Overview

Seam-carving is a content-aware image resizing technique where the image is reduced in size by one pixel of height (or width) at a time. A *vertical seam* in an image is a path of pixels connected from the top to the bottom with one pixel in each row. (A *horizontal seam* is a path of pixels connected from the left to the right with one pixel in each column.) Below is the original 507×285 pixel image; further below we see the result after removing 150 vertical seams, resulting in a 30% narrower image.

Unlike standard content-agnostic resizing techniques (e.g., cropping and scaling), the most interesting features (aspect ratio, set of objects present, etc.) of the image are preserved.

Although the underlying algorithm is simple and elegant, it was not discovered until 2007. Now, it is a core feature in Adobe Photoshop (called "Content-Aware Scale") and other computer graphics applications.

High-level overview

In a paper titled "Seam Carving for Content-Aware Image Resizing," Shai Avidan and Ariel Shamir describe a novel method of resizing images. You are welcome to read the paper,[1] but I recommend starting with the YouTube video:

http://www.youtube.com/watch?v=vIFCV2spKtg

After you've watched the video, the terminology in the rest of this document will make sense.

Some conventions

In image processing, pixel (i, j) refers to the pixel in **column** i and **row** j, with pixel $(0, 0)$ at the upper-left corner and pixel $(W - 1, H - 1)$ at the lower-right corner.

[1] https://perso.crans.org/frenoy/matlab2012/seamcarving.pdf

> **Important!** This is the opposite of the standard mathematical notation used in linear algebra, where (i, j) refers to row i and column j and $(0, 0)$ is at the lower-left corner.

We also assume that the color of each pixel is represented in RGB space, using three integers between 0 and 255. In our case, we represent a color as a tuple of three values corresponding to the red, green, and blue components.

Setup

Download all starter files from this link:

http://penoy.admu.edu.ph/~guadalupe154884/classes/csci30/seamcarving_files.zip

As with the midterm project, you will also need to set up a Python *virtual environment*, which allows us to install Python packages without messing up the system Python installation. To do this, run the command `python -m venv seamcarvingenv`. This will produce a folder called `seamcarvingenv`. Now your project has its own virtual environment. Generally, before you start using it, you'll first activate the environment by executing a script that comes with the installation:

- In Linux/macOS, run `source seamcarvingenv/bin/activate`.
- In Windows, run `seamcarvingenv\Scripts\activate`.

Once you can see the name of your virtual environment (`seamcarvingenv` in this case) in your terminal, then you know that your virtual environment is active.

Next, install the packages used by this project by running `pip install -r requirements.txt`. This file lists the packages, and more importantly, their respective version numbers, which ensures consistency.

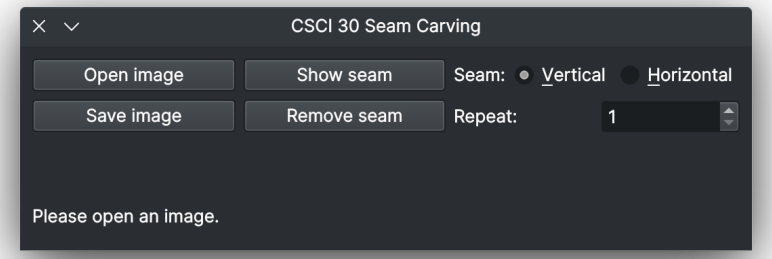
Verify that the packages have been installed correctly by running the command `pip freeze`. It should look something like this:

```
brian@thonkpad ~/classes/csci30/projects/seamcarving
(seamcarvingenv) % pip freeze
Pillow==9.3.0
PyQt6==6.4.0
PyQt6-Qt6==6.4.0
PyQt6-sip==13.4.0
```

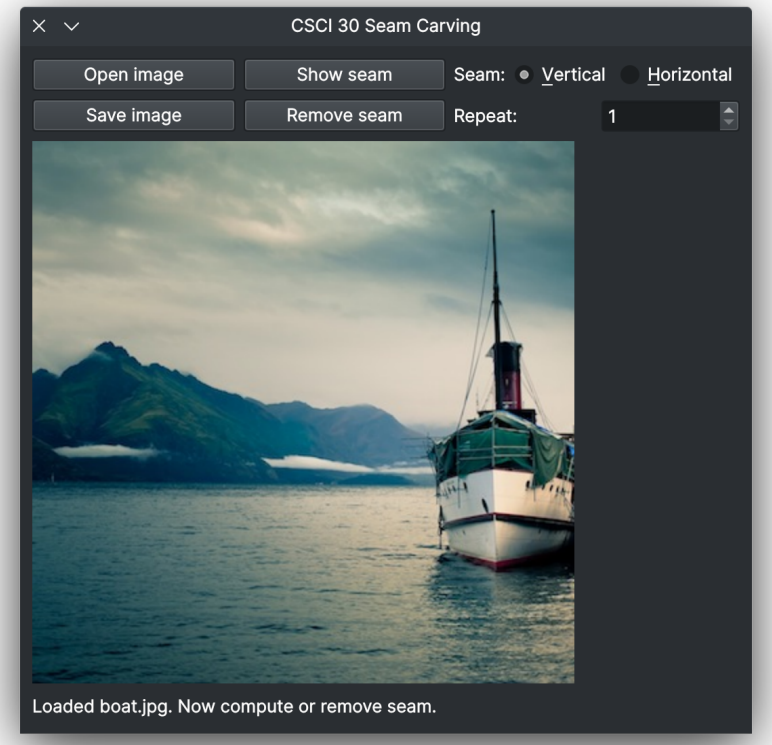
> **Important!** If you are having trouble setting up the virtual environment, please don't proceed further. Don't hesitate to ask for help!

Included as part of the starter files is the file `gui.py`, which provides a graphical user interface for the seam carver. To test whether the GUI works on your machine (and thus

the PyQt6 packages are properly installed), you can run `python gui.py` in the terminal. You should see a window pop up, like the one below.



At this point none of these buttons will work yet, except for the "Open image" and "Save image" buttons. You can try opening one of the images provided in the `data` folder.



Once you are done using the virtual environment, you can deactivate it by running the `deactivate` command. After executing the `deactivate` command, your terminal returns to normal. This change means that you've exited your virtual environment. If you interact with Python or `pip` now, you'll interact with your globally configured Python environment.

If you want to go back into a virtual environment that you've created before, you again need to run the `activate` script of that virtual environment.

The seam carver interface

Your task is to create a data type for the seam carver. Write a class named `SeamCarver` that implements the following interface:

```
class SeamCarver(Picture):
    def energy(self, i: int, j: int) -> float
        # return the energy of pixel at column i and row j
    def find_vertical_seam(self) -> list[int]:
        # return a sequence of indices representing the
        #   lowest-energy vertical seam
    def find_horizontal_seam(self) -> list[int]:
        # return a sequence of indices representing the
        #   lowest-energy horizontal seam
    def remove_vertical_seam(self, seam: list[int]):
        # remove vertical seam from picture
    def remove_horizontal_seam(self, seam: list[int]):
        # remove horizontal seam from picture
```

In addition to these methods, you also have access to the methods inside the `Picture` class (defined in `picture.py`), as `SeamCarver` inherits from `Picture`. Note that the `Picture` class inherits from `dict`, so as a result we can access the pixel at column *i* and row *j* using `self[i, j]`.

```
class Picture(dict):
    def __init__(self, picture: Image.Image):
        # constructor
    def picture(self) -> Image.Image:
        # get the current picture
    def width(self) -> int:
        # return the width of current picture
    def height(self) -> int:
        # return the height of current picture
```

The implementation of the `Picture` class is already provided for you.

Part 1: Energy calculation

The first step is to calculate the *energy* of a 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

(as you will see in the next step). For this project, you will use the *dual-gradient energy function*,[2] which is described below.

A high-energy pixel corresponds to a pixel where there is a sudden change in color (such as the boundary between the sea and sky or the boundary between the surfer on the left and the ocean behind him). In the image above, pixels with higher energy values have whiter values. The seam-carving technique avoids removing such high-energy pixels.

The energy function

The dual-gradient energy function defines the energy of pixel (i, j) as

$$e(i,j) = \Delta_x^2(i,j) + \Delta_y^2(i,j),$$

where $\Delta_x^2(i,j)$ and $\Delta_y^2(i,j)$ are the square of the x- (horizontal) and y-gradients (vertical) respectively. In turn, these are defined as

$$\Delta_x^2(i,j) = R_x(i,j)^2 + G_x(i,j)^2 + B_x(i,j)^2$$
$$\Delta_y^2(i,j) = R_y(i,j)^2 + G_y(i,j)^2 + B_y(i,j)^2$$

where the central differences $R_x(i,j)$, $G_x(i,j)$, and $B_x(i,j)$ are the absolute value in differences of red, green, and blue components between pixel $(i+1, j)$ and pixel $(i-1, j)$. Define $R_y(i,j)$, $G_y(i,j)$, and $B_y(i,j)$ analogously.

We define the energy of pixels at the border of the image to use the same pixels but to replace the nonexistent pixel with the pixel from the opposite edge.

For the following examples, consider the 3×4 image with RGB values (each component is an integer between 0 and 255) as shown in the table below.

(255, 101, 51)	(255, 101, 153)	(255, 101, 255)
(255, 153, 51)	(255, 153, 153)	(255, 153, 255)
(255, 203, 51)	(255, 204, 153)	(255, 205, 255)
(255, 255, 51)	(255, 255, 153)	(255, 255, 255)

Example 1. The energy of pixel $(1, 2)$ is calculated from pixels $(0, 2)$ and $(2, 2)$ for the x-gradient, so:

$$R_x(1,2) = 255 - 255 = 0,$$
$$G_x(1,2) = 205 - 203 = 2,$$
$$B_x(1,2) = 255 - 51 = 204,$$

yielding $\Delta_x^2(1,2) = 0^2 + 2^2 + 204^2 = 41620$; and pixels $(1, 1)$ and $(1, 3)$ for the y-gradient:

$$R_y(1,2) = 255 - 255 = 0,$$
$$G_y(1,2) = 255 - 153 = 102,$$
$$B_y(1,2) = 153 - 153 = 0,$$

[2] This function works similarly, but not exactly, to the Sobel filter, which is widely used in practice for edge detection algorithms.

yielding $\Delta_y^2(1,2) = 0^2 + 102^2 + 0^2 = 10404$.

Thus, the energy of pixel $(1,2)$ is $41620 + 10404 = 52024$.

Example 2. We calculate the energy of the border pixel $(1,0)$ in detail:

$$R_x(1,0) = 255 - 255 = 0,$$
$$G_x(1,0) = 101 - 101 = 0,$$
$$B_x(1,0) = 255 - 51 = 204,$$

yielding $\Delta_x^2(1,0) = 0^2 + 0^2 + 204^2 = 41616$.

Since there is no pixel $(i, j-1)$ we wrap around and use the corresponding pixel from the bottom row the image, thus performing calculations based on pixel $(i, j+1)$ and pixel $(i, H-1)$.

$$R_y(1,0) = 255 - 255 = 0,$$
$$G_y(1,0) = 255 - 153 = 102,$$
$$B_y(1,0) = 153 - 153 = 0,$$

yielding $\Delta_y^2(1,0) = 0^2 + 102^2 + 0^2 = 10404$.

Thus, the energy of pixel $(1,0)$ is $41616 + 10404 = 52020$.

To summarize, the energies for each pixel is given in the table below.

20808.0	52020.0	20808.0
20808.0	52225.0	21220.0
20809.0	52024.0	20809.0
20808.0	52225.0	21220.0

For convenience, you may define your own functions as well.

Part 2: Seam identification

The next step is to find a vertical seam of minimum total energy. (Finding a horizontal seam is analogous.) This is similar to the classic shortest path problem in a weighted directed graph, but there are three important differences:

- The weights are on the vertices instead of the edges.
- The goal is to find the shortest path from any of the W pixels in the top row to any of the W pixels in the bottom row.

- The directed graph is *acyclic*, where there is a downward edge from pixel (i, j) to pixels $(i-1, j+1)$, $(i, j+1)$, and $(i+1, j+1)$, assuming that the coordinates are in the prescribed ranges.

Seams cannot wrap around the image (e.g., a vertical seam cannot cross from the leftmost column of the image to the rightmost column).

Notice that the resulting graph is dense, and attempting to do a depth-first search for every possible starting pixel in the topmost row will blow up into exponential time overall. However, utilizing *dynamic programming* reduces the search to linear time in the number of image pixels.

Finding a vertical seam

The `find_vertical_seam` method should return a list of length H such that entry i is the column number of the pixel to be removed from row i of the image.

Example 3. Consider the 6×5 image below.

(78, 209, 79)	(63, 118, 247)	(92, 175, 95)	(243, 73, 183)	(210, 109, 104)	(252, 101, 119)
(224, 191, 182)	(108, 89, 82)	(80, 196, 230)	(112, 156, 180)	(176, 178, 120)	(142, 151, 142)
(117, 189, 149)	(171, 231, 153)	(149, 164, 168)	(107, 119, 71)	(120, 105, 138)	(163, 174, 196)
(163, 222, 132)	(187, 117, 183)	(92, 145, 69)	(158, 143, 79)	(220, 75, 222)	(189, 73, 214)
(211, 120, 173)	(188, 218, 244)	(214, 103, 68)	(163, 166, 246)	(79, 125, 246)	(211, 201, 98)

The corresponding pixel energies are shown below, with a minimum energy vertical seam highlighted in pink. In this case, `find_vertical_seam` returns the list `[3, 4, 3, 2, 2]`.

57685.0	50893.0	91370.0	25418.0	33055.0	37246.0
15421.0	56334.0	22808.0	54796.0	11641.0	25496.0
12344.0	19236.0	52030.0	17708.0	44735.0	20663.0
17074.0	23678.0	30279.0	80663.0	37831.0	45595.0
32337.0	30796.0	4909.0	73334.0	40613.0	36556.0

When there are multiple vertical seams with minimal total energy, your method can return any such seam.

Your `find_vertical_seam` method should work by first solving a base case subproblem, and then using the results of previous subproblems to solve later subproblems. Suppose

$M(i,j)$ is the cost of the minimum-cost path ending at (i,j), and $e(i,j)$ is the energy of pixel (i,j) as previously defined.

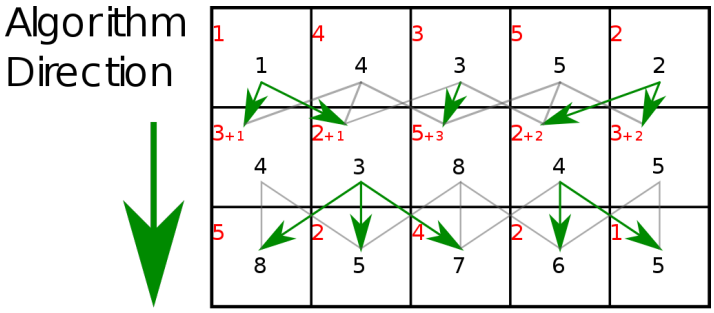
Then each subproblem is the calculation of $M(i,j)$ for some i and j. The top row is trivial, $M(i,0)$ is just $e(i,0)$ for all i. For lower rows, we can find $M(i,j)$ simply by adding the $e(i,j)$ to the minimum-cost path ending at its top left, top middle, and top right pixels, or more formally:

$$M(i,j) = e(i,j) + \min(M(i-1,j-1), M(i,j-1), M(i+1,j-1)).$$

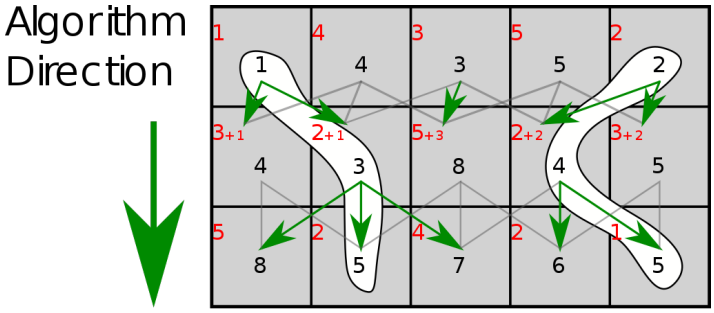
In short, we start from one side of the 2D image array and process row-by-row or column-by-column (for vertical and horizontal seam carving respectively).

The images below depict the dynamic programming process to compute one optimal seam. Each square represents a pixel, with the top-left value in red representing the energy value of that pixel. The value in black represents the cumulative sum of energies leading up to and including that pixel.

For each pixel in the rest of the rows, the energy is its own energy plus the minimal of the three energies above. Repeat until the bottom is reached.



For the lowest energies we have at the end, work back up the minimals to recover the seam with minimal energy.



Important! Python has a hard limit of 1000 nested recursive calls (you can check using the `sys.getrecursionlimit()` function). Thus a recursive (top-down) approach will almost certainly not be able to handle images of large-ish size (say 500×500). I recommend writing your code iteratively. Although you can manually override this hard limit, you will run the risk of running out of memory first before this limit would have been reached.

An equivalent (but slower approach) is to build an explicit graph object and run some sort of shortest paths algorithm. You are welcome to try this approach, but be warned it is slower and also harder to write, and it may not be possible to sufficiently optimize your code so that it passes the autograder timing tests.

(*Pro-tip:* If you need an "infinite value" in Python, you can use the constant `sys.maxsize` or any arbitrarily large number like `1e20` for this purpose.)

Finding a horizontal seam

The behavior of `find_horizontal_seam` is analogous to that of `find_vertical_seam` except that it should return an array of W such that entry y is the row number of the pixel to be removed from column y of the image. Your `find_horizontal_seam` method **should not** be a copy-and-paste of your `find_vertical_seam` method! Instead, consider transposing the image, running `find_vertical_seam`, and then transposing it back.

Part 3: Seam removal

The final step is remove from the image all of the pixels along the seam. Given a list of integers that correspond to a vertical seam, the `remove_vertical_seam` removes those pixels along the seam from the picture. Note that this is a *destructive* operation, meaning the picture will be modified after doing this, as we are not working on a copy. Typically, it will be called with the output of `find_vertical_seam`, but be sure that it works for *any* valid seam.

One way to do this is to "shift" all pixels after the seam one place to the left to fill the gap, which you will do per row of the picture. Then simply delete the last pixel in the row from the picture. (*Hint:* Since your `Picture` object is just a `dict`, you can use the `del` statement for this purpose.[3])

The `remove_horizontal_seam` method works in the same manner. Again, your implementation should not be a copy-paste of `remove_vertical_seam`; do the same trick as you did for `find_vertical_seam` of transposing the picture.

Note that the `remove_vertical_seam` method shrinks the width of the picture by 1 pixel, and `remove_horizontal_seam` shrinks the height of the picture by 1 pixel.

Other requirements

Running time

The `energy` method should only take $O(1)$ time. All other methods should run in $O(WH)$ time in the worst case. Asymptotically slower implementations may "time out" during testing.

[3] https://docs.python.org/3/tutorial/datastructures.html#the-del-statement

As a rule of thumb, my implementation can carve seams at a rate of roughly one per second for medium-sized images. If you cannot carve 10 seams in a few minutes from the provided test images, you should rethink your implementation.

Exceptions

Your code should throw an exception when called with invalid arguments.

- By convention, the indices i and j are integers between 0 and $W - 1$ and between 0 and $H - 1$ respectively. Throw an `IndexError` exception if either i or j is outside its prescribed range.
- Throw a `SeamError` exception if `remove_vertical_seam` or `remove_horizontal_seam` is called with a list of the wrong length or if the list is not a valid seam (i.e., two consecutive entries differ by more than 1).
- Throw a `SeamError` exception if `remove_vertical_seam` or `remove_horizontal_seam` is called when the width or height of the current picture is 1, respectively.

Submission

Submit the following files:

- The Python source file `seamcarver.py` (no need to include `picture.py` and `gui.py`)
- A fully-accomplished Certificate of Authorship (use whichever version is appropriate), with the filename `seamcarving-[ID1]-coa.pdf` or `seamcarving-[ID1]-[ID2]-coa.pdf` (without the brackets)
- **Do not** include the Python virtual environment folder and the `__pycache__` folder

Compress these files into a zip archive (`.zip`) or a gzipped tarball (`.tar.gz`) with the filename `seamcarving-[ID1]` or `seamcarving-[ID1]-[ID2]` (without the brackets) with the appropriate file extension, and submit it in the Canvas submission module for this project.

The final project is due on **December 5 (Monday) at 6 PM.** See the section below for the late submission policy.

Project defense

Upon submission, you will need to sign up for a project defense. Project defense days are scheduled from **December 5 (Monday)** to **December 10 (Saturday)** (except Dec 8, which is a holiday); the last day will be online-only, while the others will be done onsite.

- On-site defenses will be held at **F-204**. [**Update (Nov 17): Changed venue to F-204.**]

- Online defenses will be held via Zoom; the invite link will be posted on Canvas. Note that online defenses are not limited to groups with members from the online-only section, though on-site groups are *strongly encouraged* to have an on-site defense.

(*Note:* Should the need arises, I are willing to open more slots beyond the specified dates, but I would prefer not doing so.)

Sign-ups will open on **November 28 (Monday) at 10 AM.** Go to the following link to sign up for a slot once it opens:

(to be announced)

Slots will close one hour before their scheduled start time.

> **Important!** Groups with members from the online-only section will **not** be allowed to sign up for an onsite defense.

Have one of you in the group sign up for a slot using the link above. **Please make sure that everyone in your group will be present on your chosen time slot.** Also, please arrive about 5 to 10 minutes before your scheduled time slot. Groups that arrive late will be allowed to use only the remaining minutes of their slot.

The project defense will last **at most twenty (20) minutes**; this could end earlier if there are no major issues or problems with your project. Since I will be testing your project on my machine, no need to bring your own devices. As such, **please ensure that you have already submitted on Canvas** (especially those who will be having their defense ahead of the due date).

Grading

Your final project is graded using the following criteria:

- 70 points — correctness/efficiency (autograded)
- 10 points — other matters (e.g., coding style)
- 20 points — project defense

The last two criteria are graded on a **subtractive** scale. That means, the default score is the number of points specified if there are no problems, but this will be subject to deduction for any issues/errors found.

Extra credit opportunity

Feel free to try out your own pictures (PNG and JPEG images work best), and send me (via Discord, Canvas, or email) any interesting before/after shots. I recommend human faces. Do this by **December 9, 2022 at 11:59 PM** to receive the extra credit.

Also feel free to post these in the `#off-topic` channel in the Discord server!

Extra fun opportunity (not for extra credit)

Be warned that these are challenging, so make sure your project works first!

- Try to implement a version of the `SeamCarver` class that avoids the need to re-compute the entire energy matrix every time a seam is removed. This will require getting fancy with your data structures. If you do this, let me know! This should make your `SeamCarver` class extremely fast.
- Try to implement a way to assign arbitrary energies to pixels. For example, you can force a region of pixels to have zero energy so that it will be carved out first (hence you can do *targeted removal*). Or alternatively, you can force a region of pixels to have infinite energy so that it will be preserved (this is very useful for preserving the faces in a picture). If you watched the video towards the end, this is one way of extending the functionality of seam carving.

Other matters

Groupings

You may work alone or you may work in groups of **at most three** members (this can be different from your midterm project or pset partners). Your groupmates can come from different sections.

Despite this, a peer grading system will **not** be implemented. That means, with rare exceptions, all groupmates will receive the same grade even if groupings ended up being somewhat inequitable. Please let me know immediately if your group went really off the rails.

Late submission policy

Late submissions are allowed but strongly discouraged. Submissions made after the due date will be penalized with an 85% cap on the grade. You will not be allowed to sign up for a slot for the project defense until you have submitted.

Rescheduling a project defense

In situations where you or your groupmate (or all of you) became suddenly unavailable for your chosen time slot due to extenuating circumstances, you are advised to inform me immediately so that necessary adjustments can be made. In most cases, you will be asked to reschedule your project defense.

You have at most 48 hours to sign up for another slot if asked to reschedule. Failure to reschedule your project defense may lead to a grade of zero for the project defense component of your grade.

Any questions?

Any questions, clarifications, and concerns about the final project specs should be directed to the `#final-project` channel on the Discord server, or via email or Canvas.

Changelog

- Nov 17
  - Added some clarification for the "other matters" component in the grade
  - Added some tips for dealing with infinite values in Python
  - Changed venue from F-215 to **F-204**
- Nov 8: Initial release