# Interactive Image Graph Cut Segmentation

*Release 0.00*

David Doria, Siqi Chen

October 28, 2012

Rensselaer Polytechnic Institute, Troy NY

**Abstract**

This document presents a system to "scribble" on an image to mark foreground and background pixels and then feed these pixels to a graph cuts segmentation technique. The interaction is done using the Visualization Toolkit and the processing is done using the Insight Toolkit. The graph cut function itself is the maxflow-v2.21 algorithm written by Yuri Boykov and Vladimir Kolmogorov. This version of their algorithm is released under a GPL license.

## Contents

## 1 Introduction

Automatic image segmentation algorithms are constantly improving. However, the best results can still be obtained by requiring human input and feedback throughout the segmentation process. This system allows the user to indicate foreground and background pixels using a "scribble" technique. The segmentation result can be iteratively refined and recomputed until the desired segmentation is obtained.

We have implemented the algorithm described in [1]. The basic workflow is:

- "Scribble" to select foreground and background pixels
- Set $\lambda$
- Perform the segmentation
- Iteratively refine

Our implementation can operate on images with arbitrary pixel types. That is, we can operate on grayscale images, RGB images, or N-D images, each with components of any scalar type.

## 2 Segmentation Algorithm

As described in [1], the energy function to be minimized is

$$E(A) = \lambda R(A) + B(A) \tag{1}$$

where $A$ is a linearized vector of pixel assignments (foreground or background), $R(A)$ is the regional term (see Section 2.1) and $B(A)$ is the boundary term (see Section 2.2).

### 2.1 Regional Term

$$R(A) = \sum_{p \in P} R_p(A_p) \tag{2}$$

As suggested in [1], we have used

$$R_p(object) = -lnPr(I_p|object) \tag{3}$$

$$R_p(background) = -lnPr(I_p|background) \tag{4}$$

which reflect how the intensity of pixel $p$ fits the histogram of the foreground and background models.

## 2.2 Boundary Term

The boundary term and is described by:

$$B(A) = \sum_{\{p,q\}\in N} B_{p,q}\delta_{A_p \neq A_q} \tag{5}$$

where

$$\delta_{A_p \neq A_q} = \begin{cases} 1 & \text{if } A_p = A_q \\ 0 & otherwise \end{cases}$$

The choice of $B_{p,q}$ is taken to be:

$$B_{p,q} \propto \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right)\frac{1}{dist(p,q)} \tag{6}$$

Since we have used a 4-connected neighborhood, $dist(p,q)$ is always equal to 1.

## 2.3 General Behavior

When $\lambda$ is high, there is no regularization. That it, each pixel decides if it belongs to the foreground or background based only on its color (without looking at its neighbors). Another way to say this is that the segmentation problem degenerates into a clustering problem. If $\lambda$ is set to 0, only the boundary term is considered. This scenario produces a trivial solution, so we produce an error if the user specifies this condition.

## 3 Implementation Details

### 3.1 Structure

Our code has three main components:

- The GUI (GraphCutSegmentationWidget.h, GraphCutSegmentationWidget.cpp, GraphCutSegmentationWidget.ui)
- The “scribbling” code (in a submodule called ScribbleInteractorStyle)
- The graph cut segmentation code (in a submodule called ImageGraphCutSegmentation)

### 3.2 GUI

We used Qt for our GUI. The window is split into two main components. The left side is the input image, and the right side is the output (segmented) image.

#### Input Image

On the left of the form, the input image is displayed. Below the input image are radio buttons to chose whether the scribbling will select foreground or background pixels. A "Clear Selections" button is provided to allow the user to start over at any point.

#### Segmented (Output) Image

**Lambda** On the right of the form, a window is provided to view the result of the segmentation. Below this window, the segmentation parameters can be set. Lambda ($\lambda$) controls the relative proportion of the Regional Term in the energy function. Setting $\lambda = 0$ makes the energy function exactly equal to the Boundary Term only. We provide a combination of a slider and a text box to set $\lambda$. The text box allows the user to specify the maximum $\lambda$. The the slider sets the percentage of the maximum $\lambda$ to actually use as $\lambda$ in the graph construction.

**Number Of Histogram Bins** A slider allows the user to set the $NumberOfHistogramBins$ to use when constructing the foreground and background histograms.

## 3.3 Scribbling

To perform the scribbling, we use a vtkImageTracerWidget under the hood. Foreground scribbles are displayed in green while background scribbles are displayed in red. The vtkInteractorStyleScribble contains the vtkImageTracerWidget and provides the GraphCutSegmentationWidget with the scribbles that the user has input. We convert the list of points in the vtkPolyData returned by the vtkImageTracerWidget into a

```
std::vector<itk::Index<2> >
```

for both the Sources and Sinks (members of the ImageGraphCutBase). Since the vtkImageTracerWidget operates in a real coordinate space (non-discrete, non-pixel space), and does not guarantee a particular spacing of the points on the curve, we compute Bresenham lines between all adjacent points in the (ordered) list of points from the tracer to produce the list of pixels that lie under the stroke.

## 3.4 ImageGraphCut

The function of this class is to create the graph which will be fed to Kolmogorov's S-T min-cut implementation. The structure of the graph is a uniform (regular) grid of 4-connected nodes (the pixels). The edge weights between these nodes are called "n-weighs". Additionally, each pixel is connected to both a synthetic foreground node and a synthetic background node. Each of these edges is weighted with a "t-weight".

## 3.5 ImageGraphCutBase and ImageGraphCut

We want the segmentation to work for images of arbitrary pixel type. To allow for this, we want to template ImageGraphCut on the type of image to be segmented. However, we want to allow the user to specify this type at runtime. Therefore, we must store (in the Form class) a pointer to a base class of ImageGraphCut so that we can instantiate the ImageGraphCut template with the appropriate type when necessary.

#### ImageGraphCutBase

This class contains all of the functions which are not type-dependent. Its main purpose is to allow us to do the following in the Form class:

```
ImageGraphCutBase* GraphCut;

// ... later ...

this->GraphCut = new ImageGraphCut<TImageType>(reader->GetOutput());
```

#### ImageGraphCut

This class contains all of the type-dependent functions. It is here that we create the graph (CreateGraph()) and cut the graph (CutGraph()). In ImageGraphCut we also compute the histograms which are used in the Region Term computation (as shown in Section 2.1).

## 3.6 Creating the Nodes

Kolmogorov's S-T implementation requires each node ID to be stored as a `void*`. To keep track of this, we create an image of type:

```
typedef itk::Image<void*, 2> NodeImageType;
```

of the same size as the input image. We add all of nodes to the Graph object while storing their IDs in a NodeImage. This implicitly maintains the mapping from pixel index to node ID.

```
// Add all of the nodes to the graph and store their IDs in a "node image"
itk::ImageRegionIterator<NodeImageType> nodeImageIterator(this->NodeImage, this->NodeImage->Ge
nodeImageIterator.GoToBegin();

while(!nodeImageIterator.IsAtEnd())
  {
  nodeImageIterator.Set(this->Graph->add_node());
  ++nodeImageIterator;
  }
```

#### N-Weights

The N-weight are computed using the procedure described in Section 2.2.

```
float pixelDifference = PixelDifference(centerPixel, neighborPixel);
float weight = exp(-pow(pixelDifference,2)/(2.0*sigma*sigma));

void* node1 = this->NodeImage->GetPixel(iterator.GetIndex(center));
void* node2 = this->NodeImage->GetPixel(iterator.GetIndex(neighbors[i]));
this->Graph->add_edge(node1, node2, weight, weight);
```

#### Histograms

We use itkSampleToHistogramFilter to create a histogram of the selected foreground pixels (ForegoundHistogram) and background pixels (BackgroundHistogram). The dimensionality of the histogram is determined by the dimensionality of the input image pixel type. A parameter, *NumberOfHistogramBins*, controls the granularity of the histograms. If the user has selected to open the image as a color image (from the File menu), a 3D histogram (R, G, B) is computed and used. If the user has selected to open the image as a grayscale image, a 1D histogram (gray value) is used.

#### T-Weights

The synthetic foreground and background nodes are automatically created by Kolmogorov's implementation. We simply must set the edge weight from every N-node we have created to both the synthetic foreground (source) and synthetic background (sink) nodes. The weights as computed as described in Section 2.1.

```
this->Graph->add_tweights(nodeIterator.Get(),
                          -this->Lambda*log(sinkHistogramValue),
                          -this->Lambda*log(sourceHistogramValue));
```

(Note: $log()$ is the natural log in c++).

The nodes which were directly specified by the user via scribbling are set to definitely belong to the corresponding synthetic node by setting the weight for node it belongs to to `std::numeric_limits<float>::max()` and the node it does not belong to to 0. This translates to "you cannot cut this edge" and "there is no cost for cutting this edge", respectively. We must use `max()` rather than `infinity()` because if $\lambda$ is zero, the weight should be zero, and `0 * std::numeric_limits<float>::infinity()` is NaN.

## 4 Screenshots

### 4.1 Basic Environment

Figure 1 shows the GUI as described in Section 3.2.

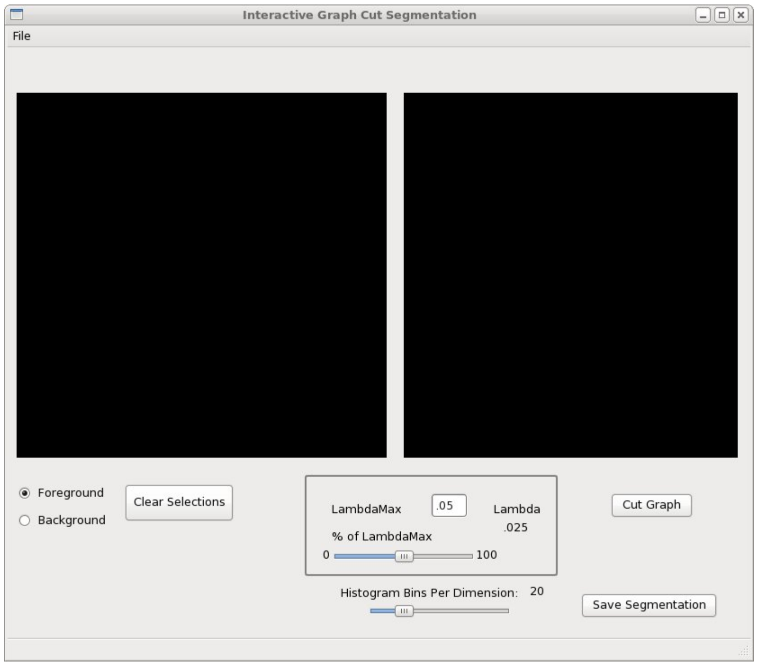
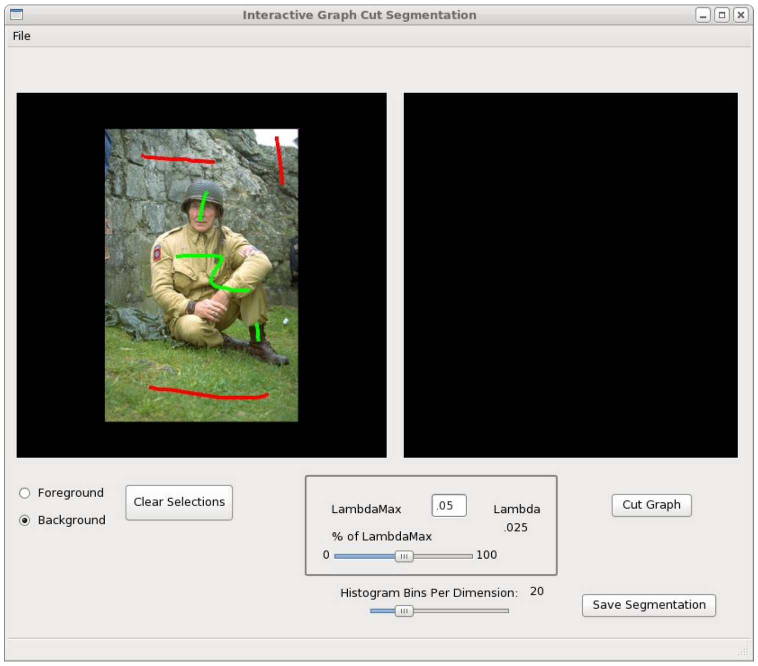


Figure 1: Default Environment

## 4.2 Scribbling

Figure 2 shows the GUI after the user has scribbled to select both foreground (green) and background (red) pixels.



LambdaMax .05 Lambda .025

% of LambdaMax 0 100

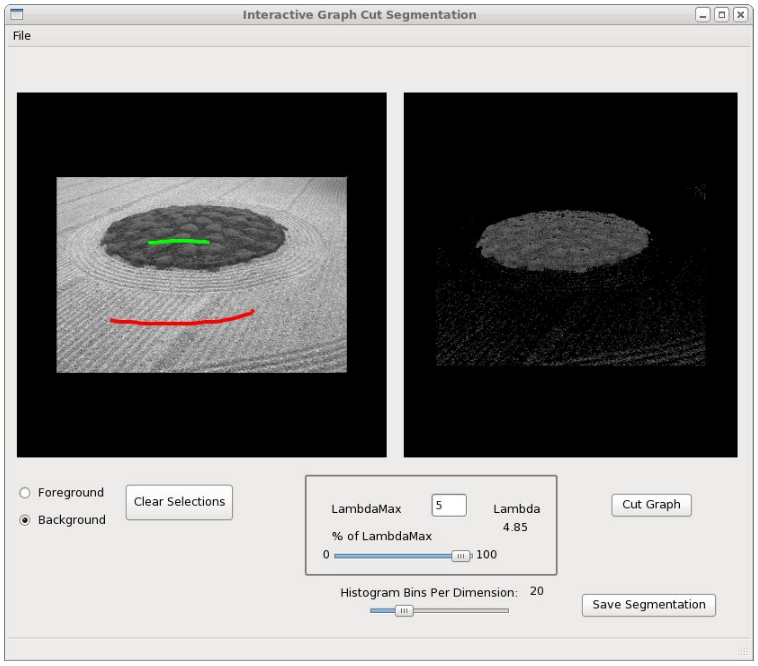
Histogram Bins Per Dimension: 20

Figure 2: Scribbling

## 4.3 Simple Example Image

A simple example image from the Berkeley Segmentation Dataset (http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/) is used for demonstrative purposes.

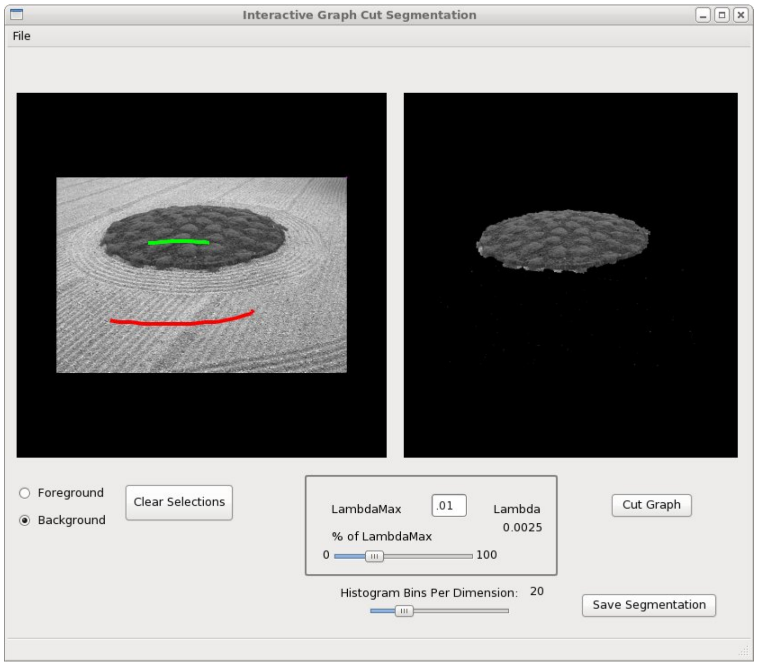
Figure 3 shows the resulting segmentation when λ is too high. Notice that many non-contiguous pixels are incorrectly considered to be in the foreground on the basis of their color only (the lines of shadows are closer to the dark colored sand than the light colored sand).



Foreground
[✓ Background]
LambdaMax 5
Lambda 4.85
% of LambdaMax 0 100
Histogram Bins Per Dimension: 20

Figure 3: Result of grayscale segmentation of Sand image with λ set too high.

Figure 4 shows the resulting segmentation when λ is set properly. This choice of λ puts enough weight on the Region Term to encourage a very connected segmentation.

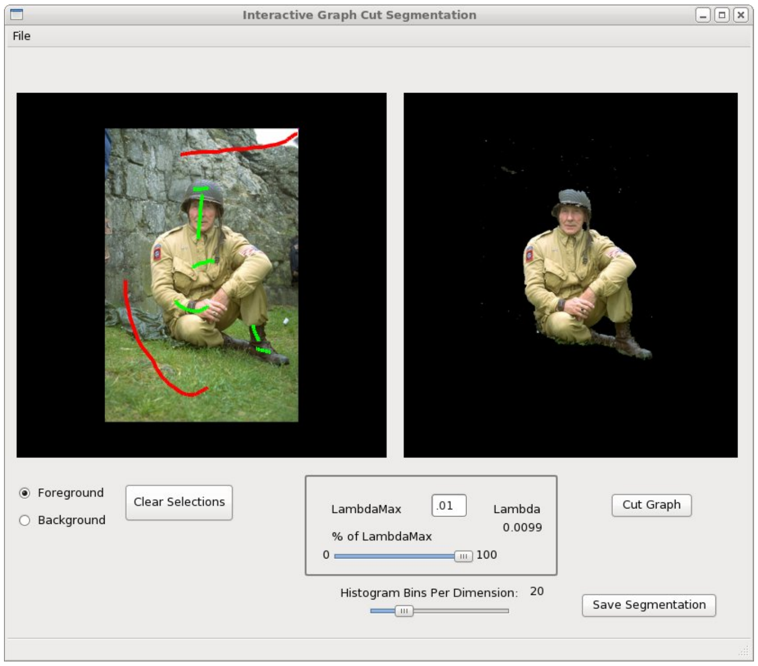


Foreground
[✓ Background]
LambdaMax .01
Lambda 0.0025
% of LambdaMax 0 100
Histogram Bins Per Dimension: 20

Figure 4: Result of grayscale segmentation of Sand image with λ set properly.

## 4.4 Difficult Image

To show that this does not just work in trivial cases, we show the result of the segmentation of a difficult image in Figure 5.



LambdaMax .01
Lambda 0.0099
% of LambdaMax
Histogram Bins Per Dimension: 20

Figure 5: Result of color segmentation of Soldier image.

## References

[1] Yuri Boykov, *Graph Cuts and Efficient N-D Image Segmentation*. International Journal of Computer Vision 2006 1, 2, 2.1