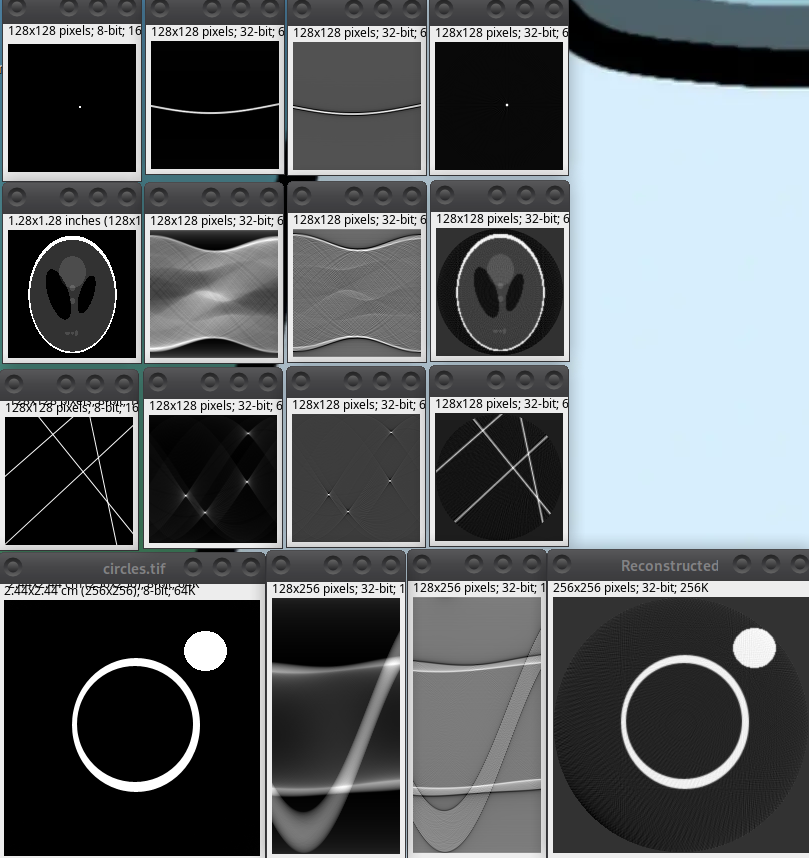
**Nome: Lucas Miranda Mendonça Rezende No. USP: 12542838**

**Relatorio.doc Transformada de Radon e retro-projeção**

**Solução Questão 1**

****

**Solução Questão 2. Transformada de retro-projeção**

Código:

public static ImageAccess inverseRadon(ImageAccess sinogram) {

int nbAngles = sinogram.getWidth();

int size = sinogram.getHeight();

double b[][] = new double[size][size];

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

b[i][j] = 0.0;

}

}

double[][] sinogramData = new double[nbAngles][size];

for (int a = 0; a < nbAngles; a++) {

for (int k = 0; k < size; k++) {

sinogramData[a][k] = sinogram.getPixel(a, k);

}

}

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

double sum = 0.0;

double x = j - size / 2.0;

double y = i - size / 2.0;

for (int a = 0; a < nbAngles; a++) {

double theta = a \* Math.PI / nbAngles;

double t = x \* Math.cos(theta) + y \* Math.sin(theta) + size / 2.0;

double value = getInterpolatedPixel1D(sinogramData[a], t);

sum += value;

}

b[i][j] = sum;

}

}

ImageAccess reconstudedImage = new ImageAccess(b);

return reconstudedImage;

}

private static double getInterpolatedPixel1D(double vector[], double t) {

int index = (int) floor(t);

double fraction = t - index;

if (index < 0 || index >= vector.length - 1) {

if (index == vector.length - 1 && fraction == 0)

return vector[index];

return 0.0;

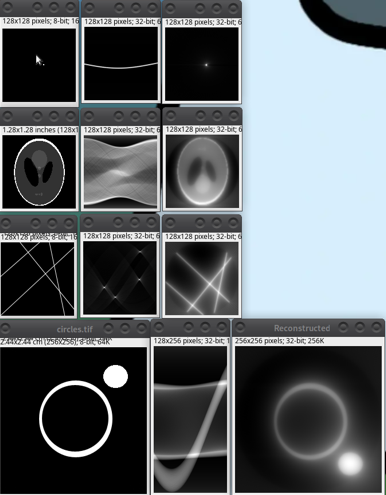
}

double interpolatedValue = vector[index] \* (1 - fraction) + vector[index + 1] \* fraction;

return interpolatedValue;

}

| Teste: |
| --- |
| Coloque uma imagem 8 bits |



**Solução Questão 3. Reconstrução de um sinograma**

Código:

public static ImageAccess applyRamLakFilter(ImageAccess sinogram)

{

int nbAngle = sinogram.getWidth();

int size = sinogram.getHeight();

double[] real = new double[size];

double[] imaginary = new double[size];

double[] filter = generateRamLak(size);

ImageAccess output = new ImageAccess(nbAngle, size);

RadonFFT1D fft = new RadonFFT1D(size);

for (int k=0; k<nbAngle; k++) {

sinogram.getColumn(k, real);

for(int l=0; l<size; l++) {

imaginary[l] = 0.0;

}

fft.transform(real, imaginary);

for(int l=0; l<size; l++) {

real[l] = real[l] \* filter[l];

imaginary[l] = imaginary[l] \* filter[l];

}

fft.inverse(real, imaginary);

output.putColumn(k, real);

}

return output;

}

public static double[] generateRamLak(int size) {

double[] filter = new double[size];

int center = size / 2;

for (int i = 0; i < size; i++) {

double omega = i - center;

filter[i] = Math.abs(omega);

}

return filter;

}

public static ImageAccess applyCosineFilter(ImageAccess sinogram) {

int nbAngle = sinogram.getWidth();

int size = sinogram.getHeight();

ImageAccess output = new ImageAccess(nbAngle, size);

RadonFFT1D fft = new RadonFFT1D(size);

double[] cosineFilter = generateCosine(size);

for (int a = 0; a < nbAngle; a++) {

double[] projReal = new double[size];

double[] projImag = new double[size];

for (int k = 0; k < size; k++) {

projReal[k] = sinogram.getPixel(a, k);

projImag[k] = 0.0;

}

fft.transform(projReal, projImag);

for (int k = 0; k < size; k++) {

projReal[k] \*= cosineFilter[k];

projImag[k] \*= cosineFilter[k];

}

fft.inverse(projReal, projImag);

for (int k = 0; k < size; k++) {

output.putPixel(a, k, projReal[k]);

}

}

return output;

}

public static double[] generateCosine(int size) {

double[] filter = new double[size];

int center = size / 2;

for (int i = 0; i < size; i++) {

double omega = Math.abs(i - center);

filter[i] = omega \* Math.cos(Math.PI \* omega);

}

return filter;

}

public static ImageAccess applyLaplacianFilter(ImageAccess sinogram) {

int nbAngle = sinogram.getWidth();

int size = sinogram.getHeight();

ImageAccess output = new ImageAccess(nbAngle, size);

for (int a = 0; a < nbAngle; a++) {

for (int k = 0; k < size; k++) {

double left, center, right;

center = sinogram.getPixel(a, k);

if (k == 0) {

left = sinogram.getPixel(a, 1);

right = sinogram.getPixel(a, k + 1);

} else if (k == size - 1) {

left = sinogram.getPixel(a, k - 1);

right = sinogram.getPixel(a, size - 2);

} else {

left = sinogram.getPixel(a, k - 1);

right = sinogram.getPixel(a, k + 1);

}

double value = 1.0 \* left - 2.0 \* center + 1.0 \* right;

output.putPixel(a, k, value);

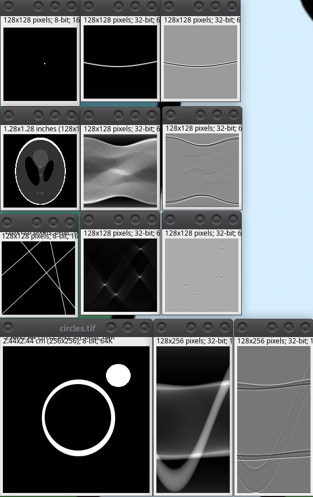
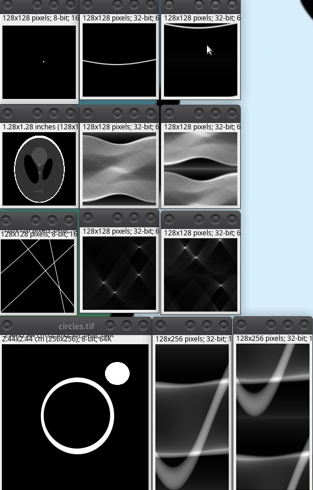
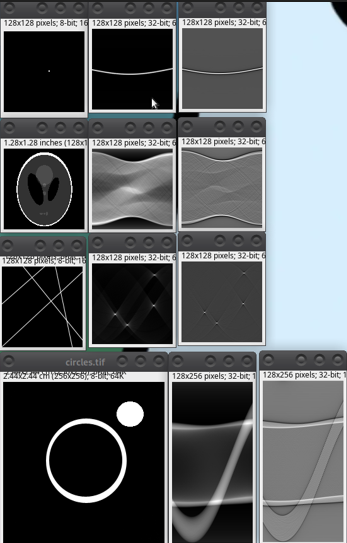
}

}

return output;

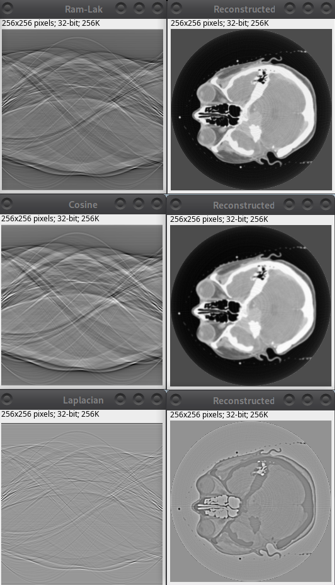
}

RamLak/Cosine/Laplace:



**Solução Questão 4. Reconstrução de um sinograma**

| Filtro: |
| --- |
| Coloque a imagem 8 bits    (RamLak) |



**Solução Questão 5. Detecção de linhas**

| Filtro: Laplace  Valor Threshold: 142 |
| --- |
| Coloque a imagem 8 bits |

