Práctica 2 de Algorítmica

Seam carving (3 sesiones)

Curso 2017-2018

En qué consiste el “seam carving”

Seam carving es una técnica para reducir el tamaño de imágenes que es alternativa a un escalado. La idea es muy sencilla y se basa en eliminar estrechas franjas (de un pixel de anchura) que pasen por píxeles *poco importantes* de la imagen. Esta técnica se puede aplicar tanto en horizontal como en vertical. Veamos un ejemplo sacado de la página de wikipedia sobre seam carving:

En qué consiste el “seam carving”

Compara el resultado de aplicar un simple escalado de la imagen:

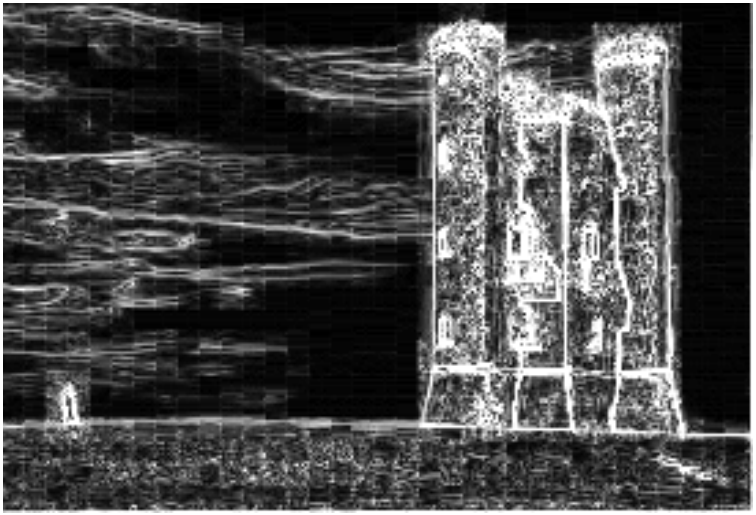
En qué consiste el “seam carving”

Con el resultado de aplicar seam carving:

En qué consiste el “seam carving”

Los pasos básicos son:

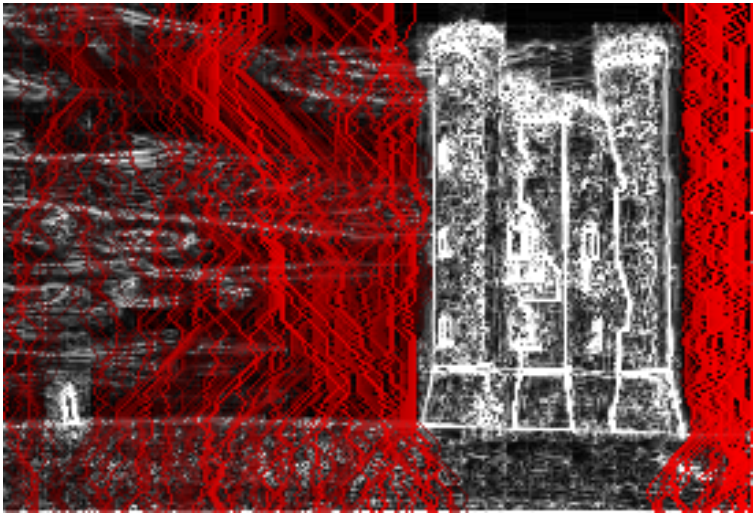
1. Calcular la *energía* o *gradiente* de cada pixel de la imagen.



En qué consiste el “seam carving”

Los pasos básicos son:

2. Calcular los caminos más cortos o *seams*



En qué consiste el “seam carving”

Los pasos básicos son:

3. Eliminar los pixels de esos caminos.

En qué consiste el “seam carving”

Algunos comentarios:

- Aunque los *seams* se pueden calcular usando diversas técnicas (por ejemplo, algoritmos de tipo flujo máximo/cortadura mínima), en esta práctica vamos a utilizar **programación dinámica**.

  En particular, encontrar el camino más corto en un grafo acíclico.

- Aunque normalmente se eliminan varios *seams* simultáneamente, empezaremos eliminándolos de uno en uno.
- Sin pérdida de generalidad vamos a reducir el **ancho** de las imágenes eliminando columnas. Todo esto se podría aplicar de manera muy similar para reducir la altura.

Detalles de implementación

En python existen varias formas de representar matrices:

1. Mediante un diccionario indexado por tuplas `(fila,columna)`.

```
m1 = {(i,j):0 for j in range(numcolumnas) for i in range(numfilas)}
m1[0,0] = 5 # azucar sintactico para m1[(0,0)] = 5
```

2. Mediante una lista de listas, que se accedería con `[fila][columna]`.

```
m2 = [[0 for j in range(numcolumnas)] for i in range(numfilas)]
m2[2][1] = 5 # m[2] es una lista python correspondiente a la 3a fila
```

3. Además de estas dos formas, algunos objetos permiten acceder a sus elementos como si se trataran de matrices usando una notación u otra. Ejemplos:
   - Los arrays de la biblioteca `numpy` se acceden de manera `[x][y]` (en general pueden tener más de dos dimensiones).
   - Los pixels de una `Image` de la biblioteca PIL se pueden indexar como `[x,y]`

Atención

- En esta práctica vamos a utilizar las coordenadas `x` e `y` para referirnos a los ejes *horizontal* y *vertical* de la imagen, respectivamente. Esos índices irán desde `0` hasta `width-1` y hasta `height-1` respectivamente.
- **PERO** vamos a utilizar la notación de matrices (fila y columna) debido a que nos interesa representar las imágenes mediante una **lista de filas**. Por ello, para acceder a un pixel usaremos `[y][x]`.

Carga y manipulación de imágenes con PIL

PIL es la abreviatura de Python Imaging Library. Se trata de una biblioteca para el lenguaje Python que permite manipular imágenes así como cargarlas y salvarlas en varios formatos. Tienes un tutorial en `https://pillow.readthedocs.org`.

Ejemplo de carga de una imagen:

```
from PIL import Image
import sys
file_name = sys.argv[1] # 1er argumento linea de comandos
img = Image.open(file_name)
max_x, max_y = img.size
```

Es interesante el comportamiento del método `load()` que devuelve un objeto que permite acceder a los pixels como si fuese un array bidimensional al estilo:

```
pix = img.load()
print pix[x, y]
pix[x, y] = value
```

En esta práctica vamos a convertir las imágenes a una **lista de listas**, como veremos a continuación.

Carga y manipulación de imágenes con PIL

Para convertir una imagen en una lista de listas procederemos en 2 etapas:

1. Convertirla en una matriz de la biblioteca numpy:

```
color_img = Image.open(file_name)
width,height = color_img.size
# convert the color image to a numpy array
color_numpy = numpy.array(color_img.getdata()).reshape(height, width,3)
```

2. Generar una lista de listas a partir del array numpy:

```
color_matrix = color_numpy.tolist()
```

Atención

- El motivo de hacer 2 etapas es que PIL y numpy tienen formas sencillas de *hablar entre sí*. En este caso numpy se está usando como una mera herramienta auxiliar que se podría eliminar a costa de utilizar bucles en python para ir leyendo los pixels de uno en uno, lo cual seguramente resultaría más lento.
- La imagen que cargamos está en color y por tanto cada pixel es una tripleta RGB, con lo que `color_matrix` es en realidad una lista de listas de listas (las últimas o más internas son las tripletas RGB, pero puedes ignorarlo/obviarlo).

Carga y manipulación de imágenes con PIL

Otras operaciones necesarias son:

- Generar una imagen a partir de la matriz:

```
def matrix_to_color_image(color_matrix):
    return Image.fromarray(numpy.array(color_matrix, dtype=numpy.uint8))
```

- Salvar la matriz en un fichero:

```
def save_matrix_as_color_image(color_matrix,filename):
    img = matrix_to_color_image(color_matrix)
    img.save(filename)
```

- Generar una versión de la imagen en escala de grises y con valores de tipo **float**:

```
grayscale_img = color_img.convert("F")
grayscale_numpy = numpy.array(grayscale_img.getdata()).reshape(height,width)
grayscale_matrix = grayscale_numpy.tolist()
```

- Crear una lista de listas similar pero para guardar el gradiente de la imagen en escala de grises:

```
gradient_matrix = [[0.0 for x in range(width)] for y in range(height)]
```

  Ambas listas de listas (`grayscale_matrix` y `gradient_matrix`) contienen ambas valores de tipo **float**.

Representación mediante listas de listas

¿Por qué tanto interés por representar las matrices mediante listas de listas (lista de filas y cada fila es otra lista)?

Para facilitar la operación de eliminar un *seam* de la imagen: Esta operación consiste en eliminar un pixel de cada fila de la imagen y este pixel se encuentra en posiciones diferentes para cada fila.

La ventaja de usar una lista de listas es que cada fila es una lista y las listas tienen un método que permite eliminar un elemento de manera muy sencilla, tal y como se observa en el siguiente ejemplo:

```
>>> a = range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a.pop(3)
>>> a
[0, 1, 2, 4, 5, 6, 7, 8, 9]
```

Tip

Que sea sencillo **no** quiere decir que sea eficiente, pues internamente las listas puede que se representen mediante vectores redimensionables y el coste de eliminar un elemento de una posición arbitraria es $O(n)$ como se ve en la siguiente página:

https://wiki.python.org/moin/TimeComplexity

Representación mediante listas de listas

De este modo, pintar un seam de un determinado color (para realizar la animación del algoritmo) resulta trivial:

```
def paint_seam(height,seam_path,color_matrix,path_color=[0,0,0]):
    for y in range(height):
        color_matrix[y][seam_path[y]] = path_color
```

Y eliminar el seam, que es lo que más nos interesa, es también muy sencillo:

```
def remove_seam(height,seam_path,matrix):
    for y in range(height):
        matrix[y].pop(seam_path[y])
```

En ambos casos, la variable `seam_path` es una lista de tamaño `height` (el alto de la imagen) que contiene, en cada posición `y`, el índice de la columna que hay que pintar o eliminar en la fila `y`.

La función `remove_seam` puede utilizarse tanto con la matriz `color_matrix` como con las matrices `grayscale_matrix` y `gradient_matrix`.

Inconveniente

Un pequeño inconveniente es que pasar las listas de listas a imagen es ineficiente. Esto no es problemático si guardamos la imagen resultante una sola vez al final, pero hace bastante más lenta la animación.

Cosas a manipular

Para los ejercicios de esta práctica vamos a necesitar manipular los siguientes objetos de tipo *lista de listas*:

- `color_matrix` es la imagen original en color.
- `grayscale_matrix` es la imagen en escala de grises para calcular el gradiente.
- `gradient_matrix` es la matriz que contiene el gradiente.
- `dp_matrix` es la matriz para los estados de programación dinámica.

Adicionalmente, los caminos o *seams* obtenidos se pueden representar directamente mediante una lista de las columnas de cada fila (es decir, una lista de talla `height`) que en el código se llama `seam_path`.

Otras variables que puedes ver en el esqueleto de programa que se os proporciona son: `width` y `height`.

Programa principal

Para facilitaros el desarrollo de esta práctica os proporcionamos dos esqueletos del programa. Ambos reciben argumentos por la línea de comandos como se muestra en este fragmento del fichero `template_seam_carving.py`:

```
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print('\n%s image_file {num_column|%%}\n'\
              % (sys.argv[0],))
        sys.exit()

    file_name = sys.argv[1]
    ncolumns  = sys.argv[2]
    ...
```

La versión inicial del programa tiene las siguientes características:

- El cálculo del *seam* devuelve un camino aleatorio (a cambiar).
- Las columnas se eliminan **de una en una**.

Puedes probar la siguiente animación:

```
python3 seam_carving_pract.py BroadwayTowerSeamCarvingA.png 0 80%
```

PERO ¡OJO! si pruebas `multi_seam.py` tal cual está entrará en un bucle infinito.

Nota sobre el entorno de programación

- El código está escrito en python3 utilizando espacios en lugar de tabuladores.
- Utiliza un editor que te facilite el modo python y el uso de espacios. Recuerda que mezclar espacios y tabuladores es fuente de *problemas* y que, a una mala, hay formas de sustituir los tabuladores por espacios. Por ejemplo, en el editor `emacs` puedes seleccionar todo y luego aplicar el comando `M-x untabify`.
- Si no aparece `python3` en el path, puedes ejecutar la siguiente línea para añadirlo:

```
PATH=$PATH:/opt/anaconda3/bin
```

- Para trabajar en tu propio ordenador es probable que necesites instalar algunos paquetes. En el caso de una distribución Ubuntu Linux suele ser necesario añadir los siguientes:

```
python3-numpy
python3-pil.imagetk
python3-tk
```

Actividades a realizar

Actividades a realizar

1. Completar la función que utiliza programación dinámica para calcular el camino más corto que atraviesa todas las filas de la imagen.
2. Calcular el gradiente de manera **incremental**.
3. Eliminar varios seams simultáneamente.

Cosas a entregar:

- Las dos primeras actividades se realizan sobre el fichero `template_seam_carving.py` que renombrarás como `seam_carving.py` y donde completarás los datos de entrega (fecha y nombres de 1 o 2 alumnos, ya que se puede realizari individualmente o por parejas).
- La tercera actividad se realiza sobre el fichero `template_multi_seam.py` que renombrarás como `multi_seam.py` y donde pondrás los mismos datos sobre la entrega.

Completar la función que calcula el camino más corto

- Un camino válido puede empezar, terminar y pasar por cualquiera de las columnas donde esté definido el gradiente (**menos la primera y la última** columna, esto lo hacemos para simplificar el código y por eso se ponen a infinito).
- Un camino válido solamente se puede mover en vertical o en diagonal. Es decir, cada vez que subes o bajas una fila, la coordenada de la columna solamente puede cambiar en -1,0 o 1.
- El coste de un camino es la suma de los gradientes de los pixels por los que pasa dicho camino.
- Debes retornar **uno de los caminos de menor coste** (puede que haya varios por temas de empates). Para ello has de recuperar uno de los mejores caminos (podría haber empates) **sin utilizar backpointers**. El camino resultante se devolverá como una lista python de talla número de filas (valor `height`) que contiene el índice de la columna en cada posición. Es decir, los pixels que forman el camino son los de coordenadas de la imagen fila `y` y columna `path[y]`.

OJO

Al acceder a la posición fila `y` y columna `path[y]` en una matriz representada mediante listas de filas, pondríamos `[y][path[y]]`.

Calcular el gradiente de manera incremental

Calcular el gradiente de manera **incremental**. Es decir, reutilizando las casillas de la matriz de gradientes de la iteración anterior que sigan siendo válidas y calculando únicamente las que habrán cambiado al eliminar un *seam* de la imagen. Para ello:

- Solamente ha de aplicarse a partir de la segunda iteración. Para ello la función recibe el seam de la iteración anterior salvo en la primera iteración en la que ese parámetro vale `None`.
- Debes eliminar el seam de las 3 matrices siguientes (eso YA está hecho en el código, es para que te quede más claro el funcionamiento):
  1. la imagen en color `color_matrix`.
  2. la imagen pasada a escala de grises `grayscale_matrix`.
  3. la matriz del gradiente `gradient_matrix`.
- Ten en cuenta que, en general, el gradiente cambia de una iteración a otra **únicamente alrededor del seam eliminado**. Para ello:
  - El gradiente utilizado se basa en el operador de Sobel.
  - Deduce qué pixels pueden depender de uno ya eliminado y calcula el umbral (en horizontal, teniendo en cuenta las filas superior e inferior y que éstas no pueden irse más allá de +-1 de una fila a otra).
  - Ten en cuenta que la columna eliminada del seam ahora es el pixel que antes estaba a su derecha.

Actividades a realizar con la plantilla `template_multi_seam.py`

Haz una versión donde el algoritmo de programación dinámica calcule varios seams simultáneamente. Los datos extra son:

- `N` es el número máximo de seams a eliminar en una iteración. Ten en cuenta que al final del algoritmo de seam carving es posible que el número de seams a eliminar sea menor que `N`. En tal caso, el valor máximo en realidad es `min(N,columnas_restantes)`.
- Un ratio (puesto por defecto a `0.8`) de manera que pararemos cuando el score del siguiente camino a eliminar multiplicado por ese ratio sea mayor que el score del mejor camino inicial.

En general puede que eliminemos menos seams de los que se pedía bien porque su score es muy malo respecto al mejor, bien porque los caminos no pueden usar pixels previamente utilizados por otros caminos y, en ese caso, han de ser descartados y pasar a otro. Una forma de detectar esto es apuntar las coordenadas de los caminos (incluso si están a medias) en un `set` del que no hace falta eliminar las coordenadas de los caminos abortados.

OJO

Esta versión requiere una versión de la función `remove_seam` capaz de recibir una lista de `seam_path` y debes de borrar las columnas de cada fila de manera correcta (pista: de derecha a izquierda).