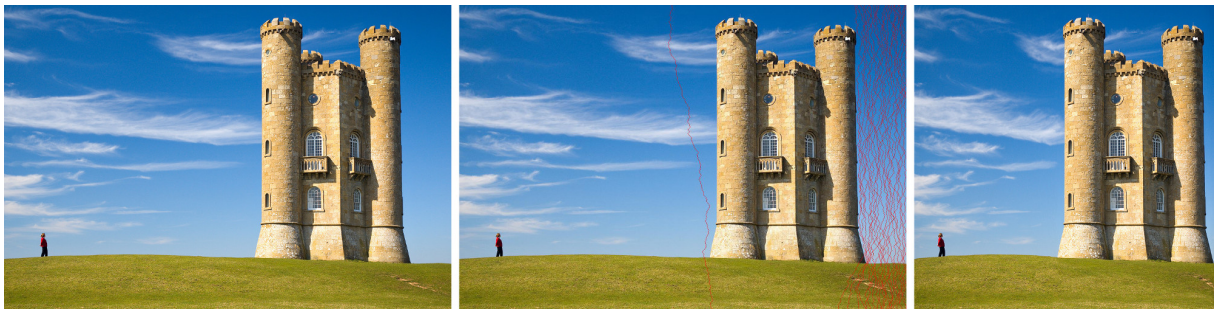
Контекстно-зависимое масштабирование изображений

Даниил Киреев, Владимир Гузов, Влад Шахуро

Graphics & Media Lab
Vision Group

Обзор задания

В данном задании предлагается реализовать алгоритм, применяющийся для контекстно-зависимого масштабирования изображений. При стандартном подходе изображение равномерно деформируется по всей длине при изменении размера (объекты на изображении уменьшаются вместе со всем изображением). Данный же алгоритм учитывает контекст, и деформация происходит так, что объекты сохраняют свои размеры. Кроме того, если с помощью маски выделить какой-нибудь объект, то его можно удалить из изображения или наоборот оставить неизменным.



Для удобства выполнения задание разделено на подпункты.
Для начала нужно скачать следующие файлы из проверяющей системы:

1. Скрипт для тестирования run.py. Добавьте ему права на выполнение (chmod +x run.py в консоли).
2. Архив с тестами. Его следует разархивировать в папку **tests**. Для запуска тестов вам понадобится библиотека **pytest**.
3. Архив с дополнительными файлами для тестирования и визуализации. В данном случае это файлы **common.py, visualization.ipynb**.

Также создайте файл **seam_carve.py**, в котором необходимо написать все функции в задании и впоследствии сдать в проверяющую систему после успешного локального тестирования.
Файлы должны располагаться следующим образом:

```
├── common.py
├── run.py
├── seam_carve.py
├── visualization.ipynb
└── tests/
    ├── 01_unittest_energy/
    └── ...
```

Описание задания

Идея алгоритма заключается в том, чтобы удалять *швы* с наименьшей *энергией* из изображения. Шов — это связанная кривая, соединяющая первую и последнюю строчки/столбцы изображения (на втором изображении швы выделены красным). Энергией же каждой точки мы будем называть модуль градиента яркости в данной точке.
Если шов проходит через малое количество перепадов яркости, это означает, что он имеет малую энергию. Заметим, что объекты обычно имеют более сложную структуру, чем фон, а значит швы, которые будут проходить через них, будут иметь более высокую энергию, следовательно, мы их не удалим.

* Для всех промежуточных вычислений (подсчет градиентов, энергии) используйте тип `float64`.

1. Подсчет энергии изображения (1 балл)

Итак, энергия точки — это модуль градиента яркости в данной точке. Вам необходимо реализовать функцию `compute_energy(image)`, которая будет возвращать матрицу энергии изображения.
Яркость изображения — компонента Y в цветовой модели YUV. Конверсия из RGB осуществляется по следующей формуле:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Чтобы найти градиент, необходимо найти частные производные по каждому из направлений (по X и по Y). Для того, чтобы найти производную по X, нужно для каждого пикселя из правого соседа вычесть левый (получим разностную производную второго порядка точности):

$$I'_x(x, y) = \frac{I(x+1, y) - I(x-1, y)}{2}$$

Будем считать, что пиксели за границей изображения повторяют значения на границе ($I(-1, y) = I(0, y)$ и $I(w, y) = I(w-1, y)$). Тогда

$$I'_x(0, y) = I(1, y) - I(0, y)$$

$$I'_x(w-1, y) = I(w-1, y) - I(w-2, y)$$

Здесь w — ширина изображения, индексация начинается с нуля. Аналогично можно найти частную производную по Y (из нижнего пикселя вычитаем верхний). Теперь для того, чтобы найти норму градиента, необходимо для каждого пикселя извлечь корень из суммы квадратов частных производных.

Проверьте свою реализацию с помощью юнит-тестов:

```
$ ./run.py unittest energy
```

2. Построение матрицы для поиска минимального шва (1 балл)

Для нахождения шва с минимальной энергией требуется построить матрицу перепадов энергии в изображении в определенном направлении (горизонтальном или вертикальном). Здесь разберем алгоритм для случая вертикального шва.

1. Создаем матрицу такого же размера, как исходное изображение, и инициализируем первую строку этой матрицы энергией соответствующих точек.

2. Построчно заполняем нашу матрицу. Для каждой точки мы находим минимального из трех верхних соседей (у крайних пикселей два соседа; в случае, если среди соседей есть одинаковые значения, то при формировании шва мы возьмем левого или верхнего), складываем его значение, значение энергии данной точки и записываем результат в матрицу.

3	4	3	5
5	4	5	6
1	1	1	1

3	4	3	5
8	7	8	9
8	8	8	9

   На данном примере: в первой матрице — значения энергии; во второй — матрица, которую требуется построить.

Теперь, чтобы найти шов на изображении, мы выбираем минимальное значение в последней строчке (если их несколько, то берем самое левое). Эта точка принадлежит минимальному шву. Теперь мы можем проследить все точки, которые принадлежат минимальному шву.

В этой части задания требуется реализовать функцию `compute_seam_matrix(energy, mode)`, которая по матрице энергии строит матрицу для поиска швов в зависимости от режима (mode может быть 'vertical' или 'horizontal'). Обратите внимание, что режим `mode` задает направление *сжатия* изображения, а не направление *шва*.
Проверьте свою реализацию с помощью юнит-тестов:
`$ ./run.py unittest seam_matrix`

3. Удаление шва с минимальной энергией (2 балл)

Эта часть задания проверяет удаление шва из изображения по матрице швов. Вам необходимо реализовать функцию `remove_minimal_seam(image, seam_matrix, mode)`, которая возвращает tuple (измененное изображение, None, маску шва). Маска шва это 2D матрица, соответствующая изображению, в которой 1 — пиксели, которые принадлежат шву, 0 — пиксели, которые не принадлежат шву.

Проверьте свою реализацию с помощью юнит-тестов:
`$ ./run.py unittest remove_seam`

4. Работа с маской (1 балл)

Далее описание будет приведено для случая горизонтального изменения изображения (для вертикального всё аналогично).

С помощью маски мы можем контролировать формирование швов. Мы можем либо увеличить энергию соответствующих точек (тогда эти точки останутся на итоговом изображении), либо наоборот уменьшить (тогда эти точки будут удалены из изображения). Увеличивать или уменьшать маску мы будем на заведомо большую величину: произведение количества строк изображения на количество столбцов изображения на 256 (верхняя оценка значения градиента в точке). С помощью такого метода мы можем, например, выделить с помощью маски лицо человека, и оно в результате работы

нашего алгоритма не будет деформировано. В рамках задания вам необходимо будет реализовать возможность удалять объекты, которые выделены маской (уменьшать энергию соответствующих точек), и защищать их от деформации (увеличивать энергию соответствующих точек).
Здесь вам нужно некоторым образом изменить функции `compute_seam_matrix(energy, mode, mask=None)`, `remove_minimal_seam(image, seam_matrix, mode)`, чтобы при наличии маски первая учитывала ее при построении матрицы швов, а вторая удаляла соответствующий шов не только из изображения, но и из маски.

Юнит-тесты для проверки: `$ ./run.py unittest seam_matrix_masked`

5. Работа с реальными изображениями (2.5 баллов за каждый тест)

В конечном итоге вам необходимо реализовать функцию `seam_carve` со следующими аргументами:

1. Входное изображение.
2. Режим работы алгоритма, одна из двух строк:

   `'horizontal shrink'` — сжатие по горизонтали,

   `'vertical shrink'` — сжатие по вертикали,
3. (Опциональный аргумент). Маска изображения — одноканальное изображение, совпадающее по размерам со входным изображением. Маска состоит из элементов `{-1, 0, +1}`. `-1` означает пиксели, подлежащие удалению, `+1` — сохранению. `0` означает, что энергию пикселей менять не надо.

Функция возвращает тройку — измененные изображение и маска, а также маска шва (1 — пиксели, которые принадлежат шву; 0 — пиксели, которые не принадлежат шву). В маске тоже необходимо удалить или добавить соответствующий шов.

Тесты для проверки всех режимов:

```
$ ./run.py unittest real_shrink
$ ./run.py unittest real_shrink_masked
```

В качестве ответа для каждой пары изображений используются сериализованные с помощью модуля pickle координаты швов во всех направлениях (горизонтальном и вертикальном).

Расширение изображения

С помощью подобного алгоритма также можно увеличить изображение. Для этого необходимо найти минимальный шов и вставить справа от него новый, который бы являлся усреднением минимального шва и следующего за ним (для каждого пикселя минимального шва берется его сосед справа). Если шов проходит по границе изображения и не имеет правого соседа, возьмем значение самого пикселя шва. Заметим, что если мы будем проводить несколько итераций нашего алгоритма, то минимальный шов каждый раз будет одним и тем же. Чтобы этого избежать необходимо увеличивать значение маски в точках данного шва (именно таким образом мы используем маску в случае расширения).

Визуализация

Работу алгоритма можно визуализировать с помощью ноутбука `visualization.ipynb` (может помочь вам с отладкой).

Полезные ресурсы

Видео-демонстрация работы алгоритма
Seam carving for content-aware image resizing — оригинальная статья с большим количеством поясняющих примеров, рекомендуется для подробного знакомства с алгоритмом.
Статья на Википедии про seam carving