Le scanner CT

Li-Thiao-Té Sébastien

October 22, 2014

1 Préliminaires

Code chunk 1: «DisplayImage.cpp»

```
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace cv;

int main(int argc, char** argv )
{
   if ( argc != 2 )
   {
      printf("usage: DisplayImage.out <Image_Path>\n");
      return -1;
   }

   Mat image;
   image = imread( argv[1], 1 );

   if ( !image.data )
   {
      printf("No image data \n");
      return -1;
   }
   namedWindow("Display Image", WINDOW_NORMAL );
   imshow("Display Image", image);

   waitKey(0);

   return 0;
}
```

Code chunk 2: «shell»

```
g++ -o DisplayImage DisplayImage.cpp -lm -lopencv_core -lopencv_imgproc -lopencv_highgui
# ./DisplayImage barbara_1.0x.tif
run="all"
```

Interpret with `shell`

```

```

2 Scanner CT : 4pixels, 4 directions

On a besoin de trois éléments :

- construire la matrice du système linéaire

Code chunk 3: «scanner44_make_matrix»

```
Mat make_matrix(int n)
{
  return (Mat_<float>(10,4) << 1, 1, 0, 0,   0, 0, 1, 1,   0, 0, 1, 0,   1, 0, 0, 1,   0, 1, 0, 0,   1, 0, 1, 0,   0, 1, 0, 1,   1, 0, 0, 0,   0
}
```

- générer les observations

Code chunk 4: «scanner44_make_observations»

```
Mat make_observations(Mat A, Mat input)
{
  Mat inputvec = input.reshape(0,4);
  Mat res = A * inputvec;
  return res;
}
```

- reconstruire l'image d'après les observations

Code chunk 5: «scanner44_reconstruct»

```
Mat reconstruct(Mat A, Mat obs)
{
  Mat res = A.inv(DECOMP_SVD) * obs;
  Mat res2 = res.reshape(0,2);
  return res2;
}
```

Code chunk 6: «`scanner44.cpp`»
`scanner44_make_matrix scanner44_make_observations scanner44_reconstruct`

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;

<<scanner44_make_matrix>>
<<scanner44_make_observations>>
<<scanner44_reconstruct>>

int main(int argc, char** argv )
{
   // Construct system matrix
   Mat A = make_matrix(2);

   Mat input = (Mat_<float>(2,2) << 0.1, 0.24, 0., 1.);
   imwrite("scanner44_input.png", input*255);

   Mat obs = make_observations(A, input);
   imwrite("scanner44_obs.png", obs.t()*255);

   Mat output = reconstruct(A, obs);
   imwrite("scanner44_output.png",output*255);

   return 0;
}
```

Code chunk 7: «`shell (part 2)`»

```
rm scanner44*.png
g++ -o scanner44 scanner44.cpp -lm -lopencv_core -lopencv_imgproc -lopencv_highgui
if [ "$run" = "all" ]; then ./scanner44; fi
```

Interpret with `shell`

```
```

3 Scanner CT : Image complète, 4 directions

On a besoin de trois éléments :

- construire la matrice du système linéaire (c'est le point difficile ici).
- générer les observations

Code chunk 8: «`scannern4_make_observations`»

```
Mat make_observations(Mat A, Mat input)
{
  Mat inputvec = input.reshape(0,input.rows*input.cols);
  Mat res = A * inputvec;
  return res;
}
```

- reconstruire l'image d'après les observations

Code chunk 9: «`scannern4_reconstruct`»

```
Mat reconstruct(Mat A, Mat obs, Size s)
{
  Mat res = A.inv(DECOMP_SVD) * obs;
  Mat res2 = res.reshape(0,s.height);
  return res2;
}
```

3.1 Construction de la matrice du système linéaire

On construit la matrice du système direction par direction. L'image est considérée comme une matrice L lignes et C colonnes. Elle est vue comme un vecteur à LC composantes. La matrice A a donc LC colonnes. Le nombre d'équations est $L + (L + C - 1) + C + (L + C - 1) = 3L + 3C - 2$.

Code chunk 10: «`scannern4_make_matrix`»

```
Mat make_matrix(Size s)
{
  int i,j = 0;
  Mat res(3*s.width+3*s.height-2,s.width*s.height,CV_32F,Scalar(0));
```

Ensuite on considère les rayons horizontaux.

Code chunk 11: «scannern4_make_matrix (part 2)»

```
  for (i = 0; i < s.height; i++)
    for (j = 0; j < s.width; j++)
      res.at<float>(i,i*s.width + j) = 1;
```

Ensuite on considère les rayons verticaux.

Code chunk 12: «scannern4_make_matrix (part 3)»

```
  for (j = 0; j < s.width; j++)
    for (i = 0; i < s.height; i++)
      res.at<float>(s.height + j,i*s.width + j) = 1;
```

Les rayons sur la deuxième diagonale

Code chunk 13: «scannern4_make_matrix (part 4)»

```
  for (i = 0; i < s.height; i++)
    for (j = 0; j < s.width; j++)
      res.at<float>(s.height+s.width + i + j, i*s.width + j) = 1;
```

On obtient l'autre diagonale en considérant la symétrie verticale

Code chunk 14: «scannern4_make_matrix (part 5)»

```
  for (i = 0; i < s.height; i++)
    for (j = 0; j < s.width; j++)
      res.at<float>(2*s.height+2*s.width-1 + i + j, i*s.width + s.width - j - 1) = 1;
```

Code chunk 15: «scannern4_make_matrix (part 6)»

```
  // std::cout << res << "\n";
  return res;
}
```

3.2 Main program

Code chunk 16: «scannern4.cpp»

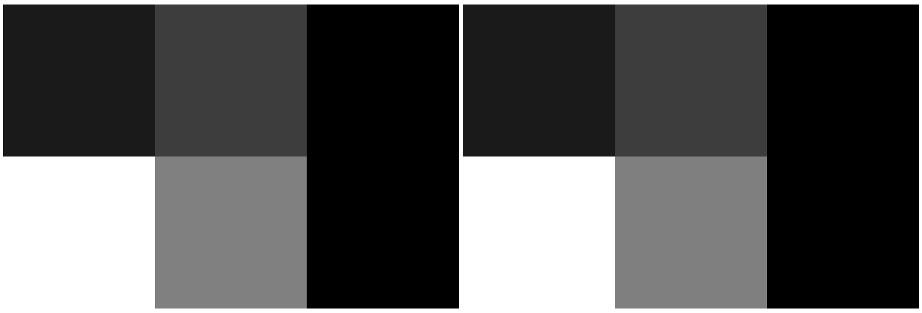
scannern4_make_matrix scannern4_make_observations scannern4_reconstruct

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;

<<scannern4_make_matrix>>
<<scannern4_make_observations>>
<<scannern4_reconstruct>>

int main(int argc, char** argv )
{
    Mat input = (Mat_<float>(2,3) << 0.1, 0.24, 0., 1., 0.5,0);

    if ( !input.data ) { printf("No image data \n"); return -1; }
    imwrite("scannern4_input.png", input*255);

    // Construct system matrix
    Mat A = make_matrix(input.size());

    Mat obs = make_observations(A, input);
    imwrite("scannern4_obs.png", obs.t()*255);

    Mat output = reconstruct(A, obs,input.size());
    imwrite("scannern4_output.png",output*255);

    return 0;
}
```

Code chunk 17: «shell (part 3)»

```
rm scannern4*.png
g++ -o scannern4 scannern4.cpp -lm -lopencv_core -lopencv_imgproc -lopencv_highgui
if [ "$run" = "all" ]; then ./scannern4; fi
```

Interpret with shell



4 Scanner CT : Image complète, p directions

On a besoin de trois éléments :

- construire la matrice du système linéaire (c'est le point difficile ici).
- générer les observations

Code chunk 18: «`scannerct_make_observations`»

```
Mat make_observations(Mat A, Mat input)
{
  Mat inputvec = input.reshape(0,input.rows*input.cols);
  Mat res = A * inputvec;
  return res;
}
```

- reconstruire l'image d'après les observations

Code chunk 19: «`scannerct_reconstruct`»

```
Mat reconstruct(Mat A, Mat obs, Size s)
{
  Mat res = A.inv(DECOMP_SVD) * obs;
  Mat res2 = res.reshape(0,s.height);
  return res2;
}
```

4.1 Construction de la matrice du système linéaire

On construit la matrice du système direction par direction. L'image est considérée comme une matrice L lignes et C colonnes. Elle est vue comme un vecteur à LC composantes. La matrice A a donc `nb_pixels`= LC colonnes. Le nombre d'équations est déterminé par le nombre de directions `nb_dir` considérées et le nombre de rayons `nb_rayons` tirés pour chaque direction.

Code chunk 20: «`scannerct_make_matrix`»

```
Mat make_matrix(Size s, int nb_dir, int nb_rayons)
{
  int i,j = 0;
  Mat res(nb_dir * nb_rayons, s.width*s.height, CV_32F, Scalar(0));
```

Géométrie de l'instrument : pour simplifier les choses, on suppose que les `nb_rayons` sont parallèles, dans une direction donnée par l'angle `a` et centrés (le centre du faisceau passe par l'origine). Les rayons sont écartés d'une unité.

La région d'intérêt est une grille de L lignes et C colonnes. Le pixel $(0, 0)$ est placé dans le repère à la position (x_0, y_0). Le centre du pixel (i, j) est donc à la position $(x_0 + 0.5 + i, y_0 + 0.5 + j)$.

Code chunk 21: «`scannerct_make_matrix (part 2)`»

```
  double i0 = -s.height / 2.0 + 0.5;
  double j0 = -s.width  / 2.0 + 0.5;
```

Pour construire la matrice, on commence par remarquer que chaque pixel de l'image est attribué à un seul rayon (pour une direction fixée). On peut donc faire la boucle sur les pixels de l'image, et calculer l'indice du rayon correspondant.

Code chunk 22: «`scannerct_make_matrix (part 3)`»

```
  double a = 0;
  double d = 0;
  double ii = 0;

  for (a = 0; a < nb_dir; a++) {
    double cosa = cos(a*3.1415/nb_dir);
    double sina = sin(a*3.1415/nb_dir);

    for (i = 0; i < s.height; i++)
      for (j = 0; j < s.width; j++)
      {
        d = (j0+j) * cosa - (i0 + i) * sina;
        ii = round(d+(nb_rayons-1)/2);
        if (ii >=0 && ii < nb_rayons) res.at<float>(a*nb_rayons+ii,i*s.width + j) = 1;
      }
  }
```

Code chunk 23: «scannerct_make_matrix (part 4)»

```
  // std::cout << res << "\n";
  return res;
}
```

4.2 Test program

Code chunk 24: «scannerct0.cpp»

scannerct_make_matrix scannerct_make_observations scannerct_reconstruct

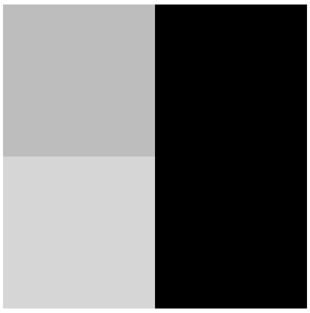
```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;

<<scannerct_make_matrix>>
<<scannerct_make_observations>>
<<scannerct_reconstruct>>

int main(int argc, char** argv )
{
    // Mat input = imread("barbara_1.0x.tif", 1);
    Mat input = (Mat_<float>(2,3) << 0.1, 0.24, 0., 1., 0.5,0);

    if ( !input.data ) { printf("No image data \n"); return -1; }
    imwrite("scannerct0_input.png", input*255);

    // Construct system matrix
    Mat A = make_matrix(input.size(),2,3);
    std::cout << "matrix A\n" << A << "\n";

    Mat obs = make_observations(A, input);
    std::cout << "observations\n" << obs << "\n";

    imwrite("scannerct0_obs.png", obs.reshape(0,2)*255);

    Mat output = reconstruct(A, obs,input.size());
    imwrite("scannerct0_output.png",output*255);
    std::cout << "reconstruction\n" << output << "\n";

    return 0;
}
```

Code chunk 25: «shell (part 4)»

```
rm scannerct*.png
g++ -o scannerct0 scannerct0.cpp -lm -lopencv_core -lopencv_imgproc -lopencv_highgui
if [ "$run" = "all" ]; then ./scannerct0; fi
```

Interpret with shell

```
matrix A
[1, 0, 0, 1, 0, 0;
  0, 1, 0, 0, 1, 0;
  0, 0, 1, 0, 0, 1;
  0, 0, 0, 1, 0, 0;
  1, 1, 0, 0, 1, 1;
  0, 0, 1, 0, 0, 0]
observations
[1.1;
  0.74000001;
  0;
  1;
  0.83999997;
  0]
reconstruction
[0.099999979, 0.34258702, 2.7150953e-08;
  1.0000001, 0.39741302, -7.233097e-08]
```


4.3 Main program

Code chunk 26: «scannerct.cpp»
scannerct_make_matrix scannerct_make_observations scannerct_reconstruct

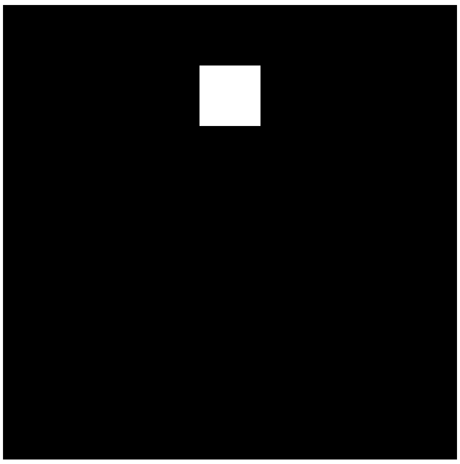
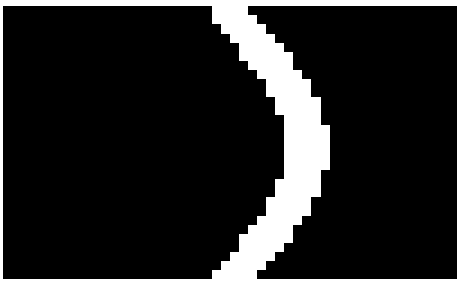
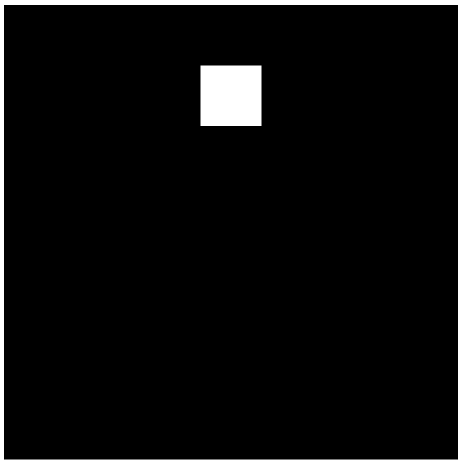
```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;

<<scannerct_make_matrix>>
<<scannerct_make_observations>>
<<scannerct_reconstruct>>

int main(int argc, char** argv )
{
    Mat input0 = imread(argv[1],0);
    if ( !input0.data ) { printf("No image data \n"); return -1; }
    Mat input;
    input0.convertTo(input,CV_32F);
    imwrite("scannerct_input.png", input);

    // Construct system matrix
    int nb_dir = atoi(argv[2]);
    int nb_rayons = atoi(argv[3]);
    std::cout << "make matrix\n";
    Mat A = make_matrix(input.size(),nb_dir,nb_rayons);

    std::cout << "make observations\n";
    Mat obs = make_observations(A, input);
    imwrite("scannerct_obs.png", obs.reshape(0,nb_dir));

    std::cout << "reconstruct\n";
    Mat output = reconstruct(A, obs,input.size());
    imwrite("scannerct_output.png",output);

    return 0;
}
```

Code chunk 27: «shell (part 5)»

```
g++ -o scannerct scannerct.cpp -lm -lopencv_core -lopencv_imgproc -lopencv_highgui
if [ "$run" = "all" ]; then ./scannerct square_top.png 30 50; fi
```

Interpret with shell

```
make matrix
make observations
reconstruct
```

5 Rétroprojection simple

Pour effectuer une rétroprojection simple, on reporte les observations sur la trajectoire des rayons. On garde l'étape de construction de la matrice A, car on en a besoin pour créer des observations, mais la reconstruction est différente.

On a besoin de trois éléments :

- construire la matrice du système linéaire.

Code chunk 28: «scannerretro_make_matrix»

```
Mat make_matrix(Size s, int nb_dir, int nb_rayons)
{
  int i,j = 0;
  Mat res(nb_dir * nb_rayons, s.width*s.height, CV_32F, Scalar(0));
  double i0 = -s.height / 2.0 + 0.5;
  double j0 = -s.width  / 2.0 + 0.5;
  double a = 0;
  double d = 0;
  double ii = 0;

  for (a = 0; a < nb_dir; a++) {
    double cosa = cos(a*3.1415/nb_dir);
    double sina = sin(a*3.1415/nb_dir);

    for (i = 0; i < s.height; i++)
      for (j = 0; j < s.width; j++)
      {
        d = (j0+j) * cosa - (i0 + i) * sina;
        ii = round(d+(nb_rayons-1)/2);
        if (ii >=0 && ii < nb_rayons) res.at<float>(a*nb_rayons+ii,i*s.width + j) = 1;
      }
  }
  // std::cout << res << "\n";
  return res;
}
```

- générer les observations

Code chunk 29: «scannerretro_make_observations»

```
Mat make_observations(Mat A, Mat input)
{
  Mat inputvec = input.reshape(0,input.rows*input.cols);
  Mat res = A * inputvec;
  return res;
}
```

- reconstruire l'image d'après les observations (c'est le point qui change).

5.1 Reconstruction par rétroprojection

Code chunk 30: «scannerretro_reconstruct»

```
Mat reconstruct(Mat obs, Size s, int nb_dir, int nb_rayons)
{
  Mat res(s,CV_32F,Scalar(0));

  int i,j = 0;
  double i0 = -s.height / 2.0 + 0.5;
  double j0 = -s.width  / 2.0 + 0.5;
  double a = 0;
  double d = 0;
  double ii = 0;

  for (a = 0; a < nb_dir; a++) {
    double cosa = cos(a*3.1415/nb_dir);
    double sina = sin(a*3.1415/nb_dir);

    for (i = 0; i < s.height; i++)
      for (j = 0; j < s.width; j++)
      {
        d = (j0+j) * cosa - (i0 + i) * sina;
        ii = round(d+(nb_rayons-1)/2);
        if (ii >=0 && ii < nb_rayons) res.at<float>(i,j) += obs.at<float>(a*nb_rayons+ii,0) / nb_dir;
      }
  }
  return res;
}
```

5.2 Main program

Code chunk 31: «scannerretro.cpp»

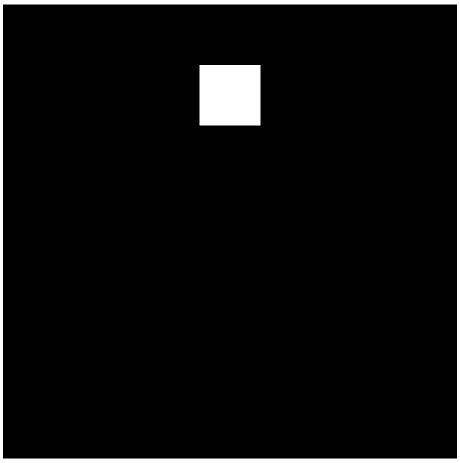
scannerretro_make_matrix scannerretro_make_observations scannerretro_reconstruct

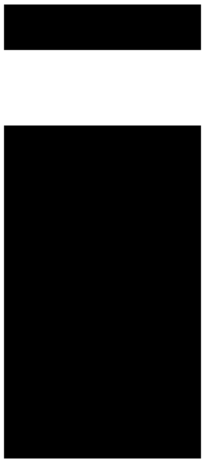
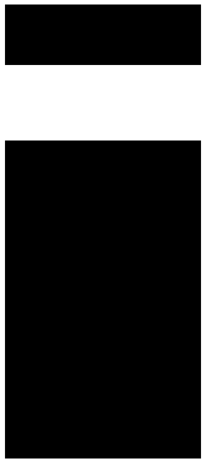
```
#include <stdio.h>
#include <string>
#include <opencv2/opencv.hpp>
using namespace cv;

<<scannerretro_make_matrix>>
<<scannerretro_make_observations>>
<<scannerretro_reconstruct>>

int main(int argc, char** argv )
{
    Mat input0 = imread(argv[1],0);
    if ( !input0.data ) { printf("No image data \n"); return -1; }
    Mat input;
    input0.convertTo(input,CV_32F);
    imwrite(std::string(argv[4]) + "_scannerretro_input.png", input);

    // Construct system matrix
    int nb_dir = atoi(argv[2]);
    int nb_rayons = atoi(argv[3]);
    Mat A = make_matrix(input.size(),nb_dir,nb_rayons);

    Mat obs = make_observations(A, input);
    imwrite(std::string(argv[4]) + "_scannerretro_obs.png", obs.reshape(0,nb_dir));

    Mat output = reconstruct(obs, input.size(), nb_dir, nb_rayons);
    imwrite(std::string(argv[4]) + "_scannerretro_output.png",output);

    return 0;
}
```

Code chunk 32: «shell (part 6)»

```
g++ -o scannerretro scannerretro.cpp -lm -lopencv_core -lopencv_imgproc -lopencv_highgui
if [ "$run" = "all" ]; then ./scannerretro square_top.png 2 50 test; fi
```

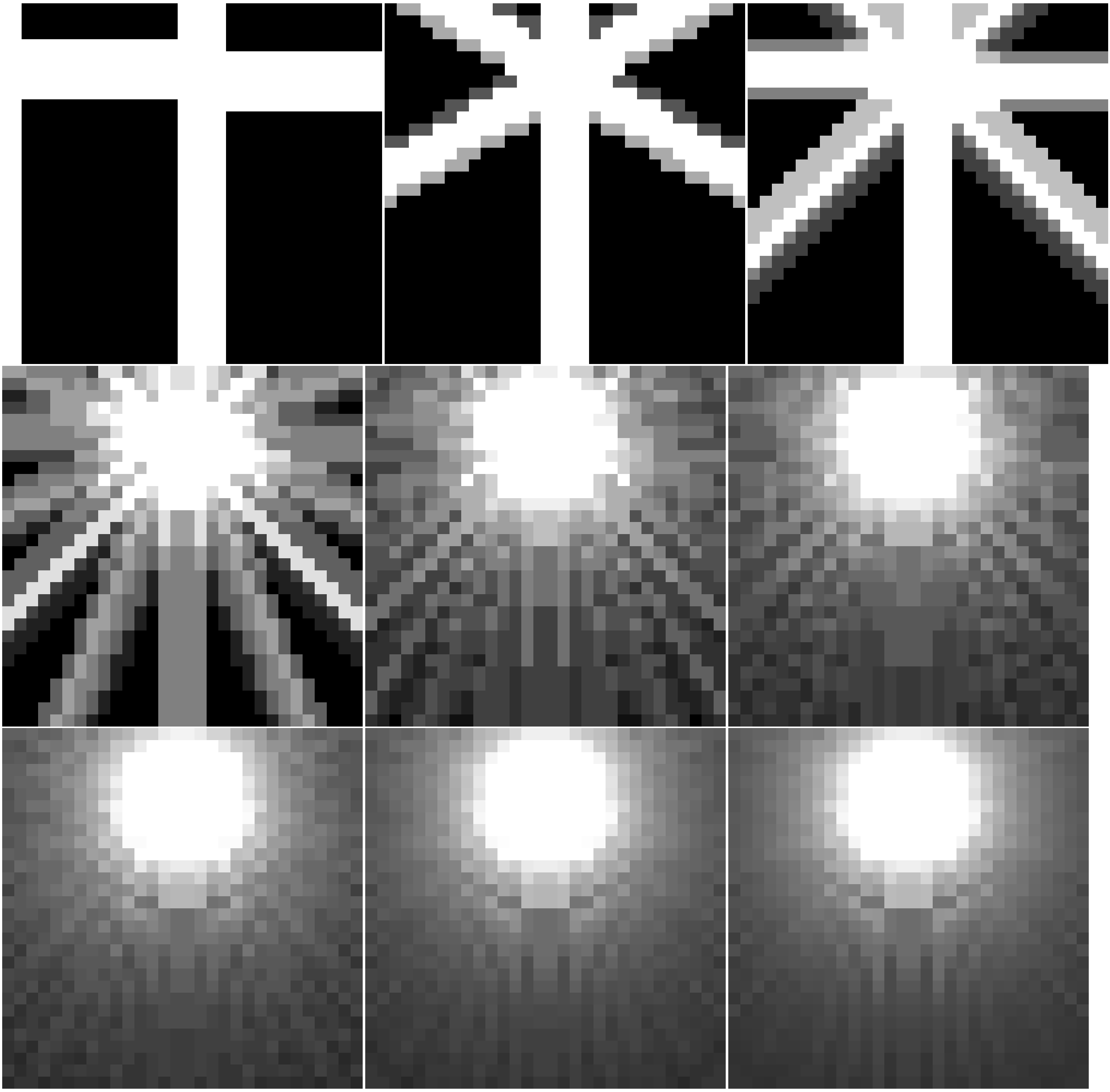
Interpret with shell



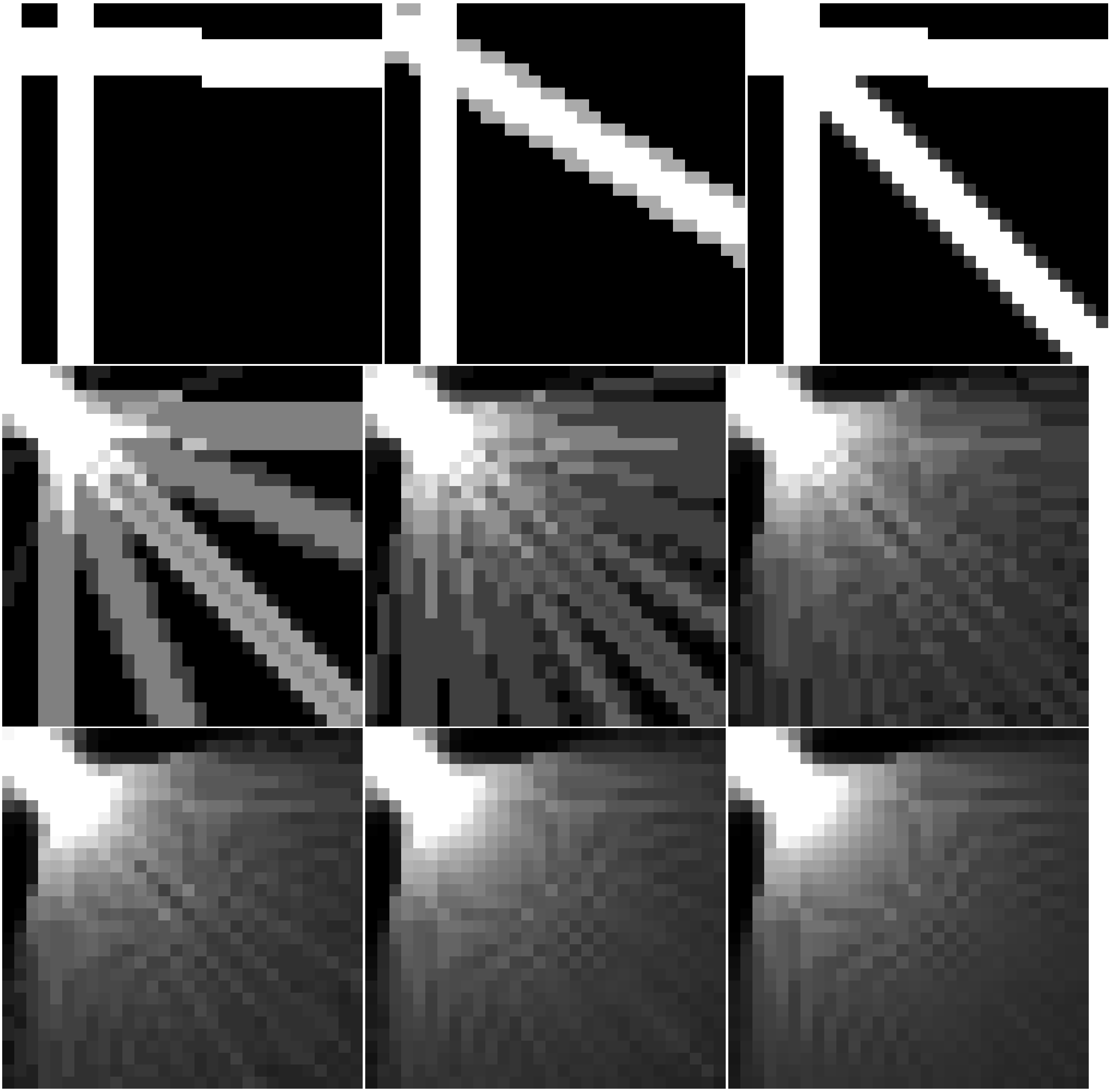
Code chunk 33: «benchmark»

```
rm dirs*_scannerretro*.png
nb_dirs="2 3 4 8 16 32 64 128 256"
for i in $nb_dirs; do
  ./scannerretro square_top.png $i 50 "dirs$i";
  echo -n "\\includegraphics[width=6.5cm]{dirs$i"; echo "_scannerretro_output.png}"
done
```



Code chunk 34: «benchmark (part 2)»

```
nb_dirs="2 3 4 8 16 32 64 128 256"
for i in $nb_dirs; do
  ./scannerretro square_corner.tif $i 25 "dirsc$i";
  echo -n "\\includegraphics[width=6.5cm]{dirsc$i"; echo "_scannerretro_output.png}"
done
```



6 Rétroprojection filtrée

Par rapport à la rétroprojection, il faut simplement ajouter une étape de filtrage des observations.

On a besoin de quatre éléments :

- construire la matrice du système linéaire.

Code chunk 35: «scannerfiltre_make_matrix»

scannerct_make_matrix

```
<<scannerct_make_matrix>>
```

- générer les observations

Code chunk 36: «scannerfiltre_make_observations»

```
Mat make_observations(Mat A, Mat input)
{
  Mat inputvec = input.reshape(0,input.rows*input.cols);
  Mat res = A * inputvec;
  return res;
}
```

- filtrage des observations (c'est le point difficile ici)
- rétroprojection (idem rétroprojection simple)

Code chunk 37: «scannerfiltre_reconstruct»
scannerretro_reconstruct

```
<<scannerretro_reconstruct>>
```

6.1 Reconstruction par rétroprojection filtrée

En rétroprojection filtrée, la partie rétroprojection est identique

Code chunk 38: «scannerfiltre_filtre»

```
Mat filtre_observations(Mat obs, int nb_dir, int nb_rayons)
{
  Mat obsf = obs.reshape(0,nb_dir);
  Mat planes[] = {Mat_<float>(obsf), Mat::zeros(obsf.size(), CV_32F)};
  Mat complexI;
  merge(planes, 2, complexI);

  // Fourier transform on the rows (direction-wise)
  dft(complexI,complexI,DFT_ROWS);

  // Compute the filter
  Mat pattern(1,nb_rayons,CV_32F, Scalar(1));
  int i;
  for (i = 0; i<nb_rayons; i++) pattern.at<float>(0,i) = 1.0 * i / nb_rayons / 2;
  Mat temp[] = {repeat(pattern,nb_dir,1),repeat(pattern,nb_dir,1)};
  Mat patternI;
  merge(temp, 2, patternI);

  // Apply the filter
  multiply(complexI,patternI,complexI);

  // std::cout << temp << "\n";
  // Inverse Fourier transform
  dft(complexI,complexI,DFT_ROWS + DFT_INVERSE + DFT_SCALE);
  split(complexI, planes);
  magnitude(planes[0],planes[1],planes[0]);

  return planes[0].reshape(0,nb_dir*nb_rayons);
}
```

6.2 Main program

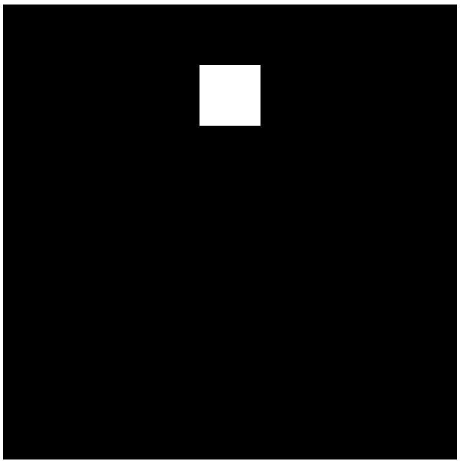
Code chunk 39: «scannerfiltre.cpp»
scannerfiltre_make_matrix scannerfiltre_make_observations scannerfiltre_filtre scannerfiltre_reconstruct

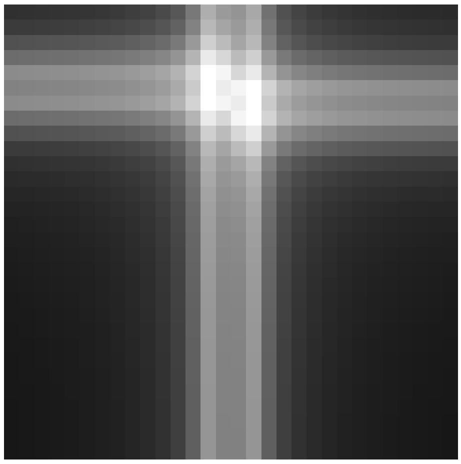
```
#include <stdio.h>
#include <string>
#include <opencv2/opencv.hpp>
using namespace cv;

<<scannerfiltre_make_matrix>>
<<scannerfiltre_make_observations>>
<<scannerfiltre_filtre>>
<<scannerfiltre_reconstruct>>

int main(int argc, char** argv )
{
    Mat input0 = imread(argv[1],0);
    if ( !input0.data ) { printf("No image data \n"); return -1; }
    Mat input;
    input0.convertTo(input,CV_32F);
    imwrite(std::string(argv[4]) + "_scannerfiltre_input.png", input);

    // Construct system matrix
    int nb_dir = atoi(argv[2]);
    int nb_rayons = atoi(argv[3]);
    Mat A = make_matrix(input.size(),nb_dir,nb_rayons);

    Mat obs = make_observations(A, input);
    imwrite(std::string(argv[4]) + "_scannerfiltre_obs.png", obs.reshape(0,nb_dir));

    Mat obs_filtre = filtre_observations(obs,nb_dir,nb_rayons);
    Mat output = reconstruct(obs_filtre, input.size(), nb_dir, nb_rayons);
    imwrite(std::string(argv[4]) + "_scannerfiltre_output.png",output);

    return 0;
}
```

Code chunk 40: «shell (part 7)»

```
g++ -o scannerfiltre scannerfiltre.cpp -lm -lopencv_core -lopencv_imgproc -lopencv_highgui
if [ "$run" = "all" ]; then ./scannerfiltre square_top.png 2 50 test; fi
```

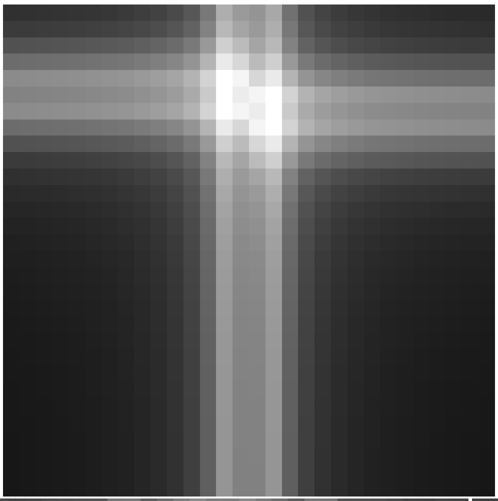
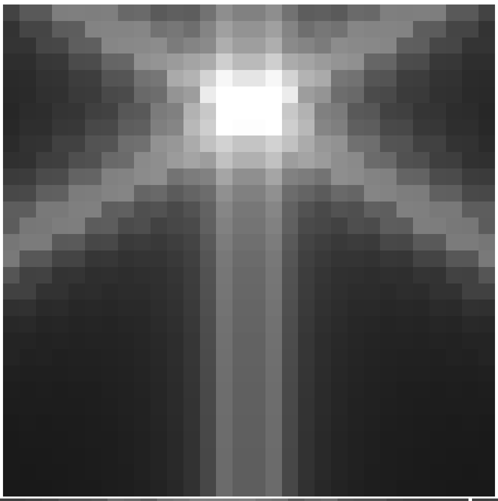
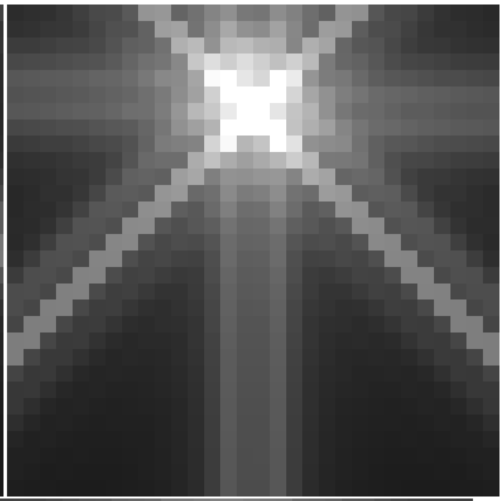
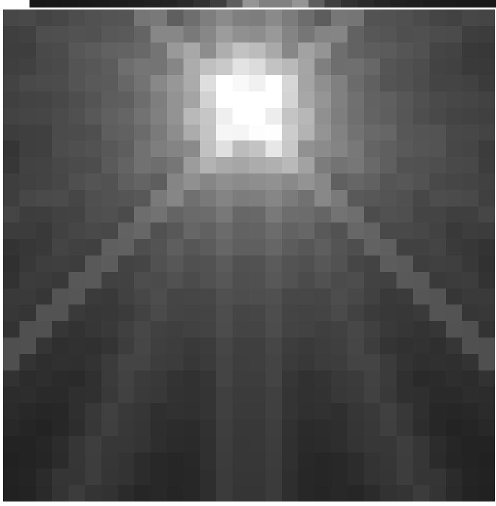
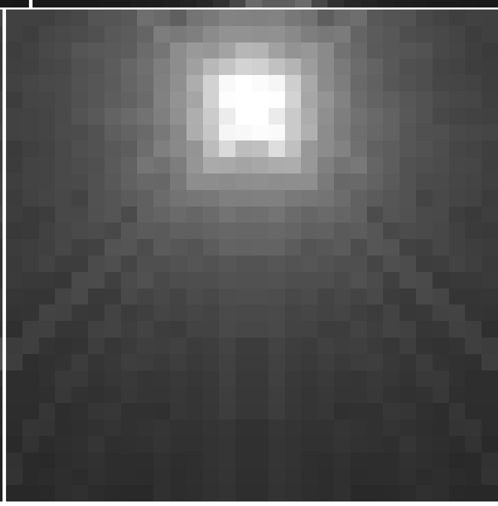
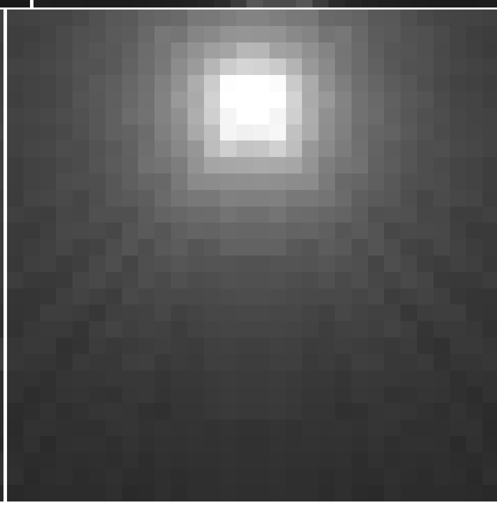
Interpret with shell

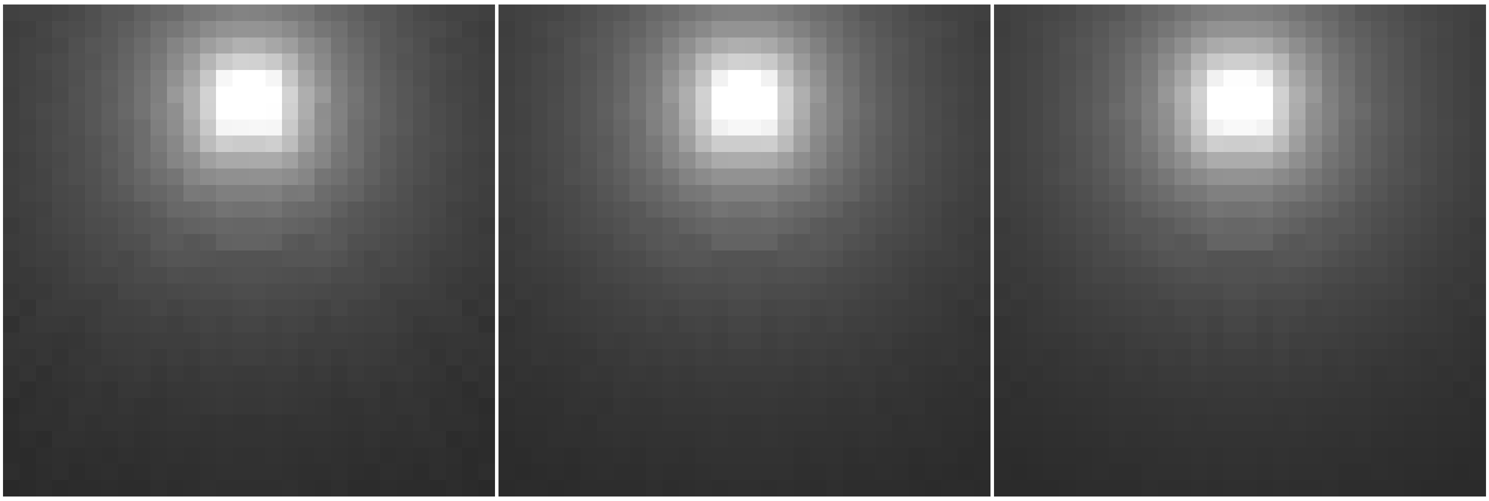




Code chunk 41: «benchmark (part 3)»

```
rm dirs*_scannerfiltre*.png
nb_dirs="2 3 4 8 16 32 64 128 256"
for i in $nb_dirs; do
  ./scannerfiltre square_top.png $i 50 "dirs$i";
  echo -n "\\includegraphics[width=6.5cm]{dirs$i"; echo "_scannerfiltre_output.png}"
done
```



Code chunk 42: «benchmark (part 4)»

```
nb_dirs="2 3 4 8 16 32 64 128 256"
for i in $nb_dirs; do
  ./scannerfiltre square_corner.tif $i 25 "dirsc$i";
  echo -n "\\includegraphics[width=6.5cm]{dirsc$i"; echo "_scannerfiltre_output.png}"
done
```