Algoritmos para Processamento de Áudio, Imagem e Vídeo

Notas de aula sobre processamento de video

Estas notas de aula correspondem a uma parte da matéria do curso que não está no livro "Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing" de S. Allen Broughton e Kurt M. Bryan. Mais informações sobre processamento de vídeo podem ser obtidas no livro "Multidimensional Signal, Image and Video Processing and Coding" de John W. Woods, especialmente nos capítulos 9 e 10.

1 Introdução: Representações de sinais de vídeo

Trataremos neste texto apenas de vídeos digitais de duração finita[1], que podem ser representados por funções $f : \mathbf{N_x} \times \mathbf{N_y} \times \mathbf{N_t} \longrightarrow \mathbb{C}$, onde $\mathbf{N_x} = \{0, \ldots, N_x - 1\}$, $\mathbf{N_y} = \{0, \ldots, N_y - 1\}$ e $\mathbf{N_t} = \{0, \ldots, N_t - 1\}$. Neste contexto, $f(x, y, t)$ representa o pixel (x, y) no instante t, e quando necessário nos referiremos ao quadro no instante t como $f_t(x, y)$ e à evolução temporal do conteúdo do pixel (x, y) como $f_{x,y}(t)$.

1.1 Transformada de Fourier Discreta (DFT) 3D

Podemos estender a definição da transformada de Fourier 1D e 2D de forma análoga para o caso tridimensional, através da expressão

$$F(k, l, s) = \langle f, E_{k,l,s} \rangle = \sum_{x \in \mathbf{N_x}} \sum_{y \in \mathbf{N_y}} \sum_{t \in \mathbf{N_t}} f(x, y, t) e^{-i \cdot 2\pi \left(k \frac{x}{N_x} + l \frac{y}{N_y} + s \frac{t}{N_t} \right)},$$

onde k, l e s são as frequências de oscilação horizontal, vertical e temporal, associadas à função básica exponencial

$$E_{k,l,s}(x, y, t) = e^{i \cdot 2\pi \left(k \frac{x}{N_x} + l \frac{y}{N_y} + s \frac{t}{N_t} \right)}.$$

Observe que

$$E_{k,l,s}(x, y, t) = E_{k,N_x}(x) E_{l,N_y}(y) E_{s,N_t}(t)$$

onde

$$E_{p,N_z}(z) = e^{i \cdot 2\pi p \frac{z}{N_z}}$$

é a p-ésima exponencial complexa unidimensional em $\mathbb{C}^{N_z}$. Esta propriedade, denominada de separabilidade, permite a interpretação da transformada de Fourier em 3 ou mais variáveis como uma composição de aplicações da transformada de Fourier unidimensional em cada variável. Seja $\mathcal{F}^j$ o operador associado à transformada de Fourier da j-ésima variável de uma função, ou seja, $\mathcal{F}[f]$ é a função definida por

$$\mathcal{F}[f](\ldots, \omega_j, \ldots) = \sum_{x_j \in \mathbf{N_j}} f(\ldots, x_j, \ldots) \overline{E_{\omega_j, N_j}(x_j)}$$

[1] Uma discussão sobre amostragem de vídeos analógicos e resultados teóricos pode ser encontrada na seção 9.2 do livro do Woods.

onde x_j é a j-ésima variável de f, com valores em $\mathbf{N_j} = \{0, \ldots, N_j - 1\}$, e ω_j é a frequência relacionada à variável x_j. Com esta definição podemos escrever

$$\begin{aligned}
F(k,l,s) &= \sum_{t\in\mathbf{N_t}} \sum_{y\in\mathbf{N_y}} \sum_{x\in\mathbf{N_x}} f(x,y,t)\overline{E_{k,N_x}(x)E_{l,N_y}(y)E_{s,N_t}(t)} \\
&= \sum_{t\in\mathbf{N_t}} \sum_{y\in\mathbf{N_y}} \underbrace{\left[\sum_{x\in\mathbf{N_x}} f(x,y,t)\overline{E_{k,N_x}(x)}\right]} \overline{E_{l,N_y}(y)E_{s,N_t}(t)} \\
&= \sum_{t\in\mathbf{N_t}} \underbrace{\left[\sum_{y\in\mathbf{N_y}} \mathcal{F}^1[f](k,y,t)\overline{E_{l,N_y}(y)}\right]} \overline{E_{s,N_t}(t)} \\
&= \sum_{t\in\mathbf{N_t}} \mathcal{F}^2 \circ \mathcal{F}^1[f](k,l,t)\overline{E_{s,N_t}(t)} \\
&= \mathcal{F}^3 \circ \mathcal{F}^2 \circ \mathcal{F}^1[f](k,l,s).
\end{aligned}$$

Analogamente (permutando as somatórias e os termos $E_{p,N_z}(z)$), podemos provar que

$$\begin{aligned}
F(k,l,s) &= \mathcal{F}^1 \circ \mathcal{F}^2 \circ \mathcal{F}^3[f](k,l,s) \\
&= \mathcal{F}^2 \circ \mathcal{F}^1 \circ \mathcal{F}^3[f](k,l,s) \\
&= \cdots
\end{aligned}$$

o que mostra que a transformada de Fourier pode ser calculada uma variável de cada vez, e na ordem que se quiser, resultado este que se estende diretamente para mais de 3 variáveis.

Exemplo: Poderíamos pensar na transformada de Fourier em vídeos recaindo em alguma das transformadas estudadas anteriormente. Por exemplo, poderíamos transformar cada função $f_{x,y}(t)$ correspondente à evolução temporal do pixel (x, y), obtendo uma transformada

$$F_{x,y}(s) = \sum_{t\in\mathbf{N_t}} f_{x,y}(t)e^{-i\cdot 2\pi s\frac{t}{N_t}},$$

e colecionando todas estas funções em uma única função $\hat{F}(x, y, s)$, que pode posteriormente ser transformada nas variáveis x e y:

$$F(k,l,s) = \sum_{x\in\mathbf{N_x}} \sum_{y\in\mathbf{N_y}} \hat{F}(x,y,s)e^{-i\cdot 2\pi\left(k\frac{x}{N_x}+l\frac{y}{N_y}\right)}.$$

Uma outra possibilidade seria tomar as transformadas 2D dos quadros $f_t(x, y)$, obtendo funções

$$F_t(k,l) = \sum_{x\in\mathbf{N_x}} \sum_{y\in\mathbf{N_y}} f_t(x,y)e^{-i\cdot 2\pi\left(k\frac{x}{N_x}+l\frac{y}{N_y}\right)}$$

para cada instante t, e colecionar estas funções em uma única função $\tilde{F}(k, l, t)$ para posteriormente transformá-la em relação à variável t:

$$F(k,l,s) = \sum_{t\in\mathbf{N_t}} \tilde{F}(k,l,t)e^{-i\cdot 2\pi s\frac{t}{N_t}}.$$

Estas possibilidades não são exaustivas, e as transformadas "parciais" como as funções $\hat{F}(x, y, s)$ e $\tilde{F}(k, l, t)$ acima (ou seja, transformadas em relação apenas a uma parte das variáveis) podem ser consideradas como representações alternativas em relação à forma original $f(x, y, t)$ do vídeo ou sua transformada completa $F(k, l, s)$.

A transformada inversa de Fourier terá a expressão

$$f(x,y,t) = \sum_{k\in\mathbf{N_x}} \sum_{l\in\mathbf{N_y}} \sum_{s\in\mathbf{N_t}} F(k,l,s) e^{i\cdot 2\pi\left(k\frac{x}{N_x} + l\frac{y}{N_y} + s\frac{t}{N_t}\right)},$$

e denotando por $\mathcal{F}^{-j}$ o operador associado à transformada inversa de Fourier em relação à j-ésima variável, ou seja,

$$\mathcal{F}^{-j}[F](\ldots,x_j,\ldots) = \sum_{\omega_j\in\mathbf{N_j}} F(\ldots,\omega_j,\ldots) E_{x_j,N_j}(\omega_j)$$

podemos provar de maneira análoga à transformada direta que

$$\begin{aligned} f &= \mathcal{F}^{-1}\circ\mathcal{F}^{-2}\circ\mathcal{F}^{-3}[F] \\ &= \mathcal{F}^{-2}\circ\mathcal{F}^{-1}\circ\mathcal{F}^{-3}[F] \\ &= \mathcal{F}^{-3}\circ\mathcal{F}^{-2}\circ\mathcal{F}^{-1}[F] \\ &= \cdots \end{aligned}$$

Também é fácil ver que as transformadas parciais possuem formas inversas, por exemplo

$$f = \mathcal{F}^{-3}\left[\hat{F}\right] = \mathcal{F}^{-1}\circ\mathcal{F}^{-2}\left[\tilde{F}\right].$$

1.2 Transformada Discreta do Cosseno (DCT) 3D

Podemos definir uma transformada discreta do cosseno seguindo os mesmos passos da transformada unidimensional, isto é, tomando a transformada de Fourier da reflexão da função f em cada um de seus eixos:

$$\tilde{f}(x,y,t) = f(r(x,N_x), r(y,N_y), r(t,N_t)),$$

onde

$$r(z,N_z) = \min\{z, 2N_z - z - 1\}$$

para $z = 0,1,\ldots,2N_z-1$ (esta expressão gera os valores $0,1,\ldots,N_z-1,N_z-1,N_z-2,\ldots,0$), e posteriormente recodificando o resultado em $N_x\times N_y\times N_t$ coeficientes, para chegarmos à expressão:

$$C(k,l,s) = \alpha_{kls} \sum_{x\in\mathbf{N_x}} \sum_{y\in\mathbf{N_y}} \sum_{t\in\mathbf{N_t}} f(x,y,t) \cos\left(\pi k\left(\frac{x+\frac{1}{2}}{N_x}\right)\right) \cos\left(\pi l\left(\frac{y+\frac{1}{2}}{N_y}\right)\right) \cos\left(\pi s\left(\frac{t+\frac{1}{2}}{N_t}\right)\right),$$

onde $\alpha_{kls} = \alpha_{k,N_x}\alpha_{l,N_y}\alpha_{s,N_t}$ e

$$\alpha_{p,N_z} = \begin{cases} \sqrt{\frac{1}{N_z}} & \text{se } p = 0 \\ \sqrt{\frac{2}{N_z}} & \text{se } p \neq 0 \end{cases}$$

Esta expressão possui o mesmo tipo de separabilidade da DFT, e portanto a DCT também pode ser calculada independentemente em cada dimensão, ou seja, podemos transformar unidimensionalmente a função em relação à variável x, depois transformar o resultado unidimensionalmente em relação a y e finalmente em relação a t, ou ainda seguindo qualquer permutação das variáveis.

A inversa da DCT (IDCT) 3D é dada por

$$f(x,y,t) = \alpha_{xyt} \sum_{k\in\mathbf{N_x}} \sum_{l\in\mathbf{N_y}} \sum_{s\in\mathbf{N_t}} C(k,l,s) \cos\left(\pi k\left(\frac{x+\frac{1}{2}}{N_x}\right)\right) \cos\left(\pi l\left(\frac{y+\frac{1}{2}}{N_y}\right)\right) \cos\left(\pi s\left(\frac{t+\frac{1}{2}}{N_t}\right)\right),$$

onde $\alpha_{xyt} = \alpha_{x,N_x}\alpha_{y,N_y}\alpha_{t,N_t}$, expressão essa que possui a mesma propriedade de separabilidade e recai em IDCTs unidimensionais nas variáveis k, l e s.

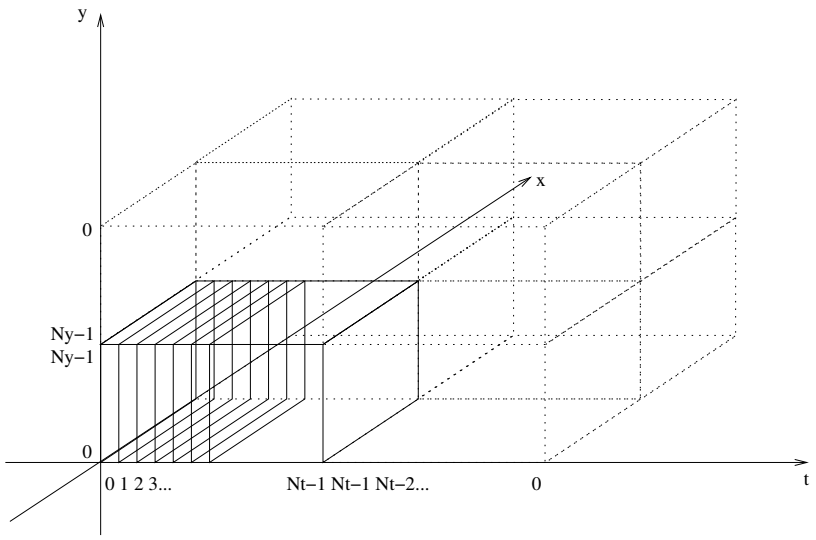

Figura 1: Espelhamento do vídeo original em relação às variáveis x, y e t (7 blocos espelhados + 1 original).

1.3 Transformada Discreta de Wavelets (DWT) 3D

Podemos definir facilmente a DWT 3D a partir de qualquer banco de filtros em $l^2(\mathbb{Z}\ )$ que possua a propriedade da reconstrução perfeita, usando a propriedade da separabilidade como no caso 2D, aplicando-se a DWT em cada uma das dimensões, independentemente da ordem:

$$DWT[f] = DWT^1 \circ DWT^2 \circ DWT^3[f] = DWT^3 \circ DWT^1 \circ DWT^2[f] = \cdots$$

(o jeito mais fácil de provar isso é pela transformada z). A inversa da DWT 3D também pode ser calculada em qualquer ordem.

Lembrando do mapeamento

$$x \longrightarrow \begin{bmatrix} X_l \\ X_r \end{bmatrix}$$

onde X_l são os coeficientes de aproximação e X_h os coeficientes de detalhe, teremos a mesma separação na aplicação da DWT^3 (dimensão temporal): O bloco de aproximação será uma versão suavizada (pelo filtro passa-baixas) e acelerada (pela sub-amostragem) do vídeo original. O bloco de detalhes contém as frequências mais altas (no caso do banco de filtros de Haar ele contém a diferença entre quadros sucessivos), o que juntamente com o bloco de aproximação permitirá reconstruir exatamente os quadros originais.

Considerando-se a interpretação da DWT 2D em cada quadro

versão suavizada	detalhes verticais
detalhes horizontais	detalhes diagonais

após a DWT 3D completa teremos a seguinte disposição da informação onde os 8 blocos (de tamanho $\frac{N_x}{2} \times \frac{N_y}{2} \times \frac{N_t}{2}$) possuem a seguinte interpretação:

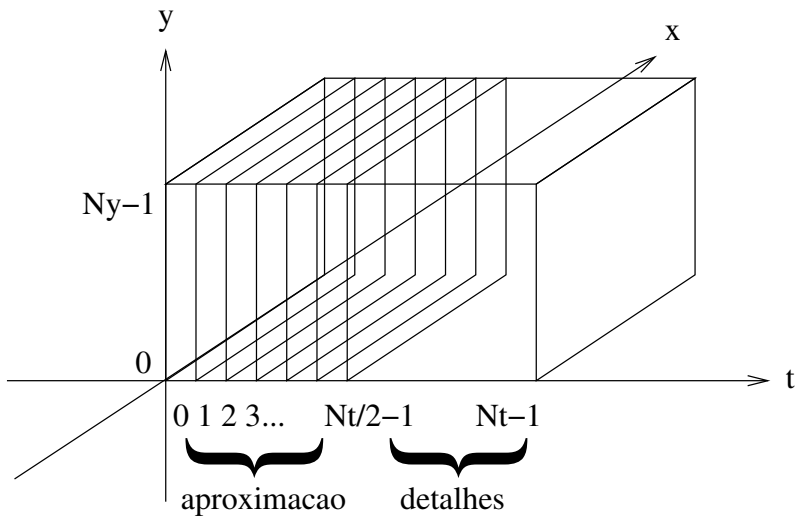

Figura 2: Blocos de aproximação temporal e detalhe temporal na $DWT^3[f]$.

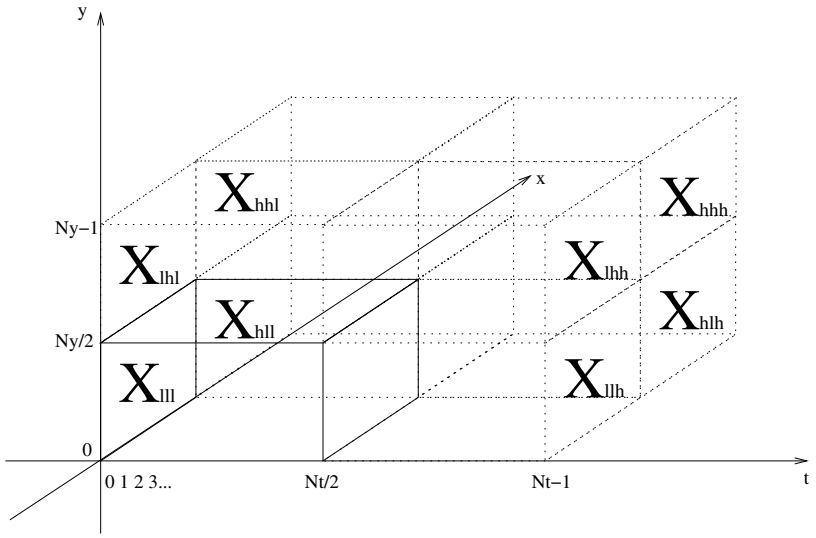

Figura 3: Blocos de aproximação e detalhes da DWT 3D.

X_{lll} = coeficientes de aproximação (nas 3 dimensões)
X_{llh} = aproximação espacial, detalhes temporais
X_{lhl} = aproximação horizontal-temporal, detalhes horizontais
X_{lhh} = aproximação horizontal, detalhes horizontais/temporais
X_{hll} = aproximação vertical-temporal, detalhes verticais
X_{hlh} = aproximação vertical, detalhes verticais/temporais
X_{hhl} = aproximação temporal, detalhes espaciais
X_{hhh} = detalhes espaço-temporais

Podemos estender a DWT 3D para a versão iterada em K etapas, aplicando a DWT ao bloco X_{lll} e obtendo 8 novos blocos $X_{lll,lll}, X_{lll,llh}, \ldots, X_{lll,hhh}$ de tamanho $\frac{N_x}{4} \times \frac{N_y}{4} \times \frac{N_t}{4}$, e depois transformando o bloco $X_{lll,lll}$ obtendo 8 blocos de tamanho $\frac{N_x}{8} \times \frac{N_y}{8} \times \frac{N_t}{8}$, e assim por diante (enquanto todas as dimensões forem divisíveis por 2).

2 Convolução e Filtros 3D

Dados dois vídeos $f, h : \mathbf{N_x} \times \mathbf{N_y} \times \mathbf{N_t} \longrightarrow \mathbb{C}$ podemos definir a convolução $g = f * h : \mathbf{N_x} \times \mathbf{N_y} \times \mathbf{N_t} \longrightarrow \mathbb{C}$ através da expressão

$$g(x,y,t) = \sum_{p \in \mathbf{N_x}} \sum_{q \in \mathbf{N_y}} \sum_{r \in \mathbf{N_t}} h(p,q,r) f(x-p, y-q, t-r).$$

Aqui deve-se interpretar todas as funções como periódicas nas 3 variáveis, ou seja,

$$f(x,y,t) = f(x \bmod N_x, y \bmod N_y, t \bmod N_t),\ \forall x,y,t \in \mathbb{Z}\ .$$

Pode-se provar um teorema da convolução 3D de maneira análoga aos casos 1D e 2D, de tal forma que se $F(k,l,s)$, $G(k,l,s)$ e $H(k,l,s)$ são as transformadas de Fourier de $f(x,y,t)$, $g(x,y,t)$ e $h(x,y,t)$, respectivamente, então

$$G(k,l,s) = H(k,l,s)F(k,l,s),\ \forall k \in \mathbf{N_x}, l \in \mathbf{N_y}, s \in \mathbf{N_t}.$$

No caso particular em que h é uma função separável, ou seja,

$$h(x,y,t) = h_1(x)h_2(y)h_3(t),$$

então podemos utilizar essa separabilidade para computar a convolução de forma mais eficiente. Denotando por $*^{(j)}$ a operação de convolução em relação à j-ésima variável, ou seja, $f *^{(j)} h$ é a função com valores

$$[f *^{(j)} h](\ldots, x_j, \ldots) = \sum_{y_j \in \mathbf{N_j}} h(\ldots, y_j, \ldots) f(\ldots, x_j - y_j, \ldots),$$

teremos

$$\begin{aligned}
g(x,y,t) &= \sum_{r \in \mathbf{N_t}} \sum_{q \in \mathbf{N_y}} \sum_{p \in \mathbf{N_x}} h_1(p)h_2(q)h_3(r) f(x-p, y-q, t-r) \\
&= \sum_{r \in \mathbf{N_t}} h_3(r) \sum_{q \in \mathbf{N_y}} h_2(q) \left[\sum_{p \in \mathbf{N_x}} h_1(p) f(x-p, y-q, t-r) \right] \\
&= \sum_{r \in \mathbf{N_t}} h_3(r) \left[\sum_{q \in \mathbf{N_y}} h_2(q) [f *^{(1)} h_1](x, y-q, t-r) \right] \\
&= \sum_{r \in \mathbf{N_t}} h_3(r) \left[[f *^{(1)} h_1] *^{(2)} h_2 \right](x, y, t-r) \\
&= \left[\left[[f *^{(1)} h_1] *^{(2)} h_2 \right] *^{(3)} h_3 \right](x,y,t).
\end{aligned}$$

Considere que cada componente h_j do filtro possui $|h_j|$ componentes não-nulas, de tal forma que h possui $|h| = |h_1| \cdot |h_2| \cdot |h_3|$ componentes não-nulas. Obter a convolução pela definição original depende de computar somatórios de tamanho $|h|$ para cada entrada do vídeo, com um custo total de $\mathcal{O}(|h| \cdot N_x N_y N_t) = \mathcal{O}(|h_1| \cdot |h_2| \cdot |h_3| \cdot N_x N_y N_t)$. Por outro lado, computar $f *^{(1)} h_1$ tem custo $\mathcal{O}(|h_1| \cdot N_x N_y N_t)$; de posse deste resultado, computar $[f *^{(1)} h_1] *^{(2)} h_2$ terá custo $\mathcal{O}(|h_2| \cdot N_x N_y N_t)$ e finalmente computar $\left[[f *^{(1)} h_1] *^{(2)} h_2 \right] *^{(3)} h_3$ terá custo $\mathcal{O}(|h_3| \cdot N_x N_y N_t)$, representando um custo total de $\mathcal{O}\left((|h_1| + |h_2| + |h_3|) \cdot N_x N_y N_t\right)$.

Exemplo: Considere o filtro da média 3D com 27 pontos, definido por

$$h(x,y,t) = \begin{cases} \frac{1}{27} & x,y,t \in \{-1,0,+1\} \\ 0 & \text{caso contrário} \end{cases}$$

Observe que $h(x,y,t) = h(x)h(y)h(t)$ onde

$$h(z) = \begin{cases} \frac{1}{3} & z \in \{-1,0,+1\} \\ 0 & \text{caso contrário} \end{cases}$$

As convoluções tridimensionais têm custo de 27 multiplicações e 26 somas por pixel por quadro, enquanto as convoluções unidimensionais têm custo de 3 multiplicações e 2 somas por pixel por quadro, o que na computação por etapas representa 9 multiplicações e 8 somas por pixel por quadro. Esta computação em etapas corresponde a aplicar o filtro da média apenas em relação à variável x, em todo o vídeo, para em seguida aplicar o filtro da média em relação a y (em todo o vídeo) e posteriormente o filtro da média em relação a t.

2.1 Filtros temporais puros

Em alguns casos podemos estar interessados em processar vídeos combinando a informação do pixel (x,y) em vários quadros sucessivos. Aplicações disso incluem o aprimoramento de imagens congeladas (por exemplo, na astrofotografia a partir de registros em vídeo), ou para completar quadros incompletos (por exemplo, frames entrelaçados). Nestes casos, pode ser interessante considerar convoluções da forma

$$g(x,y,t) = \sum_{r \in N_t} h_r f(x,y,t-r)$$

onde a resposta impulsiva do filtro é

$$h(p,q,r) = \begin{cases} h_r & \text{se } p = q = 0 \\ 0 & \text{caso contrário} \end{cases}$$

Por exemplo, um suavizador temporal de L pontos pode ser caracterizado por $h(p,q,r) = \delta(p)\delta(q)I_{[0,L)}(r)$, onde $\delta(\cdot)$ é o delta de Dirac discreto e $I_{[0,N)}(r) = \sum_{s=0}^{L-1} \delta(r-s)$ é a função indicadora do conjunto $\{0,1,\ldots,L-1\}$. Nesse caso $h_r = \frac{1}{L}, r = 0,\ldots,L-1$ na expressão do filtro temporal puro acima:

$$g(x,y,t) = \frac{1}{N} \sum_{r=0}^{L-1} f(x,y,t-r).$$

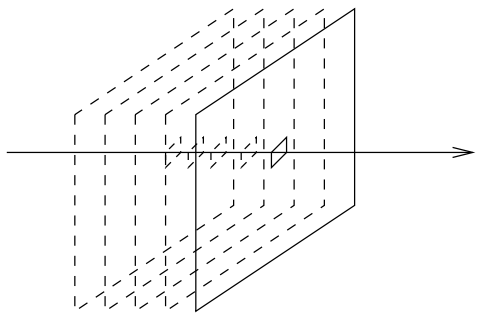

Figura 4: Filtragem temporal pura.

2.2 Filtros intra-frame

Outras vezes pode ser útil aplicar um tratamento tradicional de imagens, como filtros de suavização ou realce, aos quadros de um vídeo, especialmente quando há pouca interdependência entre os quadros. Neste caso consideraremos convoluções da forma

$$g(x,y,t) = \sum_{p\in N_x} \sum_{q\in N_y} h_{p,q} f(x-p, y-q, t),$$

que correspondem a um filtro com resposta impulsiva

$$h(p,q,t) = h_{p,q}\delta(t).$$

2.3 Filtros inter-frame

O caso geral do processamento de vídeos envolve a combinação de pixels do mesmo quadro e de quadros adjacentes, na expressão da convolução geral:

$$g(x,y,t) = \sum_{p\in \mathbf{N_x}} \sum_{q\in \mathbf{N_y}} \sum_{r\in \mathbf{N_t}} h_{p,q,r} f(x-p, y-q, t-r).$$

Neste tipo de processamento é comum utilizar-se algum esquema de compensação de movimento, onde os pixels $(x-p, y-q)$ não são estáticos como na fórmula acima, mas dependem do tempo e da dinâmica dos objetos em cena, sendo substituídos por expressões do tipo $(x^{(r)}(p), y^{(r)}(q))$, interpretadas como a resposta à pergunta "onde estava o pixel $(x-p, y-q)$ r quadros atrás?".

2.4 Transformada z 3D

Podemos definir a transformada z 3D seguindo os mesmos passos utilizados na transformada de Fourier, ou seja, a partir da expressão

$$F(z_1, z_2, z_3) = \sum_{x\in \mathbf{N_x}} \sum_{y\in \mathbf{N_y}} \sum_{t\in \mathbf{N_t}} f(x,y,t) z_1^{-x} z_2^{-y} z_3^{-t}.$$

assim como nos casos 1D e 2D, essa expressão se reduz à transformada de Fourier quando $z_1 = e^{i2\pi k/N_x}$, $z_2 = e^{i2\pi l/N_y}$ e $z_3 = e^{i2\pi s/N_t}$.

A transformada z pode ser utilizada para caracterizar o comportamento de filtros, como nos casos 1D e 2D. Por exemplo, uma expressão geral para um filtro recursivo (IIR) com equação finita em 3 dimensões teria a forma

$$g(x,y,t) = \sum_{p\in \mathbf{N_x}} \sum_{q\in \mathbf{N_y}} \sum_{r\in \mathbf{N_t}} \left[a(p,q,r) f(x-p, y-q, t-r) - b(p,q,r) g(x-p, y-q, t-r)\right],$$

para o qual deve valer (sob certas condições) a relação entre as transformadas z

$$\frac{G(z_1, z_2, z_3)}{F(z_1, z_2, z_3)} = \frac{A(z_1, z_2, z_3)}{1 + B(z_1, z_2, z_3)}$$

e entre as transformadas de Fourier

$$\frac{G(k,l,s)}{F(k,l,s)} = \frac{A(k,l,s)}{1 + B(k,l,s)}, \ \forall k,l,s$$

supondo que os denominadores não se anulem.

Observe que filtros FIR correspondem ao caso particular em que b=0 e B=0.

3 Estimação de movimento e compensação de movimento

As estratégias denominadas estimação de movimento e compensação de movimento visam fazer uso da alta correlação (típica) entre quadros sucessivos em um vídeo. Através da estimação de movimento podemos encontrar blocos muito parecidos nos quadros t e $t-1$, e com isso economizar na codificação, guardando apenas o deslocamento do bloco e a diferença ou resíduo entre as duas versões do bloco, que tipicamente tem valores numéricos baixos que demandam menos bits em suas representações. No caso de um processamento temporal ou espaço-temporal, o deslocamento dos objetos pode ser compensado através da adaptação da equação do filtro aos deslocamentos correspondentes.

Exemplo: eliminação de ruído nos quadros.

Sabemos que a eliminação de ruído em imagens pode ser feita por filtros de suavização espacial, o que se traduz no contexto de vídeos em filtragem intra-frame. Um efeito colateral desagradável é o fato das fronteiras dos objetos perderem nitidez (ficarem borrados). Se tivéssemos várias imagens iguais afetadas por ruídos independentes, seria muito mais natural combinar os pixels correspondentes, fazendo uma filtragem temporal pura, por exemplo através de um filtro de média de L pontos. Quando as imagens correspondem a cópias deslocadas de uma mesma cena, poderíamos adaptar este processamento se conhecêssemos as trajetórias $(x^{(t-r)}, y^{(t-r)})$ de cada pixel (x, y) da imagem no instante t:

$$g(x, y, t) = \frac{1}{L} \sum_{r=0}^{L-1} f(x^{(t-r)}, y^{(t-r)}, t - r).$$

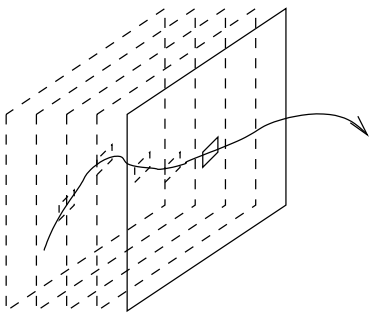

Figura 5: Filtragem temporal com compensação de movimento.

Outro exemplo: o mesmo tipo de compensação é útil em todos os casos de processamento inter-frame. Considere por exemplo a mudança de *frame rate* (número de quadros por segundo), onde é necessário gerar imagens para instantes fracionários. Usar interpolação simples entre quadros adjacentes causará também neste caso perda de nitidez, pois objetos em movimento terão suas fronteiras borradas. Conhecendo-se as trajetórias dos objetos pode-se interpolar os pixels dos objetos em suas próprias trajetórias.

3.1 Métodos para estimação de movimento

Normalmente o deslocamento de partes da cena, de um quadro para o seguinte, está associado a uma solução da equação (aproximada)

$$f(x, y, t) \approx f(x - d_x, y - d_y, t - 1),\ \forall (x, y) \in B$$

para um certo subconjunto de pixels formando um bloco B.

O problema de escolher o tamanho deste bloco, também conhecido como *problema da abertura*, não é trivial: se a abertura é muito grande, o bloco pode conter vários objetos que se movem em direções diferentes; se a abertura é muito pequena, podemos ser incapazes de perceber qualquer movimento, ou porque a abertura não contém fronteiras de objetos (a abertura seleciona por exemplo o interior de um objeto homogêneo, como um bloco completamente azul correspondente a um recorte de um céu no fundo da cena), ou porque o movimento é paralelo à fronteira do objeto.

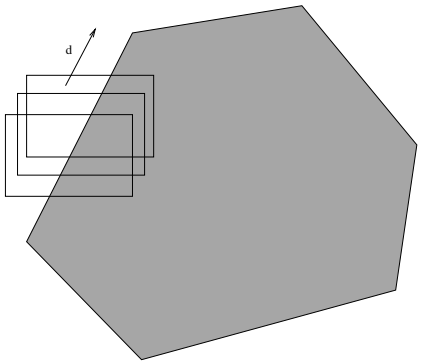

Figura 6: Movimento paralelo à fronteira do objeto com abertura pequena.

Outro problema é o da cobertura/descobertura de objetos ou da cena de fundo. Neste caso, blocos pequenos contendo porções só do objeto ou só do fundo poderão encontrar casamentos perfeitos, mas blocos contendo pedações da fronteira não produzirão vetores de deslocamento confiáveis.

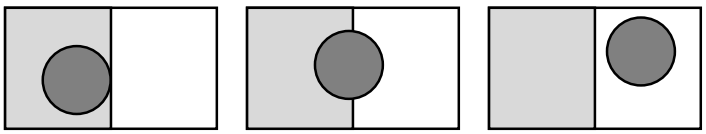

Figura 7: Problema da cobertura.

3.1.1 Método do casamento/emparelhamento de blocos

A ideia aqui é encontrar o vetor d que minimiza a discrepância entre os blocos, expressa como

$$\varepsilon^2(d) = \sum_{(x,y)\in B} \left(f(x, y, t) - f(x - d_x, y - d_y, t - 1)\right)^2$$

ou

$$\varepsilon(d) = \sum_{(x,y)\in B} \left|f(x, y, t) - f(x - d_x, y - d_y, t - 1)\right|,$$

sendo que esta última distância é menos suscetível a *outliers* (por exemplo, pixels ruidosos) e é computacionalmente mais barata. Assim definimos

$$d^* = \arg\min_d \varepsilon(d).$$

Além da busca exaustiva (que é viável considerando-se que os quadros têm tamanho finito), também podem ser empregadas buscas aproximadas, sub-amostrando o bloco no cálculo de $\varepsilon(d)$, ou subamostrando o espaço dos valores de d na busca. Por exemplo, no algoritmo de emparelhamento de blocos em três passos, é escolhido o melhor dentre 9 vetores de deslocamento "grosseiro", e o melhor destes serve de ponto de partida para um refinamento sucessivo em blocos com metade da altura e largura.

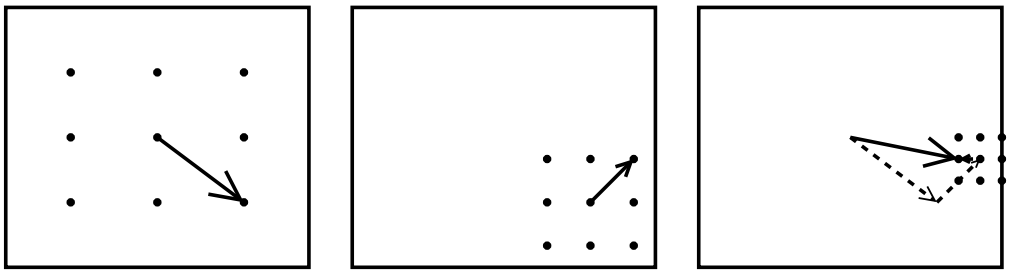

Figura 8: Emparelhamento em três passos.

Uma das medidas de qualidade da predição é o PSNR (*prediction signal-to-noise ratio*), definido como

$$PSNR = 10 \log_1 0 \left[\frac{I_0^2}{\varepsilon(d)} \right]$$

onde $I_0 = 255$ (para imagens com tons de cinza em 8 bits). Qualquer predição, por mais simples que seja, produz um PSNR melhor (maior) do que a simples diferença entre frames sucessivos, que é equivalente a usar $d = 0$ em $\varepsilon(d)$.

Para a codificação/compactação sem perdas, uma estimação imprecisa do movimento não afeta a qualidade do resultado, mas afeta o fator de compactação (pois nesse caso o resíduo será maior). Já no caso da filtragem com compensação de movimento, se o vetor d não corresponde ao movimento real, o resultado pode ficar comprometido. Outras estratégias de estimação mais cuidadosas podem produzir estimativas mais adequadas para a filtragem.

3.1.2 Emparelhamento de blocos hierárquico

A ideia aqui é aproveitar uma estrutura de representação em níveis sucessivos de detalhes, como aquela fornecida pela DWT completa (em $\log N$ etapas), para refinar sucessivamente a estimativa do vetor d.

Poderíamos começar em qualquer ponto da escala de aproximações. Para efeito de ilustração, considere a etapa de nível máximo, onde os blocos suavizados possuem tamanho 2×2. Podemos estimar o melhor deslocamento nessa resolução usando o emparelhamento de blocos simples. Este deslocamento d pode ser usado como ponto de partida para a minimização no próximo nível de resolução, com blocos de tamanho 4×4 (o vetor d também precisa ser redimensionado para $2d$). Deste modo podemos percorrer as várias escalas de detalhamento, com a expectativa de que a solução do nível seguinte esteja razoavelmente próxima daquela do nível anterior.

Caso a minimização do nível k chegue a um mínimo local ruim (com $\varepsilon(d)$ muito alto), podemos particionar o espaço de busca em 4, e com isso buscar vetores diferentes para cada partição, numa estratégia de exploração em árvores:

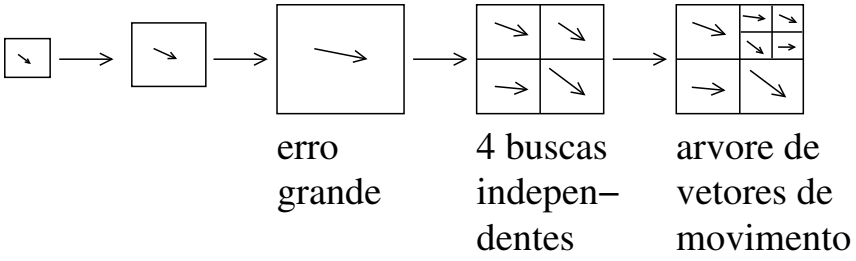

Figura 9: Emparelhamento hierárquico.

3.1.3 Compensação de movimento por blocos sobrepostos

Apesar do nome, trata-se de um método de adensamento da estimação de movimento através da interpolação dos vetores de deslocamento. A partir de um estimador qualquer, definem-se blocos adicionais, sobrepostos aos blocos usados no método anterior, e definem-se vetores de deslocamento para os blocos novos pela combinação linear dos vetores de deslocamento dos blocos originais, com pesos proporcionais aos respectivos fatores de sobreposição. Ao combinar estas informações com os blocos sobrepostos (o que corresponde a uma técnica de *overlap-add*) é possível disfarçar o efeito de "blocagem" associado às fronteiras dos blocos, através da suavização das imagens correspondentes a blocos sobrepostos. Este mecanismo é utilizado no padrão H.263 de compressão de vídeo.

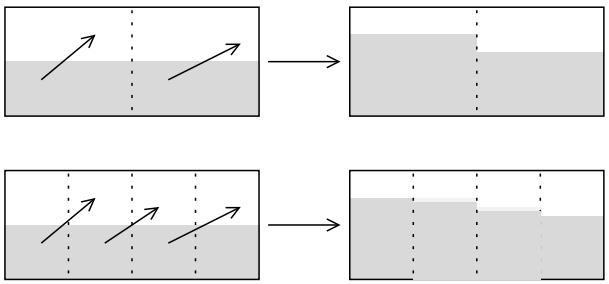

Figura 10: Compensação de movimento por blocos sobrepostos.

3.1.4 Estimação de movimento PEL-recursiva

Este método calcula vetores de deslocamento para cada pixel (PEL) do quadro. Os pixels são varridos sequencialmente, e o vetor de deslocamento ótimo do último pixel é usado como ponto inicial da minimização para encontrar o deslocamento ótimo do próximo pixel.

3.1.5 Métodos de fluxo ótico

A ideia é considerar a equação

$$f(x, y, t) = f(x - d_x, y - d_y, t - 1)$$

nas variáveis contínuas x, y e t, e calcular o lado direito de acordo com a expansão de Taylor de 1ª ordem

$$f(x - d_x, y - d_y, t - 1) = f(x, y, t) - [d_x \;\; d_y \;\; 1] \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial t} \end{bmatrix}.$$

Logo a equação original é equivalente a

$$d_x \frac{\partial f}{\partial x} + d_y \frac{\partial f}{\partial y} + \frac{\partial f}{\partial t} = 0,$$

cuja solução aproximada é obtida minimizando-se a expressão

$$\int\int\int_{R(x,y,t)} \left(d_x \frac{\partial f}{\partial x} + d_y \frac{\partial f}{\partial y} + \frac{\partial f}{\partial t} \right)^2 dx\, dy\, dt$$

numa abertura $R(x, y, t)$ pequena em torno de (x, y, t), onde as integrais são aproximadas como somas em grids e as derivadas são obtidas numericamente $\left(\frac{\partial f}{\partial x} \approx \frac{f(x+\Delta_x,y,t)-f(x,y,t)}{\Delta_x}\right)$. Este método permite gerar uma estimação densa (ao nível dos subpixels) dos vetores de deslocamento.

3.2 Filtros com compensação de movimento

Na filtragem com compensação de movimento devemos adaptar as equações dos filtros para encontrar pixels "correspondentes" nos quadros anteriores. Como exemplo, considere um filtro temporal (puro), definido pela equação

$$g(x, y, t) = \sum_{r \in N_t} h_r f(x, y, t - r),$$

e considere o fluxo de movimento do quadro $t - 1$ para o quadro t representado por uma função $d(x, y, t)$, no sentido de que o pixel (x, y) no instante t corresponde ao pixel $(x, y) - d(x, y, t)$ no instante $t - 1$. Nesse caso o filtro acima poderia ser redefinido como

$$\begin{aligned} g_c(x, y, t) &= h_0 f(x, y, t) + h_1 f(\overbrace{x - d_x(x, y, t)}^{x^{(1)}}, \overbrace{y - d_y(x, y, t)}^{y^{(1)}}, t - 1) \\ &\quad + h_2 f(\overbrace{x^{(1)} - d_x(x^{(1)}, y^{(1)}, t - 1)}^{x^{(2)}}, \overbrace{y^{(1)} - d_y(x^{(1)}, y^{(1)}, t - 1)}^{y^{(2)}}, t - 2) \\ &\quad + \cdots \\ &= \sum_{r \in N_t} h_r f(x^{(r)}, y^{(r)}, t - r) \end{aligned}$$

onde $(x^{(0)}, y^{(0)}) = (x, y)$ e

$$\begin{bmatrix} x^{(k+1)} \\ y^{(k+1)} \end{bmatrix} = \begin{bmatrix} x^{(k)} - d_x(x^{(k)}, y^{(k)}, t - k) \\ y^{(k)} - d_y(x^{(k)}, y^{(k)}, t - k) \end{bmatrix}.$$

No caso de um filtro espaço-temporal geral

$$g(x, y, t) = \sum_{p \in \mathbf{N_x}} \sum_{q \in \mathbf{N_y}} \sum_{r \in \mathbf{N_t}} h_{p,q,r} f(x - p, y - q, t - r).$$

a compensação deve ser feita para cada (p, q) independentemente:

$$g(x, y, t) = \sum_{p \in \mathbf{N_x}} \sum_{q \in \mathbf{N_y}} \sum_{r \in \mathbf{N_t}} h_{p,q,r} f(x^{(r)}(p, q), y^{(r)}(p, q), t - r),$$

onde $(x^{(0)}(p, q), y^{(0)}(p, q)) = (x, y)$ e

$$\begin{bmatrix} x^{(k+1)}(p, q) \\ y^{(k+1)}(p, q) \end{bmatrix} = \begin{bmatrix} x^{(k)}(p, q) - d_x(x^{(k)}(p, q), y^{(k)}(p, q), t - k) \\ y^{(k)}(p, q) - d_y(x^{(k)}(p, q), y^{(k)}(p, q), t - k) \end{bmatrix}.$$

Note que o custo computacional para obter $x^{(k+1)}(p, q)$ ou $y^{(k+1)}(p, q)$ é constante se conhecemos $x^{(k)}(p, q)$ e $y^{(k)}(p, q)$, e portanto o custo computacional do filtro compensado só aumenta de um fator constante em relação ao filtro não-compensado.

3.2.1 Exemplo de aplicação 1: conversão de *frame-rate*

Na conversão de frame-rate temos uma sequência de quadros $f_t(x, y)$ associados aos instantes (em segundos) $\frac{t}{FR_1}$ (para $t = 0, 1, \ldots$) e devemos gerar novos quadros $g_t(x, y)$ associados aos instantes $\frac{t}{FR_2}$ (para $t = 0, 1, \ldots$), o que equivale a dizer que devemos gerar quadros

$$f_{t-\lambda}(x, y) \quad \text{onde} \quad \lambda \in [0, 1].$$

A interpolação simples

$$f_{t-\lambda}(x, y) = \lambda f_{t-1}(x, y) + (1 - \lambda) f_t(x, y)$$

produz borramento nas fronteiras de quaisquer objetos em movimento. Usando os vetores de deslocamento $d(x, y, t)$ podemos definir a interpolação compensada através da expressão

$$\hat{f}_{t-\lambda}(x(\lambda), y(\lambda)) = \lambda f_{t-1}(x(1), y(1)) + (1 - \lambda) f_t(x(0), y(0))$$

onde

$$\begin{cases} x(\lambda) = x - \lambda d_x(x, y, t) \\ y(\lambda) = y - \lambda d_y(x, y, t) \end{cases}$$

e reamostrando a imagem $\hat{f}_{t-\lambda}$ nos pixels originais; isso pode ser feito arredondando os pixels $(x(\lambda), y(\lambda))$ ou combinando linearmente os resultados próximos para cada pixel (ou seja, fazendo uma interpolação intra-frame).

3.2.2 Exemplo de aplicação 2: *Deinterlacing*

Videos *interlaced* são compostos por quadros incompletos, onde quadros ímpares só possuem as linhas ímpares e quadros pares só possuem linhas pares (todas as outras linhas foram eliminadas na codificação). Gerar a informação faltante através de processamento intra-frame é ruim (por exemplo, um padrão listrado poderia simplesmente sumir); se não houvesse movimento as linhas faltantes poderiam ser copiadas do quadro anterior.

Na presença de movimento, podemos estimar de onde buscar a informação faltante, combinando vários quadros anteriores onde os pixels "desejados" estiveram presentes. Por exemplo, para gerar o pixel faltante $f(x, y, t)$, supondo x ímpar e t par, podemos fazer a média dos pixels

$$\begin{array}{ll} f(x^{(1)}, y^{(1)}, t - 1) & \text{se } x^{(1)} \approx \text{par} \\ f(x^{(2)}, y^{(2)}, t - 2) & \text{se } x^{(2)} \approx \text{ímpar} \\ f(x^{(3)}, y^{(3)}, t - 3) & \text{se } x^{(3)} \approx \text{par} \\ \vdots & \vdots \end{array}$$

adaptando a média à quantidade efetiva de pixels utilizados.