2-Dimensional Shape Categorization Using Polar Coordinate Representations

Introduction

Polar Coordinates

θ

θ π π

θ

θ π

θ

θ

π π

θ

Standardizing Shapes

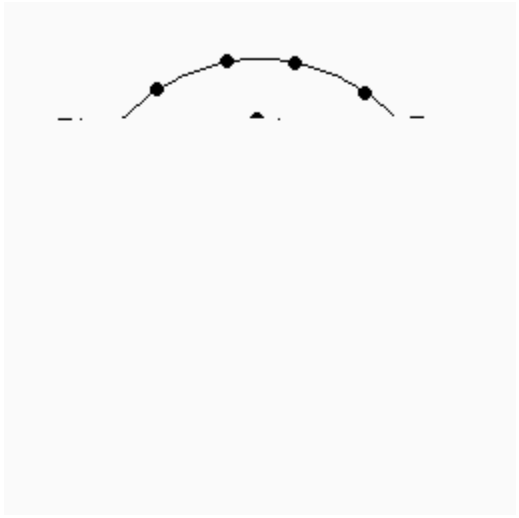
θ

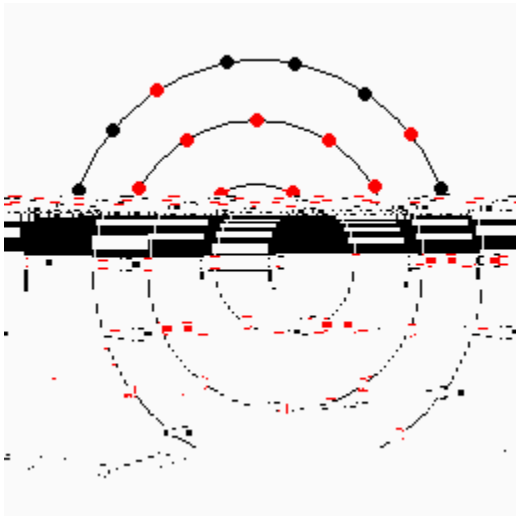
π

θ

θ π π

θ



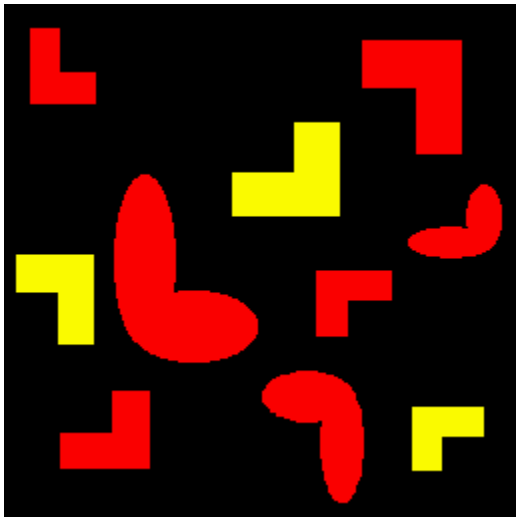


Stage 2: Grouping Shapes from a Scene in a BMP File

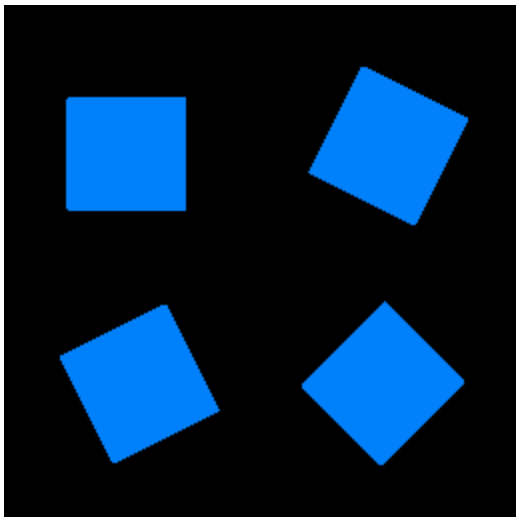
π

θ

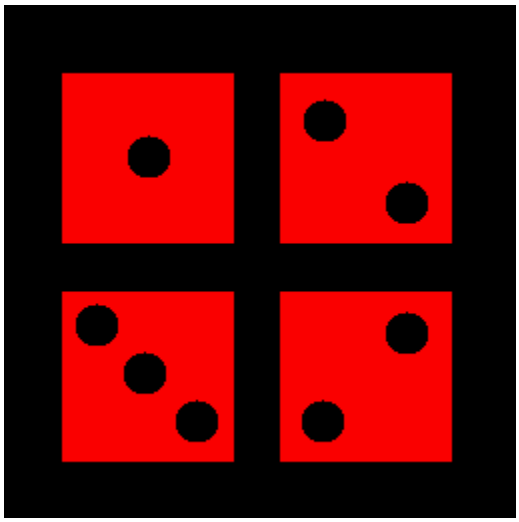
Experiments



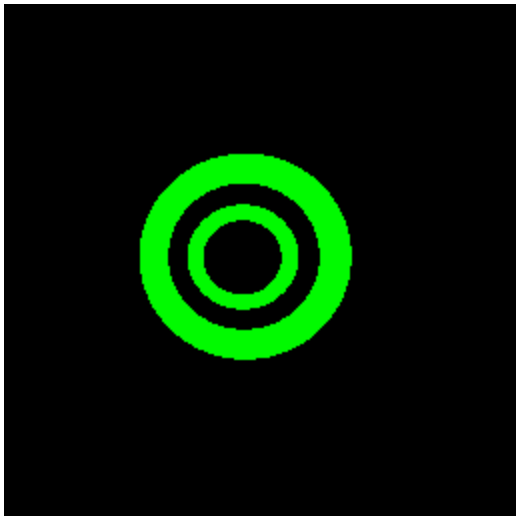
Concepts found:
Color: Red=255, Green=0, Blue=0
Center at (25.5385,34.1923)
Center at (54.2903,216.065)
Center at (170.808,145.538)
Center at (208.73,40.5421)
Instances: 4
Color: Red=255, Green=0, Blue=0
Center at (82.2759,141.342)
Center at (159.845, 208.786)
Center at (229.99,112.24)
Instances: 3
Color: Red=255, Green=255, Blue=0
Center at (29.0645,142.71)
Center at (145.555,86.8355)
Center at (218.132,213.132)
Instances: 3



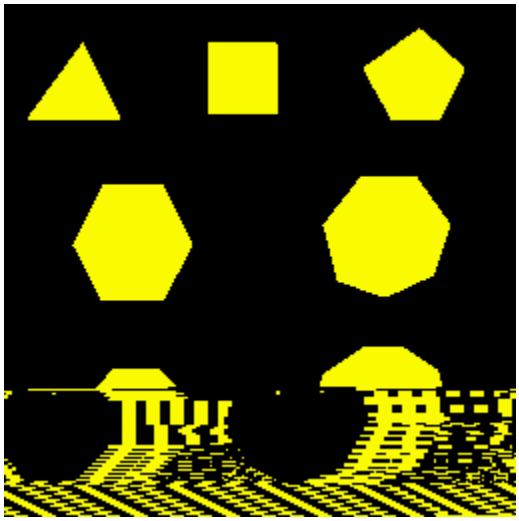
Concepts found:
Color: Red=0, Green=128, Blue=255
Center at (67.2016,188.903)
Center at (60.5173,74)
Center at (189,188.988)
Center at (191.798,69.9029)
Instances: 4



```
Concepts found:
Color: Red=255, Green==0, Blue=0
Center at (71.4746,76.0269)
Instances: 1
Concepts found:
Color: Red=255, Green==0, Blue=0
Center at (180.5,75.7894)
Center at (180.553,184.95)
Instances: 2
Concepts found:
Color: Red=255, Green==0, Blue=0
Center at (71.6412,185.259)
Instances: 1
```



```
Concepts found:
Color: Red=0, Green=255, Blue=0
Center at (121.04,125.8)
Center at (119.218,125.771)
Instances: 2
```



```
Concepts found:
Color: Red=255, Green=255, Blue=0
Center at (35.9425,44.4441)
Instances: 1
Color: Red=255, Green=255, Blue=0
Center at (118.986,36.5139)
Instances: 1
Color: Red=255, Green=255, Blue=0
Center at (118.986,36.5139)
Instances: 1
Color: Red=255, Green=255, Blue=0
Center at (63981,118.76)
Instances: 1
, Blu9:Insta.oo Red=255, Green=255, Blue=0Center at (118.986,36.511312
```

Center at (118.986,36.51122 Blue

Future Work

```
/*    Bitmap.cpp

      Function definitions for Bitmap class.
*/

#include <stdio.h>
#include <iostream.h>
#include "bitmap.h"
#include "misc.h"

/*    Bitmap::Bitmap(const char *filename)

      Constructor for Bitmap class; reads file specified
      by user.

      Parameters:

        filename - name of file to be read
*/

Bitmap::Bitmap(const char *filename)
{
      FILE *bmpfile = fopen(filename,"r");
      int i, j;

      header = new char[54];

      for (i=0;i<54;i++)
            fscanf(bmpfile,"%c",header+i);

      x = BMP_WIDTH;
      y = BMP_HEIGHT;

      color = new unsigned char **[x];

      for (i=0;i<x;i++) {
            color[i] = new unsigned char *[y];
            for (j=0;j<y;j++) {
                  color[i][j] = new unsigned char[3];
            }
      }

      for (j=y-1;j>=0;j--) {
            for (i=0;i<=x-1;i++) {
                  fscanf(bmpfile,"%c",&(color[i][j][B]));
                  fscanf(bmpfile,"%c",&(color[i][j][G]));
                  fscanf(bmpfile,"%c",&(color[i][j][R]));
            }
      }

      fclose(bmpfile);
}
```

```
/*    Scene2D.h

      Contains definition of Scene 2D class, a data representation
      of the scene being analyzed by the program.
*/

#ifndef _SCENE2DH_
#define _SCENE2DH_

#include "Bitmap.h"
#include "View2D.h"

class Scene2D {
      protected:
            int max_x, max_y; //width and height of image, respectively
            Bitmap *source;   //Bitmap object used to create Scene2D
                              //instance
            unsigned char ***color; //Color values of all coordinates
                                    //in scene

      public:
            Scene2D(char *bitmap);
            View2D * makeView2DPtr(int x1,int y1,int x,int y,int res);
            View2D * makeView2DPtr(int res) { return
makeView2DPtr(0,0,max_x,max_y,res); }
                  //view encompasses whole scene by default
            View2D * makeView2DPtr(void) { return
makeView2DPtr(0,0,max_x,max_y,1); }
                  //blur is set to 1 by default
};


#endif
```

```
/*    Scene2D.cpp

      Function definitions for the Scene2D class.
*/


#include <stdio.h>
#include <iostream.h>
#include "Scene2D.h"
#include "Bitmap.h"
#include "misc.h"

/*    Scene2D::Scene2D(char *bitmap)

      Constructor for Scene2D class; takes a filename,
      uses it to create a Bitmap object, then reads from
      the Bitmap object.

      Parameters:

        bitmap - filename of image to be analyzed
*/

Scene2D::Scene2D(char *bitmap)
{
      source = new Bitmap(bitmap);

      max_x = source->getX();
      max_y = source->getY();
      color = source->getColor();
}

/*    Pap object.
```

```
	if (x1<0) {
		x -= (int)((0 - x1)/res);
		x1 = 0;
	}
	if (y1<0) {
		y -= (int)((0 - y1)/res);
		y1 = 0;
	}
	if (x1+x*res>max_x) {
		x = (int)(max_x/res);
	}
	if (y1+y*res>max_y) {
		y = (int)(max_y/res);
	}

	new_color = new unsigned char **[x];

	for (i=0;i<x;i++) {
		new_color[i] = new unsigned char *[y];
		for (j=0;j<y;j++) {
			new_color[i][j] = new unsigned char[3];
		}
	}

	for (i=x1;i<x*res;i+=res) {
		for (j=y1;j<y*res;j+=res) {
			color_sum[R] = 0;
			color_sum[G] = 0;
			color_sum[B] = 0;
			for (k=i;k<i+res;k++) {
				for (l=j;l<j+res;l++) {
					color_sum[R] += color[k][l][R];
					color_sum[G] += color[k][l][G];
					color_sum[B] += color[k][l][B];
				}
			}
			new_color[i][j][R] = color_sum[R]/(res*res);
			new_color[i][j][G] = color_sum[G]/(res*res);
			new_color[i][j][B] = color_sum[B]/(res*res);
		}
	}

	return new View2D(new_color,x1,y1,x,y,res);
}
```

```
/*    View2D.h

      Definition of the View2D class, representing a "view", which is a
      (possibly zoomed-in) portion of a Scene2D object.  This class
      can also create a list of shapes contained within the boundaries
      of the view.
*/

#ifndef _VIEW2DH_
#define _VIEW2DH_

#include "Shape2D.h"
#include "misc.h"

class View2D
{
      protected:
            int x1, y1; //top left corner of the view
            int x, y;   //width and height of the view, respectively
            int res;    //zoom level of the view
            unsigned char ***color; //colors of coordinates within the
                                    //view
            bool **mark;      //coordinates of the view that have been
                              //checked for shapes
            void makePoint2DList(int curr_x,int curr_y,Point2DList
*pl_address);

      public:
            View2D(unsigned char ***new_color,int new_x1,int new_y1,int
new_x,int new_y,int new_res);
            Shape2DList makeShape2DList(void);
};

#endif
```

```
/*    View2D.cpp
```

```
      Shape2DList sl = NULL, new_sl, sl_counter;
      Point2DList pl = NULL, pl_killer;

      for (i=0;i<x;i++) {           //shape-finding loop
            for (j=0;j<y;j++) {
                  if ((!mark[i][j]) && !isBlack(color[i][j])) {
      //Black pixels are assumed to be background
                        makePoint2DList(i,j,&pl);
                        new_sl = new Shape2DListData;
                        new_sl->s = new Shape2D(pl);
                        if (sl==NULL) {
                              sl = new_sl;
                              sl->next = NULL;
                        }
                        else if (new_sl->s->getColorValue() >
sl->s->getColorValue()) {
                              new_sl->next = sl;
                              sl = new_sl;
                        }
                        else if (sl->next==NULL) {
                              sl->next = new_sl;
                              new_sl->next = NULL;
                        }
                        else {
                              sl_counter = sl;
                              while (sl_counter->next!=NULL) {
                                    if (new_sl->s->getColorValue() >
sl_counter->next->s->getColorValue()) {
                                          break;
                                    }
                                    sl_counter = sl_counter->next;
                              }
                              new_sl->next = sl_counter->next;
                              sl_counter->next = new_sl;
                        }

                  }
                  while (pl!=NULL) {
                        pl_killer = pl;
                        pl = pl->next;
                        delete pl_killer;
                  }
            }
```

```
        pl_address - a pointer to the point list being created
*/

void View2D::makePoint2DList(int curr_x,int curr_y,Point2DList
*pl_address)
{
      Point2DList new_pl;

      mark[curr_x][curr_y] = true;

      new_pl = new Point2DListData;
      new_pl->color = new unsigned char[3];
      new_pl->color[R] = color[curr_x][curr_y][R];
      new_pl->color[G] = color[curr_x][curr_y][G];
      new_pl->color[B] = color[curr_x][curr_y][B];
      new_pl->x = curr_x;
      new_pl->y = curr_y;
      new_pl->next = *pl_address;
      *pl_address = new_pl;

      if (curr_x>0) {
            if(isSameColor(color[curr_x][curr_y],color[curr_x-
1][curr_y]) && !mark[curr_x-1][curr_y])
                  makePoint2DList(curr_x-1,curr_y,pl_address);
      }
      if (curr_x<x) {

      if(isSameColor(color[curr_x][curr_y],color[curr_x+1][curr_y]) &&
!mark[curr_x+1][curr_y])
                  makePoint2DList(curr_x+1,curr_y,pl_address);
      }
      if (curr_y>0) {
            if(isSameColor(color[curr_x][curr_y],color[curr_x][curr_y-
1]) && !mark[curr_x][curr_y-1])
                  makePoint2DList(curr_x,curr_y-1,pl_address);
      }
      if (curr_y<y) {

      if(isSameColor(color[curr_x][curr_y],color[curr_x][curr_y+1]) &&
!mark[curr_x][curr_y+1])
                  makePoint2DList(curr_x,curr_y+1,pl_address);
      }
}
```

```
/*    Shape2D.h

      Definition of Shape2D class, used for storing polar equivalents
      of points forming shapes in the original image.  Also contains
      definitions of Ring, RingList, CenterList, StdShape2D,
      Polar2DList, and Shape2DList structures explained below.
*/

#ifndef _SHAPE2DH_
#define _SHAPE2DH_

#include <iostream.h>
#include <math.h>
#include "misc.h"

//Standardized Shape structures

/*    Ring

      A list of polar points with equal radius values
*/

struct RingData {
      float theta;      //theta value of a point in the ring
      RingData *next;   // the next counterclockwise point in the ring
};
typedef RingData *Ring;

/*    RingList

      A list of Rings with different radii
*/

struct RingListData {
      Ring ring;
      float r;    //radius of current ring
      int ring_size;    //maximum number of points in current ring
      RingListData *next;           //the next smaller ring in the list
};
typedef RingListData *RingList;

/*    CenterList

      A list of coordinates for centers of shapes, based on their scene
      coordinates.  The centers of shapes are calculated by averaging
      the x and y coordinates of points within a shape.
*/

struct CenterListData {
      float x;
      float y;
      CenterListData *next;
};
typedef CenterListData *CenterList;
```

```
/*  StdShape2D

      A standardized shape, consisting of a RingList containing
      coordinate information, the color of the shape, a list of the
      centers of known instances of the shape, and other statistical
      information about the shape.
*/

struct StdShape2DData {
      RingList rl;      //the shape's points
      int point_total, ring_total;  //total number of points and rings
                                    //in the shape
      unsigned char *color;   //the color of the shape (3-character
                              //array)
      float avg_r, stddev_r;  //mean and standard deviation of points
                              //in shape
      float r_error, theta_error;   //r and theta errors of shape
      CenterList cl;    //list of centers of known similar shapes
};
typedef StdShape2DData *StdShape2D;

//Unstandardized Shape structures

/*    Polar2DList

      A doubly-linked list of 2-dimesional polar coordinates, sorted by
      theta.
*/

struct Polar2DListData
{
      float r;
      float theta;
      Polar2DListData *next_cw, *next_ccw;//next clockwise and
                                          //counterclockwise points in
                                          //list, respectively
};
typedef Polar2DListData *Polar2DList;

/*    Shape2D

      An unstandardized shape, containing points sorted by their theta
      values.
*/

class Shape2D {
      protected:
            Polar2DList pol;  //the shape's points
            int point_total;  //the total number of points in the shape
            unsigned char *color;   //the color of the shape (3-
                                    //character array)
            float avg_r, stddev_r, max_r; //mean, standard deviation,
                                          //and maximum value of r
            int min_x, max_x, min_y, max_y;//minimum and maximum values
            //of the shape's original x and y coordinates, based on
            //their absolute positions in the BMP file
            float center_x, center_y;//the coordinates of the shape's
```

```
            //center, based on the coordinates of the points in the BMP
            //file

      public:
            Shape2D(Point2DList pl);
            int getColorValue(void)
            {
                  return color[R] + color[G] + color[B];
            }
            int GetMinX(void)
            {
                  return min_x;
            }
            int GetMaxX(void)
            {
                  return max_x;
            }
            int GetMinY(void)
            {
                  return min_y;
            }
            int GetMaxY(void)
            {
                  return max_y;
            }
            static float sameColor(Shape2D *s1,Shape2D *s2)
            {
                  return ((float)(abs(s1->color[R]-s2->color[R]) +
abs(s1->color[G]-s2->color[G]) + abs(s1->color[B]-s2->color[B])))/765;
            }
            static float sameSize(Shape2D *s1,Shape2D *s2)
            {
                  if (s1->avg_r<s2->avg_r)
                        return s1->avg_r/s2->avg_r;
                  else
                        return s2->avg_r/s1->avg_r;
            }
            static float sameStdDev(Shape2D *s1,Shape2D *s2)
            {
                  if (s1->stddev_r<s2->stddev_r)
                        return s1->stddev_r/s2->stddev_r;
                  else
                        return s2->stddev_r/s1->stddev_r;
            }
            float sameOrientation(Shape2D *s1,Shape2D *s2,float
r_error,float theta_error);
            float sameShape(Shape2D *s1,Shape2D *s2,float r_error,float
theta_error);
            StdShape2D standardize(int point_limit);
            void printInfo(void) {
                  cout << "Color: " << color[R] << "\t" << color[G] <<
"\t" << color[B] << "\n";
                  cout << "Average r: " << avg_r << "\n";
                  cout << "Std. dev. of r: " << stddev_r << "\n";
            }
};
```

```
/*    Shape2DList

      A linked list of Shape2D objects
*/

struct Shape2DListData {
      Shape2D *s;
      Shape2DListData *next;
};
typedef Shape2DListData *Shape2DList;

#endif
```

```
/*    Shape2D.cpp

      Function definitions for the Shape2D class
*/

#include <iostream.h>
#include <math.h>
#include "Shape2D.h"
#include "misc.h"

/*    Shape2D::Shape2D(Point2DList pl)

      Constructor for Shape2D class; converts a list of rectangular
      coordinates to polar coordinates, then calculates the shape's
      other attributes.

      Parameters:

        pl - a list of recangular coordinates
*/

Shape2D::Shape2D(Point2DList pl)
{
      Point2DList pl_counter;
      Polar2DList new_pol, pol_counter;
      double r_total = 0;
      double stddev_r_total = 0;
      int x_total = 0, y_total = 0;

      color = new unsigned char[3];
      color[R] = pl->color[R];
      color[G] = pl->color[G];
      color[B] = pl->color[B];

      point_total = 0;
      pl_counter = pl;

      while (pl_counter!=NULL) {
            point_total++;
            x_total += pl_counter->x;
            y_total += pl_counter->y;
            pl_counter = pl_counter->next;
      }

      center_x = (float)x_total/(float)point_total;
      center_y = (float)y_total/(float)point_total;
```

```
if (new_pol->r > max_r) {
      max_r = new_pol->r;
}
if (pl_counter->x != center_x) {
      if ((float)pl_counter->x-center_x>0.0f) {
```

```
      else {
            do {  //standard deviation calculation loop
                  stddev_r_total += fabs(pol_counter->r - avg_r);
                  pol_counter = pol_counter->next_ccw;
            } while (pol_counter!=pol);
            stddev_r = (float)(stddev_r_total / (double)point_total) /
avg_r;
      }

      pl_counter = pl;
```

```
      if (s2->avg_r==0.0f)
            return 1.0f;

      size_ratio = s1->avg_r/s2->avg_r;
      match_total = 0;
      pol_counter_1 = s1->pol;

      for (i=0;i<s1->point_total;i++) {
            pol_counter_2 = s2->pol;
            match_found = false;
            for (j=0;(j<s2->point_total) && (pol_counter_2->theta -
pol_counter_1->theta <= theta_error*PI_OVER_2/pol_counter_1->r) &&
!match_found;j++) {
                  if ((pol_counter_1->r==0.0f) && (fabs(pol_counter_2-
>r*size_ratio - pol_counter_1->r) <= r_error*size_ratio)) {
                        match_total++;
                        match_found = true;
                  }
                  else if ((fabs(pol_counter_2->r*size_ratio -
pol_counter_1->r) <= r_error*size_ratio) && (fabs(pol_counter_2->theta
- pol_counter_1->theta) <= theta_error*PI_OVER_2/pol_counter_1->r)) {
                        match_total++;
                        match_found = true;
                  }
                  pol_counter_2 = pol_counter_2->next_ccw;
            }
            pol_counter_1 = pol_counter_1->next_ccw;
      }
```

```
	result_2 = match_total/((float)s2->point_total);

	return result_1*result_2;
}

/*	Shape2D::sameShape(Shape2D *s1,Shape2D *s2,float r_error,float
theta_error)

	Compares two Shape2D objects to see if they have the same
	shape, regardless of orientaion.  Not used in final code.

	Parameters:

	  s1,s2 - the two shapes being compared
	  r_error - the maximum allowable error for the r value of a
	  point
	  theta_error - multiplier used in calculating maximum allowable
	  error for the theta value of a point

	Return Value:

	  A decimal value between 0 and 1, where 1 indicates
	  a perfect match and 0 indicates a complete mismatch.
*/

float Shape2D::sameShape(Shape2D *s1,Shape2D *s2,float r_error,float
theta_error)
{
	Polar2DList pol_counter_1, pol_counter_2, pol_counter_2_start,
best_pol_counter_2_start, pol_counter_2_mark;
	Shape2D *small_s, *large_s;
	float size_ratio;
	float result_1, result_2;
	float match_total, best_match_total;
	float theta_diff, theta_diff_2;
	bool match_found;
	int i, j, k;
	long order;
	float best_theta_diff;

	if (s1->avg_r<s2->avg_r) {
		small_s = s1;
		large_s = s2;
	}
	else {
		small_s = s2;
		large_s = s1;
	}

	if (large_s->avg_r==0.0f)
		return 1.0f;

	size_ratio = small_s->avg_r/large_s->avg_r;
	pol_counter_2_start = large_s->pol;
	best_match_total = 0;
	best_theta_diff = 0;
	best_pol_counter_2_start = pol_counter_2_start;
```

```
      order = 0;

      for (k=0;k<large_s->point_total;k++) {
            pol_counter_1 = small_s->pol;
            while ((fabs(pol_counter_2_start->r*size_ratio -
pol_counter_1->r) > r_error*size_ratio) && (k<large_s->point_total)) {
                  pol_counter_2_start = pol_counter_2_start->next_ccw;
                  k++;
            }
            pol_counter_2 = pol_counter_2_start;
            theta_diff = pol_counter_2->theta - pol_counter_1->theta;
            match_total = 0;
            for (i=0;i<small_s->point_total;i++) {
                  match_found = false;
                  pol_counter_2_mark = pol_counter_2;
                  theta_diff_2 = pol_counter_2->theta - pol_counter_1-
>theta - theta_diff;
                  if (theta_diff_2 < -PI)
                        theta_diff_2 += PI_TIMES_2;
                  if (theta_diff_2 <= theta_error * PI_OVER_2 /
pol_counter_1->r) {
                        j = 0;
                        while (!match_found && (j<large_s->point_total)
&& (theta_diff_2 <= theta_error*PI_OVER_2/pol_counter_1->r)) {
                              order++;
                              if ((pol_counter_1->r==0.0f) &&
(fabs(pol_counter_2->r*size_ratio - pol_counter_1->r) <=
r_error*size_ratio)) {
                                    match_total++;
                                    match_found = true;
                              }
                              else if ((fabs(pol_counter_2->r *
size_ratio - pol_counter_1->r) <= r_error*size_ratio) && (theta_diff_2
>= -theta_error*PI_OVER_2/pol_counter_1->r)) {
                                    match_total++;
                                    match_found = true;
                              }
                              pol_counter_2 = pol_counter_2->next_ccw;
                              theta_diff_2 = pol_counter_2->theta -
pol_counter_1->theta - theta_diff;
                              if (theta_diff_2 < -PI)
                                    theta_diff_2 += PI_TIMES_2;
                              j++;
                        }
                  }
                  pol_counter_2 = pol_counter_2_mark;
                  theta_diff_2 = pol_counter_2->theta - pol_counter_1-
>theta - theta_diff;
                  if (theta_diff_2 < -PI)
                        theta_diff_2 += PI_TIMES_2;
                  if (theta_diff_2 >= -
theta_error*PI_OVER_2/pol_counter_1->r) {
                        j = 0;
                        while (!match_found && (j<large_s->point_total)
&& (theta_diff_2 >= -theta_error*PI_OVER_2/pol_counter_1->r)) {
                              order++;
```

```
                              if ((pol_counter_1->r==0.0f) &&
(fabs(pol_counter_2->r*size_ratio - pol_counter_1->r) <= r_error *
size_ratio)) {
                                    match_total++;
                                    match_found = true;
                              }
                              else if ((fabs(pol_counter_2->r *
size_ratio - pol_counter_1->r) <= r_error*size_ratio) && (theta_diff_2
<= theta_error*PI_OVER_2/pol_counter_1->r)) {
                                    match_total++;
                                    match_found = true;
                              }
                              pol_counter_2 = pol_counter_2->next_cw;
                              theta_diff_2 = pol_counter_2->theta -
pol_counter_1->theta - theta_diff;
                              if (theta_diff_2 < -PI)
                                    theta_diff_2 += PI_TIMES_2;
                              j++;
                        }
                  }
                  pol_counter_1 = pol_counter_1->next_ccw;

            }
            if (match_total>best_match_total) {
                  best_match_total = match_total;
                  best_theta_diff = theta_diff;
                  best_pol_counter_2_start = pol_counter_2_start;
            }
            pol_counter_2_start = pol_counter_2_start->next_ccw;
      }

      result_1 = best_match_total/((float)small_s->point_total);

      pol_counter_1 = best_pol_counter_2_start;
      pol_counter_2 = small_s->pol;
      theta_diff = -best_theta_diff;
      match_total = 0;

      for (i=0;i<large_s->point_total;i++) {
            match_found = false;
            pol_counter_2_mark = pol_counter_2;
            theta_diff_2 = pol_counter_2->theta - pol_counter_1->theta
- theta_diff;
            if (theta_diff_2 < -PI)
                  theta_diff_2 += PI_TIMES_2;
            else if (theta_diff_2 > PI)
                  theta_diff_2 -= PI_TIMES_2;
            if (theta_diff_2 <= theta_error*PI_OVER_2/pol_counter_2->r)
{
                  j = 0;
                  while (!match_found && (j<small_s->point_total) &&
(theta_diff_2 <= theta_error*PI_OVER_2/pol_counter_2->r)) {
                        order++;
                        if ((pol_counter_1->r==0.0f) &&
(fabs(pol_counter_2->r/size_ratio - pol_counter_1->r) <=
r_error*size_ratio)) {
                              match_total++;
```

```
                              match_found = true;
                        }
                        else if ((fabs(pol_counter_2->r/size_ratio -
pol_counter_1->r) <= r_error/size_ratio) && (theta_diff_2 >= -
theta_error*PI_OVER_2/pol_counter_2->r)) {
                              match_total++;
                              match_found = true;
                        }
                        pol_counter_2 = pol_counter_2->next_ccw;
                        theta_diff_2 = pol_counter_2->theta -
pol_counter_1->theta - theta_diff;
                        if (theta_diff_2 < -PI)
                              theta_diff_2 += PI_TIMES_2;
                        else if (theta_diff_2 > PI)
                              theta_diff_2 -= PI_TIMES_2;
                        j++;
                  }
            }
            pol_counter_2 = pol_counter_2_mark;
            theta_diff_2 = pol_counter_2->theta - pol_counter_1->theta
- theta_diff;
            if (theta_diff_2 < -PI)
                  theta_diff_2 += PI_TIMES_2;
            else if (theta_diff_2 > PI)
                  theta_diff_2 -= PI_TIMES_2;
            if (theta_diff_2 >= -theta_error*PI_OVER_2/pol_counter_2-
>r) {
                  j = 0;
                  while (!match_found && (j<large_s->point_total) &&
(theta_diff_2 >= -theta_error*PI_OVER_2/pol_counter_2->r)) {
                        order++;
                        if ((pol_counter_1->r==0.0f) &&
(fabs(pol_counter_2->r/size_ratio - pol_counter_1->r) <= r_error /
size_ratio)) {
                              match_total++;
                              match_found = true;
                        }
                        else if ((fabs(pol_counter_2->r/size_ratio -
pol_counter_1->r) <= r_error/size_ratio) && (theta_diff_2 <=
theta_error*PI_OVER_2/pol_counter_2->r)) {
                              match_total++;
                              match_found = true;
                        }
                        pol_counter_2 = pol_counter_2->next_cw;
                        theta_diff_2 = pol_counter_2->theta -
pol_counter_1->theta - theta_diff;
                        if (theta_diff_2 < -PI)
                              theta_diff_2 += PI_TIMES_2;
                        else if (theta_diff_2 > PI)
                              theta_diff_2 -= PI_TIMES_2;
                        j++;
                  }
            }
            pol_counter_1 = pol_counter_1->next_ccw;
      }

      result_2 = match_total/((float)large_s->point_total);
```

```
      return result_1*result_2;
}

/* StdShape2D Shape2D::standardize(int point_limit)

      Converts the current Shape2D object into a StdShape2D structure.

      Parameters:

        point_limit - maximum number of points allowed in StdShape2D
        representation being created

      Return Value:

        The StdShape2D structure created from the current Shape2D
        object
*/

StdShape2D Shape2D::standardize(int point_limit)
{
      StdShape2D new_ss = new StdShape2DData;
      Ring r_counter;
      RingList rl_counter;
      Polar2DList pol_counter;
      float curr_theta, theta_inc;
      float r_total = 0.0f, stddev_total = 0.0f;
      bool match_found;
      int i, j;

      new_ss->cl = new CenterListData;
      new_ss->cl->x = center_x;
      new_ss->cl->y = center_y;
      new_ss->cl->next = NULL;

      new_ss->color = new unsigned char[3];
      new_ss->color[R] = color[R];
      new_ss->color[G] = color[G];
      new_ss->color[B] = color[B];

      if (point_total<=point_limit) {
            new_ss->ring_total = (int)max_r;
      }
      else {
            new_ss->ring_total = (int)((sqrt(4*point_limit/PI+1)-1)/2);
      }

      new_ss->r_error = 1.0f/(float)new_ss->ring_total;
      new_ss->theta_error = PI_TIMES_2/(float)((int)(PI_TIMES_2 *
new_ss->ring_total));
      new_ss->point_total = 0;
      new_ss->rl = new RingListData;
      rl_counter = new_ss->rl;

      for (i=new_ss->ring_total;i>0;i--) {      //StdShape2D creation
loop
            rl_counter->ring_size = (int)(i*PI_TIMES_2);
```

```
                        r_counter = r_counter->next;
                    } while (r_counter!=rl_counter->ring);
                }
                rl_counter = rl_counter->next;
            } while (rl_counter!=NULL);

            new_ss->stddev_r = stddev_total/new_ss->point_total;

            return new_ss;
}
```

```
/*
```

```
                        theta_1_adjusted = r_counter_1->theta +
theta_diff;
                        if (theta_1_adjusted>PI) {
                              theta_1_adjusted -= PI_TIMES_2;
                        }
                        if (theta_1_adjusted>last_theta_1_adjusted) {
                              r_counter_1 = r_counter_1->next;
                        }
                  } while (theta_1_adjusted>last_theta_1_adjusted);
                  r_counter_1_start = r_counter_1;
                  ring_1_start = true;
                  ring_2_start = true;
                  do {  //    Ring comparison loop
                        theta_1_adjusted = r_counter_1->theta +
theta_diff;
                        if (theta_1_adjusted>PI) {
                              theta_1_adjusted -= PI_TIMES_2;
                        }
                        if (fabs(theta_1_adjusted-r_counter_2->theta) <
s1->theta_error/rl_counter_1->r-0.0001f) {
                              if (ring_2_start) {
                                    match_total_2++;
                                    match_found = true;
                                    r_counter_2 = r_counter_2->next;
                                    ring_2_start = false;
                              }
                              else if (r_counter_2!=rl_counter_2->ring)
                              {
                                    match_total_2++;
                                    match_found = true;
                                    r_counter_2 = r_counter_2->next;
                              }
                              else {
                                    r_counter_1 = r_counter_1_start;
                              }
                        }
                        else if (theta_1_adjusted>r_counter_2->theta) {
                              if (ring_2_start) {
                                    r_counter_2 = r_counter_2->next;
                                    ring_2_start = false;
                              }
                              else if (r_counter_2!=rl_counter_2->ring)
                              {
                                    r_counter_2 = r_counter_2->next;
                              }
                              else {
                                    r_counter_1 = r_counter_1_start;
                              }
                        }
                        else {
                              if (match_found && new_ring_1) {
                                    match_total_1++;
                              }
                              match_found = false;
                              if (ring_1_start) {
                                    r_counter_1 = r_counter_1->next;
                                    ring_1_start = false;
```

```
                              }
                              else if (r_counter_1!=r_counter_1_start)
                              {
                                    r_counter_1 = r_counter_1->next;
                              }
                              else {
                                    r_counter_2 = rl_counter_2->ring;
                              }
                        }
                  } while ((r_counter_1!=r_counter_1_start) ||
(r_counter_2!=rl_counter_2->ring));
                  rl_counter_2 = rl_counter_2->next;
                  while (rl_counter_2!=NULL) {
                        if (rl_counter_2->ring!=NULL)
                              break;
                        else
                              rl_counter_2 = rl_counter_2->next;
                  }
                  new_ring_1 = false;
                  while ((rl_counter_1!=NULL) && (rl_counter_2!=NULL))
                  {
                        if ((fabs(rl_counter_1->r-rl_counter_2->r) <
s1->r_error-0.0001) && (rl_counter_1->ring!=NULL) && (rl_counter_2-
>ring!=NULL))
                              break;
                        if (rl_counter_1->r>rl_counter_2->r) {
                              while (rl_counter_1!=NULL) {
                                    if (((rl_counter_1->r <
rl_counter_2->r) || (fabs(rl_counter_1->r-rl_counter_2->r)<s1->r_error-
0.0001)) && (rl_counter_1->ring!=NULL))
                                          break;
                                    rl_counter_1 = rl_counter_1->next;
                                    new_ring_1 = true;
                              }
                        }
                        else {
                              while (rl_counter_2!=NULL) {
                                    if (((rl_counter_2->r<rl_counter_1-
>r) || (fabs(rl_counter_1->r-rl_counter_2->r)<s1->r_error-0.0001)) &&
(rl_counter_1->ring!=NULL))
                                          break;
                                    rl_counter_2 = rl_counter_2->next;
                              }
                        }
                  }
            } while((rl_counter_1!=NULL) && (rl_counter_2!=NULL));
            result = ((float)match_total_1/(float)s1->point_total) *
((float)match_total_2/(float)s2->point_total);
            if (result>best_result) {
                  best_match_total_1 = match_total_1;
                  best_match_total_2 = match_total_2;
                  best_result = result;
            }
            theta_diff += theta_inc;
      } while (theta_diff<PI_TIMES_2);

      return best_result;
```

```
}

/*    bool Concept::isInstance(StdShape2D nominee)

      Determines whether or not a shape is a member of this concept.

      Parameters:

        nominee - Shape being compared to current concept.

      Return Value:

        Boolean value answering the question, "Is the new shape
        an instance of the current concept?"
*/

bool Concept::isInstance(StdShape2D nominee)
{
      CenterList cl_counter;

      if ((sameShape(nominee,s)>=MIN_SAMENESS) &&
(sameColor(nominee,s)>=MIN_SAMENESS)) {
            instances++;
            cl_counter = s->cl;
            while (cl_counter->next!=NULL) {
                  cl_counter = cl_counter->next;
            }
            cl_counter->next = nominee->cl;
            if (nominee->point_total>s->point_total) {
                  nominee->cl = s->cl;
                  s = nominee;
            }
            return true;
      }
      else {
            return false;
      }
}

/*    void Concept::printStdShape(void)

      Displays information about the current concept's color
      and intstances.
*/

void Concept::printStdShape(void)
{
      CenterList cl_counter;

      cout << "Color: Red=" << (int)s->color[R] << ", Green=" <<
(int)s->color[G] << ", Blue=" << (int)s->color[B] << "\n";
      cl_counter = s->cl;

      while (cl_counter!=NULL) {
            cout << "Center at (" << cl_counter->x << "," <<
cl_counter->y << ")\n";
            cl_counter = cl_counter->next;
```

```
        }
    }
```

```
/*    Tommy.h

      The Tommy class, responsible for using the other classes to
      extract shapes from image files and for categorizing the shapes
      into concepts.
*/

#ifndef _TOMMYH_
#define _TOMMYH_

#include "Concept.h"
#include "misc.h"

class Tommy {
      protected:
            ConceptList cl, last;   //the list of known concepts, and
                                    //the last concept in that list
      public:
            Tommy(int a);
            void processFile(char *bitmap);
            void addConcept(Concept *new_c);
            void printConceptList(void);
};

#endif
```

```
/*    Tommy.cpp

      Function definitions for the Tommy class.
*/

#include <stdio.h>
#include <iostream.h>
#include "Tommy.h"
#include "Scene2D.h"
#include "View2D.h"
#include "Shape2D.h"
#include "Concept.h"

/*    Tommy::Tommy(int a)

      The constructor for the Tommy class; creates an empty concept
      list.

      Parameters:

        a - a dummy variable created to fix a glitch in the compiler
        that caused the default constructor to be called instead of
```

```
            if (std!=NULL) {
                  cl_counter = cl;
                  while (cl_counter!=NULL) {
                        if (cl_counter->c->isInstance(std))
                              break;
                        cl_counter = cl_counter->next;
                  }
                  if (cl_counter==NULL) {
                        addConcept(new Concept(std));
                  }
                  sl_counter = sl_counter->next;
            }
      }
}

/*    void Tommy::addConcept(Concept *new_c)

      Adds a new concept to the concept list, which is sorted in
      order of decreasing numbers of instances.

      Parameters:

        new_c - the new concept
*/

void Tommy::addConcept(Concept *new_c)
{
      ConceptList cl_counter, new_cl;

      new_cl = new ConceptListData;
      new_cl->c = new_c;
      new_cl->next = NULL;

      if (cl==NULL) {
            cl = new_cl;
      }
      else {
            cl_counter = cl;
            while (cl_counter->next!=NULL) {
                  if (new_c->getInstances()>=cl_counter->c-
>getInstances()) {
                        new_cl->next = cl_counter->next;
                        cl_counter->next = new_cl;
                        break;
                  }
                  cl_counter = cl_counter->next;
            }
            if (cl_counter->next==NULL) {
                  cl_counter->next = new_cl;
            }
      }
}

/*    void Tommy::printConceptList(void)

      Displays information on all of the concepts in the concept list.
*/
```

```
void Tommy::printConceptList(void)
{
      ConceptList cl_counter = cl;

      while (cl_counter!=NULL) {
            cl_counter->c->printStdShape();
            cout << "Instances: " << cl_counter->c->getInstances() <<
"\n";
            cout << "\n";
            cl_counter = cl_counter->next;
      }
}
```

```
/*    misc.h

      Miscellaneous constants, structures and
      function headers that didn't fit under any
      one class.
*/

#ifndef _MISCH_
#define _MISCH_

#define PI 3.14159265f  //Value of PI
#define PI_OVER_2 1.57079633f //Value of PI/2
#define PI_TIMES_2 6.28318531f      //Value of PI*2

#define BMP_WIDTH 256   //width, in pixels, of BMP file
#define BMP_HEIGHT 256  //height, in pixels, of BMP file

#define MAX_DIST 1000000.0f

#define R 0       //red value in a color array
#define G 1       //green value in a color array
#define B 2       //blue value in a color array

#define MIN_SAMENESS 0.9f     //threshold for determining whether two
                              //colors or shapes are the same

//"type" values for Tommy constructor; not used in current program
#define LOAD_TOMMY 0;
#define NEW_TOMMY 1;

#define POINT_LIMIT 5000      //maximum number of points in allowed in
                              //a standardized shape

/*    Point2DList

      A linked list of rectangular integer coordinates
*/

struct Point2DListData
{
      int x, y;
      unsigned char *color;   //color at coordinate (x.y)
      Point2DListData *next;
};
typedef Point2DListData *Point2DList;

bool isBlack (unsigned char *c);
bool isSameColor(unsigned char *c1,unsigned char *c2);

#endif
```

```
#include "misc.h"

/*    isBlack(unsigned char *c)

      Determines whether a given color value is black

      Parameters:

        c - A color value, stored as a 3-character array

      Return Value:

        "Is the specified color black?"
*/

bool isBlack(unsigned char *c)
{
      return (c[R]==0) && (c[G]==0) && (c[B]==0);
}

/*    isSameColor(unsigned char *c1,unsigned char *c2)

      Determines whether two color values are exactly the same

      Parameters:

        c1, c2 - the two colors being compared.

      Return Value:

        Are the two specified colors exactly the same?
*/

bool isSameColor(unsigned char *c1,unsigned char *c2)
{
      return (c1[R]==c2[R]) && (c1[G]==c2[G]) && (c1[B]==c2[B]);
}
```

```
/*    testmain.cpp

      Location of the main function, which uses an instance of the
      Tommy class to convert a BMP file into a list of Concept objects.
*/

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include "Bitmap.h"
#include "Scene2D.h"
#include "View2D.h"
#include "Shape2D.h"
#include "Concept.h"
#include "Tommy.h"
#include "misc.h"

void main(void)
{
      char bmp[40];
      Tommy *t;
      char end;
      int x = (int)22.9;

      cout << "Enter name of the image to examine: ";
      cin >> bmp;
      t = new Tommy(0);
      t->processFile(bmp);
      cout << "The image contained the following distinct concepts:\n";
      t->printConceptList();
      cin >> end;
}
```