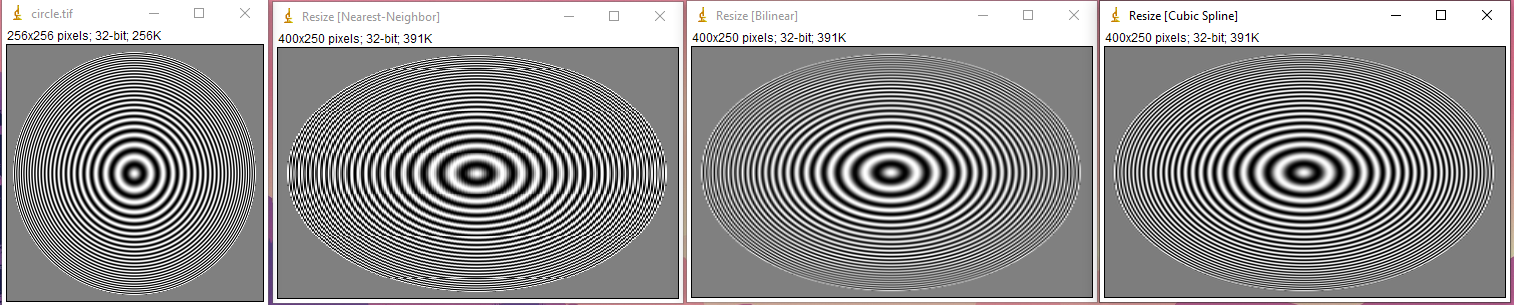
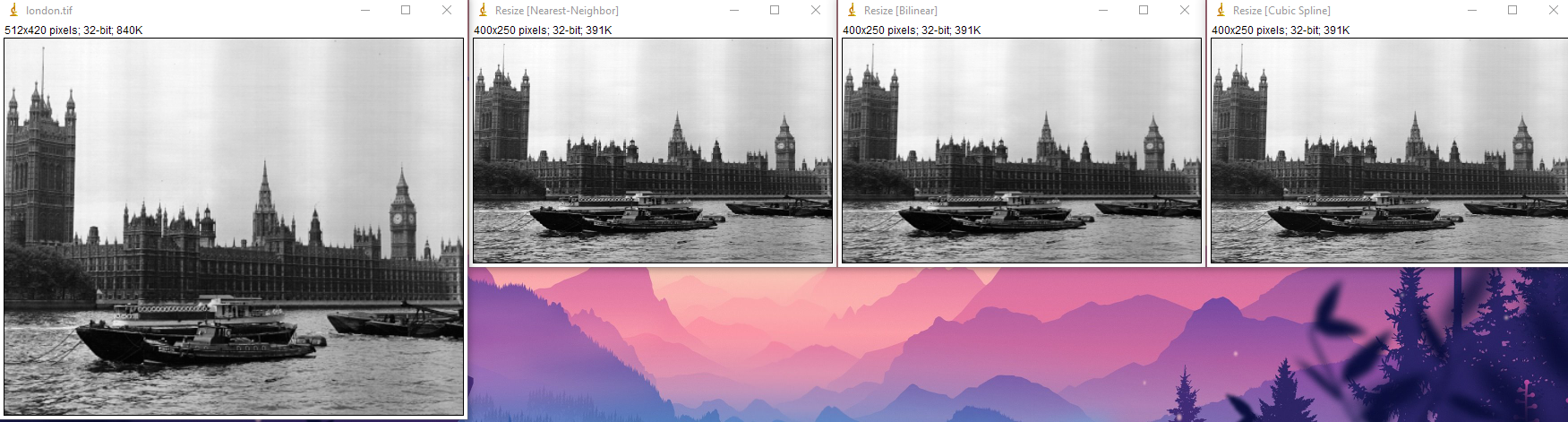
**Nome: Lucas Miranda Mendonça Rezende**

Nro. USP: 12542838

**relatório.doc Interpolação e transformação geométrica**

**Questão 1**

****

****

**Questão 1.1**

Apenas explicativa.

**Questão 1.2**

Código:

private static double getInterpolatedPixelNearestNeigbor(ImageAccess image, double x, double y) {

int i = (int) Math.round(x);

int j = (int) Math.round(y);

i = Math.min(Math.max(i, 0), image.getWidth() - 1);

j = Math.min(Math.max(j, 0), image.getHeight() - 1);

return image.getPixel(i, j);

}

**Questão 1.3**

Código:

**private static double getInterpolatedPixelCubicSpline(ImageAccess coef, double x, double y) {**

**// floor to get the “upper‑left” integer pixel**

**int m = (int) Math.floor(x);**

**int n = (int) Math.floor(y);**

**// grab the 4×4 neighborhood of SPLINE COEFFICIENTS around (m,n)**

**double[][] neighbor = new double[4][4];**

**coef.getNeighborhood(m, n, neighbor);**

**// fractional offsets inside that 4×4 block**

**double dx = x - m;**

**double dy = y - n;**

**// evaluate the 2D tensor‑product B‑spline basis**

**return getSampleCubicSpline(dx, dy, neighbor);**

**}**

**static private double getSampleCubicSpline(double x, double y, double neighbor[][]) {**

**double sum = 0.0;**

**double[] cubicSplineRow = getCubicSpline(x);**

**double[] cubicSplineCol = getCubicSpline(y);**

**for (int i = 0; i < 4; ++i) {**

**for (int j = 0; j < 4; ++j) {**

**sum += neighbor[j][i] \* cubicSplineRow[j] \* cubicSplineCol[i];**

**}**

**}**

**return sum;**

**}**

**static private double[] getCubicSpline(double t) {**

**double v[] = new double[4];**

**if (t < 0.0 || t > 1.0) {**

**throw new ArrayStoreException(**

**"Argument t for cubic B-spline outside of expected range.");**

**}**

**double t1 = 1.0 - t;**

**double t2 = t \* t;**

**v[0] = (t1 \* t1 \* t1) / 6.0;**

**v[1] = (2.0 / 3.0) + 0.5 \* t2 \* (t-2);**

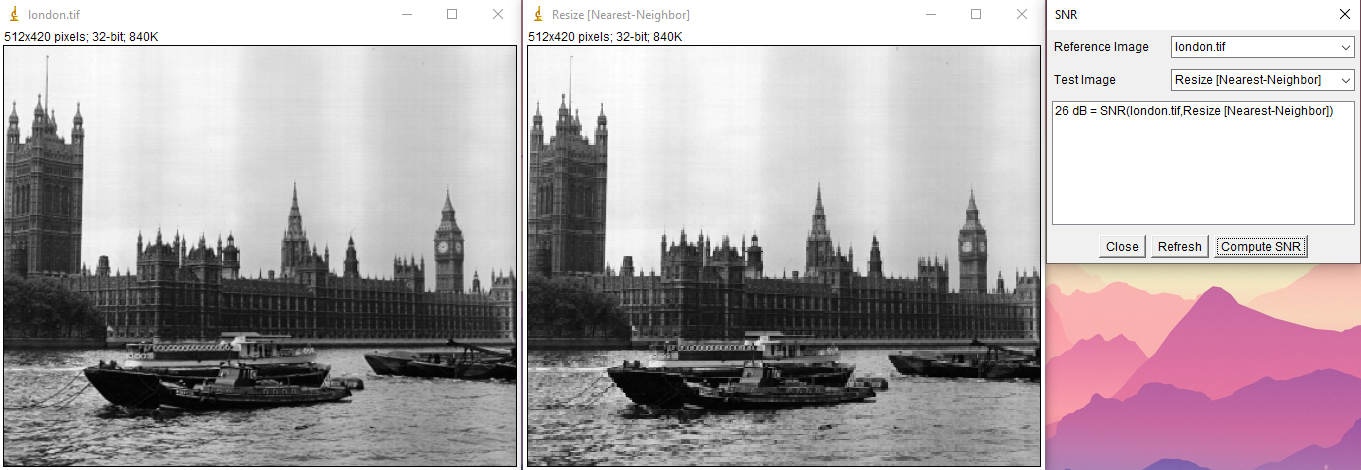
**v[3] = (t2 \* t) / 6.0;**

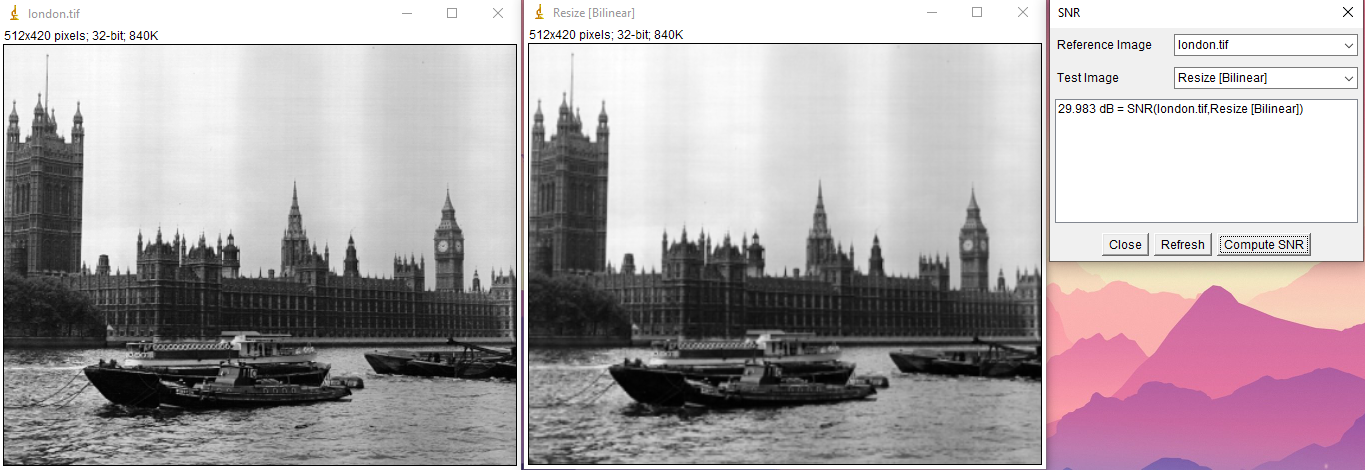
**v[2] = 1.0 - v[3] - v[1] - v[0];**

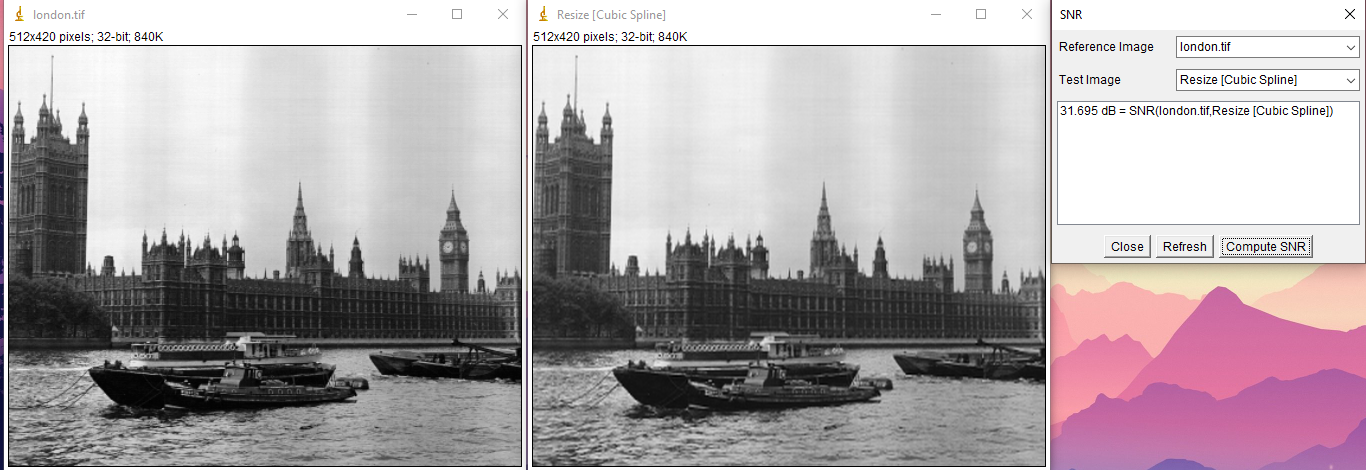
**return v;**

**}**

**Questão 1.4**

****

****

****

Reduzir/Ampliar london.tif

Interpolação SNR (relação sinal / ruído)

--------------------------------

Nearest-Neighbor *26* dB

Linear B-Spline *29.983* dB

Cubic B-Spline *31.695* dB

**Questão 2**

Código:

public static ImageAccess unwarp(ImageAccess input, double d) {

int nx = input.getWidth();

int ny = input.getHeight();

double m = Math.max(nx, ny);

double a = 4.0 \* (1.0 - d) / m;

double b = 2.0 \* d - 1.0;

double cx = (nx - 1) / 2.0;

double cy = (ny - 1) / 2.0;

ImageAccess coef = computeCubicSplineCoeffients(input);

ImageAccess output = new ImageAccess(nx, ny);

for (int xo = 0; xo < nx; xo++) {

for (int yo = 0; yo < ny; yo++) {

double dxp = xo - cx;

double dyp = yo - cy;

double rhoP = Math.hypot(dxp, dyp);

double rho;

if (rhoP == 0) {

rho = 0;

} else {

double discr = b\*b + 4\*a\*rhoP;

rho = (-b + Math.sqrt(discr)) / (2\*a);

}

double x = cx + dxp / (rhoP == 0 ? 1 : rhoP) \* rho;

double y = cy + dyp / (rhoP == 0 ? 1 : rhoP) \* rho;

double v = getInterpolatedPixelCubicSpline(coef, x, y);

output.putPixel(xo, yo, v);

}

}

return output;

}

Dados os parâmetros:

a = *(4(1 - d))/m*

b = *2d - 1*

c = *0*

Insira a imagem transformada:



Qual o melhor valor para d?

*1.18*

**Questão 3**

Código:

**public static String whatTime(ImageAccess input) {**

**int nx = input.getWidth(), ny = input.getHeight();**

**double cx = (nx - 1) / 2.0, cy = (ny - 1) / 2.0;**

**int R = (int)(Math.min(nx, ny) / 2.0);**

**// 1) obtém magnitude do gradiente (Sobel)**

**double[][] G = computeGradientMagnitude(input);**

**// 2) projetor de gradiente**

**int nAngles = 360;**

**double[] projection = new double[nAngles];**

**// Criar imagem polar: largura = 360 (ângulo), altura = R (raio)**

**ImageAccess polarImage = new ImageAccess(nAngles, R);**

**for (int t = 0; t < nAngles; t++) {**

**// converte t - ângulo de relógio (0 em 12h, CW)**

**double ang = Math.toRadians(90 - t);**

**double cosA = Math.cos(ang), sinA = Math.sin(ang);**

**double sum = 0;**

**for (int r = 0; r < R; r++) {**

**double x = cx + r \* cosA;**

**double y = cy - r \* sinA;**

**// Interpola valor do gradiente**

**double v = interpolate(G, x, y);**

**sum += v;**

**// Salva na imagem polar**

**polarImage.putPixel(t, r, v);**

**}**

**projection[t] = sum;**

**}**

**// 3) encontra os dois maiores picos em 'projection'**

**int idx1 = 0, idx2 = 0;**

**double p1 = -1, p2 = -1;**

**for (int t = 0; t < nAngles; t++) {**

**double v = projection[t];**

**if (v > p1) {**

**p2 = p1; idx2 = idx1;**

**p1 = v; idx1 = t;**

**} else if (v > p2) {**

**p2 = v; idx2 = t;**

**}**

**}**

**int minuteAngle = idx1, hourAngle = idx2;**

**// 4) converte em hora/minuto**

**int minute = (int)Math.round(minuteAngle \* 60.0 / 360.0) % 60;**

**int hour = (int)Math.round(hourAngle \* 12.0 / 360.0) % 12;**

**if (hour == 0) hour = 12;**

**// Mostra imagem polar**

**polarImage.show("Polar Projection");**

**String time = String.format("%02d:%02d", hour, minute);**

**IJ.write("Time: " + time);**

**return time;**

**}**

**private static double[][] computeGradientMagnitude(ImageAccess img) {**

**int nx = img.getWidth(), ny = img.getHeight();**

**double[][] G = new double[ny][nx];**

**for (int y = 1; y < ny-1; y++) {**

**for (int x = 1; x < nx-1; x++) {**

**// Sobel X**

**double gx =**

**-img.getPixel(x-1,y-1) + img.getPixel(x+1,y-1)**

**-2\*img.getPixel(x-1,y ) + 2\*img.getPixel(x+1,y )**

**-img.getPixel(x-1,y+1) + img.getPixel(x+1,y+1);**

**// Sobel Y**

**double gy =**

**-img.getPixel(x-1,y-1) -2\*img.getPixel(x,y-1) -img.getPixel(x+1,y-1)**

**+img.getPixel(x-1,y+1) +2\*img.getPixel(x,y+1) +img.getPixel(x+1,y+1);**

**G[y][x] = Math.hypot(gx, gy);**

**}**

**}**

**return G;**

**}**

**private static double interpolate(double[][] G, double xf, double yf) {**

**int x0 = (int)Math.floor(xf), y0 = (int)Math.floor(yf);**

**int x1 = x0+1, y1 = y0+1;**

**double a = xf - x0, b = yf - y0;**

**x0 = clamp(x0, 0, G[0].length-1);**

**x1 = clamp(x1, 0, G[0].length-1);**

**y0 = clamp(y0, 0, G.length-1);**

**y1 = clamp(y1, 0, G.length-1);**

**double v00 = G[y0][x0], v10 = G[y0][x1],**

**v01 = G[y1][x0], v11 = G[y1][x1];**

**return v00\*(1-a)\*(1-b) + v10\*(a)\*(1-b) + v01\*(1-a)\*(b) + v11\*(a)\*(b);**

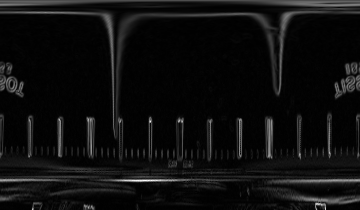
**}**

**private static int clamp(int v, int lo, int hi) {**

**return (v<lo?lo:(v>hi?hi:v));**

**}**

Insira a imagem em coordenadas polares:



Que horas são?

*09:19*

