**Roteiro**

**Interpolação e transformações geométricas**

**Notas de java**

**­­­­­­­­­­­­­­­­­­­­­­­**

|  |  |
| --- | --- |
| **Operação** | **Sintaxe** |
| Valor Absoluto | y = Math.abs(x); |
| Raiz quadrada | y = Math.sqrt(x); |
| Funções trigonométricas | y = Math.cos(a); y = Math.sin(a); a em radianos |
| Arctan [-π..π] | a = Math.atan2(dy, dx); a em radianos |
| Retorna o inteiro mais próximo de x | y = (int)Math.round(x); |
| Retorna o menor inteiro de x | y = (int)Math.ceil(x); |
| Retorna o maior inteiro de x | y = (int)Math.floor(x); |

**1. Interpolação**

Propomos aqui, comparar três métodos padrões de interpolação, o método de vizinhos próximos, o método bilinear, e o método spline bicúbica. Usamos estes interpoladores em um método de reajuste de tamanho das imagens disponível em resize(). Este método é chamado no plugin **Resize** o qual requer o método de interpolação e o fator de escala (maior que 1 para auumentar a imagem) como parâmetro.

Teste o método de interpolação com as imagens circle.tif e london.tif.

**1.1 Entendendo a interpolação linear**

O método usa as bases de funções B-spline de grau um. Leia e entenda o método getInterpolatedPixelLinear() qual retorna o valor do pixel na posição (x,y).

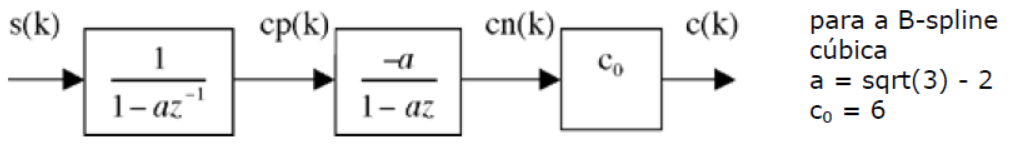
**1.2 Implementação do método de interpolação vizinhos mais próximo**

Simplifique o código deste método para retornar o valor do vizinho mais próximo na posição (x,y). Escreva o código no método getInterpolatedPixelNearestNeigbor().

**1.3 Entendendo o método de interpolação por spline cúbica**

**1.3.1. coeficientes B-spline**

Para interpolar através de spline bicúbica, devemos computar os valores dos coeficientes splines, devemos implementar um pré-filtro que computa os valores dos coeficientes c(k). Uma implementação rápida é obtida usando o uma cascata de filtros recursivos como esquematizado abaixo:



O método computeCubicSplineCoeffients() calcula os coeficientes de uma imagem de modo separável, e armazena os coeficientes em uma imagem de saída de modo separável em linhas e colunas da Imagem, de acordo com o método de Unser (ver artigo anexado) . Sua tarefa será primeiro entender o método 1D doSymmetricalExponentialFilter() que processa um filtro recursivo para computar os coeficientes Splines.

Primeiro, existe uma recursão para a implementação do filtro causal incrementando o índice de 1 até n-1; segundo, escreva a recursão para o filtro anticausal, decrementando o índice de n-2 até 0. O último passo é multiplicar por co.

Está disponível também o método que retorna os valores iniciais apropriados para condições de fronteiras reflexivas:

cp[0] = computeInitialValueCausal(s, a);,

cn[n-1] = computeInitialValueAntiCausal(cp, a);,

**1.3.2. Interpolação cúbica**

Assim como getInterpolatedPixelLinear(), escreva o método getInterpolatedPixelCubicSpline() o qual devolve o valor do pixel na posição (x,y) usando interpolação B-spline cúbica:

Somente 4\*4 pontos são necessários para calcular f(x,y), desde que *β3* tem valores não nulos apenas em [-2, 2].

Para computar o valor, é necessário:

• a vizinhança 4 x 4 de um ponto (i,j) que pode ser obtido com o método

getNeighborhood() de ImageAccess.

double arr[][] = new double[4][4];

coef.getNeighborhood(i, j, arr);

arr[0][0] <- coef(i-1,j-1)

arr[1][0] <- coef(i,j-1)

arr[1][1] <- coef(i,j) ...

• a base de funções B-spline que são dadas no método getCubicSpline() e computadas como segue (com 0 <= t <= 1):

double w[] = getCubicBSpline(t)

A saída é w[0] = β3(t+1), w[1] = β3(t), w[2] = β3(t-1), w[3] = β3(t-2);

**1.4. Comparação de interpoladores**

Compare a qualidade do resultado de cada iinterpolação após redução da imagem london.tif para [400, 250] pixels e então ampliada para o tamanho original.

• Vizinhos próximos

• Bilinear

• B-spline cúbica

Está disponível um plugin que mede a relação sinal ruído (SNR) entre a imagem de referência (london.tif) e a imagem de teste. Coloque os resultados no relatório.doc.

**Nota:** para usar a solução da implementação dos métodos acima, você pode usar a solução correspondente "solution" no seu código:

por ex. para usar o getInterpolatedPixelCubicSpline(image, x, y) pode fazer uma chamada a:

double v =

InterpolationSolution.getInterpolatedPixelCubicSpline(image, x,

y);

**2. Aplicação: desfazer uma deformação “fisheye” em uma foto**



A proposta é desenvolver um algoritmo simples para desfazer uma deformação geométrica do tipo “fisheye” (olho de peixe) produzido por uma lente com ângulo de abertura alto, e pode ser desfeito com a seguinte transformação geométrica:

ρ' = **T**(ρ) : ρ' = a.ρ2 + b.ρ + c

onde ρ é a distância de um pixel do centro da imagem original, e ρ' é a distância do centro na imagem transformada. São usados três Definimos três limites para identificar os três parâmetros

a, b, c:

• **T**(0) = 0, o centro não se move;

• **T**(m/2) = m/2, as imagens fonte e transformada são quadradas de tamanho

[m,m];

• **T**(m/4) = d\*m/4.

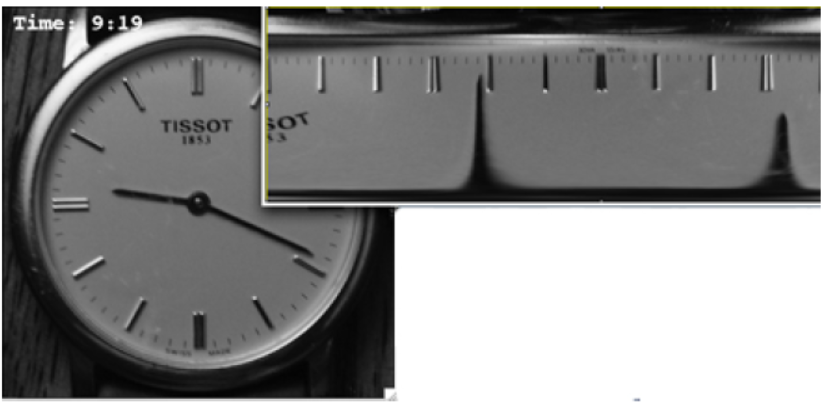
Aqui estamos usando apenas *d* como parâmetro livre para descrever a transformação. Dê os valores analíticos correspondentes de a, b, c.

Codifique o métdo unwrap() o qual retorna uma imagem com a deformação desfeita usando interpolação spline cúbica.

O plugin **RadialUnwrap** chama este método com valores para o parãmentro d dentro do intervalo [dmin, dmax] com *n* passos e produz uma pilha de imagens transformadas.

Aplique este algorítmo para produzir uma imagem plana, com uma linha reta no horizonte da imagem clock.tif. dê o melhor valor para d e insira a imagem resultante no relatório.

**3. Aplicação: Que horas são?**



O objetivo é determinar automaticamente a hora (hora +minutos) na imagem whattime.tif. Faremos as seguintes considerações: os ponteiros não são sobrepostos, o centro do relógio está no centro da imagem e o relógio está corretamente alinhado.

Codifique o método whatTime() o qual retorna o horário como uma string. O plugin **WhatTime** chama este método e mostra a string resultante sobre a imagem.

Podemos transformar a imagem em coordenadas polares usando um método de interpolação ótimo e fazer a soma dos valores de intensidades ao longo dos raios. Os 2 valores mínimos correspondem às posições angulares dos ponteiros. A partir disto podemos deduzir o número de minutos e horas. Complete o relatório.