

Visión por computador: proyecto final.

Seam carving for Content-Aware Image Resizing

Introducción

El algoritmo de seam carving es una herramienta para redimensionar las imágenes, se pueden aumentar creando nuevos píxeles o reducir quitando píxeles; al contrario de la redimensión tradicional estos no son un duplicado de los mismos, son totalmente nuevos. La filosofía principal sería quitar aquellos píxeles que no aporten un valor significativo a la imagen como podría ser en un paisaje quitar el azul del cielo, el azul del mar, marrón de regiones grandes de tierra, etc. Aquellas regiones donde haya muchos píxeles con un valor parecido donde no haya cambios importantes, la solución el gradiente. Este algoritmo utiliza la energía de los píxeles, en este caso la magnitud del gradiente; teniendo el gradiente se puede construir un camino mínimo desde un extremo de la imagen al otro por el camino de menor coste; una vez se tiene este camino se elimina de la imagen.



Figura 1.1: imagen normal 640x480.



Figura 1.2: imagen redimensionada por seam carving 490x430.

Como se observa en las figuras 1, la redimensión elimina ciertas regiones donde la densidad de el mismo pixel es el mismo pero hay que tener cuidado por que también elimina regiones donde hay sombra debido a su bajo gradiente y es posible que eliminen ciertas partes que son relevantes; para ello hay un método en el que se eligen los píxeles de la imagen que no se quieren borrar.

Como se dijo anteriormente también es posible un aumento, llegando a crear nuevos pixeles dentro de la imagen. Para lograr este resultado hay que buscar los seams de menor valor en la imagen, al igual que para reducir, sin embargo ahora no se eliminara la fila o columna sino que se abrirá una nueva linea donde su valor sera la media de los vecinos. De esta forma se conseguirá un efecto de aumento en la imagen.



Figura 2: aumento de la imagen 1.1 a 740x580

Reducir una imagen

Gradiente, la energía de los pixeles

Para resolver el coste que requiere cada camino, linea o “seam” se ha utilizado el gradiente. Mediante el método de sobel se obtiene el gradiente de toda la imagen donde cada pixel tiene un valor entre 0 y 255 siendo 0 coste nulo; unos valores como estos facilitaran el rendimiento ya que para imágenes grandes habrá que hacer miles de operaciones utilizando estos valores y el microprocesador obtiene mayor velocidad operando valores enteros que valores en coma flotante. El resultado sera algo parecido a la figura 3.



Figura 3.1: Imagen normal.

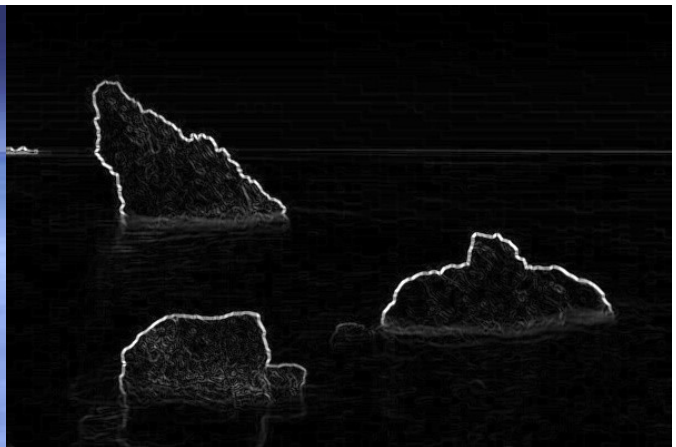


Figura 3.2: Gradiente de la 3.1.

Sabiendo donde como se aprecia en la figura 3.2 el contorno de los objetos relevantes en la imagen han sido marcados con un valor 255 un color blanco en la imagen, por lo tanto al rastrear la imagen se evitara utilizar estos pixeles, ya que si se eliminaran sin control podría aparecer resultados inesperados.

Energía de los seams

Esta claro que se conoce la energía de un pixel, pero como determinar la energía total de una linea, un seam completo. Primero un seam matemáticamente sera:

-Horizontal:

$$s^y = \{s_j^y\}_{j=1}^m = \{(j, y(j))\}_{j=1}^m, \text{ s.t. } \forall j |y(j) - y(j-1)| \leq 1.$$

-Vertical:

$$s^x = \{s_i^x\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n, \text{ s.t. } \forall i |x(i) - x(i-1)| \leq 1,$$

Donde se comprueba que para el horizontal el seam empieza desde izquierda y termina en derecha conteniendo un solo pixel de cada columna y para los verticales empiezan desde arriba hasta abajo conteniendo solo un pixel de cada fila.

Se notara a la imagen como I donde siendo s el seam eliminado :

$$I_s = \{I(s_i)\}_{i=1}^n = \{I(x(i), i)\}_{i=1}^n$$

Para finalizar sabiendo que es un seam la energía total, es decir la suma de todas las energías de sus pixeles vendrá dado por:

$$E(s) = E(I_s) = \sum_{i=1}^n e(I(s_i)).$$

Elegir que seam borrar

Sabiendo que es un seam, como obtener la energía de los pixeles y de los propios seams es hora de poder saber como eliminar todos los seams que se deseen de la imagen para llegar la dimensión que se quiera.

En primer lugar, ¿Que seam eliminar de toda la imagen? Eso es bien sencillo aquel seam donde su $E(s)$ la energía total sea la mínima de toda la imagen:

$$s^* = \min_s E(s) = \min_s \sum_{i=1}^n e(I(s_i))$$

Donde de todos los posibles seams es el que menos energía aporta al gradiente. Esto se aplica tanto a columnas como filas.

Algoritmo recursivo, eliminar todos los seams

La idea básica de este algoritmo seria fijar un método de borrado (ver sección mas abajo), seleccionar que seam borrar fila o columna y por último quitar dicho seam de la imagen. Luego volver a repetir el mismo proceso con la imagen obtenida del paso anterior y así recursivamente hasta llegar la dimensión deseada.

-¿Como se elige el seam de menor coste?

Podría pensarse en algo como la fuerza bruta, es decir comprobar todas las posibles rutas de una imagen y elegir la que menos coste tenga. Este proceso es bastante rudimentario y el coste de tiempo puede ser muy elevado, un simple proceso de eliminar un seam podría llegar a consumir tanto tiempo que este proceso sea inviable.

En lugar del proceso anterior se optara por usar programación dinámica, sabiendo el gradiente de toda la imagen se creara un mapa dinámico donde se mostrara todas las posibles rutas y su coste. Obviamente se seleccionará el camino con un coste mínimo, para ello hay que situarse al final del mapa y elegir la posición que menor coste de energía tenga, después habrá que marcha atrás elegir

el camino con coste mínimo, es decir elegir el pixel continuo o el vecino derecho o izquierdo del continuo, depende de cual tenga menor energía. El resultado sera algo parecido a la figura 4.

Seam carving da una aproximación para buscar los seams mediante programación dinámica:

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

Donde M es una matriz de nxm donde n y m son las filas y columnas de la imagen principal, $e(i, j)$ es el gradiente de dicho pixel. Se elegirá el mínimo valor de la matriz M de sus posibles opciones el pixel continuo el izquierdo o derecho.



Figura 4: La mínima columna y fila seleccionada en la imagen.

Orden de borrado

Hay cuatro posibles formas: la primera es primero borrar todas las columnas y después todas las filas, la segunda es igual que la anterior pero de forma inversa, la tercera es borrar filas y columnas alternando entre una u otra y la cuarta es elegir la forma optima probar todas las posibles combinaciones, esta última forma tiene un coste mayor de tiempo. Los resultados se pueden ver en la figura 5.

-Alternando filas y columnas, es un método rápido y en la mayoría de los casos eficaz. No siempre tiene buenos resultados pero podría ser como el comodín de los métodos, cuando no se sabe que debería borrarse primero este es un método rápido y efectivo. La implementación es sencilla: borra una fila y después una columna, fila, columna... hasta alcanzar el tamaño deseado. En el ejemplo propuesto figura 5, se observa que no tiene un buen resultado.

-Primero columnas, luego las filas; este es un método específico es conveniente aplicarlo solo cuando se sabe que la energía de las columnas es menor que la de las filas. Es rápido pero no se

recomienda usarlo demasiado.

-Primero filas, luego columnas; como el anterior pero ahora se borrarán primero todas las filas y luego todas las columnas.

-Orden optimo, el indicado cuando se quiere lograr la perfección sin importar el rendimiento. Este método propuesto por seam carving analiza todas las posibles combinaciones de borrado, elige la serie que menos energía necesita y la aplica. Suponiendo que se tiene $n \times m$ y se quiere conseguir una imagen $n' \times m'$ donde $k = c + r$, $r = (m - m')$, $c = (n - n')$. Se establece la siguiente función como la búsqueda del orden optimo:

$$\min_{s^x, s^y, \alpha} \sum_{i=1}^k E(\alpha_i s_i^x + (1 - \alpha_i) s_i^y)$$

Donde:

$$\alpha_i \in \{0, 1\}, \sum_{i=1}^k \alpha_i = r, \sum_{i=1}^k (1 - \alpha_i) = c$$

Si es 0 no se elige el seam y si es 1 se elige. Llevando esta función al terreno de la programación seam carving propone utilizar programación dinámica para encontrar el orden perfecto:

$$T(r, c) = \min(T(r-1, c) + E(s^x(I_{n-r-1 \times m-c})), T(r, c-1) + E(s^y(I_{n-r \times m-c-1})))$$

Al igual que en el apartado anterior donde se pretendía encontrar el seam optimo ahora nuevamente se utiliza la programación dinámica para encontrar el orden optimo.

$I_{n-r \times m-c}$ representa una imagen de las dimensiones $n-r \times m-c$. Y $T(r, c)$ será una matriz de tamaño $r \times c$ r =filas a borrar y c =columnas a borrar.

Creando este mapa dinámico se podrá obtener la ruta donde el coste de energía es mínimo.



Figura 5: imagen a ser redimensiada.



Figura 5.1: Alternado fila y columna.



Figura 5.2: Redimension optima.



Figura 5.3: Borrando primero columnas.



Figura 5.4: Borrando primero filas.

Como se observa en las imágenes la 5.2 es la optima, no hay ningún desperfecto, la mejor referencia que he encontrado en las imágenes para analizar la calidad de los resultados es observando el segundo árbol por la izquierda después del mas grande de todos hay uno pequeño. En la figura 5.2 se puede ver como no hay nada raro sin embargo en la 5.1 ese mismo árbol tiene un resultado un tanto deforme, en la 5.3 la deformidad del árbol es mas que evidente. La 5.4 también ha logrado buenos resultados pero de antemano no se podía saber que era necesario borrar antes las filas que las columnas, para ello se tiene el método optimo.

Aumentar una imagen

Aumentar una imagen es igual que reducirla, se hace una búsqueda del seam con menor coste pero ahora en vez de quitar dicho seam se abre un hueco a la derecha o izquierda y este es rellenado con el valor de la media de los pixeles vecinos, el pixel superior e inferior para las filas y el pixel izquierdo y derecho para las columnas. Si el proceso de reducción es denotado como I^t creado después de reducir la imagen t veces, entonces el proceso de aumento sera I^{-1} donde se introduce un nuevo seam artificial.

Intuitivamente el alcanzar el aumento de k seams podría resolverse como el aplicar I^{-1} un numero

de veces hasta alcanzar el tamaño deseado. En la practica se observa que esto no es correcto ya que se obtienen resultados como en la figura 7.1, esto se debe a que al introducir un nuevo seam se esta introduciendo el camino con menor coste de la imagen, por lo tanto al volver a elegir un seam se elegirá el mismo que anteriormente ya que es el que tiene menor coste. Volverá a ser introducido x veces.

Para resolver este problema la solución se encuentra en encontrar las k seams que se borrarán figura 6 primero y luego duplicarlas todas ella en forma de I^{-k} . Con esta técnica se consigue resolver el problema de duplicado de la misma seam una y otra vez. No es un proceso sencillo ya que este algoritmo es recursivo, se distinguirá entre posición relativa y posición absoluta; el seam elegido en una imagen 40x40 no sera el mismo que en su imagen original 100x100, osea el seam en la imagen original sera la posición absoluta, mientras que el elegido en la imagen reducida sera el relativo. Hay diversas formas de reconstruir estas lineas, aquí se a optado dada la pirámide que se obtiene al reducir las imágenes es posible saber la posición absoluta de un seam observando la posición superior es decir la imagen anterior al proceso de esta forma es posible saber que seam de una imagen 99x99 se ha eliminado de una 100x100. El resultado final se puede observar en la figura 7.2.

Figura 6: Seams seleccionados para borrar, una vez se tengan todos los seams se procederá al duplicado.



Figura 7: Imagen original a la que se aumentara las filas y columnas.



Figura 7.1: Aumento de la imagen uso recursivo.



Figura 7.2: Aumento de la imagen buscando primero los seams a eliminar.

Borrar objetos de la escena

La última opción que se explicara es la de eliminar objetos en la escena. El objetivo es marcar ciertos pixeles, dándoles un valor negativo en el gradiente de esta forma cuando se busquen seams estos tenderán a elegir dichos pixeles seleccionados para eliminarlos de la escena. Esta aplicación es delicada ya que no hay un orden optimo porque a priori no se conoce cuantas filas o columnas van a ser eliminadas. Lo mas sencillo seria decidir en cuestión de la imagen si se deben eliminar filas, columnas o ambas; cada opción dará un resultado diferente Figura 8, como se aprecia en dichas imágenes el mejor resultado ha sido obtenido borrando solo columnas.

Otro ejemplo podría mostrarse en la Figura 9 donde ahora se van a eliminar dos objetos, como se observa en los resultados ahora el primer método, borrar filas y columnas ha dado mejores resultados, por eso se repite que no hay mejor orden, método que otro eso dependerá de la imagen en si.



Figura 8: Imagen original.



Figura 8.1: Seleccionado un objeto a ser borrado.



Figura 8.2: Eliminación del objeto borrando una fila y después columna sucesivamente.



Figura 8.3: Eliminación del objeto borrando solamente columnas.

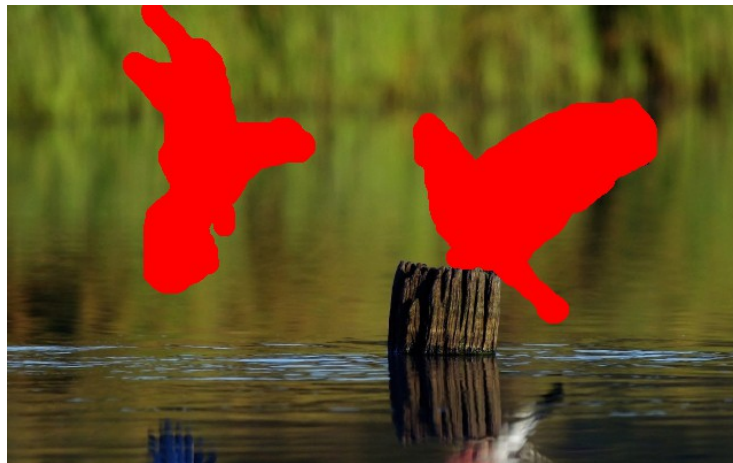


Figura 9: Dos objetos son seleccionados para borrar.

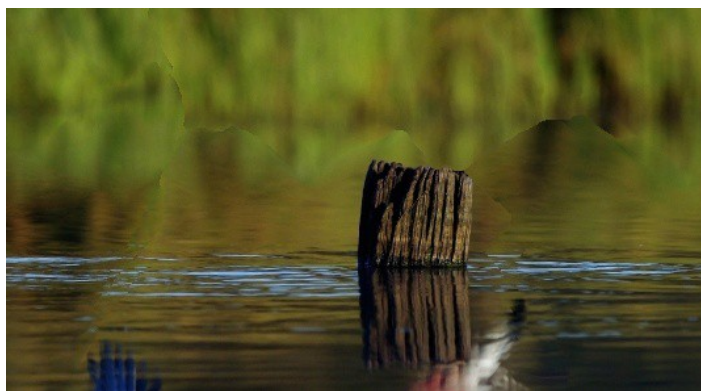


Figura 9.1: Se han borrado los dos objetos quitando seams verticales y horizontales.

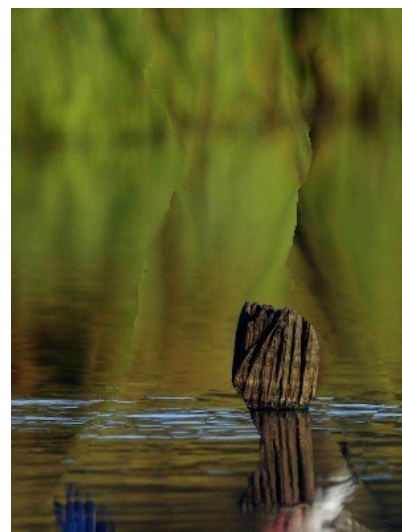


Figura 9.2: Borrado de los dos objetos unicamente eliminando columnas.

Especificación del código

Los cambios mas importantes que he realizado al código podrían ser:

-Uso de punteros para el acceso a matices, utilizando la versión de acceso “at<int>(i,j)” comprobé que el acceso era significativamente lento. La solución a este problema fue el uso de punteros por lo que en un bucle que recorre matrices:

```
for(int i=0; i<im.rows; i++)  
    for(int j=0; j<im.cols; j++)  
        mi.at<int>(i,j)
```

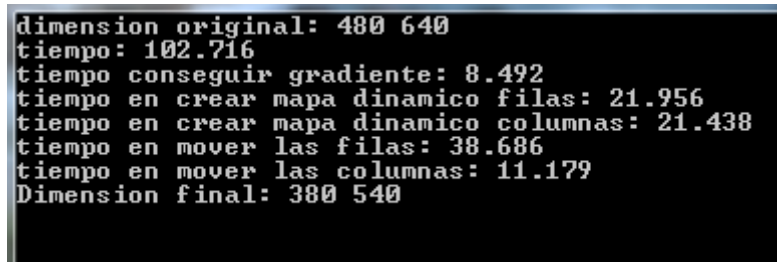
(1)

Se sustituyo por el siguiente:

```
for(int i=0; i<im.rows; i++)  
    row = Mt.ptr<int>(i)  
    for(int j=0; j<im.cols; j++)  
        row[j]
```

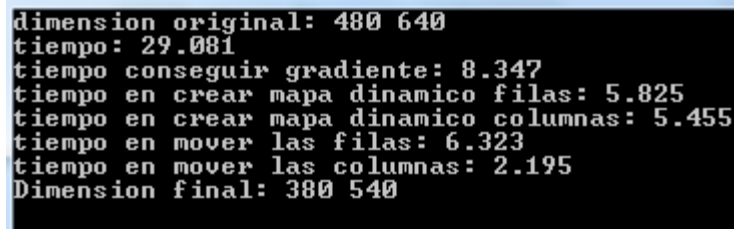
(2)

De esta forma se conseguía un rendimiento de hasta cuatro veces mejor.



```
dimension original: 480 640  
tiempo: 102.716  
tiempo conseguir gradiente: 8.492  
tiempo en crear mapa dinamico filas: 21.956  
tiempo en crear mapa dinamico columnas: 21.438  
tiempo en mover las filas: 38.686  
tiempo en mover las columnas: 11.179  
Dimension final: 380 540
```

Figura 10: tiempos de acceso a memoria sin punteros.



```
dimension original: 480 640  
tiempo: 29.081  
tiempo conseguir gradiente: 8.347  
tiempo en crear mapa dinamico filas: 5.825  
tiempo en crear mapa dinamico columnas: 5.455  
tiempo en mover las filas: 6.323  
tiempo en mover las columnas: 2.195  
Dimension final: 380 540
```

Figura 11: tiempos de acceso a memoria con punteros.

Donde se puede comprobar en los resultados de las figuras 10 y 11 como usando el método (2) se consigue un mejor rendimiento que utilizando el primer método (1).

-La búsqueda de filas ha sido creada igual que la búsqueda de columnas, tan solo que para encontrar las filas buscamos las columnas de la matriz traspuesta de la imagen original. Esto se debe a que tras varios análisis el acceso a columnas es mucho mas rápido que al de filas, por lo que buscar la columna a la matriz traspuesta podría considerarse equivalente a la búsqueda de la fila de la matriz original. En las figuras 12 y 12.1 se muestra como los resultados de la búsqueda de un seam son idénticos.

-Borrado de objetos, la idea principal de este experimento es seleccionar pixeles. Para este problema he considerado que el color rojo (0,0,255) son los pixeles candidatos, esto acarrea un problema y es que el algoritmo borrara todos aquellos pixeles con ese valor. Obviamente la mejor solución sería seleccionar “a mano” aquellos pixeles que se deseen borrar, pero por comodidad he valorado que el pintar sobre la imagen los pixeles que deben eliminarse es la mejor opción.



Figura 12: Búsqueda de un seam horizontal.

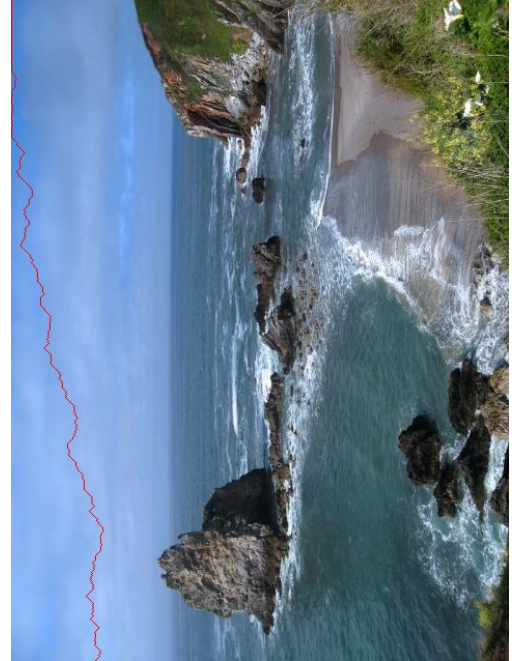


Figura 12.1: Búsqueda de un seam vertical a la matriz traspuesta.

-He implementado la forma optima de buscar los seams pero no se ejecutara en main puesto que tarda demasiado tiempo, la función se llama:

```
void reducirImagenesOptimalOrder(Mat &im, const int filas, const int columnas)
```

Para una imagen de 600x400 donde se pretende borrar 100 columnas y 100 filas puede llegar a tardar media hora, esto es debido a que tiene que crear $100 \times 100 = 10000$ matrices para comprobar cual es la que menos energía consume. En lugar de esto pensé otra forma, en vez de buscar el optimo global, obtener el optimo local; el resultado es peor pero se logra una mayor rapidez Figura 13. El proceso sería obtener la energía de una columna y fila la que posea menor coste se borrara, esto no analiza todas las posibilidades pero si la mejor de una forma local. También para mejorar la de búsqueda de un orden optimo global se podría implementar mediante hebras, esto seguro que abarataría el coste de tiempo de forma considerable.



Figura 13.1: Búsqueda optima local.



Figura 13.2: Búsqueda optima global.

En las imágenes se puede ver como la calidad es bastante parecida y habiendo conseguido esto en un tiempo bastante bueno.

Conclusión

A mi parecer es una buena alternativa a la redimensión de las imágenes tradicional donde en algunas ocasiones y sobre todo si la reducción no es equilibrada entre columnas y filas se consigue un pésimo resultado sin embargo utilizando Seam Carving se puede llegar a obtener unos resultados bastante satisfactorios para cualquier persona que desee una mínima pérdida de la calidad de la imagen.

He de decir que los resultados no son siempre buenos ya que elimina seams con la energía mínima y a veces este elimina regiones oscuras pero que si son importantes, igualmente se puede reducir o aumentar utilizando ambos métodos y comprobar cual posee el mejor resultados la elección es de cada uno.



Figura 14.1: Redimensión método tradicional.
Borradas 200 filas.



Figura 14.2: Redimensión método seam carving.
Borradas 200 filas.

Referencias

<http://www.win.tue.nl/~wstahw/edu/2IV05/seamcarving.pdf>