control points and fitting cubic splines to these points. Automatic segmentation is achieved by bimodally thresholding local windows of the target image to produce boundary pieces. The individual pieces are connected to yield the entire region boundary. By combining these two segmentation methods, a user can obtain accurate boundary descriptions with much less effort.

In the following sections, the detail of our semi-automatic segmentation algorithm is described. The software's user's menu and programer's menu are given as well.

2 Manual image segmentation

Currently, there is no perfect segmentation algorithm which is always accurate. Manual image segmentation is still an important feature in most image segmentation tools. This section gives a brief description of manual image segmentation.The usage details of the manual segmentation tool can be found in section 4

The manual segmentation procedure is straightforward. Given a 2D image slice, the user places numerous control points on a visible region boundary. A data fitting algorithm is then invoked to interpolate between these control points to generate a region contour. Several different data fitting methods can be used. The methods differ based on either the degree of interpolation function used, i.e. linear, quadratic or on the basis function used, i.e. natural spline, Bezier spline, et cetera. In our implementation, we used a cubic spline interpolation method. The basic principles and some mathematics background on interpolation can be found in many algorithm or computer aided geometry design books [RA76] [Far93].

3 Automatic image segmentation

We developed an automatic algorithm which allows users to interactively steer the segmentation process. The program starts by asking the user to pick an initial point on the image. The local edge detection and contour following programs are then started from the initial point to find the region boundaries. Only a small contour segment piece is given at a time. The program then waits for feedback from the user. The user has the option to correct the identified contour segment based on a visual inspection. If no corrections are needed, the program continues to find the next contour piece. By combining automatic boundary detection for local regions with the manual corrections, we achieve a correct and effective means of segmentation.

We use a bimodal thresholding algorithm to determine the boundary segments in the local region. When the user picks an initial point, a small local window is placed around this point. A local histogram is then computed according to the values of pixels located inside the window. If part of a region boundary passes through the local window, the histogram should have a bimodal appearance, i.e, two peaks in the distribution. The local minimum between these two peaks determines the threshold value used to segment the local image.

After a boundary segment is found, the local window then moves in the direction of the contour to the next position. Currently, we position the window such that the end point of the previous contour segment is at the center of the next local window. Although this creates some overlap between consecutive local images, the overlap reduces the chance of discontinuities between contour segments.

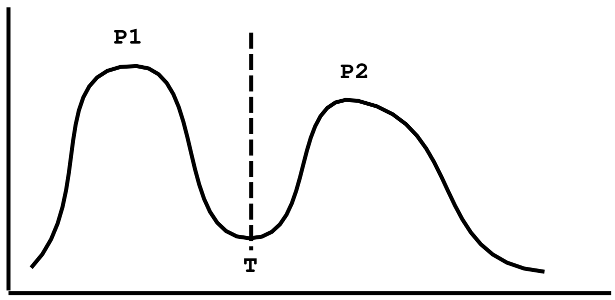

Figure 1: Basic histogram in a local window

3.1 Bimodal Threshold Finding

To determine the threshold, we need to analyze the histogram and to search for the *most significant* local minimum value. The pixel value in a histogram that separates the pixels of an image into two major groups determines the *most significant* value.

In Figure 1, we find that T is a local minimum in the histogram, although it is not the minimum of the whole histogram. By assigning the value of T as threshold, we can divide the histogram into two regions, P1 and P2. We can then assign all the pixels in P1 as one value, the other pixels in P2 as another value. The region boundary will be the pixels located between theses two peaks. However, attempting to analyze the histogram directly may be difficult, since the histogram usually consists of many oscillations. The oscillations can cause problems in finding the local minimum. See Figure 2 for an illustration of this difficulty.

If the image can be bimodaled, it should be possible to ignore the small oscillation on the histogram curve. In the algorithm, the raw histogram is cut into several sub-intervals. A new histogram is then formed by integrating every subarea. The integration is achieved by counting the number of pixels in each sub interval. The Figure 3 illustrates the histogram recalculation.

Most of the oscillations will be removed from new histogram. Therefore finding the local minimum finding is simplified. Figure 4 illustrates the appearance of new histogram.

Figure 2: Histogram with oscillations

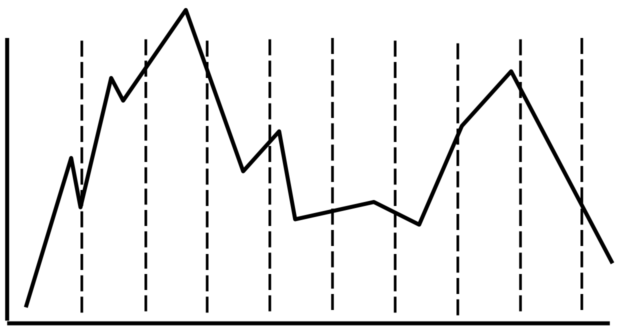

Figure 3: Histogram with subareas

If the area of a region is smaller than both sides of the neighbor's area, then we can say this region is one of the local minimum regions. In Figure 4, the shadowed region is a local minimum region.

More than one local minimum region may exist even when the image can be bimodaled. Therefore, we need a sophisticated way to find the most significant local minimum region. The algorithm begins by finding the height difference between the local minimum region and the adjacent local maximum regions. The local maximum region doesn't need to be located right next to the local minimum region. Therefore we need to search the entire histogram on both sides of the local minimum to determine the local maximum regions. In Figure 4, D2 and D3 will be the local maximum

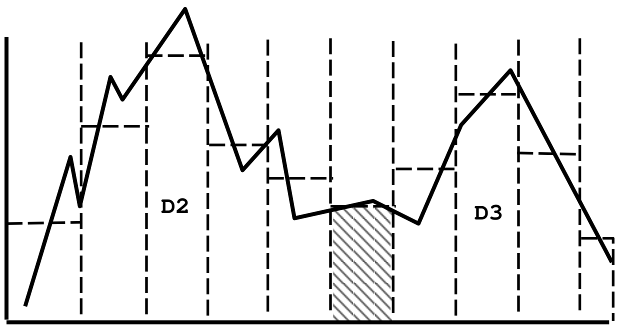

Figure 4:

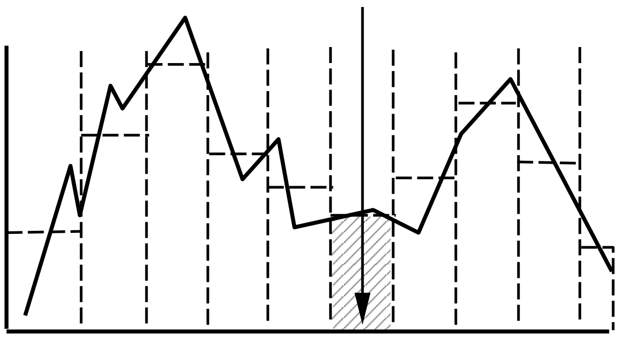

Figure 5:

regions relative to the shadowed local minimum region. The height difference will be the $|area(D2) - area(LocalMin)| + |area(D3) - area(LocalMin)|$. After determining all of the height differences for all the local minimum regions, we choose the region with the maximum height difference as the *most significant local minimum region*. The value located in the middle of the local minimum region will be the bimodal threshold. Figure 5 illustrate the threshold value.

The algorithm assumes that we can intelligently divide up the histogram. However, the histogram division is different from image to image. If the area for every region is too big, the real threshold could be missed. On the other hand, if the area of division is too small, there will be many oscillations on the curve which also increases the difficulty for finding a local minimum.

A iterative method is used to help determine the size of the local region. At the beginning, the histogram is subdivided roughly, e.g. 4 subregions. Then the number of subregion is then gradually increased at each iteration until the threshold reaches a converged state. The stopping criteria is $abs(old\_threshold - threshold)/threshold$ less then some ERROR. The ERROR criteria is adjustable.

An outline of the algorithm follows:

```
Scan the image to get its histogram;
setup the initial number to divide the histogram;
do {
increase number to divide the histogram;
divide the histogram into subregion;
find the local minimum subregions;
find the most significant local minimum subregion;
find the threshold;
} while ( error(threshold, last_threshold)>ERROR);
```

3.2 Contour Following

After the local window is positioned and the bimodal region boundary found, the local window is moved to the next position and the segmentation work is repeated. The next position is chosen such that the end point of the previous contour becomes the center of the new window. Although a circular shaped local window seems more appropriate, a square window is used for simplicity.

We use cubic splines to fit the region contour, therefore we don't keep every pixel of the contour segment. Instead, the end point of each contour segment is used as a control point for the cubic spline. One control point is generated for each local window movement.

The size of the local window can affect the accuracy of this algorithm. If the local window is too small, it is possible that no bimodal histogram exists. If the local window is too big, there might be more than two sub-regions. This will cause our bimodal threshold finding algorithm to fail. Currently the size of local window is predefined and based on empirical studies. However, adaptive window size finding is possible.

3.3 User Interaction

The semi-automatic segmentation algorithm starts by asking the user to pick a position on the image as a starting point. This starting point needs to be located somewhere on the region boundary in order to get accurate results in subsequent contour following actions. The user needs to specify the program the direction to follow for the contour following algorithm. Currently the user can specify going up(increasing y coordinate), going down(decreasing y coordinate), going right(increasing x coordinate), and going left(decreasing x coordinate). Note that most of the time two of the above choices are the equivalent.

After specifying the starting point and following direction, the user only needs to click the mouse botton to run the contour following algorithm step by step. The user needs to visually check the output of the new contour control point to ensure accurate boundary specifications. If a wrong result is generated or the user determines that the image is too noisy to do automatic boundary detection, the user can just stop the automatic selection and switch to manual segmentation.

The flexible switch between the automatic and manual features provides the user full control over the segmentation process. The automatic feature gives efficient and quick response, whereas the manual selection gives a more precise result.

4 User's Menu

The program was written in C and GLX in the mixed mode programming model of X and SGI's GL libraries which runs only on SGI's platforms. We are developing an OpenGL version which will run on different hardware with the X/Motif and OpenGL libraries.

```
man segment

                              USER COMMANDS
NAME
```

```
segment -h IMAGE\_HEIGHT -w IMAGE\_WIDTH [-p IMAGE\_PADDING]
       [-l PXL\_LEN] [-f HEADER]
```

```
Options

  -h  :   height of the input image
  -w  :   width of the input image
  -p  :   The length of padding(in Byte) at the end of each scan line
  -l  :   The length(in Byte) for each pixel
  -f  :   The length of image header (in Bytes).

Option Defaults

  -h  :   256
  -w  :   256
  -p  :   0
  -l  :   1
  -f  :   0
```

Description

Type the name **segment** with appropriate options to start the program. A GL image window and a X menu bar will appear. From the menu selection, the user can give the name of an image file. Two copies of the image will be displayed on the window. The left one is the work space which allows the user to drop control points or execute the semi-automatic segmentation module. The right one shows the resultant fitted curves.

In the manual segmentation mode, the user can place control points on boundaries, fit cubic spline curves to them, modify control points, and make the spline boundary curve final (commit it). Note that the program can only handle one boundary region at a time. Only after committing the current boundary spline, can the user proceed to the next region boundary.

In the semi-automatic segmentation mode, the user can pick a point on the image as a starting point. This point must be located on on the region boundary. After answering the question of which direction to follow the contour (up, down, left, or right), the user only needs to click on the right mouse button to increment the semi-automatic algorithm. The user should look at the image window while clicking the button to determine if the boundary is being defined properly. This could happen when the local image is too noisy or there is no visible boundary. If the boundary goes the wrong way, the user can just switch back to manual mode and place control points by hand. Whenever the user determines that the quality of image is sufficient to run the automatic algorithm, he can switch back to the semi-automatic mode. By combining the manual and automatic mode with user interaction, we achieve accurate semi-automatic image segmentation with minimal effort.

Finally, the user can output all the boundaries to files, one file per boundary. The user needs to specify how many points need to be sampled on the boundary spline curve. The file format is:
x1 y1 z1
x2 y2 z1
...
Note that we only handle 2D image slices, so the output of the z coordinates will always be zero.

In the menu bar, there are six pull down menus: *Input, Output, Segment, Spline, Image, and Quit.*

Input:

The input menu consists of three sub-menu which are RAW, TIFF, and Sequence. The RAW selection requires the user to input the name of a file which consists of unencoded binary image values. The TIFF selection requires the user to input the name of a file which has black/white TIFF image format. The Sequence selection allows the user to specify a sequence of images to be segmented. The user needs to provide the name of file containing a sequence of file names with the format:
img.1
img.2
img.3
....
The name img.x is just the file name of images in ASCII format. The images will be retrieved based on their order in the file. Three images will appear, the left image will be the previous image of the sequence, the middle image will be the working image, and the right image will be the next image in the sequence.

Output:

The Output selection has two sub-menu items, show contour or out to file. Show contour will erase the image on the right hand side of the main image window and only display the contours just defined. Out to file selection will prompt the user to input a file name and the number of points to sample along the boundary and then output the points to the named file.

Segment:

The segment menu contains two sub-menu items - Auto and Manu which represent automatic and manual image segmentation. The user can choose either selection at any time.

Spline:

The Spline selection consists of operations which perform the spline fitting, control point modification, and finalize the modified spline. These operations include: **Spline, Add, Move, Del, Cancel, Reset, and Commit**.

Spline:

This action performs the spline fitting. After generating the control points, this function is selected to fit a cubic spline to these points. As long as the commit selection is not made, the fitted spline is only temporary. The user can still use other selections to modify the control points.

Add:

The user may add new control points at any location and in any order to those previously defined. To add a new point, the user must select an old control points. The new control point will then be inserted after that selected old point.

Move:

This selection allows the user to move an existing control point. The user must mark one control point first and then mark a new position again to move that point.

Delete:

This selection allows the user to delete a control point from the control point sequence. The user just selects one of the control points.

Cancel:

This function removes the current spline curve, erasing all of the control points in the process.
Reset:

If user make this selection, the entire set of boundaries on the current image will be removed.
Commit:

This function finalizes the current boundary or spline. No changes can be made once this operation has been performed.

Image

In order to get the best result of segmentation, several image processing utilities are provided to enhance the image including: **Histogram, Gaussian, Median, Smoothing, De-Noise, Edge, Enhance, and Erase.**

Histogram:

This selection provide the histogram and simple thresholding for the entire image. After this item is selected, a new image window containing the image histogram appears. In the histogram window, the user can use the left mouse button to drag the lower bound (in blue color) and the upper bound (in red color). All the pixels with values outside the range will be filtered out. The blue line and red line are initially located at the two extreme ends of the histogram and the one which is closest to the mouse cursor will be selected. This function helps the user to erase some noise with a specific value. The image must be loaded in before selecting this function.
Gaussian:

A 3x3 Gaussian smoothing kernel is applied to the image.
Median:

A median filter is used to the image.
Smooth:

A 5x5 average filter is applied to the image.
De-Noise:

Removes all the isolated spike pixel values.
Edge:

A Sobel edge detection operator is applied to the whole image and outputs a edge image to a new window. This function currently doesn't help the segmentation process and is provided only for the convenience of certain users.
Enhance:

A enhance operator is applied to a local region of image where the mouse cursor is located.
Erase:

Completely erases a local region of the image where the mouse cursor is located.

Quit

Cancel or confirm the request.

Cursor

This is a label widget. By clicking the middle button of the mouse, the current position of mouse cursor will be tracked and displayed. This mouse tracker can be disabled by clicking the middle mouse button again. Note that the upper left image plane coordinate is (0,0).

5 Programmer's Menu

In this section, we will briefly describe the function of each module which might be useful for those who want to modify the source programs.
main.c :

This is a XGL main program which invokes routines in menu.c to create the X user interface.

menu.c :

Several X/Motif interface routines are included in this file. These routines allow fast creates of main windows, menu bars, pull-down windows, label widgets, and dialog boxes.
UI.c :

Contains most of the event handler routines or event handler auxiliary routines such as add, remove, move control points, tracking mouse cursor position, mouse button handling routine, file output event handing routines.
construct.c:

Includes the main routines to do the spline fitting.
spline.c:

This file contains routines to build up a cubic spline to fit data points. The algorithm is mainly from Segwick's algorithm book. **contour.c:**

Contains main routines which drive the automatic image segmentation features.
bimodal.c:

Contains routines which actual do the bimodal threshold finding.
mask.c:

Contains routines which take care of image processing operations on local image window.
histogram.c:

Contains routines which perform image processing operations for the whole image.
edge.c :

The main routine which actually calls the Sobel edge operator is included here.
image.c:

Contains two routines which set and get the value of a pixel.
sequence.c:

Contains routines which load a sequence of images.
readin.c:

Contains routines which read in the raw image format.
tiff.c :

Contains routines which read in the TIFF image file format.
gl_fun.c:

Contains routines which call the GL library that performs the image display, line drawing, and all other 2D graphics routines.

Acknowledgments

This work was supported in part by the Whitaker Foundation. The authors would like to John Schmit for his helpful comments and suggestions.

References

[Far93] Gerald Farin. *Curves and Surfaces for CAGD – A Practical Guide.* Academic Press, Inc., 1993.

[RA76] D.F. Rogers and J.A. Adams. *Mathematical Elements for Computer Graphics.* McGraw-Hill, Inc., 1976.

[RK82] A. Rosenfeld and A.C. Kak. *Digital Picture Processing.* Academic Press, Inc., 1982.

[Rus92] J.C. Russ. *The Image Processing Handbook.* CRC Press, Inc, 1992.

[Sch89] R.J. Schalkoff. *Digital Image Processing and Computer Vision.* John Wiley & Sons, Inc., 1989.