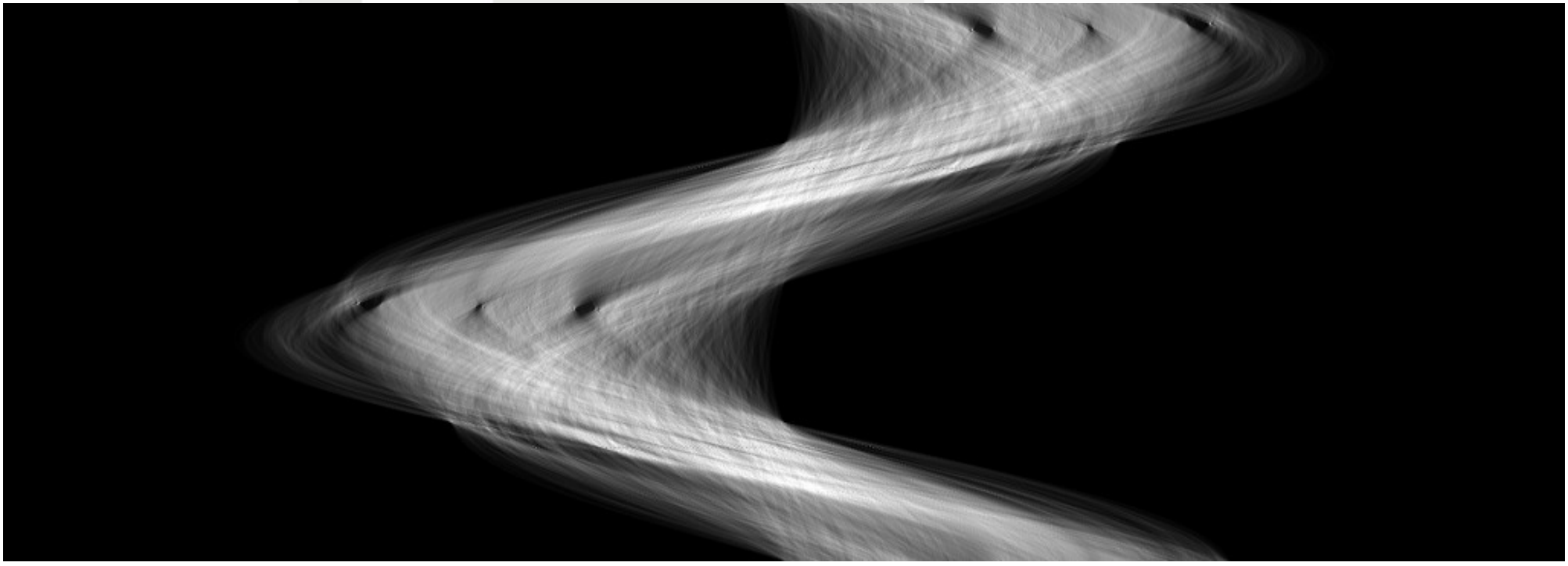
Processamento Digital de Imagens

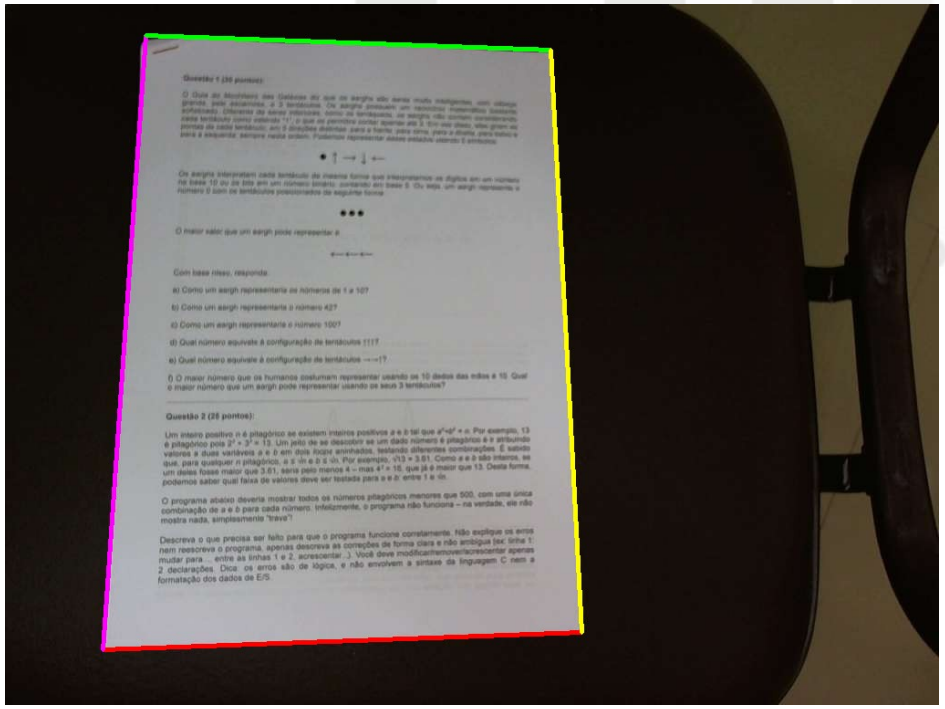
Prof. Bogdan Tomoyuki Nassu

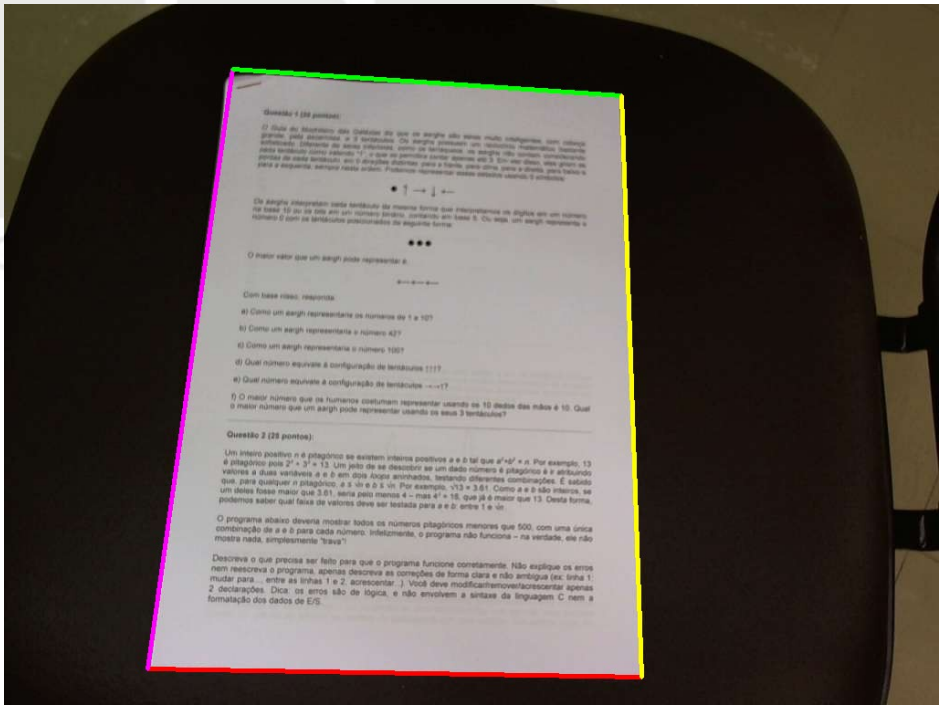


UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Desafio

- Dada uma foto de um documento, encontrar os limites da página.
  - Informação usada, por exemplo, por alguns programas que fazem reconhecimento de texto em fotografias.
  - Suponha que a página está (quase) toda na imagem, e que existe contraste forte entre a página e a superfície sobre a qual ela está.
  - Como podemos resolver o problema?





Encontrando os limites...

- Ideia geral do nosso algoritmo:
  - Encontra bordas alongadas e aproximadamente retas na imagem.
    - Representa estas bordas por equações, como retas no plano.
  - Seleciona 4 das retas encontradas, passando sobre as bordas superior, inferior, esquerda e direita.
  - Os pontos de encontro entre as retas horizontais e verticais são os cantos da página.

- Primeiro desafio: localizar retas em uma imagem, representando-as como entidades geométricas.

Detecção de Bordas

- "**Encontra bordas** alongadas e aproximadamente retas."
  - Problema #1: detectar bordas.
- O conceito de borda foi introduzido há algumas aulas.
  - O que caracteriza uma borda?

Detecção de Bordas

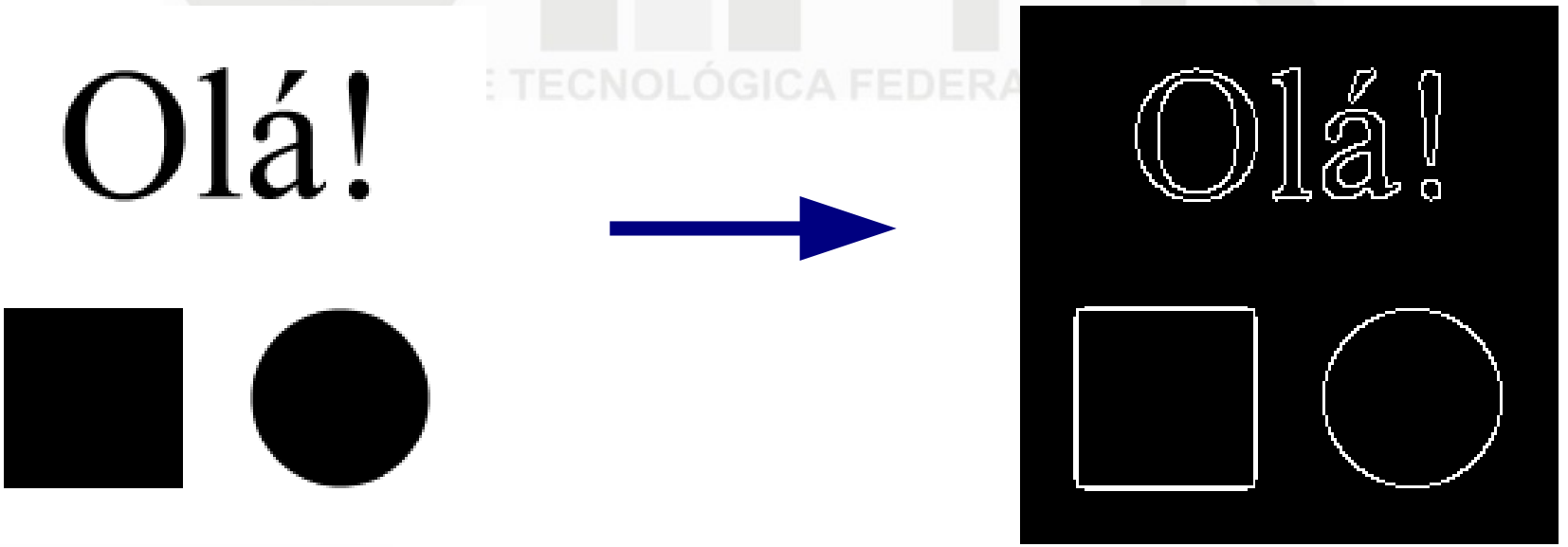
- "**Encontra bordas** alongadas e aproximadamente retas."
  - Problema #1: detectar bordas.
- O conceito de borda foi introduzido há algumas aulas.
  - O que caracteriza uma borda?
    - R: descontinuidade na intensidade.
  - Como podemos medir descontinuidades na intensidade?

Detecção de Bordas

- "**Encontra bordas** alongadas e aproximadamente retas."
  - Problema #1: detectar bordas.
- O conceito de borda foi introduzido há algumas aulas.
  - O que caracteriza uma borda?
    - R: descontinuidade na intensidade.
  - Como podemos medir descontinuidades na intensidade?
    - R: através dos gradientes.

Detecção de Bordas

- Detecção de bordas é um problema clássico em PDI.
  - Existem várias técnicas e formulações. Consideraremos:
    - Entrada: uma imagem (colorida ou em escala de cinza).
    - Saída: uma imagem binária, com as bordas “setadas”.
  - Exemplos de aplicações:
    - Reconhecimento de caracteres.
    - Detecção de formas geométricas.
    - Reconhecimento de impressões digitais.

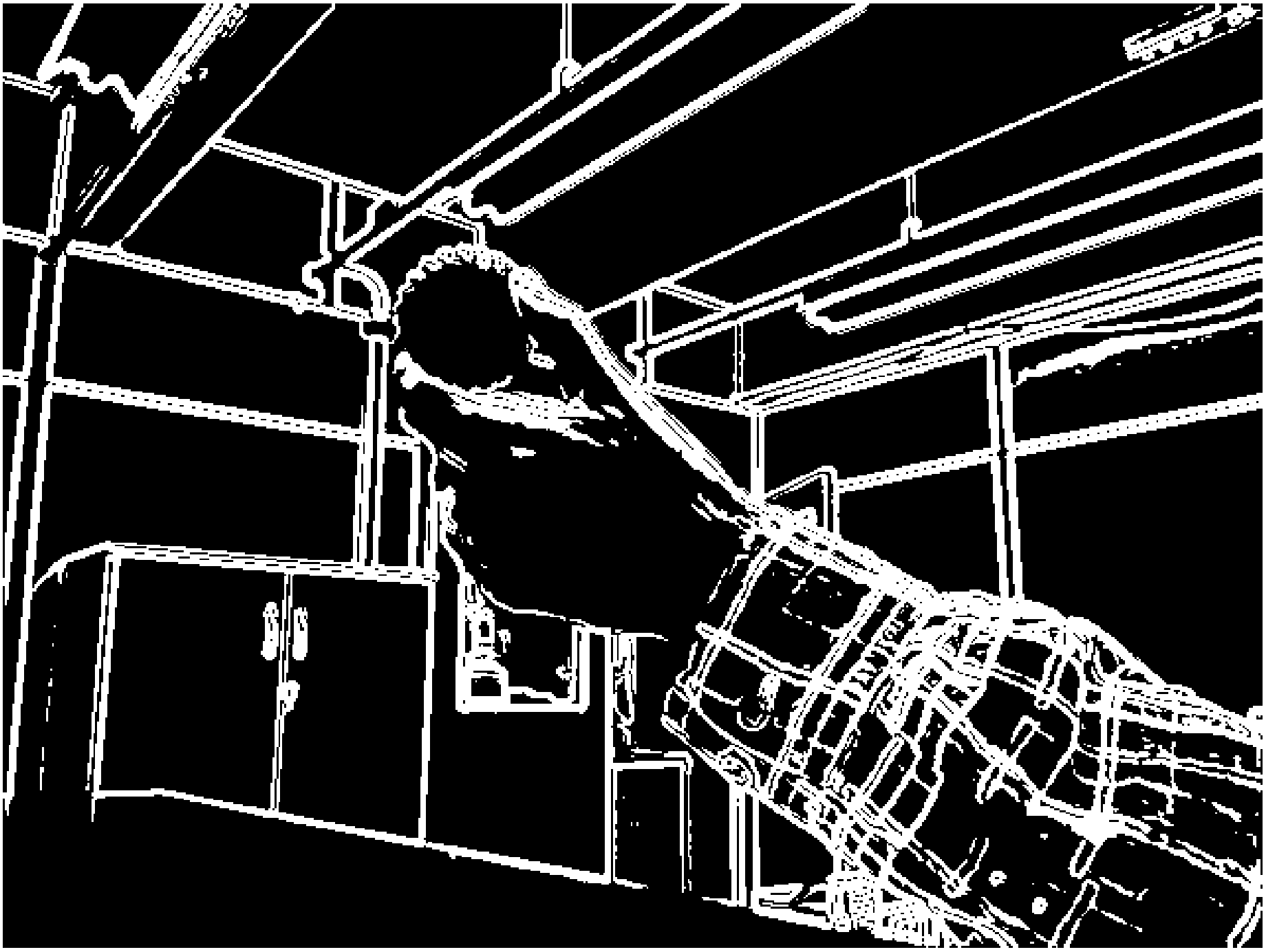


Ideias?

- Alguma ideia?

Ideia #0

- Ideia #0: limiarização com base na magnitude dos gradientes.



Problemas...

•Problema: os gradientes podem ficar fortes em vários pixels vizinhos, fazendo as bordas aparecerem muito “grossas”.
  - Isto é ruim para vários algoritmos que analisam bordas.

•Como “afinar” as bordas?

Afinando as bordas

•Como “afinar” as bordas?

•Existem vários algoritmos de afinamento (*thinning*), ou esqueletonização (*skeletonization*).

- Conceito: remover iterativamente pixels brancos, evitando:
  - Remover “terminações” de linhas.
  - Separar um componente conexo.

Algoritmo de Zhang-Suen

- Um algoritmo clássico de afinamento:

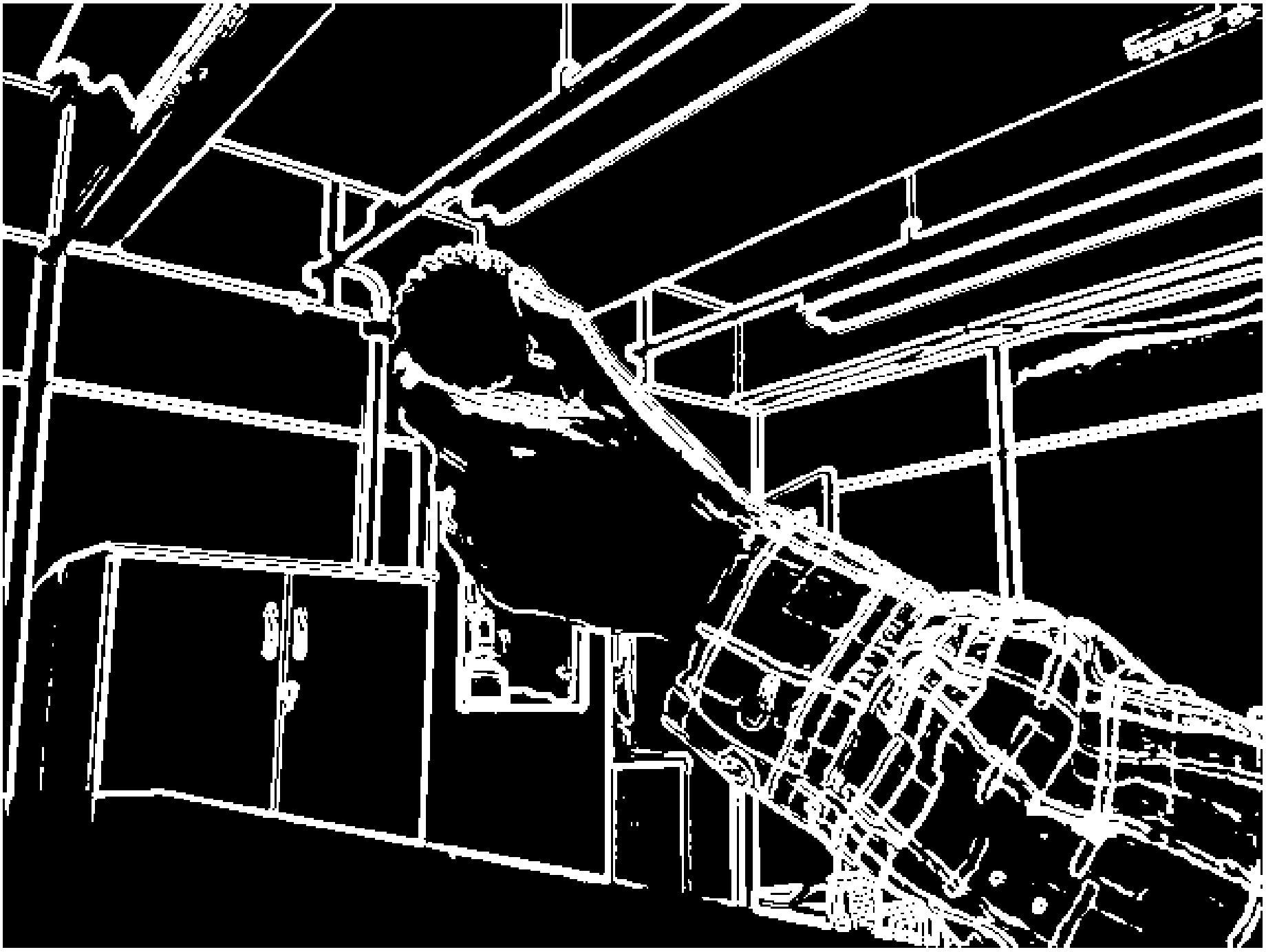
```
enquanto a imagem estiver mudando, faça:
    para cada pixel P1 setado na imagem
        A = número de transições 0-1 dando um giro ao redor de P1
        B = número de vizinhos de P1 que estão setados
        C = P2*P4*P6 em iterações ímpares, P2*P4*P8 nas pares
        D = P4*P6*P8 em iterações ímpares, P2*P6*P8 nas pares

        se (A = 1 AND 2 <= B <= 6 AND C = 0 AND D = 0)
            marca que P1 deve ser apagado

    apaga os pixels que foram marcados
```

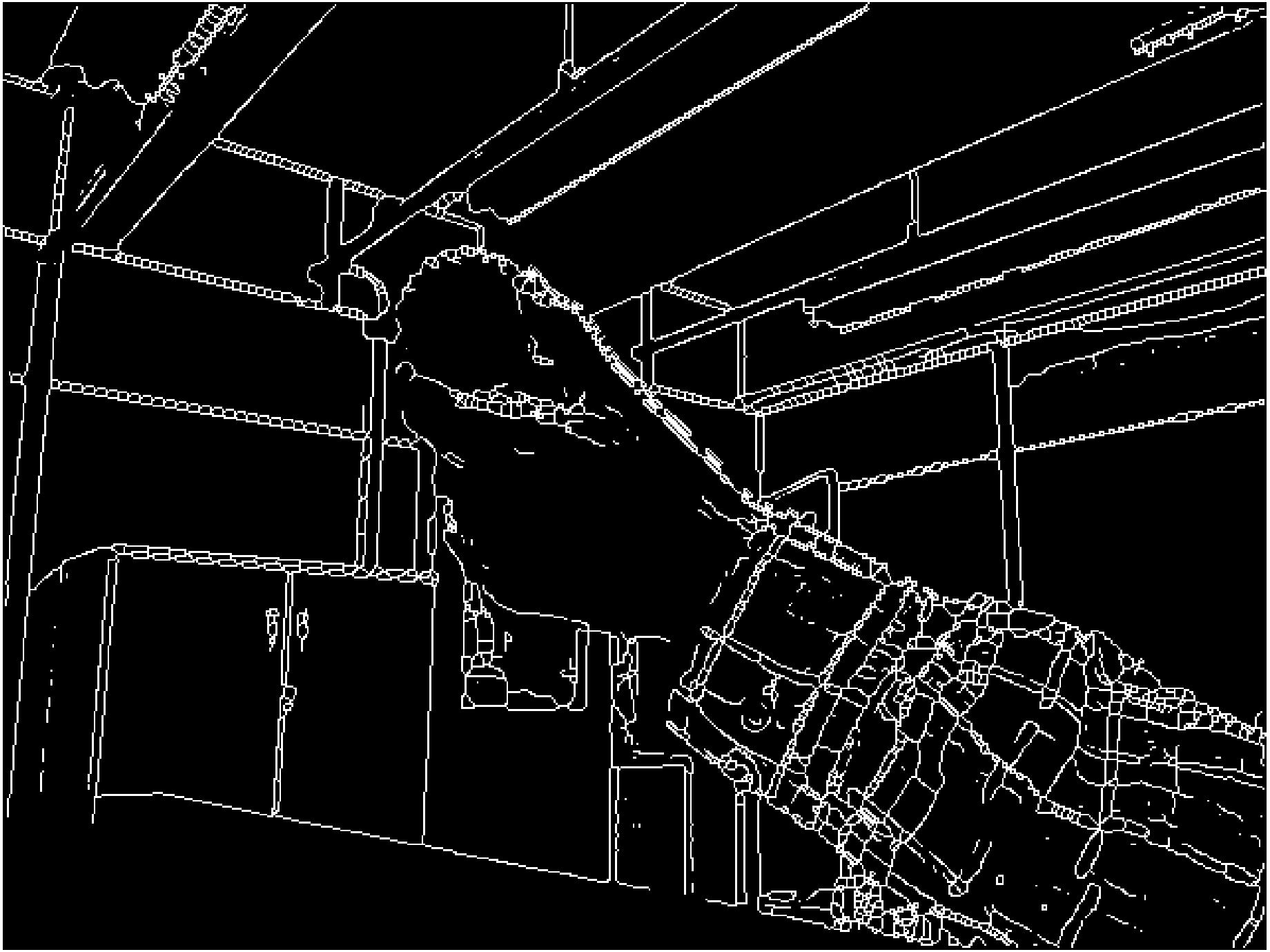
A vizinhança do pixel é interpretada assim.

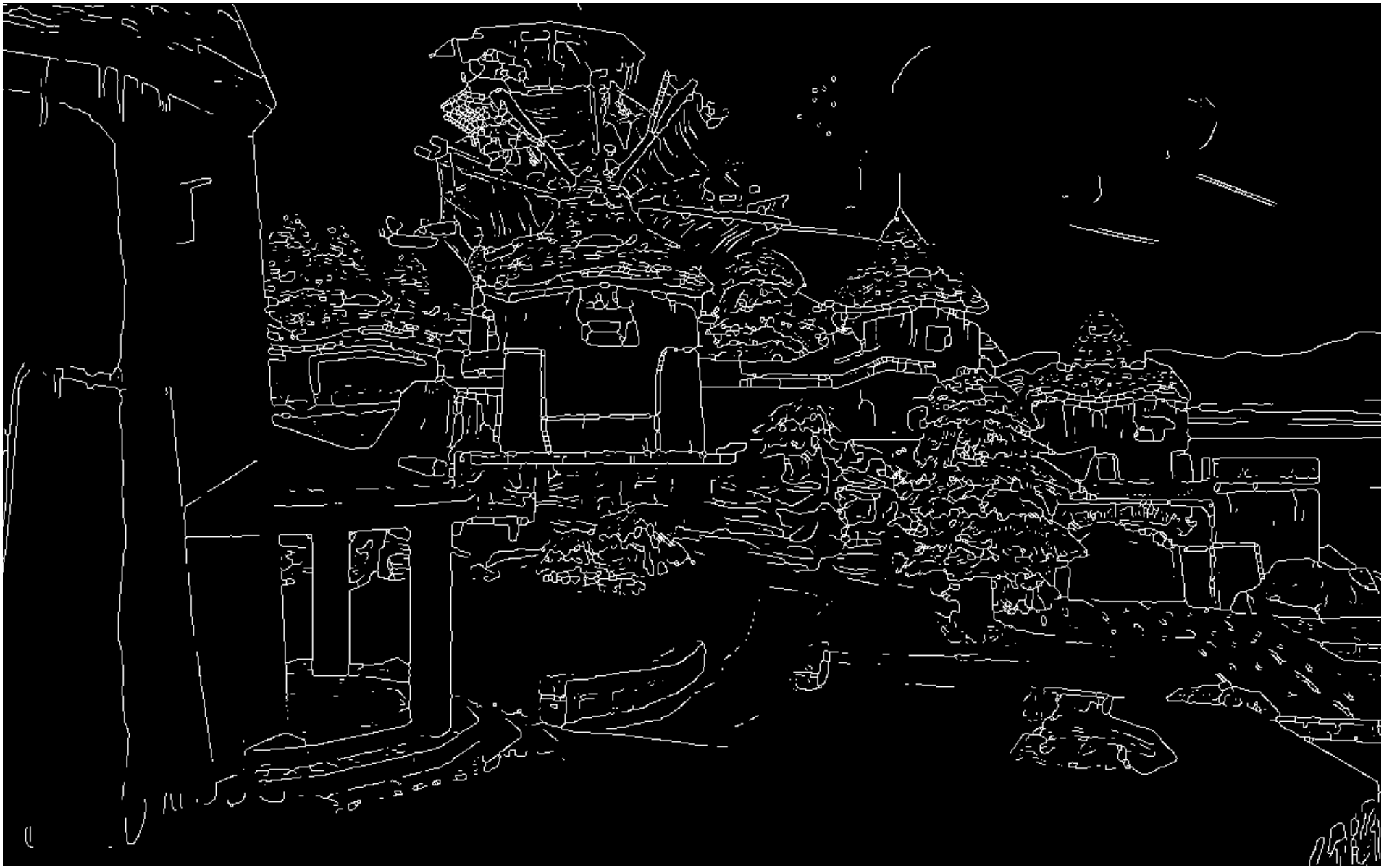
P9	P2	P3
P8	P1	P4
P7	P6	P5

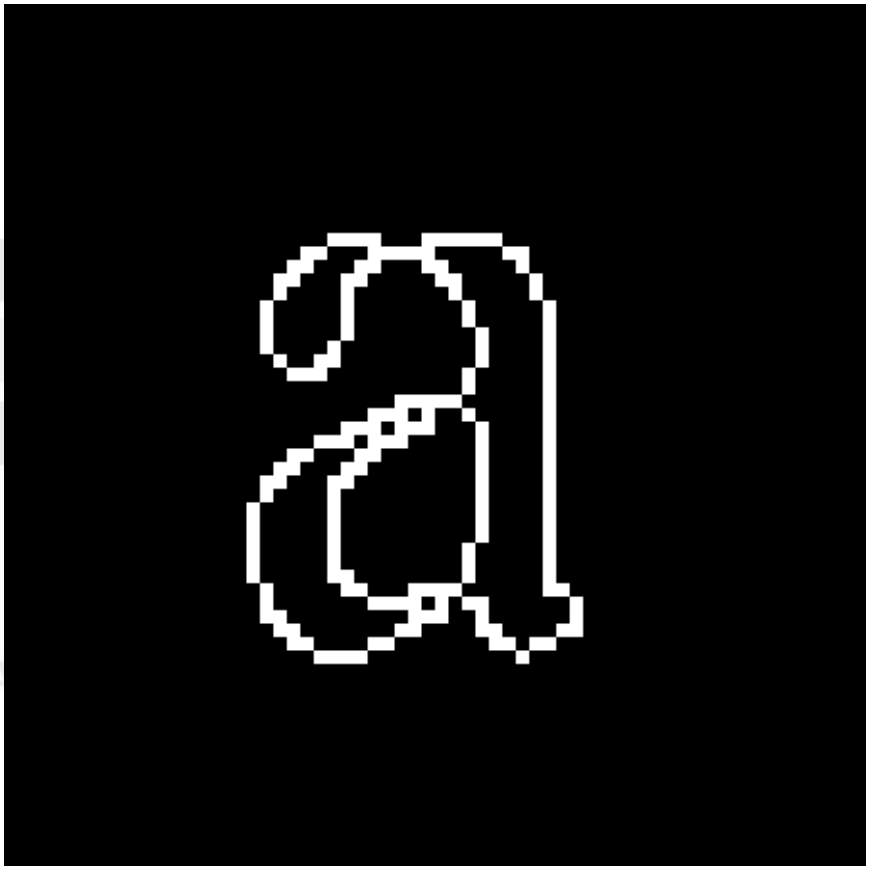
Algoritmo de Zhang-Suen

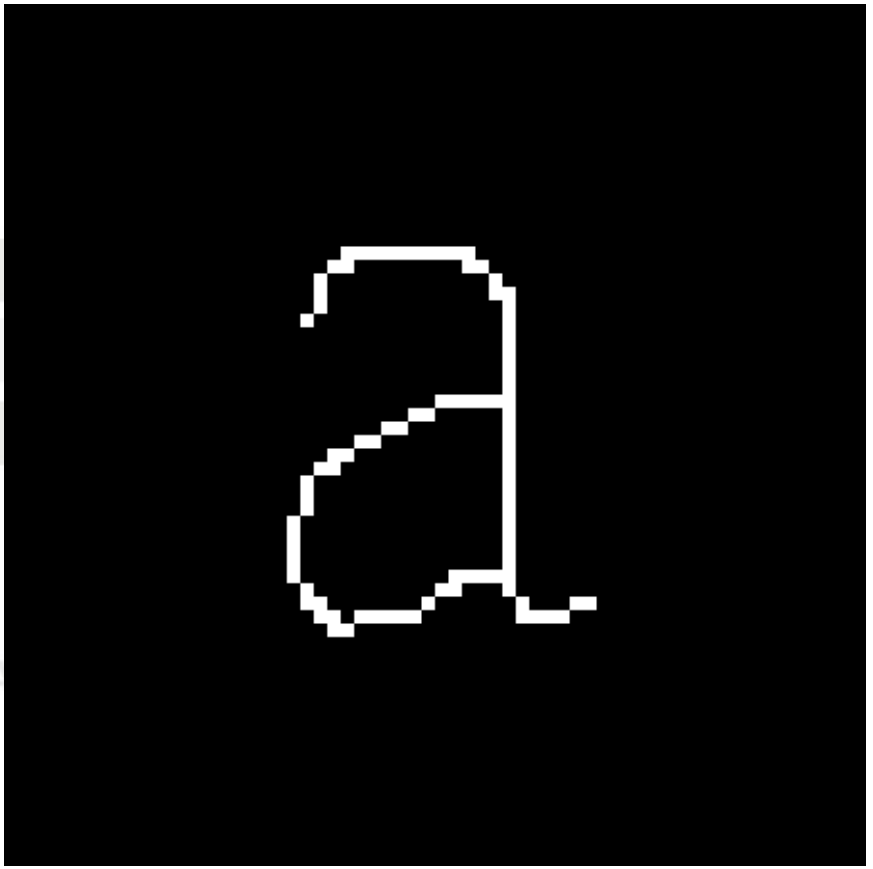
- Ideia: remover a cada iteração pixels setados que têm vários outros pixels setados como vizinhos, evitando remover certos padrões que caracterizam linhas, entroncamentos e terminações.
- Não vamos analisar este algoritmo a fundo.
- Existem vários outros algoritmos baseados no mesmo princípio.



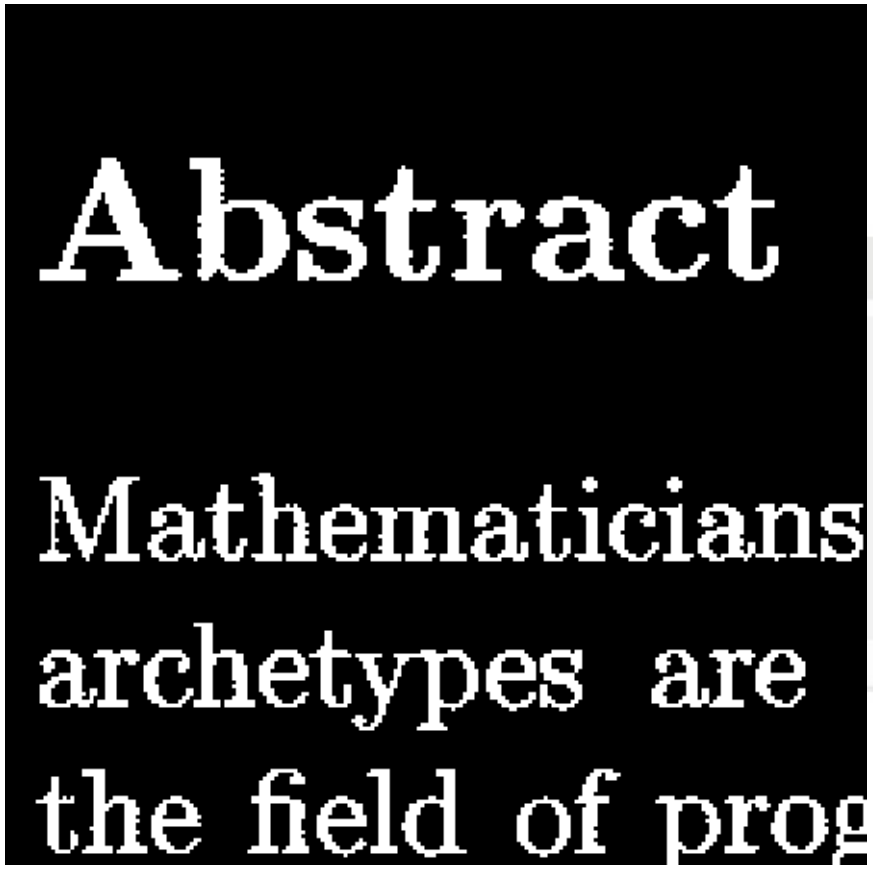


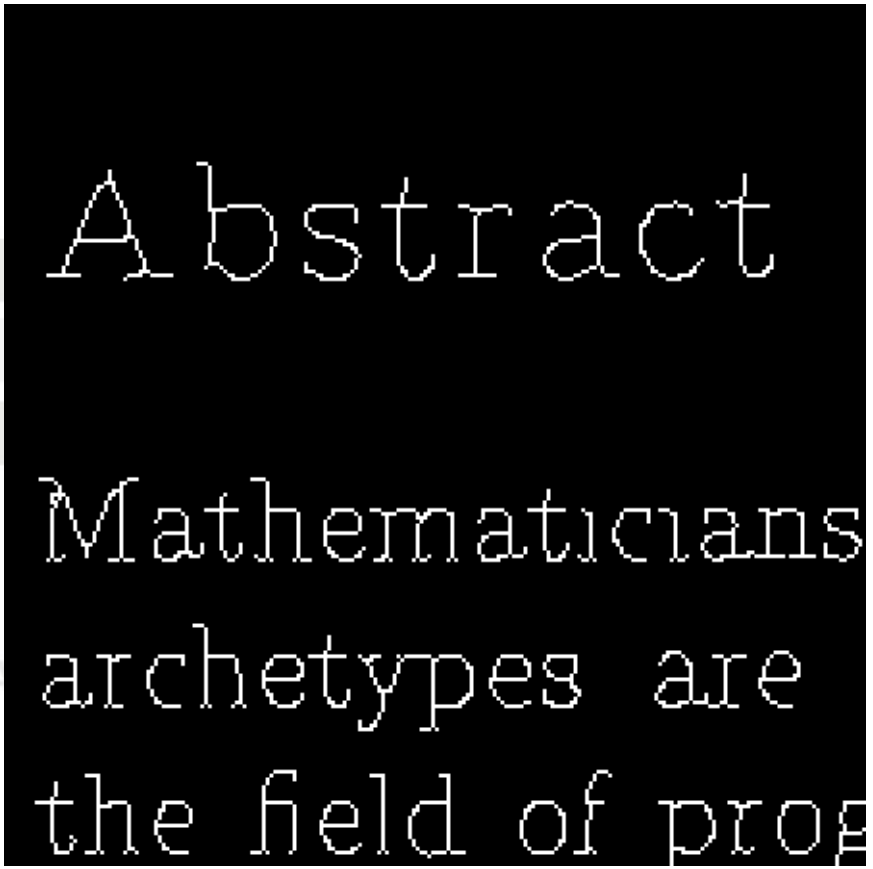






Técnicas de afinamento não servem somente para imagens de bordas!





Técnicas de afinamento não servem somente para imagens de bordas!

Esqueletonização / afinamento

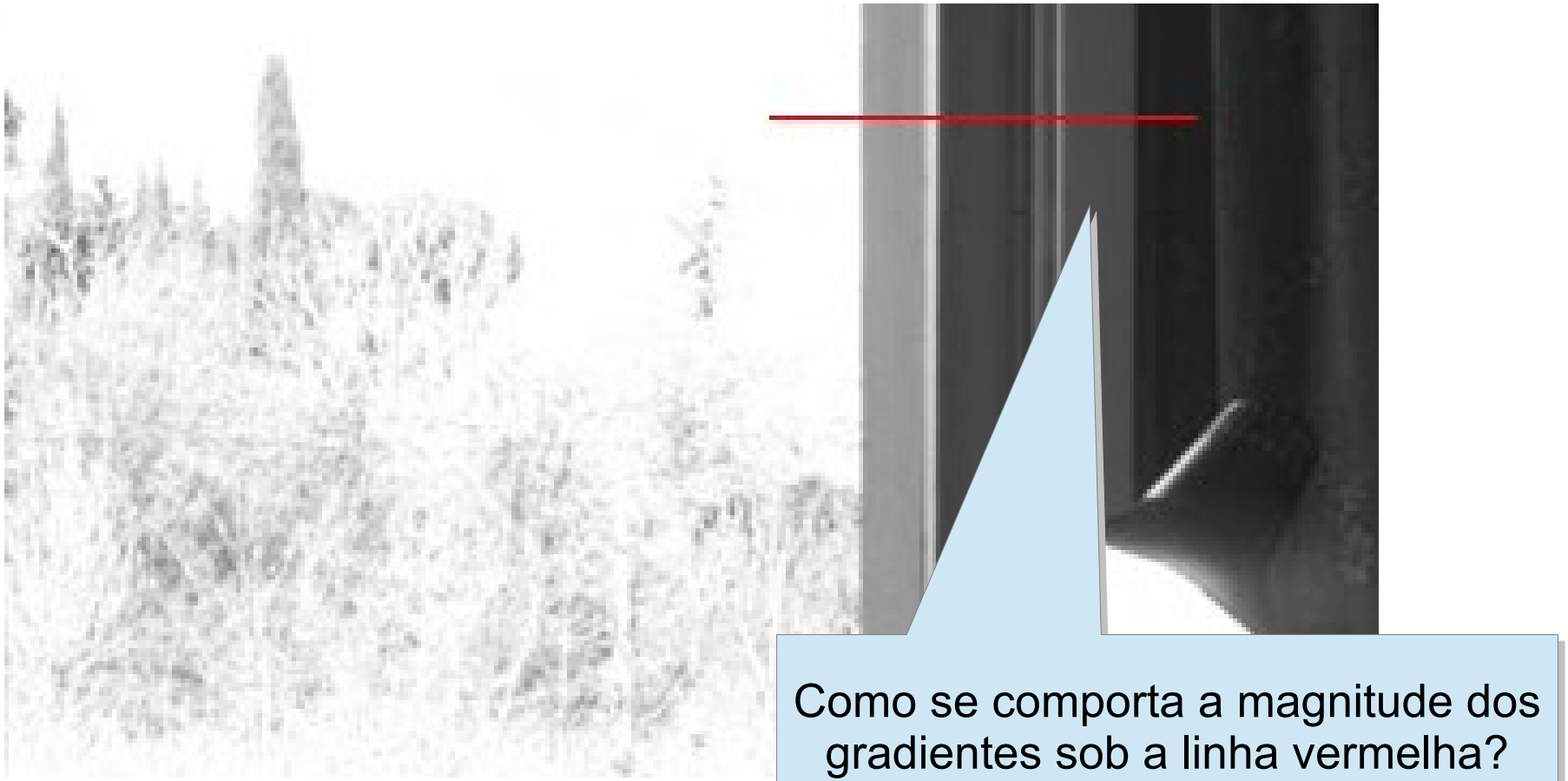
•Algoritmos de afinamento funcionam bem em certas ocasiões, mas costumam não ser a melhor alternativa para afinar bordas...

•Usar os gradientes é um caminho promissor, mas precisamos usá-los de forma diferente...

- Como?

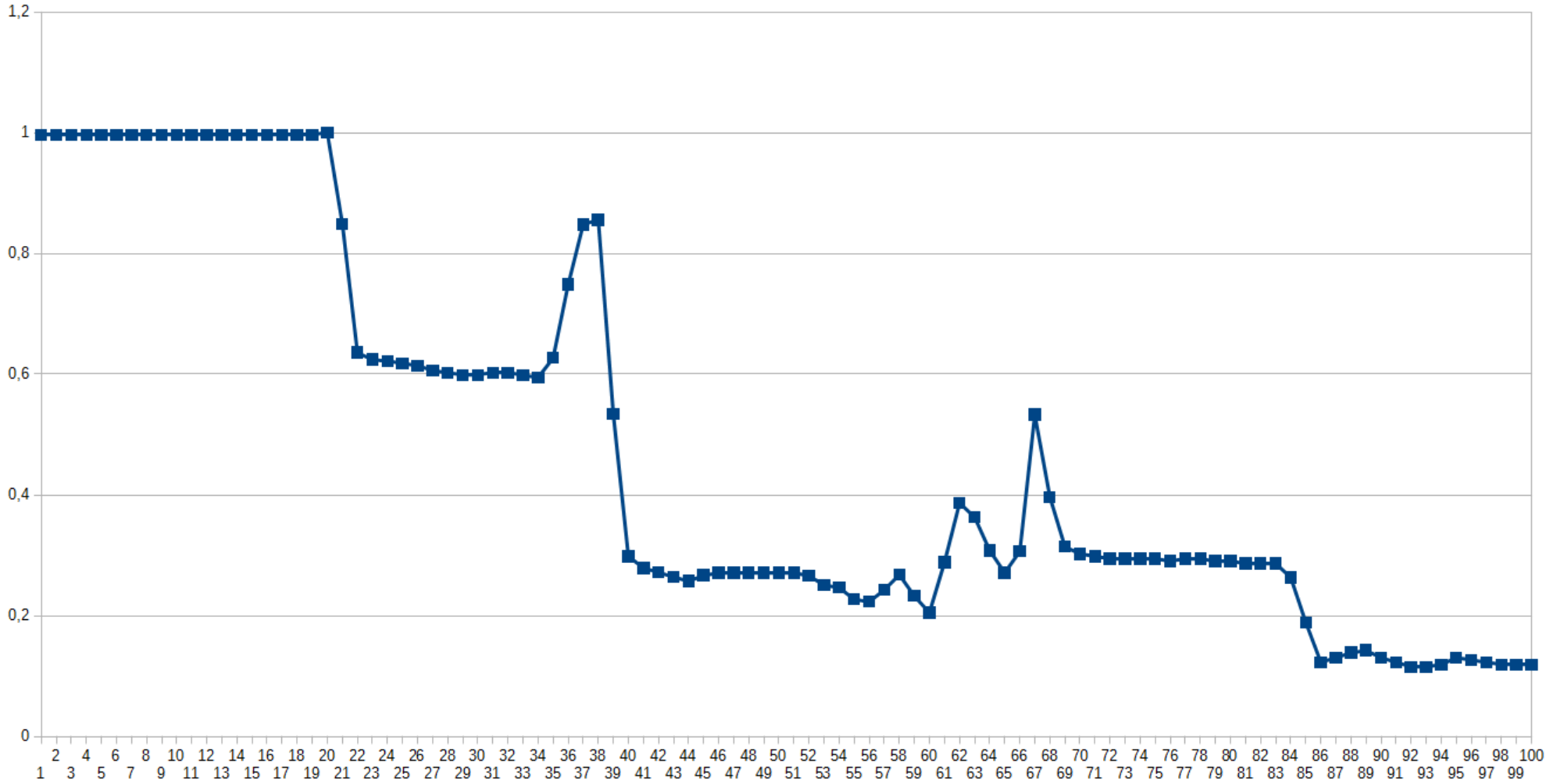
Hmmm...

- Considere uma fotografia de um objeto real.
  - A fronteira entre dois objetos aparece ligeiramente borrada.

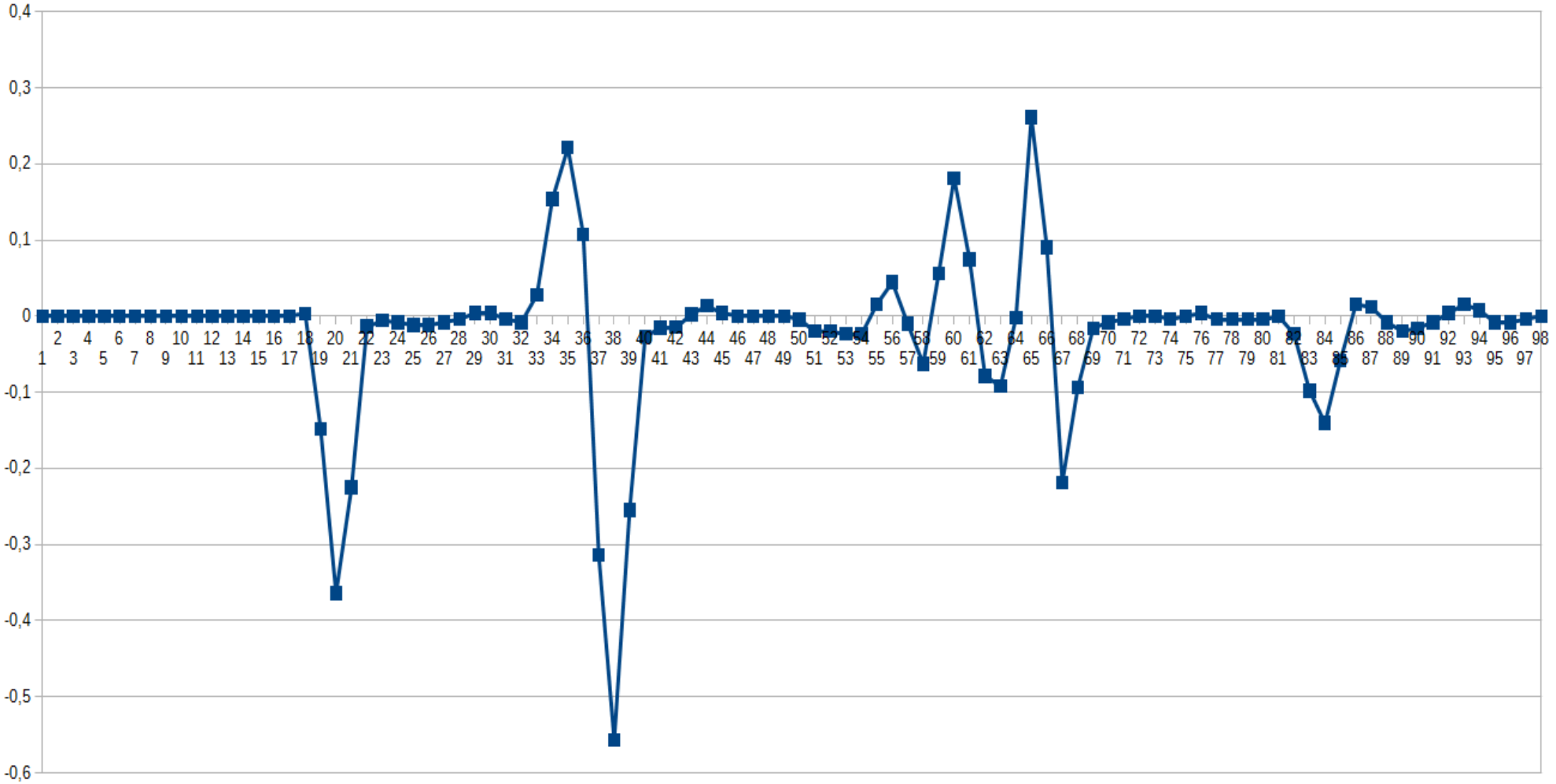


Como se comporta a magnitude dos gradientes sob a linha vermelha?

Intensidades...



Dx


0,4
0,3
0,2
0,1
0
-0,1
-0,2
-0,3
-0,4
-0,5
-0,6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98

Onde estão as bordas?

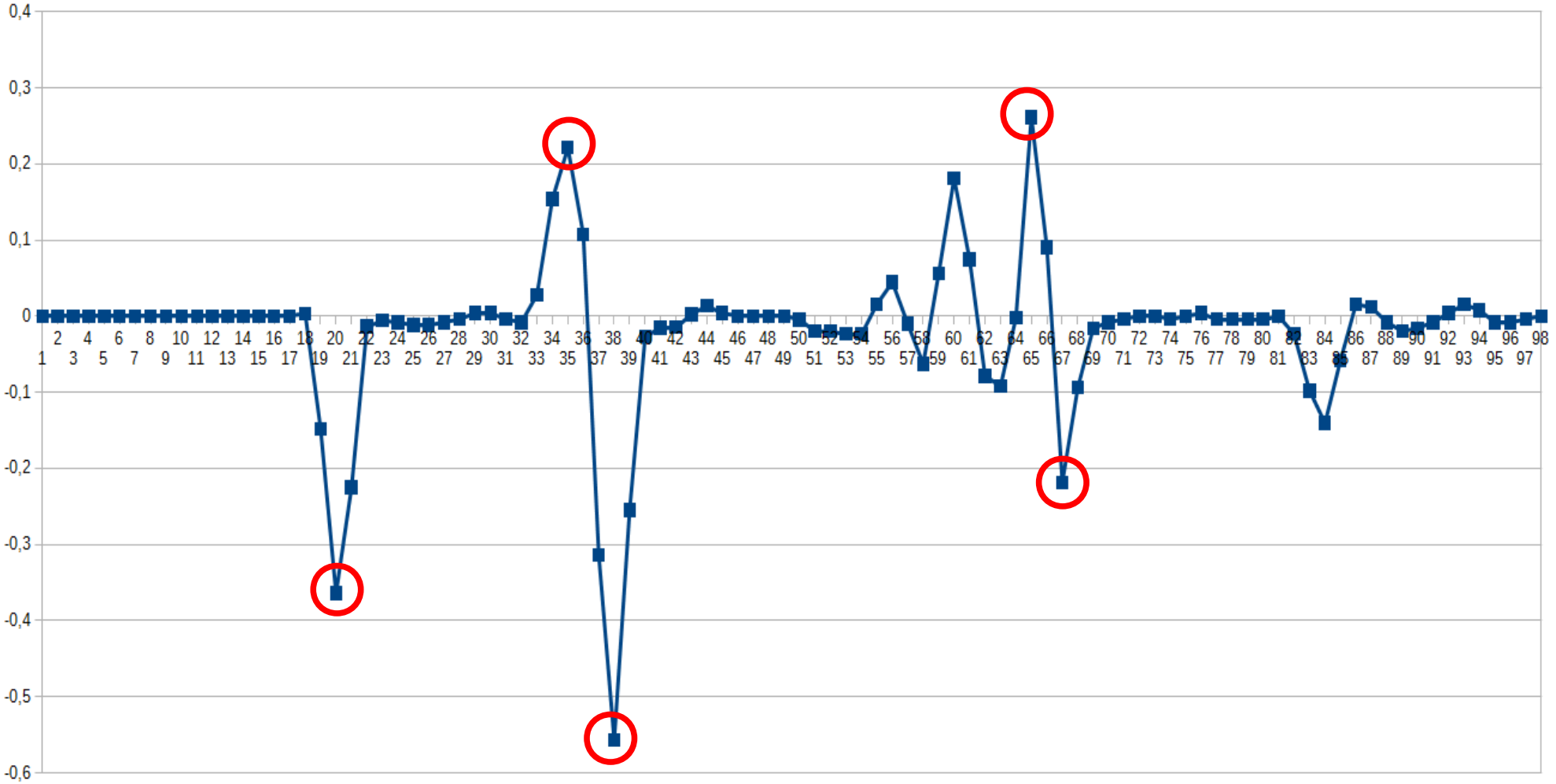
- Supondo que Dy = 0 para todos os pixels (i.e. a magnitude dos gradientes é igual a |Dx|), onde devemos marcar as bordas?

Onde estão as bordas?

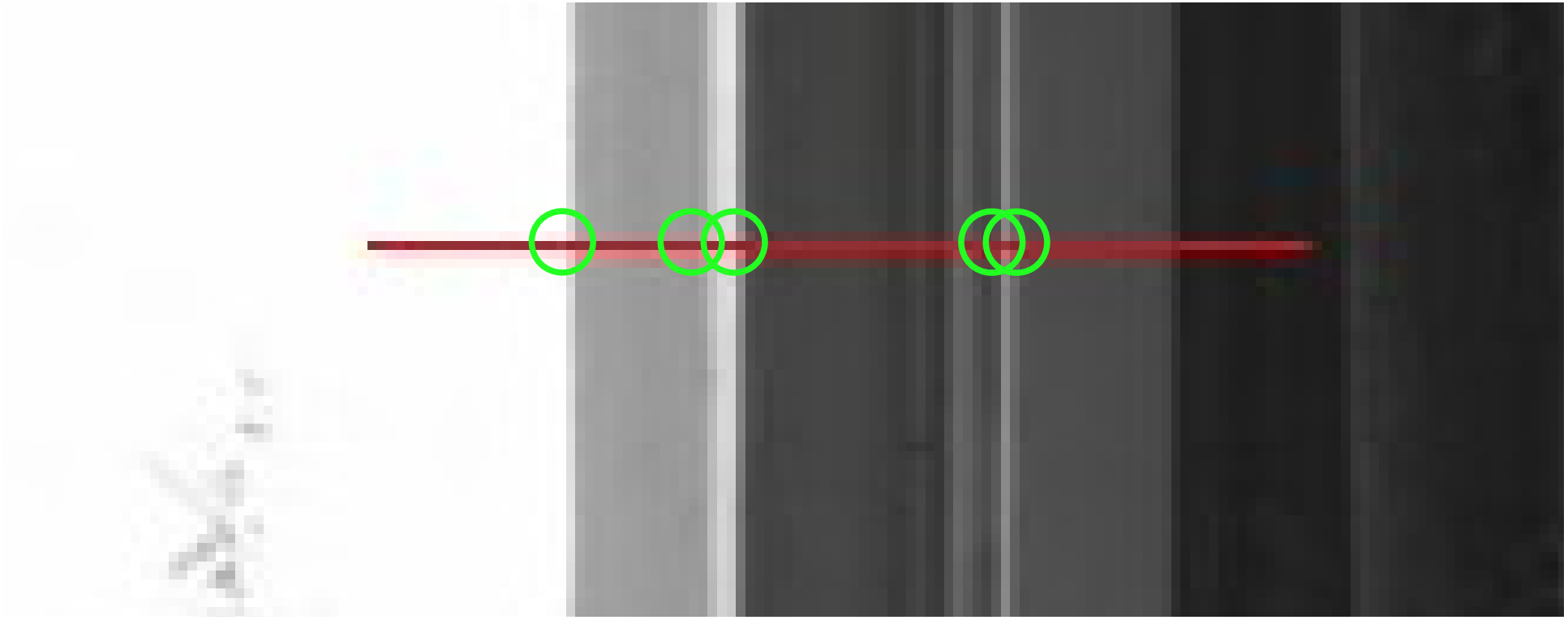
•Supondo que Dy = 0 para todos os pixels (i.e. a magnitude dos gradientes é igual a |Dx|), onde devemos marcar as bordas?

- R: as bordas estão nas posições que:
  - Possuem magnitude alta.
  - São máximos locais.

Máximos locais (mag > 0.2)



Máximos locais (mag > 0.2)



Máximos locais em que direção?

- Devemos selecionar pontos cuja magnitude do gradiente é um máximo local... em qual direção?
- Dica: analise bem o exemplo anterior...

Máximos locais em que direção?

•A orientação do gradiente em cada pixel dá uma dica sobre a direção que devemos analisar:

- Bordas verticais: orientação próxima de 0º ou 180º.
- Bordas horizontais: orientação próxima de 90º ou 270º.
- Linha com ângulo de xº: orientação perpendicular a xº.

Detector de bordas de Canny

- Algoritmo clássico: detector de bordas de Canny.

Computa a magnitude e orientação dos gradientes

Quantiza a orientação em 4 direções:

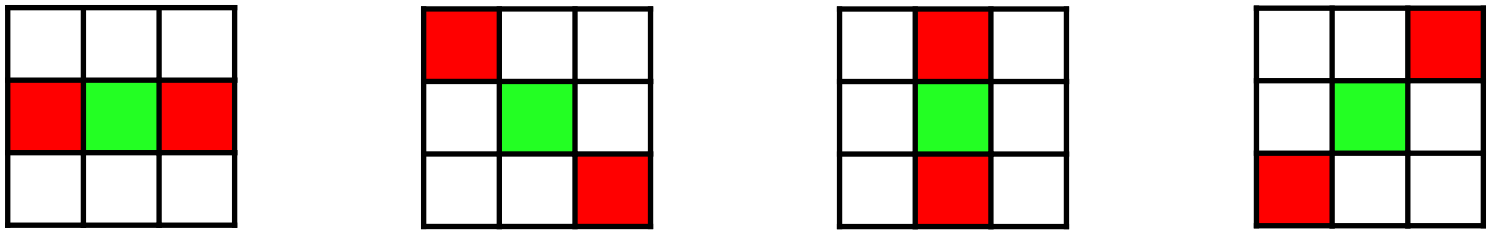
- **(quase) horizontal**
- **(quase) vertical**
- **(quase) diagonal com a direita acima**
- **(quase) diagonal com a esquerda acima**

Elimina pontos que não são máximos locais na direção do gradiente

Limiarização com histerese

Detector de bordas de Canny

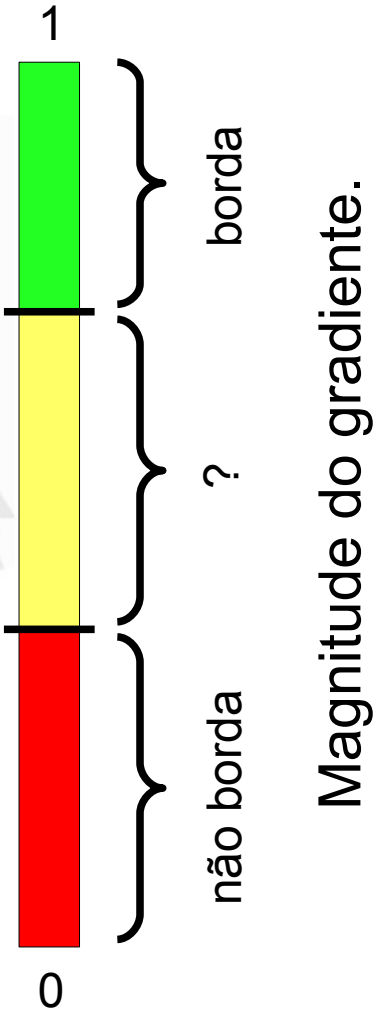
- Computa a magnitude e orientação dos gradientes:
  - Já fizemos isso!
- Quantiza a orientação em 4 direções:
  - Teste simples, independente para cada pixel, com certos intervalos.
    - (quais?)
- Elimina pontos que não são máximos locais.
  - *Non-maxima suppression*.
  - Testamos vizinhanças simples:

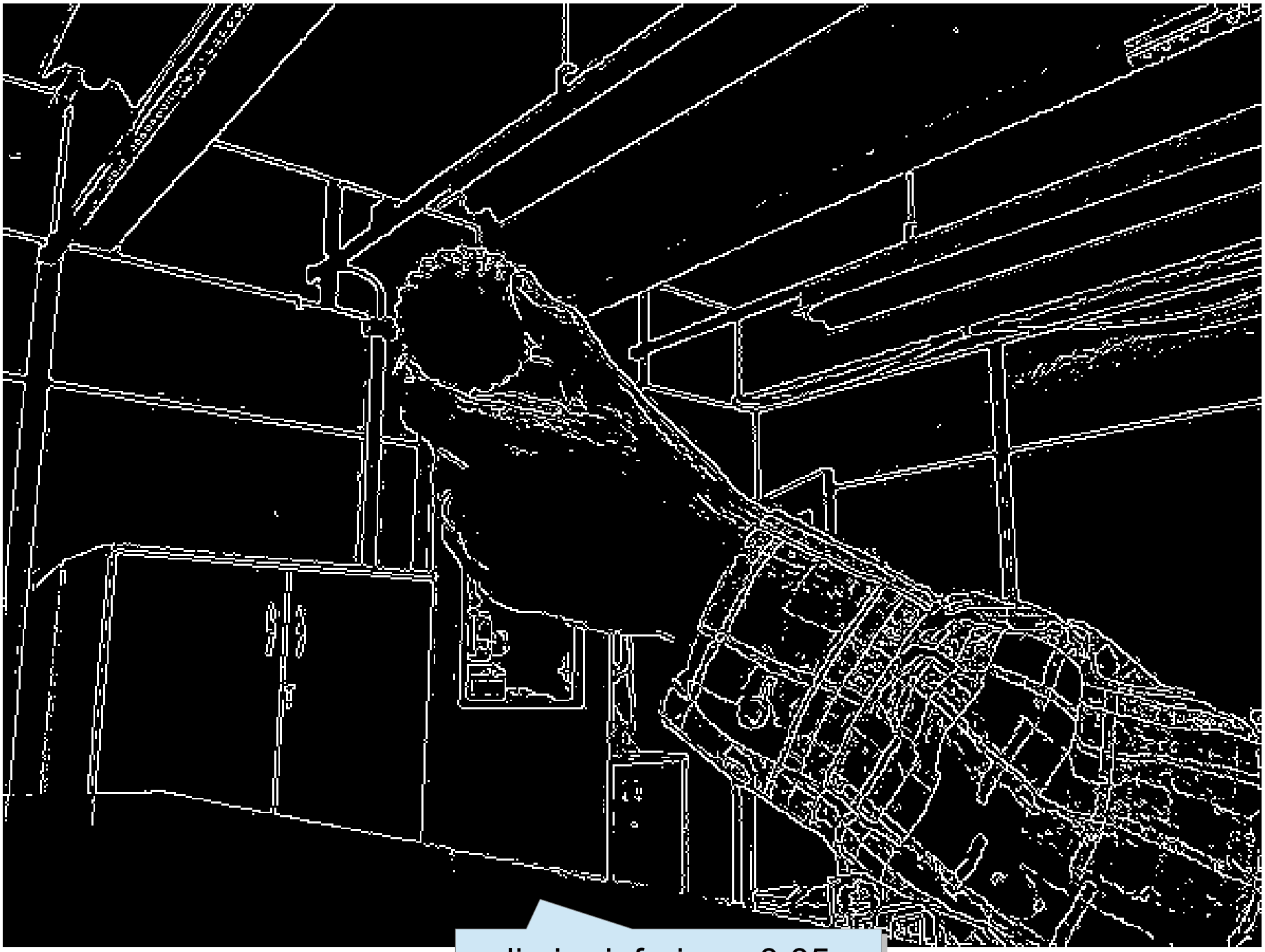


Cuidado: lembre-se que em imagens o eixo *y* é invertido, ou seja, os ângulos “crescem para baixo”.

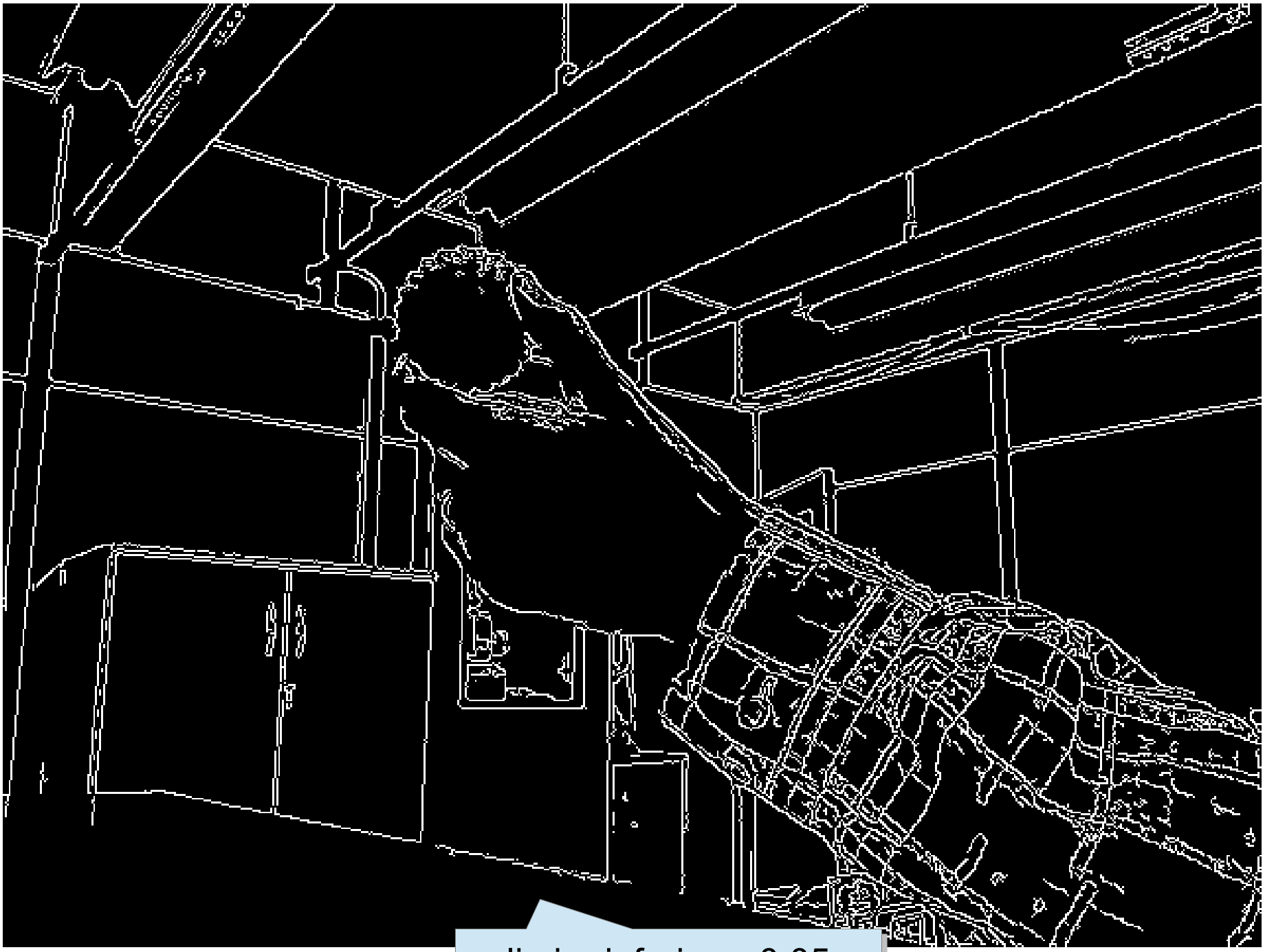
Detector de bordas de Canny

- Limiarização com histerese: usa 2 limiares.
  - Magnitude acima do limiar superior → é borda.
  - Magnitude abaixo do limiar inferior → não é borda.
  - Magnitude entre os dois limiares → borda, se houver um caminho conectando o pixel a outro pixel de borda.
    - Isso pode ser implementado usando o mesmo princípio da inundação, com vizinhança-8.
- A limiarização com histerese ajuda a reduzir ambiguidades e a evitar bordas “quebradas”.

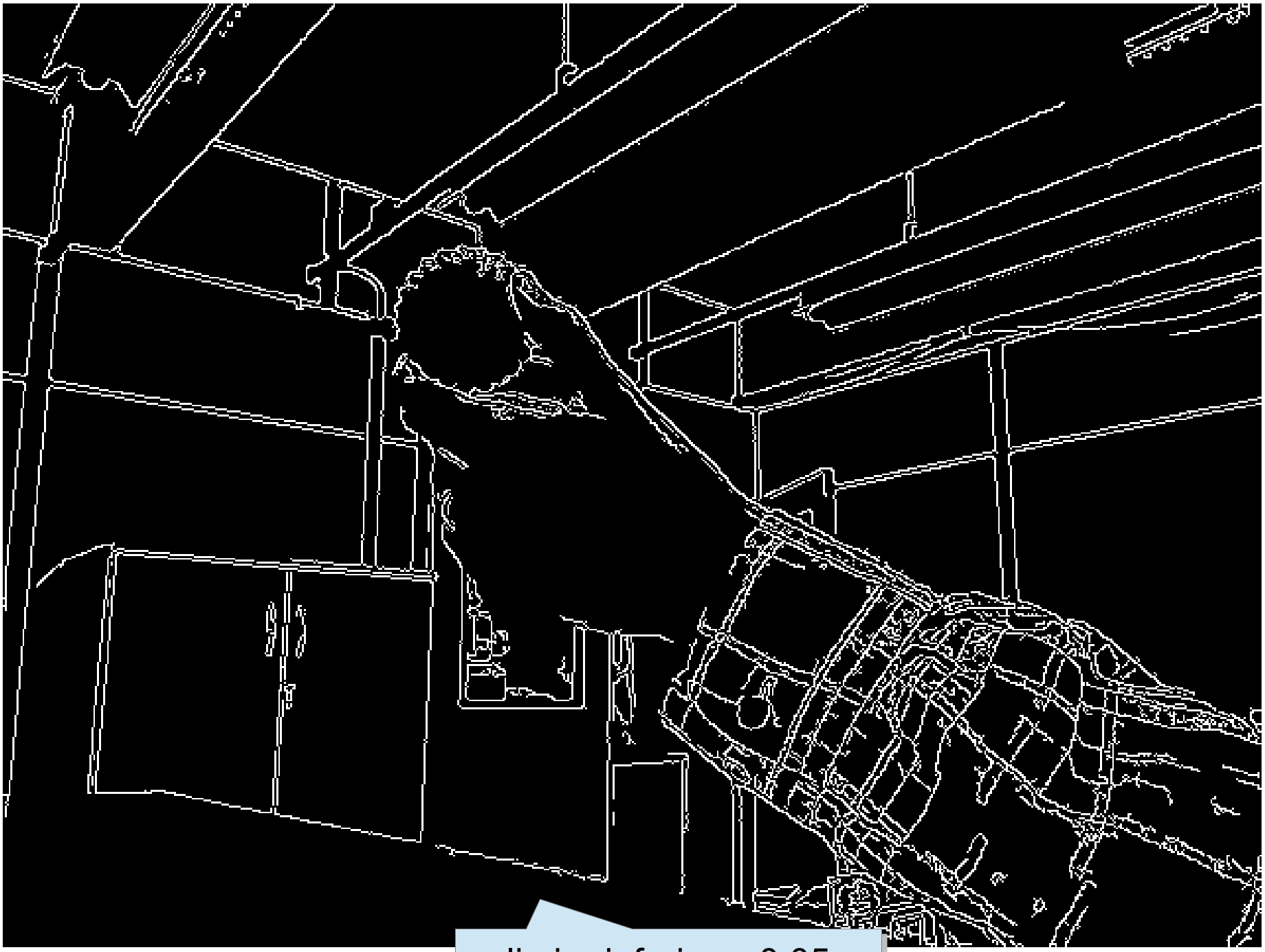




limiar inferior = 0.05
limiar superior = 0.05



limiar inferior = 0.05
limiar superior = 0.1



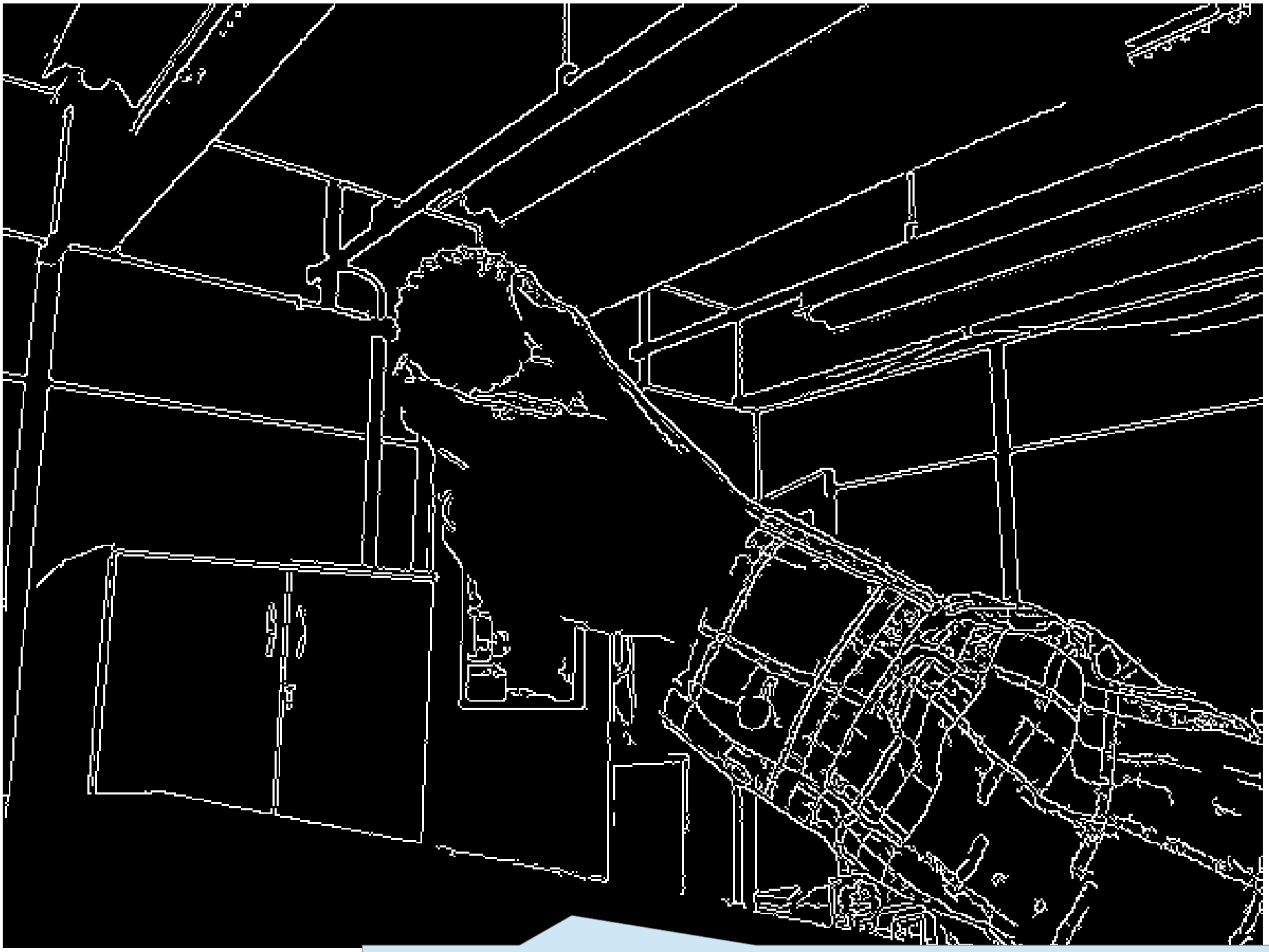
limiar inferior = 0.05
limiar superior = 0.15

Imagens coloridas

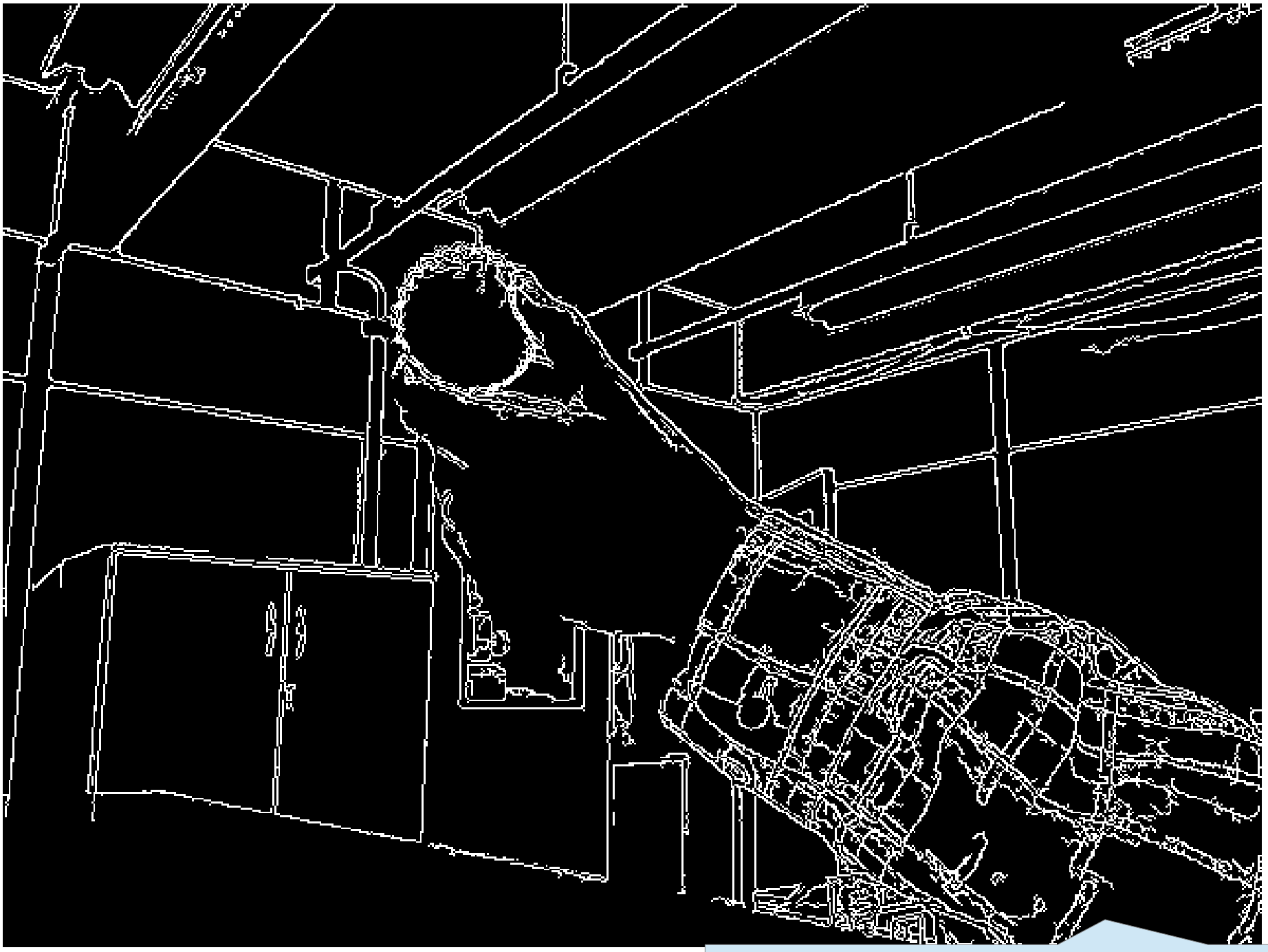
•Gradientes e derivadas não são conceitos bem definidos para imagens coloridas.

•Podemos usar técnicas simples, como:

- Converter a imagem de entrada para escala de cinza.
- Computar o gradiente para os 3 canais, selecionar aquele que tiver a maior magnitude.
- Detectar bordas nos 3 canais independentemente, considerar a união das 3 imagens.
  - Vejamos o efeito desta opção.



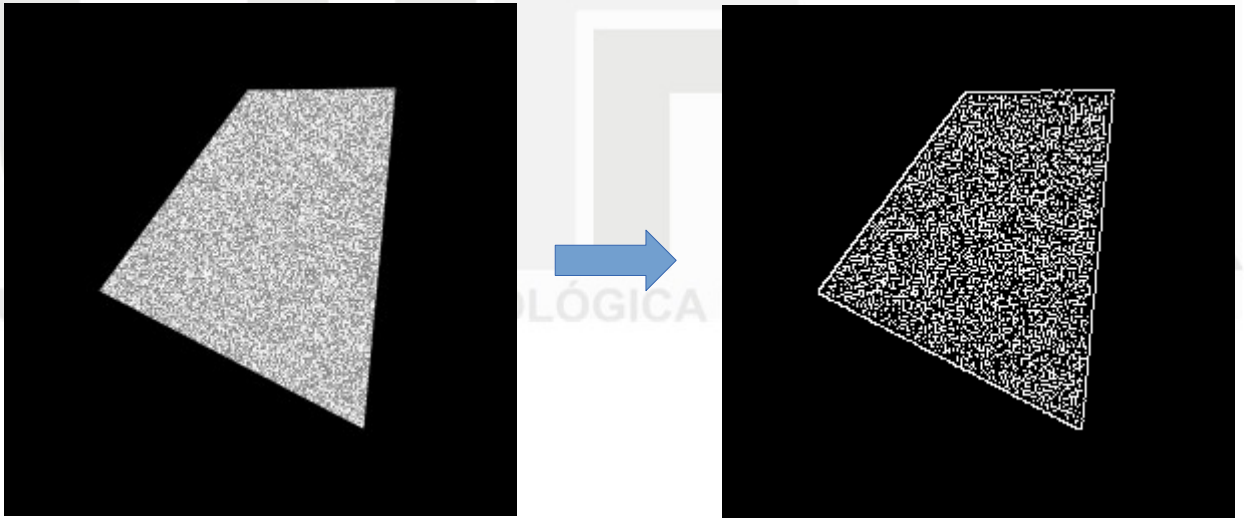
Bordas encontradas na imagem em escala de cinza.



Detectando nos 3 canais e unindo.

Redução de ruídos

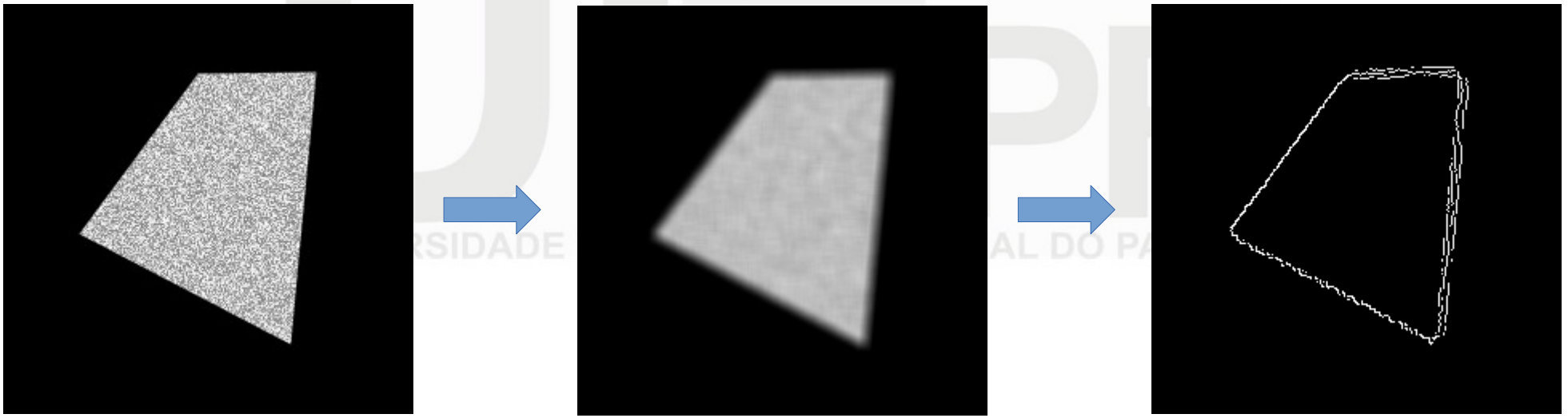
- Redução de ruídos é um passo importante quando extraímos bordas.
  - Queremos manter apenas as bordas ao redor de estruturas relevantes.



Primeiro, sem filtragem. Definitivamente, temos ruído demais...

Redução de ruídos

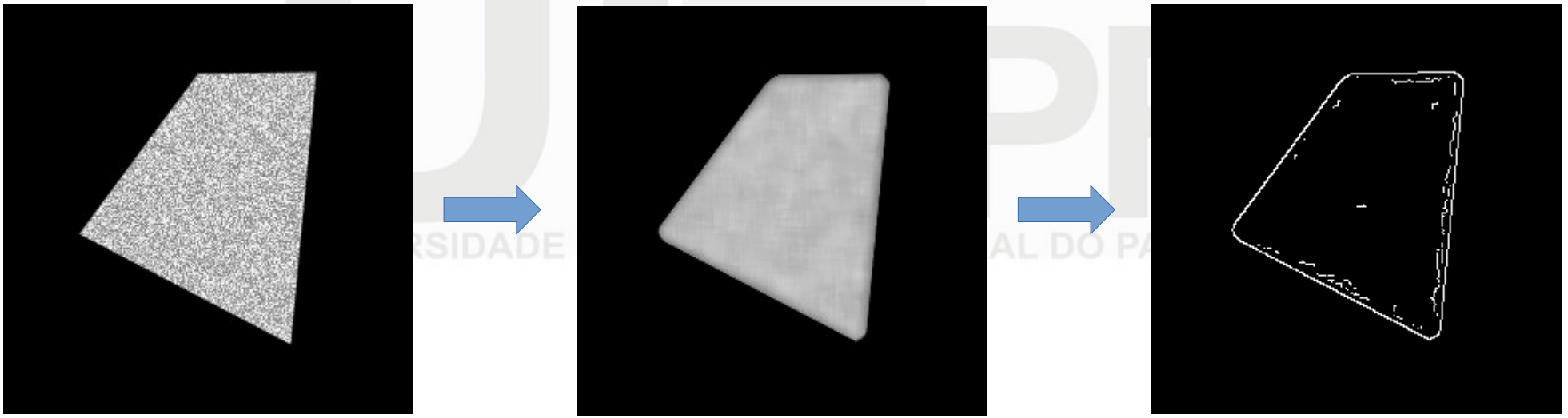
- Redução de ruídos é um passo importante quando extraímos bordas.
  - Queremos manter apenas as bordas ao redor de estruturas relevantes.



O filtro da média pode criar (duplicar) bordas. Ele também pode modificar os gradientes se as bordas não forem horizontais ou verticais.

Redução de ruídos

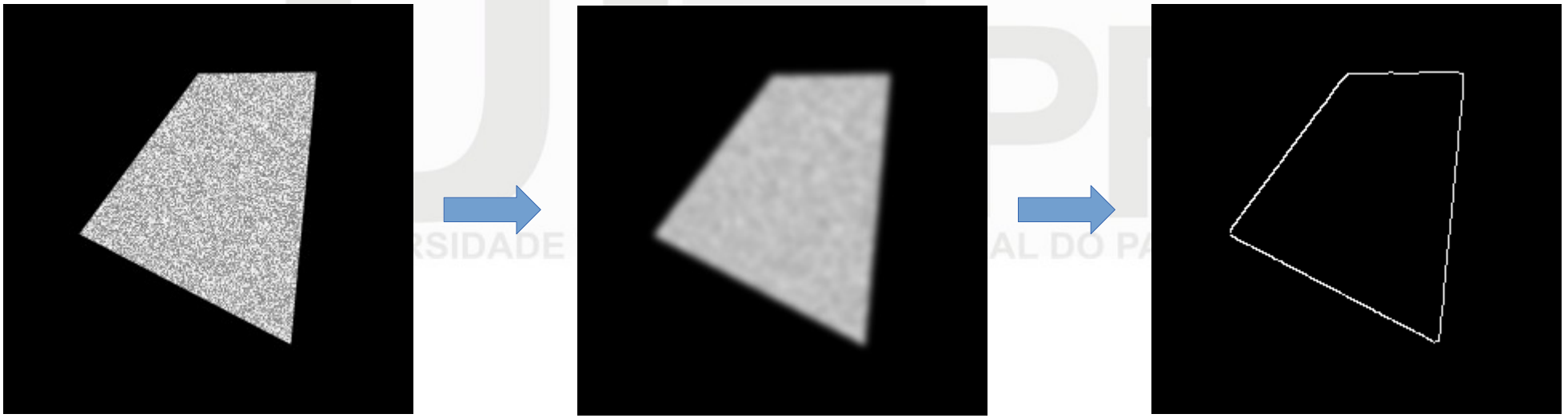
- Redução de ruídos é um passo importante quando extraímos bordas.
  - Queremos manter apenas as bordas ao redor de estruturas relevantes.



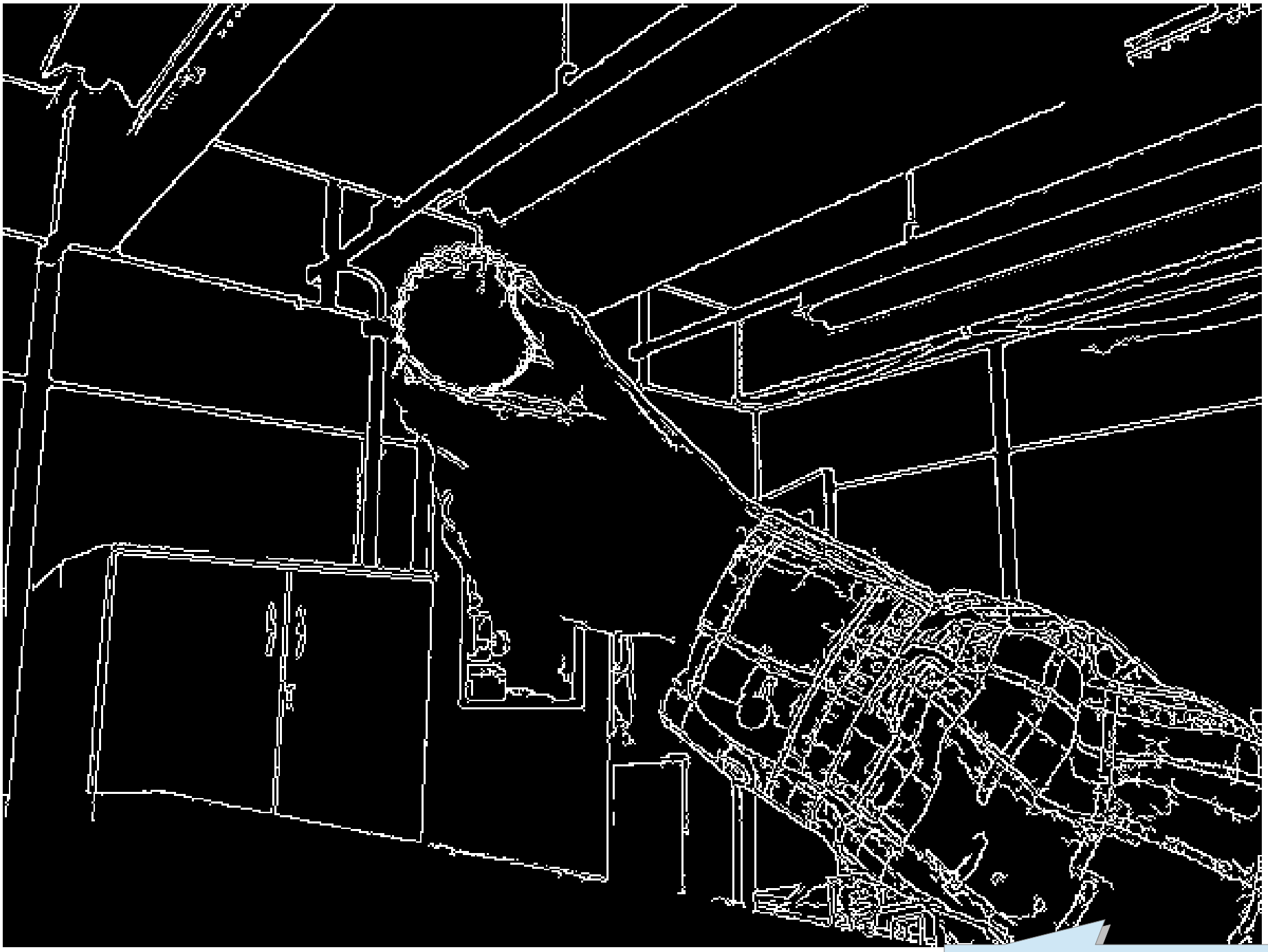
O filtro da mediana também não é muito eficaz.

Redução de ruídos

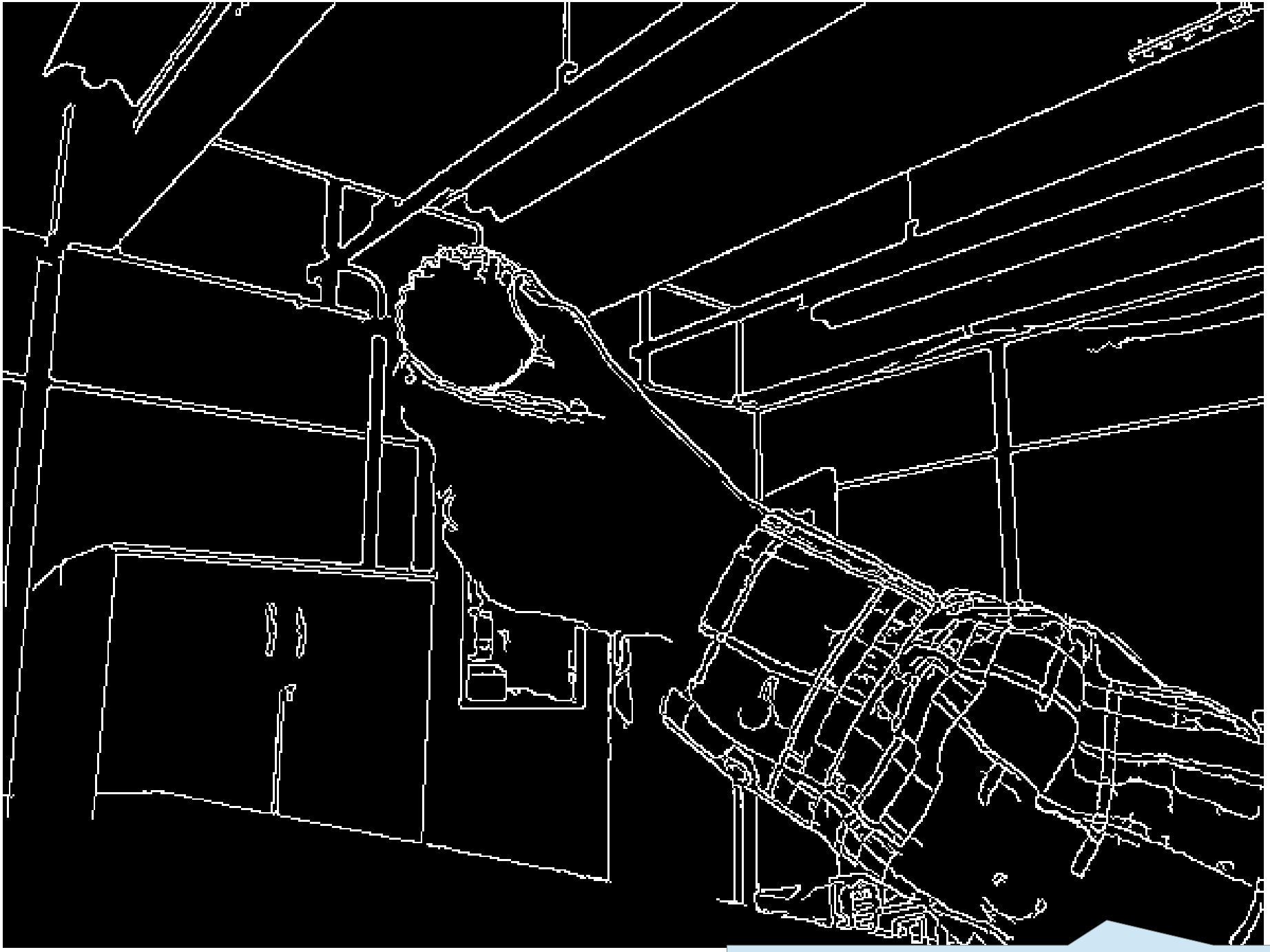
- Redução de ruídos é um passo importante quando extraímos bordas.
  - Queremos manter apenas as bordas ao redor de estruturas relevantes.



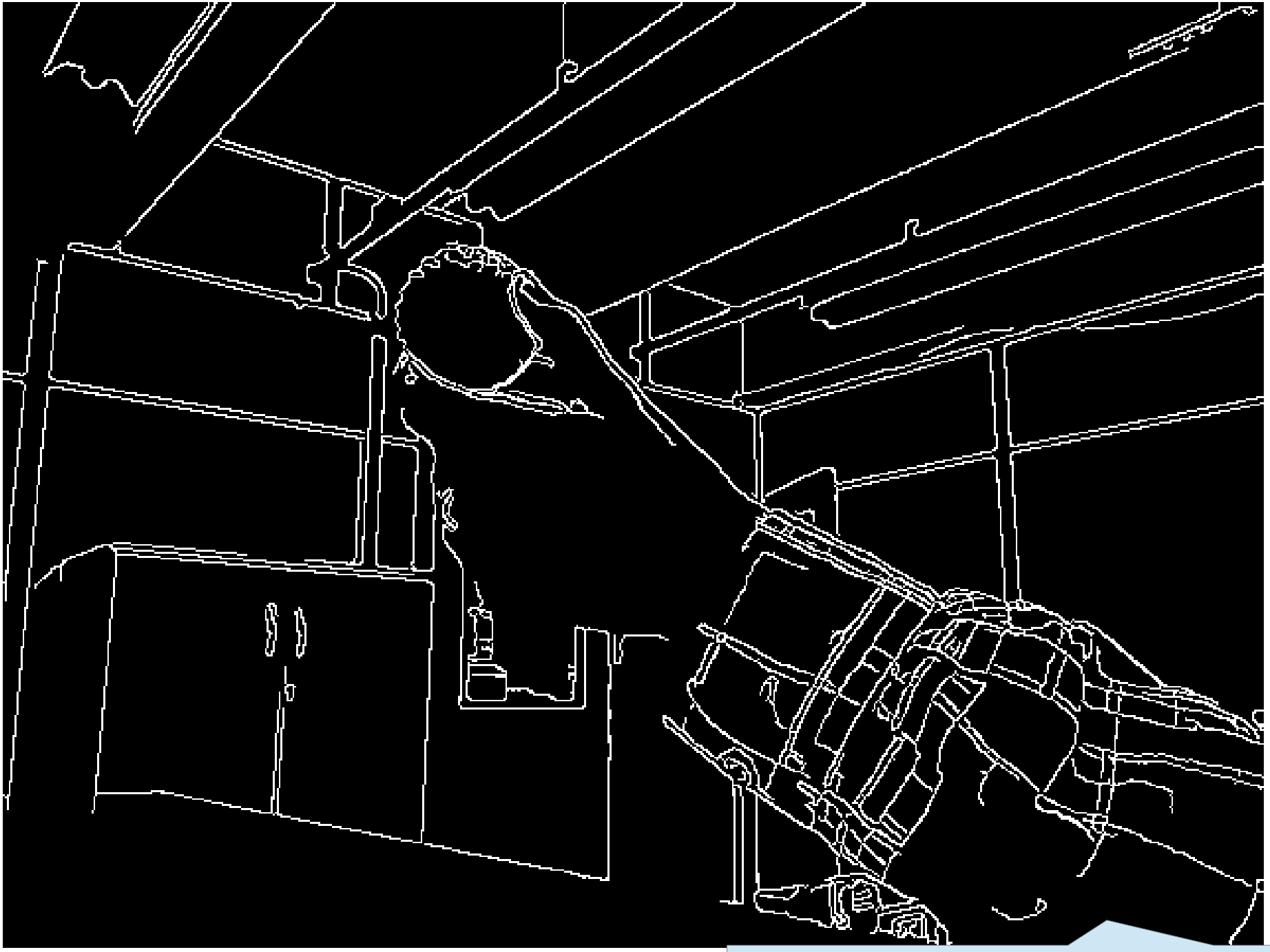
O filtro Gaussiano consegue reduzir o número de bordas sem criar novas estruturas.



Sem suavização.



Com suavização Gaussiana 3x3.



Com suavização Gaussiana 5x5.

Um pequeno truque...

- A seleção de limiares para o detector de Canny é muito dependente do contraste local entre regiões da imagem de dentrada.
  - = os parâmetros são dados em função da imagem de entrada.
- Podemos obter parâmetros mais ligados ao resultado desejado.
  - Ex: dar um intervalo de valores que indica uma proporção desejada de pixels de borda.
    - Fixamos uma proporção entre o limiar alto e o baixo (1/2, 1/3, etc.).
    - Podemos começar a inundação a partir dos gradientes mais fortes, ou repetir a parte da limiarização várias vezes, em um processo similar a uma busca binária.

Bordas: comentários finais

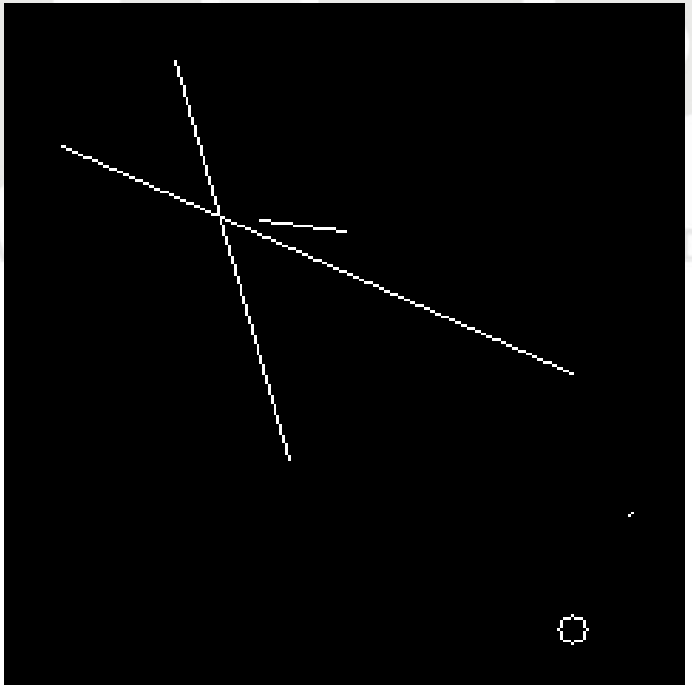
- Extração de bordas é um tópico clássico de PDI, e muitos trabalhos sobre o assunto já foram publicados.
- Apesar de existirem várias alternativas, não se conhece um algoritmo ideal para se extrair bordas em imagens arbitrárias.

Detecção de retas

- "Encontra bordas **alongadas e aproximadamente retas**."
  - Problema #1: detectar bordas.
  - Problema #2: detectar retas.

Para começar...

- Considere que uma reta é representada pela equação $y = ax+b$.
- Suponha que é dada uma imagem contendo bordas.
  - Como poderíamos localizar retas nesta imagem, de forma direta, usando um algoritmo de força bruta?



Localizando retas: força bruta

```
for (cada pixel de borda p1)
    for (cada pixel de borda p2 ≠ p1) {
        // Define a reta entre p1 e p2.
        a = (y2 - y1)/(x2 - x1);
        b = y1 – a*x1;

        // Conta quantos pixels de borda temos entre p1 e p2.
        cont = 0;
        for (cada valor xi entre x1 e x2) {
            yi = a*xi+b;
            if ((xi,yi) é um pixel de borda)
                cont++;
        }

        if (cont > limar) // Muitos pixels de borda sob esta reta!
            adiciona os valores de a e b a uma lista de retas.
    }
```

Localizando retas: força bruta

- A ideia “força bruta” tem vários problemas...
  - Alta complexidade computacional.
  - Pouca robustez a imprecisões – o contador para uma reta só é incrementado quando há um pixel exatamente sob a reta.
  - No final, podem existir muitas retas duplicadas.

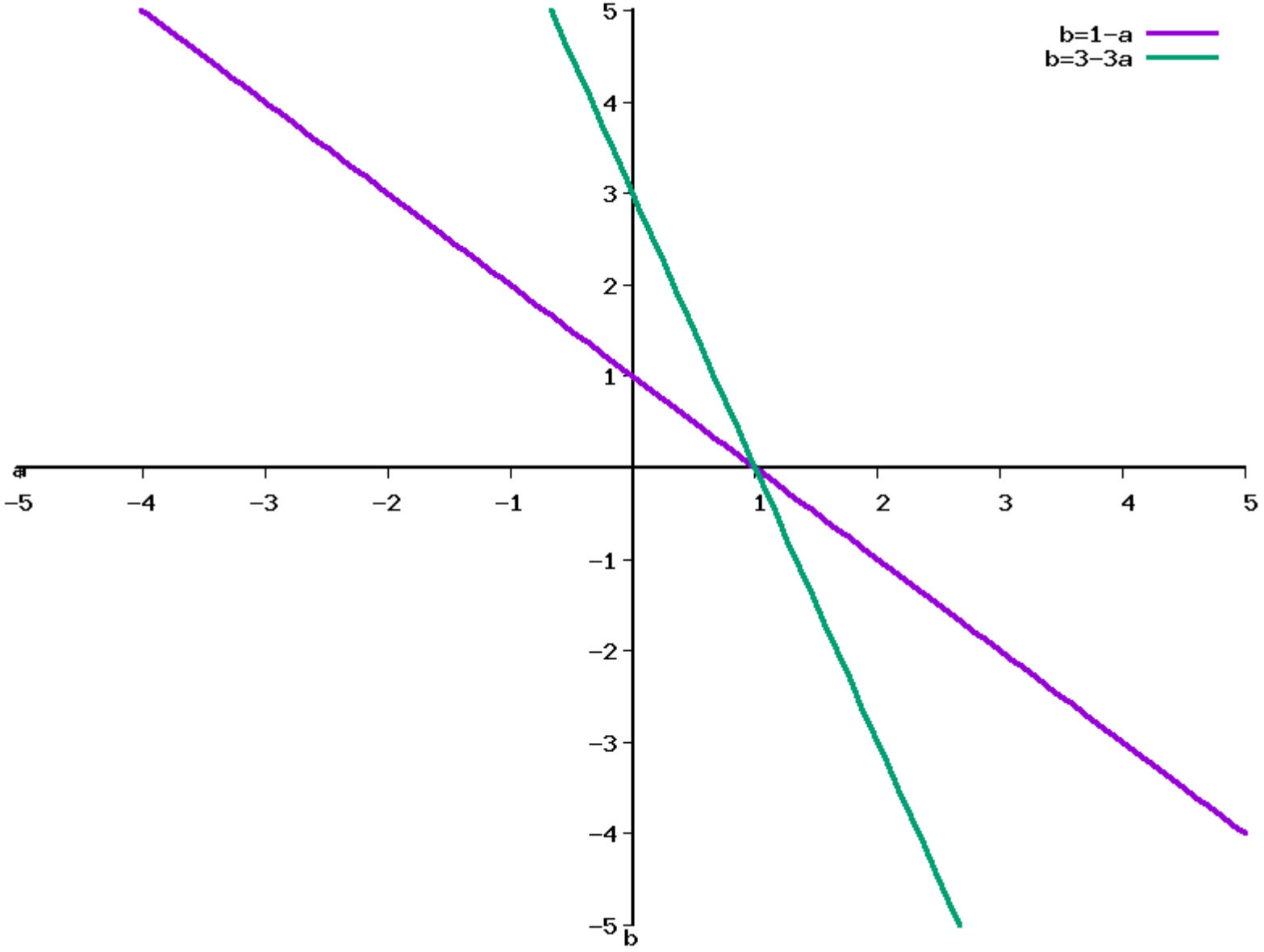
Transformadas

- Lembrando: uma imagem é uma função *f(x,y)* que associa um valor de intensidade ao pixel em uma posição *(x,y)*.
  - Como *x* e *y* são coordenadas espaciais, dizemos que uma imagem é representada no domínio espacial.
- Uma *transformada* é uma função para conversão de domínios.
  - = queremos representar a imagem em um domínio diferente do espacial.
- *Transformada de Hough*: converte a imagem para o domínio paramétrico de alguma forma geométrica.
  - Usada para localizar formas geométricas aproximadas em uma imagem de bordas.
    - "Aproximadas" = robustez a ruído e descontinuidades

Domínio paramétrico da reta

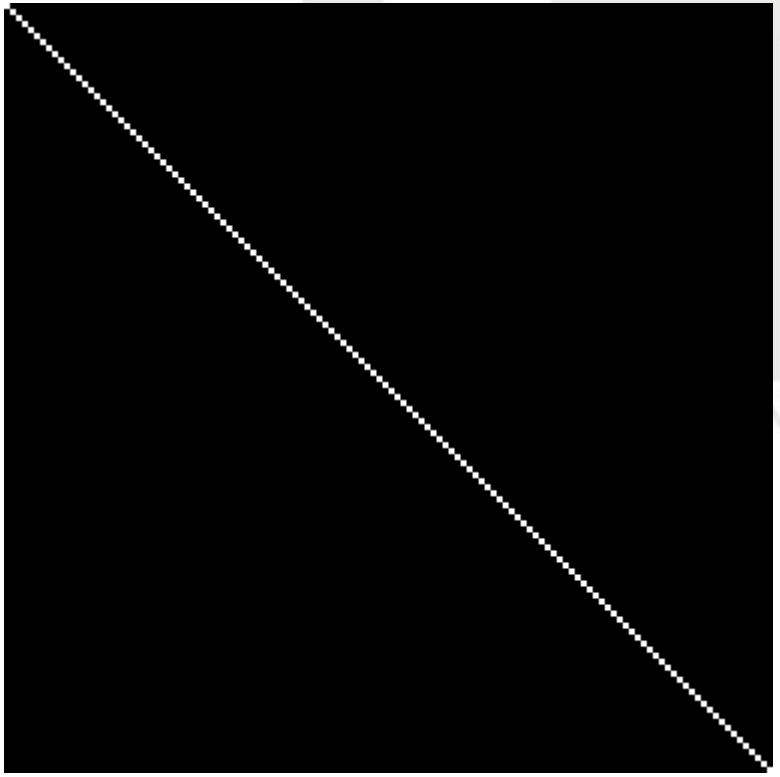
- Considere a equação da reta $y=ax+b$.
  - a e b são os *parâmetros* que definem uma reta.
  - Converter a imagem do domínio paramétrico da reta equivale a criar uma nova imagem, onde os eixos são a e b.
- Todas as retas que passam por um ponto podem ser expressas por $b=y-xa$.
  - Exemplo: (1,1) → $b=1-a$
  - Exemplo: (3,3) → $b=3-3a$
  - Como essas retas ficarão no domínio paramétrico (a,b)?

Domínio paramétrico da reta



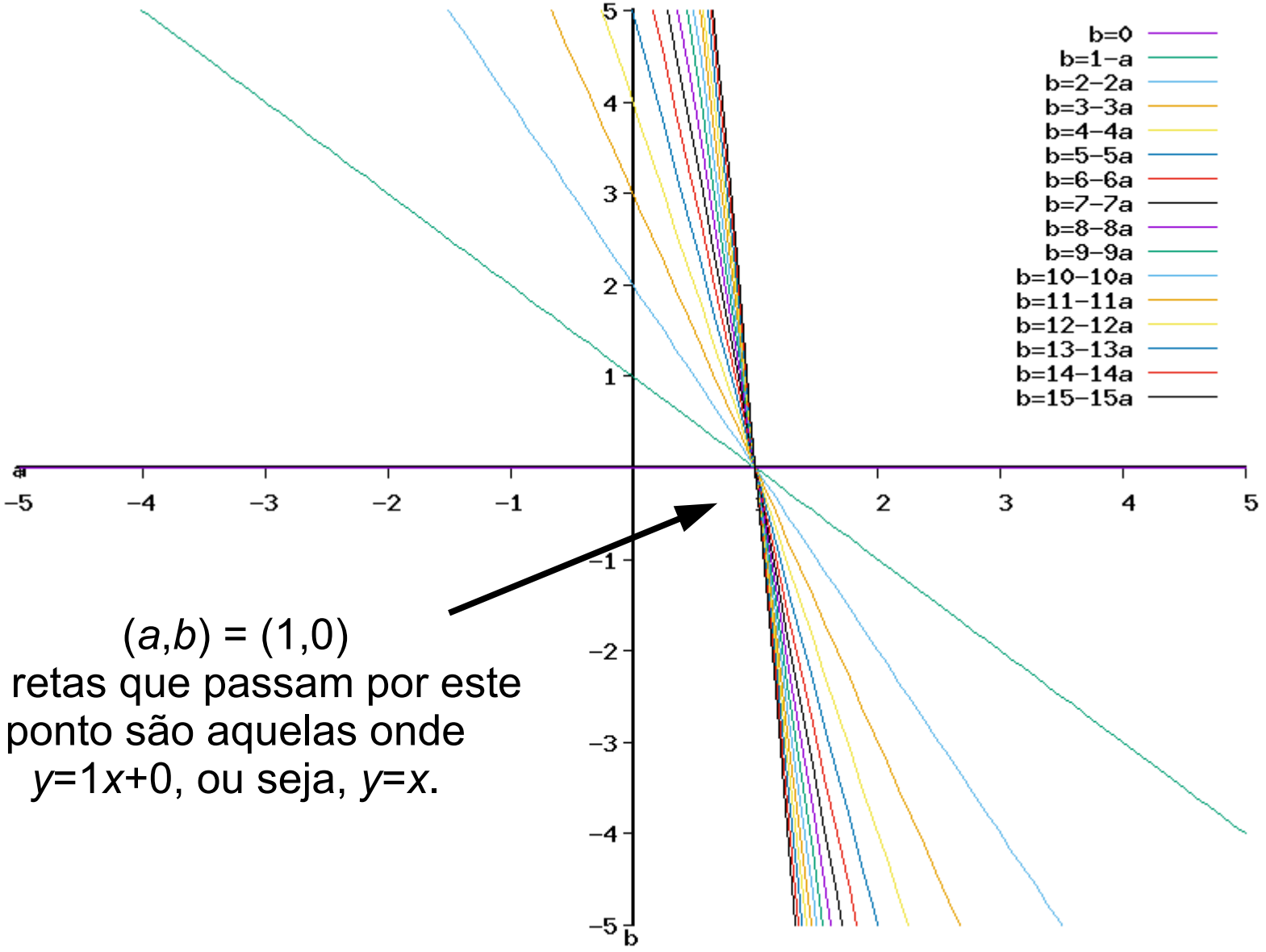
Domínio paramétrico da reta

•O que acontece se repetirmos o procedimento para todos os pixels setados na imagem abaixo?



(0,0)
(1,1)
(2,2)
(3,3)
…
(127,127)

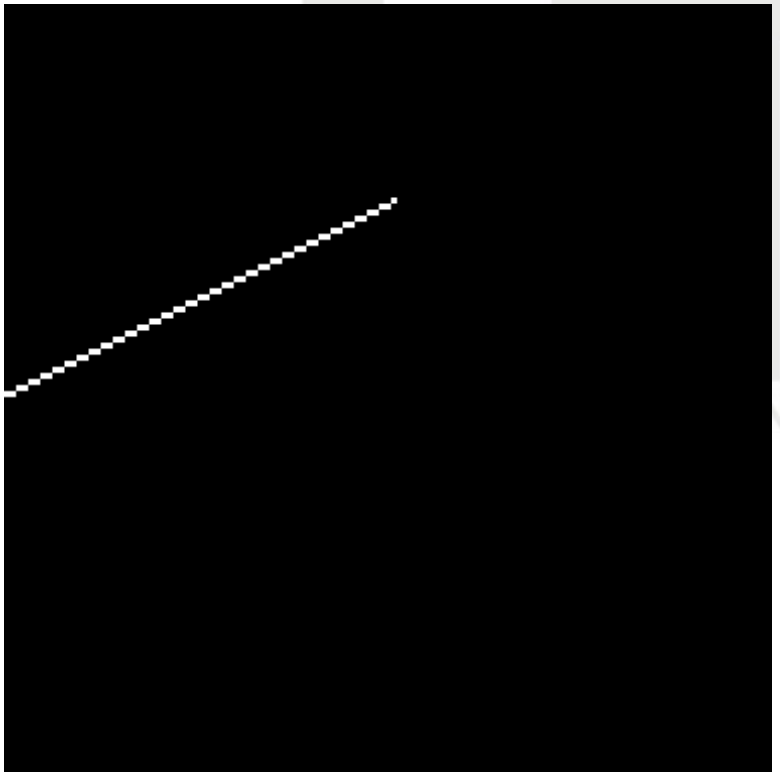
Domínio paramétrico da reta



(*a*,*b*) = (1,0)
As retas que passam por este ponto são aquelas onde *y*=1*x*+0, ou seja, *y*=*x*.

Mais um exemplo...

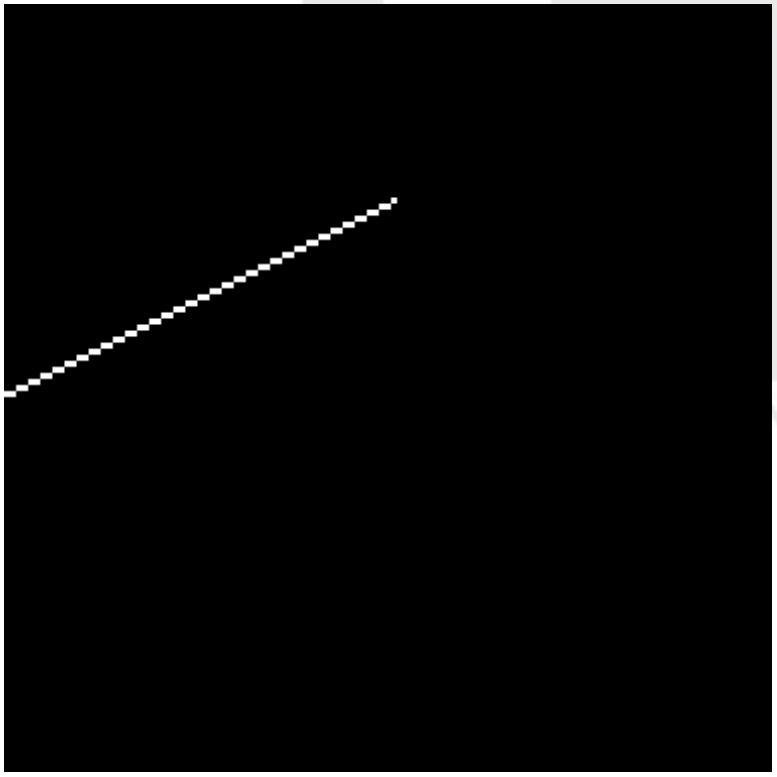
- Mais um exemplo...



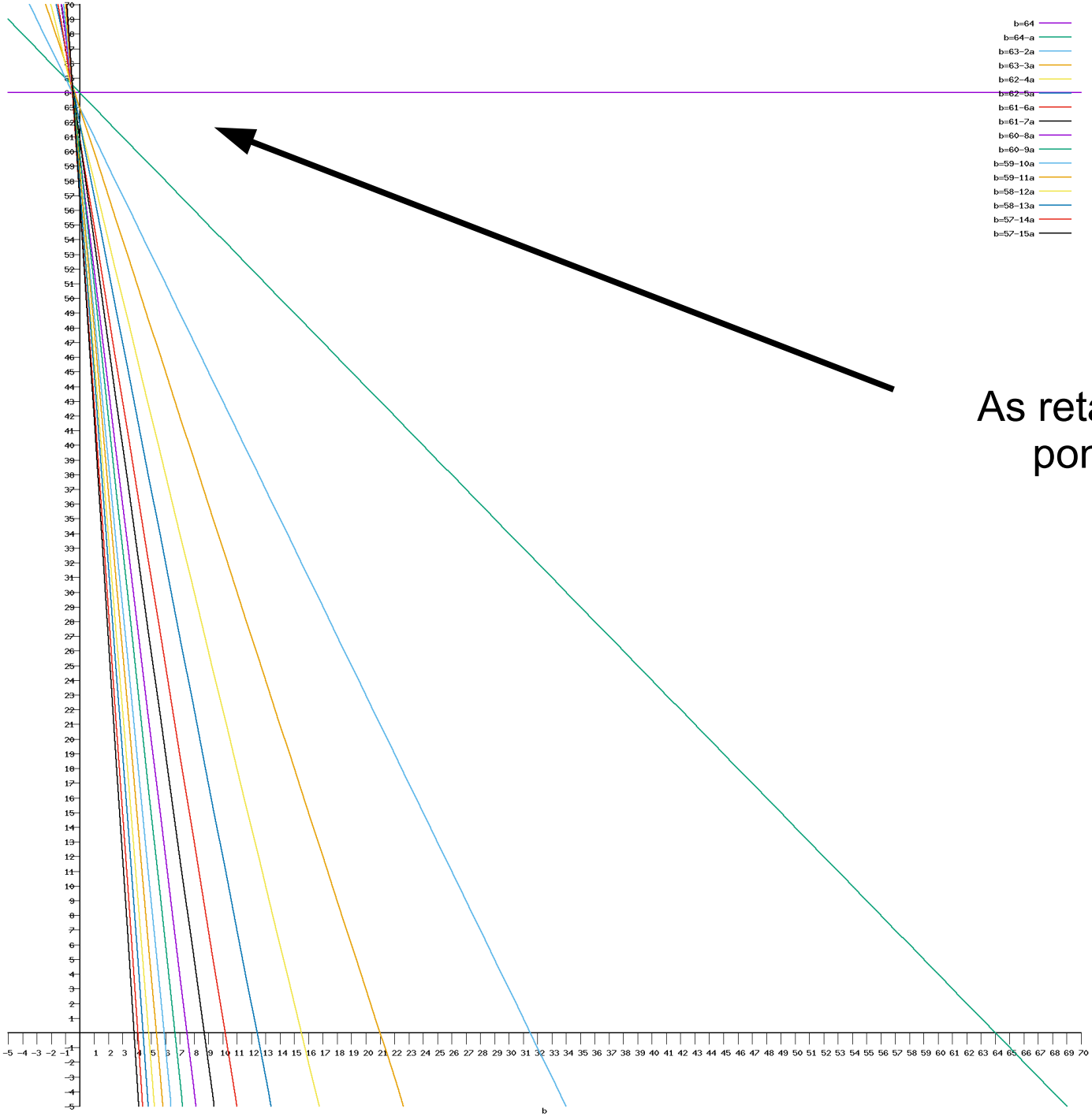
(0,64)
(1,64)
(2,63)
(3,63)
(4,62)
(5,62)
...
(64,32)

Mais um exemplo...

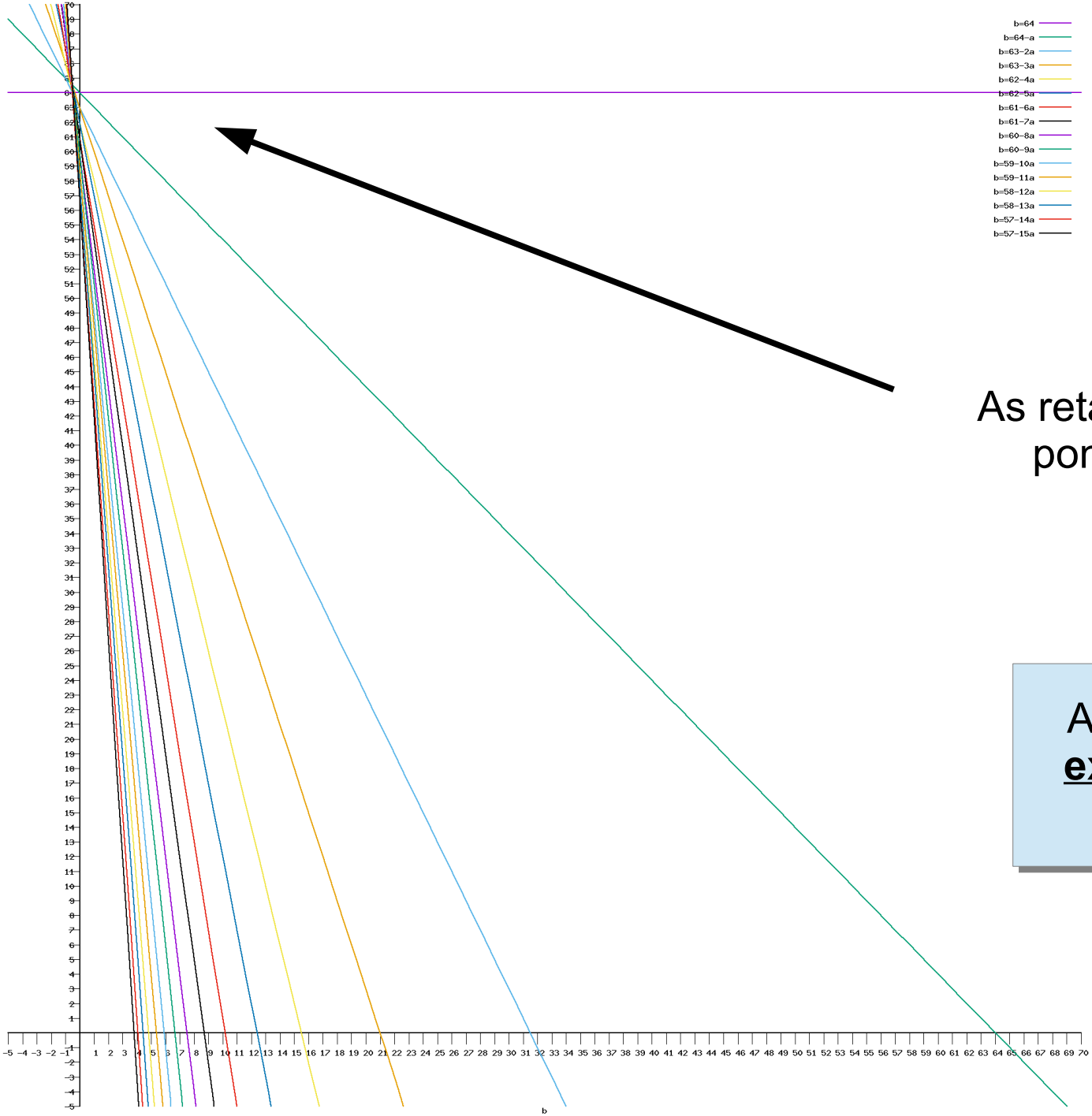
- Mais um exemplo...



$(0,64) \rightarrow b = 64$
$(1,64) \rightarrow b = 64 - a$
$(2,63) \rightarrow b = 63 - 2a$
$(3,63) \rightarrow b = 63 - 3a$
$(4,62) \rightarrow b = 62 - 4a$
$(5,62) \rightarrow b = 62 - 5a$
...
$(64,32) \rightarrow b = 32 - 64a$

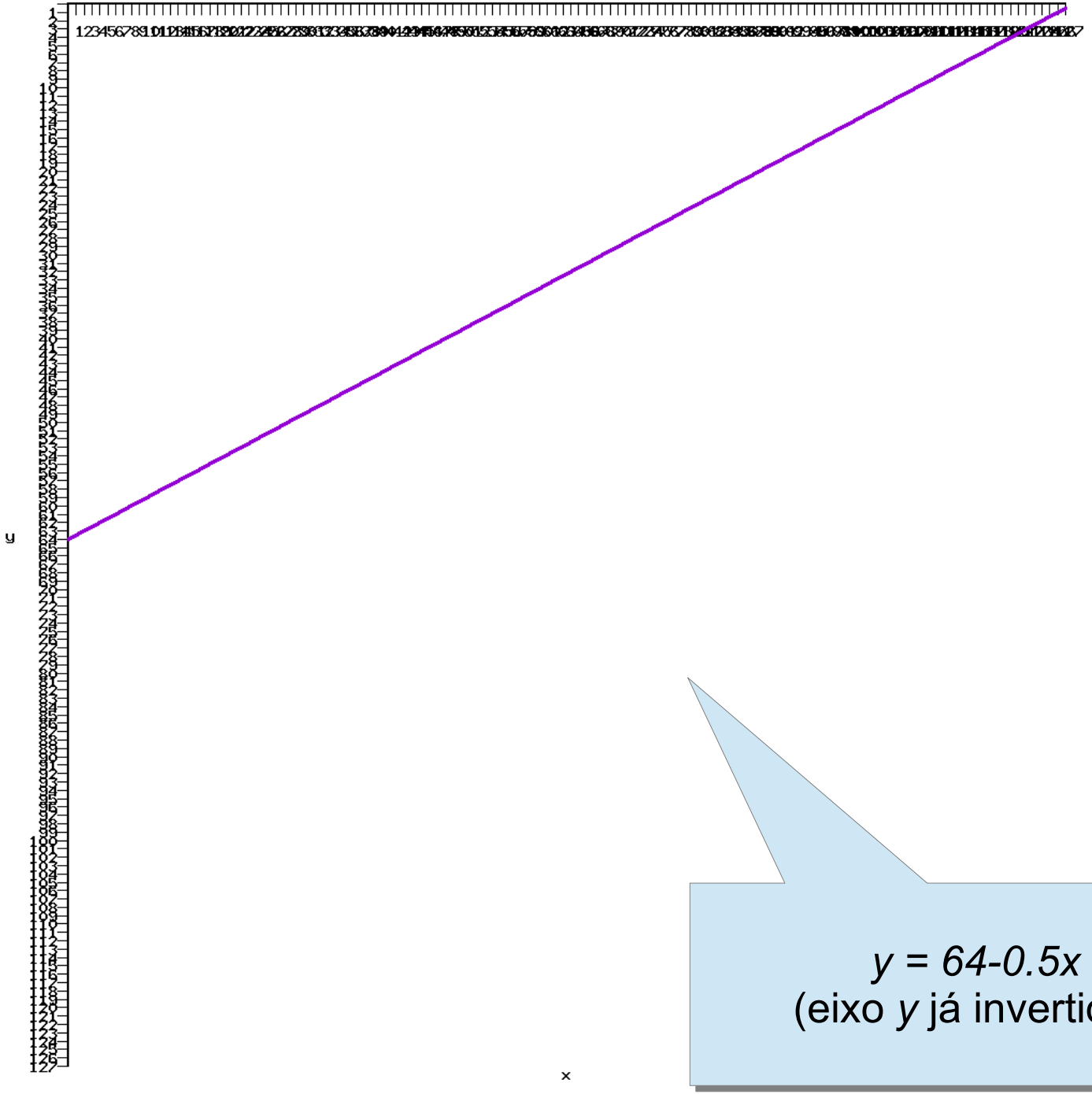


(a,b) = (0.5,64)
As retas que passam por este ponto são aquelas onde y=64-0.5x.



$(a,b) = (0.5,64)$
As retas que passam por este ponto são aquelas onde y=64-0.5x.

As retas NÃO se cruzam **exatamente** em (0.5,64).
POR QUÊ?



$y = 64-0.5x$
(eixo y já invertido).

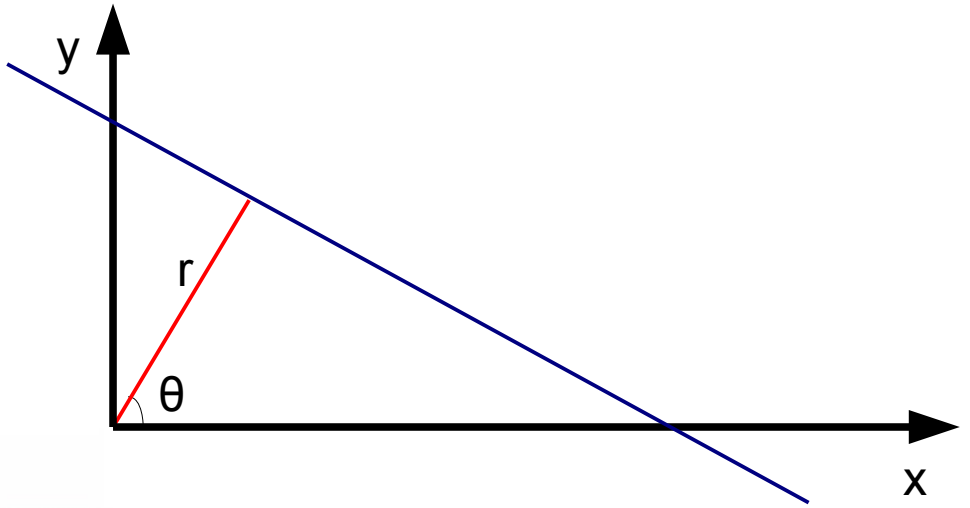
Transformada de Hough: ideia geral

- Para cada pixel de borda (x,y):
  - Gera os valores de a e b de todas as retas que passam por (x,y).
- Procura por cruzamentos no domínio paramétrico.
  - Várias retas no domínio paramétrico se cruzam em um ponto (a,b) → a reta definida por $y=ax+b$ passa por vários pixels de borda.
  - Então, pontos do domínio paramétrico onde várias retas se cruzam descrevem prováveis retas no domínio espacial!

Equação polar da reta

- O domínio (a,b) é problemático...
  - A equação $y=ax+b$ não permite representar retas verticais.
  - Os valores de b podem variar muito – são potencialmente infinitos – o que impede a geração de todas as retas que passam por um ponto.
- Na prática, usa-se a equação polar da reta: $r = x.cos\theta + y.sen\theta$
  - A reta é descrita por um ângulo θ e uma distância r.
  - Exemplo: (0,10) → $r=10 \cdot sen\theta$.
  - Exemplo: (10,0) → $r=10 \cdot cos\theta$.

Senoides!!!



Quantização

- Para reduzir a carga computacional, e para que o algoritmo seja robusto a ruído e imprecisões, o domínio paramétrico é quantizado.
  - Os valores possíveis para r e θ são discretizados – r é dado em um número de pixels e θ em radianos.
    - Aqui, temos duas opções:
      - Valor de θ entre 0 e 2π, descartando r negativo.
      - Valor de θ entre 0 e π, com r podendo ser negativo.
      - (Por quê?)
  - Montamos um histograma 2D, com cada compartimento correspondendo a uma combinação (r,θ).
    - A largura dos compartimentos diz o quanto os pixels sob uma reta precisam estar alinhados.

Transformada de Hough: algoritmo

```
cria um histograma 2D [θ][r]

for (cada pixel de borda (x,y))
{
    for (θ = 0; θ < π; θ += passo)
    {
        r = quantiza (x*cos(θ) + y*sen(θ));
        histograma [theta][r]++;
    }
}

for (cada combinação possível de [θ] e [r])
    if (histograma [θ][r] > limiar)
        adiciona os valores de r e θ a uma lista de retas;
```

Vejamos alguns exemplos...

•Vejamos alguns exemplos de retas detectadas pela transformada de Hough.

- Resultados obtidos por uma implementação simples feita pelo professor.

Transformada de Hough: variações

- A transformada de Hough tem muitas variações!
  - Probabilísticas.
  - Para segmentos de retas.
  - Multi-escala.
  - Usando a magnitude e direção dos gradientes.
  - Com suavização e interpolação do histograma.
  - Com máximos locais.

- Podemos generalizar a abordagem para qualquer forma que possa ser descrita por uma equação.
  - Formas comuns: círculos, elipses e parábolas.
  - Fatores que afetam a complexidade:
    - Número de parâmetros.
      - Normalmente, usam-se no máximo 3, talvez 4.
    - Número de compartimentos no domínio paramétrico.

Transformada de Hough: notas finais

- A transformada de Hough foi criada em 1959, originalmente para a análise de dados de experimentos de física!
- A HT é uma versão discreta / quantizada da transformada de Radon, que é usada na criação de imagens de tomografias.
- A HT é na prática similar a um algoritmo para regressão linear.
- Existem vários outros algoritmos que buscam ajustar entidades geométricas (curvas, parábolas, etc.) a bordas extraídas da imagem.

Retornando ao problema original

- Sabemos como encontrar retas na imagem. Nosso desafio agora é:
  - Selecionar 4 retas, cada uma passando sobre um dos lados da página.
  - Encontrar os pontos de cruzamento das linhas verticais e horizontais.

- Pressupostos:
  - A maior parte da página está contida na imagem.
  - Procuramos manter uma proporção pequena de pixels de bordas.
    - Nos exemplos mostrados, consideramos não mais que 0.5%.
  - As bordas detectadas incluem os limites da página.
  - A página está aproximadamente "de pé" (|rotação| < 45^{o}).
  - A página (quase) não tem ondulações.

- Neste caso, devemos ter a maior parte das retas exatamente sobre os lados da página...
  - Mas talvez outras retas tenham sido detectadas.
  - Talvez existam múltiplas retas sobre um mesmo lado da página.

Um algoritmo simples...

- Separamos as retas detectadas em aproximadamente horizontais e aproximadamente verticais.
- Encontramos para as retas aproximadamente horizontais o ponto médio de cruzamento com o eixo *y*.
- Separamos as retas entre aquelas que estão acima e abaixo do ponto médio computado.
  - O lado superior da página deve ser uma das retas acima, o lado inferior deve ser uma das retas abaixo.
  - Para cada grupo selecionamos a mediana do ponto de cruzamento com o eixo *y*.
- Repetimos o processo para as retas aproximadamente verticais.

Finalizando

- Vejamos alguns exemplos...
- A solução que apresentamos para encontrar os lados e cantos da página é bem simples.
  - Além dos pressupostos mencionados anteriormente, é preciso que a maior parte das linhas tenha sido detectada sobre os lados da página.
  - Solução mais sofisticada: separação em 2 classes, com maximização da variância entre classes (Otsu).