音声画像処理
レポート2

2012/2/14
095743H 長浜祐貴

目 次

1 A. 合成変換

適当な画像 Y を用意し、図 9.9 合成変換を実行するプログラムを作成しなさい

1.1 合成変換のソースコード

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class allMain extends Applet {
  private static final long serialVersionUID = 1L;
  Image img,img2;
  Image new_img,new_img2;

  int w_1 = 0;
  int h_1 = 0;
  int c_width = 0;//キャンパスの幅と高さ
  int c_height = 0;
  int c_axisx=0;//キャンパスの x 軸と y 軸の原点
  int c_axisy=0;

  int pix[];
  int campus[];//変換した画像を貼り付けるキャンパス

  public void init(){

    img = getImage( getCodeBase(),"test.jpg");
    MediaTracker mt = new MediaTracker(this);
    mt.addImage( img , 0 );

    try{
      mt.waitForID( 0 );
    }catch( InterruptedException e)  {
      System.out.println( e );
    }

    w_1 = img.getWidth(this);
    h_1 = img.getHeight(this);

    c_width = w_1*4;
    c_height = h_1*4;
    c_axisx = w_1*2;
    c_axisy = h_1*2;
    pix = new int[w_1*h_1];
    campus = new int[ c_width* c_height ];
    setPix();
```

```
}

void setPix(){
  try{
    PixelGrabber pg = new PixelGrabber( img, 0, 0, w_1, h_1, pix, 0, w_1 );
    pg.grabPixels();
  }catch( InterruptedException e ){
    System.out.println( e );
  }

  int startX = c_width / 2+50;//キャンパスに張り付ける x 軸の点
  int startY = c_height / 2;//キャンパスに張り付ける y 軸の点
  //画像をキャンパスに張り付ける。
  ImagePaste( campus, pix, c_width, w_1, h_1, startX, startY);
  //合成変換
  campus = composite( campus, c_width, c_height, Math.PI/4, true);

  setAxis( campus, c_width,c_height);//キャンパスの中央に縦横の軸を描画
  MemoryImageSource mimg = new MemoryImageSource( c_width,c_height, campus,0,c_width);
  System.out.println("end");
  new_img = createImage( mimg );
}

//四捨五入
public int round( double i ){
  if( i > 0 ){
    return (int) (i += 0.5);
  }else  {
    return (int) (i -= 0.5);
  }
}

//画面の中央に x 軸と y 軸を描画
public void setAxis(int[] dst, int dstX, int dstY){

  int axisX = dstX/2;
  int axisY = dstY/2;

  for( int i = 0 ; i < dstX; ++i ){
    dst[axisY*dstX+i] =  0xff << 24;
  }

  for( int i = 0 ; i < dstY; ++i ){
    dst[ i*dstX+axisX] =  0xff << 24;
  }
}
```

```
//src の画像を dst に張り付け
public void ImagePaste( int[] dst, int[] src,int dstX, int srcX, int srcY, int width, int height){

  for( int i = height ; i < height+srcY; ++i ){
    for( int j = width; j < width+srcX; ++j ){
      dst[i*dstX+j] = src[(i-height)*srcX+j-width];
    }
  }
}

//キャンパスの中央を原点として回転する。
public int[] rotation(int[] src ,int srcX, int srcY,double angle ){

  int[] temp = new int[ srcX*srcY];
  angle = -angle;//座標系の問題により入れ替える。
  for( int i = 0 ; i < srcY; ++i ){
    for( int j = 0 ; j < srcX; ++ j ){

      double newX = Math.cos( angle )*(j-srcX/2)-Math.sin( angle )*(i-srcY/2);
      double newY = Math.sin( angle )*(j-srcX/2)+Math.cos( angle )*(i-srcY/2);

      newX = round(newX);//四捨五入
      newY = round(newY);

      //位置を修正
      int x = (int)newX+srcX/2;
      int y = (int)newY+srcY/2;

      //画像内に収まっているならばデータを代入する。
      if( x > 0 && x < srcX && y > 0 && y < srcY ){
        temp[ y*srcX+x] = src[i*srcX+j];
      }
    }
  }
  return temp;
}

//鏡面反射画像を作成。元データは残さない。
public int[] reflection_y( int[] src, int srcX, int srcY ){

  System.out.println(srcX+","+srcY);
  int axisX = srcX/2;
  int[] reflection = new int[srcX*srcY];

  for( int i = 0; i < srcY; ++i ){
    for( int j = 0 ; j < srcX; ++j ){
      int newX = -1*(j -axisX)+axisX-1;
```

```
        reflection[ i *srcX + newX ] = src[i*srcX+j];
      }
    }
    return reflection;
  }

  //合成変換。ここでは回転と鏡面
  public int[]  composite(int[] src, int srcX,int srcY,double angle, boolean pattern){

    if(pattern){
      src = rotation( src, srcX, srcY, angle);
      src = reflection_y( src, srcX ,srcY );
    }else{
      src = reflection_y( src, srcX ,srcY );
      src = rotation( src, srcX, srcY, angle);
    }
    System.out.println("end composite");
    return src;
  }

  public int ByteRound( int i ){

    if( i > 255 ) i = 255;
    else if(i < 0 ) i = 0;
    return i;
  }

  public void paint( Graphics g ){
    g.drawImage( new_img, 0,0,this);
    System.out.println("print Image");
  }

}
```

1.2 実行結果

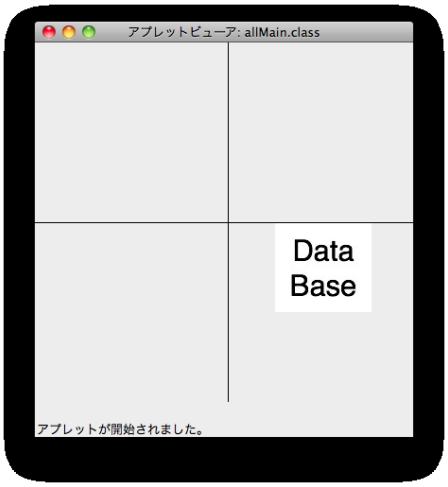

図 1: 合成変換前

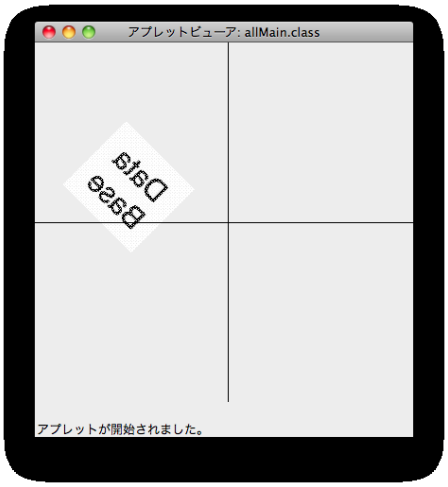

図 2: 合成変換後

2 B. アフィン変換、射影変換

P162 のアフィン変換と射影変換を実行するプログラムを作成し、両者の相違を考察しなさい

2.1 アフィン変換のソースコード

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class allMain extends Applet {
    private static final long serialVersionUID = 1L;
    Image img,img2;
    Image new_img,new_img2;

    int w_1 = 0;
    int h_1 = 0;
    int c_width = 0;//キャンパスの幅と高さ
    int c_height = 0;
    int c_axisx=0;//キャンパスの x 軸と y 軸の原点
    int c_axisy=0;

    int pix[];
    int campus[];//変換した画像を貼り付けるキャンパス

    public void init(){

        img = getImage( getCodeBase(),"test.jpg");
        MediaTracker mt = new MediaTracker(this);
        mt.addImage( img , 0 );

        try{
            mt.waitForID( 0 );
        }catch( InterruptedException e)    {
            System.out.println( e );
        }

        w_1 = img.getWidth(this);
        h_1 = img.getHeight(this);

        c_width = w_1*4;
        c_height = h_1*4;
        c_axisx = w_1*2;
        c_axisy = h_1*2;
        pix = new int[w_1*h_1];
        campus = new int[ c_width* c_height ];
        setPix();
```

```
}

void setPix(){
    try{
        PixelGrabber pg = new PixelGrabber( img, 0, 0, w_1, h_1, pix, 0, w_1 );
        pg.grabPixels();
    }catch( InterruptedException e ){
        System.out.println( e );
    }

    int startX = c_width / 2+50;//キャンパスに張り付ける x 軸の点
    int startY = c_height / 2;//キャンパスに張り付ける y 軸の点
    //画像をキャンパスに張り付ける。
    ImagePaste( campus, pix, c_width, w_1, h_1, startX, startY);

    campus = affine( campus,c_width,c_height,20,20,Math.PI/4,2);//アフィン変換

    setAxis( campus, c_width,c_height);//キャンパスの中央に縦横の軸を描画
    MemoryImageSource mimg = new MemoryImageSource( c_width,c_height, campus,0,c_width);
    System.out.println("end");
    new_img = createImage( mimg );
}

//四捨五入
public int round( double i ){
    if( i > 0 ){
        return (int) (i += 0.5);
    }else    {
        return (int) (i -= 0.5);
    }
}

//画面の中央に x 軸と y 軸を描画
public void setAxis(int[] dst, int dstX, int dstY){

    int axisX = dstX/2;
    int axisY = dstY/2;

    for( int i = 0 ; i < dstX; ++i ){
        dst[axisY*dstX+i] =  0xff << 24;
    }

    for( int i = 0 ; i < dstY; ++i ){
        dst[ i*dstX+axisX] =  0xff << 24;
    }
}
```

```
//src の画像を dst に張り付け
public void ImagePaste( int[] dst, int[] src,int dstX, int srcX, int srcY, int width, int height){

    for( int i = height ; i < height+srcY; ++i ){
        for( int j = width; j < width+srcX; ++j ){
            dst[i*dstX+j] = src[(i-height)*srcX+j-width];
        }
    }
}

//キャンパスの中央を原点として回転する。
public int[] rotation(int[] src ,int srcX, int srcY,double angle ){

    int[] temp = new int[ srcX*srcY];
    angle = -angle;//座標系の問題により入れ替える。
    for( int i = 0 ; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++ j ){

            double newX = Math.cos( angle )*(j-srcX/2)-Math.sin( angle )*(i-srcY/2);
            double newY = Math.sin( angle )*(j-srcX/2)+Math.cos( angle )*(i-srcY/2);

            newX = round(newX);//四捨五入
            newY = round(newY);

            //位置を修正
            int x = (int)newX+srcX/2;
            int y = (int)newY+srcY/2;

            //画像内に収まっているならばデータを代入する。
            if( x > 0 && x < srcX && y > 0 && y < srcY ){
                temp[ y*srcX+x] = src[i*srcX+j];
            }
        }
    }
    return temp;
}

//鏡面反射画像を作成。元データは残さない。
public int[] reflection_y( int[] src, int srcX, int srcY ){

    System.out.println(srcX+","+srcY);
    int axisX = srcX/2;
    int[] reflection = new int[srcX*srcY];

    for( int i = 0; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++j ){
            int newX = -1*(j -axisX)+axisX-1;
```

```
            reflection[ i *srcX + newX ] = src[i*srcX+j];
        }
    }
    return reflection;
}

//アフィン変換を行う。線形変換は pattern に入れた値で決める。
//アフィン変換は線形変換と平行移動を組み合わせたものなので、線形変換を行った後に平行移動を行う形にし、
//メソッドの再利用を行う。
public int[] affine(int[] src, int srcX, int srcY,int offsetX, int offsetY, double angle,int pattern
    //pattern に入れた値により、実行する線形変換を変更する。
    switch(pattern){
        case 1://回転
            src = rotation( src, srcX, srcY, angle); break;
        case 2://鏡面
            src = reflection_y(src, srcX, srcY); break;
        case 3://せん断 (x 軸)
            src = skewing_x( src, srcX, srcY, angle);break;
        case 4:
            src = skewing_y( src, srcX, srcY, angle);break;
        default:
    }

    int[] temp = new int[srcX * srcY];

    //平行移動
    for( int i = 0 ; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++j ){
            if( j + offsetX < srcX && i+offsetY < srcY ){
                temp[ (i+offsetY)*srcX + (j+offsetY)] = src[ i*srcX+j];
            }
        }
    }
    return src;
}

//x 方向のせん断
public int[] skewing_x(int[] src, int srcX,int srcY, double radian_x ){
    int[] temp = new int[ srcX*srcY ];
    for( int i = 0 ; i < srcY; ++i ){
        for(int j = 0 ; j < srcX; ++j ){
            int newX = (int)(j+Math.tan( radian_x )*i);
            int newY = i;
            if( newX > 0 && newX < srcX && newY > 0 && newY < srcY ){
                temp[newY*srcX+newX] = src[ i*srcX+j];
            }
        }
```

```
    }
    return temp;
}

//スキュー (せん断)。
public int[] skewing_y(int[] src, int srcX,int srcY, double radian_y ){

    int[] temp = new int[ srcX*srcY ];
    for( int i = 0 ; i < srcY; ++i ){

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

            int newX = j;
            int newY = (int) ( Math.tan(radian_y)*j+i);

            if( newX > 0 && newX < srcX && newY > 0 && newY < srcY ){
                temp[newY*srcX+newX] = src[ i*srcX+j];
            }
        }
    }
    return temp;
}

//ピボットが 0 の場合を避けるためにピボット選択をする。
void pivot(int i,double mat[][],int N){

    int pivot_row = i;//まずは一番初めの行を最大とする。
    double max_col = mat[i][i];
    if( max_col != 0 ) return;//ピボットが 0 でないなら終わり。
    //i 行目から最後の行までで注目列の列の値が最大のものを求める。
    for( int j =i; j < N; ++j ){
        if( mat[j][i] != 0 ){
            pivot_row = j;
            max_col = mat[j][i];
            break;
        }
    }

    //最大の行が入れ替わっていたら行を入れ替える
    if( pivot_row != i ){
        for( int j = 0 ; j < N+N; ++j ){
            double swap = mat[i][j];
            mat[i][j] = mat[pivot_row][j];
            mat[pivot_row][j] = swap;
        }
    }
```

```
    }

    //掃出し法を実行
    void gauss(int i,double a[][],int N){

        double pivot = a[i][i];//ピボットを求めたので、その値を代入
        for(int j = 0 ; j < N+N; ++j ){
            //ピボットに選択した値のある行をピボットで割る。
            a[i][j] /= pivot;
        }

        //各行に対して計算を施す。
        for( int j = 0 ; j <N; ++j ){
            if( j != i ){
                double col_i = a[j][i];
                for( int k = i ; k < N+N; ++k )    {
                    a[j][k] -= col_i*a[i][k];
                }
            }
        }
    }

    //逆行列を求める演算
    void mat_inv(double a[][],int N){
        for(int i=0; i<N; ++i){
            pivot(i,a,N);//ピボットを 0 以外に調整する。
            gauss(i,a,N);
        }
    }



    public int ByteRound( int i ){
        if( i > 255 ) i = 255;
        else if(i < 0 ) i = 0;
        return i;
    }



    public void paint( Graphics g ){
        g.drawImage( new_img, 0,0,this);
        System.out.println("print Image");
    }
}
```

2.2 実行結果

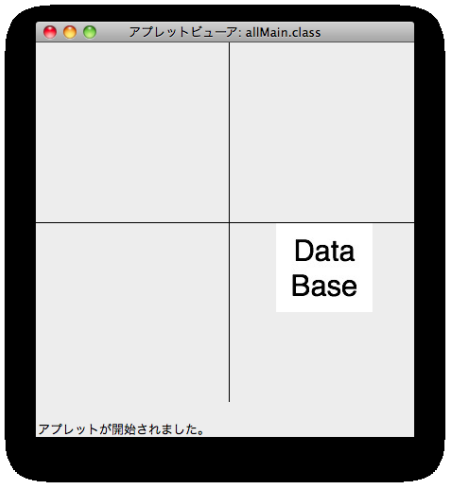

図 3: アフィン変換前

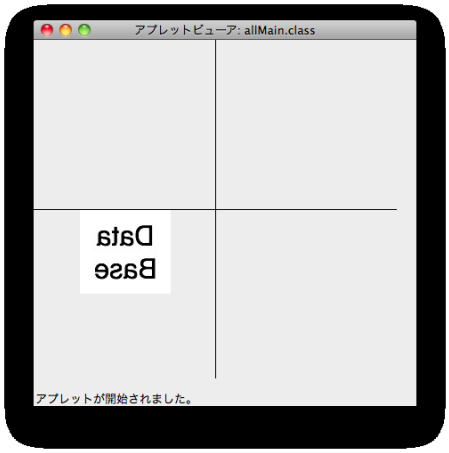

図 4: アフィン変換後

2.3 射影変換のソースコード

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class allMain extends Applet{

    private static final long serialVersionUID = 1L;
    Image img,img2;
    Image new_img,new_img2;

    int w_1 = 0;
    int h_1 = 0;
    int c_width = 0;//キャンパスの幅と高さ
    int c_height = 0;
    int c_axisx=0;//キャンパスの x 軸と y 軸の原点
    int c_axisy=0;

    int pix[];
    int campus[];//変換した画像を貼り付けるキャンパス


    public void init(){
        img = getImage( getCodeBase(),"test.jpg");

        MediaTracker mt = new MediaTracker( this );
        mt.addImage( img , 0 );

        try{
            mt.waitForID( 0 );
        }catch( InterruptedException e){
            System.out.println( e );
        }

        w_1 = img.getWidth( this );
        h_1 = img.getHeight( this );

        c_width = w_1*4;
        c_height = h_1*4;
        c_axisx = w_1*2;
        c_axisy = h_1*2;
        pix = new int[w_1*h_1];
        campus = new int[ c_width* c_height ];
        setPix();
    }//end init
```

```
void setPix()
{
    try
    {
        PixelGrabber pg = new PixelGrabber( img, 0, 0, w_1, h_1, pix, 0, w_1 );
        pg.grabPixels();
    }catch( InterruptedException e )
    {
        System.out.println( e );
    }//end try-catch

    int startX = c_width / 2+50;//キャンパスに張り付ける x 軸の点
    int startY = c_height / 2;//キャンパスに張り付ける y 軸の点

    ImagePaste( campus, pix, c_width, w_1, h_1, startX, startY);//画像をキャンパスに張り付ける。

    //射影変換
    campus = projective( campus , c_width, c_height, 0, 0, c_width-100,60, 60, c_height-130, c_width,
    setAxis( campus, c_width,c_height);//キャンパスの中央に縦横の軸を描画する
    MemoryImageSource mimg = new MemoryImageSource( c_width,c_height, campus,0,c_width);

    new_img = createImage( mimg );
}

//四捨五入
public int round( double i ){
    if( i > 0 ){
        return (int) (i += 0.5);
    }else{
        return (int) (i -= 0.5);
    }
}

//画面の中央を縦横に横切る軸 (数学の関数の座標系みたいなもの) を描画する。
public void setAxis(int[] dst, int dstX, int dstY){

    int axisX = dstX/2;
    int axisY = dstY/2;

    for( int i = 0 ; i < dstX; ++i ){
        dst[axisY*dstX+i] =  0xff << 24;
    }

    for( int i = 0 ; i < dstY; ++i ){
        dst[ i*dstX+axisX] =  0xff << 24;
    }
}
```

```
//src の画像を dst に張り付ける。
public void ImagePaste( int[] dst, int[] src,int dstX, int srcX, int srcY, int width, int height )
{
    for( int i = height ; i < height+srcY; ++i )
    {
        for( int j = width; j < width+srcX; ++j )
        {
            dst[i*dstX+j] = src[(i-height)*srcX+j-width];
        }
    }
}//end ImagePaste

//画像の回転。キャンパスの中央を原点として回転する。
public int[] rotation(int[] src ,int srcX, int srcY,double angle ){

    int[] temp = new int[ srcX*srcY];
    angle = -angle;//座標系の問題により入れ替える。
    for( int i = 0 ; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++ j ){

            //画像の中央を原点とする。
            double newX = Math.cos( angle )*(j-srcX/2)-Math.sin( angle )*(i-srcY/2);
            double newY = Math.sin( angle )*(j-srcX/2)+Math.cos( angle )*(i-srcY/2);

            newX = round( newX );//四捨五入
            newY = round( newY );
            //位置を修正
            int x = (int)newX+srcX/2;
            int y = (int)newY+srcY/2;

            //画像内に収まっているならばデータを代入する。
            if( x > 0 && x < srcX && y > 0 && y < srcY ){
                temp[ y*srcX+x] = src[i*srcX+j];
            }
        }
    }
    return temp;
}

//鏡面反射画像を作成。元データは残さない。
public int[] reflection_y( int[] src, int srcX, int srcY ){

    int axisX = srcX/2;
    int[] reflection = new int[srcX*srcY];
    for( int i = 0; i < srcY; ++i )    {
        for( int j = 0 ; j < srcX; ++j ){
```

```
            int newX = -1*(j -axisX)+axisX-1;
            reflection[ i *srcX + newX ] = src[i*srcX+j];
        }
    }
    return reflection;
}

//ピボットが 0 の場合を避けるためにピボット選択をする。
void pivot(int i,double mat[][],int N){
    int pivot_row = i;//まずは一番初めの行を最大とする。
    double max_col = mat[i][i];
    if( max_col != 0 ) return;//ピボットが 0 でないなら終わり。
    //i 行目から最後の行までで注目列の列の値が最大のものを求める。
    for( int j =i; j < N; ++j ){
        if( mat[j][i] != 0 ){
            pivot_row = j;
            max_col = mat[j][i];
            break;
        }
    }

    //最大の行が入れ替わっていたら行を入れ替える
    if( pivot_row != i ){
        for( int j = 0 ; j < N+N; ++j ){
            double swap = mat[i][j];
            mat[i][j] = mat[pivot_row][j];
            mat[pivot_row][j] = swap;
        }
    }
}

//掃出し法を実行
void gauss(int i,double a[][],int N){
    double pivot = a[i][i];//ピボットを求めたので、その値を代入
    for(int j = 0 ; j < N+N; ++j ){
        //ピボットに選択した値のある行をピボットで割る。
        a[i][j] /= pivot;
    }

    //各行に対して計算を施す。
    for( int j = 0 ; j <N; ++j ){
        if( j != i ){
            double col_i = a[j][i];
            for( int k = i ; k < N+N; ++k )    {
                a[j][k] -= col_i*a[i][k];
            }
        }
```

```
    }
}

//逆行列を求める演算
void mat_inv(double a[][],int N){
    for(int i=0; i<N; ++i){
        pivot(i,a,N);//ピボットを 0 以外に調整する。
        gauss(i,a,N);
    }
}
//射影変換を行うプログラム
public int[] projective( int[] src,int srcX, int srcY, int x1, int y1, int x2, int y2, int x3, int y3
    //行列を作成。
    double[][]a = new double[8][16];//掃出し法で逆行列を求めるので、列の数は 2 倍
    a[0][0] = 0; a[0][1] = 0; a[0][2] = 0; a[0][3] = 0;
    a[0][4] = 1; a[0][5] = 0; a[0][6] = 0; a[0][7] = 0;
    a[1][0] = 0; a[1][1] = 0; a[1][2] = 0; a[1][3] = 0;
    a[1][4] = 0; a[1][5] = 1; a[1][6] = 0; a[1][7] = 0;
    a[2][0] = srcX; a[2][1] = 0; a[2][2] = 0; a[2][3] = 0;
    a[2][4] = 1; a[2][5] = 0; a[2][6] = -srcX*x2; a[2][7] = 0;
    a[3][0] = 0; a[3][1] = 0; a[3][2] = srcX; a[3][3] = 0;
    a[3][4] = 0; a[3][5] = 1; a[3][6] = -srcX*y2;a[3][7] = 0;
    a[4][0] = 0; a[4][1] = srcY; a[4][2] = 0; a[4][3] = 0;
    a[4][4] = 1; a[4][5] = 0; a[4][6] =0; a[4][7] = -srcY*x3;
    a[5][0] = 0; a[5][1] = 0; a[5][2] = 0; a[5][3] = srcY;
    a[5][4] = 0; a[5][5] = 1; a[5][6] = 0; a[5][7] = -srcY*y3;
    a[6][0] = srcX; a[6][1] = srcY; a[6][2] = 0; a[6][3] = 0;
    a[6][4] = 1; a[6][5] = 0; a[6][6] = -srcX*x4; a[6][7] = -srcY*x4;
    a[7][0] = 0; a[7][1] = 0; a[7][2] = srcX; a[7][3] = srcY;
    a[7][4] = 0; a[7][5] = 1; a[7][6] = -srcX*y4; a[7][7] = -srcY*y4;
    a[0][8] = 1.0; a[1][9] = 1.0;a[2][10] = 1.0;a[3][11] = 1.0;
    a[4][12] = 1.0; a[5][13] = 1.0; a[6][14] = 1.0; a[7][15] = 1.0;
    double[] p = new double[8];

    mat_inv(a,8);
    for( int i = 0 ; i < 8; ++i ){
        p[i] = a[i][8]*x1+a[i][9]*y1+a[i][10]*x2+a[i][11]*y2+
        a[i][12]*x3+a[i][13]*y3+a[i][14]*x4+a[i][15]*y4;
    }

    int[] temp = new int[ srcX* srcY];
    for( int i = 0 ; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++j ){
            double X = (p[0]*j+p[1]*i+p[4])/(p[6]*j+p[7]*i+1);
            double Y = (p[2]*j+p[3]*i+p[5])/(p[6]*j+p[7]*i+1);
            if( X > 0 && X < srcX && Y > 0 && Y < srcY ){
                temp[ (int)Y * srcX + (int)X ] = src[ i * srcX + j ];
```

```
                }
            }
        }

        System.out.println("end projective");
        return temp;
    }

    public int ByteRound( int i ){
        if( i > 255 ) i = 255;
        else if(i < 0 ) i = 0;
        return i;
    }

    public void paint( Graphics g ){
        g.drawImage( new_img, 0,0,this);
    }
}
```

2.4 実行結果

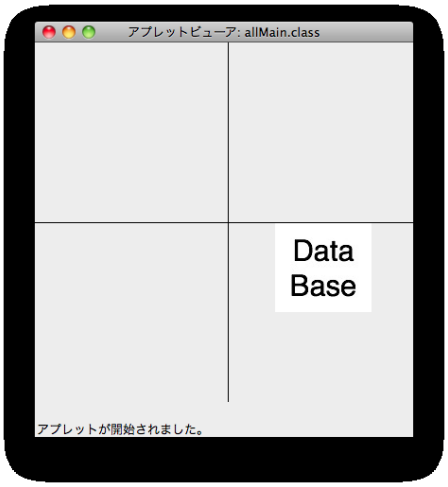

図 5: 射影変換前

図 6: 射影変換後

3　C. 各種法（補間法）

適当な画像を図 9.19 のように nearest neighbor 法と bilinear 法、bicubic 法で拡大表示するプログラムを作成し、各種法（補間法）の相違を考察しなさい。

3.1　nearest neighbor 法

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class allMain extends Applet {
    private static final long serialVersionUID = 1L;
    Image img,img2;
    Image new_img,new_img2;

    int w_1 = 0;
    int h_1 = 0;
    int c_width = 0;//キャンパスの幅と高さ
    int c_height = 0;
    int c_axisx=0;//キャンパスの x 軸と y 軸の原点
    int c_axisy=0;

    int pix[];
    int campus[];//変換した画像を貼り付けるキャンパス

    public void init(){

        img = getImage( getCodeBase(),"test.jpg");
        MediaTracker mt = new MediaTracker(this);
        mt.addImage( img , 0 );

        try{
            mt.waitForID( 0 );
        }catch( InterruptedException e)    {
            System.out.println( e );
        }

        w_1 = img.getWidth(this);
        h_1 = img.getHeight(this);

        c_width = w_1*4;
        c_height = h_1*4;
        c_axisx = w_1*2;
        c_axisy = h_1*2;
        pix = new int[w_1*h_1];
        campus = new int[ c_width* c_height ];
```

```
        setPix();
    }

    void setPix(){
        try    {
            PixelGrabber pg = new PixelGrabber( img, 0, 0, w_1, h_1, pix, 0, w_1 );
            pg.grabPixels();
        }catch( InterruptedException e ){
            System.out.println( e );
        }

        int startX = c_width / 2+50;//キャンパスに張り付ける x 軸の点
        int startY = c_height / 2;//キャンパスに張り付ける y 軸の点

        ImagePaste( campus, pix, c_width, w_1, h_1, startX, startY);//画像をキャンパスに張り付ける。


        double scale =8.0;//補間法を使う場合の画像の拡大率
        int[] campus = nearest_neighbor( pix, w_1, h_1, scale);//ニアレストネイバー法

        MemoryImageSource mimg = new MemoryImageSource( (int) (w_1*scale),(int)(h_1*scale), campus,0,(int
        System.out.println("end");
        new_img = createImage( mimg );
    }

    //src の画像を dst に張り付ける。
    public void ImagePaste( int[] dst, int[] src,int dstX, int srcX, int srcY, int width, int height )
    {
        for( int i = height ; i < height+srcY; ++i ){
            for( int j = width; j < width+srcX; ++j ){
                dst[i*dstX+j] = src[(i-height)*srcX+j-width];
            }
        }
    }

    //x 方向のせん断
    public int[] skewing_x(int[] src, int srcX,int srcY, double radian_x ){
        int[] temp = new int[ srcX*srcY ];
        for( int i = 0 ; i < srcY; ++i ){
            for(int j = 0 ; j < srcX; ++j ){
                int newX = (int)(j+Math.tan( radian_x )*i);
                int newY = i;
                if( newX > 0 && newX < srcX && newY > 0 && newY < srcY ){
                    temp[newY*srcX+newX] = src[ i*srcX+j];
                }
            }
        }
```

```
        System.out.println("end skewing_x");
        return temp;
    }

    //スキュー (せん断)。
    public int[] skewing_y(int[] src, int srcX,int srcY, double radian_y ){
        int[] temp = new int[ srcX*srcY ];
        for( int i = 0 ; i < srcY; ++i ){
            for(int j = 0 ; j < srcX; ++j ){
                int newX = j;//(int)(j+Math.tan( radian_x )*i);
                int newY = (int) ( Math.tan(radian_y)*j+i);
                if( newX > 0 && newX < srcX && newY > 0 && newY < srcY ){
                    temp[newY*srcX+newX] = src[ i*srcX+j];
                }
            }
        }
        System.out.println("end skewing_y");
        return temp;
    }


    //ニアレストネイバー法
    public int[] nearest_neighbor( int[] src, int srcX, int srcY, double scale ){
        int dstX = (int)(srcX*scale);
        int dstY = (int)(srcY*scale);
        int[] data = new int[ dstX*dstY];
        for( int i = 0; i < dstY; ++i ){
            int pixelY = (int) Math.round( (double)i/ scale );
            for(int j = 0 ;j < dstX; ++j ){
                int pixelX = (int) Math.round( (double)j / scale );
                int pix;
                if((pix = ( pixelY *srcX + pixelX) )>= src.length ){
                    pix = src.length-1;
                }
                data[i*dstX+j] = src[ pix ];
            }
        }
        System.out.println("end nearest_neighbor");
        return data;
    }


    public void paint( Graphics g ){
        g.drawImage( new_img, 0,0,this);
        System.out.println("print Image");
    }
}
```

```
void setPix(){
    try{
        PixelGrabber pg = new PixelGrabber( img, 0, 0, w_1, h_1, pix, 0, w_1 );
        pg.grabPixels();
    }catch( InterruptedException e ){
        System.out.println( e );
    }

    int startX = c_width / 2+50;//キャンパスに張り付ける x 軸の点
    int startY = c_height / 2;//キャンパスに張り付ける y 軸の点
    //画像をキャンパスに張り付ける。
    ImagePaste( campus, pix, c_width, w_1, h_1, startX, startY);

    campus = affine( campus,c_width,c_height,20,20,Math.PI/4,2);//アフィン変換

    setAxis( campus, c_width,c_height);//キャンパスの中央に縦横の軸を描画
    MemoryImageSource mimg = new MemoryImageSource( c_width,c_height, campus,0,c_width);
    System.out.println("end");
    new_img = createImage( mimg );
}

//四捨五入
public int round( double i ){
    if( i > 0 ){
        return (int) (i += 0.5);
    }else    {
        return (int) (i -= 0.5);
    }
}

//画面の中央に x 軸と y 軸を描画
public void setAxis(int[] dst, int dstX, int dstY){

    int axisX = dstX/2;
    int axisY = dstY/2;

    for( int i = 0 ; i < dstX; ++i ){
        dst[axisY*dstX+i] =  0xff << 24;
    }

    for( int i = 0 ; i < dstY; ++i ){
        dst[ i*dstX+axisX] =  0xff << 24;
    }
}

//src の画像を dst に張り付け
```

```
public void ImagePaste( int[] dst, int[] src,int dstX, int srcX, int srcY, int width, int height){

    for( int i = height ; i < height+srcY; ++i ){
        for( int j = width; j < width+srcX; ++j ){
            dst[i*dstX+j] = src[(i-height)*srcX+j-width];
        }
    }
}

//キャンパスの中央を原点として回転する。
public int[] rotation(int[] src ,int srcX, int srcY,double angle ){

    int[] temp = new int[ srcX*srcY];
    angle = -angle;//座標系の問題により入れ替える。
    for( int i = 0 ; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++ j ){

            double newX = Math.cos( angle )*(j-srcX/2)-Math.sin( angle )*(i-srcY/2);
            double newY = Math.sin( angle )*(j-srcX/2)+Math.cos( angle )*(i-srcY/2);

            newX = round(newX);//四捨五入
            newY = round(newY);

            //位置を修正
            int x = (int)newX+srcX/2;
            int y = (int)newY+srcY/2;

            //画像内に収まっているならばデータを代入する。
            if( x > 0 && x < srcX && y > 0 && y < srcY ){
                temp[ y*srcX+x] = src[i*srcX+j];
            }
        }
    }
    return temp;
}

//鏡面反射画像を作成。元データは残さない。
public int[] reflection_y( int[] src, int srcX, int srcY ){

    System.out.println(srcX+","+srcY);
    int axisX = srcX/2;
    int[] reflection = new int[srcX*srcY];

    for( int i = 0; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++j ){
            int newX = -1*(j -axisX)+axisX-1;
            reflection[ i *srcX + newX ] = src[i*srcX+j];
```

```
        }
    }
    return reflection;
}

//アフィン変換を行う。線形変換は pattern に入れた値で決める。
//アフィン変換は線形変換と平行移動を組み合わせたものなので、線形変換を行った後に平行移動を行う形にし、
//メソッドの再利用を行う。
public int[] affine(int[] src, int srcX, int srcY,int offsetX, int offsetY, double angle,int pattern
    //pattern に入れた値により、実行する線形変換を変更する。
    switch(pattern){
        case 1://回転
            src = rotation( src, srcX, srcY, angle); break;
        case 2://鏡面
            src = reflection_y(src, srcX, srcY); break;
        case 3://せん断 (x 軸)
            src = skewing_x( src, srcX, srcY, angle);break;
        case 4:
            src = skewing_y( src, srcX, srcY, angle);break;
        default:
    }

    int[] temp = new int[srcX * srcY];

    //平行移動
    for( int i = 0 ; i < srcY; ++i ){
        for( int j = 0 ; j < srcX; ++j ){
            if( j + offsetX < srcX && i+offsetY < srcY ){
                temp[ (i+offsetY)*srcX + (j+offsetY)] = src[ i*srcX+j];
            }
        }
    }
    return src;
}

//x 方向のせん断
public int[] skewing_x(int[] src, int srcX,int srcY, double radian_x ){
    int[] temp = new int[ srcX*srcY ];
    for( int i = 0 ; i < srcY; ++i ){
        for(int j = 0 ; j < srcX; ++j ){
            int newX = (int)(j+Math.tan( radian_x )*i);
            int newY = i;
            if( newX > 0 && newX < srcX && newY > 0 && newY < srcY ){
                temp[newY*srcX+newX] = src[ i*srcX+j];
            }
        }
    }
```

```
    return temp;
}

//スキュー (せん断)。
public int[] skewing_y(int[] src, int srcX,int srcY, double radian_y ){

    int[] temp = new int[ srcX*srcY ];
    for( int i = 0 ; i < srcY; ++i ){

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

            int newX = j;
            int newY = (int) ( Math.tan(radian_y)*j+i);

            if( newX > 0 && newX < srcX && newY > 0 && newY < srcY ){
                temp[newY*srcX+newX] = src[ i*srcX+j];
            }
        }
    }
    return temp;
}

//ピボットが 0 の場合を避けるためにピボット選択をする。
void pivot(int i,double mat[][],int N){

    int pivot_row = i;//まずは一番初めの行を最大とする。
    double max_col = mat[i][i];
    if( max_col != 0 ) return;//ピボットが 0 でないなら終わり。
    //i 行目から最後の行までで注目列の列の値が最大のものを求める。
    for( int j =i; j < N; ++j ){
        if( mat[j][i] != 0 ){
            pivot_row = j;
            max_col = mat[j][i];
            break;
        }
    }

    //最大の行が入れ替わっていたら行を入れ替える
    if( pivot_row != i ){
        for( int j = 0 ; j < N+N; ++j ){
            double swap = mat[i][j];
            mat[i][j] = mat[pivot_row][j];
            mat[pivot_row][j] = swap;
        }
    }

}
```

```
    //掃出し法を実行
    void gauss(int i,double a[][],int N){

        double pivot = a[i][i];//ピボットを求めたので、その値を代入
        for(int j = 0 ; j < N+N; ++j ){
            //ピボットに選択した値のある行をピボットで割る。
            a[i][j] /= pivot;
        }

        //各行に対して計算を施す。
        for( int j = 0 ; j <N; ++j ){
            if( j != i ){
                double col_i = a[j][i];
                for( int k = i ; k < N+N; ++k )    {
                    a[j][k] -= col_i*a[i][k];
                }
            }
        }
    }

    //逆行列を求める演算
    void mat_inv(double a[][],int N){
        for(int i=0; i<N; ++i){
            pivot(i,a,N);//ピボットを 0 以外に調整する。
            gauss(i,a,N);
        }
    }



    public int ByteRound( int i ){
        if( i > 255 ) i = 255;
        else if(i < 0 ) i = 0;
        return i;
    }



    public void paint( Graphics g ){
        g.drawImage( new_img, 0,0,this);
        System.out.println("print Image");
    }
}
```

3.2　実行結果

図 7: ニアストレイバー法での拡大

3.3 bilinear 法

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class allMain extends Applet{

    private static final long serialVersionUID = 1L;
    Image img,img2;
    Image new_img,new_img2;

    int w_1 = 0;
    int h_1 = 0;
    int c_width = 0;//キャンパスの幅と高さ
    int c_height = 0;
    int c_axisx=0;//キャンパスの x 軸と y 軸の原点
    int c_axisy=0;

    int pix[];
    int campus[];//変換した画像を貼り付けるキャンパス


    public void init(){
        img = getImage( getCodeBase(),"test.jpg");

        MediaTracker mt = new MediaTracker( this );
        mt.addImage( img , 0 );

        try{
            mt.waitForID( 0 );
        }catch( InterruptedException e){
            System.out.println( e );
        }

        w_1 = img.getWidth( this );
        h_1 = img.getHeight( this );

        c_width = w_1*4;
        c_height = h_1*4;
        c_axisx = w_1*2;
        c_axisy = h_1*2;
        pix = new int[w_1*h_1];
        campus = new int[ c_width* c_height ];
        setPix();
    }//end init
```

```
void setPix(){
    try    {
        PixelGrabber pg = new PixelGrabber( img, 0, 0, w_1, h_1, pix, 0, w_1 );
        pg.grabPixels();
    }catch( InterruptedException e ){
        System.out.println( e );
    }

    int startX = c_width / 2+50;//キャンパスに張り付ける x 軸の点
    int startY = c_height / 2;//キャンパスに張り付ける y 軸の点

    ImagePaste( campus, pix, c_width, w_1, h_1, startX, startY);//画像をキャンパスに張り付ける。


    double scale =8.0;//補間法を使う場合の画像の拡大率
    int[] campus = bilinear( pix, w_1, h_1, scale);//バイリニア法

    MemoryImageSource mimg = new MemoryImageSource( (int) (w_1*scale),(int)(h_1*scale), campus,0,(int
    System.out.println("end");
    new_img = createImage( mimg );
}

//src の画像を dst に張り付ける。
public void ImagePaste( int[] dst, int[] src,int dstX, int srcX, int srcY, int width, int height ){
    for( int i = height ; i < height+srcY; ++i ){
        for( int j = width; j < width+srcX; ++j ){
            dst[i*dstX+j] = src[(i-height)*srcX+j-width];
        }
    }
}

public int ByteRound( int i ){
    if( i > 255 ) i = 255;
    else if(i < 0 ) i = 0;
    return i;
}

//バイリニア法
public int[] bilinear(int[] src, int srcX, int srcY, double scale ){
    int dstX = (int)(srcX*scale);
    int dstY = (int)(srcY*scale);
    int[] data = new int[dstX* dstY];
    for(int i = 0; i < dstY; ++i ){
        double pY = ((double)i/ scale) ;
        //範囲から漏れてたら修正
        if(pY+1 >= srcY){
            pY =srcY-2;
```

```
            }

            for(int j = 0; j < dstX; ++j ){
                double pX = ((double)j/ scale);
                if(pX+1 >= srcX ){
                    pX =srcX-2;
                }
                //あとは教科書通りの書き方
                double x1 =((int)pX+1-pX);
                double x2 = (pX-(int)pX);
                double y1 = (int)pY+1-pY;
                double y2 = pY -(int)pY;
                double index1 = (int)pY*srcX+(int)pX;
                double index2 = ((int)pY+1)*srcX+(int)pX;
                double index3 = ((int)pY)*srcX+(int)pX+1;
                double index4 = ((int)pY+1)*srcX+(int)pX+1;
                int red;
                int green;
                int blue;
                red = (int) (x1*y1*(double)((src[(int)index1] & 0x00ff0000)>>16)+
                             x1*y2*(double)((src[(int)index2] & 0x00ff0000) >> 16)+
                             x2*y1*(double)((src[(int)index3] & 0x00ff0000) >> 16)+
                             x2*y2*(double)((src[(int)index4] & 0x00ff0000) >> 16));
                green = (int) (x1*y1*(double)((src[(int)index1] & 0x0000ff00)>>8)+
                               x1*y2*(double)((src[(int)index2] & 0x0000ff00) >> 8)+
                               x2*y1*(double)((src[(int)index3] & 0x0000ff00) >> 8)+
                               x2*y2*(double)((src[(int)index4] & 0x0000ff00) >> 8));
                blue = (int) (x1*y1*(double)((src[(int)index1] & 0x000000ff))+
                              x1*y2*(double)((src[(int)index2] & 0x000000ff))+
                              x2*y1*(double)((src[(int)index3] & 0x000000ff))+
                              x2*y2*(double)((src[(int)index4] & 0x000000ff)));
                data[i*dstX+j] =  0xff <<24 | ByteRound(red) <<16 | ByteRound(green)<<8 | ByteRound(blue)
            }
        }
        System.out.println("end bilinear");
        return data;
    }

    public void paint( Graphics g ){
        g.drawImage( new_img, 0,0,this);
        System.out.println("print Image");
    }
}
```

3.4　実行結果

図 8: バイリニア法での拡大

3.5 bicubic 法

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class allMain extends Applet{

    private static final long serialVersionUID = 1L;
    Image img,img2;
    Image new_img,new_img2;

    int w_1 = 0;
    int h_1 = 0;
    int c_width = 0;//キャンパスの幅と高さ
    int c_height = 0;
    int c_axisx=0;//キャンパスの x 軸と y 軸の原点
    int c_axisy=0;

    int pix[];
    int campus[];//変換した画像を貼り付けるキャンパス


    public void init(){
        img = getImage( getCodeBase(),"test.jpg");

        MediaTracker mt = new MediaTracker( this );
        mt.addImage( img , 0 );

        try{
            mt.waitForID( 0 );
        }catch( InterruptedException e){
            System.out.println( e );
        }

        w_1 = img.getWidth( this );
        h_1 = img.getHeight( this );

        c_width = w_1*4;
        c_height = h_1*4;
        c_axisx = w_1*2;
        c_axisy = h_1*2;
        pix = new int[w_1*h_1];
        campus = new int[ c_width* c_height ];
        setPix();
    }//end init
```

```
void setPix(){
    try    {
        PixelGrabber pg = new PixelGrabber( img, 0, 0, w_1, h_1, pix, 0, w_1 );
        pg.grabPixels();
    }catch( InterruptedException e ){
        System.out.println( e );
    }

    int startX = c_width / 2+50;//キャンパスに張り付ける x 軸の点
    int startY = c_height / 2;//キャンパスに張り付ける y 軸の点

    ImagePaste( campus, pix, c_width, w_1, h_1, startX, startY);//画像をキャンパスに張り付ける。


    double scale =8.0;//補間法を使う場合の画像の拡大率
    int[] campus = bicubic( pix, w_1, h_1, scale);//バイキュービック法法

    MemoryImageSource mimg = new MemoryImageSource( (int) (w_1*scale),(int)(h_1*scale), campus,0,(int
    System.out.println("end");
    new_img = createImage( mimg );
}

//src の画像を dst に張り付ける。
public void ImagePaste( int[] dst, int[] src,int dstX, int srcX, int srcY, int width, int height ){
    for( int i = height ; i < height+srcY; ++i ){
        for( int j = width; j < width+srcX; ++j ){
            dst[i*dstX+j] = src[(i-height)*srcX+j-width];
        }
    }
}

public int ByteRound( int i ){
    if( i > 255 ) i = 255;
    else if(i < 0 ) i = 0;
    return i;
}

//バイキュービック法
public int[] bicubic(int[] src, int srcX, int srcY, double scale ){
    //拡大率によってサイズが変わるので新たな配列の大きさを決める
    int dstX = (int)(srcX*scale);
    int dstY = (int)(srcY*scale);
    int[] data = new int[dstX* dstY];

    for(int i = 0; i < dstY; ++i ){
        //元画像の注目ピクセルを取り出す
        double pY = ((double)i/ scale);
```

```
            for(int j = 0; j < dstX; ++j ){
                double pX = ((double)j/ scale);
                int dstRed=0;
                int dstGreen=0;
                int dstBlue = 0;

                //注目ピクセルの周囲 4 マスを計算
                for(int iy = (int)pY-1; iy <= (int)pY+2; ++iy){
                    for(int ix = (int)pX-1; ix <= (int)pX+2; ++ix ){
                        double wx =Math.abs(pX-ix);
                        double wy =Math.abs(pY-iy);

                        if( wx < 1 ){
                            wx = (wx-1)*(wx*wx-wx-1);
                        }else{
                            wx = -(wx-1)*(wx-2)*(wx-2);
                        }

                        if( wy < 1 ){
                            wy = (wy-1)*(wy*wy-wy-1);
                        }else{
                            wy =-(wy-1)*(wy-2)*(wy-2);
                        }

                        double x0 = ix;
                        double y0 = iy;
                        //画像内に収まらなかったら範囲を修正
                        if( x0 < 0 || x0 > srcX-1 )    {
                            x0 = (int)pX;
                        }

                        if( y0 < 0 || y0 > srcY-1 )    {
                            y0 = (int)pY;
                        }

                        dstRed += ((src[(int) (y0*srcX+x0)] & 0x00ff0000) >> 16 )*wx*wy;
                        dstGreen += ((src[(int) (y0*srcX+x0)] & 0x0000ff00) >> 8 )*wx*wy;
                        dstBlue += ((src[(int) (y0*srcX+x0)] & 0x000000ff) )*wx*wy;
                    }
                }
                //0xff << 24 を入れないと色がおかしくなる。
                data[i*dstX+j] =  0xff << 24 | ByteRound(dstRed) <<16 | ByteRound(dstGreen) << 8 | ByteRo
            }
        }
        System.out.println("end bicubic");
        return data;
    }
```

```
    public void paint( Graphics g ){
        g.drawImage( new_img, 0,0,this);
        System.out.println("print Image");
    }
}
```

3.6　実行結果

図 9: バイキュービック法での拡大