Edge detection using wavelets

Computation of gradients

In[18]:=

```
dir2vec[a_]:=Module[{aa},
(*
Computing the octant in which vector a={ax,ay} lies
*)
If[a=={0,0},Return[{1,0}]];
aa=Mod[N[ArcTan[a[[1]],a[[2]]]+Pi/2],Pi,-Pi/2];
Which[
Abs[aa]<Pi/8,{1,0},
Pi/8<aa≤3 Pi/8,{1,1},
-3Pi/8<aa≤- Pi/8,{1,-1},
True,{0,1}]
]
```

In[2]:=

```
grad[A_,hfilter_,vfilter_]:=
(*
Computing the arrays for length (val) and direction (dir)
of gradient vectors by filtering array A with filter hfilter and vfilter
*)
Module[{AA,val,dir,m,n,Ah,Av,h,v},
{m,n}=Dimensions[A];
AA=ArrayPad[A,{1,1}];
Ah=ListConvolve[hfilter,AA];
Av=ListConvolve[vfilter,AA];
val=Table[0,{m},{n}];
dir=Table[0,{m},{n}];
Do[
{h,v}={Ah[[x+1,y+1]],Av[[x+1,y+1]]};
val[[x,y]]=N[Sqrt[h^2+v^2]];
dir[[x,y]]=dir2vec[{h,v}],
{y,1,n},{x,1,m}];
{val,dir}
]
```

Determining edge vertices using gradient analysis

In[3]:=

```
edges1[A_,hfilter_,vfilter_,level_]:=
(*
Detecting egde vertices in array A using gradient analysis.
 The output cand is a 0-1-array of the same dimension as A
 which shows all candidates.
 Shown are all vertices which satisfy the edge vertex
 criterion and where the local gradient maximum i
s ≥ "level"*average gradient
*)
Module[{m,n,val,dir,mean,VVal,Cand,
 x,x1,x2,y,y1,y2,val0,val1,val2},
 {m,n}=Dimensions[A];
 {val,dir}=grad[A,hfilter,vfilter];
 mean=Mean[Flatten[val]];
 VVal=ArrayPad[val,{1,1}];
 Cand=Table[0,{m},{n}];
 Do[
 {x1,y1}={x+1,y+1}+dir[[x,y]] ;
 {x2,y2}={x+1,y+1}-dir[[x,y]];
 val0=VVal[[x+1,y+1]];
 val1=VVal[[x1,y1]];
 val2=VVal[[x2,y2]];
 If[
 val0≥Max[val1,val2]&&val0≥level mean,
 Cand[[x,y]]=1],
 {x,1,m},{y,1,n}];
 Cand
 ]
```

In[4]:=
```
edges2[A_,B_,level_]:=
(*
Detecting egde vertices in array A using gradient analysis.
 Input is given as two x- and y-filtered array A and B.
 The output cand is a 0-1-array of the same dimension as A
 which shows all candidates.
 Shown are all vertices which satisfy the edge vertex criterion
and where the local gradient maximum is ≥ "level"*average gradient
*)
Module[{m,n,val,dir,mean,VVal,Cand,h,v,
x,x1,x2,y,y1,y2,val0,val1,val2},
{m,n}=Dimensions[A];
If[{m,n}≠Dimensions[B],
Throw["Dimensions don't match"]
];
val=Table[0,{m},{n}];
dir=Table[0,{m},{n}];
Do[
{h,v}={A[[x,y]],B[[x,y]]};
val[[x,y]]=N[Sqrt[h^2+v^2]];
dir[[x,y]]=dir2vec[{h,v}],
{x,1,m},{y,1,n}
];
mean=Mean[Flatten[val]];
VVal=ArrayPad[val,{1,1}];
Cand=Table[0,{m},{n}];
Do[
{x1,y1}={x+1,y+1}+dir[[x,y]] ;
{x2,y2}={x+1,y+1}-dir[[x,y]];
val0=VVal[[x+1,y+1]];
val1=VVal[[x1,y1]];
val2=VVal[[x2,y2]];
If[
val0≥Max[val1,val2]&&val0≥level mean,
Cand[[x,y]]=1],
{x,1,m},{y,1,n}
];
Cand
]
```

Test image

In[19]:=
```
img = Import["~/Lehre/Wavelets-All/WTBV-10/CWT/zebras.jpg"]
```

Out[19]=

Grayscale version of the test image

In[20]:=
```
img = ColorConvert[img, "Grayscale"]
```

Out[20]=

Smoothed image

In[21]:=
```
img = GaussianFilter[img, 2]
```

Out[21]=

In[22]:=
```
imdim = ImageDimensions[img]
```

Out[22]=
```
{500, 357}
```

Naive gradient method

Horiziontal and vertical Haar filter (high-pass)

In[23]:=
```
h = {{1, -1}}; h // MatrixForm
v = {{1}, {-1}}; v // MatrixForm
```

Out[23]//MatrixForm=

$$\begin{pmatrix} 1 & -1 \end{pmatrix}$$

Out[24]//MatrixForm=

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Filtering the image with the Haar filter

In[25]:=
```
imgh = ImageConvolve[img, h];
```

In[26]:=
```
imgv = ImageConvolve[img, v];
```

In[30]:=
```
IA = ImageAdjust; isl = ImageSize → Large;
```

Displaying the filtered image

In[31]:=
```
GraphicsGrid[{{imgh, imgv}, {IA[imgh], IA[imgv]}}, isl]
```

Out[31]=

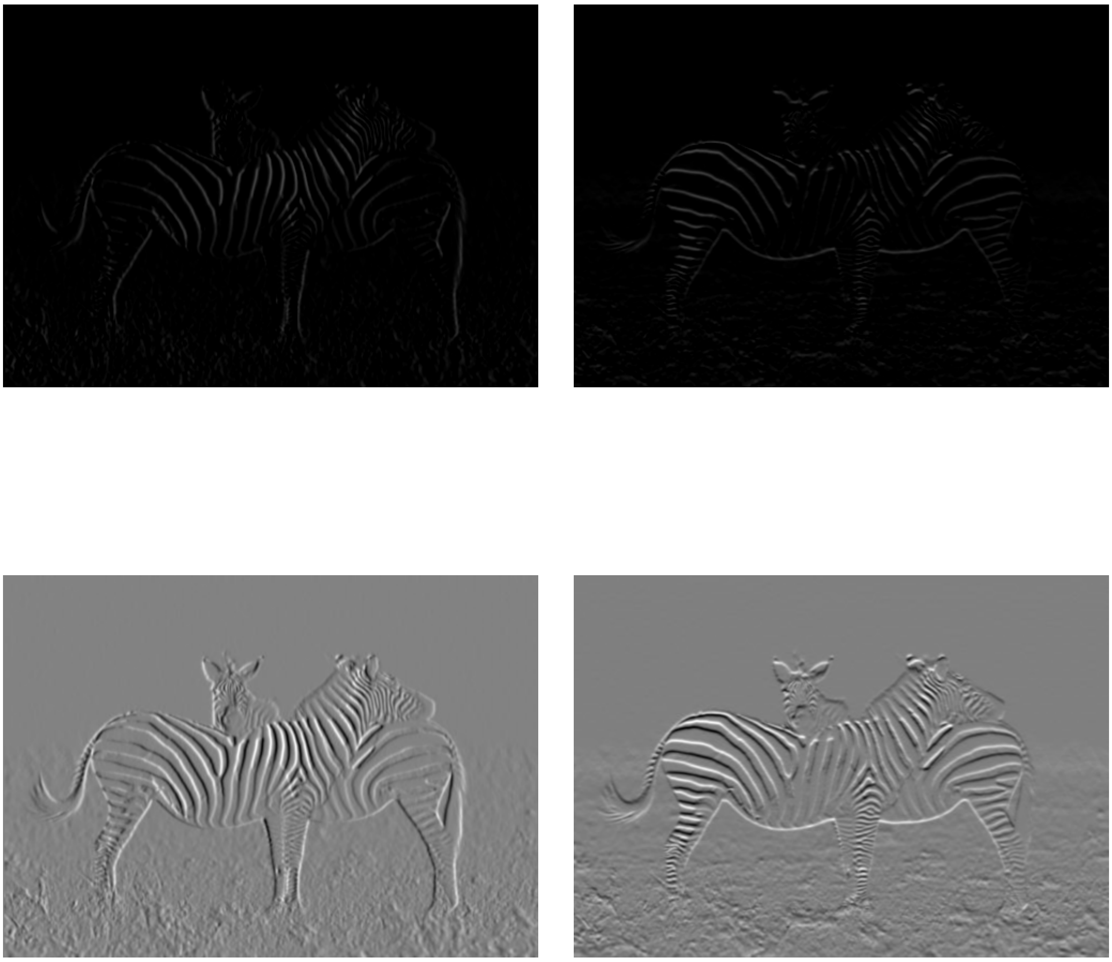


Image data as an array

In[32]:=
```
imgdata = ImageData[img];
```

Edge detection based on the filtered image data

In[33]:=
```
ImageDimensions[img]
```

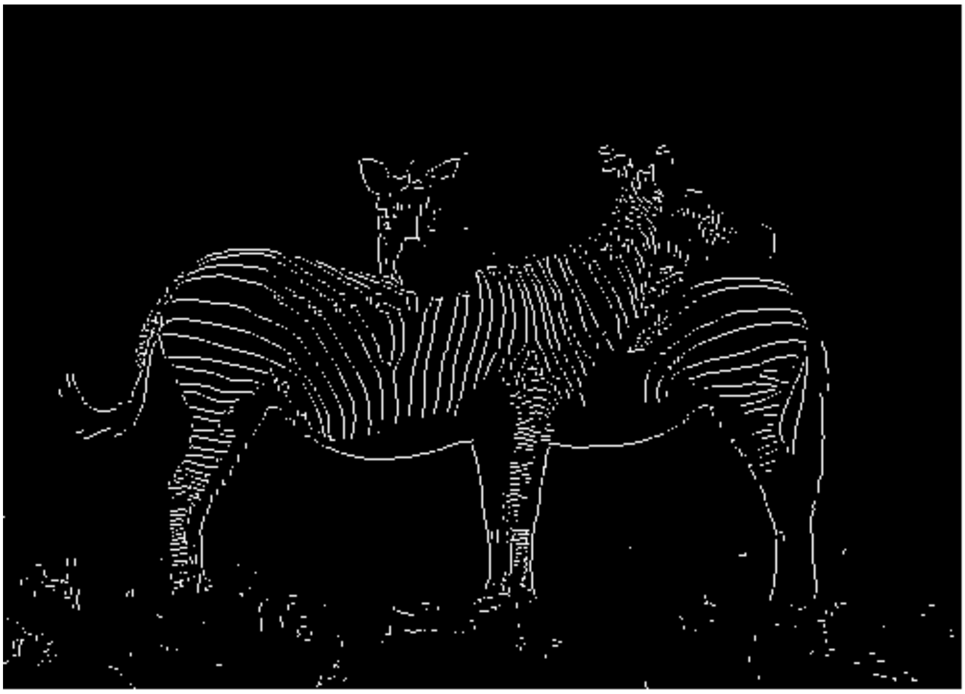
Out[33]= {500, 357}

In[34]:=
```
Dimensions[imgdata]
```

Out[34]= {357, 500}

In[35]:=
```
Image[edges1[imgdata, h, v, 3.0]]
```

Out[35]=



Gradients and the à-trous algorithm

Spreading (upsampling) a finite filter

In[36]:=
```
spread[fil_]:=Most[Flatten[Map[{#,0}&,fil]]]
```

Iterated spreading of a finite filter

In[37]:=
```
spread[fil_,n_]:=Nest[spread[#]&,fil,n]
```

Example: double spreading of a filter of length 3

In[38]:=
```
spread[{a, b, c}, 2]
```

Out[38]=
```
{a, 0, 0, 0, b, 0, 0, 0, c}
```

One-level WT for images on approximation data

```
WTstep[AX_,AY_,H_,V_,h_,v_]:=
(*
One-level wavelet transform based on x-and y-filtered arrays AX and AY
using low-pass filters H and V to obtain the approximation data Ax and Ay
and using high-pass filters h and v to obtain the detail data Wx and Wy.
Filters are 2-dimensional!
*)
Module[
{Ax,Ay,Wx,Wy},
If[Dimensions[AX]≠Dimensions[AY],Throw["dimensions don't match"]];
Ax=ListConvolve[H,AX,{1,1},0];
Ay=ListConvolve[V,AY,{1,1},0];
Wx=ListConvolve[h,AX,{1,1},0];
Wy=ListConvolve[v,AY,{1,1},0];
{Ax,Ay,Wx,Wy}
]
```

Bspline filters of length 2 and 3

In[39]:=
```
spline1 = {1, 1}/2
spline2 = {1, 2, 1}/4
```

Out[39]= $\left\{\frac{1}{2}, \frac{1}{2}\right\}$

Out[40]= $\left\{\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right\}$

2D low-pass filters constructed from the spline filters

In[8]:=
```
H[k_]:=KroneckerProduct[spread[spline2,k],spread[spline1,k]];
```

In[9]:=
```
V[k_]:=KroneckerProduct[spread[spline1,k],spread[spline2,k]];
```

```
H[0] // MatrixForm
```

$$\begin{pmatrix} \frac{1}{8} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{4} \\ \frac{1}{8} & \frac{1}{8} \end{pmatrix}$$

```
V[2] // MatrixForm
```

$$\begin{pmatrix} \frac{1}{8} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{8} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{8} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{8} \end{pmatrix}$$

In[45]:=
```
Clear[h, v]
```

Spreading the Haar high-pass filter

In[46]:=
```
h[k_]:={spread[{1,-1},k]};
v[k_]:=Transpose[h[k]];
```

In[48]:=
```
h[2] // MatrixForm
```

Out[48]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 \end{pmatrix}$$

In[49]:=
```
v[2] // MatrixForm
```

Out[49]//MatrixForm=

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

Initialize the *AX* and *AY* arrays

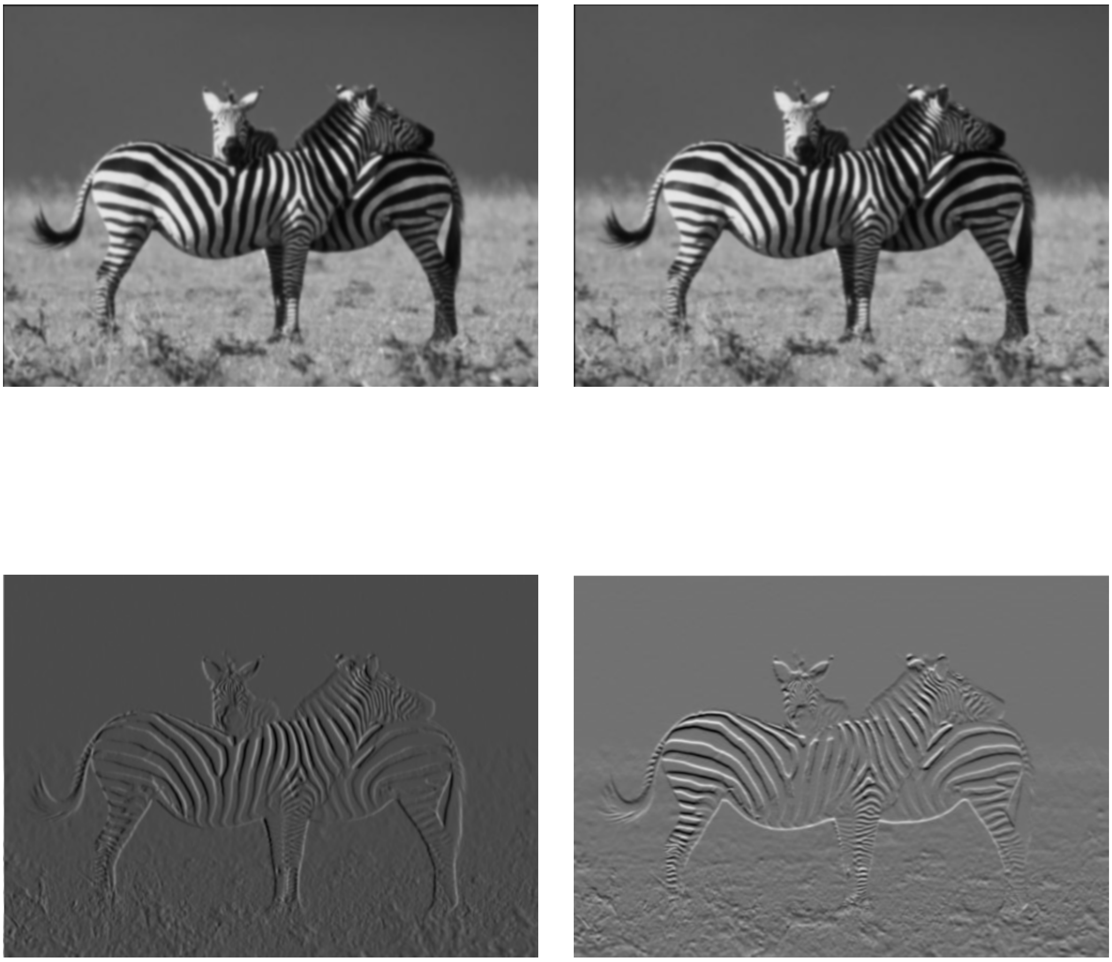
In[50]:=
```
AX = ImageData[img]; AY = ImageData[img];
```

First level of the 2D wavelet transform (using unspreaded filters)

In[51]:=
```
{A1x, A1y, W1x, W1y} = WTstep[AX, AY, H[0], V[0], h[0], v[0]];
```

In[69]:=
```
GraphicsGrid[{{Image[A1x], Image[A1y]}, {IA[Image[W1x]], IA[Image[W1y]]}}, isl]
```

Out[69]=

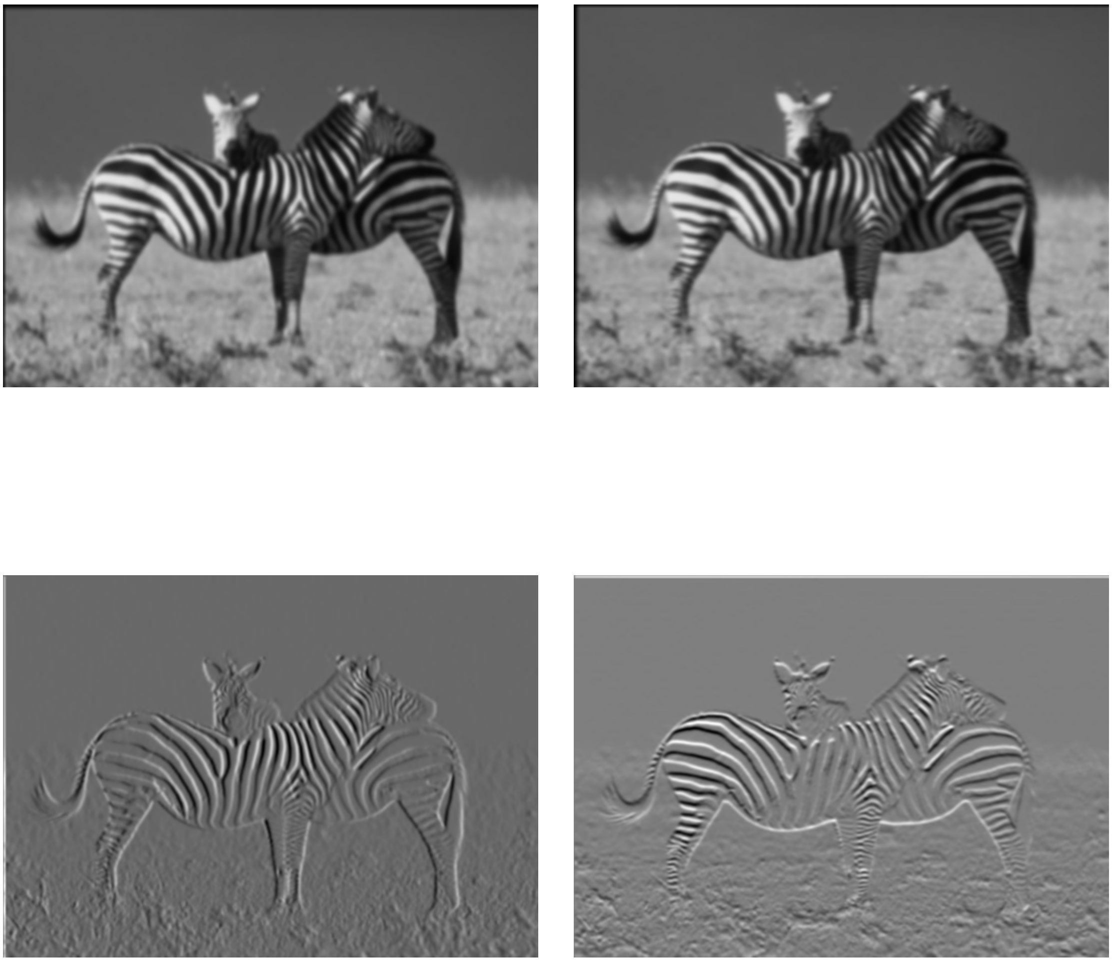


Second level of the 2-dim wavelet transform (using filters spreaded once)

In[53]:=
```
{A2x, A2y, W2x, W2y} = WTstep[A1x, A1y, H[1], V[1], h[1], v[1]];
```

In[68]:=
```
GraphicsGrid[{{Image[A2x], Image[A2y]}, {IA[Image[W2x]], IA[Image[W2y]]}}, isl]
```

Out[68]=

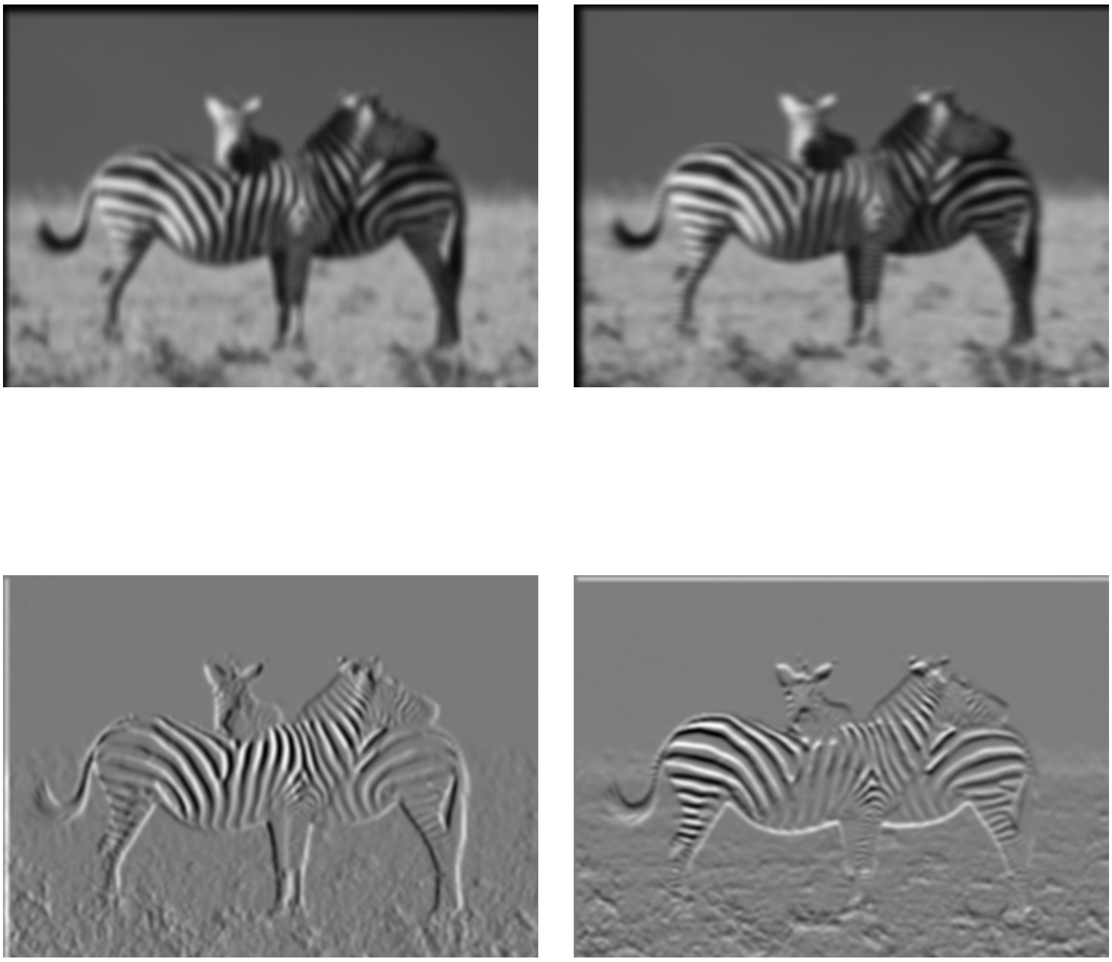


Third level of the 2D wavelet transform (using filters spreaded twice)

In[55]:=
```
{A3x, A3y, W3x, W3y} = WTstep[A2x, A2y, H[2], V[2], h[2], v[2]];
```

In[67]:=
```
GraphicsGrid[{{Image[A3x], Image[A3y]}, {IA[Image[W3x]], IA[Image[W3y]]}}, isl]
```

Out[67]=

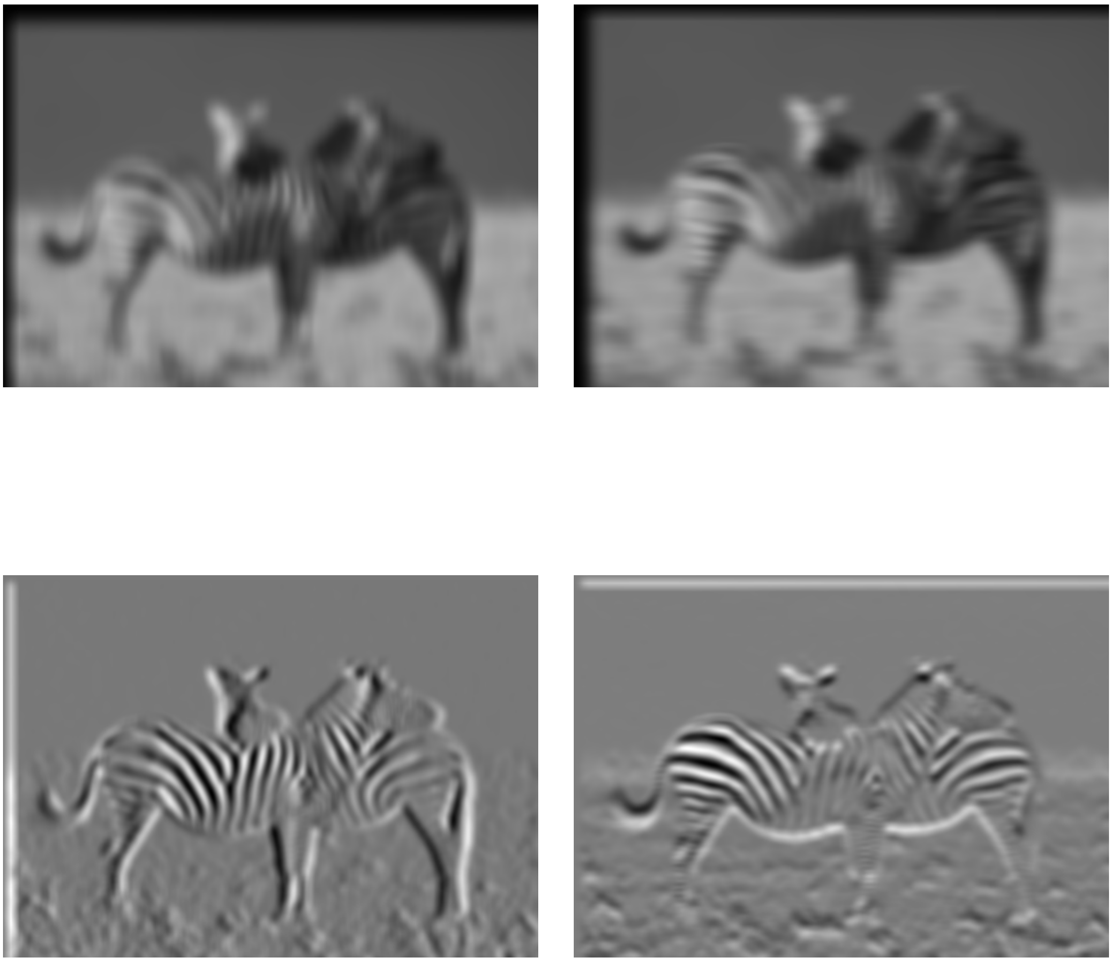


Fourth level of the 2D wavelet transform (using filters spreaded three times)

In[57]:=
```
{A4x, A4y, W4x, W4y} = WTstep[A3x, A3y, H[3], V[3], h[3], v[3]];
```

In[66]:=
```
GraphicsGrid[{{Image[A4x], Image[A4y]}, {IA[Image[W4x]], IA[Image[W4y]]}}, isl]
```

Out[66]=



Clipping the images to ignore boundary effects

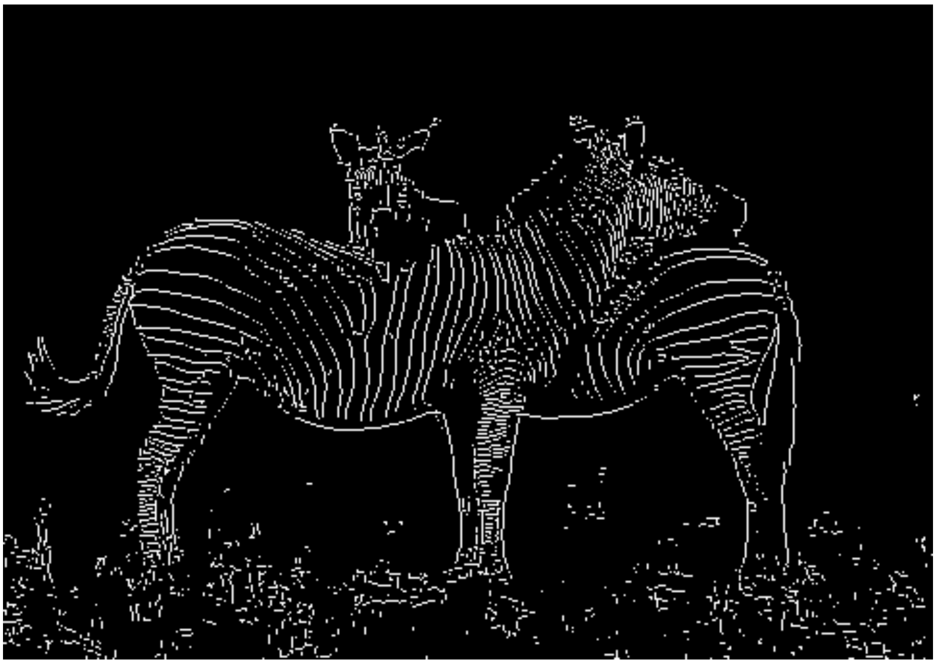
In[59]:=
```
cut = Sequence[{17, imdim[[2]]}, {17, imdim[[1]]}]
```

Out[59]=
```
Sequence[{17, 357}, {17, 500}]
```

Edge detection based on level-1 detail data

In[60]:=
```
kanten1 =
 ImageTake[Image[edges2[W1x, W1y, 2]], cut]
```

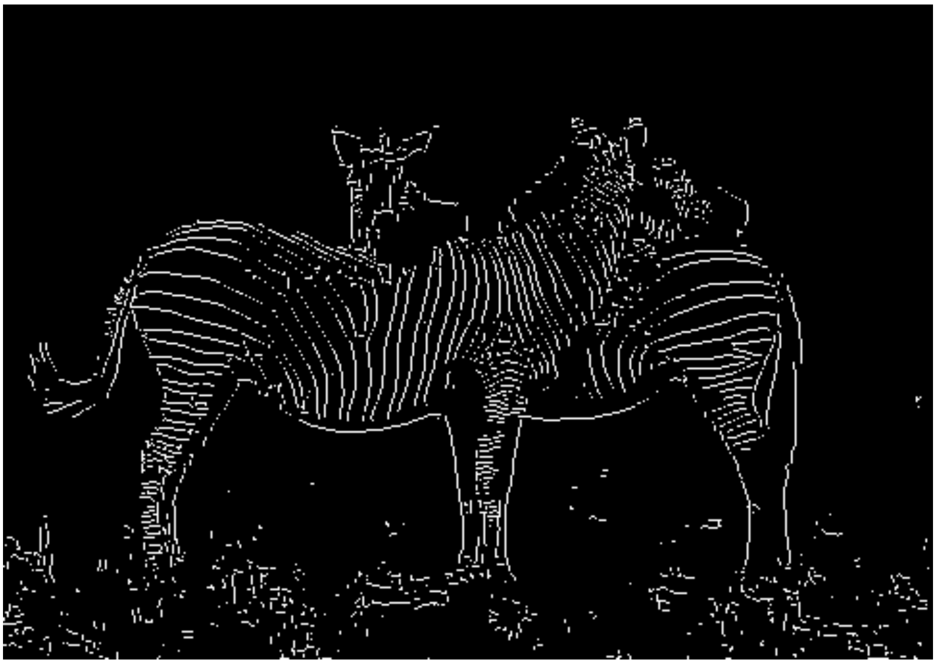
Out[60]=



Edge detection based on level-2 detail data

In[61]:=
```
kanten2 =
 ImageTake[Image[edges2[W2x, W2y, 2]], cut]
```

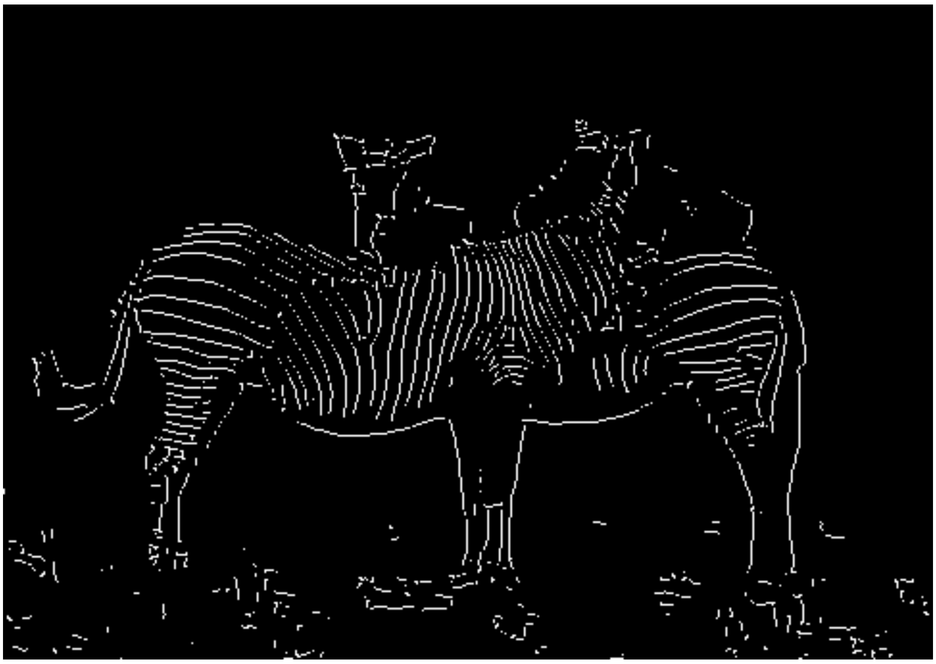
Out[61]=



Edge detection based on level-3 detail data

In[62]:=
```
kanten3 =
 ImageTake[Image[edges2[W3x, W3y, 2]], cut]
```

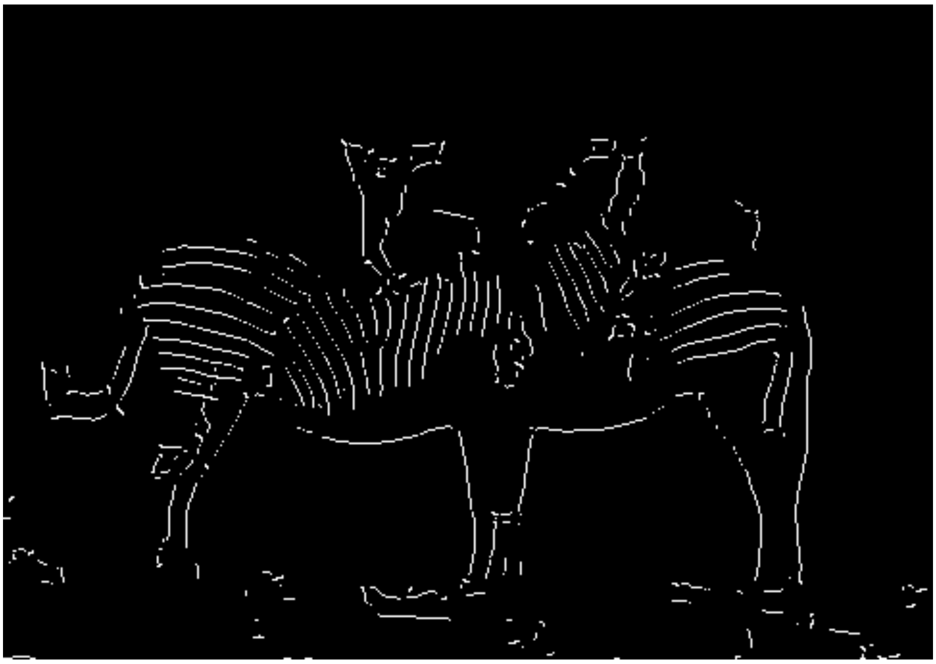
Out[62]=



Edge detection based on level-4 detail data

In[63]:=
```
kanten4 = ImageTake[
  Image[edges2[W4x, W4y, 2]], cut]
```

Out[63]=



```
In[65]:= GraphicsGrid[{{kanten1, kanten2}, {kanten3, kanten4}}, isl]
```

Out[65]=

Operations needed to compensate shifting effects of filtering in different resolution

In[12]:=
```
RD[A_,k_]:=RotateRight[A,k];
RU[A_,k_]:=RotateLeft[A,k];
RR[A_,k_]:=(RD[Aᵀ,k])ᵀ;
RL[A_,k_]:=(RU[Aᵀ,k])ᵀ
```

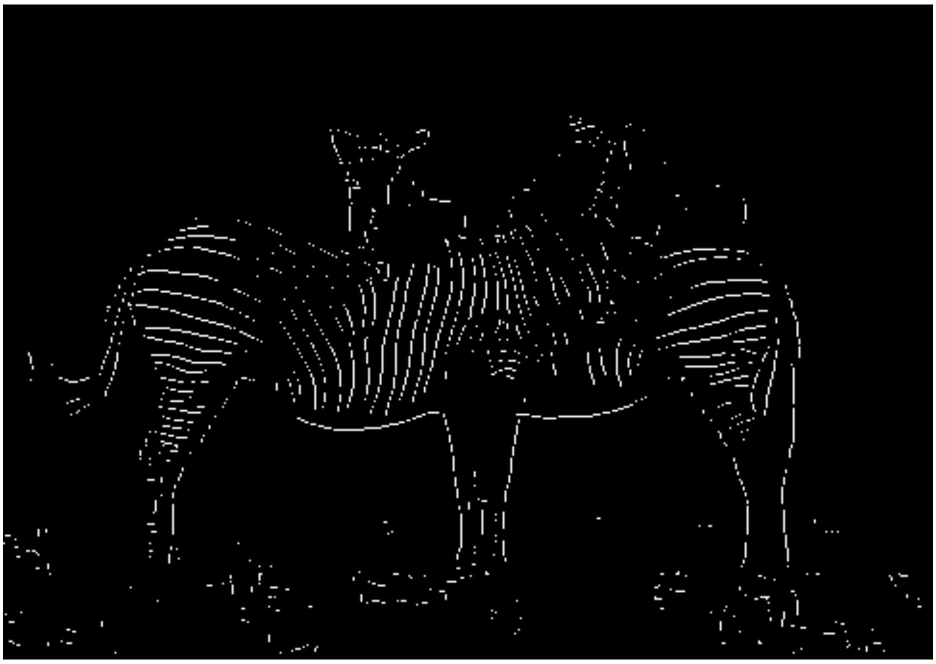
```
RL[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}, 2] // MatrixForm
```

$$\begin{pmatrix} 3 & 1 & 2 \\ 6 & 4 & 5 \\ 9 & 7 & 8 \end{pmatrix}$$

Multiplying the data from edge detection at different resolution

In[70]:=
```
Image[
 (ImageData[kanten1] *
    RU[RL[ImageData[kanten2], 1], 1]) * RU[RL[ImageData[kanten3], 3], 3]
]
```

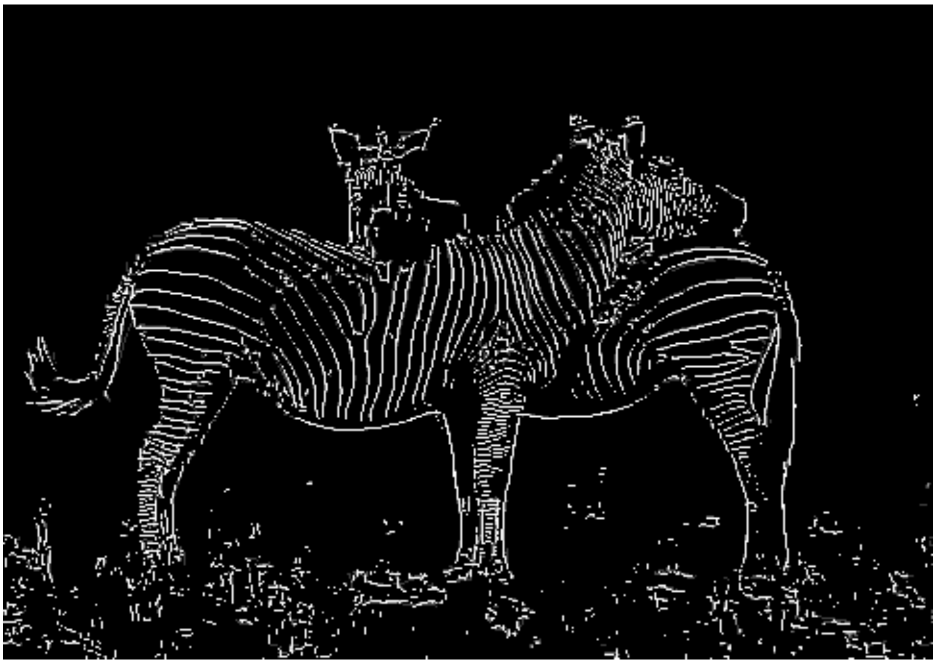
Out[70]=



Weighted sum of the edge detection data

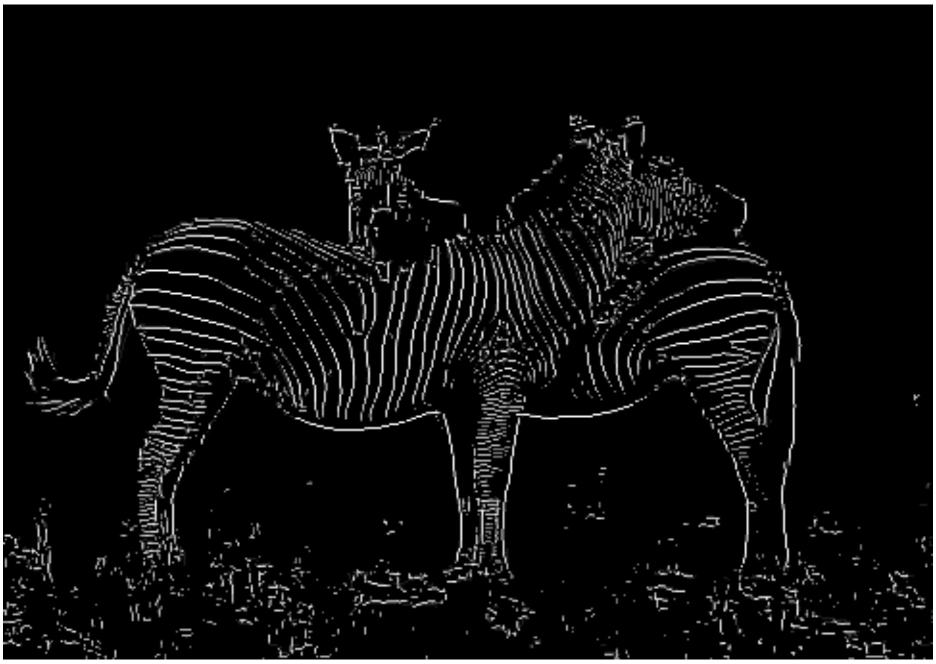
In[71]:=
```
Image[
 ImageData[kanten1] +
  RU[RL[ImageData[kanten2], 1], 1] / 2 +
  RU[RL[ImageData[kanten3], 3], 3] / 3 +
  RU[RL[ImageData[kanten4], 7], 7] / 4]
```

Out[71]=



In[72]:=
```
ImageAdjust[%]
```

Out[72]=



Two-level edge detection

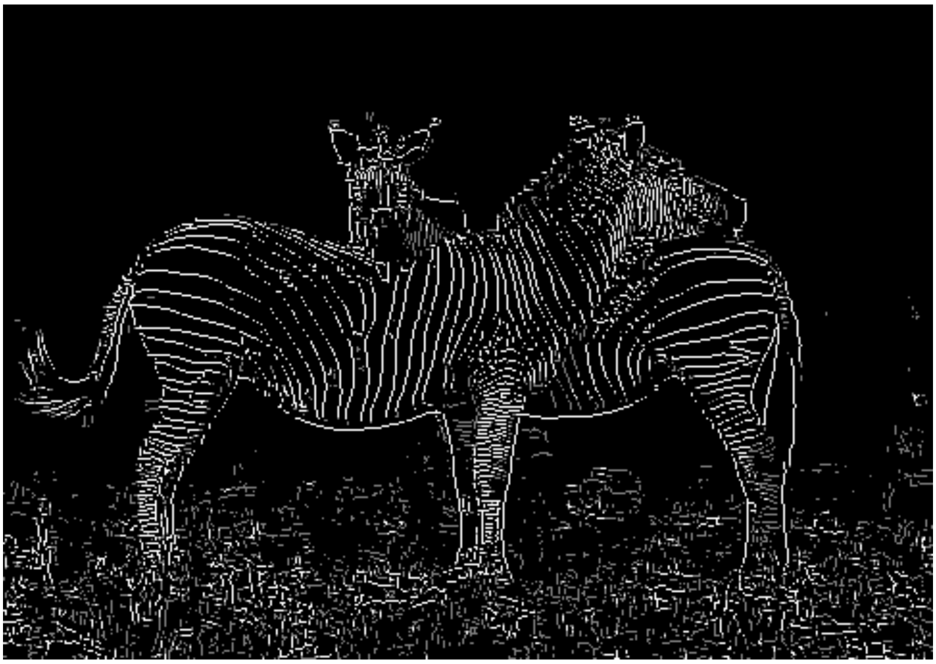
In[16]:=

```
edges3[A_,B_,low_,high_]:=
(*
Detecting edge vertices using gradient analysis.
Input is given as two x- and y-filtered data arrays A and B.
 Output is given in two arrays Clow and Chigh of same
dimensions as A and B. Shown are all vertices which satisfy
the edge vertex criterion and where the
local gradient maximum is ≥ "level"*average gradient
*)
Module[{m,n,Val,Dir,mean,VVal,Clow,Chigh,h,v,
x,x1,x2,y,y1,y2,val,val1,val2},
{m,n}=Dimensions[A];
If[{m,n}≠Dimensions[B],
Throw["Dimensions don't match"]
];
Val=Table[0,{m},{n}];
Dir=Table[0,{m},{n}];
Do[
{h,v}={A[[x,y]],B[[x,y]]};
Val[[x,y]]=N[Sqrt[h^2+v^2]];
Dir[[x,y]]=dir2vec[{h,v}],
{x,1,m},{y,1,n}
];
mean=Mean[Flatten[Val]];
VVal=ArrayPad[Val,{1,1}];
Clow=Table[0,{m},{n}];
Chigh=Table[0,{m},{n}];
Do[
{x1,y1}={x+1,y+1}+Dir[[x,y]];
{x2,y2}={x+1,y+1}-Dir[[x,y]];
val=VVal[[x+1,y+1]];
val1=VVal[[x1,y1]];
val2=VVal[[x2,y2]];
If[
val≥Max[val1,val2]&&val≥low mean,
Clow[[x,y]]=1;
If[val≥high mean,Chigh[[x,y]]=1]],
{x,1,m},{y,1,n}
];
{Clow,Chigh}
]
```

In[73]:=
```
edges3[W1x, W1y, 1, 2];
```

In[74]:=
```
ImageAdjust[ImageTake[Image[%[[1]] + %[[2]]], cut]]
```

Out[74]=



Canny-Iteration

In[17]:=

```
canny[A_,B_,low_,high_,iter_]:=
(*
Uses edges3 for 2-level edge detection
by turning weak (low level) edge points
into strong (high level) edge vertices
*)
Module[{k,m,n,Alow,AAlow,Ahigh,AAhigh,Diff},
{m,n}=Dimensions[A];
{Alow,Ahigh}=edges3[A,B,low,high];
AAlow=ArrayPad[Alow,{1,1}];
AAhigh=ArrayPad[Ahigh,{1,1}];
Print["strong edge vertices: ",Total[Flatten[Ahigh]]];
Print["weak edge vertices: ",Total[Flatten[Alow]]];
Print["further edge vertices: "]
For[k=1,k≤iter,k++,
Diff=AAlow-AAhigh;
Do[
If[
Diff[[x,y]]==1,
Diff[[x,y]]=
Max[Take[AAhigh,{x-1,x+1},{y-1,y+1}]]
],
{x,2,m+1},{y,2,n+1}
];
Print["round ",k," : ",Total[Flatten[Diff]]];
AAhigh=AAhigh+Diff
];
Take[AAhigh,{2,m+1},{2,n+1}]
]
```

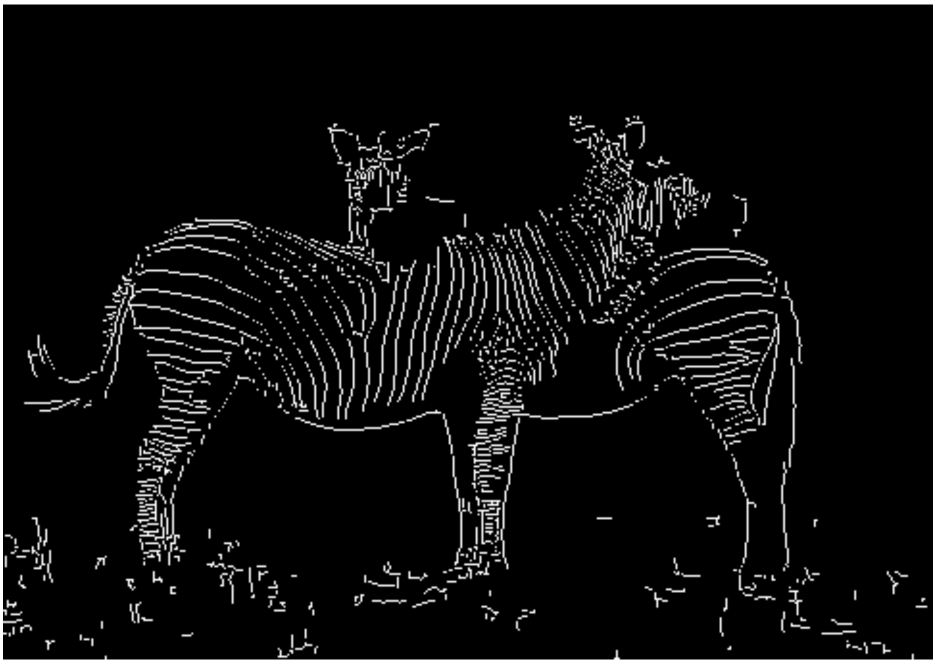
Canny iteration of level-1 wavelet data

In[75]:=
```
canny[W1x, W1y, 1.5, 3, 10];
```

```
strong edge vertices: 6249
weak edge vertices: 11911
further edge vertices:
round 1 : 803
round 2 : 351
round 3 : 221
round 4 : 145
round 5 : 94
round 6 : 66
round 7 : 39
round 8 : 26
round 9 : 21
round 10 : 15
```

In[76]:=
```
c1 = ImageAdjust[ImageTake[Image[%], cut]]
```

Out[76]=



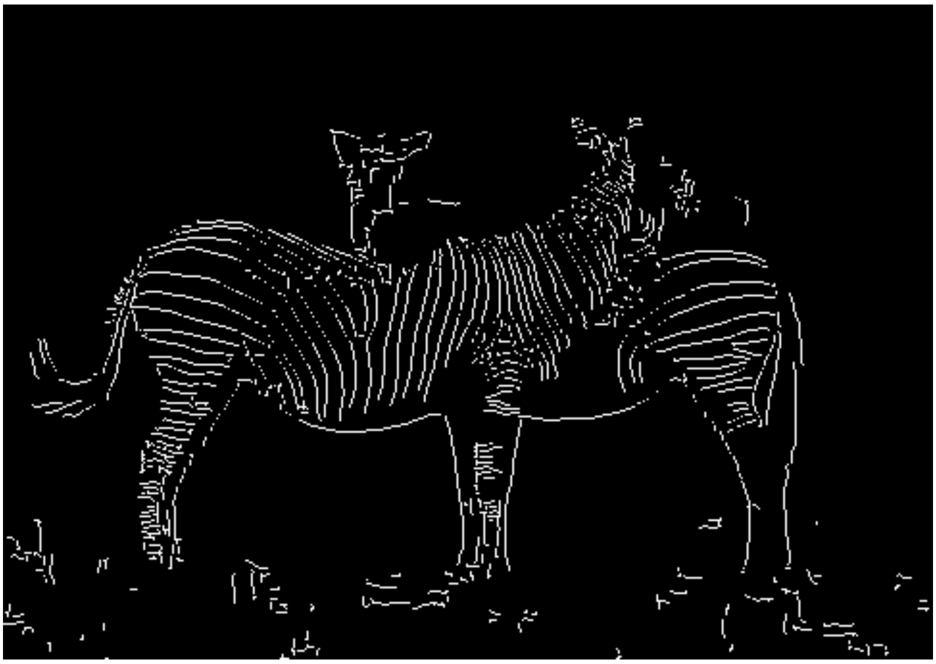
Canny iteration of level-2 wavelet data

In[77]:=
```
canny[W2x, W2y, 1.5, 3, 10];
```

```
strong edge vertices: 5382
weak edge vertices: 10 186
further edge vertices:
round 1 : 589
round 2 : 274
round 3 : 183
round 4 : 128
round 5 : 89
round 6 : 67
round 7 : 45
round 8 : 35
round 9 : 25
round 10 : 20
```

In[78]:=
```
c2 = ImageAdjust[ImageTake[Image[%], cut]]
```

Out[78]=



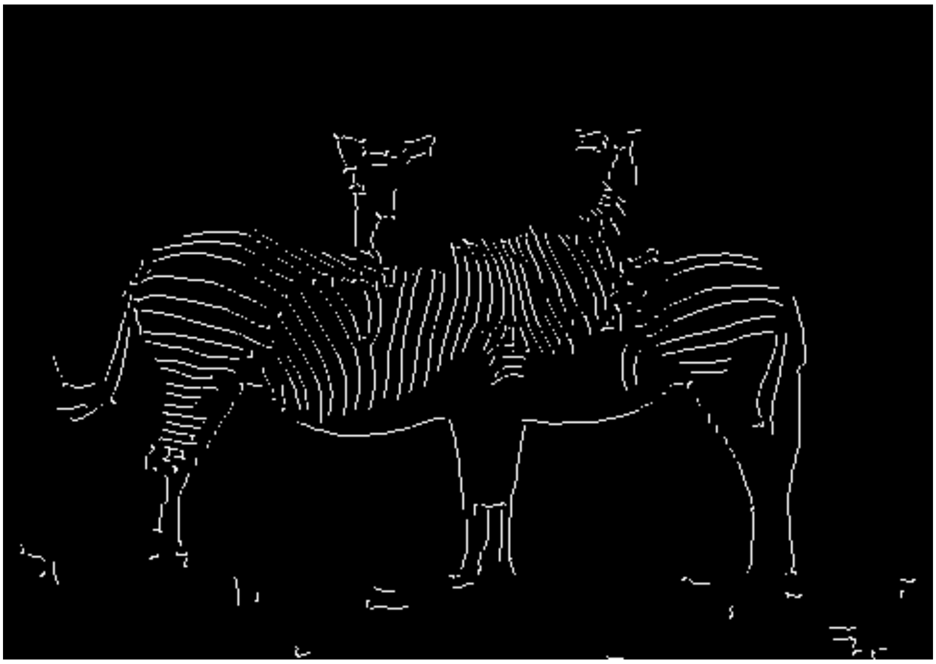
Canny iteration of level-3 wavelet data

In[79]:=
```
canny[W3x, W3y, 1.5, 3, 10];
```

```
strong edge vertices: 4245
weak edge vertices: 7368
further edge vertices:
round 1 : 289
round 2 : 128
round 3 : 81
round 4 : 60
round 5 : 45
round 6 : 39
round 7 : 32
round 8 : 28
round 9 : 22
round 10 : 19
```

In[80]:=
```
c3 = ImageAdjust[ImageTake[Image[%], cut]]
```

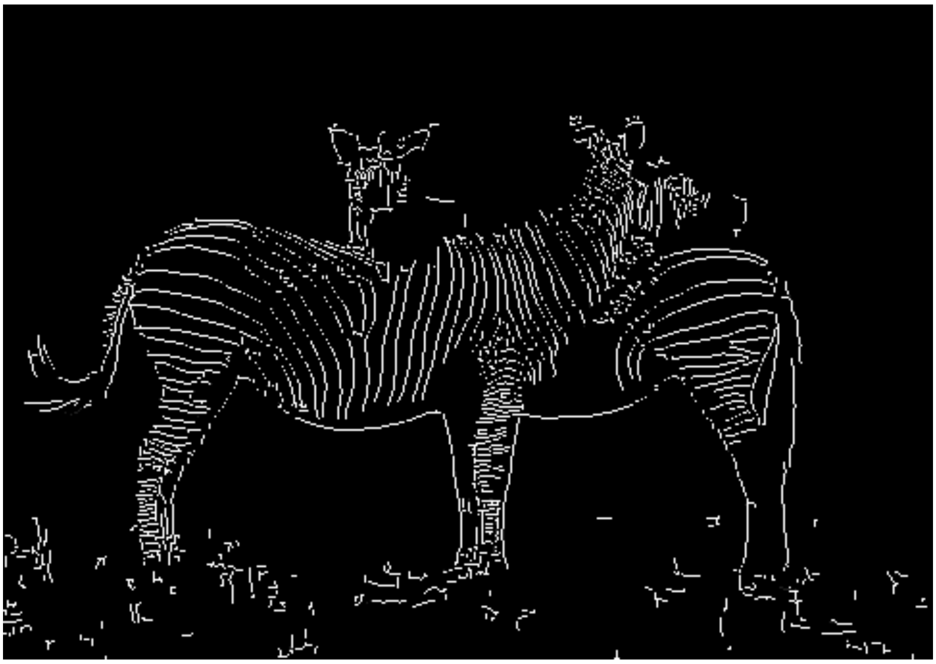
Out[80]=



Weighted average of the data from levels 1-3

In[81]:=
```
Image[
 ImageData[c1] +
  RU[RL[ImageData[c2], 1], 1] / 2
   RU[RL[ImageData[c3], 3], 3] / 4]
```

Out[81]=



In[82]:=
```
EdgeDetect[img]
```

Out[82]=