IFT 6145 – Vision tridimensionnelle

TP3 - Structured-light 3D

student: Zhibin Lu

Use salvi papers as references.
For testing use your own transformation of the images (ex: imagerotate). Do at least one test with a very big distorsion (ex: imagetransformation)
1. code graycode (real GRAY code)
2. in the presence of some noise, compare real gray code with non gray code.
3. code phase shift (sin wave stuff)
use more shift than 4, say 10, and compare 4 with 10 in the presence of noise. -> use equation 8 of salvi-2010
for phase unwrapping, use your previous graycode solution.
4. optinal : try with camera + projector en real life

Explanation: I did 3 types of experiments, planes, distortion effects, and projection effects. But it should be noted that the function of the distortion I am using, it actually copies the image on the left to the right (the actual right has been lost), while doing the distortion. This effect also affects the all codes images for the distortion part. This is just for the case of experimental distortion. But in the final projection part, the function projectionPattern used simulates the real situation and there is no such problem.

Images:

In[•]:= img =  ;

img2 = 

;

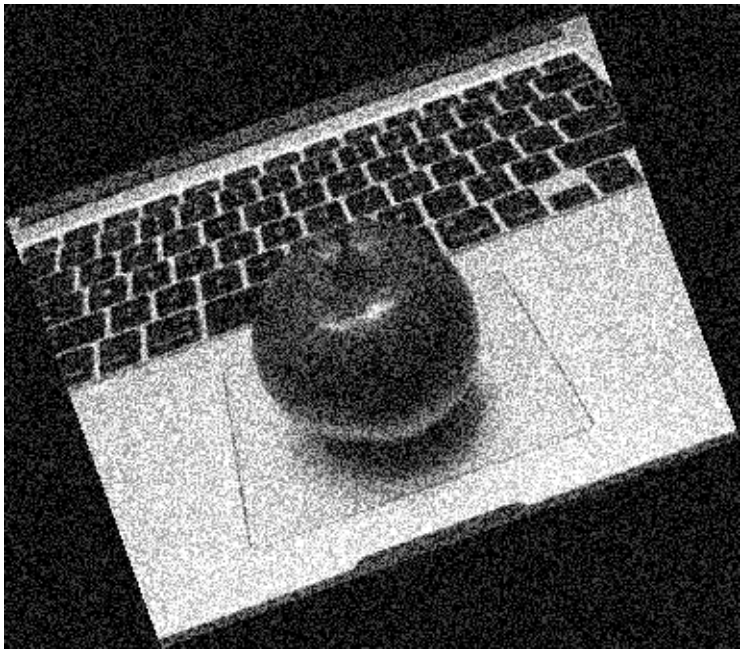
```
{w, h} = ImageDimensions[img]
         dimensions d'image
(*Image[ImageData[img][[;;,;;,3]]]*)
  image  données d'image
(*Image[Map[Min,ImageData[img],{2}]]*)
  image  app··  mi···  données d'image
noisify[im_, v_] :=
  Image[Map[(Max[0, Min[1, # + RandomReal[{-v, v}]]]) &, ImageData[im], {2}]];
  image  appl···  maxi···  minimum  nombre réel aléatoire  données d'image
camera[im_, v_, deg_] :=
  noisify[ImageRotate[im, deg], v];
          fais pivoter image
```

Out[•]= {320, 240}

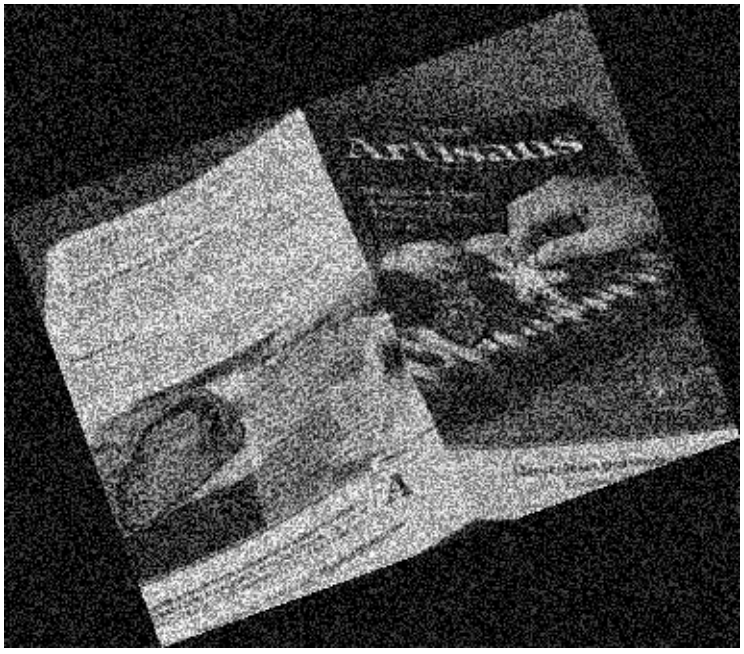
In[ ]:=
```
(*parameters*)
nAddr = Length[IntegerDigits[w - 1, 2]]
        longueur chiffres d'entier
noisyValue = 0.3;
rotDegree = 20. Degree;
                degré
camImg = camera[img, noisyValue, rotDegree]
disCamImg = camera[img2, noisyValue, rotDegree]
```

Out[ ]= 9

Out[ ]=



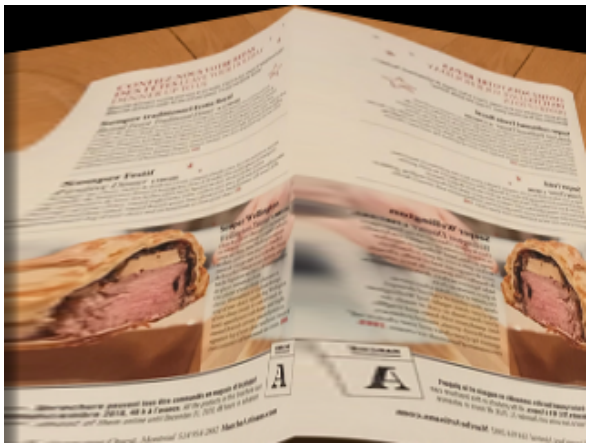
Out[ ]=



In[ ]:=
```
(*distortion image*)
f[x_, y_] := With[
                 avec
   {r = Abs[x - .5], R = .5},
        valeur absolue
   {x - x * r / R, y + y * r / 2 R}];
f2[x_, y_] := With[{r = N@Sqrt[(x - .5) ^ 2 + (y - .5) ^ 2],
                  avec      ··· racine carrée
    a = ArcTan[x - .5, y - .5], R = .5}, rn = Sqrt[r * R];
        arc tangente                          racine carrée
   {rn * Cos[a] + .5, rn * Sin[a] + .5}];
        cosinus              sinus
distortCamera[im_, v_, deg_] :=
  noisify[ImageRotate[ImageTransformation[im, f @@ # &], deg], v];
           fais pivoter im··· transformée d'image
distortCamera2[im_, v_, deg_] :=
  noisify[ImageRotate[ImageTransformation[im, f2 @@ # &], deg], v];
           fais pivoter im··· transformée d'image
Print["Test the distortion function:"];
imprime
ImageTransformation[img2, f @@ # &]
transformée d'image
Print["original image:"];
imprime
img2
(*ImageTransformation[img,f2@@#&]*)
  transformée d'image
(*disCamImg=distortCamera[img,noisyValue,rotDegree]*)
```

Test the distortion function:

Out[ ]=



original image:

Out[ ]=

distortCamera function actually copies the image on the left to the right (the actual right has been lost), while doing the distortion.

---

1. gray code function and binary code function

In[ ]:=
```
(*output: binary of row+invers of row+col+invers of col *)
binarycode[x_, y_, len_] := Module[{xc, yc}, (
                                 module
    xc = IntegerDigits[x, 2, len];
         chiffres d'entier
    yc = IntegerDigits[y, 2, len];
         chiffres d'entier
    (*1-xc,1-yc: In order to obtain the light intensity difference between
                 dans
       the 0 and 1 of the pixel,it is convenient to decode. *)
    Join[xc, 1 - xc, yc, 1 - yc]
    joins
   )];
(*input a sequence of gray degree
 like: (x coordinate,1-x coordinate,y coordinate,1-y coordinate*)
(*output (no. of row, no. of col, 0) *)
binDecode[bits_, len_] :=
  {FromDigits[Sign[bits[[1 ;; len]] - bits[[len + 1 ;; 2 * len]]] /. {-1 → 0}, 2],
   depuis chiffres signe
   FromDigits[Sign[bits[[2 * len + 1 ;; 3 * len]] - bits[[3 * len + 1 ;; 4 * len]]] /.
   depuis chiffres signe
     {-1 → 0}, 2],
   0};
graycode[b_] :=
  BitXor[b, BitShiftRight[b, 1]];
  ou exlusif ···  déplace bits à droite
```

```
graycode[x_, y_, len_] := Module[{xc, yc}, (
                          module
    (*https://fr.wikipedia.org/wiki/Code_de_Gray*)
    xc = BitXor[x, BitShiftRight[x, 1]];
         ou exlusif ··· déplace bits à droite
    yc = BitXor[y, BitShiftRight[y, 1]];
         ou exlusif ··· déplace bits à droite
    xc = IntegerDigits[xc, 2, len];
         chiffres d'entier
    yc = IntegerDigits[yc, 2, len];
         chiffres d'entier
    (*1-xc,1-yc: In order to obtain the light intensity difference between
                 dans
       the 0 and 1 of the pixel,it is convenient to decode. *)
    Join[xc, 1 - xc, yc, 1 - yc]
    joins
   )];
grayDecode[b_] := Module[{v, w}, (
                  module
    (*https://fr.wikipedia.org/wiki/Code_de_Gray*)
    v = IntegerDigits[b, 2, 16];
        chiffres d'entier
    w = Table[BitXor @@ v[[1 ;; i]], {i, 1, Length[v]}];
        table  ou exlusif de bits          longueur
    FromDigits[w, 2]
    depuis chiffres
   )];
(*input a sequence of gray degree like
 (x coordinate,1-x coordinate,y coordinate,1-y coordinate*)
grayDecode[bits_, len_] := Module[{xbits, ybits, x, y}, (
                           module
    (*1~len is the coordinate of x,
    len+1~2*len is the inverse coordinate of x, the same for y*)
    xbits = Sign[bits[[1 ;; len]] - bits[[len + 1 ;; 2 * len]]] /. {-1 → 0};
            signe
    ybits =
     Sign[bits[[2 * len + 1 ;; 3 * len]] - bits[[3 * len + 1 ;; 4 * len]]] /. {-1 → 0};
     signe
    x = Table[BitXor @@ xbits[[1 ;; i]], {i, 1, Length[xbits]}];
        table  ou exlusif de bits          longueur
    y = Table[BitXor @@ ybits[[1 ;; i]], {i, 1, Length[ybits]}];
        table  ou exlusif de bits          longueur
    {FromDigits[x, 2], FromDigits[y, 2], 0}
     depuis chiffres      depuis chiffres
   )];
```

Test binary code and gray code

In[ ]:=
```
codes = Table[binarycode[x, y, 4], {y, 0, 15}, {x, 0, 15}];
codes = Flatten[codes, {{3}, {1}, {2}}];
img = Image /@ codes
test = Join @@ codes[[1 ;; 4, 1 ;; 1]];
Print["Binary code:"]
Image[test]
```

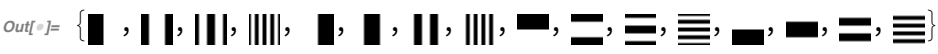
Out[ ]=

```
Binary code:
```

Out[ ]=

In[ ]:=
```
codes = Table[graycode[x, y, 4], {y, 0, 15}, {x, 0, 15}];
codes = Flatten[codes, {{3}, {1}, {2}}];
img = Image /@ codes
test = Join @@ codes[[1 ;; 4, 1 ;; 1]];
Print["Gray code:"]
Image[test]
Table[{i, graycode[i], grayDecode[graycode[i]]}, {i, 0, 10}] // MatrixForm
```

Out[ ]=



```
Gray code:
```

Out[ ]=

Out[ ]//MatrixForm=

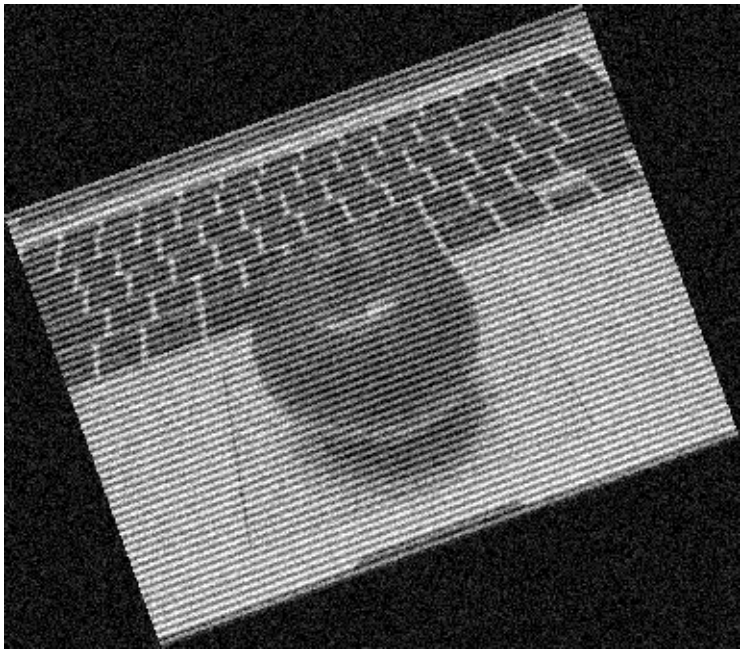
0	0	0
1	1	1
2	3	2
3	2	3
4	6	4
5	7	5
6	5	6
7	4	7
8	12	8
9	13	9
10	15	10

2. in the presence of some noise, compare real gray code with non gray code.

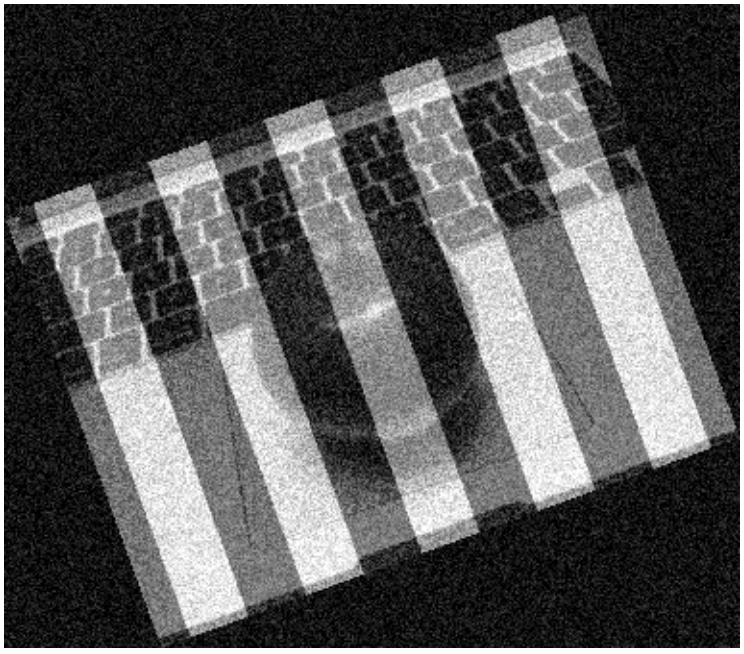
code image and capture image

```
In[ ]:= bcodes = Table[binarycode[x, y, nAddr], {y, 0, h - 1}, {x, 0, w - 1}];
        (*shape of data array is transpose with shape of image*)
        Dimensions[bcodes] (* {240,320,36}, 240 is row for array*)
        bcodes = Flatten[bcodes, {{3}, {1}, {2}}];
        bimgs = Image /@ bcodes;
        gcodes = Table[graycode[x, y, nAddr], {y, 0, h - 1}, {x, 0, w - 1}];
        gcodes = Flatten[gcodes, {{3}, {1}, {2}}];
        gimgs = Image /@ gcodes;
        (*ListAnimate[bimgs]
         ListAnimate[gimgs]*)
        camBCodes0 = camera[#, noisyValue, rotDegree] & /@ bimgs;
        camGCodes0 = camera[#, noisyValue, rotDegree] & /@ gimgs;
        imdata =
          Flatten[{#, ImageData[camImg]}, {{2}, {3}, {1}}] & /@ (ImageData /@ camBCodes0);
        camBCodes = Image[Map[Mean, #, {2}]] & /@ imdata;
        imdata =
          Flatten[{#, ImageData[camImg]}, {{2}, {3}, {1}}] & /@ (ImageData /@ camGCodes0);
        camGCodes = Image[Map[Mean, #, {2}]] & /@ imdata;
        Print["Codes Example:"]
        camBCodes[[35]]
        camGCodes[[5]]

Out[ ]= {240, 320, 36}

        Codes Example:
```

Out[ ]=



Out[ ]=



Decode processing

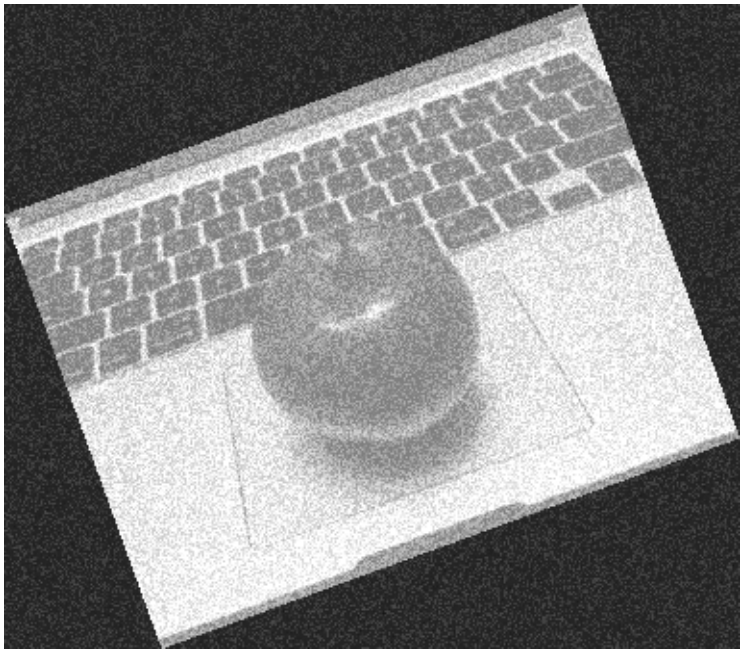
In[•]:=

```
camBCodesDecode = ImageData[ColorConvert[#, "GrayScale"]] & /@ camBCodes;
                  données d'··· convertis couleur
camBCodesDecode = Flatten[camBCodesDecode, {{2}, {3}, {1}}];
                  aplatis
Dimensions[camBCodesDecode]
dimensions
Map[Min, camBCodesDecode, {2}] // Image
app· minimum                   image
Map[Max, camBCodesDecode, {2}] // Image
app· maximum                   image
Map[Mean, camBCodesDecode, {2}] // Image
app· valeur moyenne             image
(Map[Max, camBCodesDecode, {2}] - Map[Min, camBCodesDecode, {2}]) // Image
 app· maximum                    app· minimum                      image
```

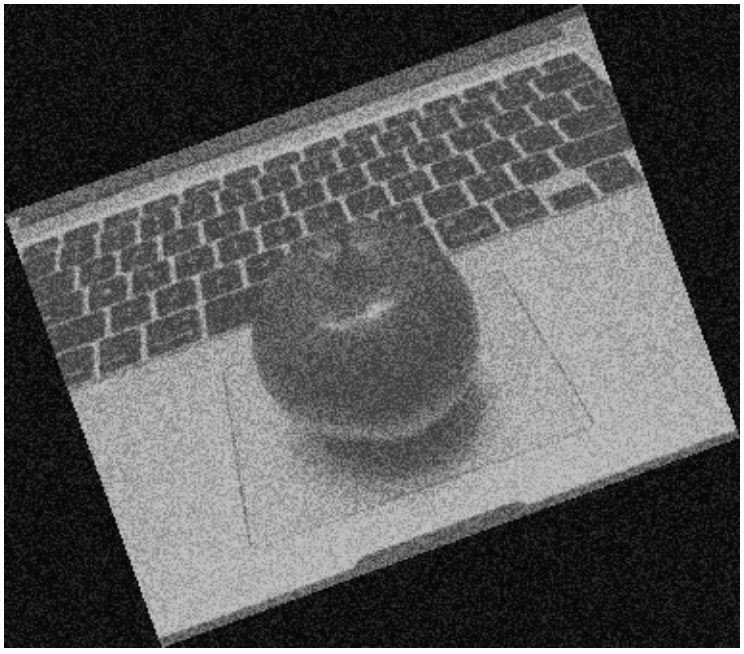
Out[•]= {335, 383, 36}

Out[•]=

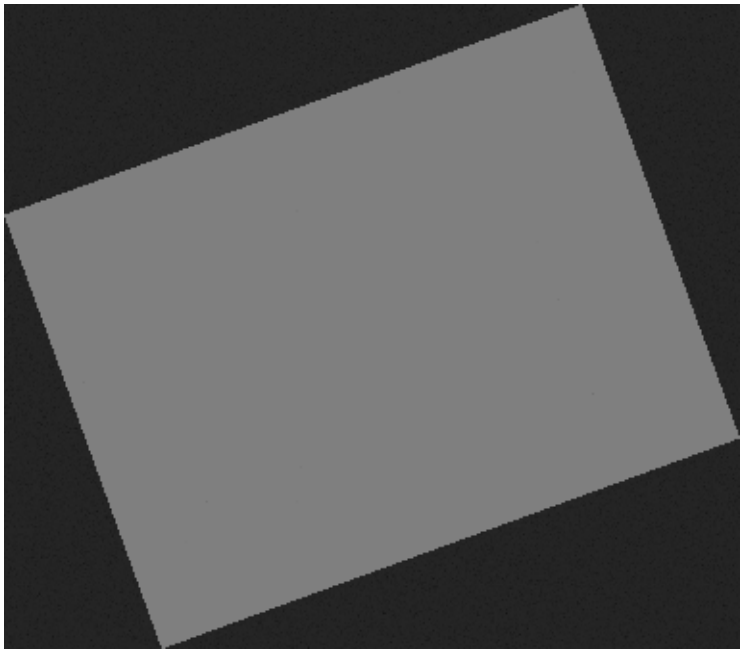
Out[ ]=



Out[ ]=



Out[•]=



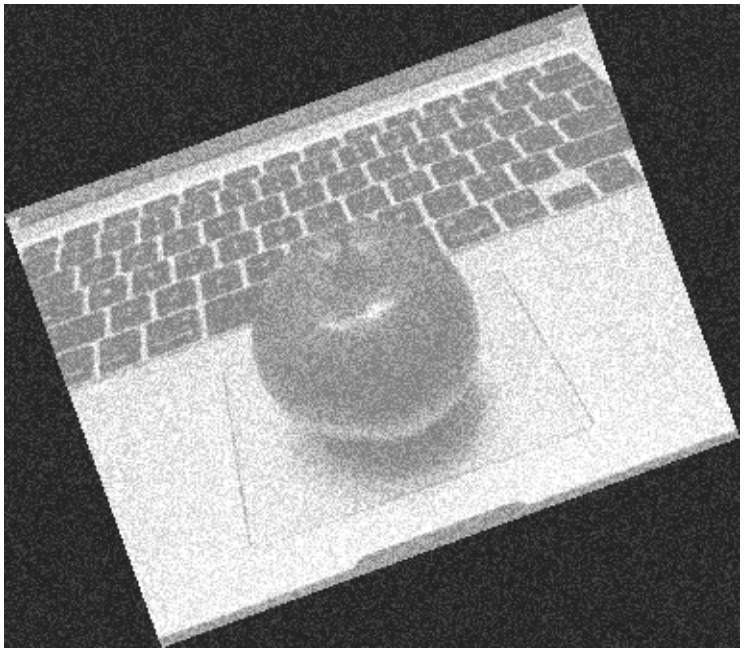
In[•]:=

```
camGCodesDecode = ImageData[ColorConvert[#, "GrayScale"]] & /@ camGCodes;
                  données d'··· convertis couleur

camGCodesDecode = Flatten[camGCodesDecode, {{2}, {3}, {1}}];
                  aplatis

Dimensions[camGCodesDecode]
dimensions

Map[Min, camGCodesDecode, {2}] // Image
app· minimum                   image

Map[Max, camGCodesDecode, {2}] // Image
app· maximum                   image

Map[Mean, camGCodesDecode, {2}] // Image
app· valeur moyenne             image

(Map[Max, camGCodesDecode, {2}] - Map[Min, camGCodesDecode, {2}]) // Image
 app· maximum                    app· minimum                     image
```

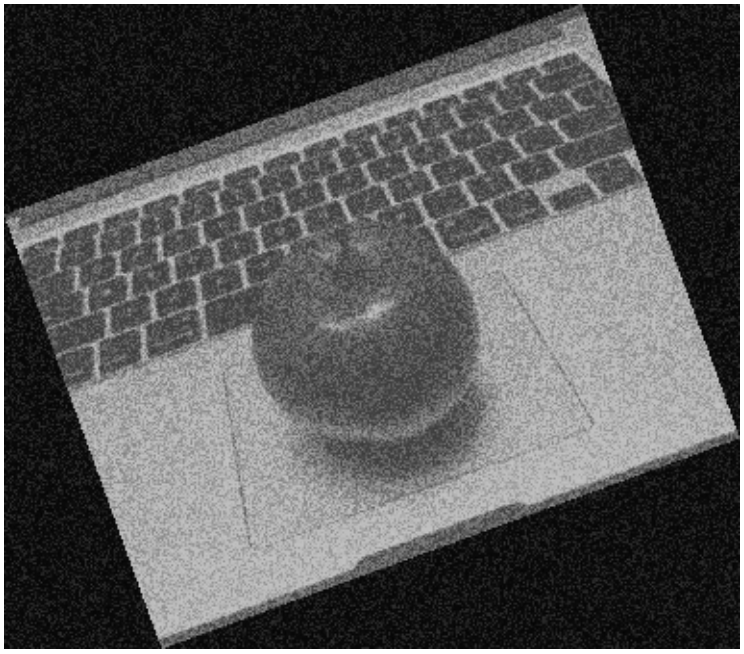
Out[•]= {335, 383, 36}

Out[ ]=

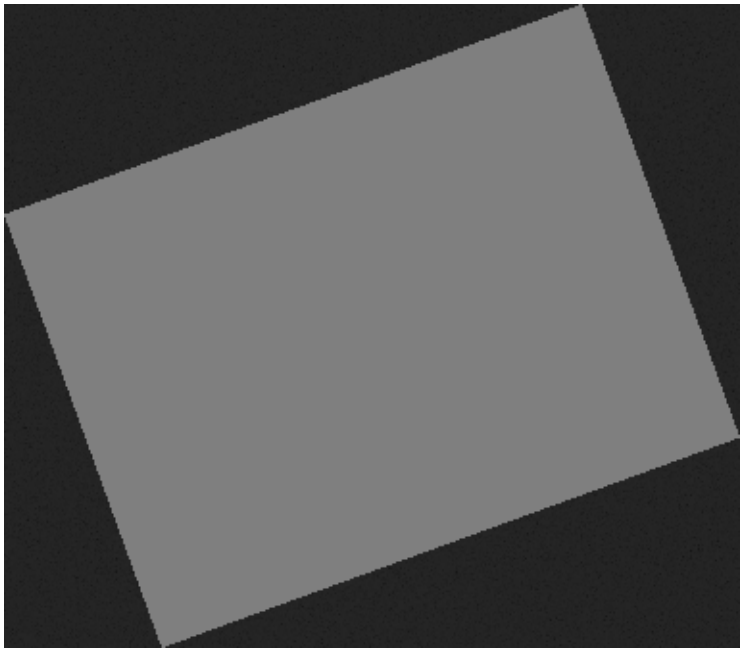
Out[ ]=



Out[ ]=



Out[ ]=



Now obtain the coordinates after projection

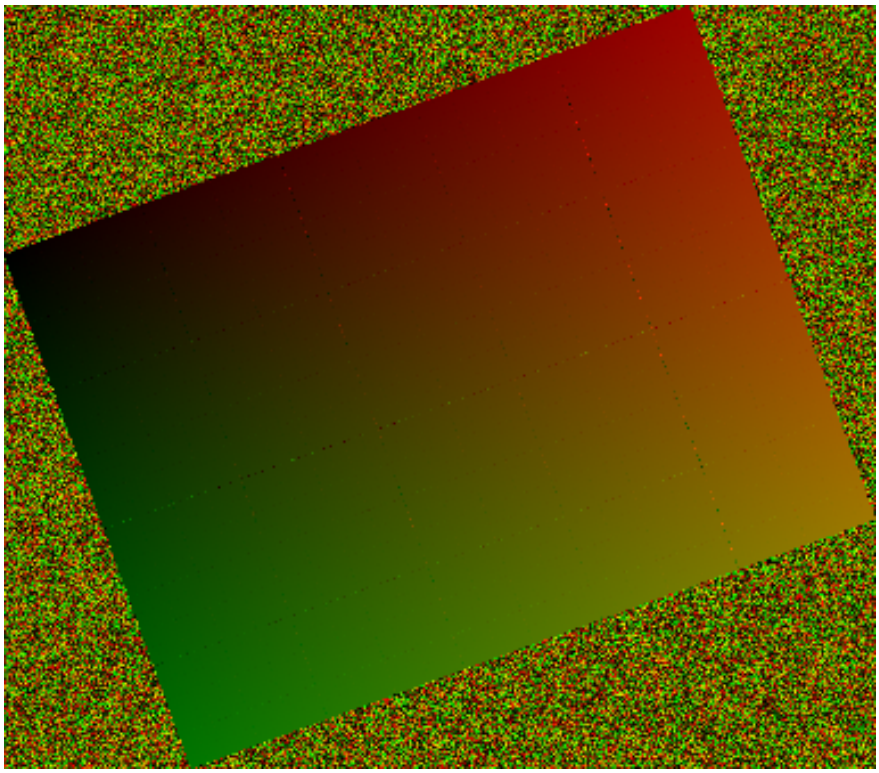
In[ ]:=

```
bCoordDecode = Map[binDecode[#, nAddr] &, camBCodesDecode, {2}];
               applique
Dimensions[bCoordDecode]
dimensions
bCoordDecode[[150 ;; 154, 190]]
Image[bCoordDecode] // ImageAdjust
image                          ajuste image

gCoordDecode = Map[grayDecode[#, nAddr] &, camGCodesDecode, {2}];
               applique
gCoordDecode[[150 ;; 154, 190]]
Dimensions[gCoordDecode]
dimensions
Image[gCoordDecode] // ImageAdjust
image                          ajuste image
```

Out[ ]= {335, 383, 3}

Out[ ]= {{164, 102, 0}, {161, 103, 0}, {163, 104, 0}, {163, 104, 0}, {162, 106, 0}}

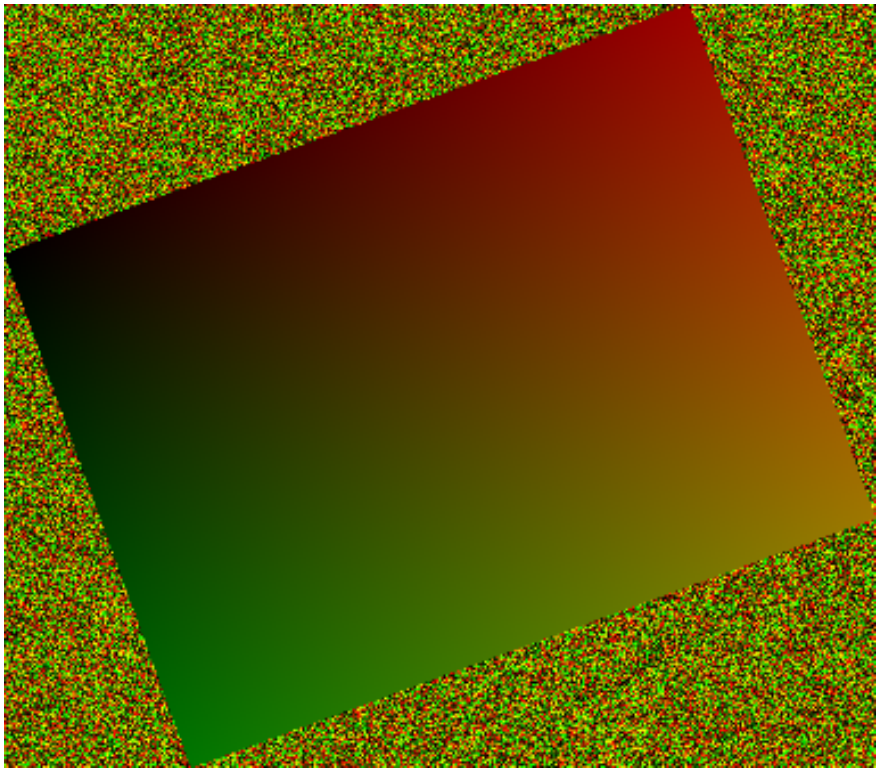
Out[ ]=



Out[ ]= {{164, 102, 0}, {163, 103, 0}, {163, 104, 0}, {163, 105, 0}, {162, 105, 0}}

Out[ ]= {335, 383, 3}

Out[ ]=



Compare the two codes on a rotation with some noise. We see that some failed lines appear after the binary code is decoded. These are usually the edges of two codes, because the number of two adjacent code change many bits, resulting in instability, such as 0111 to 1000 points. But we see that there is no failed line after the gray code is decoded. This is the ability of the gray code. It only transforms one bit in any two adjacent code, which is more robust. But the disadvantage of gray code is that the calculation speed is much slower than the binary code.

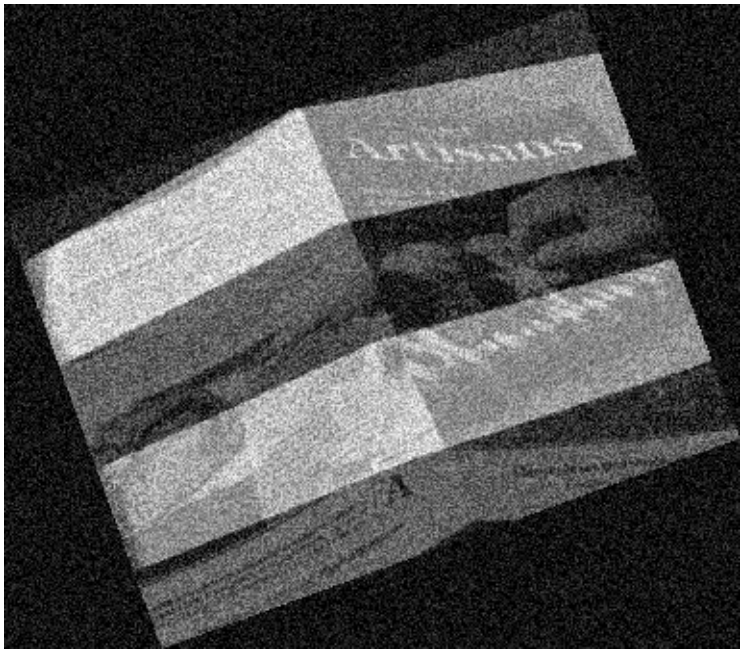
Test a fake distortion image:
Obtain the fake camera images: noisy images + distorted noisy code images.

In[•]:=

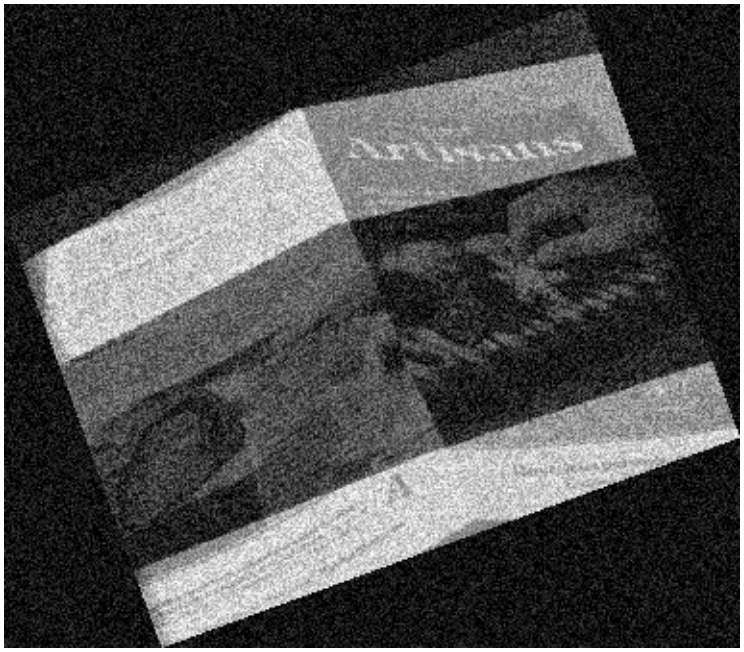
```
camBCodesDistorted0 = distortCamera[#, noisyValue, rotDegree] & /@ bimgs;
camGCodesDistorted0 = distortCamera[#, noisyValue, rotDegree] & /@ gimgs;

imdata = Flatten[{#, ImageData[disCamImg]}, {{2}, {3}, {1}}] & /@
          aplatis          données d'image
    (ImageData /@ camBCodesDistorted0);
     données d'image
camBCodesDistorted = Image[Map[Mean, #, {2}]] & /@ imdata;
                     image  app·  valeur moyenne
imdata = Flatten[{#, ImageData[disCamImg]}, {{2}, {3}, {1}}] & /@
          aplatis          données d'image
    (ImageData /@ camGCodesDistorted0);
     données d'image
camGCodesDistorted = Image[Map[Mean, #, {2}]] & /@ imdata;
                     image  app·  valeur moyenne
Print[
imprime
 "Simulated object+bar code projection effect using distortion function:"]
camBCodesDistorted[[30]]
camGCodesDistorted[[30]]
Print["The origianl graycode image of distortion:"]
imprime
ImageTransformation[gimgs[[30]], f@@# &]
transformée d'image
Print["The origianl image that like have same distortion:"]
imprime
img2
```

```
Simulated object+bar code projection effect using distortion function:
```

Out[ ]=

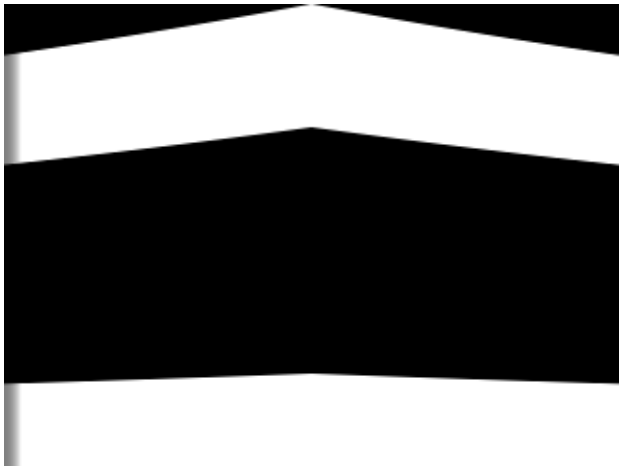


Out[ ]=



The origianl graycode image of distortion:

Out[ ]=



The origianl image that like have same distortion:

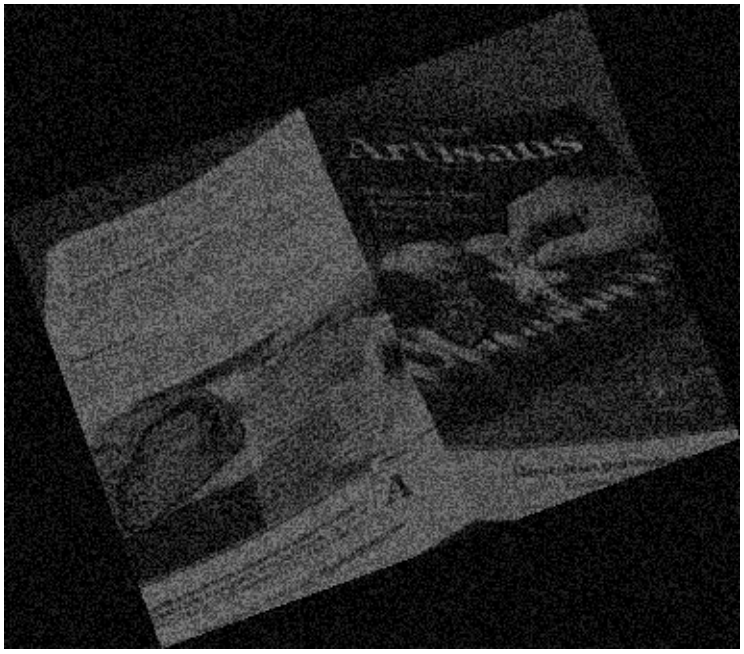
Out[ ]=

In[ ]:=

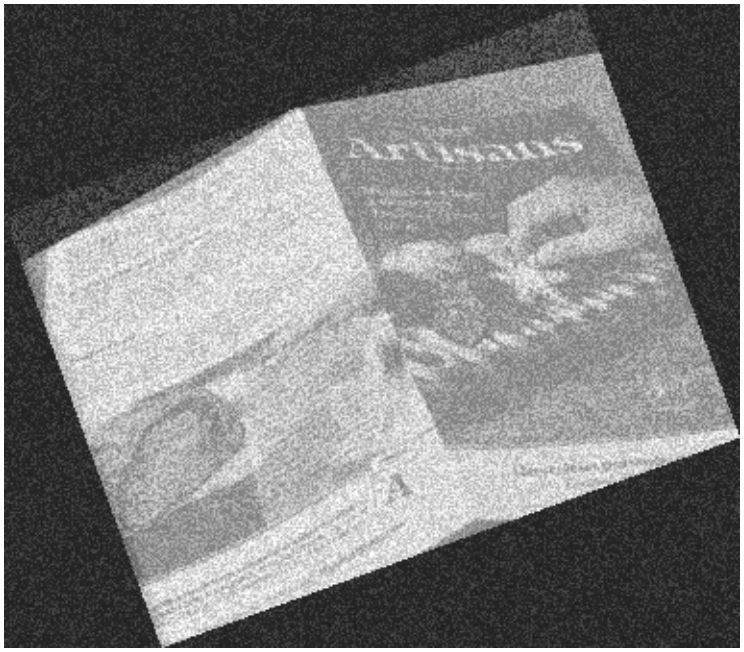
```
camBCodesDistDecode =
  ImageData[ColorConvert[#, "GrayScale"]] & /@ camBCodesDistorted;
camBCodesDistDecode = Flatten[camBCodesDistDecode, {{2}, {3}, {1}}];
Dimensions[camBCodesDistDecode]
Map[Min, camBCodesDistDecode, {2}] // Image
Map[Max, camBCodesDistDecode, {2}] // Image
Map[Mean, camBCodesDistDecode, {2}] // Image
maskBcode =
  (Map[Max, camBCodesDistDecode, {2}] - Map[Min, camBCodesDistDecode, {2}]);
maskBcode // Image
maskBcode = Map[If[# < 0.15, 0, 1] &, maskBcode, {2}];
Image[maskBcode]

camGCodesDistDecode =
  ImageData[ColorConvert[#, "GrayScale"]] & /@ camGCodesDistorted;
camGCodesDistDecode = Flatten[camGCodesDistDecode, {{2}, {3}, {1}}];
Dimensions[camGCodesDistDecode]
Map[Min, camGCodesDistDecode, {2}] // Image
Map[Max, camGCodesDistDecode, {2}] // Image
Map[Mean, camGCodesDistDecode, {2}] // Image
maskGcode =
  (Map[Max, camGCodesDistDecode, {2}] - Map[Min, camGCodesDistDecode, {2}]);
maskGcode // Image
maskGcode = Map[If[# < 0.15, 0, 1] &, maskGcode, {2}];
Image[maskGcode]
```

Out[ ]= {335, 383, 36}

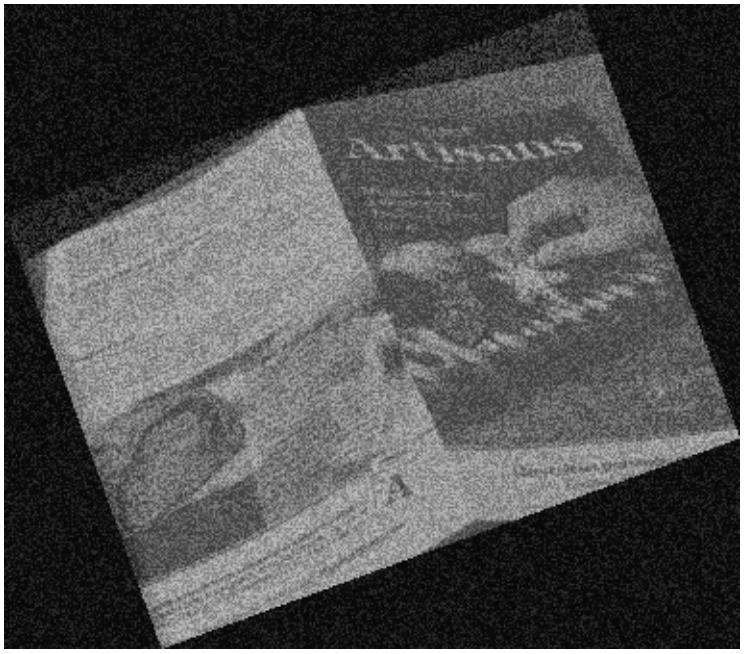
Out[ ]=



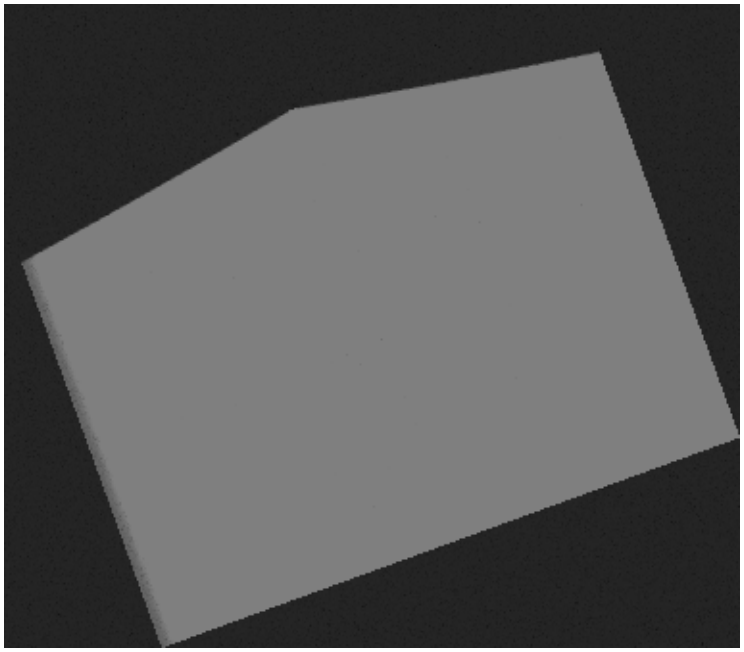
Out[ ]=



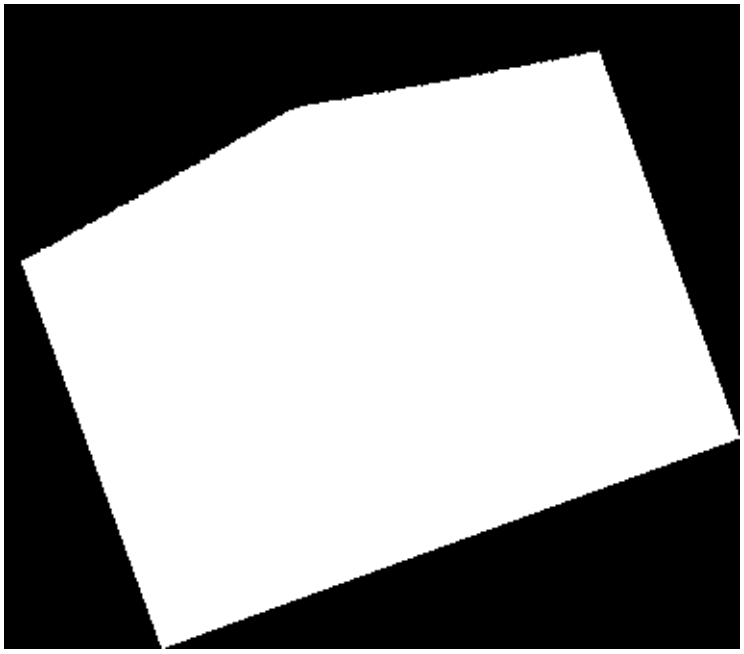
Out[ ]=



Out[ ]=

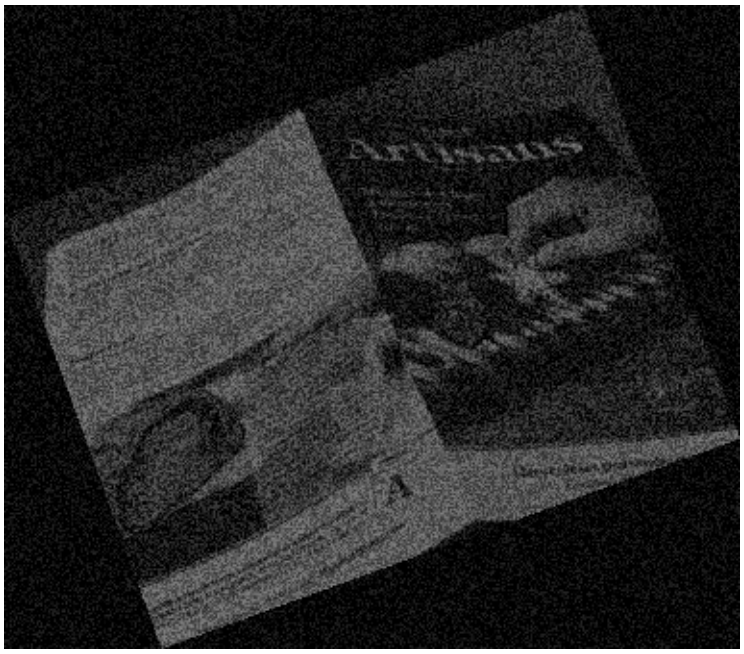


Out[•]=

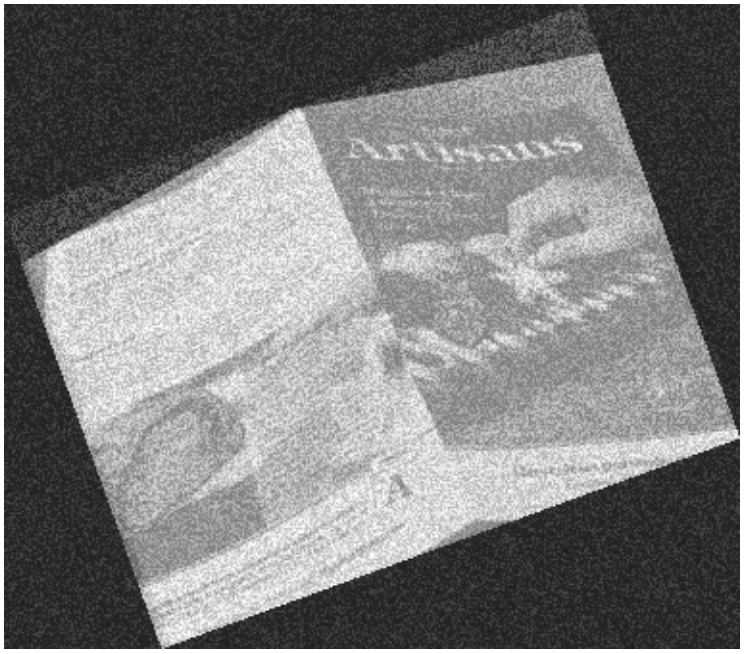


Out[•]= {335, 383, 36}

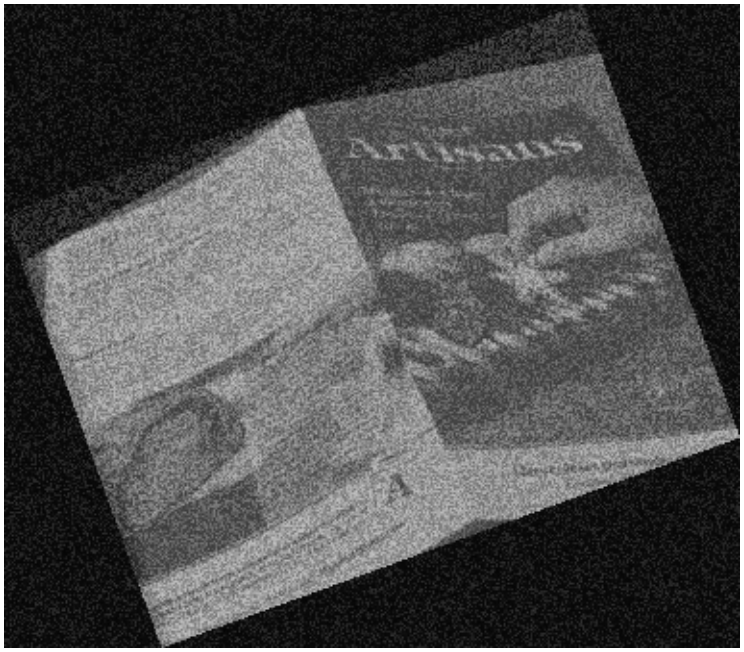
Out[•]=



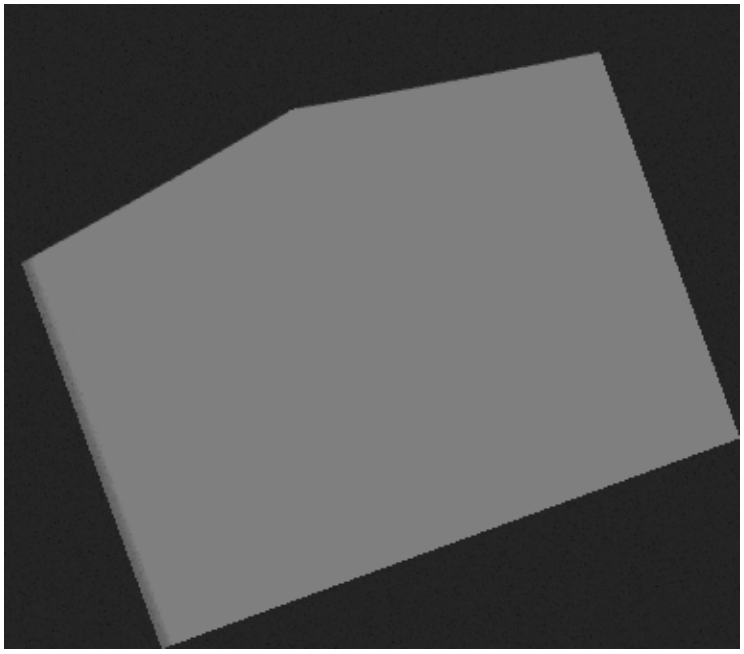
Out[ ]=



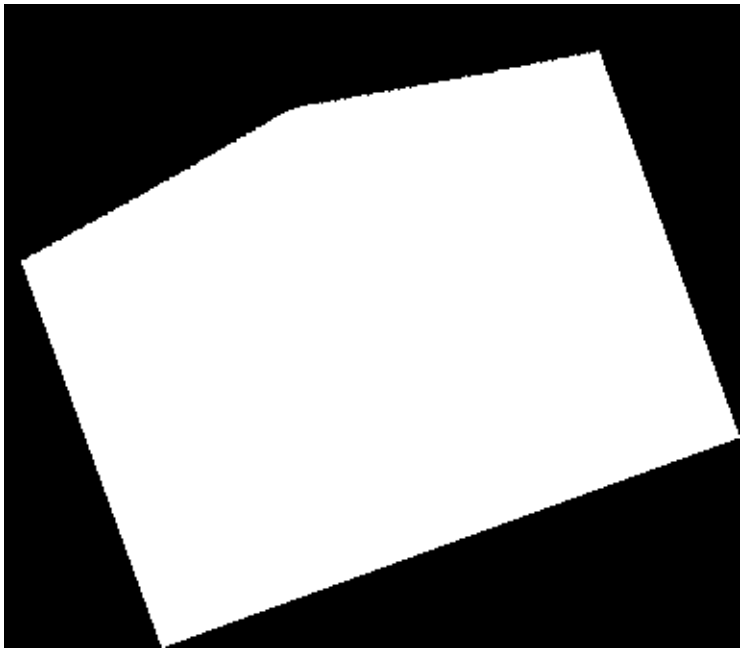
Out[ ]=



Out[ ]=



Out[ ]=



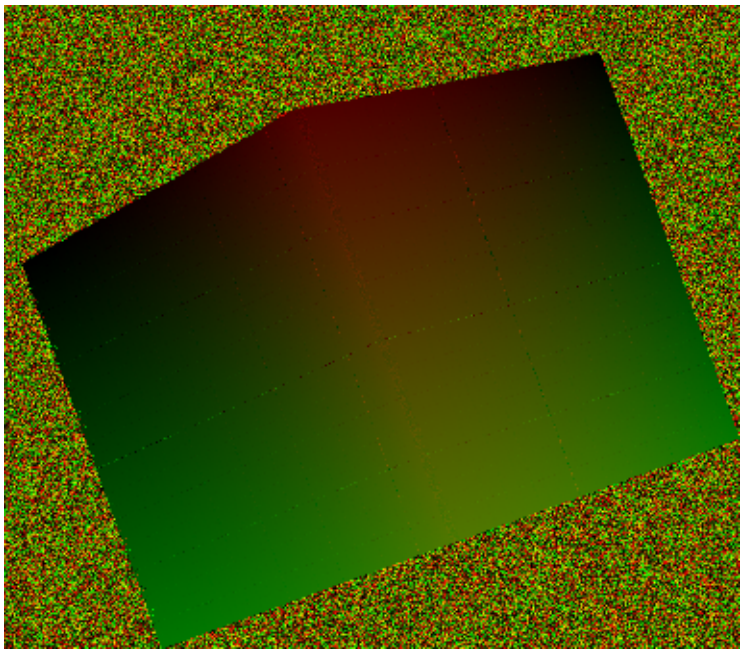
In[ ]:=

```
bDistCoordDecode = Map[binDecode[#, nAddr] &, camBCodesDistDecode, {2}];
                   applique
Dimensions[bDistCoordDecode]
dimensions
bDistCoordDecode[[150 ;; 154, 190]]
Image[bDistCoordDecode] // ImageAdjust
image                   ajuste image

gDistCoordDecode = Map[grayDecode[#, nAddr] &, camGCodesDistDecode, {2}];
                   applique
gDistCoordDecode[[150 ;; 154, 190]]
Dimensions[gDistCoordDecode]
dimensions
Image[gDistCoordDecode] // ImageAdjust
image                   ajuste image
```

Out[ ]= {335, 383, 3}

Out[ ]= {{153, 103, 0}, {151, 102, 0}, {157, 103, 0}, {155, 105, 0}, {151, 105, 0}}

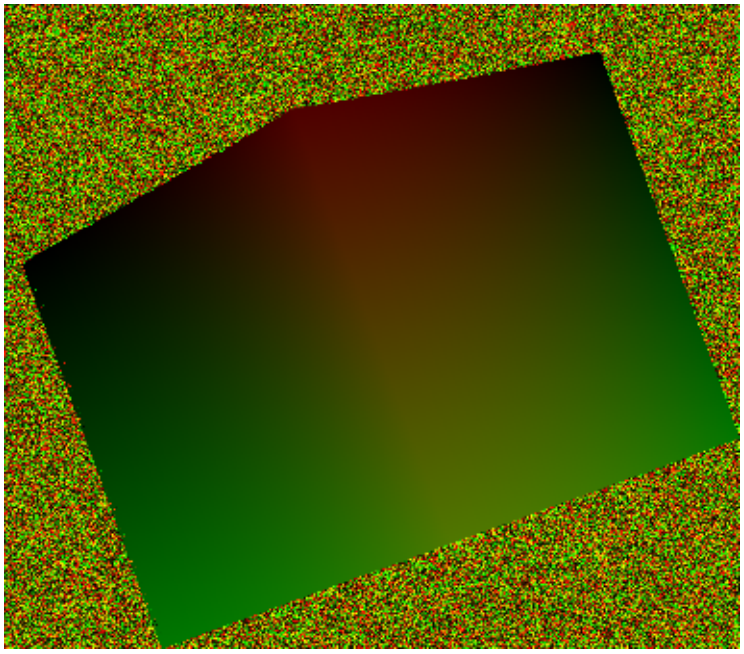
Out[ ]=



Out[ ]= {{160, 101, 0}, {159, 102, 0}, {159, 103, 0}, {159, 104, 0}, {159, 105, 0}}

Out[ ]= {335, 383, 3}

Out[ ]=



Compare the two codes on a distored image with rotation and some noise. We see that same fact as the case without distortion above.

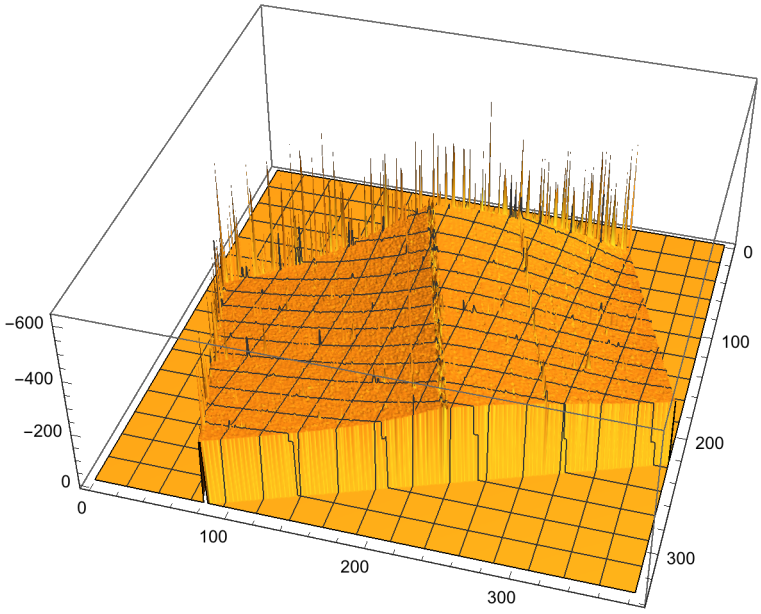
Reconstruct height (an approximate height)

In[ ]:=
```
Print["Reconstruct using Binary code:"]
L = 800; (*mm*)
B = 600; (*mm*)
(*subcoord=bDistCoordDecode-bCoordDecode;
subcoord[[150,191]]*)
allD = Map[Norm, bDistCoordDecode, {2}] // N;
allD[[150, 191]]
(*heights=allD*L/B//N;*)
ListPlot3D[-allD * maskBcode]
Print["Reconstruct using Gray code:"]
(*subcoord=gDistCoordDecode-gCoordDecode;
subcoord[[150,191]]*)
allD = Map[Norm, gDistCoordDecode, {2}] // N;
(*heights=allD*L/B//N;*)
plot3dGcode = ListPlot3D[-allD * maskGcode]
ImageTransformation[img2, f@@# &]
```

```
Reconstruct using Binary code:
```

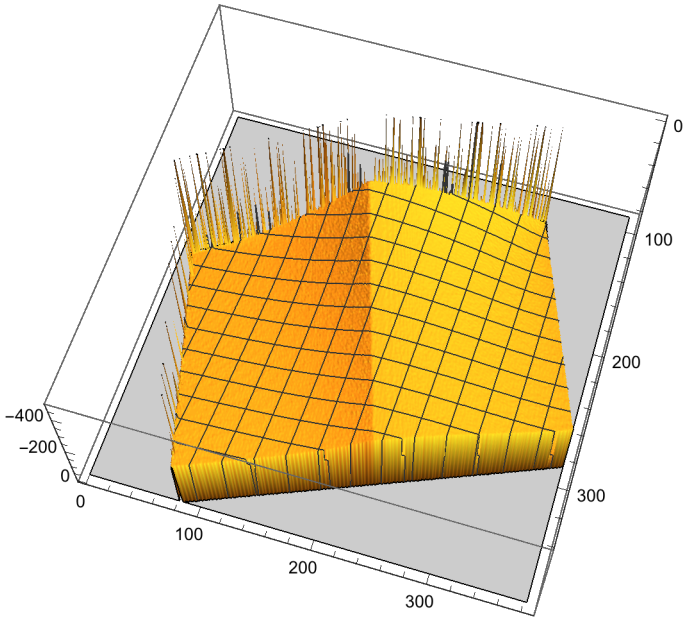
Out[ ]=
```
188.367
```

Out[•]=

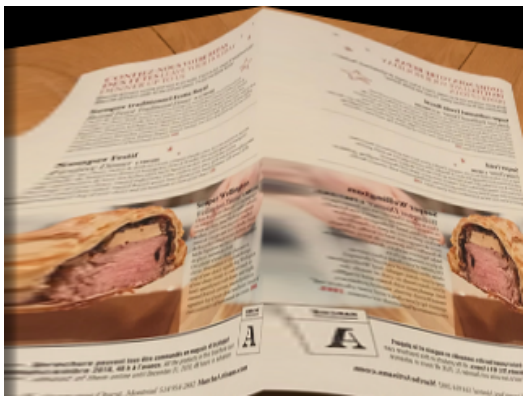


Reconstruct using Gray code:

Out[•]=



Out[•]=



The reconstruction using Gray code is better than Binary code.

3. code phase shift (sin wave stuff)
use more shift than 4, say 10, and compare 4 with 10.

In[ ]:=

```
atan[0, 0] := 0;
atan[0., 0.] := 0;
atan[x_, y_] := ArcTan[x, y];
                arc tangente
sincode[x_, y_, cycle_, resolution_, upperN_] := Module[{a, b, s, PHI}, (
                                                 module
    a = 0.5;
    b = 0.5;
    PHI = (x / resolution) * 2 Pi * cycle;
                               nombre pi
    Table[Sin[PHI + 2 Pi * s / upperN] * b + a, {s, 0, upperN - 1}]
    table  sinus          nombre pi
   )];
sinDecodeN4[{i1_, i2_, i3_, i4_}, cycle_, resolution_] :=
  (resolution / cycle) * (0.5 + atan[i1 - i3, i4 - i2] / (2 Pi));
                                                        nombre pi
sinAbsoluteDecodeN4[{i1_, i2_, i3_, i4_, k_}, cycle_, resolution_] :=
  Module[{PHI, phi}, (
  module
    phi = atan[i4 - i2, i3 - i1];
    PHI = phi + 2 * Pi * k;
                 nombre pi
    (0.5 + PHI / (2 Pi)) * (resolution / cycle)
                 nombre pi
   )];

(*k is in i[[-1]], uppeN is the number of phase par cycle,
adjust is the calibration of the phase for difference upperN*)
(*N=3, adjust=-0.3, N=4, adjust=0, N=5, adjust=0.3,
  valeur numérique      valeur numérique    valeur numérique
N=6, adjust=0.5, N=7, adjust=0.5, N=8, adjust=0.8,
valeur numérique      valeur numérique    valeur numérique
N=9, adjust=1, N=10, adjust=1.25, N=20, adjust=3.4*)
valeur numérique   valeur numérique      valeur numérique
sinAbsoluteDecode[i_, cycle_, resolution_, upperN_, adjust_] :=
```

```
  Module[{s, PHI, phi}, (
  |module
    phi = -atan[Sum[i[[s]] * Cos[2 * Pi * (s + adjust) / upperN], {s, 1, upperN}],
                |somme          |cosinus |nombre pi
        Sum[i[[s]] * Sin[2 * Pi * (s + adjust) / upperN], {s, 1, upperN}]];
       |somme          |sinus    |nombre pi
    PHI = phi + 2 * Pi * i[[-1]];
                  |nombre pi
    (0.5 + PHI / (2 Pi)) * (resolution / cycle)
                  |nombre pi
   )];

unwrap[p_, cycle_, resolution_] :=
  (*p[[1]] is y*)
  IntegerPart[p[[1]] * cycle / resolution];
  |partie entière
```

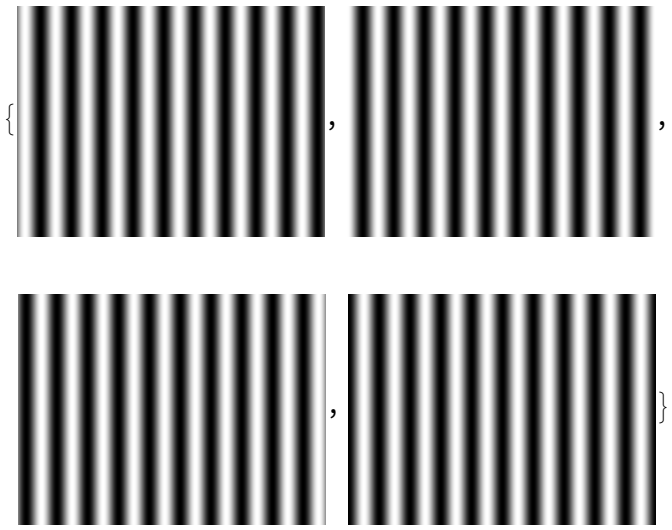
For N=4

In[ ]:=

```
Print["sinus N=4 :"]
|imprime      |valeur numérique
upperN = 4;
cycle = 10;
scodesN4 = Table[sincode[x, y, cycle, w, upperN], {y, 0, h - 1}, {x, 0, w - 1}];
           |table
scodesN4 = Flatten[scodesN4, {{3}, {1}, {2}}];
           |aplatis
(*ListLinePlot[scodesN4[[All,1,All]]]*)
  |tracé de liste de ligne      |tout  |tout
simgsN4 = Image /@ scodesN4
          |image
```

```
sinus N=4 :
```

Out[ ]=



Decode processing without unwrap function

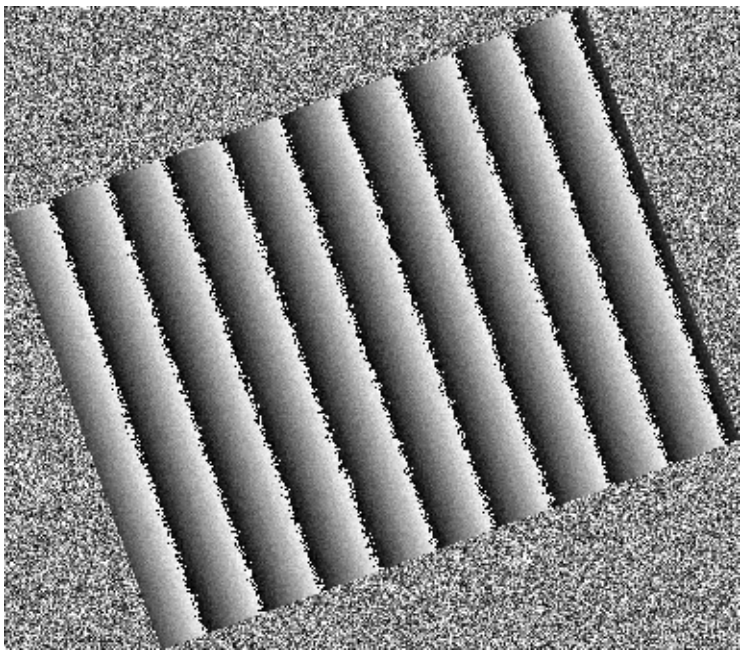
In[ ]:=

```
camSCodes0 = camera[#, noisyValue, rotDegree] & /@ simgsN4;
sDecode0 = ImageData[ColorConvert[#, "GrayScale"]] & /@ camSCodes0;
           données d'… convertis couleur
sDecode0 = Flatten[sDecode0, {{2}, {3}, {1}}];
           aplatis
Dimensions[sDecode0]
dimensions
Map[Min, sDecode0, {2}] // Image;
app… minimum              image
Map[Max, sDecode0, {2}] // Image;
app… maximum              image
Map[Mean, sDecode0, {2}] // Image;
app… valeur moyenne       image
(Map[Max, sDecode0, {2}] - Map[Min, sDecode0, {2}]) // Image;
 app… maximum              app… minimum                image
phases0 = Map[sinDecodeN4[#, cycle, w] &, sDecode0, {2}];
          applique
Dimensions[phases0]
dimensions
Image[phases0] // ImageAdjust
image                 ajuste image
```

Out[ ]= {335, 383, 4}

Out[ ]= {335, 383}

Out[ ]=



Decode processing with unwrap function

In[•]:=
```
Dimensions[sDecode0]
```
dimensions
```
gCoordDecode[[150, 250]]
Print["No. of cycle for (150,250):"]
```
imprime
```
unwrap[gCoordDecode[[150, 250]], cycle, w]
allK = Map[unwrap[#, cycle, w] &, gCoordDecode, {2}];
```
applique
```
Dimensions[allK]
```
dimensions
```
Print["phase cycle for (150,250) without wrap:"]
```
imprime
```
sinDecodeN4[sDecode0[[150, 250]], cycle, w]
Print["phase cycle for (150,250) with wrap:"]
```
imprime
```
Print["Test decode, DecodeN4 = general decode function?"]
```
imprime
```
sinAbsoluteDecodeN4[Join[sDecode0[[150, 250]], {allK[[150, 250]]}], cycle, w]
```
joins
```
sinAbsoluteDecode[Join[sDecode0[[150, 250]], {allK[[150, 250]]}], cycle, w, 4]
```
joins

Out[•]= {335, 383, 4}

Out[•]= {220, 122, 0}

No. of cycle for (150,250):

Out[•]= 6

Out[•]= {335, 383}

phase cycle for (150,250) without wrap:

Out[•]= 4.46141

phase cycle for (150,250) with wrap:

Test decode, DecodeN4 = general decode function?

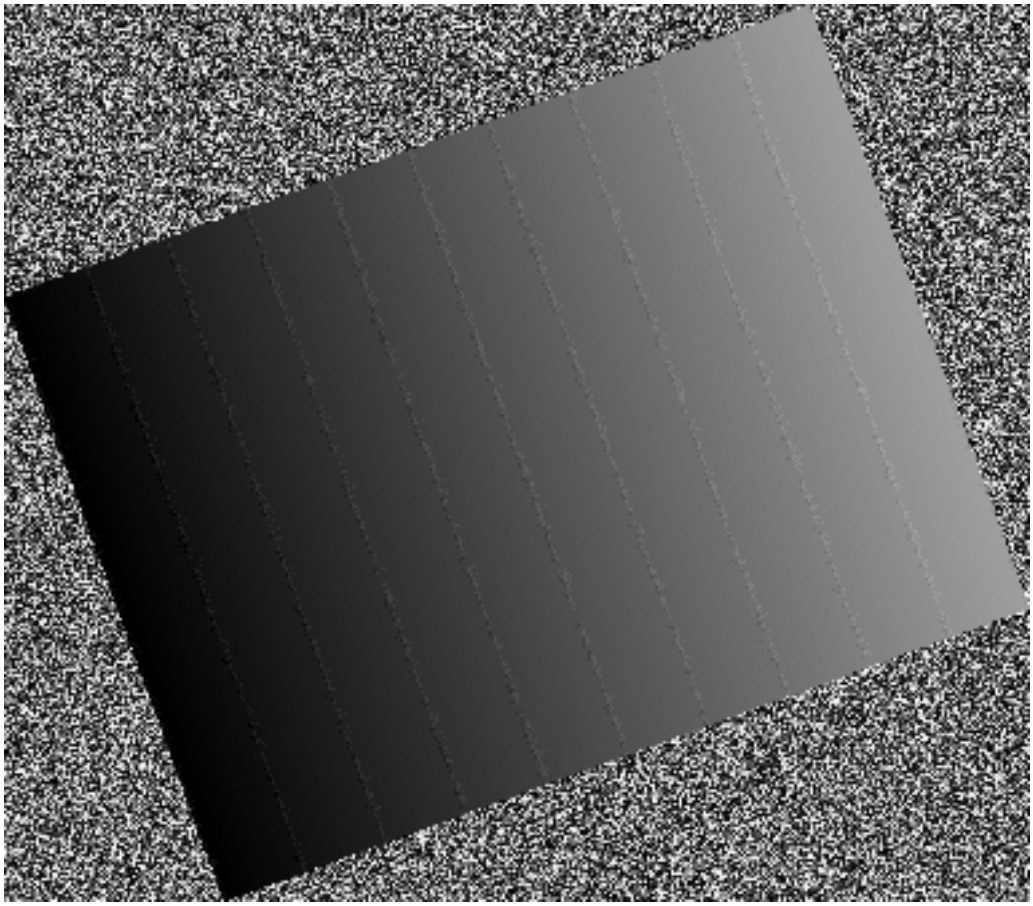
Out[•]= 220.461

Out[•]= sinAbsoluteDecode[{0.455455, 0.570171, 0.838233, 0.110898, 6}, 10, 320, 4]

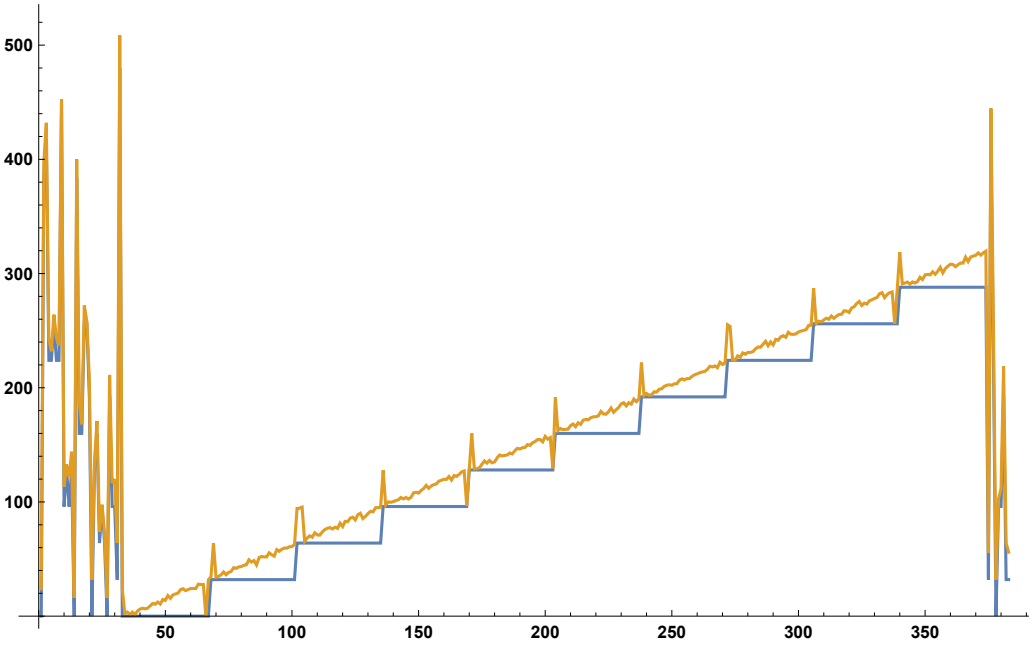
```
In[•]:= sDecodeWithK0 = Table[Join[sDecode0[[i, j]], {allK[[i, j]]}],
                        table   joins
    {i, 1, Dimensions[allK][[1]]}, {j, 1, Dimensions[allK][[2]]}];
           dimensions                     dimensions
  Dimensions[sDecodeWithK0]
  dimensions
  sDecodeWithK0[[150, 250]]
  phasesN4Unwrap0 = Map[sinAbsoluteDecodeN4[#, cycle, w] &, sDecodeWithK0, {2}];
                    applique
  (*phasesN4Unwrap0=Map[sinAbsoluteDecode[#,cycle,w,4]&,sDecodeWithK0,{2}];*)
                    applique
  Dimensions[phasesN4Unwrap0]
  dimensions
  Dimensions[phasesN4Unwrap0[[200]]]
  dimensions
  Image[phasesN4Unwrap0] // ImageAdjust
  image                  ajuste image
  plotN4 = ListPlot[
           tracé de liste
    {sDecodeWithK0[[200, ;; , 5]] * w / cycle, phasesN4Unwrap0[[200]]}, Joined → True]
                                                                  joint    vrai
```

```
Out[•]= {335, 383, 5}

Out[•]= {0.455455, 0.570171, 0.838233, 0.110898, 6}

Out[•]= {335, 383}

Out[•]= {383}
```

Out[ ]=



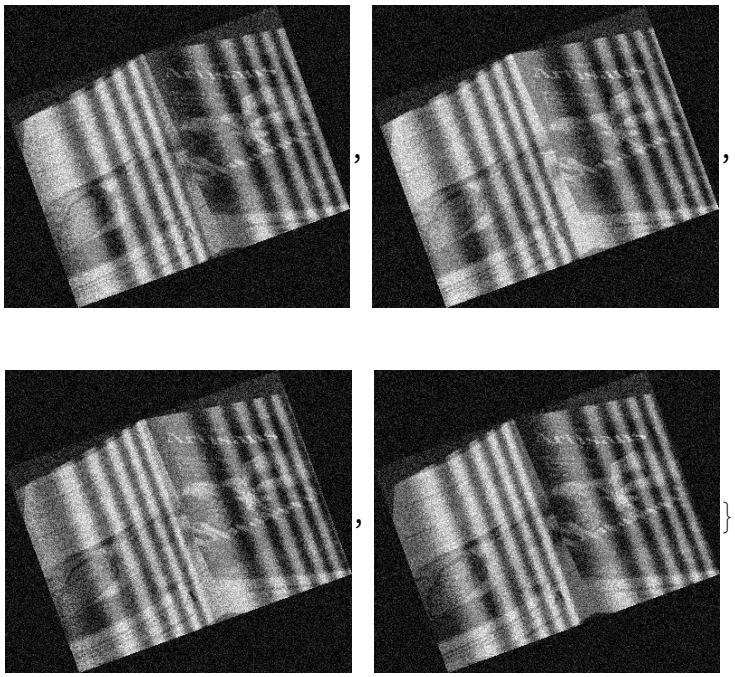
Out[ ]=



For the distortion, Obtain the fake camera images for the sinus image.

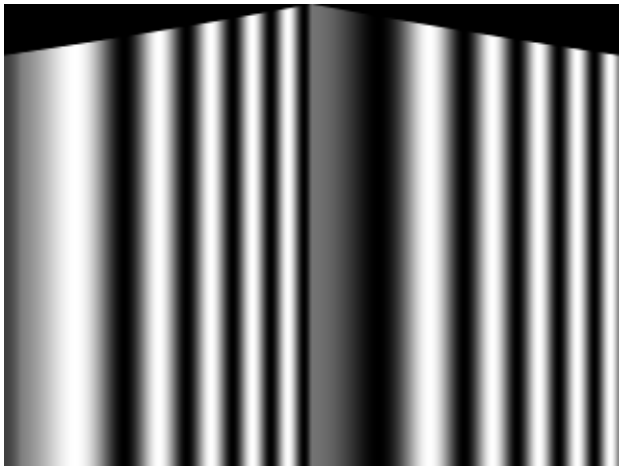
In[ ]:=
```
camSCodesN4Distorted0 = distortCamera[#, noisyValue, rotDegree] & /@ simgsN4;

imdata = Flatten[{#, ImageData[disCamImg]}, {{2}, {3}, {1}}] & /@
          aplatis          données d'image
    (ImageData /@ camSCodesN4Distorted0);
     données d'image
camSCodesN4Distorted = Image[Map[Mean, #, {2}]] & /@ imdata
                       image  app·  valeur moyenne

Print["The origianl sinus image of distortion:"]
imprime
ImageTransformation[simgsN4[[1]], f@@# &]
transformée d'image
Print["The origianl image that like have same distortion:"]
imprime
img2
(*ListAnimate[camSCodesN4Distorted]*)
  anime liste
```

Out[ ]=



```
The origianl sinus image of distortion:
```

Out[•]=



The origianl image that like have same distortion:

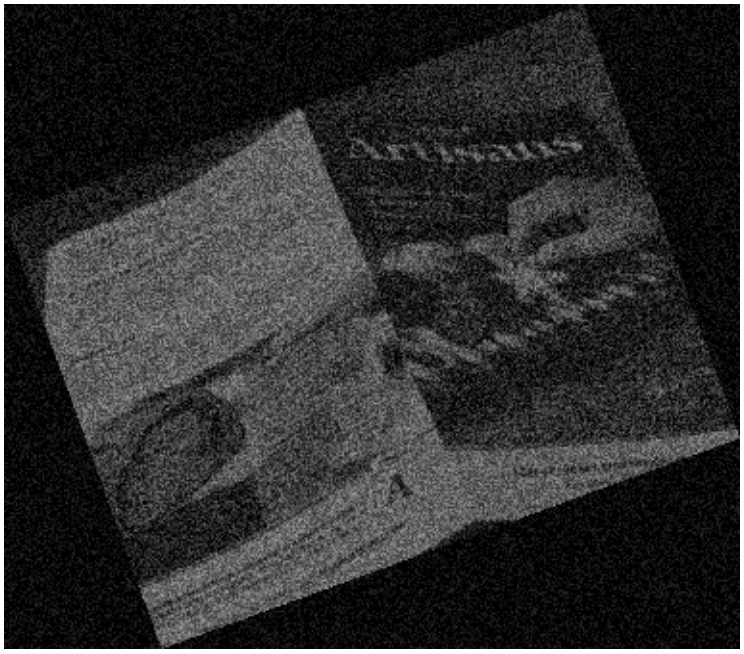
Out[•]=

Decode processing without unwrap function

In[•]:=
```
camSCodesN4DistDecode =
  ImageData[ColorConvert[#, "GrayScale"]] & /@ camSCodesN4Distorted;
camSCodesN4DistDecode = Flatten[camSCodesN4DistDecode, {{2}, {3}, {1}}];
Dimensions[camSCodesN4DistDecode]
Map[Min, camSCodesN4DistDecode, {2}] // Image
Map[Max, camSCodesN4DistDecode, {2}] // Image
Map[Mean, camSCodesN4DistDecode, {2}] // Image
maskN4 =
  (Map[Max, camSCodesN4DistDecode, {2}] - Map[Min, camSCodesN4DistDecode, {2}]);
maskN4 // Image
maskN4 = Map[If[# < 0.15, 0, 1] &, maskN4, {2}];
Image[maskN4]
```

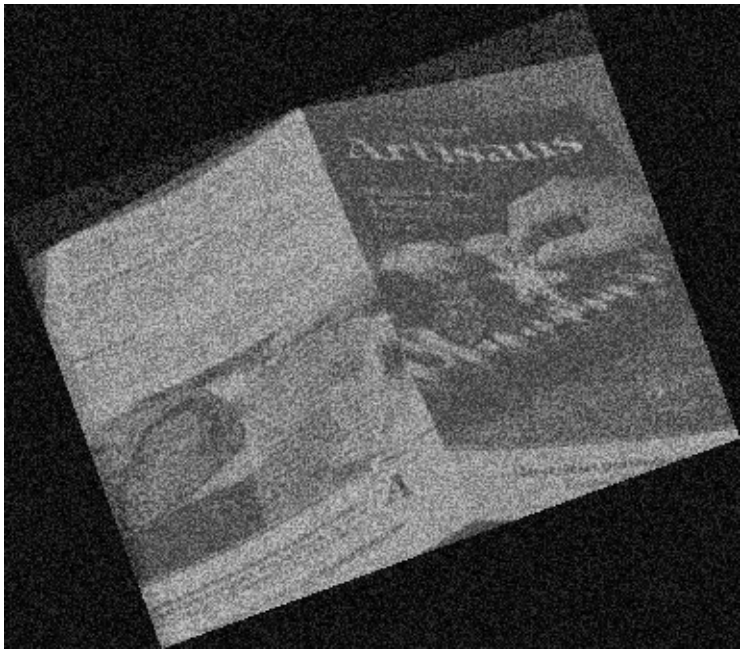
Out[•]=
```
{335, 383, 4}
```

Out[ ]=

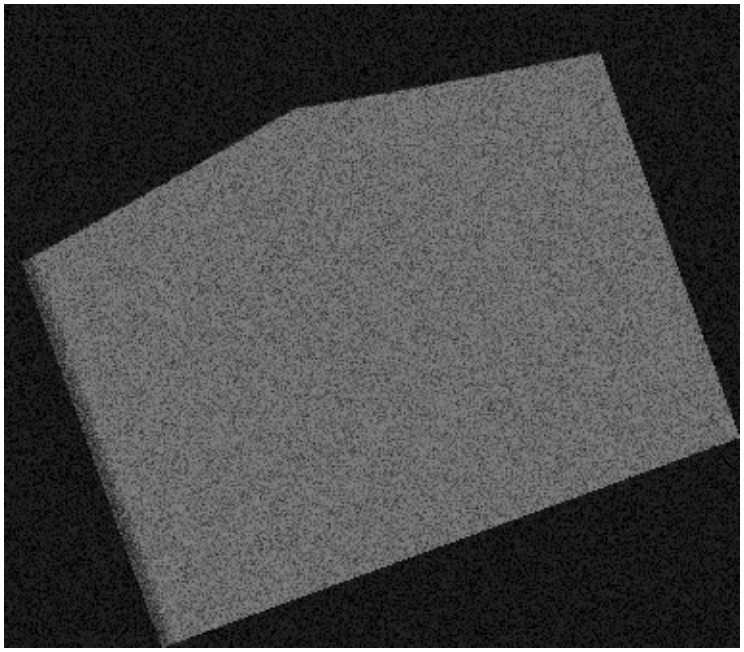


Out[ ]=

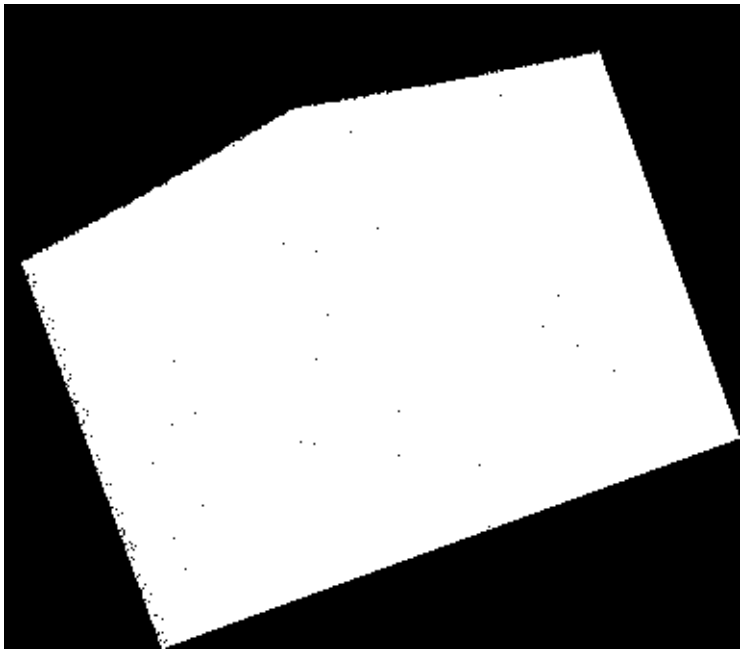
Out[ ]=



Out[ ]=



Out[ ]=



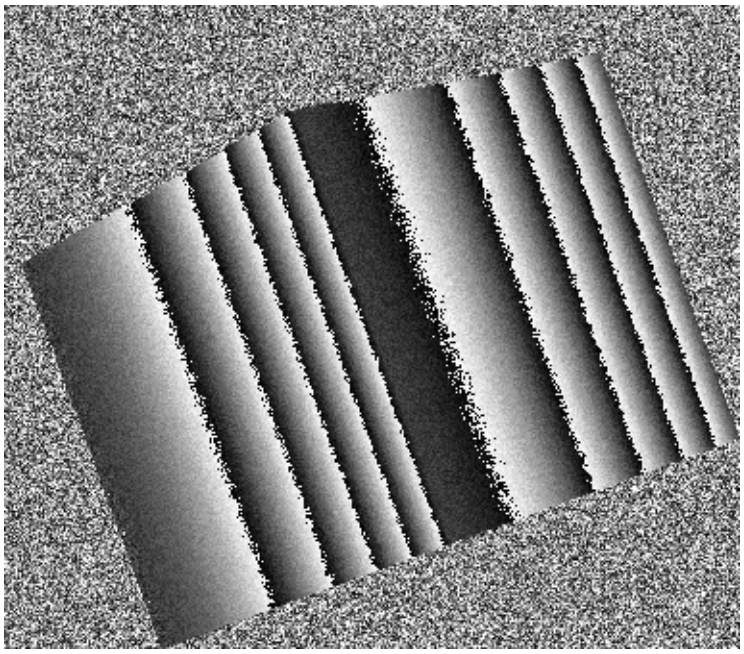
Decode processing without unwrap function (for comparaison)

In[ ]:=

```
phasesDistortN4 = Map[sinDecodeN4[#, cycle, w] &, camSCodesN4DistDecode, {2}];
                  applique
Dimensions[phasesDistortN4]
dimensions
Image[phasesDistortN4] // ImageAdjust
image                  ajuste image
```

Out[ ]= {335, 383}

Out[ ]=



Decode processing with unwrap function

In[ ]:=

```
gDistCoordDecode[[150, 250]]
Print["No. of cycle for (150,250):"]
unwrap[gDistCoordDecode[[150, 250]], cycle, w]
allDistK = Map[unwrap[#, cycle, w] &, gDistCoordDecode, {2}];
Dimensions[allDistK]
Print["phase cycle for (150,250) without unwrap:"]
sinDecodeN4[camSCodesN4DistDecode[[150, 250]], cycle, w]
Print["phase cycle for (150,250) with unwrap:"]
sinAbsoluteDecodeN4[
 Join[camSCodesN4DistDecode[[150, 250]], {allDistK[[150, 250]]}], cycle, w]
```

Out[ ]= {136, 117, 0}

```
No. of cycle for (150,250):
```

Out[ ]= 4

Out[ ]= {335, 383}

```
phase cycle for (150,250) without unwrap:
```

Out[ ]= 16.3399

```
phase cycle for (150,250) with unwrap:
```

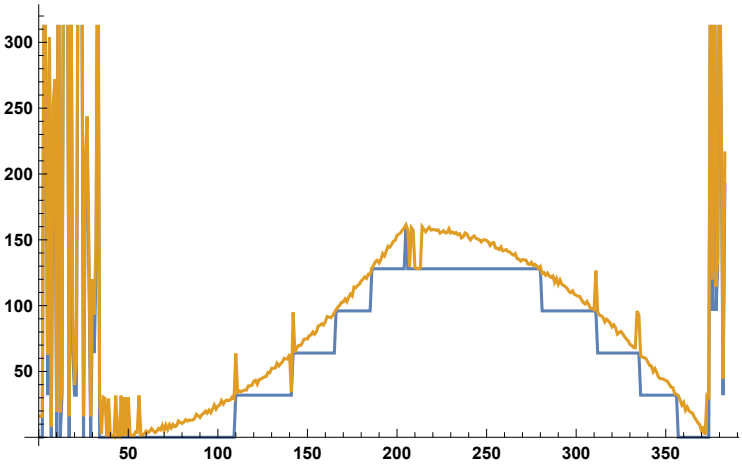
Out[ ]= 136.34

In[ ]:=

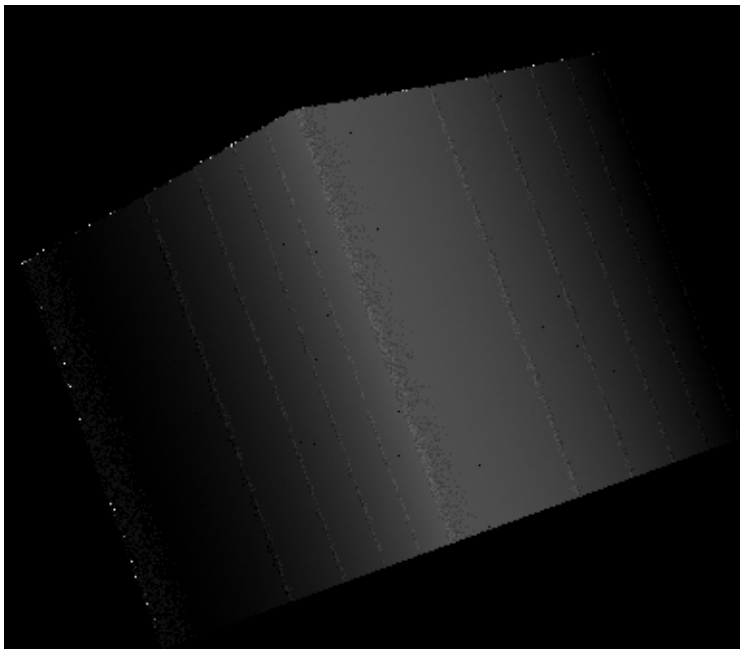
```
camSCodesN4DistDecodeWithK =
  Table[Join[camSCodesN4DistDecode[[i, j]], {allDistK[[i, j]]}],
   {i, 1, Dimensions[allDistK][[1]]}, {j, 1, Dimensions[allDistK][[2]]}];
camSCodesN4DistDecodeWithK[[150, 250]]
phasesDistortN4Unwrap =
  Map[sinAbsoluteDecodeN4[#, cycle, w] &, camSCodesN4DistDecodeWithK, {2}];
ListPlot[{camSCodesN4DistDecodeWithK[[200, ;; , upperN + 1]] * w / cycle,
  phasesDistortN4Unwrap[[200]]}, Joined → True]
Image[phasesDistortN4Unwrap * maskN4] // ImageAdjust
Print["The origianl sinus image of distortion:"]
ImageTransformation[simgsN4[[1]], f @@ # &]
```

Out[•]= {0.364547, 0.167715, 0., 0.192079, 4}

Out[•]=

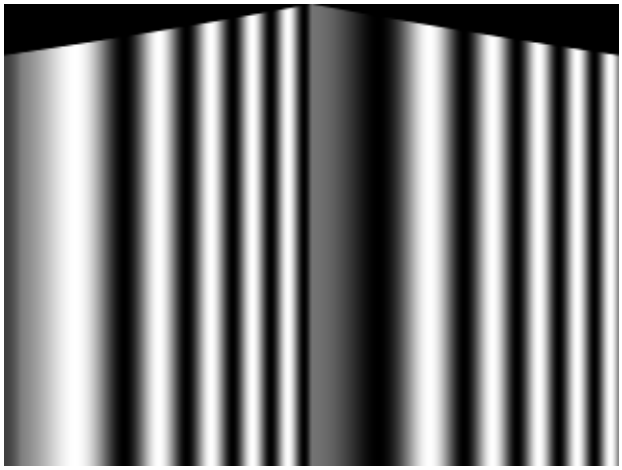


Out[•]=



The origianl sinus image of distortion:

Out[•]=



The cause of the anomaly of the image of the highest area:

The distortion function copies the image on the left to the right when it doing the distortion.

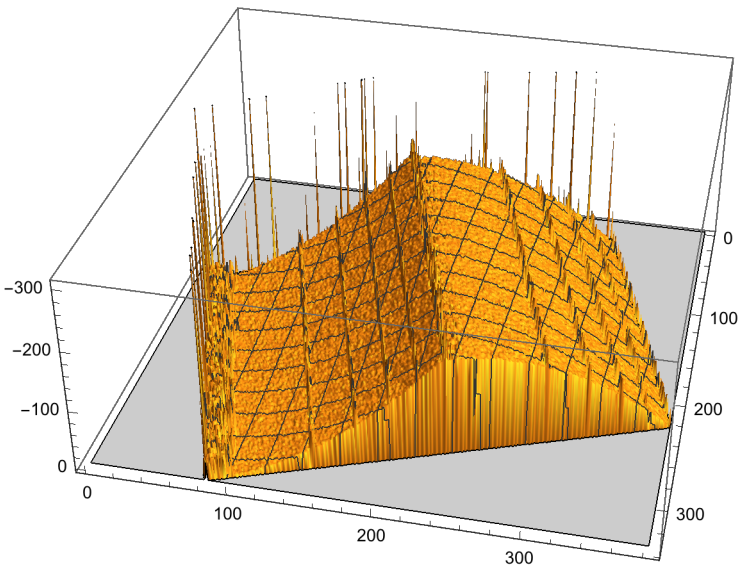
Reconstruct height (an approximate height) For N=4:

In[ ]:=
```
Print["Reconstruct using shift code:"]
L = 800;(*mm*)
B = 600;(*mm*)
(*subPhaseN4=phasesDistortN4Unwrap-phasesN4Unwrap0;
subPhaseN4[[150,191]]*)
phasesDistortN4Unwrap[[150, 191]]
(*phasesN4Unwrap0[[150,191]]
   heightsN4=subPhaseN4*L/B//N;*)
plotN4mask = ListPlot3D[-phasesDistortN4Unwrap * maskN4]
Print["comparaison with Graycode reconstruction:"]
plot3dGcode
Print["The original distortion:"]
ImageTransformation[simgsN4[[1]], f@@# &]
```

```
Reconstruct using shift code:
```

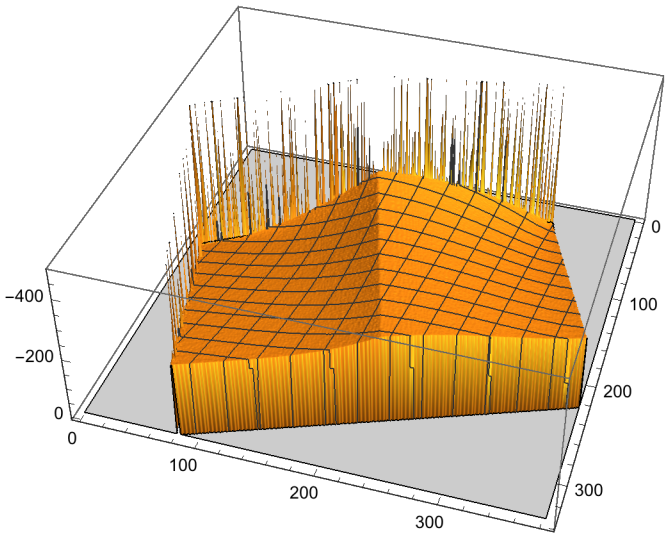
Out[ ]=
```
158.154
```

Out[ ]=



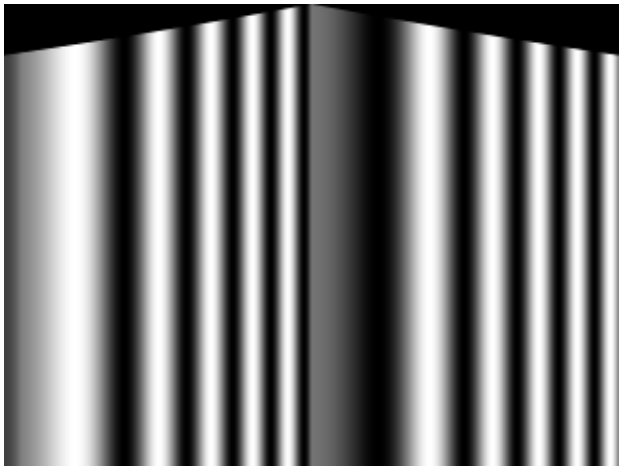

```
comparaison with Graycode reconstruction:
```

Out[ ]=



The original distortion:

Out[ ]=



For N=10

```
In[ ]:= Print["sinus N=10 :"]
        (imprime  valeur numérique)

        upperN = 10;

        adjust = 1.25

        cycle = 10;

        scodesN10 = Table[sincode[x, y, cycle, w, upperN], {y, 0, h - 1}, {x, 0, w - 1}];
                    (table)

        scodesN10 = Flatten[scodesN10, {{3}, {1}, {2}}];
                    (aplatis)

        (*ListLinePlot[scodesN4[[All,1,All]]]*)
          (tracé de liste de ligne  (tout  (tout)

        simgsN10 = Image /@ scodesN10
                   (image)

        sinus N=10 :
```

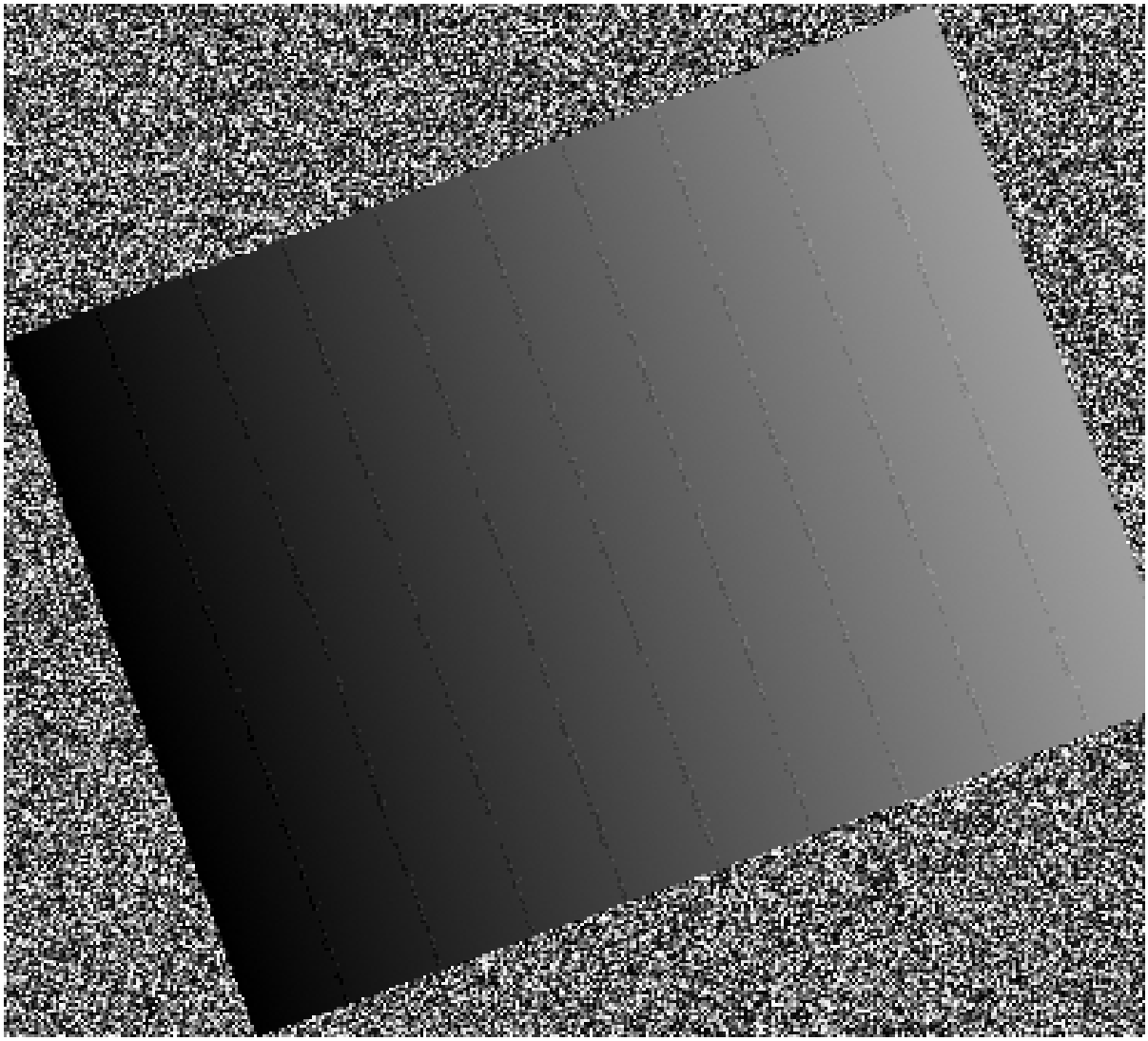
```
Out[ ]= 1.25
```

```
Out[ ]= { <image>, <image>,
          <image>, <image>,
          <image>, <image>, <image>,
          <image>, <image>, <image> }
```

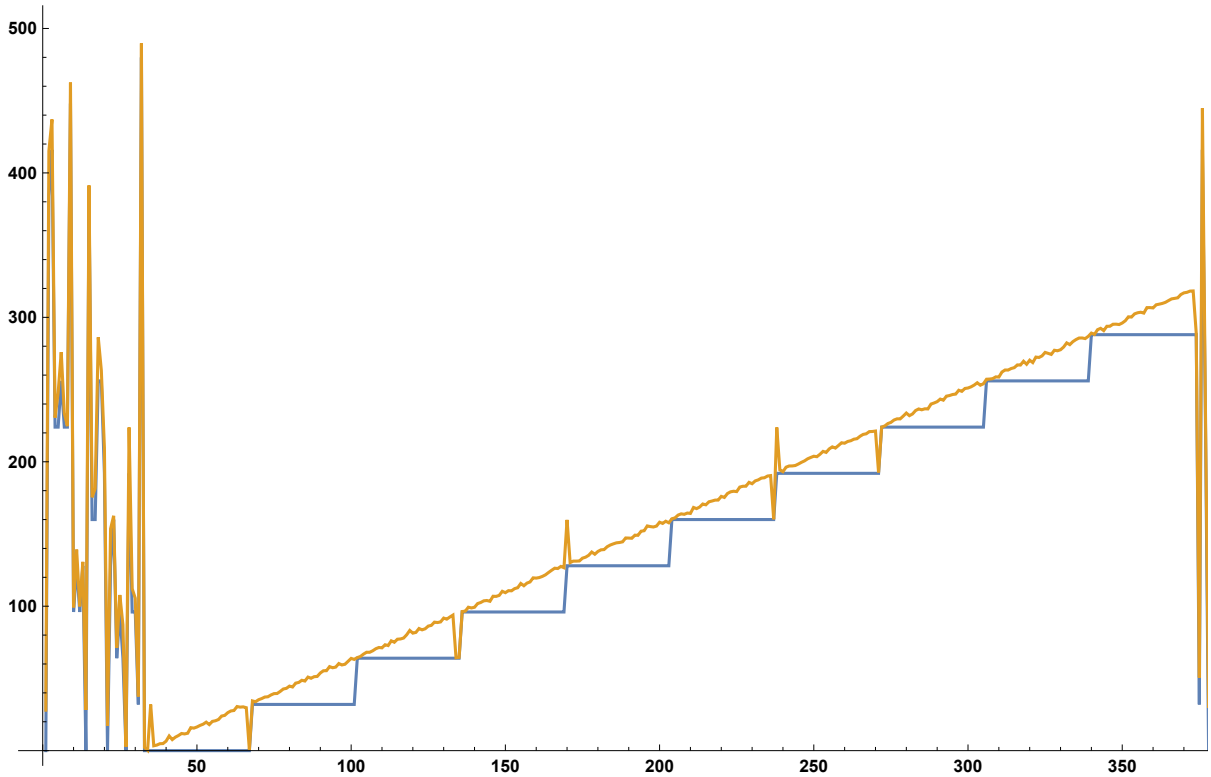
In[ ]:=

```
(*get plat image--sans object, sans distorsion, for N=10*)
                                                    valeur numérique
camSCodesN100 = camera[#, noisyValue, rotDegree] & /@ simgsN10;
camSCodesN10Decode0 =
  ImageData[ColorConvert[#, "GrayScale"]] & /@ camSCodesN100;
  données d'··· convertis couleur
camSCodesN10Decode0 = Flatten[camSCodesN10Decode0, {{2}, {3}, {1}}];
                      aplatis
camSCodesN10DecodeWithK =
  Table[Join[camSCodesN10Decode0[[i, j]], {allK[[i, j]]}],
  table  joins
   {i, 1, Dimensions[allK][[1]]}, {j, 1, Dimensions[allK][[2]]}];
         dimensions                    dimensions

phasesN10Unwrap0 = Map[sinAbsoluteDecode[#, cycle, w, upperN, adjust] &,
                    applique
   camSCodesN10DecodeWithK, {2}];
Image[phasesN10Unwrap0] // ImageAdjust
image                          ajuste image
ListPlot[{camSCodesN10DecodeWithK[[200, ;; , upperN + 1]] * w / cycle,
tracé de liste
  phasesN10Unwrap0[[200]]}, Joined → True]
                            joint    vrai
Print["Comapraison with N=4:"]
imprime                    valeur numérique
plotN4
```

Out[ ]=

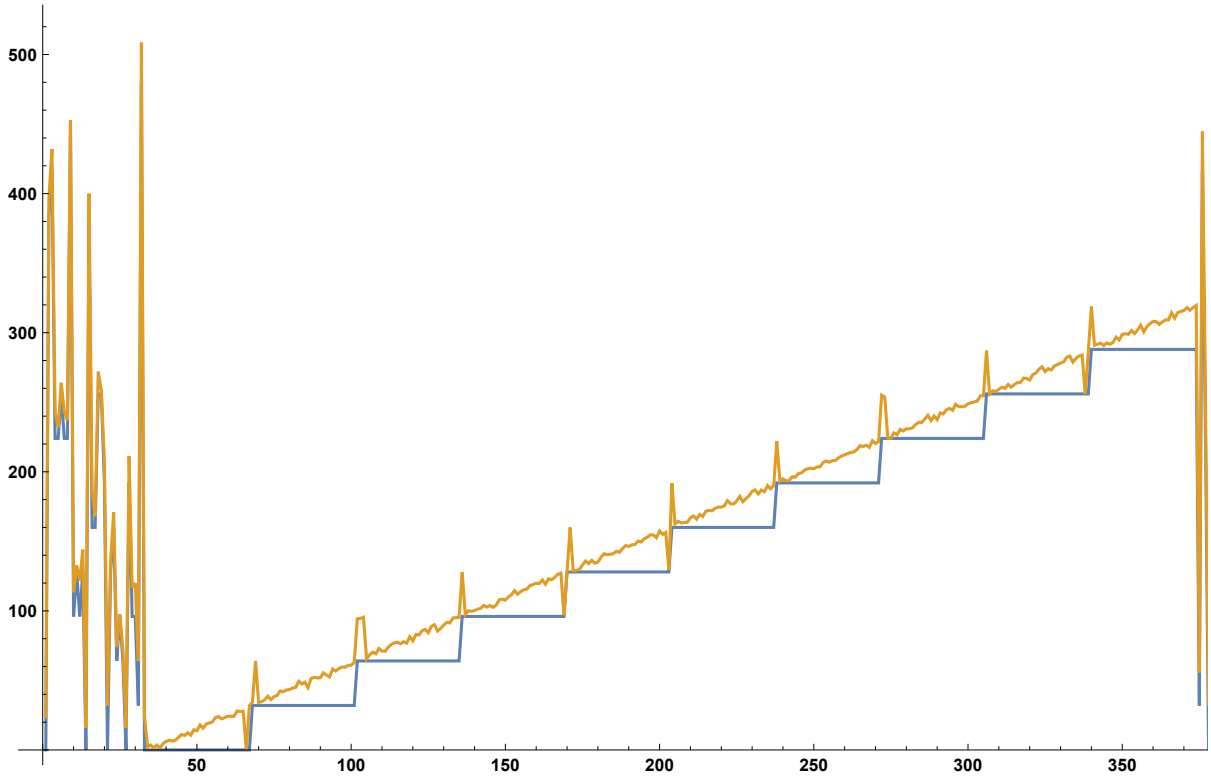


Out[•]=



Comapraison with N=4:

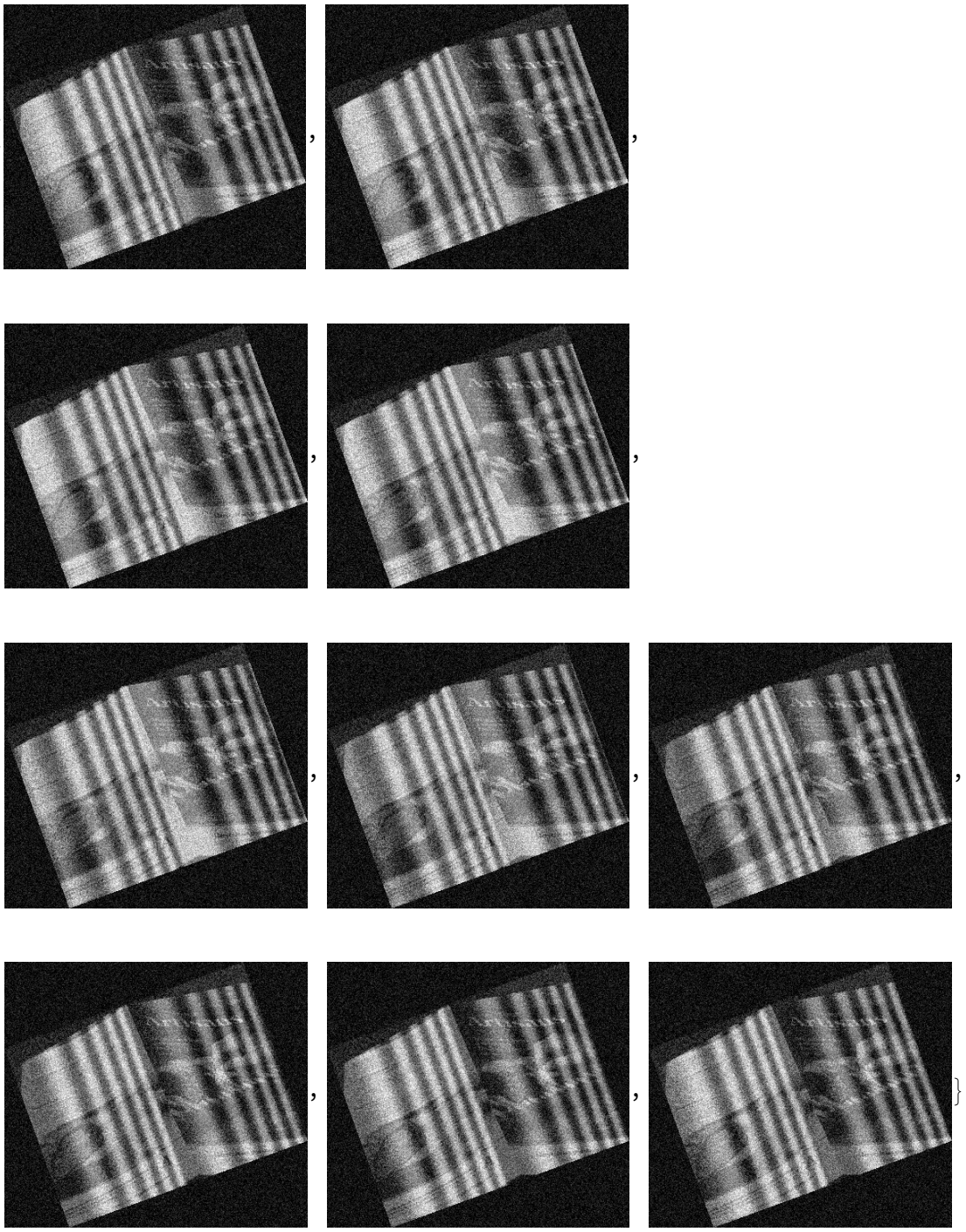
Out[•]=



We observed that the curve is smoother when N=10 than N=4.

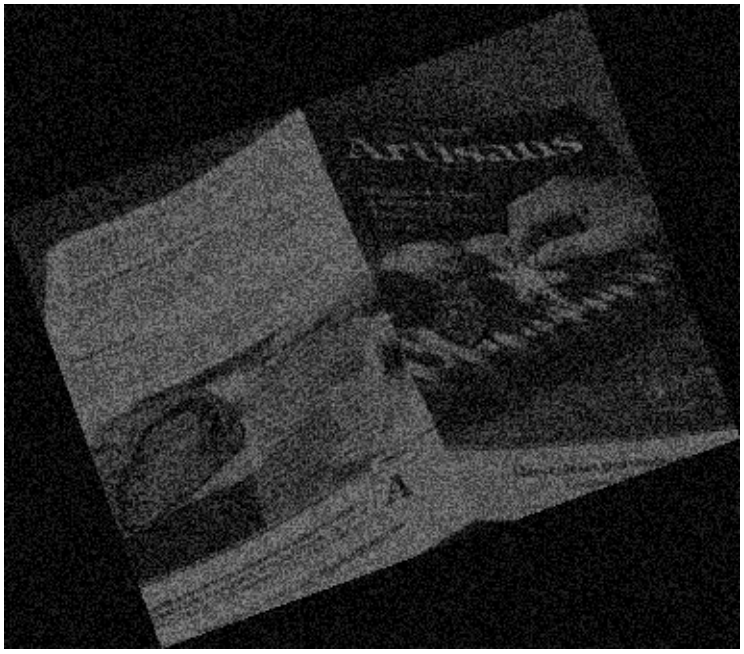
In[ ]:=

```
(*get phases distorted for N=10*)
camSCodesN10Distorted0 = distortCamera[#, noisyValue, rotDegree] & /@ simgsN10;
imdata = Flatten[{#, ImageData[disCamImg]}, {{2}, {3}, {1}}] & /@
    (ImageData /@ camSCodesN10Distorted0);
camSCodesN10Distorted = Image[Map[Mean, #, {2}]] & /@ imdata
camSCodesN10DistDecode =
  ImageData[ColorConvert[#, "GrayScale"]] & /@ camSCodesN10Distorted;
camSCodesN10DistDecode = Flatten[camSCodesN10DistDecode, {{2}, {3}, {1}}];
Dimensions[camSCodesN10DistDecode]
Map[Min, camSCodesN10DistDecode, {2}] // Image
Map[Max, camSCodesN10DistDecode, {2}] // Image
Map[Mean, camSCodesN10DistDecode, {2}] // Image
maskN10 = (Map[Max, camSCodesN10DistDecode, {2}] -
    Map[Min, camSCodesN10DistDecode, {2}]);


maskN10 // Image
maskN10 = Map[If[# < 0.15, 0, 1] &, maskN10, {2}];
Image[maskN10]


gDistCoordDecode[[150, 250]]
Print["No. of cycle for (150,250):"]
unwrap[gDistCoordDecode[[150, 250]], cycle, w]
allDistK = Map[unwrap[#, cycle, w] &, gDistCoordDecode, {2}];
```

Out[•]= {  }

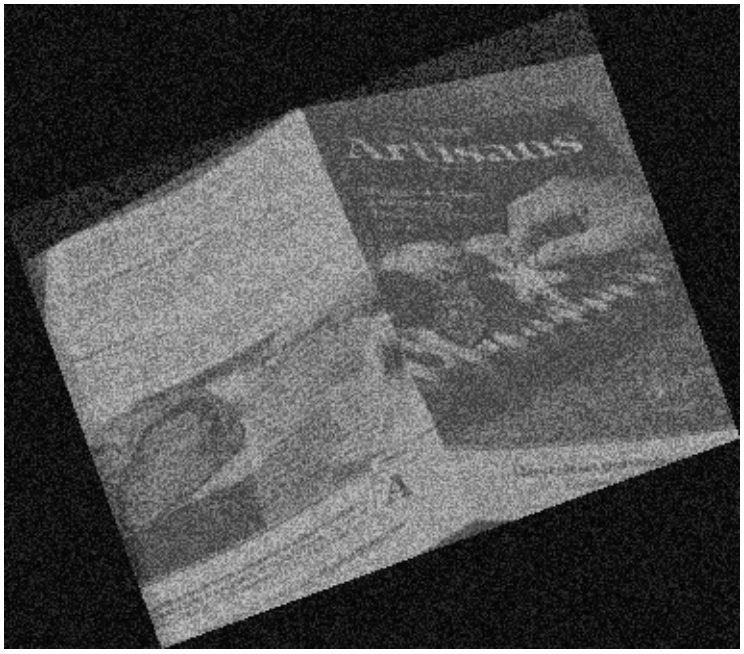
Out[•]= {335, 383, 10}

Out[ ]=

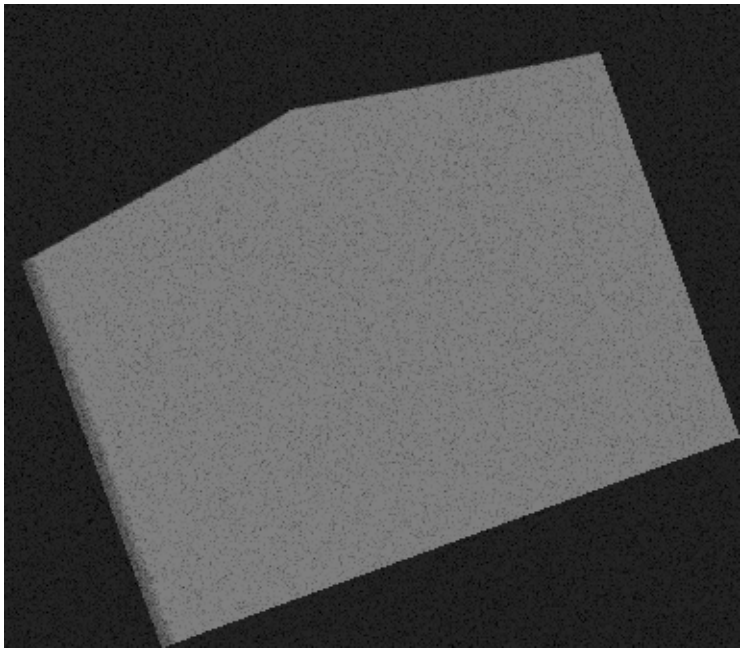


Out[ ]=

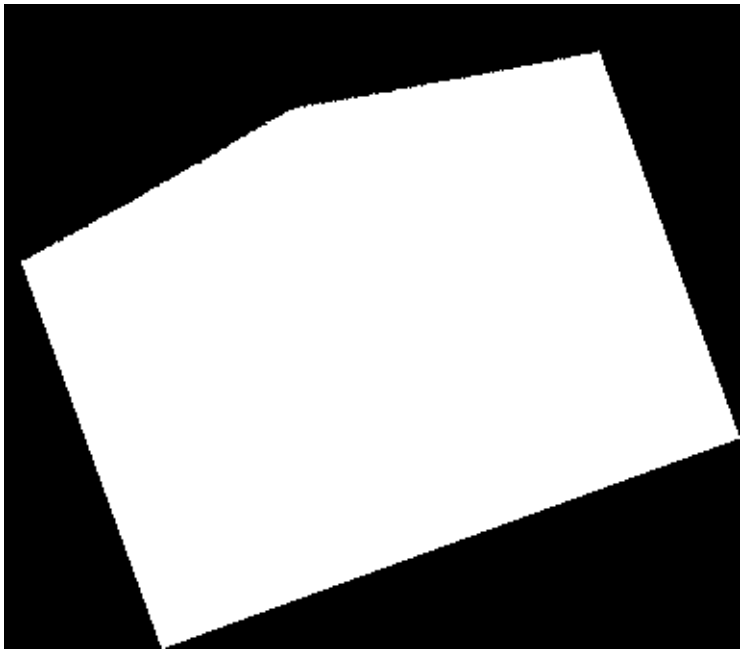
Out[ ]=



Out[ ]=



Out[•]=



Out[•]= {136, 117, 0}

No. of cycle for (150,250):

Out[•]= 4

In[•]:=

```
camSCodesN10DistDecodeWithK =
  Table[Join[camSCodesN10DistDecode[[i, j]], {allDistK[[i, j]]}],
  |table   |joins
   {i, 1, Dimensions[allDistK][[1]]}, {j, 1, Dimensions[allDistK][[2]]}];
          |dimensions                          |dimensions
camSCodesN10DistDecodeWithK[[150, 250]]
Print["phase cycle for (150,250) with upwrap:"]
|imprime
sinAbsoluteDecode[camSCodesN10DistDecodeWithK[[150, 250]], cycle, w, upperN]
```

Out[•]= {0.5, 0.401139, 0.417542, 0.0543826,
 0.156067, 0., 0.0437588, 0.191476, 0.426993, 0.5, 4}

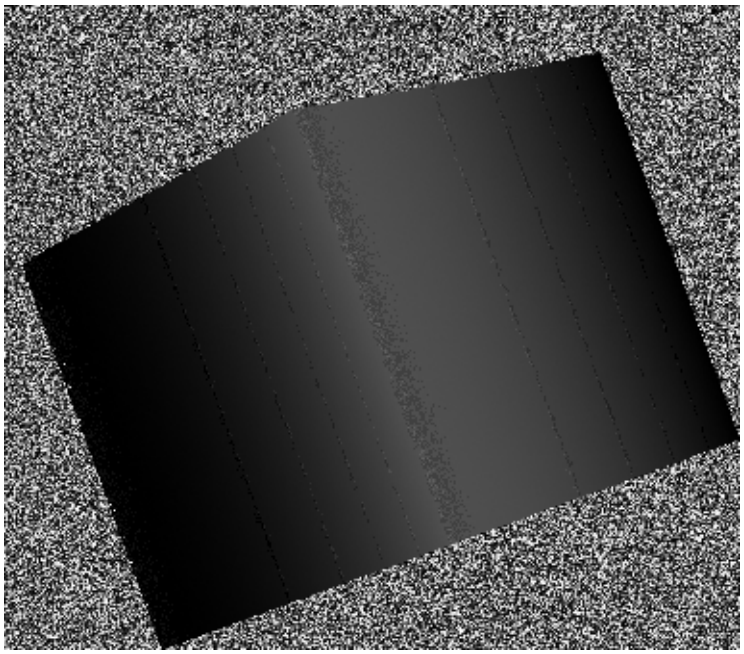
phase cycle for (150,250) with upwrap:

Out[•]= sinAbsoluteDecode[{0.5, 0.401139, 0.417542, 0.0543826,
  0.156067, 0., 0.0437588, 0.191476, 0.426993, 0.5, 4}, 10, 320, 10]

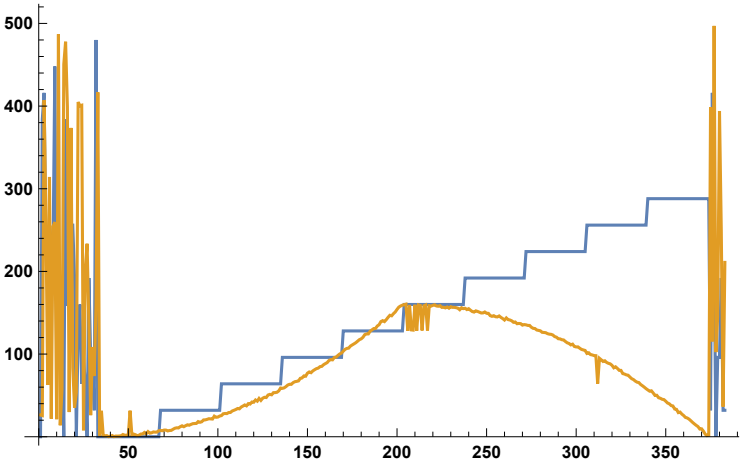
In[ ]:=

```
phasesDistortN10Unwrap = Map[sinAbsoluteDecode[#, cycle, w, upperN, adjust] &,
                         applique
    camSCodesN10DistDecodeWithK, {2}];
Image[phasesDistortN10Unwrap] // ImageAdjust
image                                ajuste image

ListPlot[{camSCodesN10DecodeWithK[[200, ;; , upperN + 1]] * w / cycle,
tracé de liste
  phasesDistortN10Unwrap[[200]]}, Joined → True]
                                  joint       vrai
Print["The origianl sinus image of distortion:"]
imprime
ImageTransformation[simgsN10[[1]], f @@ # &]
transformée d'image
```

Out[ ]=

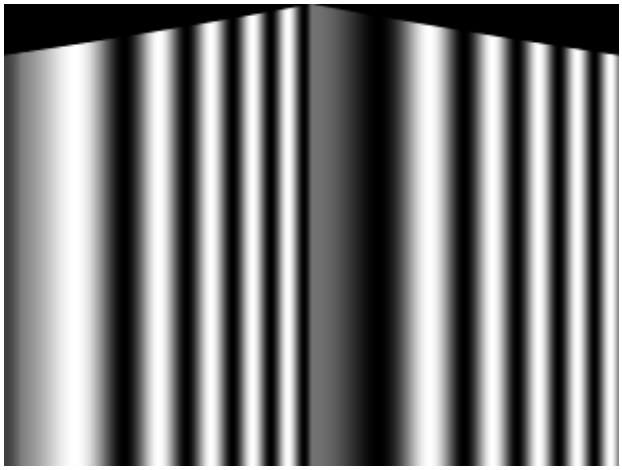


Out[ ]=



The origianl sinus image of distortion:

Out[ ]=



The cause of the anomaly of the image of the highest area:
The distortion function copies the image on the left to the right when it doing the distortion.

Reconstruct height (an approximate height) For N=10 :

In[ ]:=
```
Print["Reconstruct using shift code:"]
```
imprime

```
L = 800;(*mm*)
B = 600;(*mm*)
(*subPhaseN10=phasesDistortN10Unwrap-phasesN10Unwrap0;
subPhaseN10[[150,191]]
  heightsN10=subPhaseN10*L/B//N;*)
```
valeur numérique

```
Print["with N=10 : "]
```
imprime valeur numérique

```
ListPlot3D[-phasesDistortN10Unwrap * maskN10]
```
tracé de liste 3D

```
Print["comparaison with N=4 : "]
```
imprime valeur numérique

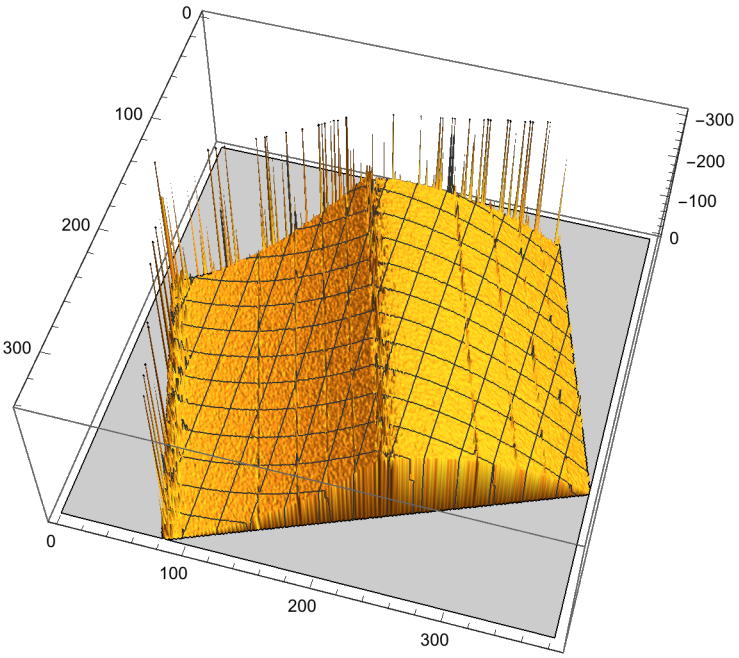
```
plotN4mask
Print["comparaison with Graycode reconstruction:"]
```
imprime

```
plot3dGcode
Print["The original distortion:"]
```
imprime

```
ImageTransformation[img2, f@@# &]
```
transformée d'image

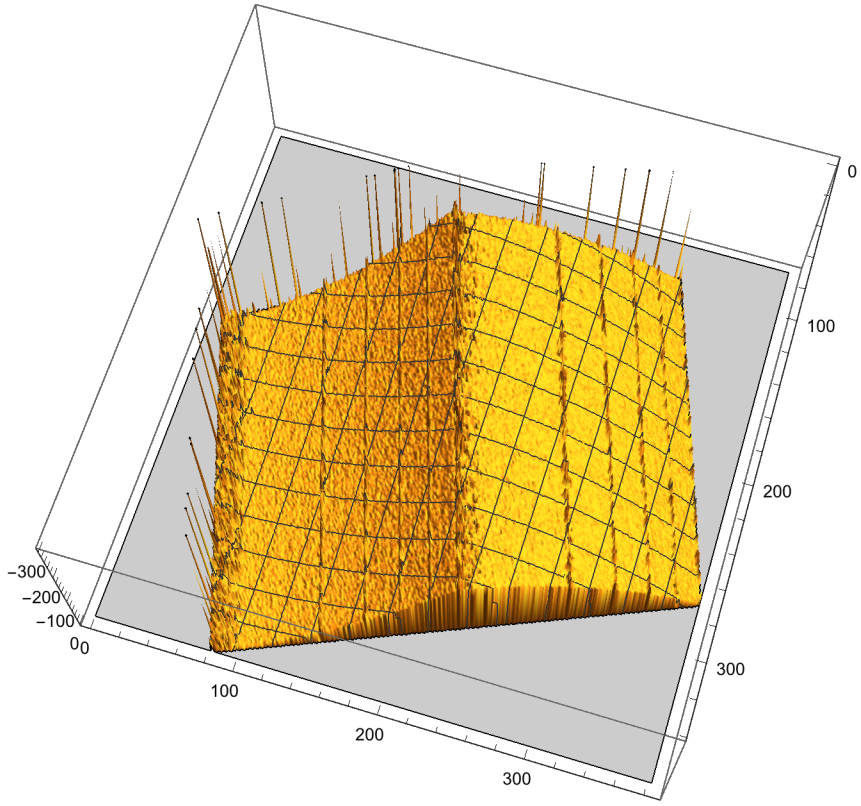
```
Reconstruct using shift code:
with N=10 :
```

Out[ ]=



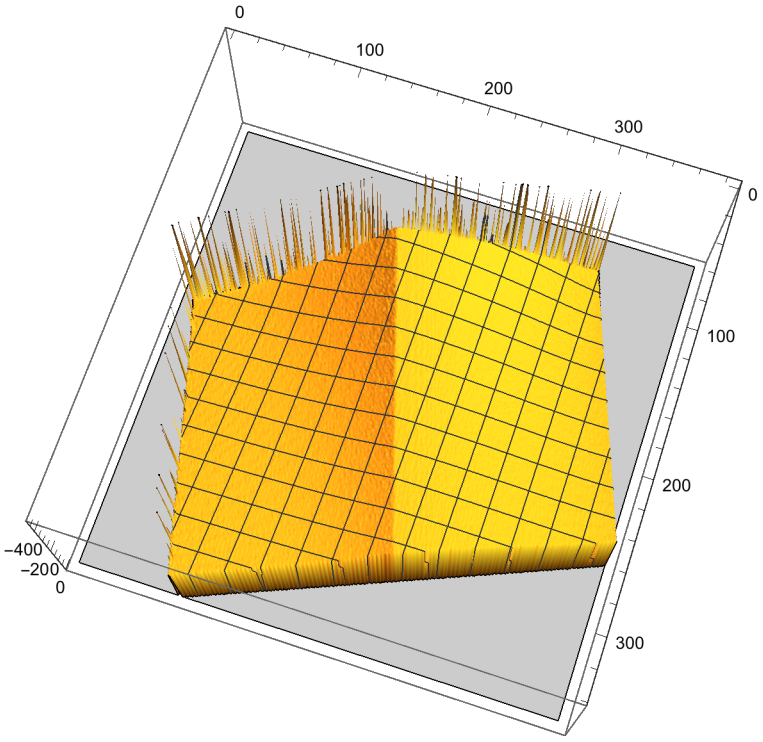
comparaison with N=4 :

Out[ ]=



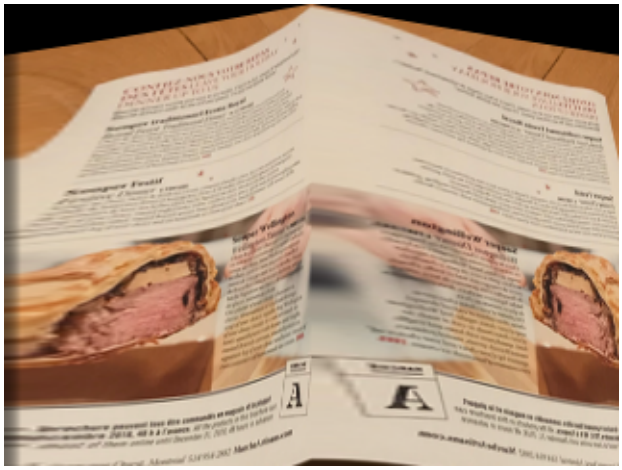
comparaison with Graycode reconstruction:

Out[ ]=



The original distortion:

Out[ ]=



In[ ]:=

We observed that n=10 has more detail and more smooth than n=4.

Normally a cycle can contain one to several graycode addresses (pixel precision), then the solution of gray code + phase shifting will provide high quality and smoothing 3D reconstruction.

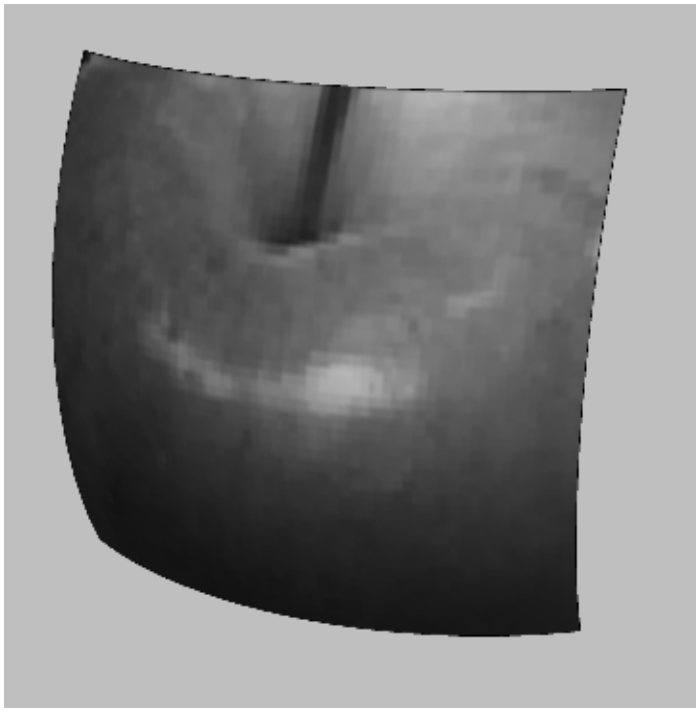
In[ ]:=

Projection part:

Gray Code :

```
In[ ]:= projectionPattern[shape_, pattern_, view_] := Module[{yo, im}, (
                                                       module

    yo = ContourPlot3D[shape, {x, -2, 2}, {y, -2, 2}, {z, -2, 2},
         tracé 3D de champ scalaire par ses contours

      Lighting → "Neutral", ContourStyle → Texture[pattern],
      éclairage              style de contour   texture

      TextureCoordinateFunction → ({#1, #2} &), ViewPoint → view,
      fonction de coordonnées de texture          point de vue spatial

      ViewVertical → {0, 1, 0}, ViewCenter → {0, 0, 0}, Boxed → False,
      direction de vue verticale  centre de vue          entouré···faux

      Axes → False, Mesh → None, Background -> Lighter[Gray, 0.5],
      axes  faux  maille  aucun  image de fond  plus clair  gris

      PlotRange → {{-2, 2}, {-2, 2}, {-2, 2}}, BoxRatios → Automatic];
      zone de tracé                             rapports de b···automatique

    im = Rasterize[yo];
         convertis en carte de bits

    ColorConvert[im, "Grayscale"]
    convertis couleur

  )];

pattern =  ;


noisify[im_, v_] :=
  Image[Map[(Max[0, Min[1, # + RandomReal[{-v, v}]]]) &, ImageData[im], {2}]];
  image  appl···maxi···  minimum  nombre réel aléatoire          données d'image

img3 = projectionPattern[x^2 + y^2 + (z + 2)^2 == 16, pattern, {1, 1, 3}]

{w, h} = ImageDimensions[img3]
         dimensions d'image


nAddr = Length[IntegerDigits[w - 1, 2]];
        longueur chiffres d'entier

noisyValue = 0.2;

rotDegree = 20. Degree;
                degré

camera[pattern_, v_, deg_] :=
  noisify[ImageRotate[
          fais pivoter image

    projectionPattern[x^2 + y^2 + (z + 2)^2 == 16, pattern, {1, 1, 3}], deg], v];

camImg = camera[pattern, noisyValue, rotDegree];
```

Out[•]=

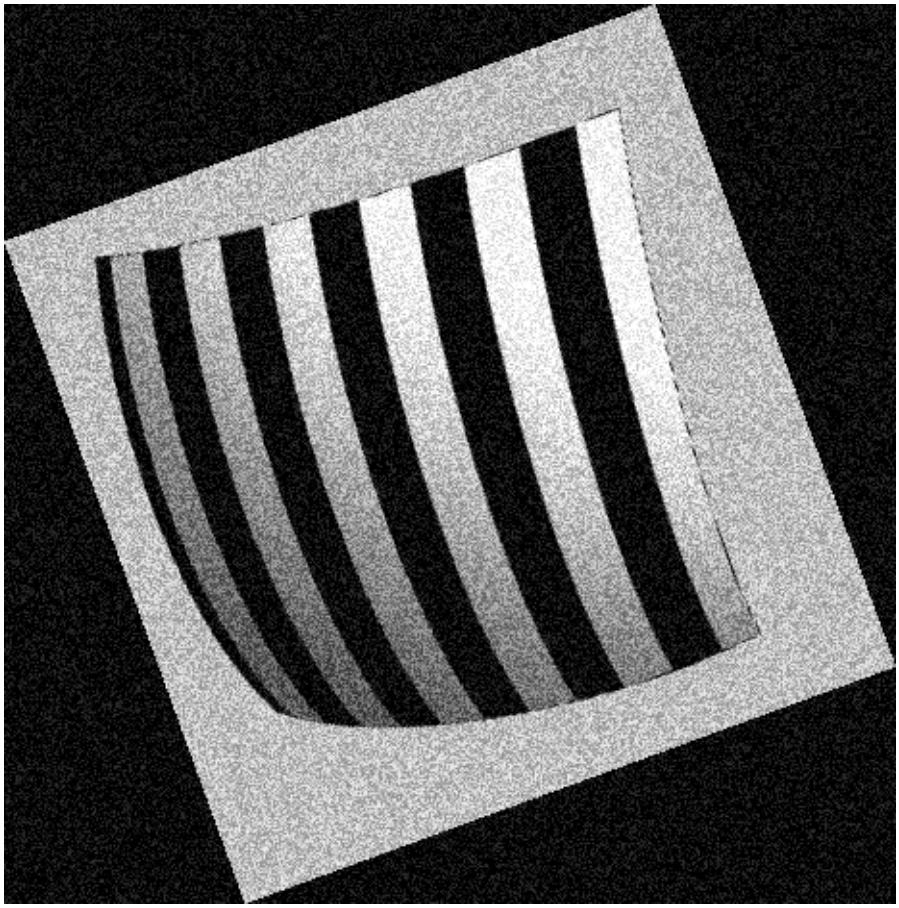


Out[•]= {360, 366}

In[•]:=

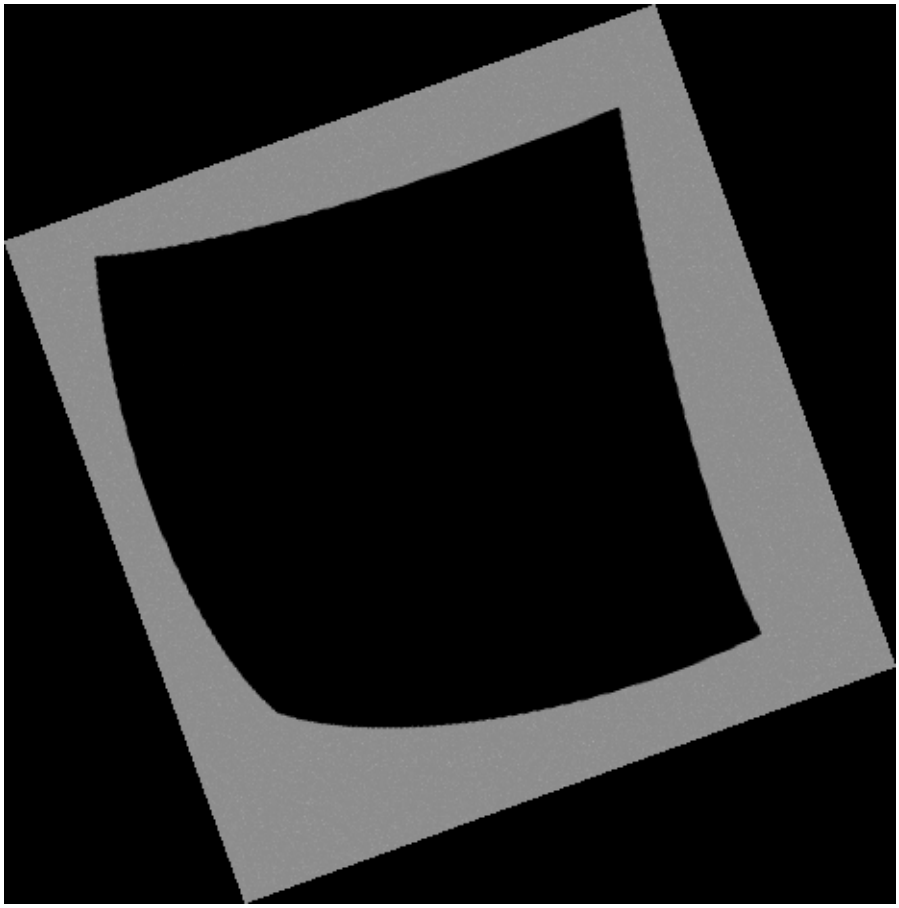
```
gcodes = Table[graycode[x, y, nAddr], {y, 0, h - 1}, {x, 0, w - 1}];
gcodes = Flatten[gcodes, {{3}, {1}, {2}}];
gimgs = Image /@ gcodes;
camGCodes0 = camera[#, noisyValue, rotDegree] & /@ gimgs;
camGCodes0[[5]]

camGCodesDecode = ImageData[ColorConvert[#, "GrayScale"]] & /@ camGCodes0;
camGCodesDecode = Flatten[camGCodesDecode, {{2}, {3}, {1}}];
Dimensions[camGCodesDecode]
Map[Min, camGCodesDecode, {2}] // Image
Map[Max, camGCodesDecode, {2}] // Image;
Map[Mean, camGCodesDecode, {2}] // Image;
maskGcode = (Map[Max, camGCodesDecode, {2}] - Map[Min, camGCodesDecode, {2}]);
maskGcode // Image;
maskGcode = Map[If[# < 0.4, 0, 1] &, maskGcode, {2}];
Image[maskGcode]
```

Out[ ]=

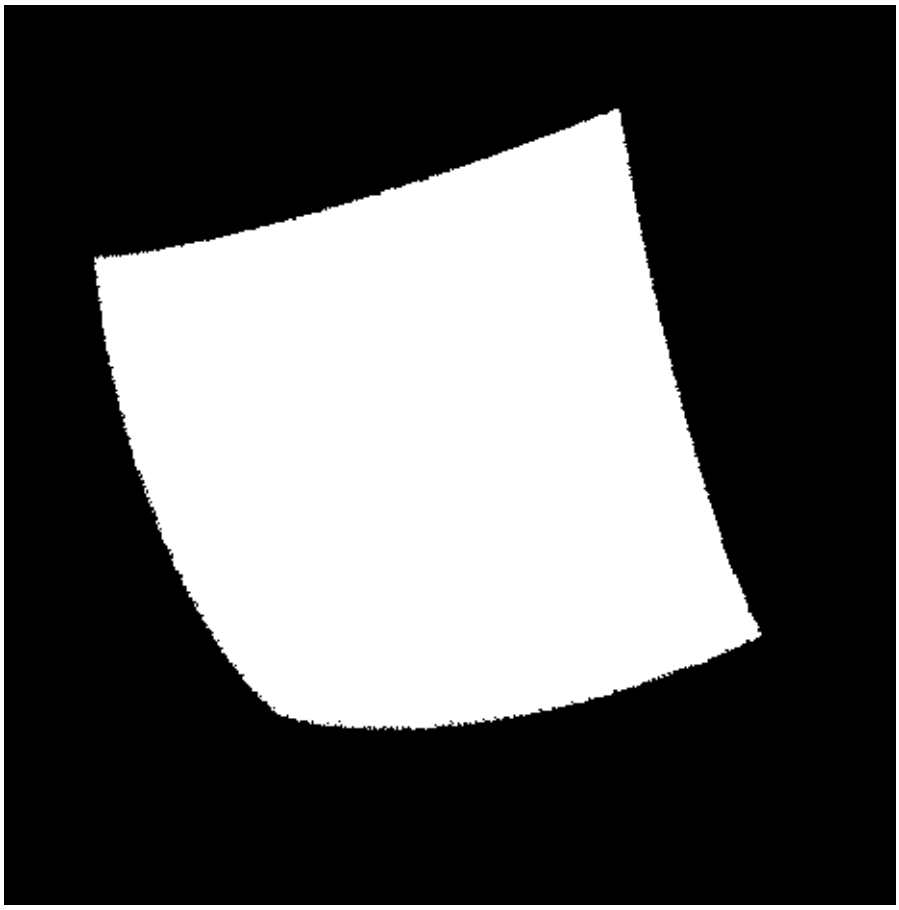


Out[ ]= {468, 464, 36}

Out[ ]=



Out[ ]=



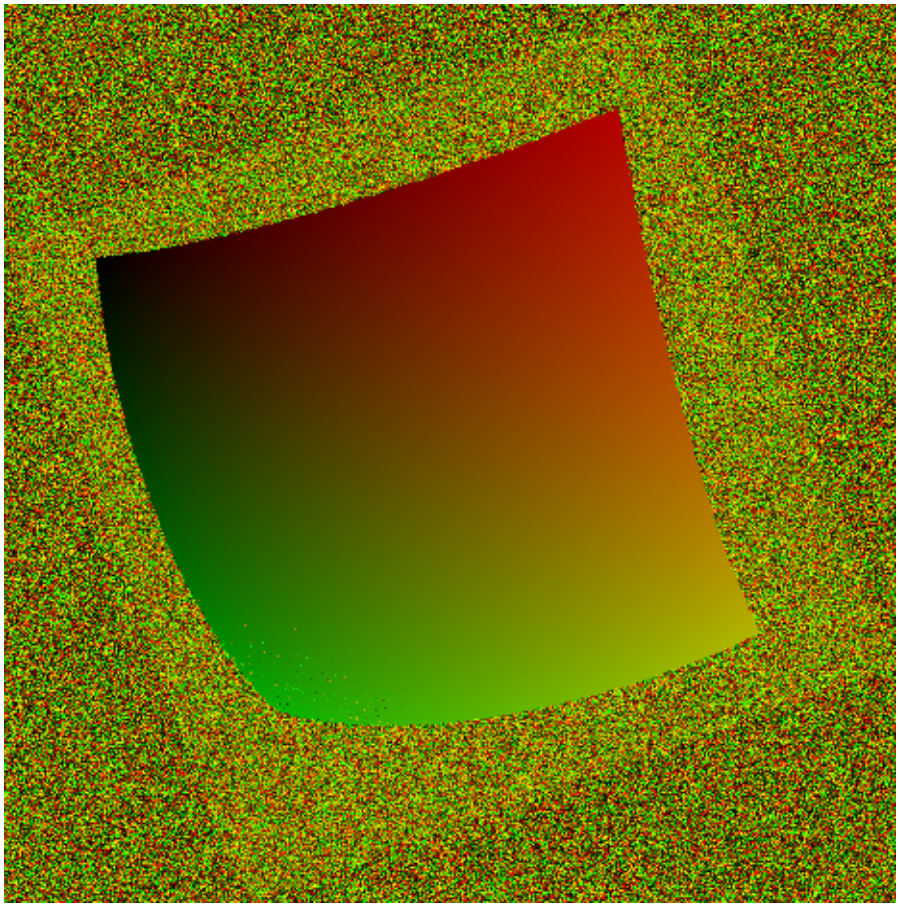
In[ ]:=

```
gCoordDecode = Map[grayDecode[#, nAddr] &, camGCodesDecode, {2}];
               applique
gCoordDecode[[150 ;; 154, 190]]
Dimensions[gCoordDecode]
dimensions
Image[gCoordDecode] // ImageAdjust
image                  ajuste image


allD = Map[Norm, gCoordDecode, {2}] // N;
       app  norme                    valeur numérique
allD[[150, 191]]
plotGcode = ListPlot3D[-allD * maskGcode]
            tracé de liste 3D
```

Out[ ]= {{204, 49, 0}, {203, 50, 0}, {203, 51, 0}, {203, 52, 0}, {203, 53, 0}}

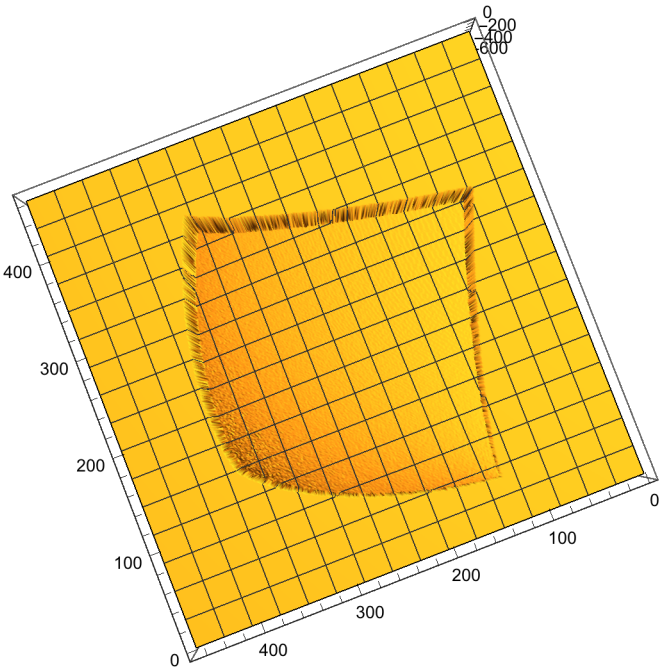
Out[ ]= {468, 464, 3}

Out[ ]=



Out[ ]= 210.775

Out[ ]=



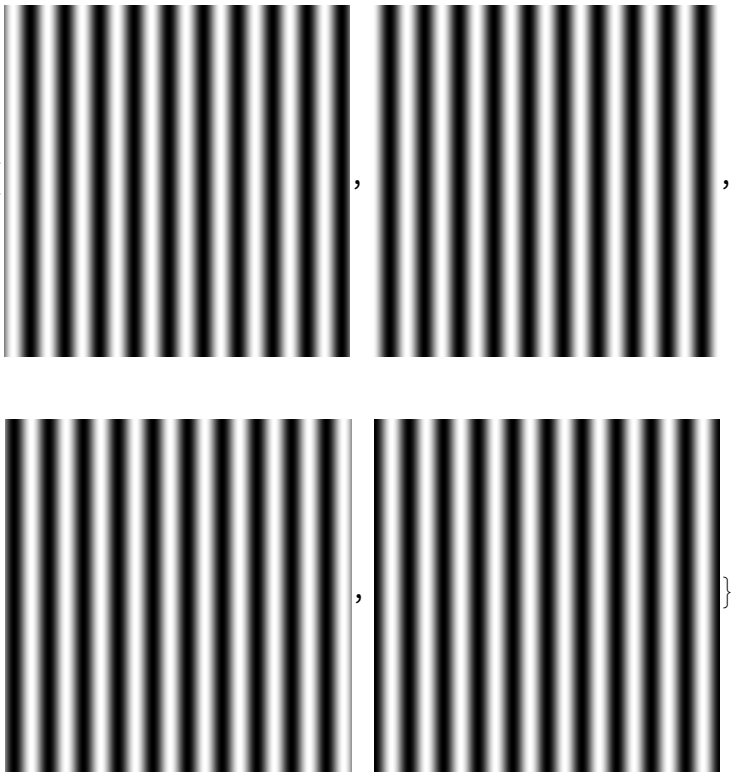
Sinus Code, N = 4

In[ ]:=

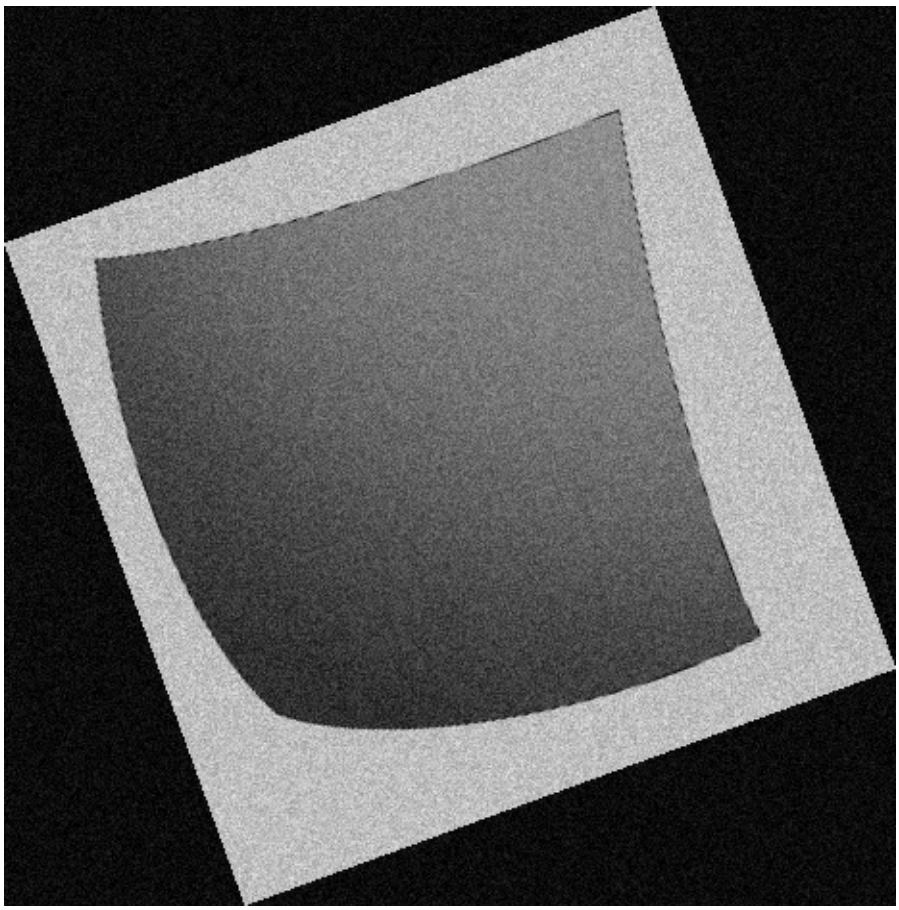
```
Print["sinus N=4 :"]
(* imprime  valeur numérique *)

upperN = 4;

adjust = 0 (*10->1.25*)

cycle = 10;

scodesN10 = Table[sincode[x, y, cycle, w, upperN], {y, 0, h - 1}, {x, 0, w - 1}];
(* table *)

scodesN10 = Flatten[scodesN10, {{3}, {1}, {2}}];
(* aplatis *)

(*ListLinePlot[scodesN4[[All,1,All]]]*)
(* tracé de liste de ligne  tout  tout *)

simgsN10 = Image /@ scodesN10
(* image *)


camSCodesN100 = camera[#, noisyValue, rotDegree] & /@ simgsN10;

camSCodesN10Decode0 =
  ImageData[ColorConvert[#, "GrayScale"]] & /@ camSCodesN100;
(* données d'···  convertis couleur *)

camSCodesN10Decode0 = Flatten[camSCodesN10Decode0, {{2}, {3}, {1}}];
(* aplatis *)


Map[Min, camSCodesN10Decode0, {2}] // Image;
(* app·  minimum  image *)

Map[Max, camSCodesN10Decode0, {2}] // Image;
(* app·  maximum  image *)

Map[Mean, camSCodesN10Decode0, {2}] // Image
(* app·  valeur moyenne  image *)

maskN10 =
  (Map[Max, camSCodesN10Decode0, {2}] - Map[Min, camSCodesN10Decode0, {2}]);
(* app·  maximum  app·  minimum *)

maskN10 // Image;
(* image *)

maskN10 = Map[If[# < 0.2, 0, 1] &, maskN10, {2}];
(* app··  si *)

Image[maskN10]
(* image *)

sinus N=4 :
```

Out[ ]= 0

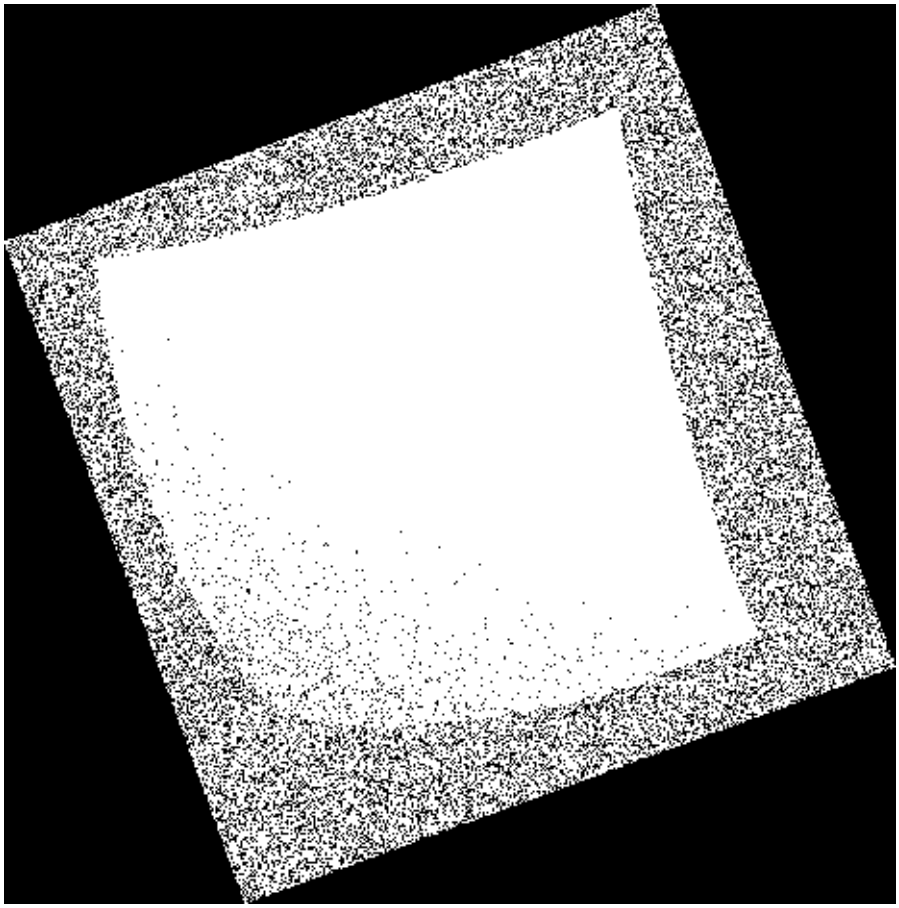
Out[ ]=



Out[ ]=



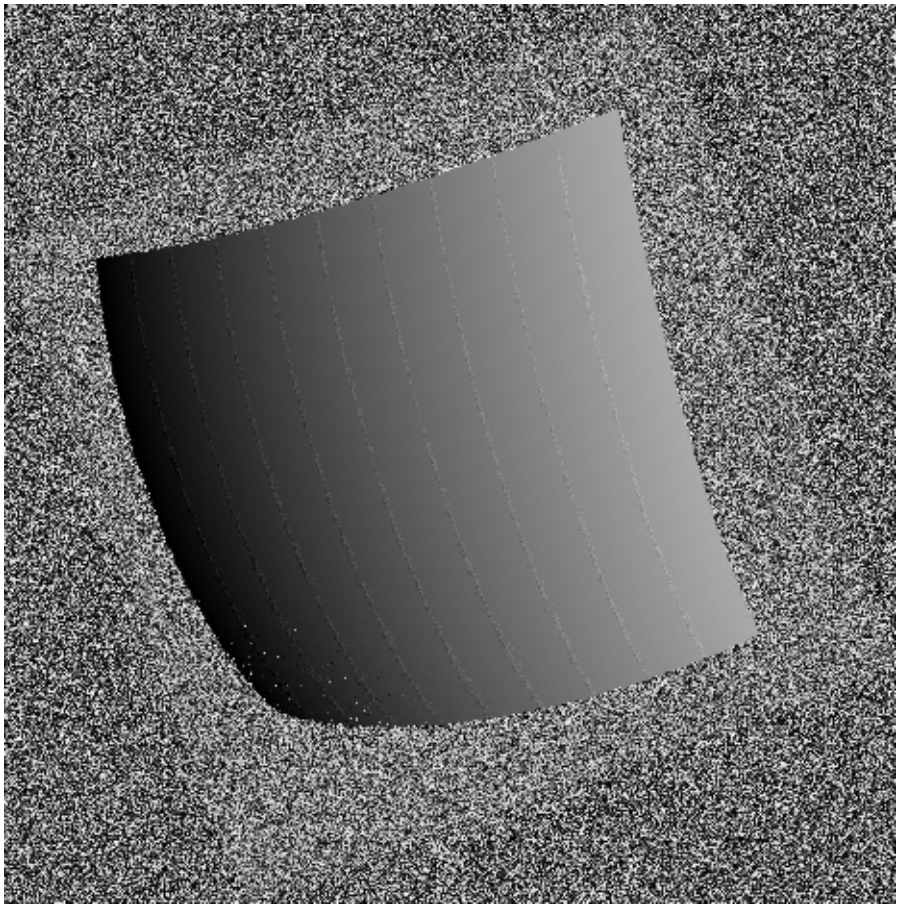
Out[ ]=



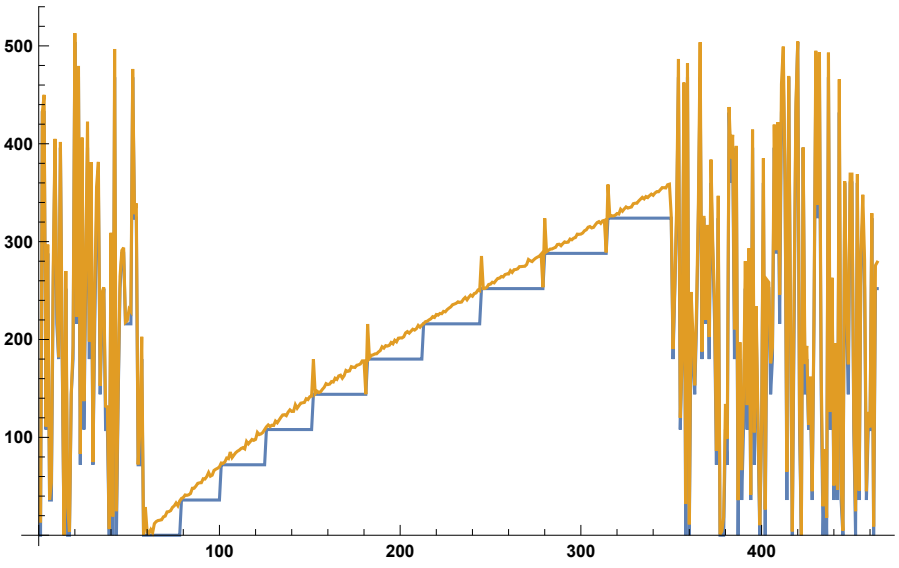
In[ ]:=

```
allK = Map[unwrap[#, cycle, w] &, gCoordDecode, {2}];
       applique
camSCodesN10DecodeWithK =
  Table[Join[camSCodesN10Decode0[[i, j]], {allK[[i, j]]}],
  table   joins
   {i, 1, Dimensions[allK][[1]]}, {j, 1, Dimensions[allK][[2]]}];
          dimensions                    dimensions

phasesN10Unwrap0 = Map[sinAbsoluteDecode[#, cycle, w, upperN, adjust] &,
                       applique
    camSCodesN10DecodeWithK, {2}];
Image[phasesN10Unwrap0] // ImageAdjust
image                      ajuste image
ListPlot[{camSCodesN10DecodeWithK[[200, ;; , upperN + 1]] * w / cycle,
tracé de liste
  phasesN10Unwrap0[[200]]}, Joined → True]
                            joint      vrai
```

Out[ ]=



Out[ ]=

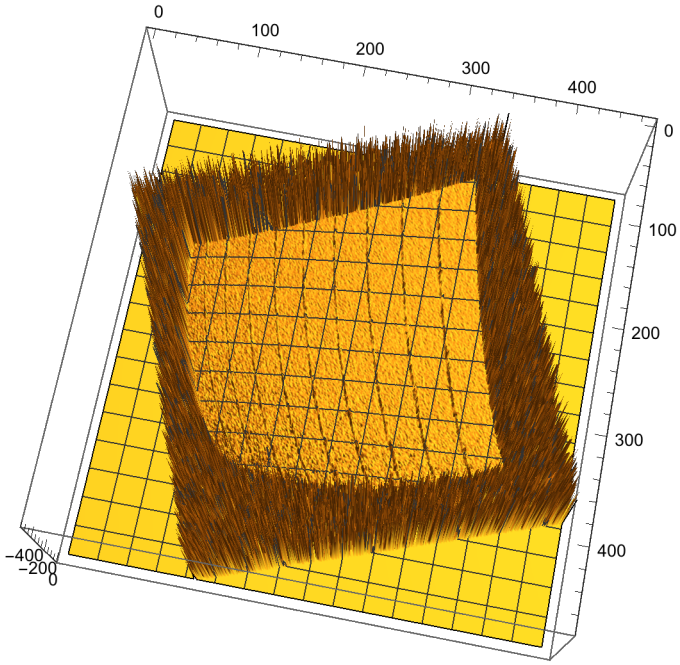


In[ ]:=

```
Print["Reconstruct with sinus N=4:"]
imprime                                valeur numé
ListPlot3D[-phasesN10Unwrap0 * maskN10]
tracé de liste 3D
Print["Comparison with graycode:"]
imprime
plotGcode
```

Reconstruct with sinus N=4:

Out[ ]=



Comparison with graycode:

Out[ ]=