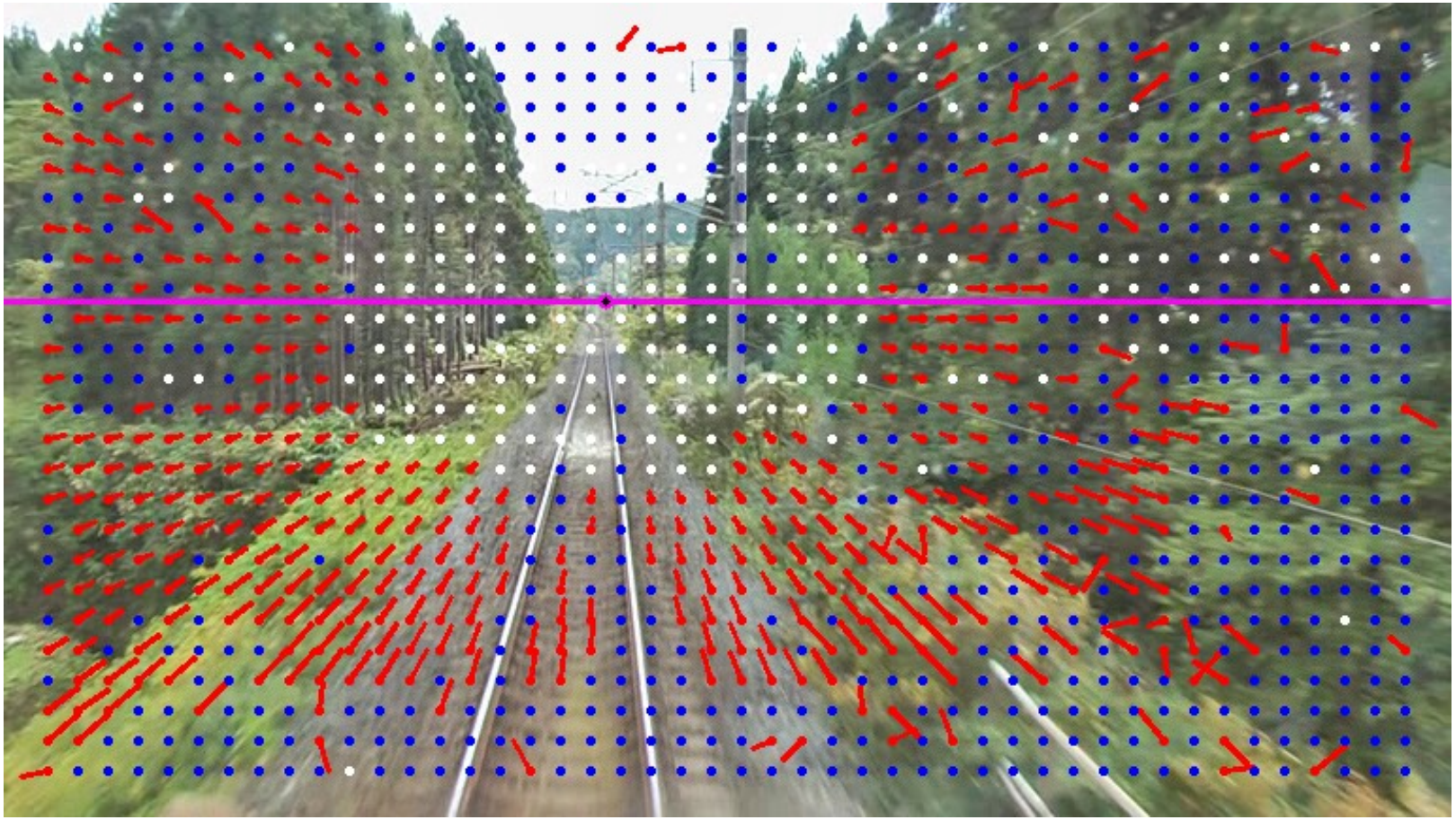
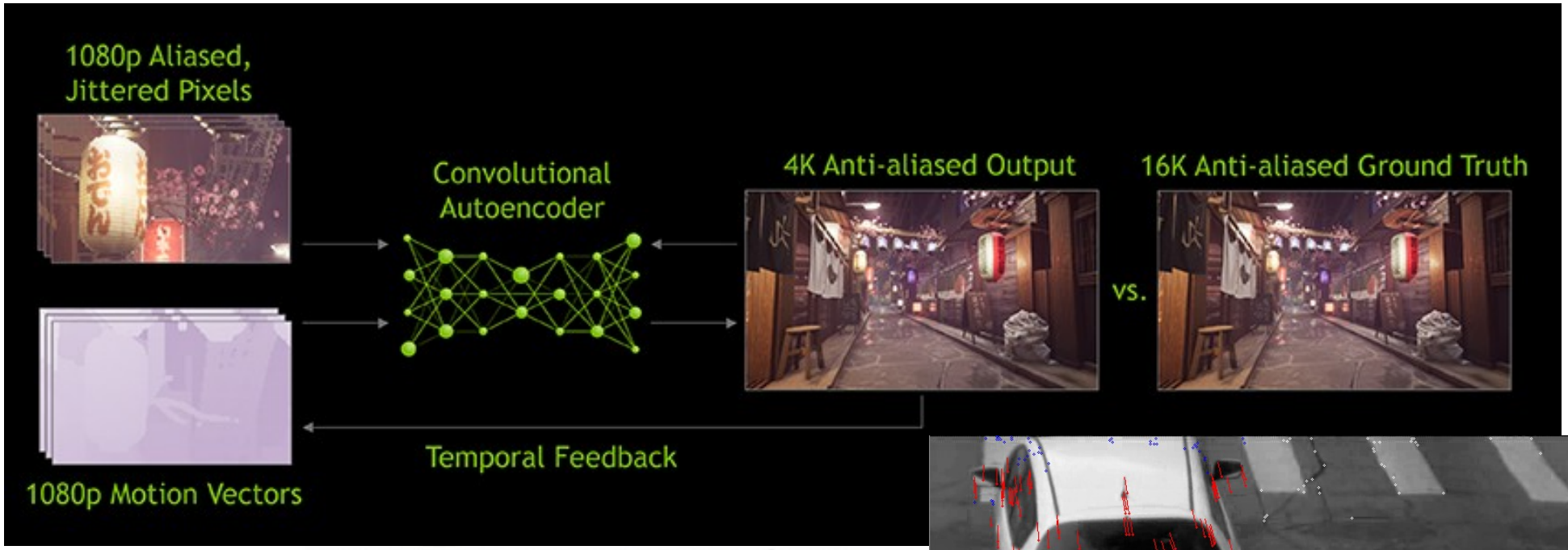
Visão Computacional

Prof. Bogdan Tomoyuki Nassu



Desafio: detecção de movimento



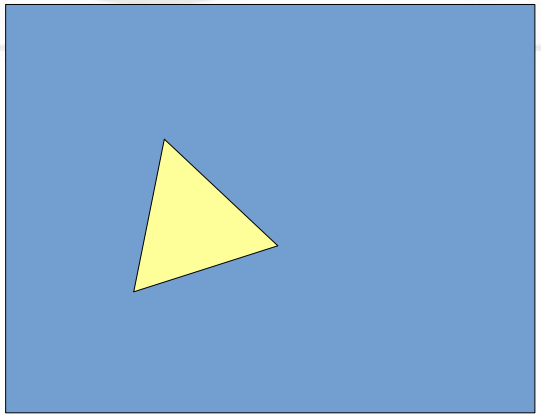
https://www.youtube.com/watch?v=3IaKJuZN55k

wccftech.com/neural-supersampling-is-a-hardware-agnostic-dlss-alternative-by-facebook/

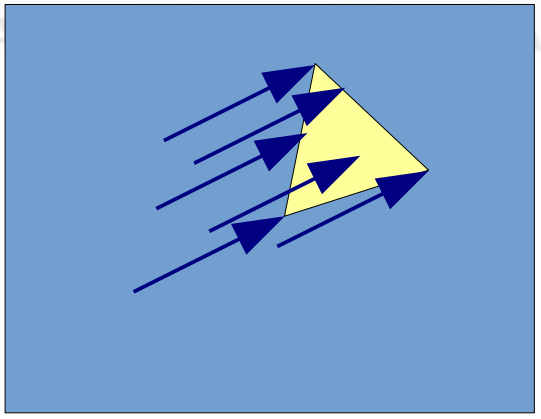
www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/

Fluxo ótico

- Fluxo ótico (*optical flow*).
  - Padrão de movimento de objetos, superfícies, cenas, etc.
    - = descrição do movimento de cada pixel entre 2 frames de vídeo.
    - Vetores de movimento.
  - Origem na biologia: modelagem de um componente da visão de animais, ligado à percepção de movimentos, forma e distância de objetos.



Frame *t*



Frame *t+1.* A setas são os vetores de movimento para 6 pixels.

Aplicações?

Aplicações

- Estabilização de câmera.
- Super-resolução de vídeos.
- *Structure from motion*.
- Interpolação temporal de vídeo para aumento de taxa de quadros.
- Rastreamento e segmentação de objetos.
- Compressão.
- (Mostrar exemplo do FOE).

Como fazer?

- Suponha um único pixel de um frame anterior. Como descobrir onde este pixel está no frame atual?
  - Nota: rigorosamente, os pixels não “se movem”, o que acontece é que movimentos da câmera ou do objeto fazem com que a luz refletida sobre um mesmo ponto do mundo real esteja projetada em uma coordenada diferente da imagem! Mas por simplicidade, vamos falar que os pixels “se movem”.

Pressupostos

- Pressupostos:
  - Constância do brilho: um pixel permanece com (quase) a mesma intensidade/cor entre os 2 frames.
  - Coerência espacial: pontos vizinhos em um frame permanecem próximos no frame seguinte.
    - = a região ao redor de um pixel muda lentamente no tempo.
- Voltando à pergunta: como localizar um pixel do frame anterior no frame atual?

Como fazer?

- Como localizar um pixel do frame anterior no frame atual?
- Podemos tomar uma janela (um *patch*) ao redor do pixel no frame anterior, e procurar por uma janela similar no frame atual?
  - (ou vice-versa)
  - Precisamos medir a similaridade entre uma região de um frame e regiões de outro frame.
    - Como?

Block matching

•Precisamos medir a similaridade entre uma região de um frame e regiões de outro frame.

- Como?
  - R: *template/block matching*.
    - O “template” é uma região ao redor do pixel em um frame.
    - Procuramos por uma região parecida em outro frame.
      - = “De onde veio este pixel?”

Fluxo ótico denso

- *Fluxo ótico denso* - ideia geral:
  - Para cada pixel $p=(x_t,y_t)$ do frame do tempo t:
    - Seleciona uma janela de largura $2w+1$ com centro em p.
    - Busca regiões similares à janela no frame do tempo *t-1,* computando uma pontuação para cada alinhamento possível.
      - A posição (x_{t-1},y_{t-1}) do alinhamento com maior pontuação é a posição de p no tempo *t-1.*
    - O vetor de movimento do pixel p é obtido por $[x_t - x_{t-1}, y_t - y_{t-1}]$.
- Como comparar duas janelas / patches?

Fluxo ótico denso

- *Fluxo ótico denso* - ideia geral:
  - Para cada pixel $p=(x_t,y_t)$ do frame do tempo t:
    - Seleciona uma janela de largura $2w+1$ com centro em p.
    - Busca regiões similares à janela no frame do tempo *t-1,* computando uma pontuação para cada alinhamento possível.
      - A posição (x_{t-1},y_{t-1}) do alinhamento com maior pontuação é a posição de p no tempo *t-1.*
    - O vetor de movimento do pixel p é obtido por $[x_t - x_{t-1}, y_t - y_{t-1}]$.
- Como comparar duas janelas / patches?
  - Diferença quadrática.
  - Coeficiente de correlação.
  - etc.

Fluxo ótico denso: refinando

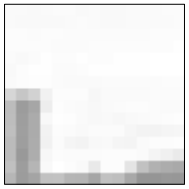
- Problema: custo computacional **extremamente** alto.
  - Formas simples de melhorar?

Fluxo ótico denso: refinando

- Problema: custo computacional **extremamente** alto.
  - Formas simples de melhorar:
    - Imagens em escala de cinza.
    - Em vez de procurar todos os pixels do frame *t* no frame *t-1*, fazer amostragem em espaços regulares.
    - Em vez de procurar por um pixel $p=(x_t,y_t)$ em todo o frame *t-1*, procurar apenas em uma região próxima a (x_t,y_t).

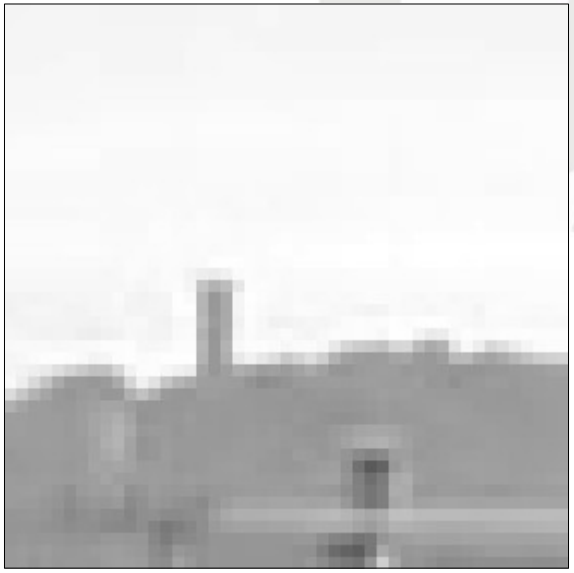
- 3 parâmetros:
  - Largura da janela ao redor de cada pixel $(2w+1)$.
    - Nos exemplos: janela quadrada com $w=7$.
  - Espaçamento entre os pontos que serão procurados a cada frame.
    - Nos exemplos: 16 pixels.
  - Tamanho da vizinhança onde cada janela será procurada.
    - Nos exemplos: $x=[x_0-16,x_0+16]$ e $y=[y_0-16, y_0+16]$.

Exemplo

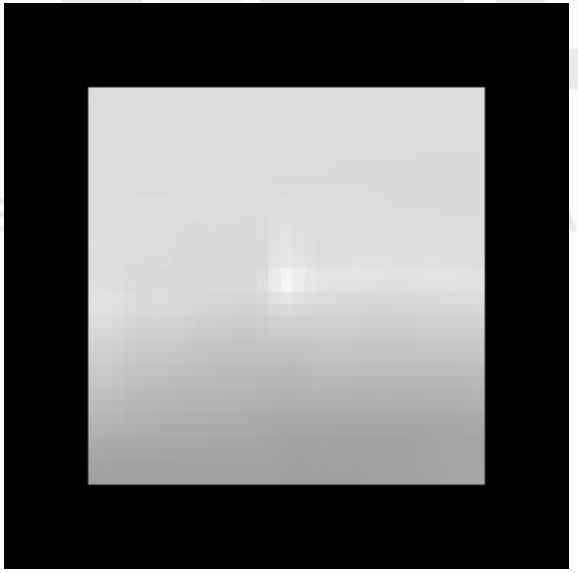


Região no frame *t*

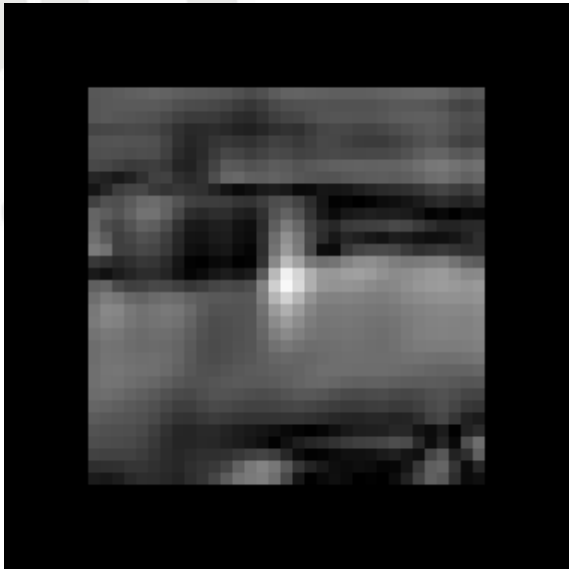
Janela no frame *t-1*



1 - Diferença quadrática



Coeficiente de correlação

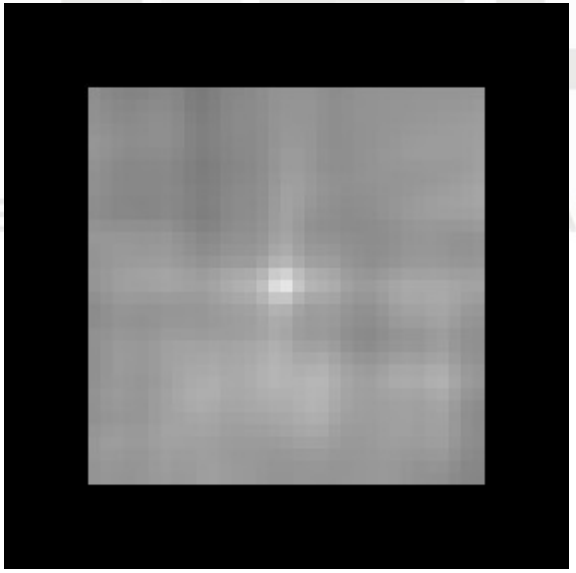


Exemplo

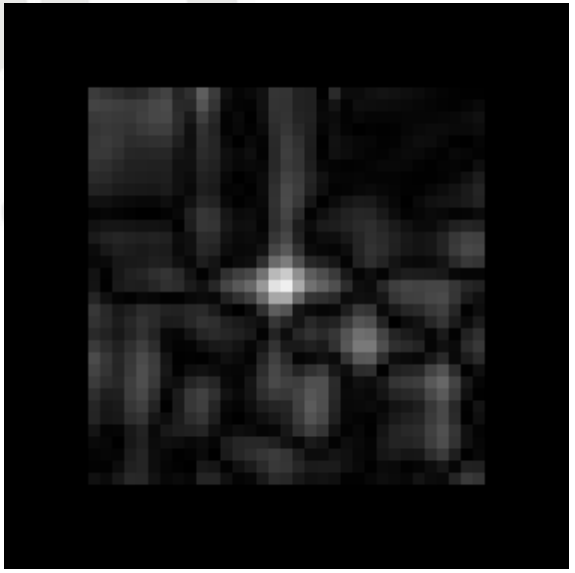
Região no frame *t*

Janela no frame *t-1*

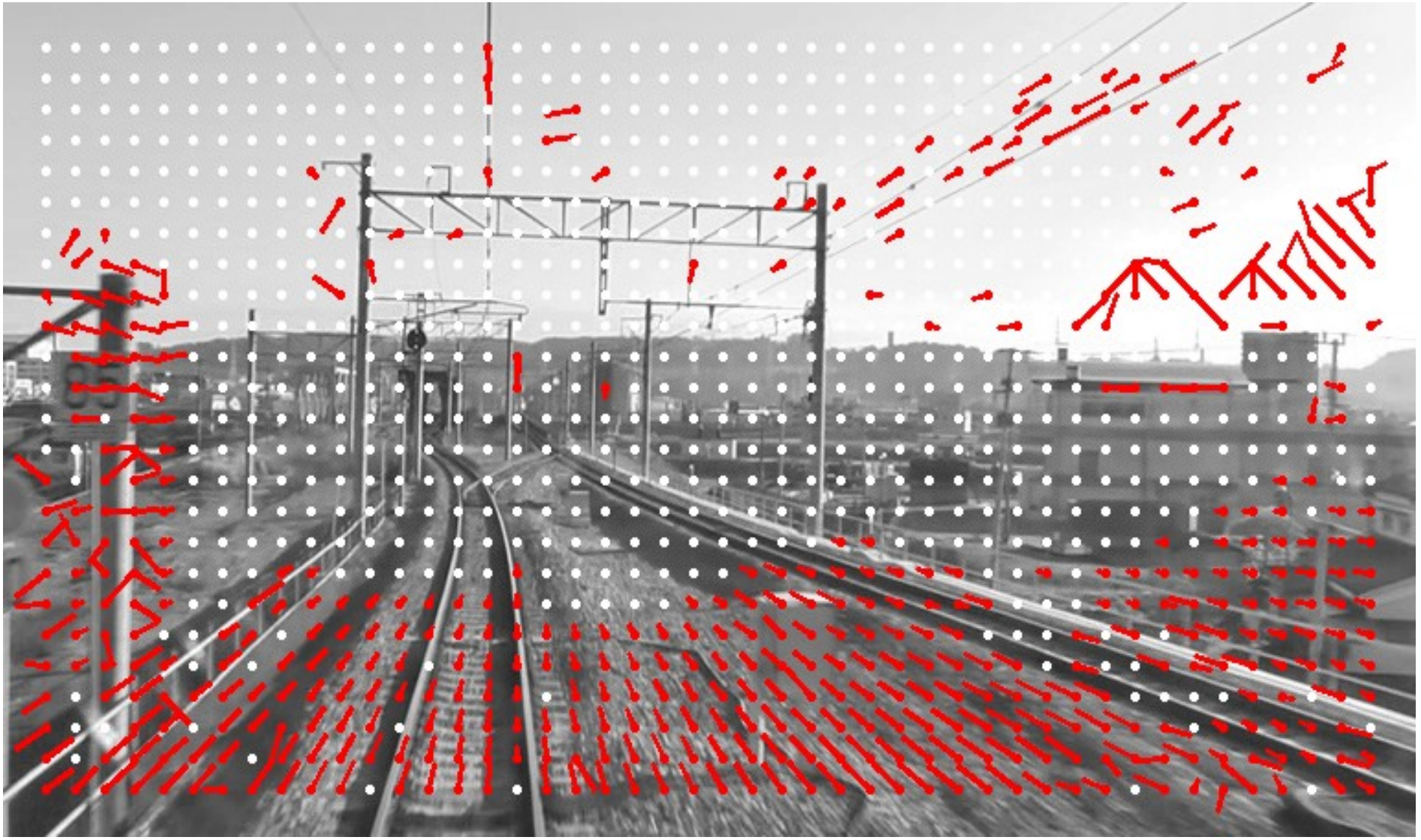
1 - Diferença quadrática



Coeficiente de correlação

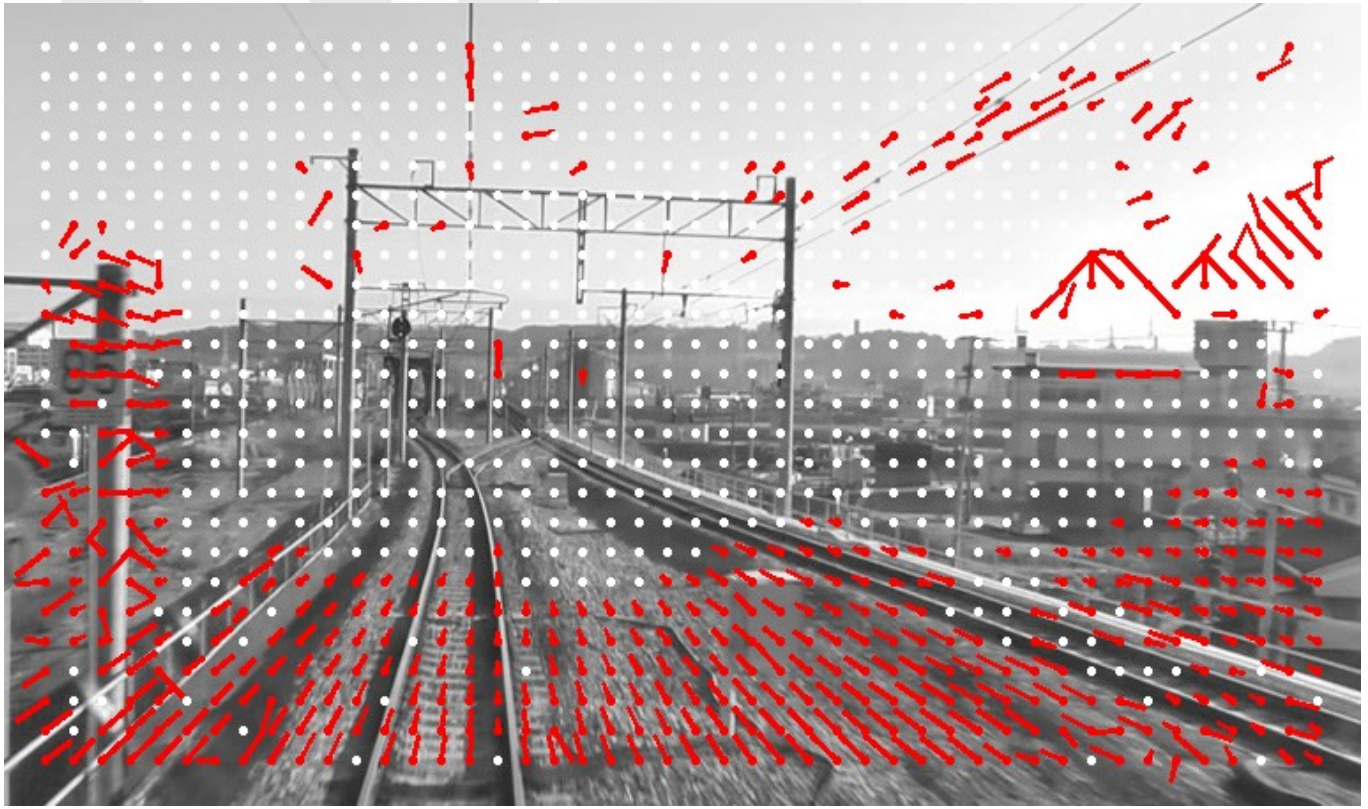


Exemplo



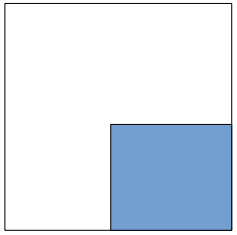
Analisando os resultados

- Observemos:
  - Como o comprimento dos vetores varia de acordo com a distância?
  - Por que existem “pontos parados”?
  - Por que alguns vetores parecem tão errados???

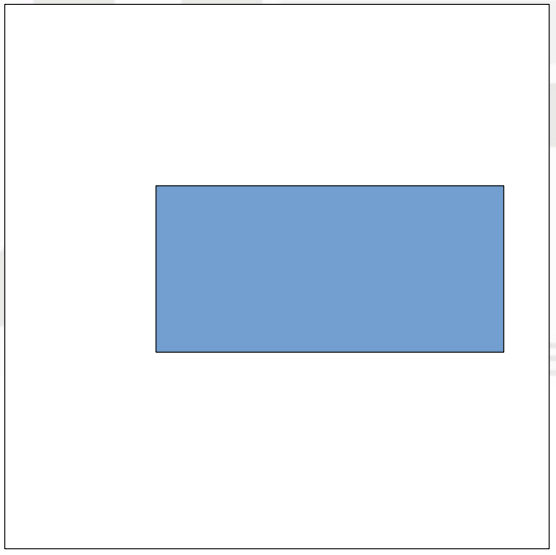


O problema da abertura

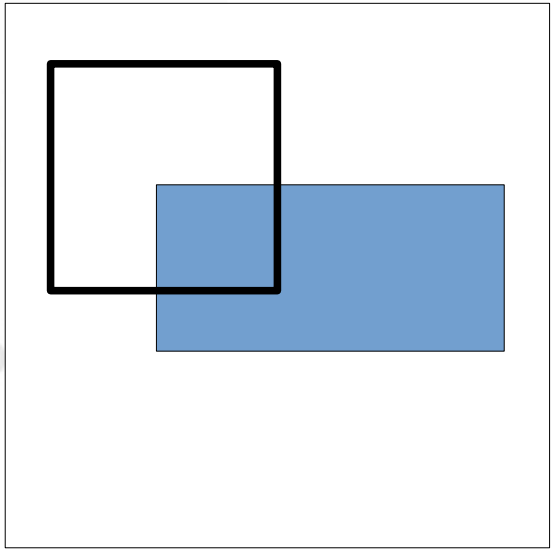
•O problema mais fundamental que temos quando analisamos o fluxo ótico é conhecido como “problema da abertura”.



Janela ao redor de um pixel no frame *t*.

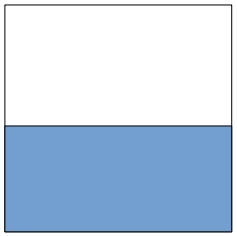


Região de busca no frame *t+1*.

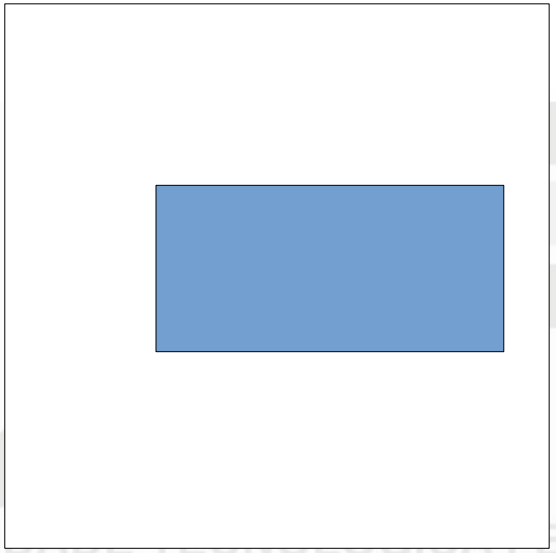


Podemos facilmente determinar o melhor casamento!

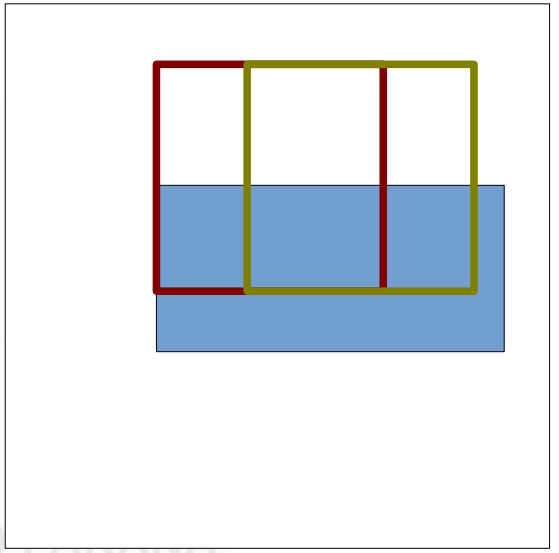
O problema da abertura



Janela ao redor de um pixel no frame *t*.

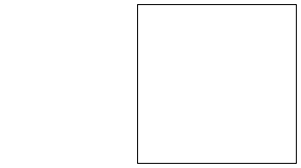


Região de busca no frame *t+1*.

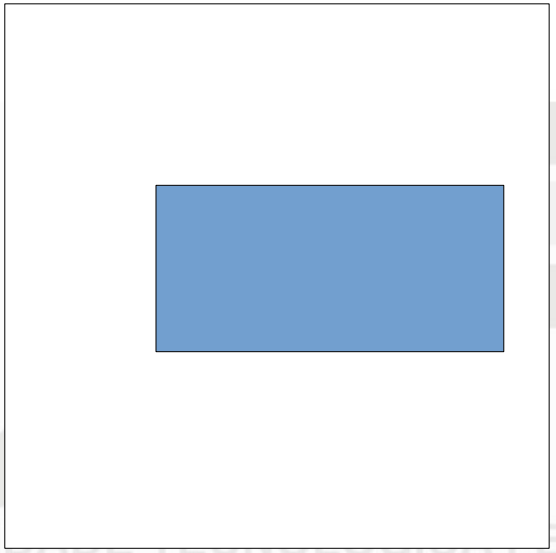


Temos vários casamentos com a mesma pontuação!

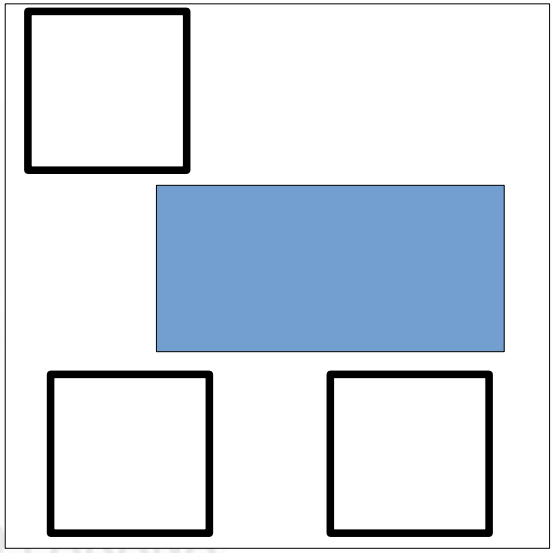
O problema da abertura

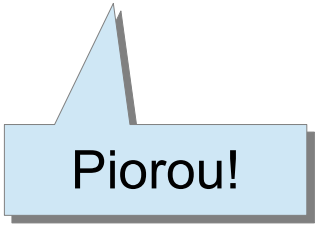


Janela ao redor de um pixel no frame *t*.



Região de busca no frame *t+1*.





O problema da abertura

•O problema da abertura ocorre quando a região que estamos buscando é pouco distintiva, ou pode ser facilmente confundida com outras regiões.

- Como resolver?

O problema da abertura

•O problema da abertura ocorre quando a região que estamos buscando é pouco distintiva, ou pode ser facilmente confundida com outras regiões.

- Buscar uma região maior da imagem resolveria o problema?

O problema da abertura

•O problema da abertura ocorre quando a região que estamos buscando é pouco distintiva, ou pode ser facilmente confundida com outras regiões.

- Buscar uma região maior da imagem resolveria o problema?
  - O casamento pode ficar muito difícil se a janela for grande demais, a não ser que uma região grande mude muito pouco.
  - Como determinar o tamanho da janela?

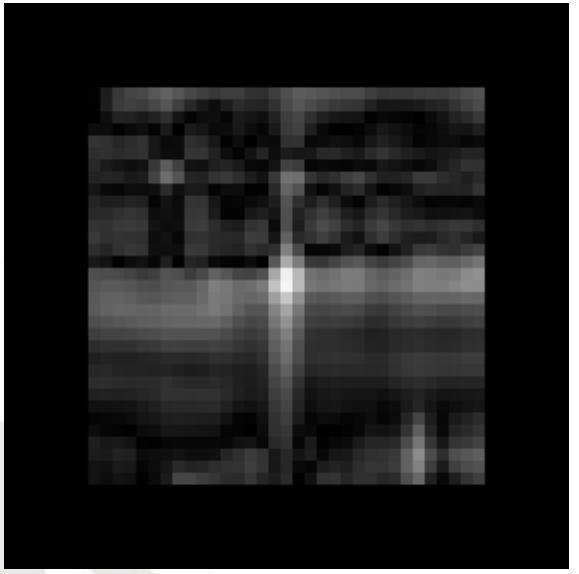
Fluxo ótico denso: mais desafios

- Desafios para fluxo ótico denso:
  - Movimentos que não fazem parte da cena (ex: televisão ligada).
  - Movimentos falsos, criados por brilhos e reflexos.
    - Ex: esfera brilhante parada e uma luz móvel.
  - Movimentos de objetos não-rígidos (animais, tecidos, líquidos, etc.).
- Vários algoritmos já foram propostos para o problema do fluxo ótico.
  - Outras formas de se comparar regiões.
  - Múltiplos candidatos a vetor para um mesmo ponto.
    - Pode ser útil para tratar incertezas ou mesmo transparências!
  - Precisão sub-pixel.
  - Pirâmides de imagens, para análise multi-escala.
- Alternativa?

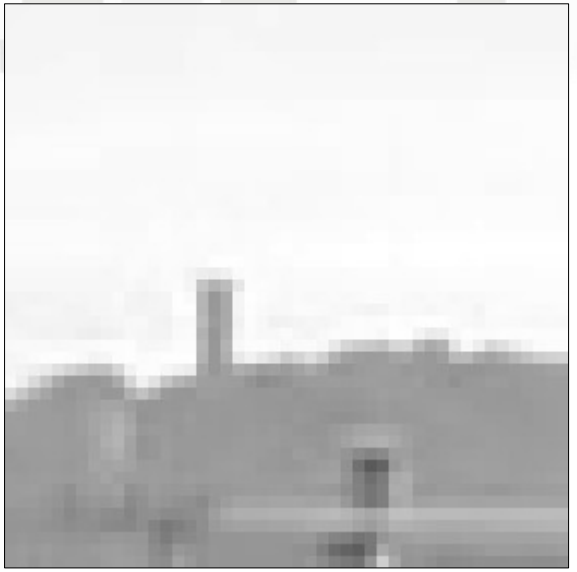
Qual a diferença?

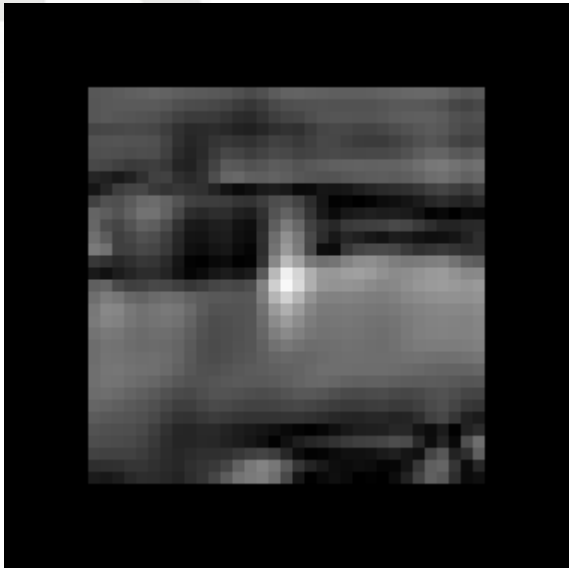
- Qual a diferença entre as regiões com casamentos bons e ruins?

Bons casamentos...

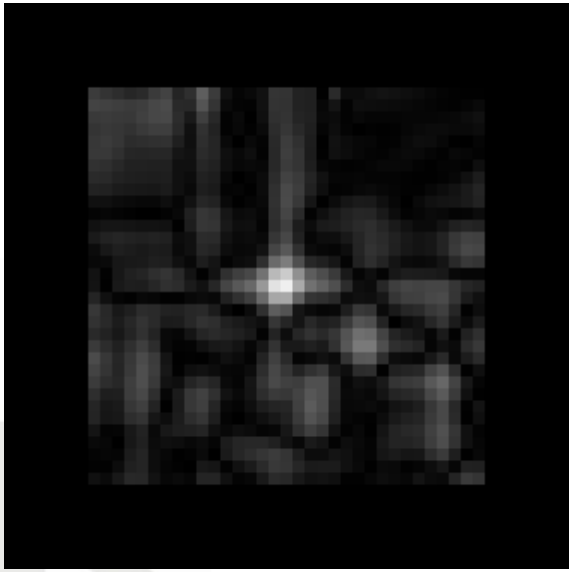


1. Região no frame *t.*
2. Janela no frame *t-1*.
3. Pontuações.

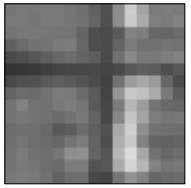


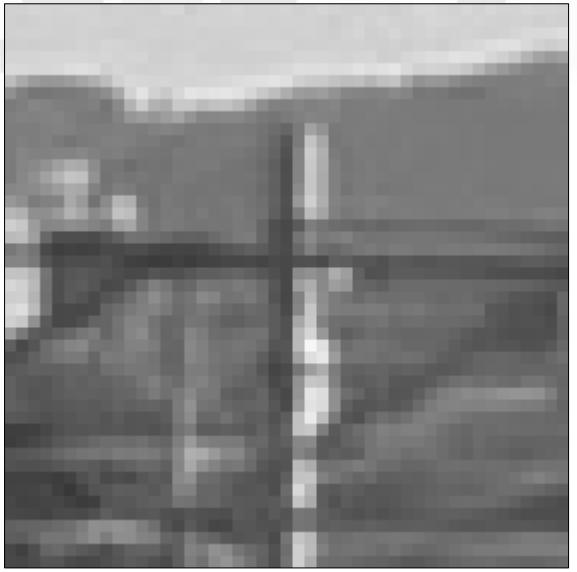


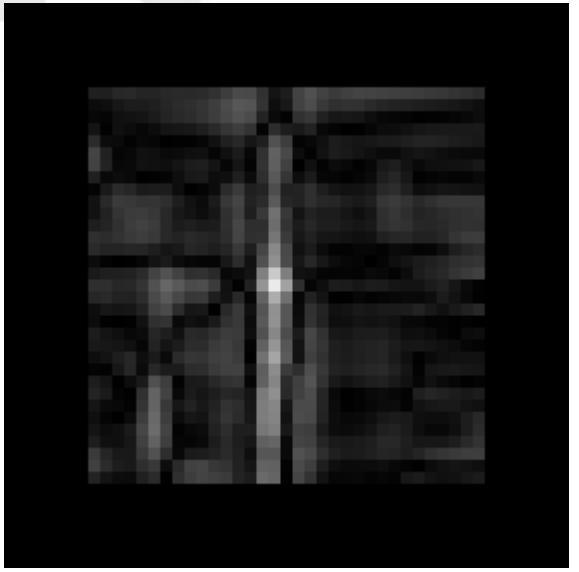
Bons casamentos...



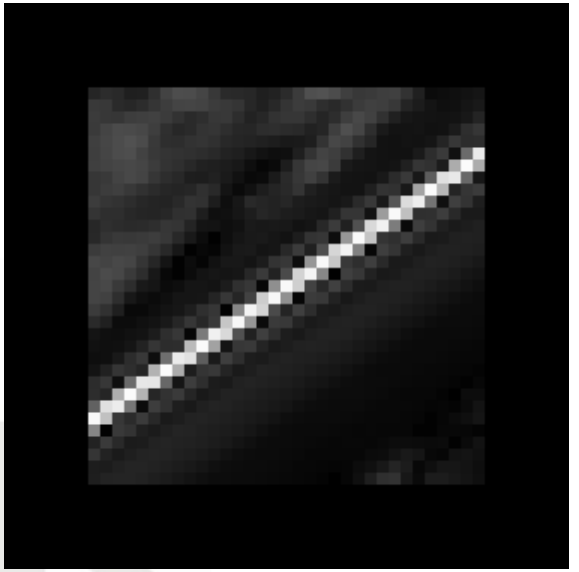
1. Região no frame *t.*
2. Janela no frame *t-1*.
3. Pontuações.



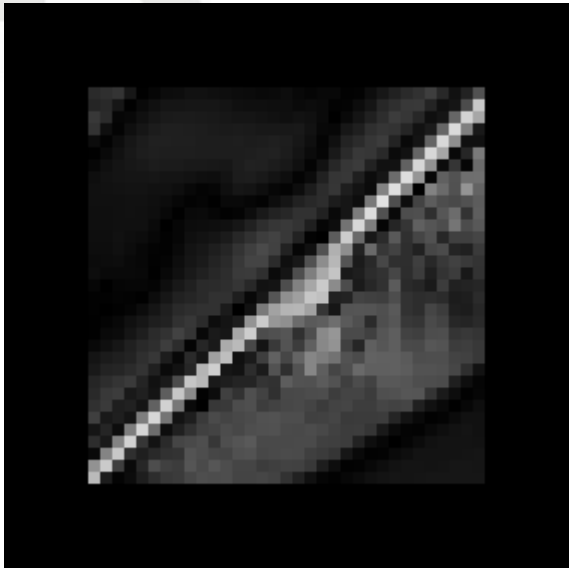




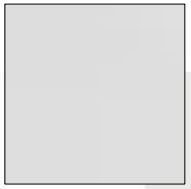
Maus casamentos...

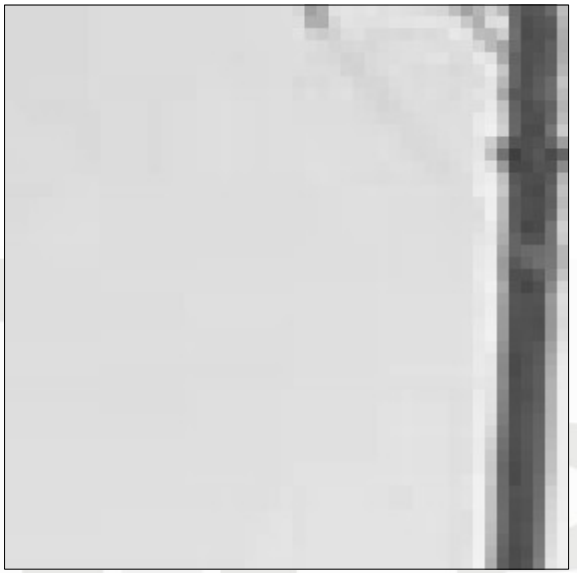


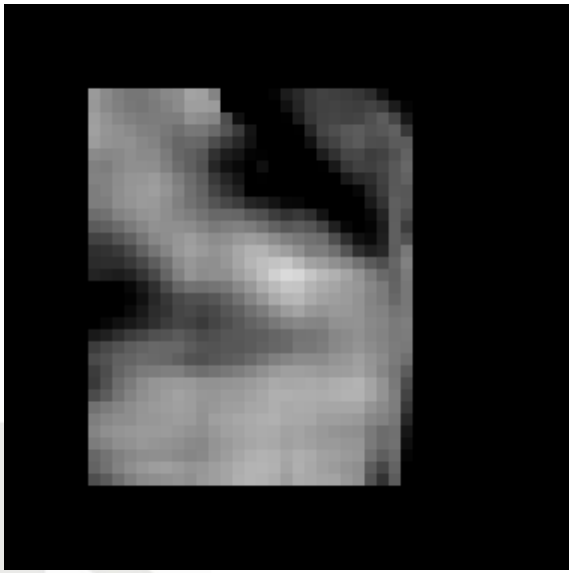
1. Região no frame *t.*
2. Janela no frame *t-1*.
3. Pontuações.



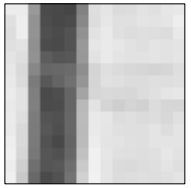
Maus casamentos...

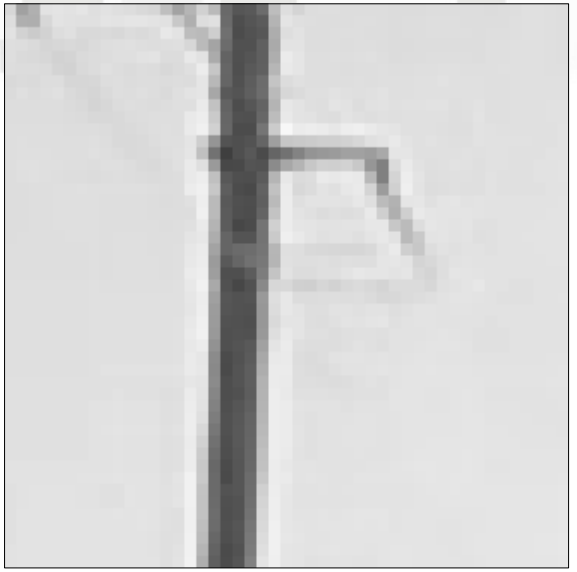


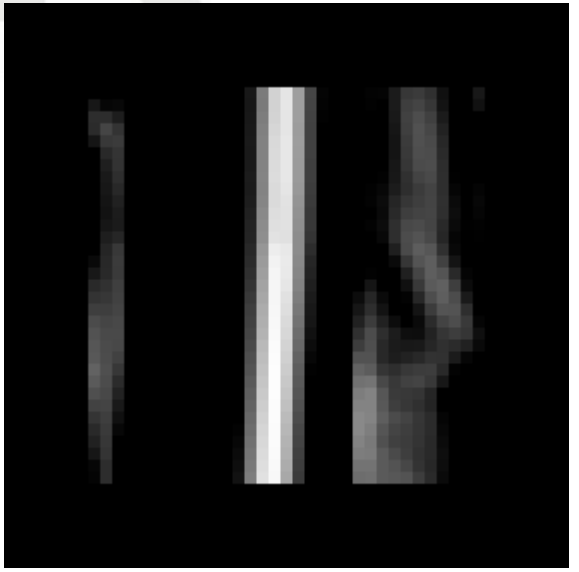




1. Região no frame *t.*
2. Janela no frame *t-1*.
3. Pontuações.







Qual a diferença?

- Qual a diferença entre as regiões com casamentos bons e ruins?

Qual a diferença?

- Qual a diferença entre as regiões com casamentos bons e ruins?
  - Regiões ruins têm textura pobre ou muito simples.
    - Ex: um traço em uma direção.
  - Regiões com bons casamentos possuem “cantos”.
- Como definir um “canto”?

Qual a diferença?

- Qual a diferença entre as regiões com casamentos bons e ruins?
  - Regiões ruins têm textura pobre ou muito simples.
    - Ex: um traço em uma direção.
  - Regiões com bons casamentos possuem “cantos”.
- Como definir um “canto”?
  - R: um canto é um encontro entre duas bordas!
- E o que caracteriza uma borda?

Qual a diferença?

- Qual a diferença entre as regiões com casamentos bons e ruins?
  - Regiões ruins têm textura pobre ou muito simples.
    - Ex: um traço em uma direção.
  - Regiões com bons casamentos possuem “cantos”.
- Como definir um “canto”?
  - R: um canto é um encontro entre duas bordas!
- E o que caracteriza uma borda?
  - A existência de um gradiente forte.
- → o que caracteriza um canto (*corner*) é a existência de gradientes fortes em pelo menos duas direções.

Mudando o nosso jeito de pensar

- Suponha que já temos um algoritmo que detecta cantos. Como isso poderia ser explorado para computar o fluxo ótico?

Mudando o nosso jeito de pensar

- Suponha que já temos um algoritmo que detecta cantos. Como isso poderia ser explorado para computar o fluxo ótico?
  - Toma a vizinhança de cada canto detectado no frame *t*, e procura no frame *t*-1 pela região mais similar?
    - (ou vice-versa)
  - Toma a vizinhança de cada canto detectado no frame *t*, e procura entre os cantos detectados no frame *t*-1 o que tem o entorno mais similar?
    - (ou vice-versa)
  - *Fluxo ótico esparso*: acompanha somente pontos promissores.
- Já sabemos como computar a similaridade entre duas regiões.
  - Precisamos de um algoritmo para detectar cantos!

Auto-similaridade

- A primeira ideia é detectar cantos com base na “auto-similaridade”.
  - Detector de Moravec (1980).
  - Conceito similar ao do fluxo ótico denso, mas aplicado ao próprio frame:
    - ???

Auto-similaridade

- A primeira ideia é detectar cantos com base na “auto-similaridade”.
  - Detector de Moravec (1980).
  - Conceito similar ao do fluxo ótico denso, mas aplicado ao próprio frame:
    - Para cada pixel da imagem, toma uma vizinhança e a “desliza” sobre a própria imagem, em uma região ao redor do pixel.
      - Janela e variações de posição pequenas!
    - Para cada alinhamento, computa uma pontuação.
      - Diferença quadrática.
  - Região homogênea: ???
  - Borda: ???
  - Canto: ???

Auto-similaridade

- A primeira ideia é detectar cantos com base na “auto-similaridade”.
  - Detector de Moravec (1980).
  - Conceito similar ao do fluxo ótico denso, mas aplicado ao próprio frame:
    - Para cada pixel da imagem, toma uma vizinhança e a “desliza” sobre a própria imagem, em uma região ao redor do pixel.
      - Janela e variações de posição pequenas!
    - Para cada alinhamento, computa uma pontuação.
      - Diferença quadrática.
  - Região homogênea: pontuações boas para muito alinhamentos.
  - Borda: várias pontuações boas quando a região estiver sobre a borda.
  - Canto: pequenas mudanças no alinhamento reduzem a pontuação.

- Problemas...
  - Alto custo computacional.
  - Deslocamentos discretos.
  - Ruídos nas respostas.

Harris corners

- O detector de cantos de Harris (“Harris corners”), de 1988, é um dos trabalhos fundamentais sobre a detecção de cantos.
- O algoritmo parte do conceito de auto-similaridade e, através de manipulações algébricas, transforma a equação da diferença quadrática para um deslocamento.

Derivadas de 2ª ordem

•O detector de Harris é baseado nas derivadas de segunda ordem *Dxx, Dxy* e *Dyy*.

•Para computar as derivadas de segunda ordem, podemos calcular os “gradientes dos gradientes” (i.e. usar um filtro como Sobel duas vezes), ou montar kernels apropriados.

- Exemplo: kernels 3x3.

Dxx:

1	-2	1
2	-4	2
1	-2	1

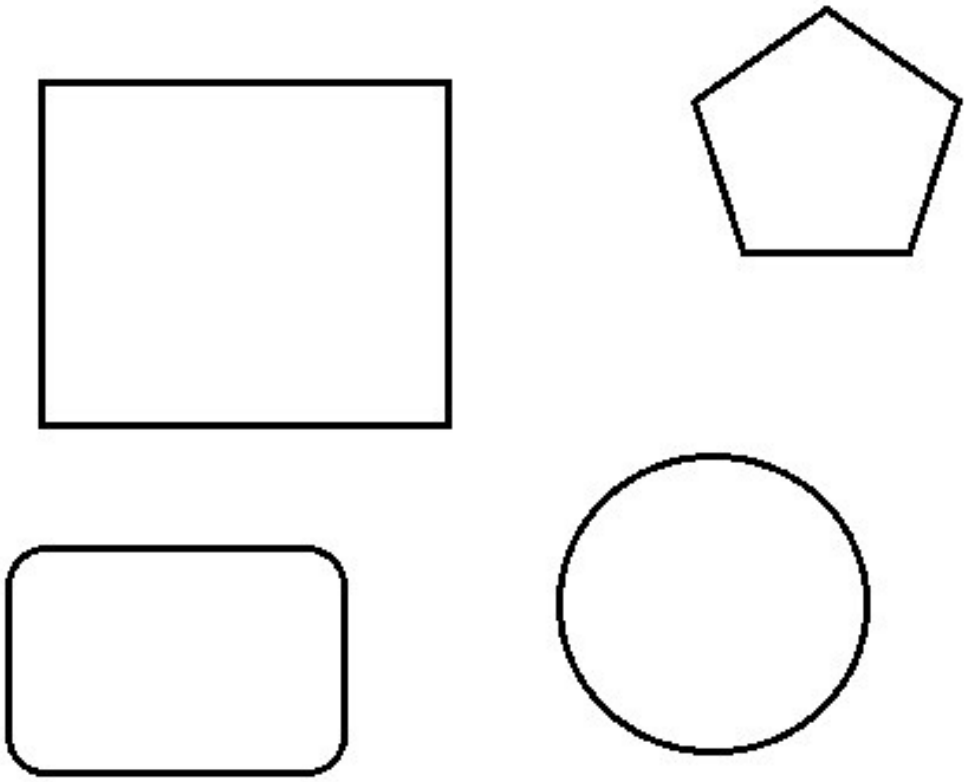
Dyy:

1	2	1
-2	-4	-2
1	2	1

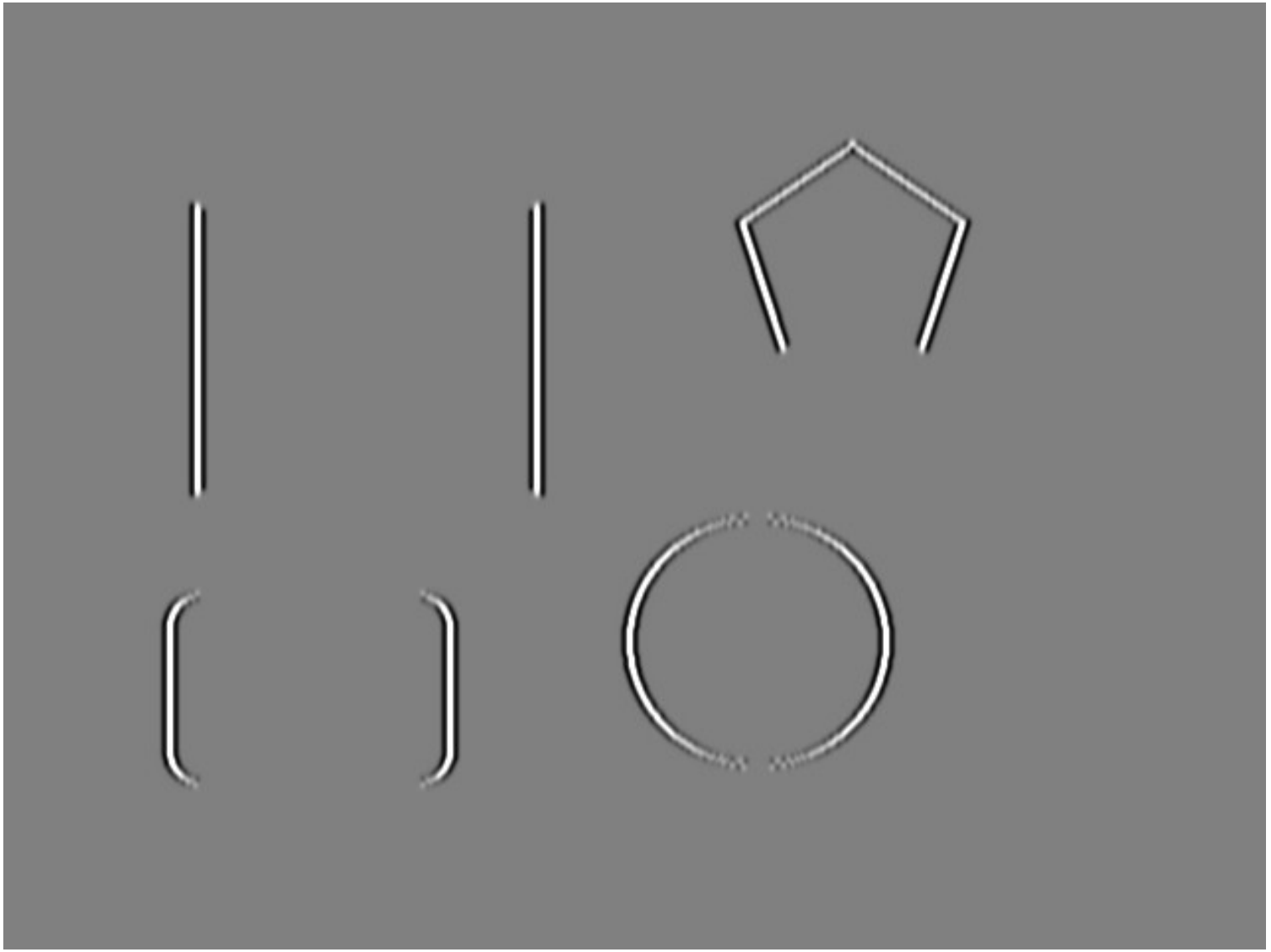
Dxy:

1	0	-1
0	0	0
-1	0	1

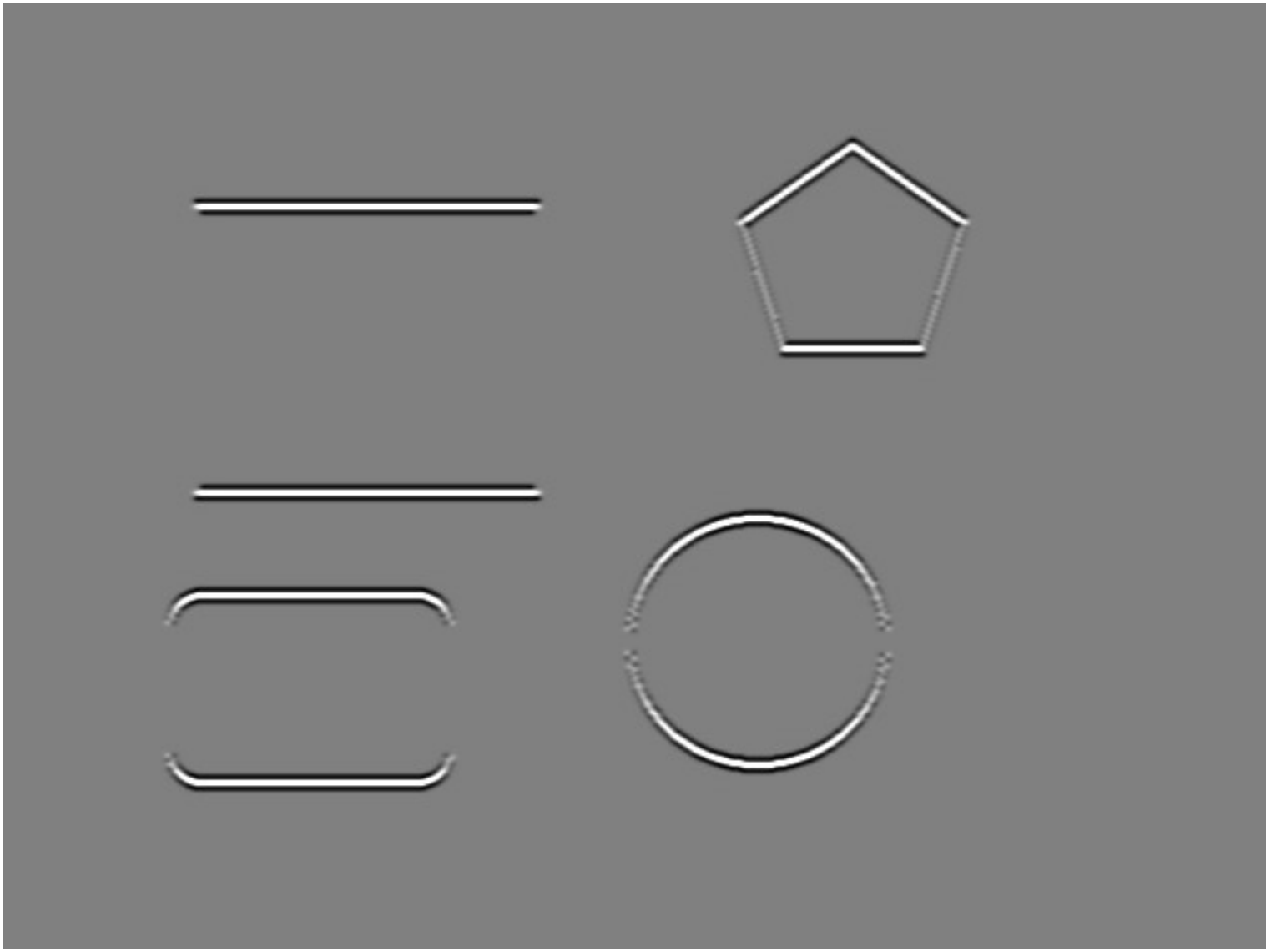
Exemplo: imagem original



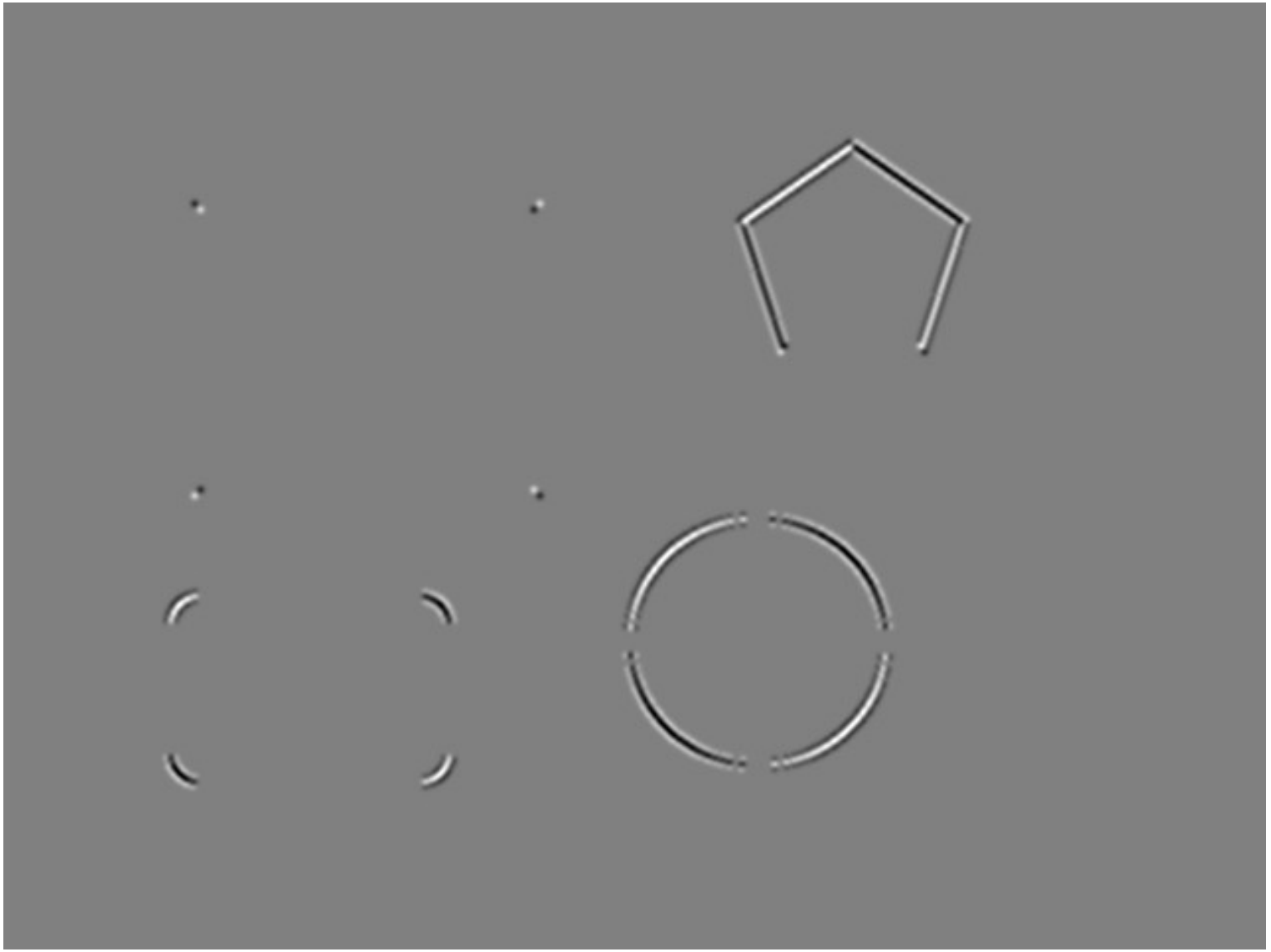
Exemplo: Dxx



Exemplo: Dyy

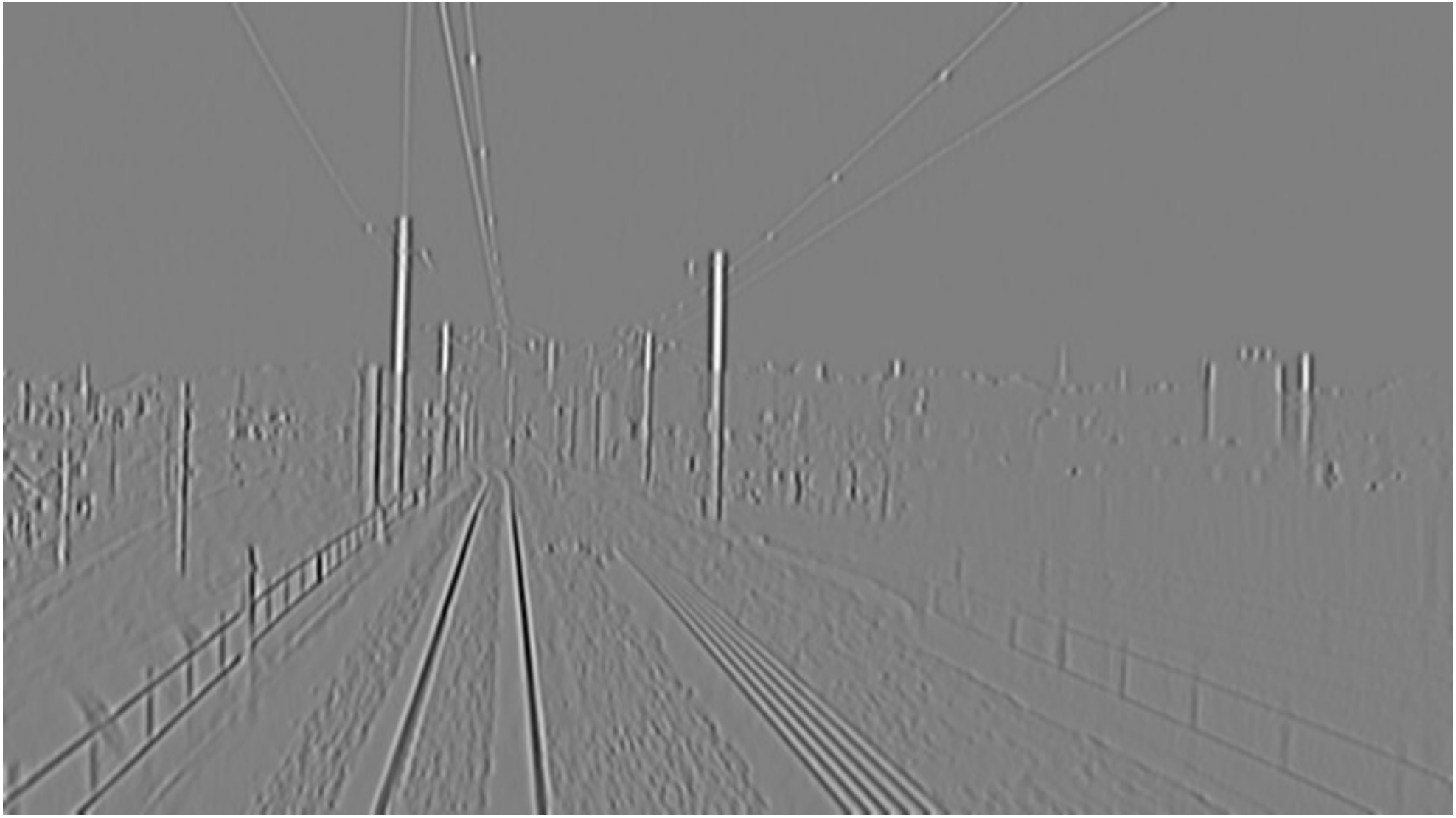


Exemplo: Dxy

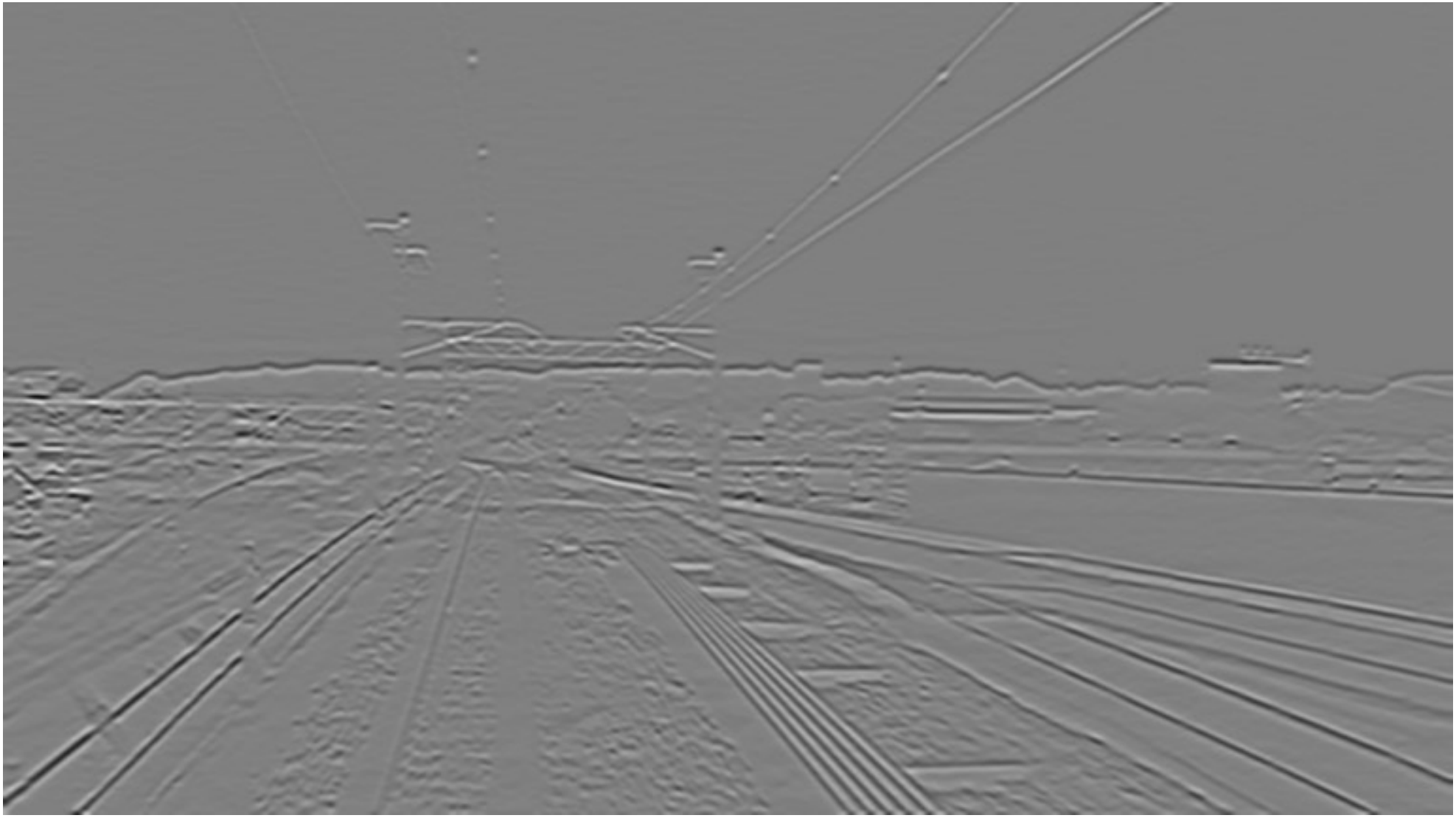


Exemplo: imagem original

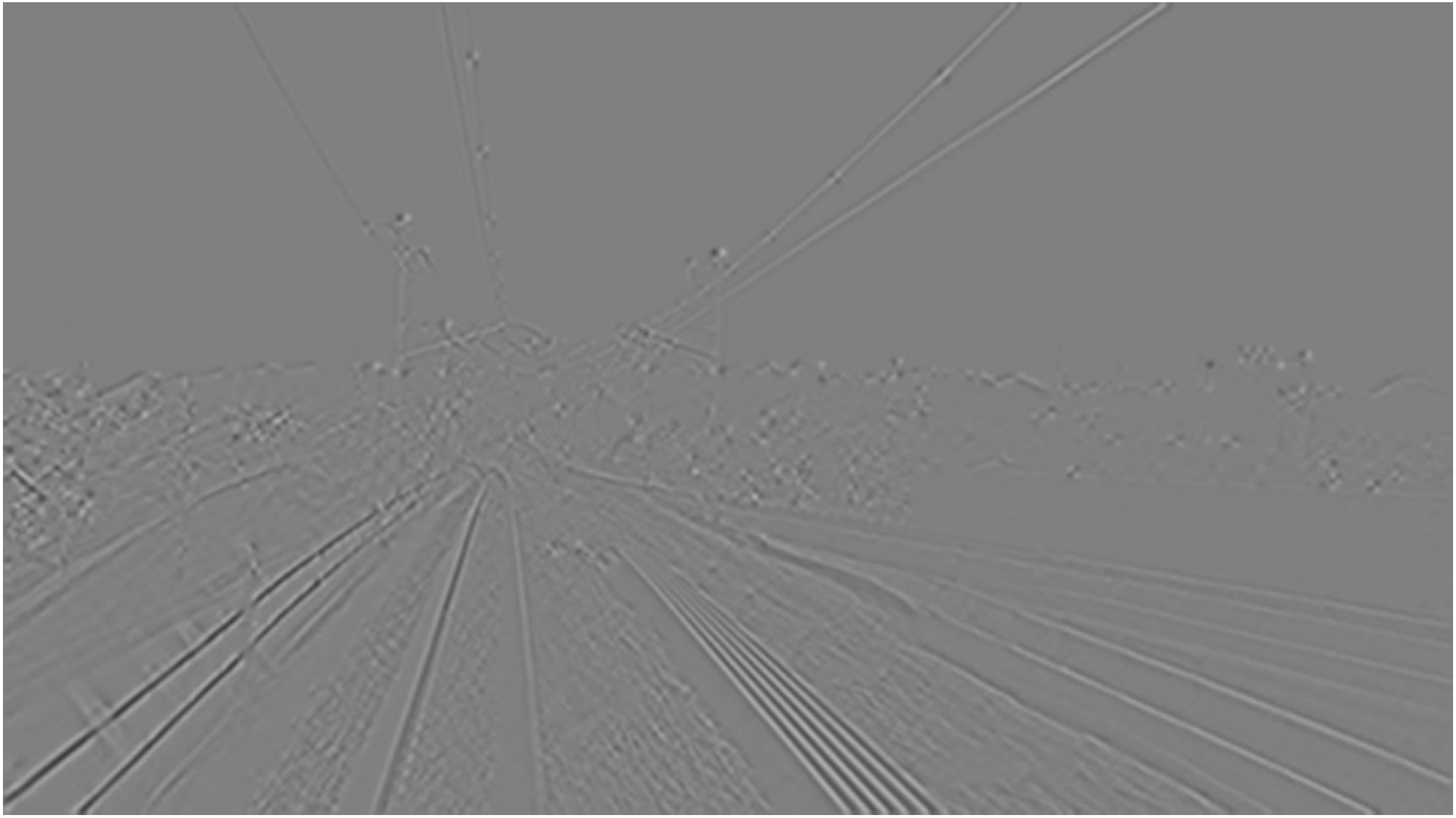
Exemplo: Dxx



Exemplo: Dyy



Exemplo: Dxy



Matriz de autocorrelação

•O próximo passo do algoritmo de Harris é computar uma matriz de autocorrelação, definida por:

$$A(x,y)=\begin{vmatrix} Sxx & Sxy \\ Sxy & Syy \end{vmatrix}=\sum w \begin{vmatrix} Dxx & Dxy \\ Dxy & Dyy \end{vmatrix}$$

•Aqui, *w* é uma função de pesagem.

- A versão original da auto-similaridade teria *w*=1 para todas as posições em uma janela quadrada.
- Para reduzir a sensibilidade a ruído, *w* é normalmente uma função Gaussiana centrada em (x,y).
  - Em termos de implementação: basta aplicar um filtro Gaussiano sobre as imagens Dxx, Dxy e Dyy; e tomar os valores na posição (x,y) dessas imagens filtradas.
  - O kernel usado normalmente é pequeno – por exemplo, a implementação do OpenCV usa por padrão um kernel de largura 3.

Análise dos autovalores

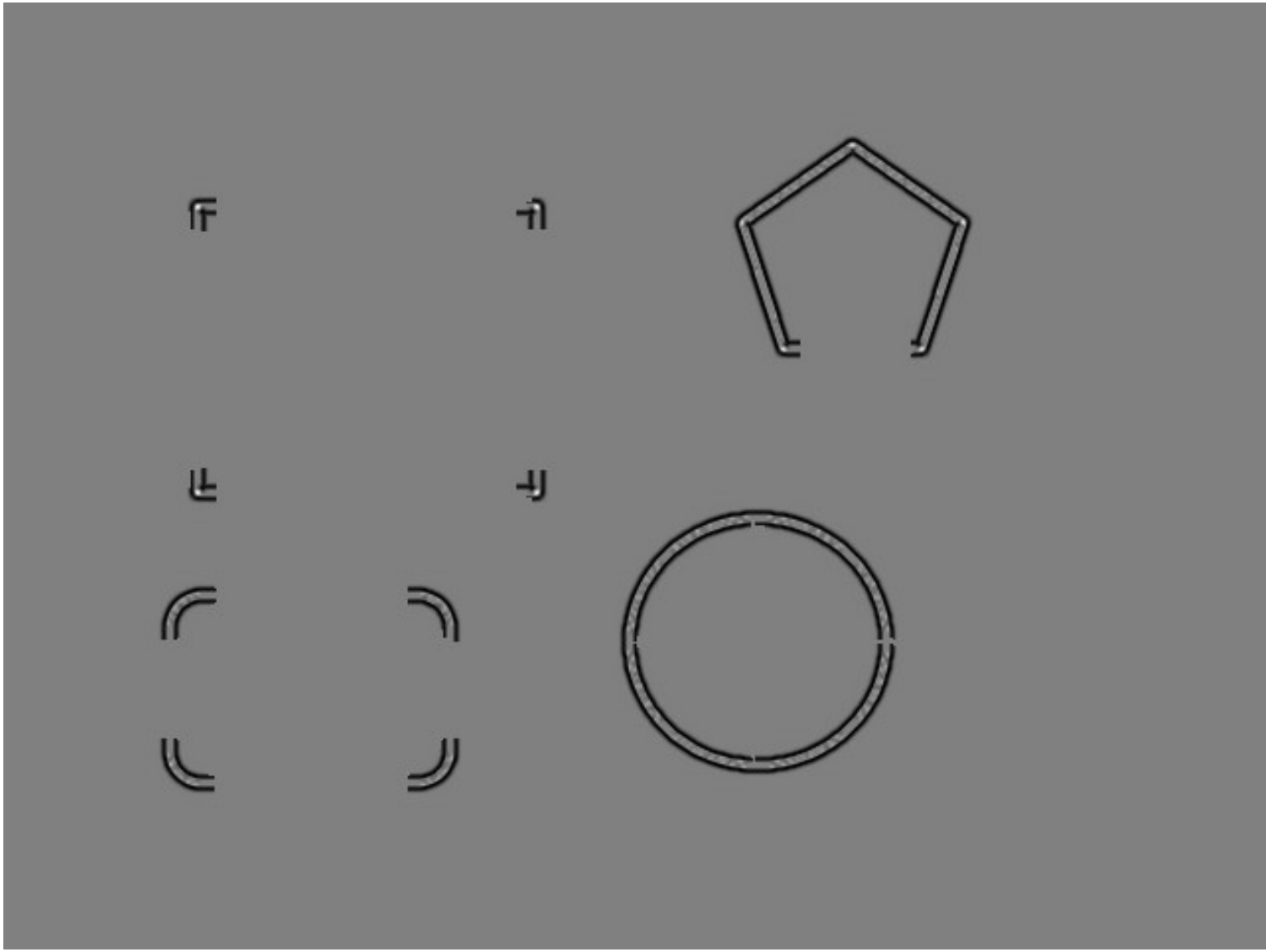
•Calculamos o determinante e o traço da matriz de autocorrelação:

$Det\,(A) = Sxx*Sxy - Sxy^2$
$Tr\,(A) = Sxx + Syy$

•Neste passo, existem várias alternativas.
- Algoritmo original de Harris: é canto se $Det(A) - \alpha Tr(A)^2 >$ um limiar.
- Alternativa de Shi e Tomasi: é canto se o menor autovalor da matriz for maior que um limiar. O menor autovalor λ pode ser calculado assim:

$$\lambda = Tr(A)/2 - \sqrt{Tr(A)^2/4 - Det(A)}$$

Exemplo: menores autovalores



Exemplo: menores autovalores

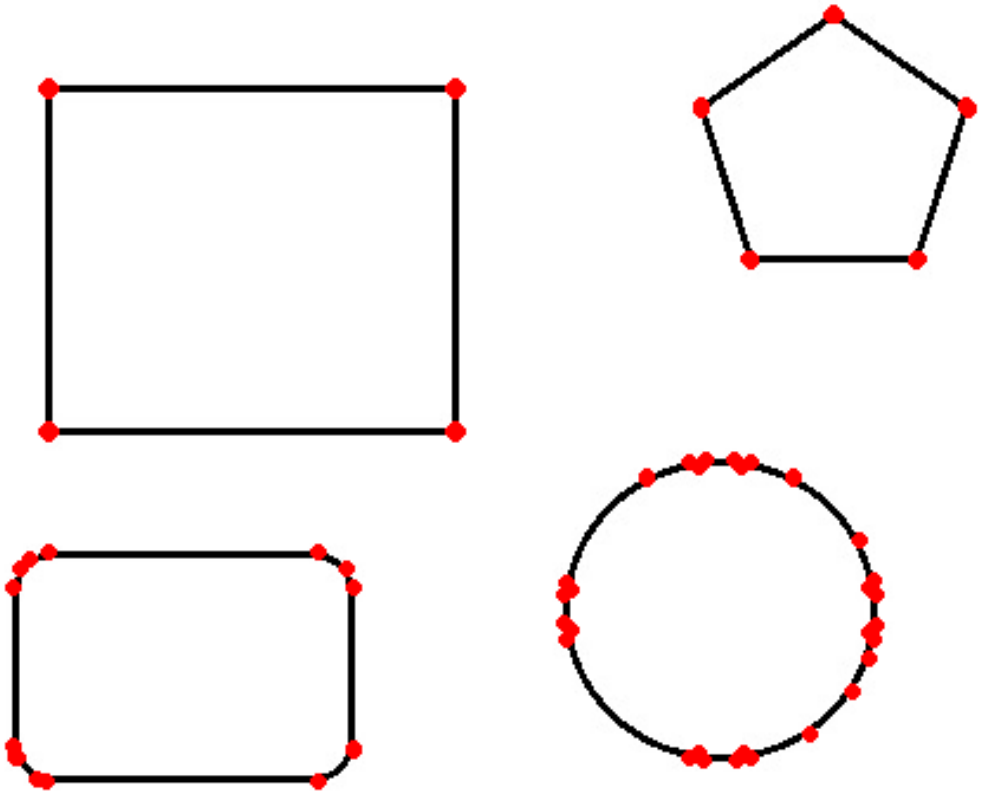
Um pequeno “truque”...

- A seleção de limiares para os autovalores tem uns problemas:
  - O limiar é um número absoluto, não relativo à própria imagem.
  - O limiar “ideal” pode variar de acordo com as características da imagem.
- Uma forma de facilitar a seleção de limiares é descrevê-los em função dos próprios resultados.
  - Seja $I\lambda$ a imagem contendo os menores autovalores calculados para cada posição.
  - Seja λm o maior valor em $I\lambda$ (o “maior entre os menores autovalores”).
  - Definimos que (x,y) é um canto se $I\lambda(x,y) > t \cdot \lambda m$, onde t é um limiar.
    - Ou seja, o limiar agora é dado em função dos autovalores calculados.
    - = Estamos detectando “os melhores cantos” de uma imagem dada.

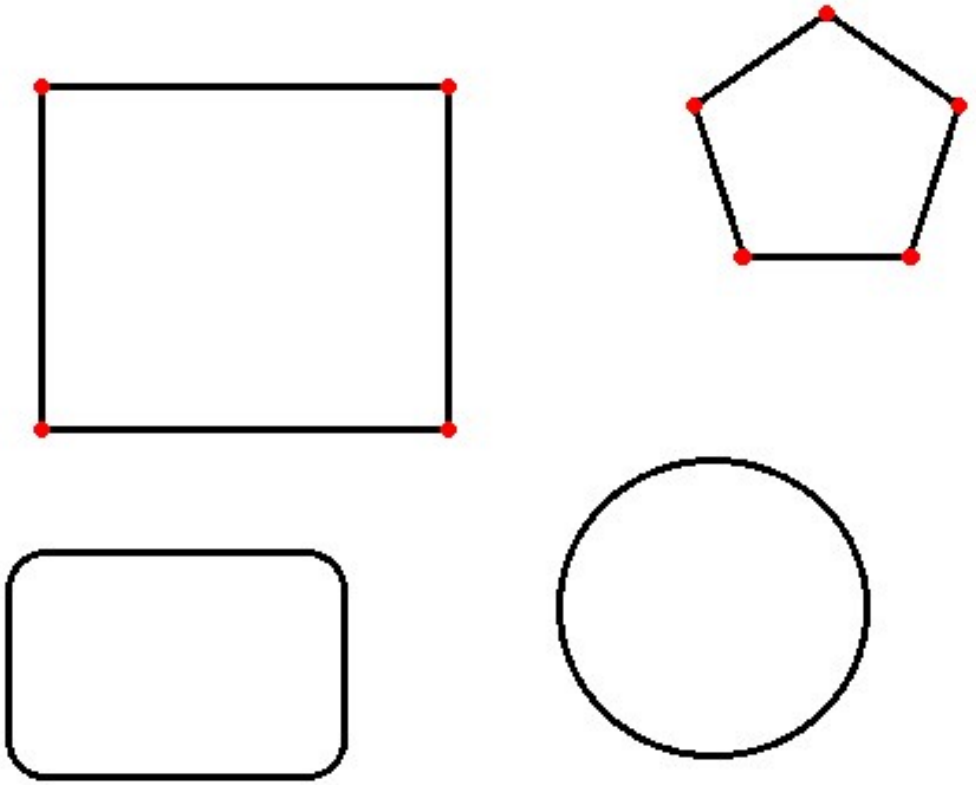
Vejamos alguns exemplos...

- Vejamos exemplos de cantos detectados em imagens e vídeos.
- O OpenCV inclui uma implementação otimizada do algoritmo, em uma função com o sugestivo nome de **`goodFeaturesToTrack`**.
  - *Good Features to Track* também é o título do artigo de Shi & Tomasi que descreve o algoritmo KLT (mais sobre ele em breve)...
- O OpenCV também tem uma função **`cornerSubPix`**, que estima a posição dos cantos detectados com precisão subpixel.
  - Consulte a documentação se precisar.

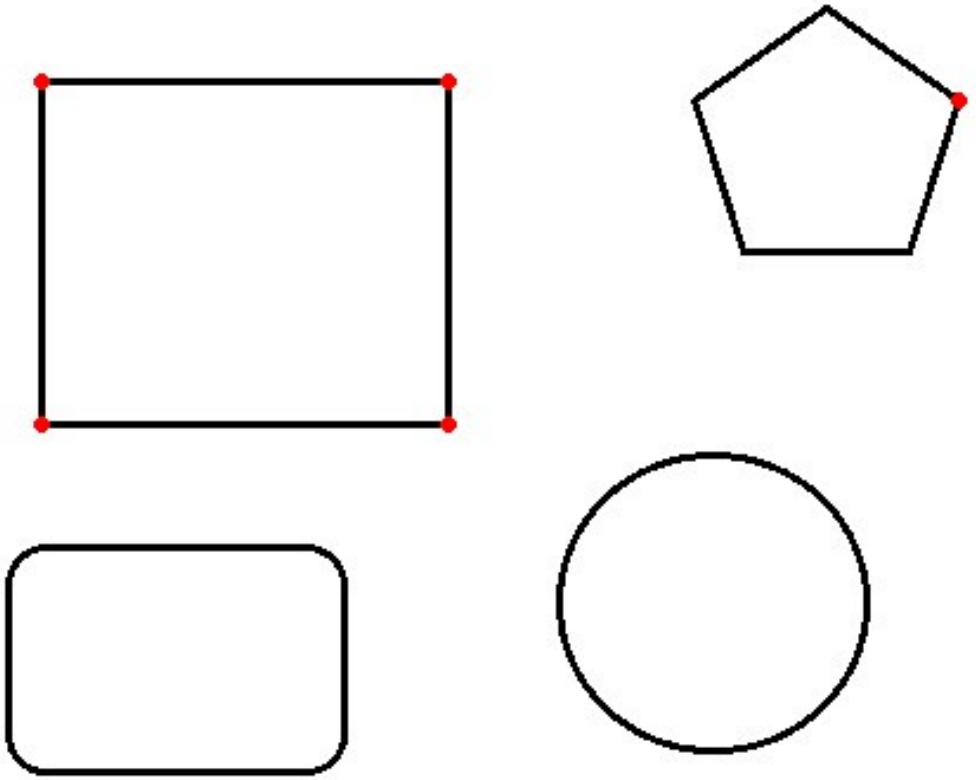
Exemplo (t = 0.25)



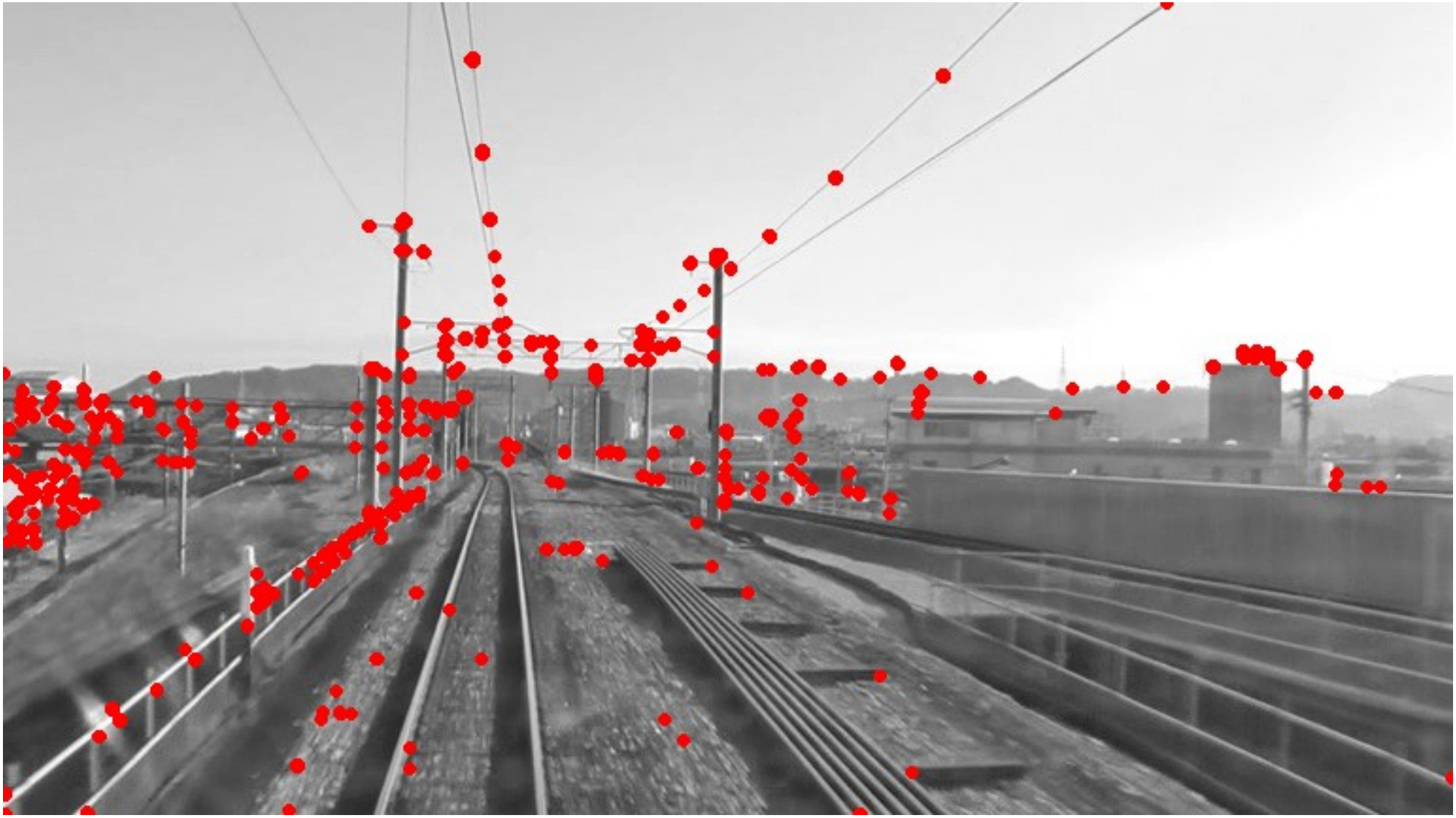
Exemplo (t = 0.5)



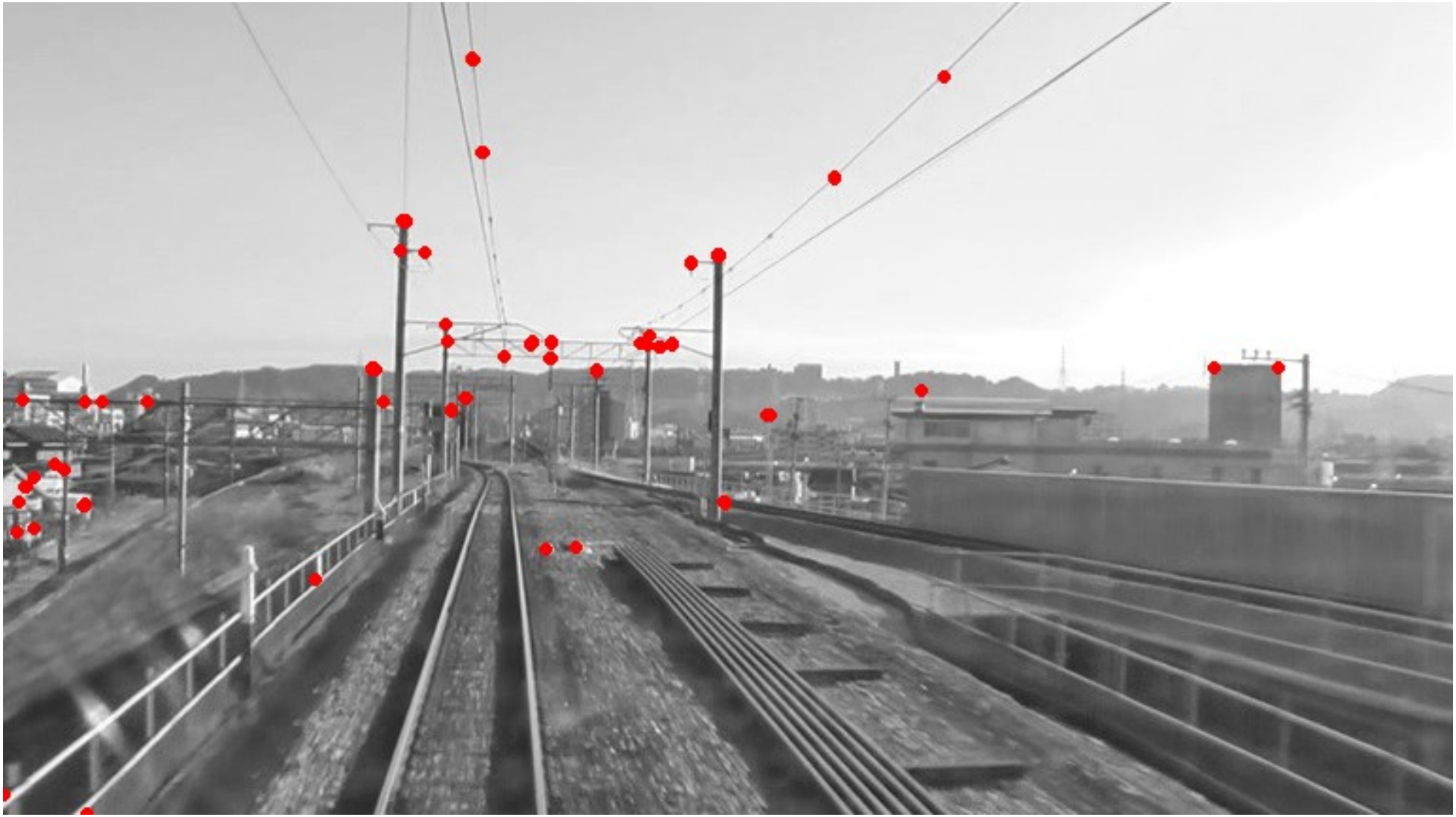
Exemplo (t = 0.75)



Exemplo (t = 0.25)



Exemplo (t = 0.5)



Exemplo (t = 0.75)

Voltando ao fluxo ótico

- Nossa primeira tentativa de usar os cantos detectados para computar o fluxo ótico é bem simples:
  - Detecta cantos em todos os frames.
  - Para cada canto *c0* detectado no frame *f*:
    - Pega uma vizinhança ao redor de *c0*.
    - Para cada canto c1 detectado no frame *f*+1:
      - Se a distância euclideana entre *c0* e *c1* estiver dentro de certos limites (i.e. se *c0* e *c1* não estiverem muito distantes)*:*
        - Pega uma pequena vizinhança ao redor de *c1*.
        - Calcula uma pontuação para o casamento entre *c0* e *c1*.
    - A posição atual de *c0* é o *c1* que produzir o melhor casamento.
- Este algoritmo depende que o detector de cantos seja *estável.*

Voltando ao fluxo ótico

- Nossa primeira tentativa de usar os cantos detectados para computar o fluxo ótico é bem simples:
  - Detecta cantos em todos os frames.
  - Para cada canto *c0* detectado no frame *f*:
    - Pega uma vizinhança ao redor de *c0*.
    - Para cada canto c1 detectado no frame *f*+1:
      - Se a distância euclideana entre *c0* e *c1* estiver dentro de certos limites (i.e. se *c0* e *c1* não estiverem muito distantes)*:*
        - Pega uma pequena vizinhança ao redor de *c1*.
        - Calcula uma pontuação para o casamento entre *c0* e *c1*.
    - A posição atual de *c0* é o *c1* que produzir o melhor casamento.
- Este algoritmo depende que o detector de cantos seja *estável.*
  - Um canto do mundo real detectado no frame *f* deve ser também detectado no frame *f+1*.
  - Esta é uma propriedade importante para detectores de *features*.

Vejamos alguns exemplos...

- Vejamos alguns exemplos...
  - (Os parâmetros foram escolhidos de acordo com o tipo de vídeo).

O algoritmo de Lucas-Kanade

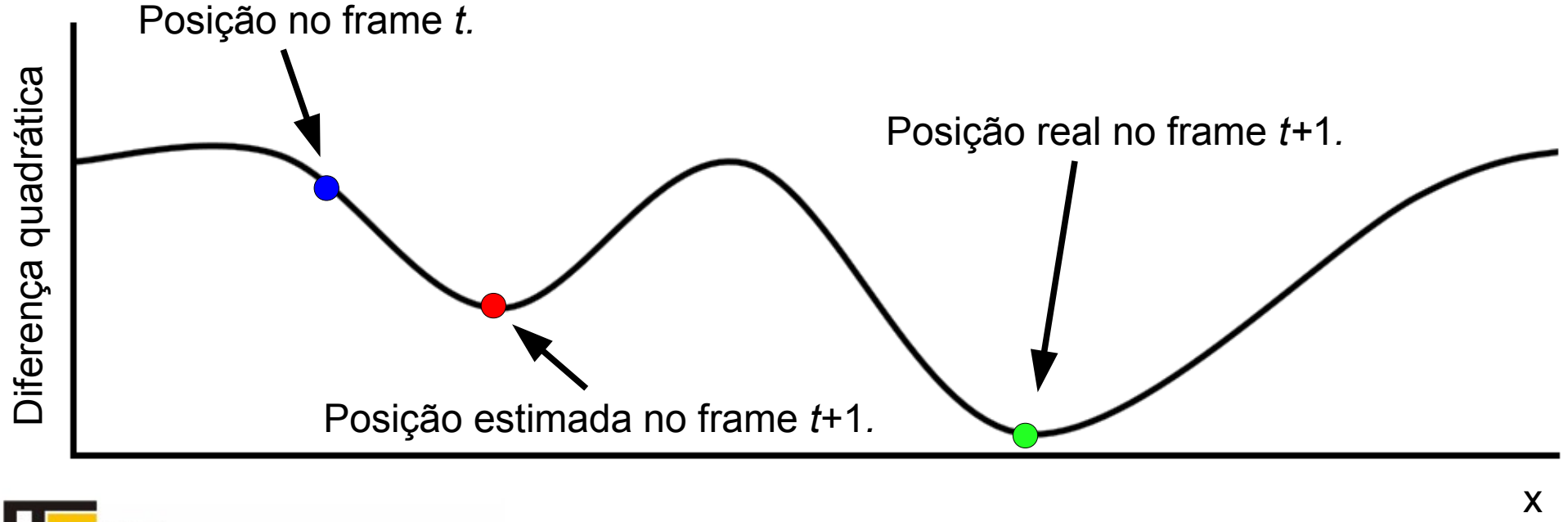
- O algoritmo de Lucas-Kanade (LK) serve para computar o fluxo ótico.
  - Novamente, pressupondo constância de brilho e coerência espacial.
- Conceito:
  - É dado um conjunto de pontos no frame *t*.
  - Considera-se uma janela ao redor de cada ponto.
    - (A janela padrão na implementação do OpenCV tem largura 21).
  - Para cada janela, busca-se no frame *t*+1 uma posição que minimiza uma medida de similaridade (usa-se a diferença quadrática).
    - Podemos começar supondo a mesma posição nos dois frames, e ir "descendo" progressivamente até o "vale" mais profundo.
      - = o algoritmo é iterativo (é baseado no *método de Newton*).
    - A posição contendo o erro mínimo é estimada com precisão subpixel.
- Note que, diferente do nosso algoritmo simples, o LK não detecta pontos no frame *t*+1!

O algoritmo de Lucas-Kanade

- O LK não computa explicitamente a diferença entre “patches”.
  - Da mesma forma que no detector de Harris, esta diferença é manipulada algebricamente, produzindo um sistema cujos componentes são dados por derivadas – mas aqui, também teremos a derivada no tempo (que é a diferença entre os valores dos pixels nos frames *t* e *t*+1).
    - De fato, a matriz de correlação baseada nas derivadas de 2ª ordem é usada aqui também - o LK é como uma evolução do detector de Harris.
- Para mais detalhes, consulte a literatura.

Mínimos locais

- Uma característica importante do algoritmo de Lucas-Kanade é que ele faz uma "descida" iterativa.
  - A cada iteração, a estimativa da posição do pixel no frame *t*+1 é obtida observando uma vizinhança da posição estimada na iteração anterior.
  - Isso quer dizer que, se o deslocamento for maior do que a janela observada, o algoritmo pode ficar preso em um mínimo local!



Pirâmides de imagens

•Para resolver o problema dos mínimos locais, não basta aumentar a largura da janela...

- Por que?

Pirâmides de imagens

- Para resolver o problema dos mínimos locais, não basta aumentar a largura da janela...
  - Uma janela larga demais pode tornar as regiões muito diferentes entre os frames.
- A solução adotada é o uso de pirâmides de imagens:
  - Começamos fazendo a busca em versões reduzidas da imagem.
  - A posição do casamento em um nível da pirâmide (menor) é usada como estimativa inicial para o casamento no próximo nível (maior).
  - Cada nível da pirâmide é uma *oitava*.
    - O número de oitavas normalmente não é muito grande – por exemplo, a implementação do OpenCV usa como padrão 4 oitavas.

Construindo uma pirâmide Gaussiana

- A oitava 0 é a própria imagem de entrada.
- Para gerar a imagem na oitava *i:*
  - Aplica-se uma suavização Gaussiana sobre a imagem da oitava *i-1*.
    - Normalmente com um kernel pequeno (largura 3 ou 5).
  - A imagem filtrada é sub-amostrada – pegamos apenas os pixels nas linhas e colunas pares.
    - Repare que isso é um tipo de interpolação.
- A imagem na oitava *i* terá a metade da altura e da largura da imagem na oitava *i*-1.
  - Se a pirâmide tem *N* oitavas, a imagem original deve ter largura e altura divisíveis por 2^{n-1}.

Exemplo

- Vejamos como se comporta o LK piramidal...
- O OpenCV tem uma implementação do algoritmo LK piramidal.
  - Ele exige que se forneça os pontos cujo fluxo ótico será calculado – para isso, podemos usar novamente a função **`goodFeaturesToTrack`**.

O rastreador KLT

•O algoritmo de Lucas-Kanade serviu como base para diversos trabalhos envolvendo fluxo ótico.

•O rastreador KLT (Kanade-Lucas-Tomasi) é uma evolução do LK piramidal que inclui o conceito de rastreamento.

- Detectar novos pontos a cada frame e buscá-los no frame seguinte não permite conhecer a trajetória dos pixels na sequência completa.
- Se rastreamos cada ponto com base somente na aparência da sua vizinhança no frame anterior, a trajetória pode se desviar progressivamente do pixel original ("*drifting*").
- O KLT compara a região ao redor de um ponto com a primeira aparição daquele ponto, considerando um modelo de distorção afim.
  - Em comparação, o LK considera apenas translações.
- O KLT também pode usar uma estimativa da posição do ponto, que pode ser obtida projetando a sua trajetória passada.

Finalizando

- Cantos são *features* locais (baseadas em vizinhanças do pixel).
  - O detector de Harris detecta as *features*, mas não as descreve.
  - O nosso algoritmo simples descreve as *features* simplesmente como um "patch" ao redor do ponto central.
  - O LK e o KLT não criam descritores para as *features* explicitamente.
- Note que os algoritmos que vimos são invariantes a translações, e robustos a variações pequenas de posição dentro de um "patch", mas não contemplam transformações de perspectiva, rotação e escala.
  - O KLT tem alguma robustez a variações de escala e rotação, incluídas no modelo de distorção afim, mas ainda trabalha com translações entre frames consecutivos.

Finalizando

- Detecção e rastreamento de cantos estão entre os problemas mais fundamentais em visão computacional.
  - É uma instância de um problema mais geral, que é a detecção de "pontos de interesse" ou "pontos chave".
    - Cantos são pontos de interesse, mas nem todo ponto de interesse é um canto.
- Um bom detector de pontos de interesse deve ser estável.
- Detecção/rastreamento de cantos é útil para várias aplicações.
  - Ex: visão estéreo.
    - Na busca por correspondências entre duas imagens da mesma cena.