Facultad de Ingeniería de la
Universidad de la República

Herramientas de programación para el
procesamiento de señales

Curso 2013

Difusión Anisotrópica

Autor:
Matías Tailanián

Tutor:
Juan Cardelino

Índice general

Introducción

Una buena forma de modelar una imágen es suponer que es la composición de la imágen limpia con ruido de media nula, independiente de la imagen e independiente entre píxeles. La forma más evidente para hacer *denoising* consiste simplemente en aplicar un filtro pasabajos. El problema inmediato que se presenta es la elección del parámetro σ del filtro. Si se utiliza un σ alto se realizará un suavizado muy fuerte sobre la imágen, corriendo el riesgo de destruir las estructuras interesantes de la imagen, mientras que utilizando un σ muy chico puede no eliminarse el satisfactoriamente ruido. El siguiente paso razonable es aplicar un banco de filtros, de resolución decreciente (*sigma* creciente). En definitiva, aplicar sucesivamente este banco de filtros a una imagen equivale a resolver la ecuación del calor, o *heat flow*:

$$I_t(x, y, t) = \nabla^2 I(x, y, t) \tag{1.1}$$

La analogía con la transferencia de calor explica perfectamente el fenómeno que produce resolver esta ecuación para *denoising* de una imágen. Tal como pasa con la uniforimización del calor, llegando a una temperatura de equilibrio, en la imágen se suavizan las regiones, eliminando el ruido. Si se dejara correr demasiado tiempo el algoritmo, llegaríamos a obtener una imagen plana, completamente.

Entonces, siguiendo este razonamiento es natural pensar en un algoritmo que, basado en un principio similar a la ecuación del calor, uniformice las regiones, pero no difunda los bordes, conservando así la estructura de la imágen. Para ello puede utilizarse un coeficiente de conducción variable en la ecuación de calor:

$$I_t(x, y, t) = \nabla \cdot (g\ \nabla I(x, y, t)) \tag{1.2}$$

Para lograr regularizar la imágen dentro de las regiones y no a través de los bordes basta con encontrar un funcional g que valga 1 dentro de las regiones y 0 en los bordes. Para ello debemos encontrar un buen descriptor de bordes, por ejemplo dependiendo del gradiente de la imágen. En este caso, la ecuación 1.2 se transforma en la siguiente:

$$I_t = \nabla \cdot (g(|\nabla I|)\nabla I) \tag{1.3}$$

El funcional g debe elegirse de modo que tome valor nulo donde el gradiente es grande, y 1 cuando es pequeño, tal como se suguiere en [1]:

$$\begin{cases} \lim_{x\to 0} & g(x) = 1 \\ \lim_{x\to \infty} & g(x) = 0 \end{cases}$$

Se consideran 2 diferentes elecciones para el funcional g:

- Detector de bordes **Lorentziano**

$$g(|\nabla I|) = \frac{1}{1 + \frac{|\nabla I|^2}{2\sigma^2}}$$

- Detector de bordes de **Leclerc**

$$g(|\nabla I|) = e^{-\frac{|\nabla I|^2}{2\sigma^2}}$$

En la figura 1.1 se muestran los dos detectores de bordes para $\sigma = 0.2$.

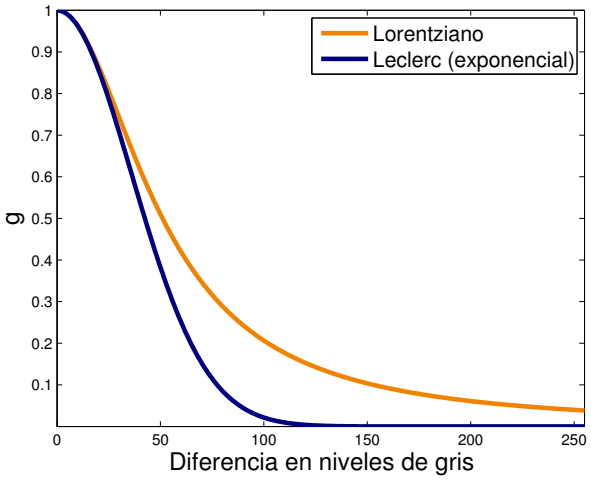

Figura 1.1: Detectores de bordes

Enfoque de cálculo variacional

Resulta muy interesante cómo es posible llegar a obtener la ecuación de la Difusión Anisotrópica, desde el punto de vista del cálculo variacional.

Las ecuaciones de **Euler-Lagrange** [2] aseguran que para minimizar la función:

$$\min_I \int_\Omega \rho(|\nabla I|) d\Omega$$

se debe resolver la ecuación:

$$\frac{\partial I(x,y,t)}{\partial t} = div\left(\rho' \frac{\nabla I}{|\nabla I|}\right)$$

En particular, eliguiendo $\rho(a) = a$ obtenemos la cuación de *Total Variation*, donde la función a minimizar es

$$\int_\Omega |\nabla I| d\Omega$$

lo cual se calcula resolviendo

$$I_t = div\left(\frac{\nabla I}{|\nabla I|}\right)$$

Como el argumento de la divergencia es, de hecho, una parametrización normalizada de una curva, es decir el *Arc Lenght*, la dinámica de evoulución de la imágen es en realidad de acuerdo a la curvatura de sus curvas de nivel, ya que dicha expresión coincide exactamente con la definición de curvatura:

$$\kappa = div\left(\frac{\nabla I}{|\nabla I|}\right)$$

Es interesante llegar al mismo resultado para la Difusión Anisotrópica estudiándolo como un problema de cálculo variacional, y a su vez concluir que la solución del problema es la evulución de las curvas de nivel de la imágen según su curvatura, llegando a obtener un fuerte vínculo entre *Curve Evolution*, *Level Sets* y *Cálculo Variacional*

Estudio de costo computacional

Sean:

- **N**: Ancho de la imágen, recorrido con el subíndice j
- **M**: Alto de la imágen, recorrido con el subíndice i

Para cáda píxel se debe calcular la diferencia con sus 4 vecinos. Sean $d^N_{i,j}$, $d^S_{i,j}$, $d^E_{i,j}$, $d^W_{i,j}$ las diferencias del píxel i, j hacia arriba, abajo, derecha e izquierda respectivamente, con la notación haciendo referencia a los puntos cardinales. Entonces:

- $d^N_{i,j} = I_{i-1,j} - I_{i,j}$; $\quad d^S_{i,j} = I_{i+1,j} - I_{i,j}$
- $d^E_{i,j} = I_{i,j+1} - I_{i,j}$; $\quad d^W_{i,j} = I_{i,j-1} - I_{i,j}$

Además se debe evaluar la función g en las diferencias anteriores (los gradientes):

- $g^N_{i,j} = \frac{1}{1+\frac{(d^N_{i,j})^2}{2\sigma^2}}$; $\quad g^S_{i,j} = \frac{1}{1+\frac{(d^S_{i,j})^2}{2\sigma^2}}$; $\quad g^E_{i,j} = \frac{1}{1+\frac{(d^E_{i,j})^2}{2\sigma^2}}$; $\quad g^W_{i,j} = \frac{1}{1+\frac{(d^W_{i,j})^2}{2\sigma^2}}$

Por último, se calcula el valor del píxel i, j en el tiempo siguiente $(t+1)$ como:

$$I^{t+1}_{i,j} = I^t_{i,j} + \lambda \left(g^N_{i,j} d^N_{i,j} + g^S_{i,j} d^S_{i,j} + g^E_{i,j} d^E_{i,j} + g^W_{i,j} d^W_{i,j} \right)$$

Para el cálculo de $d_{i,j}$ se necesita 1 operación, mientras que precalculando el valor de $2\sigma^2$, el cálculo de $g_{i,j}$ lleva 3 operaciones. La actualización de $I_{i,j}$ lleva 9 operaciones. Resumiendo, para un píxel, para una iteración se necesitan:

$$4 \times (1 + 3) + 9 \;=\; 25 \text{ operaciones}$$

Por lo tanto, se necesitan:

$$25 \times M \times N \text{operaciones por iteración}$$

A modo de ejemplo, para una imágen de 1024×768 que se itera unas 50 veces se necesitan casi mil millones de operaciones (unas 983.040.000 operaciones).

Valgrind

```
$ valgrind --leak-check=full ./difusionAnisotropica ../../pics/botes.ppm salida
==21696== Memcheck, a memory error detector
==21696== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==21696== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==21696== Command: ./difusionAnisotropica ../../pics/botes.ppm salida
==21696==
==21696==
==21696== HEAP SUMMARY:
==21696==     in use at exit: 0 bytes in 0 blocks
==21696==   total heap usage: 14 allocs, 14 frees, 33,817,712 bytes allocated
==21696==
==21696== All heap blocks were freed -- no leaks are possible
==21696==
==21696== For counts of detected and suppressed errors, rerun with: -v
==21696== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Profiler

% time	cum. s	self s	calls	self ms/call	total ms/call	name
92.69	12.17	12.17	158515200	0.00	0.00	difusionIteration
6.26	12.99	0.82	3	0.27	4.33	difusionAnisotropicaGrises
0.08	13.00	0.01	1	0.01	13.00	difusionAnisotropica
0.00	13.00	0.00	6	0.00	0.00	destruir_imagen
0.00	13.00	0.00	6	0.00	0.00	inicializar_imagen
0.00	13.00	0.00	1	0.00	0.00	duplicar_imagen
0.00	13.00	0.00	1	0.00	0.00	escribir_datos_p6
0.00	13.00	0.00	1	0.00	0.00	escribir_imagen
0.00	13.00	0.00	1	0.00	0.00	leer_datos_p6
0.00	13.00	0.00	1	0.00	0.00	leer_encabezado
0.00	13.00	0.00	1	0.00	0.00	leer_imagen

Resultados

(a) Imágen original (b) Salida del algoritmo

Figura 6.1: Salida para imagen vectorial

Conclusiones

Bibliografía

[1] Pietro Perona, Takahiro Shiota, and Jitendra Malik, *Anisotropic diffusion*, Geometry-driven diffusion in computer vision, pp. 73–92, Springer, 1994.

[2] Richard Courant and David Hilbert, *Methods of mathematical physics*, CUP Archive, Volume 1, pp. 184–185, 1966.