Intelligent Scissors

# TA: Mohamed Raafat

## T093

|  |  |
| --- | --- |
| منة الله إبراهيم محمد عفيفي  أحمد محمد توفيق محمد  أيمن مصطفى حسن حسين | 20191700665  20191700062  20191700162 |

Abstract

Selection tools can be used to select objects in an image to resize/delete/copy/move the objects. There are many types of selection tools such as rectangles or free-form selection tool, sometimes free-form selection tools are called Lasso’s. You can imagine a lasso as a rope surrounding your selection. Unfortunately, selection using ordinary lasso’s can be tedious and boring. In Photoshop, there is a more advanced version of ordinary lasso’s called Magnetic Lasso Tool. Magnetic Lasso is a tool that automatically snaps to an Objects’ Boundaries.

The technical term for the Magnetic Lasso Tool is Livewire or Intelligent Scissors. In this project we want to implement a simple magnetic lasso to learn more about image processing, graphs, and greedy algorithms.



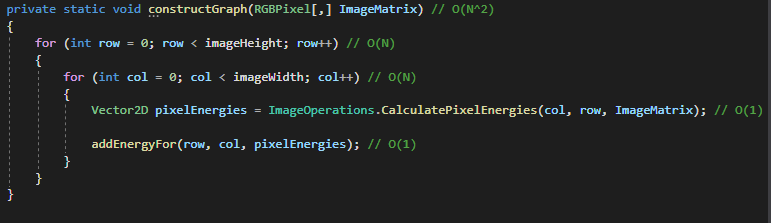
Contents

* [Graph Construction](#_Graph_Construction)
* [Calculating Shortest Path](#_Calculating_Shortest_Path)
* [Backtracking Shortest Path](#_Backtracking_the_Shortest)
* [Frequency](#_Frequency)
* [Scaling small Images](#_Scaling_small_Images)
* [Cropping Final Image](#_Cropping_Final_Image)

# Graph Construction

The graph is constructed by calculated the gradient between pixel intensities so a high gradient indicates a high change in color which can be interpreted to be an edge of an object, since the calculation is only a mathematical equation so the only overhead is introduced from going through all the pixels of the image to calculate the gradients (energies).

## private static void constructGraph(RGBPixel[,] ImageMatrix)



• Loops on pixels to add the weights of neighboring pixels

• Main Complexity:

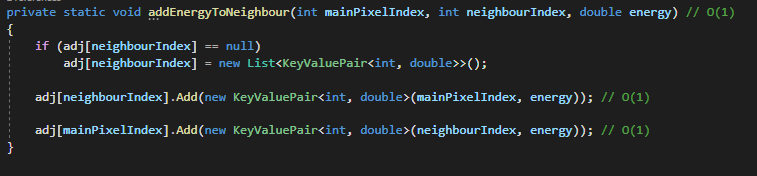
• Two nested loops looping on the pixels of the Image -> ->

• Helpers:

• addEnergyFor(int row, int col, Vector2D pixelEnergies) ->

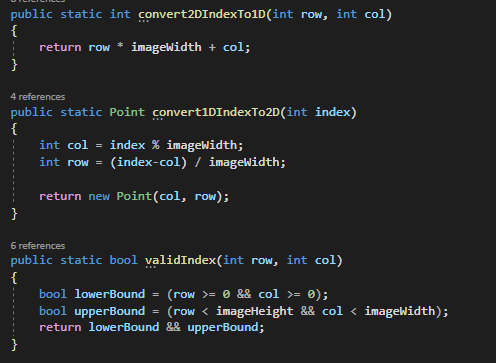


• addEnergyToNeighbour(..) ->



Note: List<>.Add() is considered **O(1)** since the initial capacity of the List after adding the first element is **4** and the maximum edges per vertex is **4** so the List<> won’t need the O(N) expansion

• Misc. Functions ->



• Final Complexity:

# Calculating Shortest Path

If we need to generate a livewire between two pixels P1(i,j) and P2(x,y), it is the same as getting the shortest path between the two corresponding vertices V1(i,j) and V2(x,y), because the low edge-weights are at the image-edges on which we want our livewire to snap on.

A picture containing text

Description automatically generatedTo get the shortest path between two vertices in an undirected weighted graph we use Dijkstra Algorithm.

For each Node, we keep track of the minimum distance found to it in the “dist” Array and its previous Node in “parent” Array, The Distance array is initialized with a big value to overestimate the cost to each node, and the parent Array is set to -1 to denote that the node’s previous has not been set.

For images with small to moderate sizes this takes no time, but for images with big sizes initializing the arrays in O(V) takes a lot of time which causes a delay in drawing of the livewire.

The solution was to make Dijkstra work only on a given boundary box that is set to be around the source node, Only the nodes in the given bounds have their distance and parent values initialized (Lines 22 to 33), this greatly reduced the running time on big images.

The boundary box size is set to be the size of the panel (the visible part of the image in picture box).

Analysis of Dijkstra

How fast is Dijkstra? It uses a Priority Queue to keep track of the smallest Cost, we use a Fibonacci Heap to implement the priority queue needed; which improves the asymptotic running time.

Fibonacci Heaps have a better amortized running time than many other priority queue Implementations including binary heaps.

The main Fibonacci Heap Operations used are:

* Enqueue (Insert in Fibonacci heap) -> which takes constant time O(1)
* Dequeue (Extract Min) -> which works in O(log n) amortized time

Dequeue is used in the outer while loop (Line 47) which iterates for every vertex in the graph,

Enqueue is used in the inner for loop (Line 61) which visits all edges for every vertex.

Therefore, Dijkstra has a complexity of O( V’ log V’ + E’) using Fibonacci Heaps,

Where V’ and E’ are number of vertices and edges checked until reaching the Destination.

# Backtracking the Shortest path

After using Dijkstra to get the shortest path from the last anchor to a free point, we need to Construct the path by backtracking from the destination node to the source node, We use the “parent” array used in Dijkstra to get to each node’s parent till we find a node whose parent has not been set, this is the source node.

Graphical user interface, text, application

Description automatically generated

Analysis

The while loop starts with the destination and ends when we have found the source,

Therefore, it has a complexity of O(N) where N is the length of the shortest path.

# Frequency

Text

Description automatically generatedYou can move the mouse freely and the program will place anchors along the path automatically based on a given frequency.

If frequency is enabled an anchor will be placed along the path when the length of the livewire exceeds the frequency given. Graphical user interface, text, application

Description automatically generated

The last free point is added to the linked list of anchors along with its path, this takes O(1).

# Scaling small Images



* Sets and variables
  + Checks if zoom mode made picture vertical scaled or horizontal scaled
  + Divides size of the opposite scaled side to get factor of zoom
  + Multiply scaled size with scaled factor to get scale size
  + Gets the non-filled part in the opposite side of the container

Complexity:



* Get Point on image
  + Divide scaled side on zoom factor
  + Get other side but with the non-filled area removed

Complexity:

# Cropping Final Image



* Add Points to single array to create polygon
* Connect first and last point of the points array to get a polygon
* Get Bounds of the polygon to get new image width and height
* Only draw inside of the polygon
* Crop the polygon rectangle out of the image
* Display it

Complexity: