CS 312: **Intelligent Scissors**

Overview:

In this project, you will implement Dijkstra's algorithm for finding the best path between selection points along the boundary of an image.

Objectives:

- To apply Dijkstra's algorithm to a real-world problem.
- To solve a non-trivial image processing problem.
- To understand the importance of choosing data structures wisely in order to achieve good performance.
- To compare a very greedy "Simple Scissors" algorithm with the Dijkstra's-based "Intelligent Scissors" algorithm.

Background:

As originally defined by Mortensen and Barrett [1], "Intelligent Scissors" is an interactive image segmentation technique that uses Dijkstra's algorithm to find minimal cost segmentation (or boundary) paths in an image. An interactive implementation of the intelligent scissors algorithm would work as follows:

1. The user clicks on a "selection point" (also referred to as "segmentation point") somewhere along the desired image boundary.
2. Dijkstra's algorithm is used to find a minimal cost path from the selection point to every other pixel in the image. The paths are cached so that ...
3. As the user moves the cursor to a location further along the boundary, the minimum cost path back to the selection point is displayed.
4. The user clicks on the image boundary again to place another selection point, and the minimum cost path back to the selection point is added to the segmentation boundary.
5. Steps 2 through 4 are repeated using the new selection point until the path reaches the original selection point, making a closed boundary.

Intelligent Scissors works well in practice and is now the basis of a number of segmentation algorithms in commercial image editing software. For example, the right image below was segmented with just three selection points. Contrast this with the image on the left, produced by the Simple Scissors algorithm that just follows the least cost edge at each pixel. The Simple Scissors algorithm adheres to the edge in this case, but passes right by the point it is supposed to meet up with (near the eye socket of the skeleton) and then gets stuck on the cheek bone of the skeleton.

For this assignment, you will be programming both a "Simple Scissors" algorithm and "Intelligent Scissors" using Dijkstra's algorithm. (Both are greedy algorithms, as we define that notion in CS 312. The difference lies in the degree of greediness, as we will explain below.)

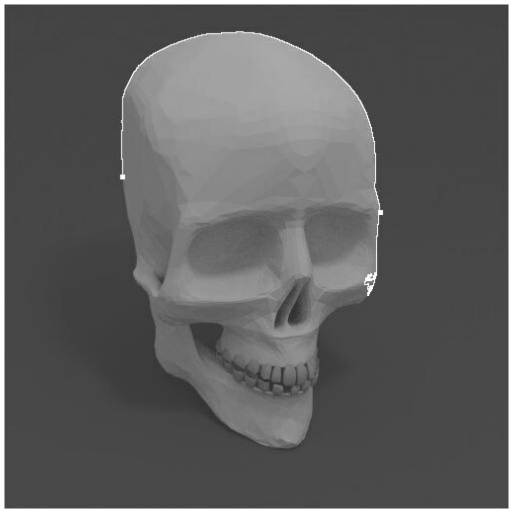
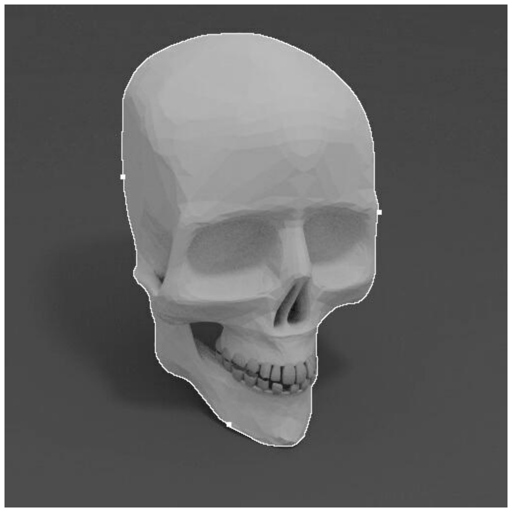


Figure 1 Simple Scissors



Intelligent Scissors
using Dijkstra's Algorithm

Images as Graphs:

An image can be treated as a graph by considering each pixel in the image as a vertex in the graph. Edges in the graph connect adjacent pixels. For this assignment, we will assume that pixels are only adjacent to their vertical and horizontal neighbors. We define the weight of an edge from one pixel to another as follows:

$$Weight = 1 + MG - G$$

where:
G = the gradient magnitude for the destination pixel
MG = the maximum gradient magnitude for any pixel in the image

To compute the gradient magnitude of pixels in the image, we use an operator called the "Sobel Filter" that calculates a discrete approximation of the x and y partial derivatives of the image. Using the Sobel filter, the gradient magnitude G_{xy} for the pixel P_{xy}, is defined as follows:

$$G_{xy} = \sqrt{GX_{xy}^2 + GY_{xy}^2}$$

where:

$$GX_{xy} = ((P_{x+1,y+1} - P_{x-1,y+1}) + 2(P_{x+1,y} - P_{x-1,y}) + (P_{x+1,y-1} - P_{x-1,y-1}))/4$$

and

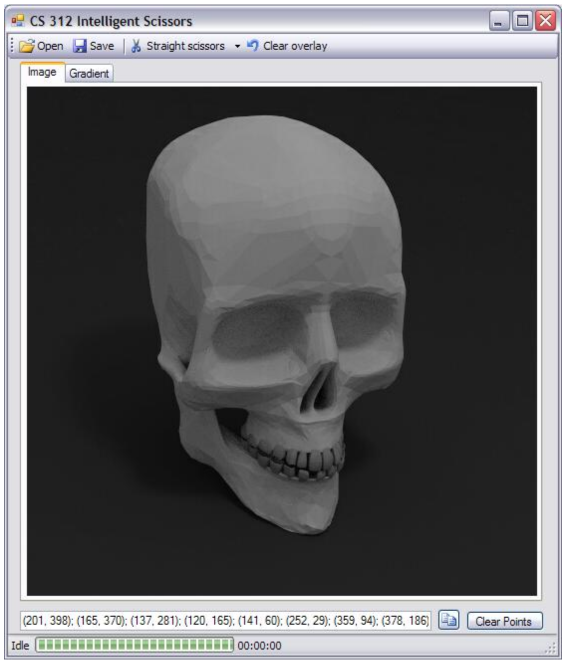
$$GY_{xy} = ((P_{x+1,y+1} - P_{x+1,y-1}) + 2(P_{x,y+1} - P_{x,y-1}) + (P_{x-1,y+1} - P_{x-1,y-1}))/4$$

Working with Images:

We provide a VisualStudio 2005 project that contains a class for reading and writing ASCII pgm format images (we have also included some pgm test images). PGM files allow comments. You will be given images that includes the selection points in a comment in the second line of the image file. The segmentation points will be given in the order in which they were selected by the user in a semi-colon separated list of tuples. The tuples consist of parentheses around an ordered pair of integers separated by commas For example, your image file might look something like this:

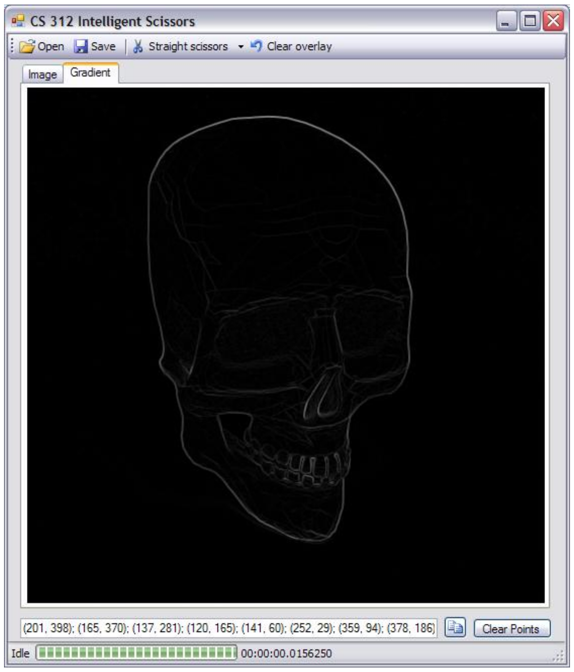
```
P2
# (30,40); (50,60); (10,20); (50,80)
# Created by Ifranview
384 256
255
...
```

The project distribution is already quite functional. As shown in the figure below, it will load and display images. It will also display the image gradient, computed automatically at image load time, in a separate tab.



```
(201, 398); (165, 370); (137, 281); (120, 165); (141, 60); (252, 29); (359, 94); (378, 186)
Idle 00:00:00
```

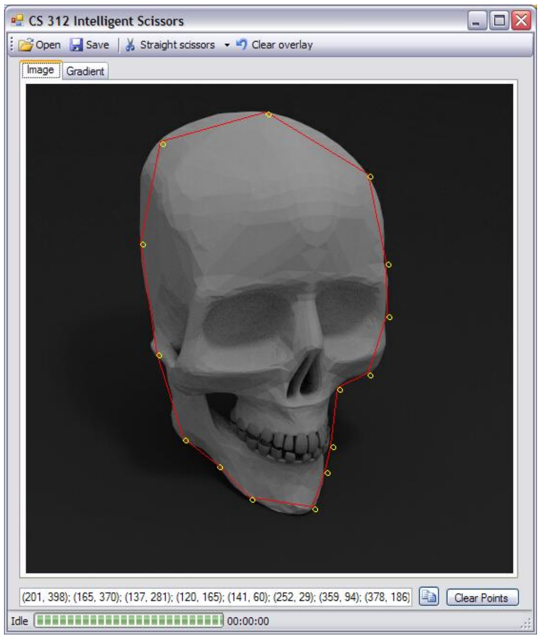
Image Tab



```
(201, 398); (165, 370); (137, 281); (120, 165); (141, 60); (252, 29); (359, 94); (378, 186)
Idle 00:00:00.0156250
```

Gradient Tab

The project distribution includes everything except implementations of the Dijkstra-based Intelligent Scissors and the Simple Scissors scissors algorithms. You should implement your scissors in the DijkstraScissors and SimpleScissors classes. The FindSegmentation() method in each class is the method that is called when the user selects your scissors implementation on the toolbar. The project contains several other files that you will not need to modify. Note that we have provided a very simple scissors example in the file StraightScissors.cs as an example. The following screenshot shows this algorithm in action:



(201, 398); (165, 370); (137, 281); (120, 165); (141, 60); (252, 29); (359, 94); (378, 186)

<u>“Simple Scissors”</u>

The Simple Scissors algorithm does not use Dijkstra’s algorithm. Instead, at each vertex (pixel) along the path, this algorithm simply chooses the edge to the neighboring pixel in a very greedy fashion until the goal is reached, or the path cannot continue (i.e., the path is about to enter a loop). The next pixel is chosen by selecting the immediate neighbor with the smallest weight such that it does not lead to a cycle in the accumulated path..

<u>“Intelligent Scissors” Using Dijkstra’s Algorithm</u>

The Intelligent Scissors algorithm uses Dijkstra’s algorithm to find the optimal (least-cost) paths between selection points in a given image. You will not be expected to implement the interactive version of Intelligent Scissors sketched on page 1. Rather, given a set of selection points, your algorithm will find the segmentation path in “batch” mode.

Pseudocode for Dijkstra's algorithm can be found on page 110 of the textbook, and further explanations can be found in the class lecture slides.

The definition of the cost of moving between adjacent pixels is given above in the “Images as Graphs” section.

Implementation Notes

One key to success in this project is to think carefully about how you might restrict Dijkstra's algorithm to an appropriate sub-image so that it is tractable. A 512x512 pixel image is a large graph; you'll want to be sure that you only work with a subset of those at any given time. You may make assumptions that will cause your algorithm to be incorrect in certain corner cases.

Keep in mind that the simple scissors can become stuck in a cycle. That is, the simple scissors may choose the same pixels over and other unless you do something to detect and handle this situation. For example, the simple scissors in Figure 1 on page 2 have become stuck in a cycle on the cheekbone. When you detect a cycle, you may continue searching at the next segmentation point, quit searching altogether or write your simple scissors to avoid pixels and a cycle but continue searching for the next segmentation point.

There are a few programming idioms that will help you function in the project. Here are a few of them:

- Image.Bitmap.SetPixel() is the method you'll want to use to set the color of a pixel.
- After you set a pixel, you need to tell the image to update. You can use Program.MainForm.RefreshImage() to do that. You can change as many pixels as you want between updates and they'll all get updated with a single RefreshImage().
- DijkstraScissors.cs and SimpleScissors.cs are the files that contain the classes you want to implement. The method FindSegmentation() in each file is the entry point for the class that will be invoked when the corresponding button is clicked. The parameters are explained in the files.
- Feel free to use any built-in (.NET) data-structure you like, and feel free to download a priority queue class if you don't want to write your own.

To Do:

1. Code up both the Simple Scissors and Intelligent Scissors image segmentation algorithms as described above. Be sure to read the implementation notes, it will save you time. Each algorithm computes the shortest path between adjacent pairs of selection points -- this is called the "segmentation path". Remember that the segmentation path is a cycle that ends at the starting selection point.

2. Use the DrawEllipse() method to place a small (preferably) yellow circle on each selection point in the overlay. (See the StraightScissors class for example.)

3. Set the color of each pixel on the segmentation path to white or red or some other very visible (and printable) color in the overlay. (Again, see the StraightScissors class for an example.)

4. Compare and contrast the Simple Scissors and the Intelligent Scissors algorithms by solving three different segmentation problems.
    a. At least two must come from the "with-segmentation-points" subdirectory of the "TestImages" directory in the project distribution.
    b. The other one can come from anywhere you like, including the "with-segmentation-points" subdirectory.
    c. For all three problems you select, your algorithm must complete the segmentation using Dijkstra's algorithm in 7 seconds or less. Note that the code uses the StopWatch class to measure how long it takes to perform steps 1-3 and places that time in the status bar alongside the progress bar at the bottom of the form (see the figures above).

Report:

1. Include screenshots of the images, including the running times, produced by each algorithm on all three problems.
    a. You can capture the image of the window (using the ctrl-shift-alt-PrtSc facility for capturing an image of the window in focus). Looking at the resulting segmentation gives one a good idea of how good the segmentation is. We want to be able to see what you end up with and compare the results.
    b. Thus, you should have at least six window shots.

2. Write a few sentences about the trade offs between running time and precision that you observe in your experiments.

3. Write a few sentences to address the following question: This algorithm could be used in tools like Adobe Photoshop or the GIMP as described in the Background section. How would you modify your algorithm to make it fast enough to run interactively?

4. Say whether or not you met the 7-second performance requirement for Dijkstra's.

5. Include a copy of your class that implements Dijkstra's algorithm. (We will verify that you used Dijkstra's algorithm. If you are keen to create your own specialized shortest path algorithm, then do that for the improvement.)

Improvement:

Think of a way to improve your Intelligent Scissors algorithm and implement your improvement. Determine if your improvement had the desired effect. Write a page or two (one page will certainly suffice) that

1. describes the purpose of your improvement
2. describes your improvement implementation
3. provides some empirical evidence to support a conclusion that your improvement either did or did not do what you thought it would do
4. explain the evidence in #3

Here are a few suggestions for improvements:

- Make the intelligent scissors interactive. That is, allow the user to click on selection points one at a time and incrementally find the segmentation path between those points.
- Support color images. You'll need to pick a new image format (and probably want to use Microsoft's built-in tools for loading images) and need to come up with a gradient equation that accounts for color.
- Improve the running time of your Intelligent Scissors algorithm by improving the implementation (would a different priority queue help?). This might include using an algorithm other than Dijkstra's algorithm.
- Add a button to zoom in on the image so you can get a better idea of whether or not your algorithm is following the gradient like you think it does.
- Come up with an automatic way to evaluate the quality of a segmentation path.
- Render the segmented view of the image as a height-field so that it looks like a 3-dimensional terrain. Then, draw your segmented path on the terrain and see if the algorithm is following the gradient like you think it does. DirectX9 would probably be useful for this.
- Reduce the amount of memory that your Intelligent Scissors algorithm consumes.
- Make up your own improvement based on something that interests you and is related to the project. Talk to your professor if you aren't sure if your idea qualifies.

Reference:

[1] E. N. Mortensen and W. A. Barrett, "Intelligent Scissors for Image Composition," in *Computer Graphics (SIGGRAPH `95)*, pp. 191-198, Los Angeles, CA, Aug. 1995.

Revised: 2/2/2007