****

SOFTWARE ENGINEERING DEPARTMENT

COMPUTER VISION FINAL PROJECT REPORT

**CONTENT AWARE IMAGE RESIZING**

(SEAM CARVING)

|  |  |
| --- | --- |
| **NAMES** | **STUDENT ID** |
| BASIROU NDOW | 2120236018 |
| BAFOKENG ERNEST RAMALITSE | 2120236023 |
| MASS JAMMEH | 2120236017 |
| ALHAJI UMARU GBOW | 2120236042 |

**INTRODUCTION**

Digital images are often viewed in many different display devices (such as mobile phones, laptops, tablets, etc.) with a variety of resolutions. Variation of resolution makes viewing images difficult because they usually are resized to accommodate limited space.

Simple attempts at resizing include scaling and cropping. However, these approaches have limitations as they alter the content of the image which may lead to loosing important in the image. Scaling reduces perceivable detail and cropping cannot be done automatically. Also, cropping alters the image composition and is not always desirable.

The main motivation or goal of our project is to overcome the drawbacks of the approaches named above, by implementing a content-aware resizing algorithm known as **Seam Carving**. It is a technique that intelligently resized images by removing or adding seams “connected paths of pixels”. It is a useful for preserving important content in an image while adjusting its dimension. A vertical seam is a path of pixels connected from the top to the bottom with one pixel in each row and a horizontal seam is a path of pixels connected from the left to the right with one pixel in each column.

**ALGORITHM USED**

Our Seam carving algorithm is accomplished by a 3-step process:

1. Calculating the energy of each pixel. This is done by smoothing the image and then computing the first x- and y-derivative at each point using a 3x3 Sobel filter.

***Energy = |Ix| + |Iy|***

2. Using dynamic programming to find a seam from top to bottom with the least total energy. A seam is defined as a continuous series of pixels. Thus, if you have a vertical seam, it consists of one pixel per row, with pixels in adjacent rows in either the same or an adjacent column.

3. Removing the pixels along the lowest-energy seam to create a smaller image. This will remove a less-interesting part of the image, while still preserving continuity in the image. These steps are repeated until the image has shrunk to the desired size.

The processes named above are implemented as follows:

1. **ENERGY MAP**

* The first step of the algorithm is to calculate an energy value of every pixel, which is a measure of its importance—the higher the energy, the less likely that the pixel will be included as part of a seam. We used the function below to derive the energy map:

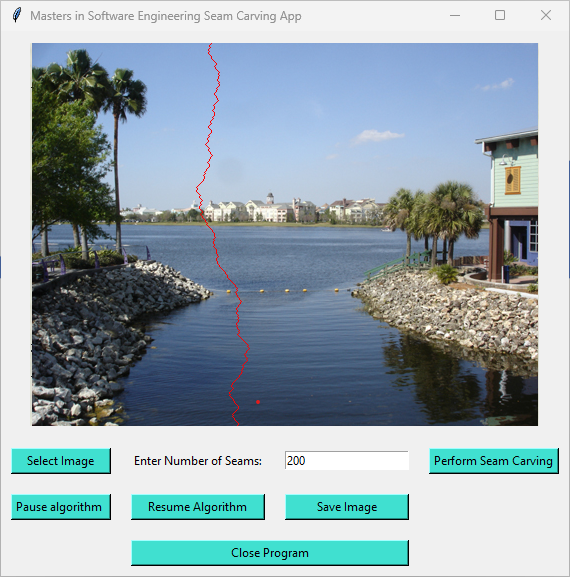
**ei(I) = | ∂ /∂x I|+| ∂/ ∂y I|**

From the function, we perform the following:

* Find the partial derivative in the x-axis
* Find the partial derivative in the y-axis
* Sum their absolute values
* NOTE: the absolute value will be the energy value for that pixel.
* Ater computing the energy value, the Sobel filter will be applied to compute the gradient (derivative) of the image representing the energy at each pixel. The gradient of an image represents the rate of change of intensity at each pixel. The Sobel filter is a convolutional kernel that is run over the image on every channel. Here is the filter in two different directions of the image:
  + First, we will consider a grayscale image represented as a 2D matrixwhere represents the intensity of the pixel at coordinates **.** The Sobel Operator calculates the gradient in both the horizontal and vertical directions as shown below:

1. **SEAM WITH LEAST ENERGY (CUMMULATIVE ENERGY)**

After computing the energy map, we use the concept of dynamic programing to find a path from the top of the image to the bottom of the image with the least energy. This line must be 8-connented: this means that every pixel in the line must be touched the next pixel in the line either via edge or corner. For the example illustrated below, it is the red line:



Finding the seam with the least energy as captured on the image above was derived from the function below:

**M(i , j) = e(i , j ) + min(M(i-1, j-1), M(i-1 , j), M(i-1 , j+1))**

We create a two-dimensional (2D) array call M to store the minimum energy value seen upto that pixel. This basically mean that M[i ,j] will contain the smallest energy at that point in the image, considering all the possible seams upto that point from the top of the image. So, the minimum energy required to traverse from the top of the image to bottom will be present in the last row of M.

1. **DELETING PIXEL FROM THE SEAM WITH LEAST ENERGY AND REPEATING FOR EVERY COLOUM**

This is the final stage of our Algorithm, as all the work has been implemented. Here, the seam with the lowest energy will be removed. This will be repeated for all the columns, thereby resizing the image and returning a new image without loosing important content of the original image.

**TECHNOLOGIES USED TO IMPLEMENT THE ALGORITHM**

* **PYTHON:** is a high-level, interpreted, and general-purpose programming language. This is going to be our primary language for implementing the seam carving algorithm.
* OpenCV: (Open-Source Computer Vision Library) is an open-source computer vison and machine learning software library which provides tool for image and video analysis including functions of image processing. We will use it image loading, processing and displaying the images
* **NumPy:** is a powerful numerical computing library for Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on them. We will use it to utilized for numerical operations and array manipulations.
* **Tkinter:** is the standard GUI (Graphical User Interface) toolkit for Python. It provides a set of tools to create desktop applications with a graphical interface. We will use it to develop the Seam Carving Application user interface.

**DEMO IMAGES GENERATED FROM THE ALGORITHM**

The Implemented seam carving algorithm successfully resized images preserving important features The GUI provides an initiative way for users to interact with the algorithm and observe the resizing in real time as shown below:

1. This allows the user to select an image, input the number of seams, and perform the seam carving algorithm.

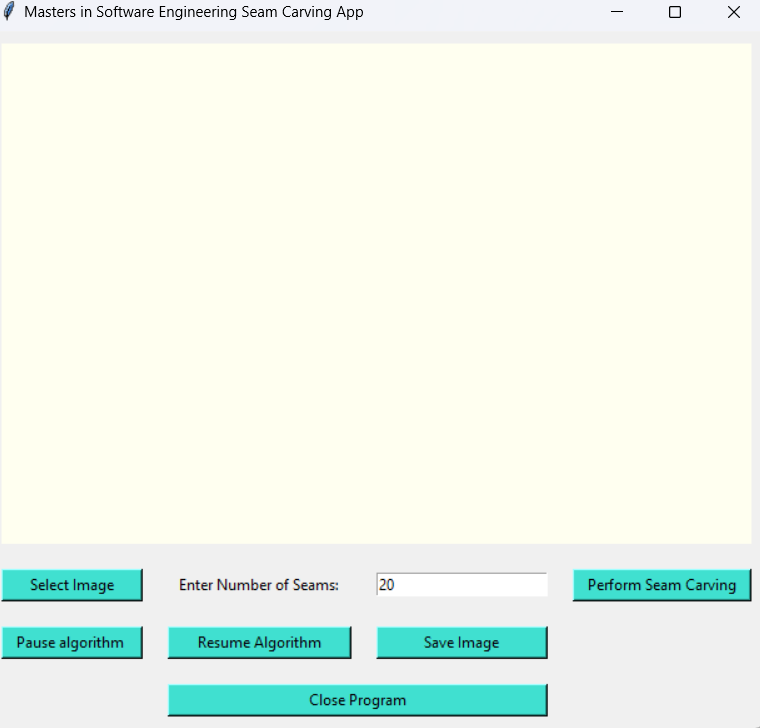
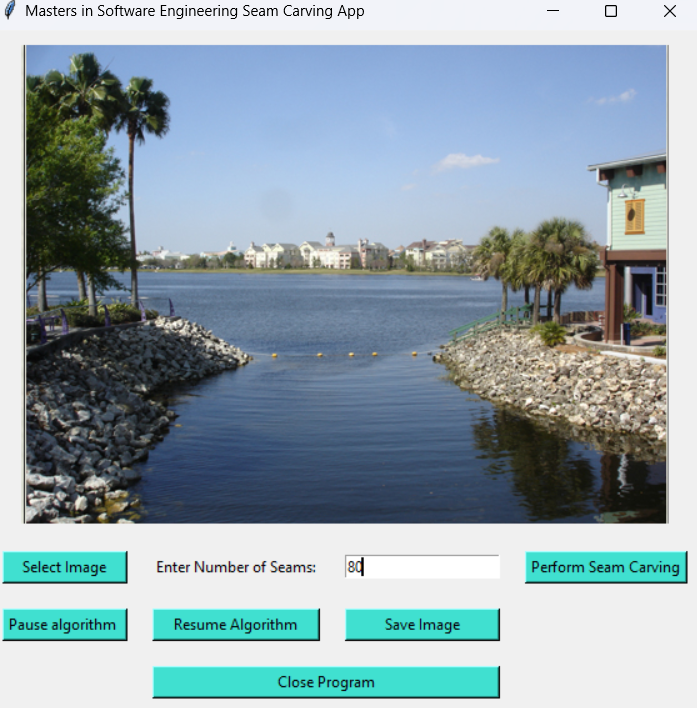
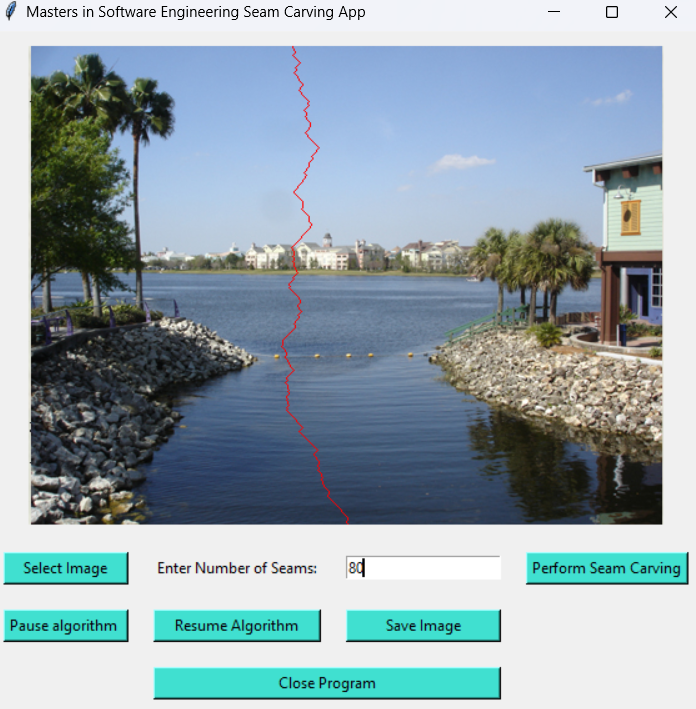
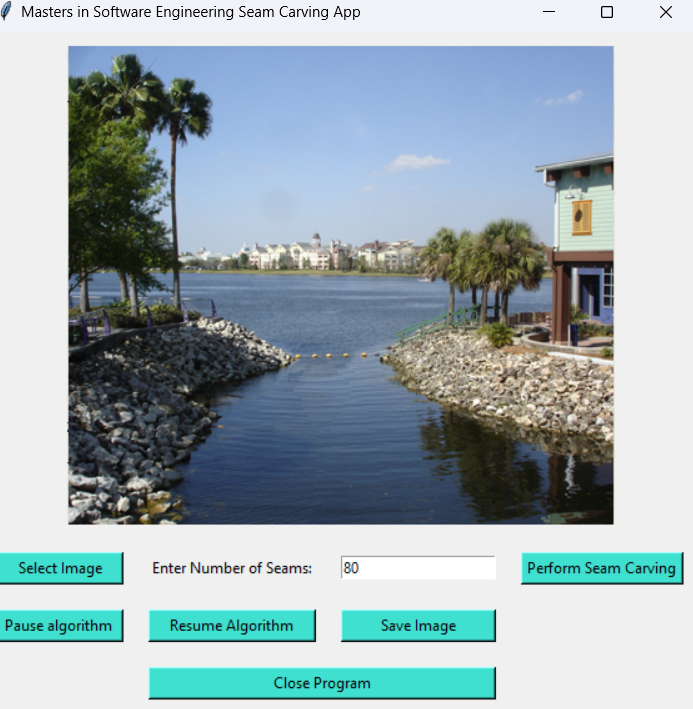


Figure User Interface before the user can upload the image

1. The image choosed by the user will be displayed on the inteface and subsequently, the Algortihm will be perfromed on the image and a new smaller image will finally be genated as shown below:



**CHALLENGES AND LIMITATIONS**

* **Compatibility Issues:** some compatibility issues arose, particularly with the ‘pathlib’ package during the freezing process. This required explicit exclusion during the creation of the standard alone executable file.
* **Real time Visualization:** while the algorithm visualizes the seam carving process in real-time there might be performance issue with the large images, Further optimizations could enhance the user experience.
* **The GUI:** the user interface is not fixed, when the image is larger than the value we set, it expands the grid and some of the features disappear like buttons.

**FUTURE IMPPROVEMENTS**

* Resizing of video using the idea of seam carving
* Currently, our system only allows images to be shrunk. Possible future work would include image expansion. Upon finding a seam of minimal interest we would interpolate to determine the intensity between the seams neighbors and then add an adjacent seam.
* Another interesting feature that could exploit this function is object removal. This would require some user input to define the area or object to be removed. Then the algorithm would simply do the seam carving while making sure to select pixels inside the area to be removed.
* To improve the quality of the shrunken carved image we could change the ordering of seam removal. Instead of simply alternating between removing vertical and horizontal seams would could implement a second dynamic programming algorithm to determine the exact ordering of seams to remove that would minimize the energy removed.

**CONCLUSIONS**

In our project, we were able to achieve the goal and that is to solve the drawbacks of scaling and cropping by implementing the idea of content-aware image resizing using seam carving. A three-step algorithm was implemented in our project and we were able to meet most of our design specifications.

Our algorithm only perform image resizing by shrinking the image without deleting or removing important features of the image. It would have been interesting to include an algorithm that expand the image by performing the same task.

Although, the technique we used was able to perform all the steps of the algorithm by displaying their corresponding results on the user’s interface in real time.

Code Reference

# Import necessary libraries

import numpy as np

import cv2

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import tkinter.messagebox

# Define a class for the Seam Carving App

class Seam:

def \_\_init\_\_(self, root):

# Initialize the Tkinter root window

self.root = root

self.root.title("Masters in Software Engineering Seam Carving App")

# Initialize variables

self.input\_image = None

self.output\_image = None

self.num\_seams\_var = tk.StringVar()

self.num\_seams\_var.set("20")

self.paused = False

# Create widgets for the UI

self.image\_frame = tk.Frame(root, width=600, height=400, bg='ivory')

self.image\_label = tk.Label(self.image\_frame)

self.load\_button = tk.Button(root, text="Select Image", command=self.load\_image, bg='turquoise')

self.num\_seams\_label = tk.Label(root, text="Enter Number of Seams:")

self.num\_seams\_entry = tk.Entry(root, textvariable=self.num\_seams\_var)

self.run\_button = tk.Button(root, text="Perform Seam Carving", command=self.run\_seam\_carving, bg='turquoise')

self.pause\_button = tk.Button(root, text="Pause algorithm ", command=self.pause\_seam\_carving, bg='turquoise')

self.resume\_button = tk.Button(root, text="Resume Algorithm", command=self.resume\_seam\_carving,bg='turquoise')

self.save\_button = tk.Button(root, text="Save Image", command=self.save\_image, bg='turquoise')

self.exit\_button = tk.Button(root, text="Close Program", command=root.destroy, bg='turquoise')

# Grid layout for widgets

self.image\_frame.grid(row=0, column=0, columnspan=4, padx=10, pady=10)

self.load\_button.grid(row=1, column=0, padx=10, pady=10, sticky="ew")

self.num\_seams\_label.grid(row=1, column=1, padx=10, pady=10)

self.num\_seams\_entry.grid(row=1, column=2, padx=10, pady=10, sticky="ew")

self.run\_button.grid(row=1, column=3, padx=10, pady=10, sticky="ew")

self.pause\_button.grid(row=2, column=0, padx=10, pady=10, sticky="ew")

self.resume\_button.grid(row=2, column=1, padx=10, pady=10, sticky="ew")

self.save\_button.grid(row=2, column=2, padx=10, pady=10, sticky="ew")

self.exit\_button.grid(row=3, column=1, columnspan=2, padx=10, pady=10, sticky="ew")

# Set resizing behavior for columns and rows

for i in range(4):

root.grid\_columnconfigure(i, weight=1)

root.grid\_rowconfigure(1, weight=1)

# Method to load an image from the file system

def load\_image(self):

# Open a file dialog to select an image file

file\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.png;\*.jpg;\*.jpeg;\*.bmp;\*.gif")])

if file\_path:

# Read the selected image using OpenCV

self.input\_image = cv2.imread(file\_path)

# Display the loaded image

self.display\_the\_image(self.input\_image)

# Method to display an image in the UI

def display\_the\_image(self, image):

if image is not None:

#where we Convert the image from BGR to RGB

#BGR "BLACK, GREEN, RED"

#RGB "RED, GREE, BLUE"

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

#where we Convert the image to a PIL Image

image = Image.fromarray(image)

# Create a PhotoImage from the PIL Image

photo = ImageTk.PhotoImage(image=image)

# Configure the image label to display the PhotoImage

self.image\_label.config(image=photo)

self.image\_label.image = photo

# Pack the image label to fill the available space

self.image\_label.pack(fill="both", expand=True)

# Method to perform seam carving based on user input

def run\_seam\_carving(self):

if self.input\_image is not None:

try:

# Get the number of seams from the entry widget

num\_seams = int(self.num\_seams\_var.get())

# Reset paused state

self.paused = False

# Perform seam carving on the input image

self.output\_image = self.seam\_carving(self.input\_image, num\_seams)

# Display the output image

self.display\_the\_image(self.output\_image)

except ValueError:

tkinter.messagebox.showinfo("Error"," Please enter a valid integer for the number of seams.")

# Method to save the output image

def save\_image(self):

if self.output\_image is not None:

# Open a file dialog to select the save location and file name

file\_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "\*.png")])

if file\_path:

# Save the original color image using PIL

output\_pil\_image = Image.fromarray(cv2.cvtColor(self.output\_image, cv2.COLOR\_BGR2RGB))

output\_pil\_image.save(file\_path)

# root window title and dimension

tkinter.messagebox.showinfo("Seamed Image saved Successfully", {file\_path})

# Method to calculate energy map of an image

def energy\_map(self, image):

# where we Convert the image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# where we Calculate the gradient using Sobel operators

dx = cv2.Sobel(gray, cv2.CV\_64F, 1, 0, ksize=3)

dy = cv2.Sobel(gray, cv2.CV\_64F, 0, 1, ksize=3)

# where we Calculate the gradient magnitude

gradient\_magnitude = np.sqrt(dx\*\*2 + dy\*\*2)

return gradient\_magnitude

# Method to calculate cumulative energy map

def cumulative\_energy\_map(self, energy\_map):

height, width = energy\_map.shape

#where we Initialize an array for the cumulative energy map

cumulative\_map = np.zeros\_like(energy\_map)

#where we Calculate the cumulative energy map

cumulative\_map[0, :] = energy\_map[0, :]

for i in range(1, height):

for j in range(width):

if j == 0:

cumulative\_map[i, j] = energy\_map[i, j] + min(cumulative\_map[i - 1, j], cumulative\_map[i - 1, j + 1])

elif j == width - 1:

cumulative\_map[i, j] = energy\_map[i, j] + min(cumulative\_map[i - 1, j - 1], cumulative\_map[i - 1, j])

else:

cumulative\_map[i, j] = energy\_map[i, j] + min(cumulative\_map[i - 1, j - 1], cumulative\_map[i - 1, j], cumulative\_map[i - 1, j + 1])

return cumulative\_map

# Method to find the vertical seam with the minimum cumulative energy

def find\_vertical\_seam(self, cumulative\_map):

height, width = cumulative\_map.shape

#where we Initialize an array for the vertical seam

seam = np.zeros(height, dtype=np.int64)

#where we Find the minimum index in the bottom row of the cumulative map

min\_index = np.argmin(cumulative\_map[height - 1, :])

seam[height - 1] = min\_index

# where we Trace the vertical seam from bottom to top

for i in range(height - 2, -1, -1):

if min\_index == 0:

min\_index = np.argmin(cumulative\_map[i, min\_index:min\_index + 2])

elif min\_index == width - 1:

min\_index = np.argmin(cumulative\_map[i, min\_index - 1:min\_index + 1]) + min\_index - 1

else:

min\_index = np.argmin(cumulative\_map[i, min\_index - 1:min\_index + 2]) + min\_index - 1

seam[i] = min\_index

return seam

# Method to remove a vertical seam from the image

def remove\_vertical\_seam(self, image, seam):

height, width, \_ = image.shape

# Create a new image with one less column

new\_image = np.zeros((height, width - 1, 3), dtype=np.uint8)

# Remove the pixels corresponding to the seam

for i in range(height):

new\_image[i, :, :] = np.delete(image[i, :, :], seam[i], axis=0)

return new\_image

# Method to perform seam carving on the image

def seam\_carving(self, image, num\_seams):

for \_ in range(num\_seams):

# Check if paused

while self.paused:

self.root.update()

# Calculate the energy map of the current image

energy\_map\_image = self.energy\_map(image)

# Calculate the cumulative energy map

cumulative\_map = self.cumulative\_energy\_map(energy\_map\_image)

# Find the vertical seam with the minimum cumulative energy

seam = self.find\_vertical\_seam(cumulative\_map)

# Create a display image with the same color as the original

display\_the\_image = np.copy(image)

# Highlight the seam in red for visualization

for i in range(image.shape[0]):

display\_the\_image[i, seam[i], :] = [0, 0, 255] # Mark the seam in red

# Display the image with the highlighted seam

self.display\_the\_image(display\_the\_image)

# Update the Tkinter window to show the changes

self.root.update()

# Remove the vertical seam from the image

image = self.remove\_vertical\_seam(image, seam)

return image

# Method to pause seam carving

def pause\_seam\_carving(self):

self.paused = True

# Method to resume seam carving

def resume\_seam\_carving(self):

self.paused = False

# Method to perform seam insertion based on user input

def insert\_seams\_operation(self, image, num\_seams):

for \_ in range(num\_seams):

# Calculate the energy map of the current image

energy\_map\_image = self.energy\_map(image)

# Calculate the cumulative energy map

cumulative\_map = self.cumulative\_energy\_map(energy\_map\_image)

# Find the vertical seam with the maximum cumulative energy

seam = self.find\_vertical\_seam(cumulative\_map)

# Insert the seam into the image

image = self.insert\_vertical\_seam(image, seam)

return image

# Main entry point

if \_\_name\_\_ == "\_\_main\_\_":

# Create the Tkinter root window

root = tk.Tk()

# Create an instance of the SeamCarvingApp class

app = Seam(root)

# Start the Tkinter event loop

root.mainloop()