SEAM CARVING DOCUMENTATION

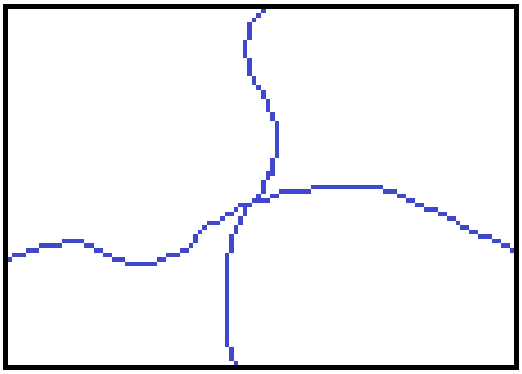
ORAZALIN, Alibek
RUBAB, Tamzid Morshed
SAMIN, Thanic Nur

Introduction

Seam Carving is a technique for content-aware image resizing. Traditional resizing techniques such as cropping or uniform resizing do not preserve the aspect ratio of the important parts of the image. Seam carving is able to recognize important parts of an image and preserve those while resizing the image.

Methodology

Seam carving does image resizing via deleting and/or inserting seams. A seam is a continuous path of pixels from top to bottom or left to right of an image.



Seams

In order to do so, we assign each pixel an 'energy' value depending on how 'important' it is. The energy value of a pixel is calculated by the formula:

$$\sqrt{\sum_{\text{RGB}} (\text{left\_pixel.color} - \text{right\_pixel.color})^2 + (\text{up\_pixel.color} - \text{down\_pixel.color})^2}$$

If the pixels are at the boundary, they themselves are assigned to be the directional pixels.
Thus, pixels where the colors change drastically have higher energy. The energy of a seam is the sum of the energy of the pixels of the seam. Therefore, seams with lower energy will tend to avoid boundaries of important objects in the image, and thus avoid the objects altogether.
If we want to specifically avoid or remove cells, we can forcibly change their energy. For example, if we want to remove a certain pixel, we will assign a very low energy to it so that seams are more likely to go through that pixel. Similarly, if we want to keep a pixel, we assign a very high energy to it so that seams do not go through that pixel. One such case is faces. We do not want to edit the faces, so we use **facial recognition** to assign very high energy to faces. Similarly, when enlarging an image, we don't want to add the same seam over and over again. So, when we add a seam, we assign high energy to it, so that it doesn't get chosen again.
Seams with low energy can be be found using **Dijkstra's algorithm**, since this is essentially a shortest path problem.

Variables and Functions

SeamCarver.h

class SeamCarver is the class where the image is edited. It consists of the functions that are called to modify the image as required using the seam carving method.
`Mat original` is the image passed to the program.

`double** energy` is the 2d array consisting of the energy of each pixel.

`long long rows, cols` are the number of rows and columns of pixels in the image.

`int type` is the image type.

`int num_channels` is the number of channels in the image.

`long long** index` is what is modified each time. Modifying the picture each time is resource intensive, which is why we just modify the index. After 'removing' the desired number of seams, we look at the index array and actually modify the picture according to that.

`double large` is the large number assigned as an energy. This is a very large number, so that even if all other pixels in a row/column had the maximum possible energy, their sum wouldn't exceed this value. This lets us mark a pixel as 'never remove' or 'always remove'. This is chosen so that the sum of other pixels do not influence the fact that this pixel has to be removed/must be kept.

`void copy_Vec_to_vec(const pixel_type v1, vector<pixel_eachChannel_type>& v2, int num_elements)` Copies a vector. Used to make copy of neighbor pixels.

`static double distance(vector<T> a, vector<T> b)` Computes the euclidean distance of two vectors. It is used in finding the energy of a pixel.

`void assign_energy_to_pixel(long long row, long long col)` Computes and assigns energy of a pixel to the energy array

`void assign_infty_cell(long long row, long long col, bool plus)` Assigns positive or negative large number as an energy to the corresponding pixel depending on `plus`.

`SeamCarver(string filename, ImreadModes x)` is the constructor. It sets up everything needed for the modifications. given the path to the image and the mode it should be read at.

`SeamCarver(const Mat& mat)` is also the constructor, but it directly takes the image in `Mat` format.

`long long* vertical_path()` returns an array of indices such that in the i'th row, the pixel at the i'th place in the array is a part of the seam with minimal energy. This is found using **dijkstra's algorithm**.

`long long* horizontal_path()` returns an array of indices such that in the i'th column, the pixel at the i'th place in the array is a part of the seam with minimal energy. This is found using **Dijkstra's algorithm**.

`void remove_vertical_path(long long* to_be_removed)` removes a vertical path. The given array is derived from the function `vertical_path`.

`void remove_vertical_paths(long long num_cols)` removes the given number of vertical seams with minimal energy.

`void remove_horizontal_path(long long* to_be_removed)` removes a horizontal path. The given array is derived from the function `horizontal_path`.

`void remove_horizontal_paths(long long num_rows)` removes the given number of horizontal seams with minimal energy.

`void add_vertical_path(long long* to_be_added)` adds a vertical path by duplicating the given path. The given array is derived from the function `vertical_path`. So this is the path with minimal energy. To avoid duplicating the same path twice, the energy of the pixel of these paths are assigned to be very large.

`void add_vertical_paths(long long num_cols)` adds a vertical path `num_cols` times.

Note that, `add_horizontal_path` and `add_horizontal_paths` are not implemented. These affects are achieved by rotating the image, using the equivalent functions for the vertical paths and rotating the image back.

`void rescale(double width_ratio, double height_ratio)` rescales the image from the given ratio by using the functions `add_vertical_paths`,`remove_vertical_paths`,`remove_horizontal_paths` appropriately.

`void remove_object(bool** location)` the boolean array of locations denote the boundary of the objects that have to be removed. We assign very low energy to the pixels inside of this boundary, so that the seams that go through this part are prioritized. Then we apply the `remove_vertical_paths` or `remove_horizontal_paths` appropriately to perform the removal.

`void detectFaces(Mat& img)` uses facial recognition to recognize faces in an image and assign high energy to the pixels consisting of the faces. This is done so that the program does not distort faces in given images.

`Mat get_pic() const` returns the original image used.

BinHeap.h

class MinHeap represents a min binary heap that will be used as the priority queue required by Dijkstra's algorithm in detecting vertical and horizontal seam.
`struct node` structure of each node in the heap. Each node in this heap will represent a pixel. It contains `long long row; long long col; double key; int parent; bool n_is_extracted; long long index;`. row, col are just their position in the pixel matrix. key is the sum of energies from the beginning of the path (that contains that pixel) to that pixel (by Dijkstra's algorithm). parent is used to indicate its previous pixel in that path. `n_is_extracted` denotes whether it is already removed from the heap by the function `extract_min`. index denotes its position
`struct node` structure of each node in the heap. Each node in this heap will represent a pixel. It contains `long long row; long long col; double key; int parent; bool n_is_extracted; long long index;`.

row, col are just their position in the pixel matrix. **key** is the sum of energies from the beginning of the path (that contains that pixel) to that pixel (by Dijkstra's algorithm). parent is used to indicate its previous pixel in that path. **n_is_extracted** denotes whether it is already removed from the heap by the function `extract_min`. **index** denotes its position in 'heap' array.

`node** keep` is an array of `node*`.

`keep[index]` points to the node that represents the pixel at position index. So each element in keep array will always point to the same pixel regardless of its position in the heap.

`node* heap` contains the nodes in the heap.

`long long* heap_to_keep` gives us the index of a node in keep, given the index of it in heap.

`long long size` denotes the total size of the heap. It is fixed from declaration of an object of MinHeap.

`long long num_element` is the current number of nodes in heap array.

`long long num_element_keep` is the current number of nodes in keep array.

`void swap_nodes(long long index1, long long index2)` swaps the nodes `heap[index1]` and `heap[index2]`.

`void adjust_down()` is called after we extract the minimum node and replace that with the last node (according to extract minimum algorithm for binary heap). It makes sure that the top node has the minimum key.

`void adjust_up(long long index)` is called from insert and decrease_key. It makes sure that each node has keys greater than or equal to the key of its parent node.

`void heap_insert(long long row, long long col, double key)` inserts a node with the given information in the heap array. It's a helper private function used in the actual insert function.

`void insert(long long row, long long col, double key)` is a public function. The setback of this function is that it can only be used 'size' number of times in the beginning before we use other operations - decrease_key, extract_min. And we need to insert the pixels in the following order: we traverse each row from left to right starting from the top one and insert the pixels in that order. This is actually enough for our algorithm.

`bool decrease_key(long long index, double key);` decreases the key of the node pointed by `keep[index]`, if the new key is smaller. If it can decrease, it returns true and otherwise, it returns false.

`void extract_min(long long& row, long long& col, double& key)` extracts out the node with minimum key.

`void set_parent(long long index, int parent)` sets the parent of a node. parent is either -1, 0, or 1 because every node has 3 neighbors (the 3 below in case of vertical and right in case of horizontal).

`bool is_extracted(long long index) const` checks if `keep[index]` has been extracted.

`bool is_empty() const` checks if the heap is empty.

`int get_parent(long long index) const` returns the parent of `keep[index]`.

`double get_key(long long index) const` returns the key of `keep[index]`.

mainwindow.h

class MainWindow Main window class that inherits from QMainWindow class that basically shows main user interface window. It consists of a toolbar with several actions (that will be discussed later) and graphics view that should display the image to be modified.

`void delete_picture()` deletes seam carver.

`void delete_marker()` clears graphics view from object removal markers.

`void create_marker()` preprocessing for object removal.

`void cancel_drawing()` undo object removal mode and switch to the normal by showing corresponding actions in the toolbar.

`void on_actionImport_image_triggered()` import an image.

`void on_actionClear_image_triggered()` clear the image.

`void on_actionResize_triggered()` resize the image : displays another dialog and connects it to the main window.

`void on_actionExport_image_triggered()` exports the displayed image.

`void on_actionObject_removal_triggered()` switched to the object removal mode by turning the marker on and hiding some action in the toolbar.

`void on_actionRemove_triggered()` removes the chosen object.

`void on_actionCancel_triggered()` cancels drawing.

`void on_actionProject_triggered()` displays short information about the app.

`void on_actionAuthors_triggered()` displays short information about authors.

`void rescale_image()` displays rescaled image and removes the original image.

`void receive_and_draw(QPoint p1, QPoint p2, QPen pen)` signals to draw to choose part of the image to remove.

super_label.h

class super_label is defined to be a child class of QLabel used to convert value of the horizontal slider (in range [0, 99]) to the scale range [0.5, 1.5] and display it. To do that, signal from the slider was given to a slot setVal that converts and shows scales correspondingly.

drawingview.h

class drawingview is a child class of QGraphicsView (widget that shows images) used to override mouse events to draw on image.
`void mousePressEvent(QMouseEvent* event)` Changes the last clicked point.

`void mouseMoveEvent(QMouseEvent* event)` Handles the event when the mouse is moved.

`void mouseReleaseEvent(QMouseEvent* event)` Handles the event when the mouse is released.

`void drawingview::drawLineTo(const QPoint& endPoint)` Draws the line by emitting the signal to graphicsView and updates the last pressed point correspondingly.

dialog.h

class dialog a user interface dialog that opens when you press the resize button. Consists of 2 sliders reading horizontal and vertical scales, 2 labels of class super_label (to display values of the input), and a button to resize the image.

`Dialog(QWidget *parent = nullptr)` Creates the dialog itself and does necessary connections of slots and signals. Changes initial value on the slider.
`double get_width()` Return width scale.

`double get_height()` Return height scale.

`void on_pushButton_clicked()` Closes the dialog and emits the signal to main window to rescale when the resize button is clicked.

`void setWidthScale(int w);` Slot used to convert the default value on the width slider appropriately.

`void setHeightScale(int h)` Slot used to convert the default value on the height slider appropriately.

Possible Actions

Import image: Imports an image to be modified.
Clear image: Clears the image selected by 'Import image'.
Export image: Export the modified image.
Resize: Resizes the image by adding/removing seams as necessary. Capped at $0.5x$ to $1.5x$ in both width and height.
Marker: Lets you mark the boundary of the part of the image you want to remove.
Remove: [Only appears after 'Marker' is selected] starts the removal of the marked part of the image.
Cancel: [Only appears after 'Marker' is selected] de-selects the marker.
Project: Opens a dialogue box stating what the project is about: 'This project implements seam carving algorithm for content-aware resizing'
Authors: Opens a dialogue box stating the authors of the project.

Dependencies

This project is developed using OpenCV(version 4.5.0) and Qt(version 5.15.1 msvc 2019). We used Microsoft Visual Studio 2019 with the extension Qt VS Tools(version 2.6.0) to build the program.