Relazione: rotazione di un’immagine

# Premessa

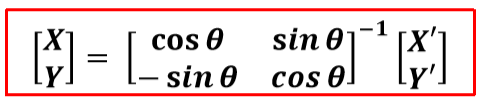
Il progetto prevede l’implementazione di classi il cui scopo è ruotare una data immagine ed eseguire una piccola animazione con essa, inoltre l’utilizzo del buffer multiplo ha permesso di evitare l’effetto “sfarfallio” dato dalla rotazione.

Contenuto:

* Classe Rotation: permette la rotazione dell’immagine;
* Classe RotationTest: classe di test che valuta il funzionamento di Rotation;
* Classe Animation: permette l’animazione dell’immagine utilizzando un thread parallelo a quello principale;
* Classe Input: permette l’invio di input da tastiera;
* Classe MultipleBuffer: permette l’utilizzo del buffer doppio nella rotazione

# Parte 1: rotazione

La rotazione avviene tramite l’utilizzo della classe Rotation, essa infatti esegue tutti i calcoli per calcolare la nuova posizione dei pixel in base al centro di rotazione e all’angolo dato, per farlo è stata utilizzata la seguente formula:



La formula sopra citata però non utilizza il centro di rotazione, quindi, dato che il raster parte da 0 come tutte le matrici, l’immagine verrà ruotata rispetto all’angolo in alto a sinistra, di conseguenza la suddetta formula è stata modificata secondo i requisiti richiesti utilizzando il seguente ragionamento: un primo tentativo è stato quello di aggiungere le coordinate del centro rispettivamente a x e y, il risultato ottenuto è un’immagine con l’angolo in alto a sinistra al centro:

|  |
| --- |
| x= x’ \* cos(tetha) + y’\*sin(theta) + centerX;  y= -x’ \* sin(tetha) + y’\*cos(theta) + centerY; |

Per centrare l’immagine, invece di utilizzare x’ e y’, si usa la differenza di questi valori con le rispettive coordinate del centro, queste differenze permettono di ottenere delle coordinate negative per i pixel che si trovano in alto a sinistra rispetto al centro, esse però verranno bilanciate da centerX e centerY e quindi un’immagine che ruota rispetto al centro dato, le formule diventano quindi le seguenti:

|  |
| --- |
| x= (x’ - centerX) \* cos(tetha) + (y’ - centerY) \* sin(theta) + centerX;  y= -(x’ - centerX) \* sin(tetha) + (y’ - centerY) \* cos(theta) + centerY; |

Dopo aver calcolato le coordinate x e y, si controlla se queste ultime siano effettivamente nel range dato e, se è così, si va a prendere il pixel (x,y) dall’immagine originale e lo si inserisce in quella nuova utilizzando x’ e y’ come coordinate, altrimenti si considera il pixel (x’-y’) come se fosse di colore nero, come si può vedere nel seguente pezzo di codice:

|  |
| --- |
| public void calculate()  {  double[][] matrix=invert(getMatrix());  for(int xi=0;xi<source.getWidth();xi++)  for(int yi=0;yi<source.getHeight();yi++)  {  int i=xi-centerX;  int j=yi-centerY;  int x= (int) (i\*matrix[0][0] + j\*matrix[1][0])+centerX;  int y= (int) (i\*matrix[0][1] + j\*matrix[1][1])+centerY;  int sample=0;  int bands=source.getRaster().getNumBands();  for(int b=0;b<bands;b++)  {  if(x<0 || x>=source.getWidth()) sample=0;  else if(y<0 || y>=source.getHeight()) sample=0;  else sample=source.getRaster().getSample(x, y, b);  result.setSample(xi, yi, b, sample);  }  }  } |

Come è possibile vedere dal codice sopra, la matrice viene calcolata utilizzando una funzione apposita e in seguito invertita, il primo metodo calcola i valori della matrice prendendo l’angolo, convertendolo in radianti ed calcolando coseni e seni:

|  |
| --- |
| private double[][] getMatrix()  {  double degree=Math.toRadians(tetha);  double[][] matrix=new double[2][2];  matrix[0][0]=Math.cos(degree);  matrix[0][1]=-Math.sin(degree);  matrix[1][0]=Math.sin(degree);  matrix[1][1]=Math.cos(degree);  return matrix;  } |

Il secondo metodo inverte la matrice precedentemente creata utilizzando il determinante (calcolato con getDet()), la matrice trasposta (matrice originale con righe e colonne scambiate) e il complemento algebrico, quest’ultimo indica il segno del valore (i,j) in base alla somma dei due, se è pari allora è positivo, altrimenti non lo è:

|  |
| --- |
| private double[][] invert(double[][] matrix)  {  double det=getDet(matrix);  double[][] trasp=traspose(matrix);  double[][] inverse=new double[2][2];  for(int i=0;i<2;i++)  for(int j=0;j<2;j++)  inverse[i][j]=Math.pow(-1, i+j)\*trasp[i][j]/det;  return trasp;  }    private double getDet(double[][] matrix)  {  double a=matrix[0][0];  double b=matrix[1][0];  double c=matrix[0][1];  double d=matrix[1][1];  return a\*d - b\*c;  }    private double[][] traspose(double[][] matrix)  {  double trasp[][]=new double[2][2];  for(int i=0;i<2;i++)  for(int j=0;j<2;j++)  trasp[i][j]=matrix[2-i-1][2-j-1];  return trasp;  } |

Il metodo setSourceData permette di inserire l’immagine su cui effettuare la rotazione, inoltre inizializza il buffer multiplo con il numero di buffer da utilizzare, dimensioni e tipo dell’immagine sorgente, quest’ultimo verrà spiegato nella parte finale della relazione.

|  |
| --- |
| public void setSourceData(Object src)  {  source=(BufferedImage) src;  result=new MultipleBuffer(2,source.getWidth(),source.getHeight(),source.getType());  } |

Tutte le altre funzioni setter delle classe servono per impostare centro e angolo di rotazione dell’immagine, il metodo getResults invece restituisce il risultato ottenuto dopo aver effettuato calculate.

Per testare il tutto, si utilizza la classe di test RotationTest, essa utilizza un’istanza di Rotation per effettuare una rotazione di 90 gradi e mostrare a schermo il risultato (utilizzando il metodo show), se tutto avviene correttamente, la foto è orizzontale.

|  |
| --- |
| void testCalculate()  {  BufferedImage src=null;  try  {  src=ImageIO.read(new File("piera-resized.png"));  }  catch (IOException e)  {  e.printStackTrace();  }  Rotation rotation=new Rotation();  rotation.setSourceData(src);  rotation.setAngle(90);  rotation.setCenter(src.getWidth()/2,src.getHeight()/2);  //rotation.setCenter(0,0);  rotation.setSourceData(src);  rotation.calculate();  BufferedImage result=(BufferedImage) rotation.getResults();  show(result);  try {  Thread.sleep(30000);  } catch (InterruptedException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } |

## Parte 2: animazione

L’animazione avviene grazie alla classe Animation, essa avviene su un thread parallelo a quello principale dato che viene implementata l’interfaccia Runnable, l’immagine continuerà a girare finchè non verrà premuto il tasto invio, facendo uscire il thread dal ciclo facendolo terminare di conseguenza (per farlo si utilizza setEnd() impostandolo a true).

|  |
| --- |
| public static void main(String[] args)  {  Animation animation=new Animation("piera-resized.png");  Thread thread=new Thread(animation);  thread.start();  Input.readString("Premere INVIO per finire");  animation.setEnd(true);  } |

Prima di effettuare l’animazione, bisogna innanzitutto aprire la finestra in cui si svolge la rotazione, per farlo si utilizza il metodo show, esso crea la finestra inserendo inizialmente l’immagine sorgente e restituendo la label, su quest’ultima verranno inseriti i risultati ottenuti dopo i calcoli.

|  |
| --- |
| private JLabel show(BufferedImage image) //crea la finestra dell'immagine data.  {  JLabel label= new JLabel(new ImageIcon(image));  JFrame f = new JFrame("Original picture");  f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  f.getContentPane().add(label);  f.pack();  f.setLocation(20,20);  f.setVisible(true);  return label;  } |

A una frequenza di 60 fotogrammi al secondo, l’angolo viene incrementato e con esso viene eseguita la rotazione dell’immagine originale, andando a inserire il risultato nella label, quello che si ottiene è una lieve rotazione della foto all’interno della finestra, dando una sensazione di movimento.

|  |
| --- |
| public void run()  {  BufferedImage result=null;  //rotation.setSourceData(source);  rotation.setCenter(source.getWidth()/2,source.getHeight()/2);  JLabel label=show(source);    while(!end) //ogni 167 millisecondi (ovvero 60fps), l'angolo viene incrementato  {  angle=(angle+1)%360;  rotation.setAngle(angle);  rotation.calculate();  result=(BufferedImage) rotation.getResults();  label.setIcon(new ImageIcon(result));  try  {  Thread.sleep(1000/60);  }  catch (InterruptedException e)  {  e.printStackTrace();  }  }  } |

## Parte 3: buffer multiplo

Il buffer multiplo viene implementato con la classe MultipleBuffer, esso permette di evitare lo sfarfallio dell’immagine durante lo spostamento o comunque di diminuire la sua visibilità, infatti l’effetto risulta ancora visibile quando l’incremento dell’angolo è piccolo, cosa che non succede con valori grandi di quest’ultimo. Nella rotazione, il buffer corrente viene ottenuto attraverso il metodo get, esso inoltre passa a successivo permettendo di effettuare su quest’ultimo i calcoli nell’iterazione successiva.

|  |
| --- |
| public BufferedImage get() //restituisce il buffer e passa a quello successivo  {  int index=current;  current=(current+1)%size;  return buffers[index];  } |

Il metodo setSample() della classe ha il compito di impostare il campione dato nella banda b del pixel (x-y) del buffer corrente, l’implementazione di questa funzione è dovuta al fatto che non è possibile accedere al raster all’esterno della classe stessa, la sua presenza infatti permette di semplificare il tutto.

|  |
| --- |
| public void setSample(int x,int y, int b,int s) //imposta il campione sul buffer corrente  {  buffers[current].getRaster().setSample(x, y, b, s);  } |