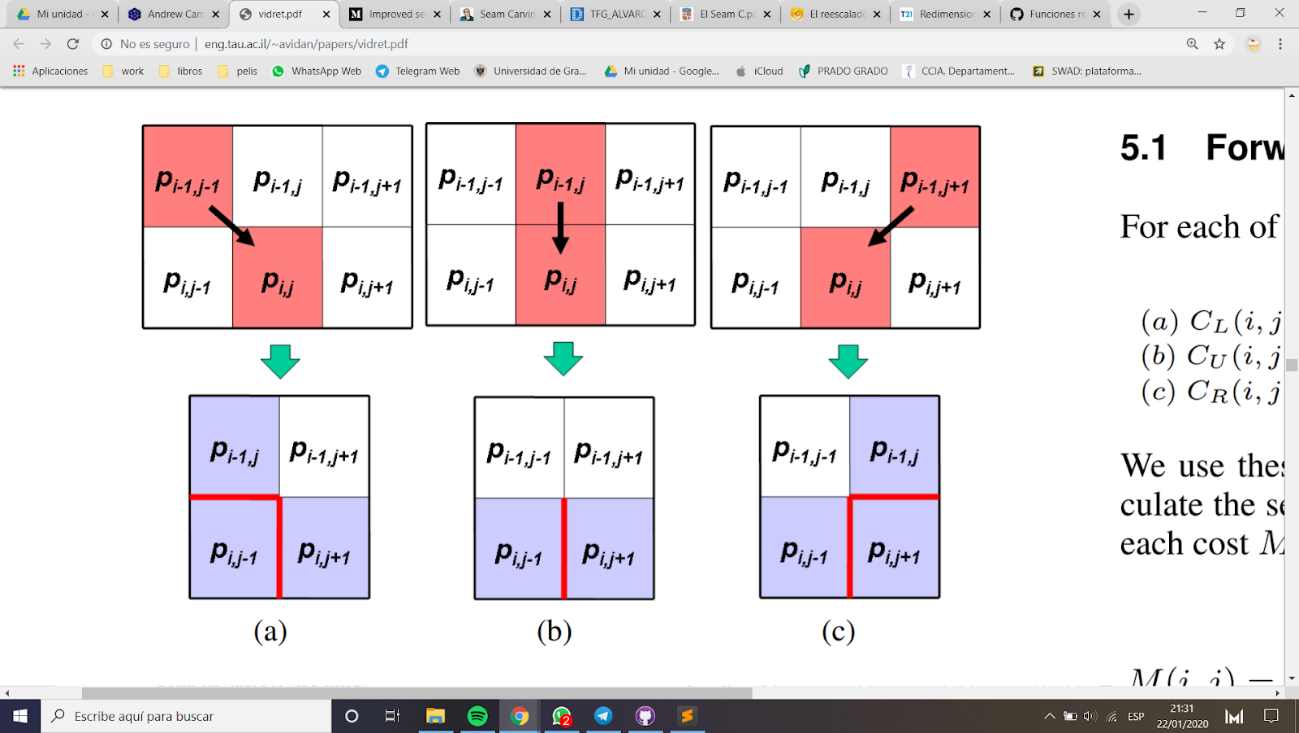
Forward Energy

Un año más tarde, los mismos autores publicaron el paper “Improved Seam Carving for Video Retargeting”, donde el objetivo era aplicar sean carving a vídeos, pero aparte introdujeron una mejora respecto al algoritmo original que puede aplicarse en imágenes estáticas: forward energy, que resulta en una mejora muy significativa, como veremos a continuación.

Como su nombre indica, mejora la función de energía. Hasta ahora el criterio para eliminar las seams se basaba en escoger la de menor energía, pero no teníamos en cuenta la **energía global** que se añade a la imagen al eliminar una seam. Esto último sucede porque píxeles vecinos que antes no lo eran, ahora lo son, por lo que la energía global de la imagen aumenta.

Forward energy predice qué píxeles serán adyacentes y se basa en ello para elegir la mejor seam a eliminar:



Definimos la diferencia de color entre dos píxeles arbitrarios p0 y p1:



La energía asociada a un píxel tendrá tres costes distintos, atendiendo a los tres casos que pueden darse al eliminar un píxel: si este está conectado a una seam arriba a la izquierda, a una sean justo arriba, o a una seam arriba a la derecha, tal y como se muestra en la imagen.

Los costes serán en cada caso:



Podemos ver como el primer coste  aparece en los tres casos, ya que los píxeles a los lados del píxel a eliminar se convierten siempre en vecinos.

Para resolver este problema volvemos a aplicar programación dinámica.

A cada píxel le asociaremos la energía de la energía mínima final de la seam que **acaba** en ese píxel. Para esto nos fijaremos en cada subproblema en las seams de los píxeles arriba a la izquierda, arriba en el centro y arriba a la derecha del píxel en cuestión.

Para escoger por cuál de estos píxeles continuamos, escogeremos el mínimo de los siguientes costes:



Como es natural, el paper no explica los casos particulares de las esquinas y bordes de la imagen. Nosotras hemos aprovechado la función de crear un borde con OpenCV para crear un borde que replique los píxeles (cv2.BORDER\_REPLICATE) creando bordes superior, izquierdo y derecho.

img = cv2.copyMakeBorder(img, top=1, bottom=0, left=1, right=1

borderType=cv2.BORDER\_REPLICATE)

Con ayuda de np.roll calculamos la fila superior, y la columna izquierda y derechas para cada píxel de la imagen con bordes.

U = np.roll(img, 1, axis=0)

L = np.roll(img, 1, axis=1)

R = np.roll(img, -1, axis=1)

Procedemos a calcular los costes, siguiendo la fórmula anterior:

cU = np.abs(R - L)

cL = np.abs(U - L) + cU

cR = np.abs(U - R) + cU

Y finalmente eliminamos los bordes con slicing, por lo que no hemos tenido que tratar las esquinas de forma especial.

cU = cU[1:, 1:-1]

cL = cL[1:, 1:-1]

cR = cR[1:, 1:-1]

M[0] será igual a cU[0], el caso base en el que no hay seams en la fila anterior.

Ahora para cada fila empezando en 1 calculamos tres matrices mU, mL y mR volviendo a utilizar np.roll:

mU = M[i-1]

mL = np.roll(mU, 1)

mR = np.roll(mU, -1)

Finalmente sumamos los costes como se especifica en la función y escogemos el mínimo coste:

mLUR = np.array([mL, mU, mR])

cLUR = np.array([cL[i], cU[i], cR[i]])

mLUR += cLUR

argmins = np.argmin(mLUR, axis=0)

M[i] = np.choose(argmins, mLUR)

energy[i] = np.choose(argmins, cLUR)

Ya tenemos nuestra matriz energy, que tendrá para cada píxel la energía mínima final de la seam que **acaba** en ese píxel.