National Institute of Technology
Karnataka, Surathkal

Computer Graphics

Mini-Project

IT255

Concept Screensaver
using OpenGL

Team:
Apoorva Chandra S
11IT09
Dilip Mallya 11IT19
Kartik Sreenivasan 11IT37
Kaushik R 11IT65
S Avinash 11IT74

19th April, 2013

Designed using LaTeX

Contents

Introduction

This project involves the implementation of all the algorithms we have had at our disposal, to create an interactive and immersive screensaver.

Computer graphics is a growing field with a variety of different algorithms for implementing the most basic of primitives as well as for implementing complex transformations and animation. One of the most common places we find a mixture of drawing primitives and animating them is in screen savers. There are a wide variety of screen savers each of which use combinations of graphics primitives and complicated transformations to capture the attention of the user. Since a screensaver offers the scope of implementing the largest number of algorithms, we have chosen this to be our project.

The platform we are using is OpenGL but we are implementing even the most basic primitives such as lines, polygons using our own functions and libraries. All these programs are written in C and compiled by the gcc compiler on Linux platform.

Project Description

Our project is divided into different modules, each of which represent a component that is complete in itself. This format aids in debugging and also helps in understanding and improving the code.
The files Triangle, Circle, and Square are each responsible for drawing their respective figures, filling them as well as performing transformations such as rotation and shearing on them.
The triangle file draws the triangle given its centroid using Bresenham's line drawing algorithm. It is filled using our fill algorithm as described below. The lines are drawn using a slightly modified Bresenham's line drawing algorithm as we have to draw lines with negative slope as well. The $specialPlot()$ Function takes care of lines which have negative slope.
The Square file draws a square given its center by drawing 4 lines using DDA algorithm. We need to use DDA algorithm here even though it is slower as the other algorithms are limited to gentle slope. It then fills the square.
The Circle file uses Mid Point circle drawing algorithm to draw a circle given a center and radius. It then fills the circle by checking which points lie in the circle using the function : $f(c) = x^2 + y^2 - r^2$.
The *ScreenSaver.c* file contains the main functions required to get the screensaver working. Until now we have only drawn static objects which are much easier to draw as you dont have to keep track of their location at each instant of time. We achieve motion of multiple objects at the same time by repeatedly calling the above functions at slightly different locations to give the illusion of motion. This is achieved by calling these functions in a loop. To decide which object is drawn on the screen we have a variable shapeDecider which is seeded by random numbers. Depending on its value at that iteration of the loop, it chooses which function to call. Hence the objects that appear on the screen are randomised and you cannot predict which object will be drawn next.
The file *OpeningScreen.c* is the opening screen that you view when you execute the program. The program is run using the script make ($./execfilename$) which was written by us. *OpeningScreen.c* draws the buttons and chooses color of the button from a group of random colors that we have defined earlier in a structure. It then draws the smiley at its last known position using the display file *smiDisp.txt*. Each time the program executes it reads from the display file to get its location. Also there is a mouse handling function that is active during the entire execution of the program that takes the user's input. By clicking on the 1st button the user changes the color of the button, and the next click on the screen draws the smiley at that location with that color. *glbx* and *glby* are global variables that hold the final x and y coordinates of the most recent mouse click after some transformations have been done to convert it to our coor-

dinated system. This is needed because openGL gives mouse coordinates from computer perspective and the values vary from top left corner $(0, 0)$ to bottom right corner(r, r). Whereas we follow the cartesian system where center of the screen is the origin. Based on where you click on the screen this file guides you to the screen saver or to an alternate window where you can draw a smiley using our rubber banding algorithm. Each triangle on the border of the screen has been drawn from the display file.

Algorithms

3.1 DDA Line Drawing Algorithm

- Input the two endpoint pixel positions.
- Horizontal and vertical differences dx and dy are calculated.
- The difference with the greater magnitude determines the value of parameter *steps*.
- Starting with pixel position (x_0, y_0), determine the offset needed at each step to generate the next pixel position along the line path.
- Loop through this process *steps* times

3.2 Bresenham Line Drawing Algorithm $(for|m| < 1.0)$

- Input the two line end-points, storing the left end-point in (x_0, y_0).
- Plot the point (x_0, y_0).
- Calculate the constants $\Delta x, \Delta y, 2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as : $p_0 = 2dy - dx$.
- At each x_k along the line, starting at $k = 0$, perform the following test. If $pk < 0$, the next point to plot is $(x_k + 1, y_k)$and : $p_k + 1 = p_k + dy$.

  Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and: $p_k + 1 = p_k + 2dy - 2dx$.
- Repeat step 4 $(\Delta x - 1)$ times

3.3 Mid-Point Circle Drawing Algorithm

- Input radius rand circle centre (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centred on the origin as: $(x_0, y_0) = (0, r)$.
- Calculate the initial value of the decision parameter as:$p_0 = 5/4 - r$.

- Starting with $k = 0$ at each positionx_k, perform the following test. If$p_k < 0$, the next point along the circle centred on $(0, 0)$ is $(x_k + 1, y_k)and$: $p_k + 1 = 2x_k + 1 + 1$.
- Otherwise the next point along the circle is $(x_k + 1, y_k - 1)$ and: $p_k + 1 = p_k + 2x_k + 1 + 1 - 2y_k + 1$.
- Determine symmetry points in the other seven octants.
- Move each calculated pixel position (x, y) onto the circular path centred at (xc, yc) to plot the coordinate values: $x = x + x_c, y = y + y_c$.
- Repeat steps 3 to 5 until $x \geq y$.

3.4 Filling Algorithms

The algorithm we have used in our project is similar to the scan-line fill algorithm but slightly modified. The scan line fill algorithm involves a lot of computations as it varies X and Y along the entire screen. We reduce the computations by reducing the size of the raster over which we vary X and Y.

- Define a Square boundary that encompasses the figure that needs to be filled.
- Use the sides of the square to find $X_{min}, X_{max}, Y_{min}, Y_{max}$.
- Vary X from X_{min} to X_{max}, Y from Y_{min} to Y_{max}.
- For each (X, Y) check if the point lies within the figure using a function $f(x)$ defined specifically for each figure.
- If it lies within the figure, draw the point, else move to the next point.

3.5 Translation

- Translation works by shifting the point, by certain offsets to translate an object or a point.
- Set tx,ty be the translation distance, we have,
  $x' = x + t_x$ and $y' = y + t_y$
- Using matrices this could be depicted as,
  $P' = P + T$
  $i.e, \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$
- Translating an object, say a triangle, could be done in many ways.
  It could be translated by translating every point by the offset.
  Or it could be done by translating the endpoints and then redrawing the lines connecting them.
  In the case of objects like circles, a single point is all it takes to translate the object.

3.6 Rotation

- Rotation is basically repositioning objects along a circular path in the xy plane.
- We need to specify,
  - rotation angle
  - A position (x,y) - rotation point (pivot point)
  - direction (clockwise (-), counter clockwise(+))
- The original coordinates of the point in polar coordinates

$$x = rcos\phi, y = rsin\phi$$

- Transformed coordinates are,

$$x' = rcos(\phi + \theta) = rcos\phi cos\theta - rsin\phi sin\theta$$
$$y' = rsin(\phi + \theta) = rcos\phi sin\theta + rsin\phi cos\theta$$

  - r=the constant distance of the point from the origin.
  - ϕ =the original angular position of the point from the horizontal.
  - θ =the rotation angle
  - **Here the pivot point is at the co-ordinate origin.**
- Transformation for rotating a point at position (x, y) through an angle θ about the origin.

$$x' = xcos\theta - ysin\theta$$
$$y' = xsin\theta + ycos\theta$$

- In matrix form, we represent it as,

$$P' = R \cdot P = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

3.7 Scaling

- To alter the size of an object by multiplying the coordinates with scaling factor s_x and s_y

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

- In matrix format, where S is a 2×2 scaling matrix,
  i.e,

$$P' = P \cdot S \ ;$$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

3.8 Exterior Clipping

- Here we are preserving the region outside the clipping window, i.e masking the clipping window instead.
- Procedures for clipping objects to the interior of concave polygon windows can be used.
- **Applications of Exterior Clipping**
  - In designing multiple window systems.
  - In the design of page layouts in advertising or publishing.
  - Adding labels or design patterns to a picture.

Coding

The code has been written in a modular fashion, which is essential when a the program is to be easily updated with newer ideas and allow newer techniques to be used.

We have utilised a rolling release model, where we constantly updated our project as newer algorithms were implemented. This meant that we could start developing our project quite early on in our timeline. This also meant, of course, that we could easily rollback any changes that may caused the program to fail.

To run the file we have included a makefile (*basically a shellscript that would compile and run the code*).
This program was developed for OpenGL on Linux based systems. If OpenGL is not installed, it can be installed using the instructions below. The packages required are available on most Linux repositories.

sudo apt-get update sudo apt-get install mesa-common-dev sudo apt-get install freeglut3-dev

To run, use the make file included, or compile as follows;
gcc OpeningScreen.c -lglut -lGL -lGLU -lX11 -lm ¿ /dev/null
./a.out

4.1 Basic Primitives - square.c

```
1  #include<GL/glut.h>
2  #include<time.h>
3  #include<math.h>
4  #include<stdio.h>
5  //file that contains the rotating facility also for the square
6  float s=12;
7  struct sqlast
8  {
9   float xsqmem;  // a data structure that holds the last position of
        the square
10  float ysqmem;
11 }sql;
12 struct trans  // a data structure to do the rotation
13 {
14  float ao,bo,co,doo;
15 }tr;
16 struct trans rotator(float x0,float y0,float x1,float y1,float rad)
       // the rotation algorithm->the matrix form
17 {
18  tr.ao=x0*cos(rad)-y0*sin(rad);
```

```
19   tr.bo=x0*sin(rad)+y0*cos(rad);
20   tr.co=x1*cos(rad)-y1*sin(rad);
21   tr.doo=x1*sin(rad)+y1*cos(rad);
22   return tr;// returning of a structure  object that holds the
         transformed position
23 }
24 void ddaline(float x1,float y1,float x2,float y2,float r,float g,
       float b) // the digital differential analyser line drawing
       algorithm
25 {

       // to draw the sides of the square
26   float xinc,yinc,k,x=x1,y=y1,steps,dx,dy;
27   dx=x2-x1;    //change in x
28   dy=y2-y1;    //change in y
29   if(abs(dx)>abs(dy))
30   steps=abs(dx);
31   else
32   steps=abs(dy);
33   xinc=((float)dx/(float)steps);  // increment for the line drawn
         along the x axis
34   yinc=((float)dy/(float)steps);  // increment for the line drawn
         along the y axis
35   glColor3f(sin(r),cos(g),sin(b));
36   glBegin(GL_POINTS);
37   glVertex2f(x,y);
38   for(k=0;k<steps;k++)
39    {
40       x=x+xinc;
41       y=y+yinc;
42       glVertex2f(x,y);
43    }
44   glEnd();
45 //glutSwapBuffers();
46 //glutPostRedisplay();
47 }
48 //function that draws a square
49 //sides are drawn using the dda algorithm
50 //the filling of the square is done using a custom made "scan line
       filling" algorithm
51 //parameters to this function are
52 //parameter      type                  function
53 // x1            float           x coordinate of transalation of the
       square
54 // y1            float           y coordinate of transalation of the
       square
55 // red           float           red component of the square's color
56 // green         float           green component of the square's
       color
57 // blue          float           blue component of the square's color
58 //ang            float           angle wrt the horizonatal at which
       the square is to be drawn->rotational component
59 struct sqlast drawSquare(float x1,float y1,float red,float green,
       float blue,float ang)
60 {
61   int j;
62   float x,y,rad,i;
63   rad=ang*3.14159263/180;//conversion of angle in degrees to radians
64
65   rotator(x1,y1-s,x1,y1,rad);  //coordinates are first rotated
66   ddaline(tr.ao,tr.bo,tr.co,tr.doo,red,green,blue);//then drawn
67   rotator(x1,y1,s+x1,y1,rad);
```

```
68   ddaline(tr.ao,tr.bo,tr.co,tr.doo,red,green,blue);
69   rotator(x1,y1-s,s+x1,y1-s,rad);
70   ddaline(tr.ao,tr.bo,tr.co,tr.doo,red,green,blue);
71   rotator(x1+s,y1-s,s+x1,y1,rad);
72   ddaline(tr.ao,tr.bo,tr.co,tr.doo,red,green,blue);
73
74
75 //custom made scan filling
76   glColor3f(sin(red),cos(green),sin(blue));
77   glBegin(GL_POINTS);
78   for(y=0;y<=s;y=y+0.1)
79   {
80    for(x=0;x<=s;x=x+0.1)
81     {
82       tr.ao=(x+x1)*cos(rad)-(y-s+y1)*sin(rad);
83       tr.bo=(x+x1)*sin(rad)+(y-s+y1)*cos(rad);
84       glVertex2f(tr.ao,tr.bo);// filling is done pixel by pixel
85     }
86   }
87   glEnd();
88   glFlush();
89   sql.xsqmem=x1; // coordinates  of the square are stored in memory
90   sql.ysqmem=y1;
91 }
```

4.2 Basic Primitives - circle.c

```
1 #include<GL/glut.h>
2 #include<time.h>
3 #include<math.h>
4 float radius=20; // for the associated math library functions
5 struct crlast    //data structure to store the prevoius positions
      of the circle
6 {
7  float xcrmem;
8  float ycrmem;
9 }crl;
10 //fcirc checks if the point it inputs is inside the circle
11 //if yes it returns a value<=0
12 float fcirc(float x,float y,float x1,float y1,float r)//circle
      filling primitive
13 {
14  return ((x1-x)*(x1-x)+(y1-y)*(y1-y)-r*r);//x^2+y^2-r^2
15 }
16 //function that draws a circle according to midpoint circle drawing
       algorithm
17 //parameter          type            function
18 //x1                 float           x coordinate of the center
      of the circle
19 //y1                 float           y coordinate of the center
      of the circle
20 //red                float           red component of the circle'
      s color
21 //green              float           green component of the
      circle's color
22 //blue               float           blue component of the circle
      's color
23 struct crlast drawCircle(float x1,float y1,float red,float green,
      float blue,float shear)
```

```
{
 int j;
 float x,y,p,rad;
 x=0;
 y=radius;  // radius of the circle
 p=1-radius; //initial value of the decision parameter
 glColor3f(sin(red),cos(green),sin(blue));
 glBegin(GL_POINTS);
 while(x<y)
 {
  if(p<0) //choose the adjacent pixel
  {
   x++;
   y=y+0;
   p=p+2*x+1;
  }
  else  //choose the lower pixel
  {
   x++;
   y--;
   p=p+2*(x-y)+1;
  }
  //first quadrant
  glVertex2f(x+x1+shear*(y+y1),y+y1);  // the other points can be
      calculated by virtue of the circle's symmetry
  glVertex2f(y+x1+shear*(x+y1),x+y1);
  //second quadrant
  glVertex2f(x+x1+shear*(-y+y1),-y+y1);
  glVertex2f(y+x1+shear*(-x+y1),-x+y1);
  //third quadrant
  glVertex2f(-x+x1+shear*(-y+y1),-y+y1);
  glVertex2f(-y+x1+shear*(-x+y1),-x+y1);
  //fourth quadrant
  glVertex2f(-x+x1+shear*(y+y1),y+y1);
  glVertex2f(-y+x1+shear*(x+y1),x+y1);
 }
 //custom made scan filling for the circle begins
 glColor3f(sin(red),cos(green),sin(blue));
 for(y=-radius+y1;y<=radius+y1;y=y+0.01)//bottom to top
  {
   for(x=-radius+x1;x<=radius+x1;x=x+0.01)//left to right
    {
    if(fcirc(x,y,x1,y1,radius)<=0)//if the point is inside the
        circle or on the boundary then fill it
    glVertex2f(x+shear*y,y);
    }
  }
 glEnd();
 glFlush();
 crl.xcrmem=x1;
 crl.ycrmem=y1;
//glutSwapBuffers();
//glutPostRedisplay();
return crl; // returning the last position of the circle
}
```

4.3 Basic Primitives - triangle.c

```
#include<GL/glut.h>//standard graphics library
```

```
2 #include<time.h>
3 #include<math.h> // for associated mathematical functions
4 #include<stdio.h>
5 float side=12.5;
6
7
8 struct rota      //data structure to hold the rotated coordinates
9 {
10  float ao,bo,co,doo;
11 }rt;
12 struct rota rotr(float x0,float y0,float x1,float y1,float rad)  //
       function to rotate the triangle
13 {
14  float rmat[3][3]={cos(rad),-sin(rad),0,sin(rad),cos(rad),0,0,0,1};
15  float vset1[3][1]={x0,y0,1};
16  float vset2[3][1]={x1,y1,1};
17  float nvset1[3][1]={0,0,0};
18  float nvset2[3][1]={0,0,0};
19  int i,j,k;
20  for(i=0;i<3;i++)
21  {
22   for(j=0;j<1;j++)
23    {
24      for(k=0;k<3;k++)
25      {
26       nvset1[i][j]+=rmat[i][k]*vset1[k][j];  //rotation by matrix
             multiplication
27       nvset2[i][j]+=rmat[i][k]*vset2[k][j];
28      }
29    }
30  }
31  rt.ao=nvset1[0][0];
32  rt.bo=nvset1[1][0];
33  rt.co=nvset2[0][0];
34  rt.doo=nvset2[1][0];
35  return rt; // structure holds the position of line ends after
       rotation
36 }
37 //functions usualplot as well specialplot are manifestations of the
       bresenham's algorithm
38 //usualplot draws lines for which 0<m<1
39 //(x1,y1)->starting coordinate of the line to be drawn
40 //dx->change along x direction
41 //dy->change along y direction
42 void usualplot(float x1,float y1,float dx,float dy,float p,float r,
      float g,float b)
43 {
44  float x=x1,y=y1,k;
45  glColor3f(sin(r),cos(g),sin(b)); //setting of colors to draw the
       line in a particular color
46  glBegin(GL_POINTS);
47  glVertex2f(x,y);
48  for(k=0;k<(dx-1);k++) //repeat the process "dx-1" times
49  {
50   if(p<0)   // choosing of the lower pixel
51   {
52   x++;
53   p=p+2*dy;
54   }
55   else      // choosing of the upper pixel
56   {
57    x++;
```

```
58      y++;
59      p=p+2*(dy-dx);
60     }
61    glVertex2f(x,y);
62    }
63  glEnd();
64  }
65  //specialplot draws lines for which -1<m<0 i.e. negative slope
66  void specialplot(float x1,float y1,float dx,float dy,float p,float
        r,float g,float b)
67  {
68    float x=x1,y=y1,k;
69    p=-2*dy+dx;  //notice the difference in the initial value of the
          decision parameter
70    glColor3f(sin(r),cos(g),sin(b));
71    glBegin(GL_POINTS); // drawing of the line point by point
72    for(k=0;k<(dx-1);k++)
73    {
74     if(p<0)       // choose the adjacent pixel
75     {
76     x++;
77     p=p-2*dy;
78     }
79     else         // choose the lower pixel
80     {
81      x++;
82      y--;
83      p=p-2*(dy-dx);
84     }
85    glVertex2f(x,y-abs(dy)/2);
86    }
87  glEnd();
88  }
89  void line(float x1,float y1,float x2,float y2,float r,float g,float
        b)
90  {
91    float p,dx,dy,k,x=x1,y=y1,m;
92    dx=x2-x1; // change in x direction
93    dy=y2-y1; // change in y direction
94    m=dy/dx; // calculation of the slope to find if usual plot or
          special plot is to be used
95    p=2*dy-dx;
96    if(m>=0)  // for positive slopes use usualplot
97    usualplot(x1,y1,dx,dy,p,r,g,b);
98    else      // for negative slopes use specialplot
99    specialplot(x1,y1,dx,dy,p,r,g,b);
100   glEnd();
101 //glutSwapBuffers();
102 //glutPostRedisplay();
103 }
104 //function to draw the triangle
105 //parameter       type          function
106 //x1              float         x coordinate part of triangle
        transalation
107 //y1              float         y coordinate part of triangle
        transalation
108 //r               float         red component of triangle's color
109 //g               float         green component of triangle's color
110 //b               float         blue component of triangle's color
111 //ang             float         angle wrt horizontal to which the
        triangle is drawn -> rotation component
```

```
112 void drawTriangle(float x1,float y1,float r,float g,float b,float
        ang)
113 {
114  float x,y,rad;
115  rad=3.14159263*ang/180; // conversion of input angle(degrees)
         parameter to radians
116  rotr(0+x1,side/(sqrt(3))+y1,side/2+x1,-side/(2*sqrt(3))+y1,rad);
         // endpoints of the line rotated
117  //line(rt.ao,rt.bo,rt.co,rt.doo,r,g,b); //line is then drawn
118  rotr(side/2+x1,-side/(2*sqrt(3))+y1,-side/2+x1,-side/(2*sqrt(3))+
         y1,rad);
119  //line(rt.ao,rt.bo,rt.co,rt.doo,r,g,b);
120  rotr(-side/2+x1,-side/(2*sqrt(3))+y1,0+x1,side/(sqrt(3))+y1,rad);
121  //line(rt.ao,rt.bo,rt.co,rt.doo,r,g,b);
122 //custom made "scan filling for the triangle" begins
123  glColor3f(sin(r),sin(g),cos(b)*sin(r*g));
124  glBegin(GL_POINTS);
125  for(y=-side/(2*sqrt(3));y<=side/(sqrt(3));y=y+0.1)
126   {
127    for(x=-side/2;x<=side/2;x=x+0.1)
128    {
129     if(sqrt(3)*x-y+side/(sqrt(3))>=0)  //the triangle boundary is
           expressed as an intersection
130      {                  //of the inequality planes 2.2x -y -3.6>0 ,
          y>=3 and -2.2x -y +31.6>0
131       if(y>=-side/(2*sqrt(3)))          //when triangle is inclined
            at an angle of 0 degrees to the horizontal
132        {
133          if(-sqrt(3)*x-y+side/(sqrt(3))>=0)
134          {
135          rt.ao=x1+(x)*cos(rad)-(y)*sin(rad);
136          rt.bo=y1+(x)*sin(rad)+(y)*cos(rad);
137          glVertex2f(rt.ao,rt.bo);
138          }
139        }
140      }
141    }
142   }
143  glEnd();
144  glFlush();
145 }
```

4.4 ScreenSaver.c

```
1 #include<GL/glut.h>
2 #include<stdio.h>
3 #include<time.h>
4 #include<math.h>
5 #include "circle.c"
6 #include "TeamName.c"
7 #include "triangle.c"
8 int counter=0;
9
10 struct randomnumbers  // a data structure to hold random color for
       the three different shapes
11 {
12  float redc,greenc,bluec,reds,blues,greens,redt,greent,bluet;
13 }rns;
14
```

```
15 struct coordinategenerator
16 {
17  int x1,y1,x2,y2,x3,y3,x4,y4;
18 }cog;
19 //function mechanism or method of working:
20 // first  psedo random number generator is seeded with a different
       value each time so that
21 // it generates a different random number
22 // the number is adjusted accordingly to suit the range of colors
23
24
25 struct randomnumbers generateRandomColor()// a function to generate
       the random colors for shapes
26 {
27  int j;
28  srand(time(0));
29  j=rand()%10;
30  rns.redc=j*10/2.6;    //red component of circle
31  srand(time(0));
32  j=rand()%10;
33  rns.greenc=j*10/3;  //green component of circle
34  srand(time(0));
35  j=rand()%10;
36  rns.bluec=j*10/7.4;     // blue component of circle
37
38
39 //square's colours
40  srand(time(0));  // seeding of the pseudo random number generator
41  j=rand()%10;               // generation of a random number in the
       range 0-9
42  rns.reds=j*10/2.6;
43  srand(time(0));
44  j=rand()%10;
45  rns.greens=j*10/3;
46  srand(time(0));
47  j=rand()%10;
48  rns.blues=j*10/7.4;
49
50 //traingles's colours
51  srand(time(0));
52  j=rand()%10;
53  rns.redt=j*10/7.8;
54  srand(time(0));
55  j=rand()%10;
56  rns.greent=j*10/3.6;
57  srand(time(0));
58  j=rand()%10;
59  rns.blues=j*10/4.5;
60  return rns;
61 }
62
63
64 void Init()//initiliase the screen dimensions
65 {
66  glClear(GL_COLOR_BUFFER_BIT);
67  glMatrixMode(GL_PROJECTION);
68  glLoadIdentity();
69  gluOrtho2D(-35,35,-35,35);
70 }
71
72
73 void smallDelay()// to create a small time delay less than 1s
```

```
74 {
75  int i,j,k;
76  for(i=0;i<=1000;i++)
77  for(j=0;j<=1000;j++);
78 }
79
80 struct coordinategenerator decideScreen()//function to decide the
       bouncing coordinates for each shape
81 {
82  int decider;
83
84  //generation of the first bounce point
85  srand(time(0));
86  cog.x1=27+(rand()%5);//very close to the rightmost corner of the
       screen
87  smallDelay();
88  smallDelay();
89  srand(time(0));
90  cog.y1=rand()%17;
91  srand(time(0)+rand()%655);
92  decider=rand()%3;
93  if(decider==0)
94  cog.y1=-cog.y1;
95  else if(decider==1)
96  cog.y1=cog.y1;
97  else
98  cog.y1=0;
99
100
101  smallDelay();
102  smallDelay();
103  srand(time(0));
104
105  //generation of
106  cog.x2=rand()%20;
107  srand(time(0)+rand()%14);
108  decider=rand()%3;
109  if(decider==0)
110  cog.x2=cog.x2;
111  else if(decider==1)
112  cog.x2=-cog.x2;
113  else
114  cog.x2=0;
115  smallDelay();
116  smallDelay();
117  srand(time(0));//make the random number seeder take different time
118  cog.y2=28+(rand()%5); //very close to the topmost corner of the
       screen
119  smallDelay();
120  smallDelay();
121  srand(time(0));
122  cog.x3=-27-(rand()%7);//very close to the leftmost corner of the
       screen
123  srand(time(0));
124  cog.y3=rand()%25;
125  smallDelay();
126  smallDelay();
127  srand(time(0));
128  decider=rand()%3;
129  if(decider==0)
130  cog.y3=0;
131  else if(decider==1)
```

```
132   cog.y3=cog.x2;
133   else
134   cog.y3=-cog.y3;
135   srand(time(0)+rand()%rand());
136   cog.y4=-27-(rand()%7);
137   smallDelay();
138   smallDelay();
139   srand(time(0));
140   cog.x4=rand()%26;
141   srand(time(0)+rand()%rand());
142   decider=rand()%3;
143   if(decider==0)
144   cog.x4=0;
145   else if(decider==1)
146   cog.x4=-cog.x2;
147   else
148   cog.x2=cog.x2;
149   return cog;
150  }
151
152
153  void linepath(float x1,float y1,float x2,float y2)//function tht
         generates points on a straight line
154  {                                                    // through (x1,y1
         ) and (x2,y2) for the shapes to move
155   static int shapeDecider=0;                          //using digital
          differential analyser algorithm
156   struct randomnumbers r;
157   radius=7;
158   r=generateRandomColor();
159   float steps,dx,dy,xinc,yinc,k,x,y,ang=0;
160   dx=x2-x1;
161   dy=y2-y1;
162
163  if(abs(dx)>abs(dy))
164   steps=abs(dx);
165   else
166   steps=abs(dy);
167   xinc=(float)dx/(float)steps;
168   yinc=(float)dy/(float)steps;
169   x=x1;
170   y=y1;
171   //shapeDecider is a variable that decides which shape will appear
          when on the screen
172
173   for(k=0;k<steps;k++)
174   {
175    glClear(GL_COLOR_BUFFER_BIT);
176    if(shapeDecider%3==0)
177    {
178     teamTheme();  // to display the team theme
179    drawCircle(x,y,r.redc,r.greenc,r.bluec,0);// to draw circle along
           the line (x1,y1) to (x2,y2)
180    glutSwapBuffers();
181    drawSquare(-x,-y,r.reds,r.greens,r.blues,0);
182    glutSwapBuffers();
183    }
184    else if(shapeDecider%3==1)
185    {
186     teamTheme();
187    glutSwapBuffers();
188    drawSquare(x,y,r.reds,r.greens,r.blues,0);
```

```
189     glutSwapBuffers();
190     drawCircle(-x,-y,r.redc,r.greenc,r.bluec,0.25);
191     smallDelay();
192     glutSwapBuffers();
193     //smallDelay();
194     }
195     else
196     {
197     teamTheme();
198     glutSwapBuffers();
199     drawTriangle(x,y,1,1,1,ang);
200     glutSwapBuffers();
201     drawCircle(-x,-y,r.redc,r.greenc,r.bluec,0);
202     glutSwapBuffers();
203     smallDelay();
204
205     }
206     x+=xinc;
207     y+=yinc;
208     ang+=100; // this parameter is to rotate the triangle
209   }
210   if(shapeDecider<25)
211   shapeDecider++;
212   else
213   shapeDecider=0;
214 }
215
216
217 void display()//display function to be called by the DisplayFunc()
        in mousecoord.c
218 {
219 struct coordinategenerator cgen;
220   while(1)
221 {
222   cgen=decideScreen();//deciding the bouncing coordinates for one
          iteration
223   linepath(cgen.x1,cgen.y1,cgen.x2,cgen.y2);//bouncing algorithm
224   counter++;
225   linepath(cgen.x2,cgen.y2,cgen.x3,cgen.y3);
226   counter++;
227   linepath(cgen.x3,cgen.y3,cgen.x4,cgen.y4);
228   counter++;
229   linepath(cgen.x4,cgen.y4,cgen.x1,cgen.y1);
230   counter++;
231   if(counter%2==0)
232   {
233     if(counter>=20)
234     counter=0;
235   s+=4;
236   }
237   if(s>=20)
238   s=12;
239   smallDelay();
240   smallDelay();
241   smallDelay();
242 }
243 }
```

4.5 TeamName.c

```
#include "square.c"
void teamTheme()
{
  int i;
  for(i=0;i<2;i++)
  {
  ddaline(-11+i,5+i,-7+i,5+i,0,0,0);
  ddaline(-9+i,5+i,-9+i,0+i,0,0,0);

  ddaline(-6+i,5+i,-2+i,5+i,0,0,0);
  ddaline(-6+i,3+i,-2+i,3+i,0,0,0);
  ddaline(-6+i,0+i,-2+i,0+i,0,0,0);
  ddaline(-6+i,5+i,-6+i,0+i,0,0,0);

  ddaline(-1+i,0+i,2+i,5+i,0,0,0);
  ddaline(5+i,0+i,2+i,5+i,0,0,0);
  ddaline(1+i,3+i,3+i,3+i,0,0,0);
  ddaline(6+i,0+i,6+i,5+i,0,0,0);
  ddaline(10+i,0+i,10+i,5+i,0,0,0);
  ddaline(10+i,5+i,8+i,3+i,0,0,0);
  ddaline(6+i,5+i,8+i,3+i,0,0,0);

  ddaline(-8+i,15+i,-18+i,-5+i,0,0,0);
  ddaline(-18+i,-5+i,-5+i,-5+i,0,0,0);
  ddaline(-8+i,-3+i,-8+i,-10+i,0,0,0);

  ddaline(-2+i,-5+i,-2+i,-10+i,0,0,0);
  ddaline(2+i,-5+i,2+i,-10+i,0,0,0);
  ddaline(-2+i,-5+i,2+i,-5+i,0,0,0);
  ddaline(-2+i,-7+i,2+i,-7+i,0,0,0);
  ddaline(-2+i,-10+i,2+i,-10+i,0,0,0);

  ddaline(5+i,-5+i,5+i,-10+i,0,0,0);
  ddaline(4+i,-5+i,6+i,-5+i,0,0,0);
  ddaline(4+i,-10+i,6+i,-10+i,0,0,0);

  ddaline(7+i,-5+i,11+i,-5+i,0,0,0);
  ddaline(9+i,-5+i,9+i,-10+i,0,0,0);

  ddaline(13+i,-5+i,17+i,-5+i,0,0,0);
  ddaline(13+i,-7+i,17+i,-7+i,0,0,0);
  ddaline(13+i,-10+i,17+i,-10+i,0,0,0);
  ddaline(13+i,-5+i,13+i,-7+i,0,0,0);
  ddaline(17+i,-7+i,17+i,-10+i,0,0,0);
}
}
```

4.6 Display File

```
0 -18.000000 8.000000 -18.000000 8.000000 10.000000 1
0 -23.000000 11.000000 -23.000000 11.000000 1.000000 2
0 -13.000000 11.000000 -13.000000 11.000000 1.000000 3
1 -18.000000 8.000000 -18.000000 8.000000 10.000000 4
2 0.000000 8.000000 -18.000000 8.000000 10.000000 5
2 2.000000 8.000000 -18.000000 8.000000 10.000000 6
2 4.000000 8.000000 -18.000000 8.000000 10.000000 7
2 6.000000 8.000000 -18.000000 8.000000 10.000000 8
2 8.000000 8.000000 -18.000000 8.000000 10.000000 9
2 10.000000 8.000000 -18.000000 8.000000 10.000000 10
```

11	2	12.000000	8.000000	−18.000000	8.000000	10.000000	11
12	2	14.000000	8.000000	−18.000000	8.000000	10.000000	12
13	2	16.000000	8.000000	−18.000000	8.000000	10.000000	13
14	2	18.000000	8.000000	−18.000000	8.000000	10.000000	14
15	2	20.000000	8.000000	−18.000000	8.000000	10.000000	15
16	2	22.000000	8.000000	−18.000000	8.000000	10.000000	16
17	2	24.000000	8.000000	−18.000000	8.000000	10.000000	17
18	2	26.000000	8.000000	−18.000000	8.000000	10.000000	18
19	2	28.000000	8.000000	−18.000000	8.000000	10.000000	19
20	2	30.000000	8.000000	−18.000000	8.000000	10.000000	20
21	2	32.000000	8.000000	−18.000000	8.000000	10.000000	21
22	2	34.000000	8.000000	−18.000000	8.000000	10.000000	22
23	2	36.000000	8.000000	−18.000000	8.000000	10.000000	23
24	2	38.000000	8.000000	−18.000000	8.000000	10.000000	24
25	2	40.000000	8.000000	−18.000000	8.000000	10.000000	25
26	2	42.000000	8.000000	−18.000000	8.000000	10.000000	26
27	2	44.000000	8.000000	−18.000000	8.000000	10.000000	27
28	2	46.000000	8.000000	−18.000000	8.000000	10.000000	28
29	2	48.000000	8.000000	−18.000000	8.000000	10.000000	29
30	2	50.000000	8.000000	−18.000000	8.000000	10.000000	30
31	2	52.000000	8.000000	−18.000000	8.000000	10.000000	31
32	2	54.000000	8.000000	−18.000000	8.000000	10.000000	32
33	2	56.000000	8.000000	−18.000000	8.000000	10.000000	33
34	2	58.000000	8.000000	−18.000000	8.000000	10.000000	34
35	2	60.000000	8.000000	−18.000000	8.000000	10.000000	35
36	2	62.000000	8.000000	−18.000000	8.000000	10.000000	36
37	2	64.000000	8.000000	−18.000000	8.000000	10.000000	37
38	2	66.000000	8.000000	−18.000000	8.000000	10.000000	38
39	3	68.000000	8.000000	−18.000000	8.000000	10.000000	39
40	3	0.000000	8.000000	−18.000000	8.000000	10.000000	40
41	3	2.000000	8.000000	−18.000000	8.000000	10.000000	41
42	3	4.000000	8.000000	−18.000000	8.000000	10.000000	42
43	3	6.000000	8.000000	−18.000000	8.000000	10.000000	43
44	3	8.000000	8.000000	−18.000000	8.000000	10.000000	44
45	3	10.000000	8.000000	−18.000000	8.000000	10.000000	45
46	3	12.000000	8.000000	−18.000000	8.000000	10.000000	46
47	3	14.000000	8.000000	−18.000000	8.000000	10.000000	47
48	3	16.000000	8.000000	−18.000000	8.000000	10.000000	48
49	3	18.000000	8.000000	−18.000000	8.000000	10.000000	49
50	3	20.000000	8.000000	−18.000000	8.000000	10.000000	50
51	3	22.000000	8.000000	−18.000000	8.000000	10.000000	51
52	3	24.000000	8.000000	−18.000000	8.000000	10.000000	52
53	3	26.000000	8.000000	−18.000000	8.000000	10.000000	53
54	3	28.000000	8.000000	−18.000000	8.000000	10.000000	54
55	3	30.000000	8.000000	−18.000000	8.000000	10.000000	55
56	3	32.000000	8.000000	−18.000000	8.000000	10.000000	56
57	3	34.000000	8.000000	−18.000000	8.000000	10.000000	57
58	3	36.000000	8.000000	−18.000000	8.000000	10.000000	58
59	3	38.000000	8.000000	−18.000000	8.000000	10.000000	59
60	3	40.000000	8.000000	−18.000000	8.000000	10.000000	60
61	3	42.000000	8.000000	−18.000000	8.000000	10.000000	61
62	3	44.000000	8.000000	−18.000000	8.000000	10.000000	62
63	3	46.000000	8.000000	−18.000000	8.000000	10.000000	63
64	3	48.000000	8.000000	−18.000000	8.000000	10.000000	64
65	3	50.000000	8.000000	−18.000000	8.000000	10.000000	65
66	3	52.000000	8.000000	−18.000000	8.000000	10.000000	66
67	3	54.000000	8.000000	−18.000000	8.000000	10.000000	67
68	3	56.000000	8.000000	−18.000000	8.000000	10.000000	68
69	3	58.000000	8.000000	−18.000000	8.000000	10.000000	69
70	3	60.000000	8.000000	−18.000000	8.000000	10.000000	70
71	3	62.000000	8.000000	−18.000000	8.000000	10.000000	71
72	3	64.000000	8.000000	−18.000000	8.000000	10.000000	72

73	3	66.000000	8.000000	−18.000000	8.000000	10.000000	73
74	4	68.000000	8.000000	−18.000000	8.000000	10.000000	74
75	4	0.000000	8.000000	−18.000000	8.000000	10.000000	75
76	4	2.000000	8.000000	−18.000000	8.000000	10.000000	76
77	4	4.000000	8.000000	−18.000000	8.000000	10.000000	77
78	4	6.000000	8.000000	−18.000000	8.000000	10.000000	78
79	4	8.000000	8.000000	−18.000000	8.000000	10.000000	79
80	4	10.000000	8.000000	−18.000000	8.000000	10.000000	80
81	4	12.000000	8.000000	−18.000000	8.000000	10.000000	81
82	4	14.000000	8.000000	−18.000000	8.000000	10.000000	82
83	4	16.000000	8.000000	−18.000000	8.000000	10.000000	83
84	4	18.000000	8.000000	−18.000000	8.000000	10.000000	84
85	4	20.000000	8.000000	−18.000000	8.000000	10.000000	85
86	4	22.000000	8.000000	−18.000000	8.000000	10.000000	86
87	4	24.000000	8.000000	−18.000000	8.000000	10.000000	87
88	4	26.000000	8.000000	−18.000000	8.000000	10.000000	88
89	4	28.000000	8.000000	−18.000000	8.000000	10.000000	89
90	4	30.000000	8.000000	−18.000000	8.000000	10.000000	90
91	4	32.000000	8.000000	−18.000000	8.000000	10.000000	91
92	4	34.000000	8.000000	−18.000000	8.000000	10.000000	92
93	4	36.000000	8.000000	−18.000000	8.000000	10.000000	93
94	4	38.000000	8.000000	−18.000000	8.000000	10.000000	94
95	4	40.000000	8.000000	−18.000000	8.000000	10.000000	95
96	4	42.000000	8.000000	−18.000000	8.000000	10.000000	96
97	4	44.000000	8.000000	−18.000000	8.000000	10.000000	97
98	4	46.000000	8.000000	−18.000000	8.000000	10.000000	98
99	4	48.000000	8.000000	−18.000000	8.000000	10.000000	99
100	4	50.000000	8.000000	−18.000000	8.000000	10.000000	100
101	4	52.000000	8.000000	−18.000000	8.000000	10.000000	101
102	4	54.000000	8.000000	−18.000000	8.000000	10.000000	102
103	4	56.000000	8.000000	−18.000000	8.000000	10.000000	103
104	4	58.000000	8.000000	−18.000000	8.000000	10.000000	104
105	4	60.000000	8.000000	−18.000000	8.000000	10.000000	105
106	4	62.000000	8.000000	−18.000000	8.000000	10.000000	106
107	4	64.000000	8.000000	−18.000000	8.000000	10.000000	107
108	4	66.000000	8.000000	−18.000000	8.000000	10.000000	108
109	5	68.000000	8.000000	−18.000000	8.000000	10.000000	109
110	5	0.000000	8.000000	−18.000000	8.000000	10.000000	110
111	5	2.000000	8.000000	−18.000000	8.000000	10.000000	111
112	5	4.000000	8.000000	−18.000000	8.000000	10.000000	112
113	5	6.000000	8.000000	−18.000000	8.000000	10.000000	113
114	5	8.000000	8.000000	−18.000000	8.000000	10.000000	114
115	5	10.000000	8.000000	−18.000000	8.000000	10.000000	115
116	5	12.000000	8.000000	−18.000000	8.000000	10.000000	116
117	5	14.000000	8.000000	−18.000000	8.000000	10.000000	117
118	5	16.000000	8.000000	−18.000000	8.000000	10.000000	118
119	5	18.000000	8.000000	−18.000000	8.000000	10.000000	119
120	5	20.000000	8.000000	−18.000000	8.000000	10.000000	120
121	5	22.000000	8.000000	−18.000000	8.000000	10.000000	121
122	5	24.000000	8.000000	−18.000000	8.000000	10.000000	122
123	5	26.000000	8.000000	−18.000000	8.000000	10.000000	123
124	5	28.000000	8.000000	−18.000000	8.000000	10.000000	124
125	5	30.000000	8.000000	−18.000000	8.000000	10.000000	125
126	5	32.000000	8.000000	−18.000000	8.000000	10.000000	126
127	5	34.000000	8.000000	−18.000000	8.000000	10.000000	127
128	5	36.000000	8.000000	−18.000000	8.000000	10.000000	128
129	5	38.000000	8.000000	−18.000000	8.000000	10.000000	129
130	5	40.000000	8.000000	−18.000000	8.000000	10.000000	130
131	5	42.000000	8.000000	−18.000000	8.000000	10.000000	131
132	5	44.000000	8.000000	−18.000000	8.000000	10.000000	132
133	5	46.000000	8.000000	−18.000000	8.000000	10.000000	133
134	5	48.000000	8.000000	−18.000000	8.000000	10.000000	134

135	5	50.000000	8.000000	−18.000000	8.000000	10.000000	135
136	5	52.000000	8.000000	−18.000000	8.000000	10.000000	136
137	5	54.000000	8.000000	−18.000000	8.000000	10.000000	137
138	5	56.000000	8.000000	−18.000000	8.000000	10.000000	138
139	5	58.000000	8.000000	−18.000000	8.000000	10.000000	139
140	5	60.000000	8.000000	−18.000000	8.000000	10.000000	140
141	5	62.000000	8.000000	−18.000000	8.000000	10.000000	141
142	5	64.000000	8.000000	−18.000000	8.000000	10.000000	142
143	5	66.000000	8.000000	−18.000000	8.000000	10.000000	143
144	6	68.000000	8.000000	−18.000000	8.000000	10.000000	144
145	6	0.000000	8.000000	−18.000000	8.000000	10.000000	145
146	6	2.000000	8.000000	−18.000000	8.000000	10.000000	146
147	6	4.000000	8.000000	−18.000000	8.000000	10.000000	147
148	6	6.000000	8.000000	−18.000000	8.000000	10.000000	148
149	6	8.000000	8.000000	−18.000000	8.000000	10.000000	149
150	6	10.000000	8.000000	−18.000000	8.000000	10.000000	150
151	6	12.000000	8.000000	−18.000000	8.000000	10.000000	151
152	6	14.000000	8.000000	−18.000000	8.000000	10.000000	152
153	6	16.000000	8.000000	−18.000000	8.000000	10.000000	153
154	6	18.000000	8.000000	−18.000000	8.000000	10.000000	154
155	6	20.000000	8.000000	−18.000000	8.000000	10.000000	155
156	6	22.000000	8.000000	−18.000000	8.000000	10.000000	156
157	6	24.000000	8.000000	−18.000000	8.000000	10.000000	157
158	6	26.000000	8.000000	−18.000000	8.000000	10.000000	158
159	6	28.000000	8.000000	−18.000000	8.000000	10.000000	159
160	6	30.000000	8.000000	−18.000000	8.000000	10.000000	160
161	6	32.000000	8.000000	−18.000000	8.000000	10.000000	161
162	6	34.000000	8.000000	−18.000000	8.000000	10.000000	162
163	6	36.000000	8.000000	−18.000000	8.000000	10.000000	163
164	6	38.000000	8.000000	−18.000000	8.000000	10.000000	164
165	6	40.000000	8.000000	−18.000000	8.000000	10.000000	165
166	6	42.000000	8.000000	−18.000000	8.000000	10.000000	166
167	6	44.000000	8.000000	−18.000000	8.000000	10.000000	167
168	6	46.000000	8.000000	−18.000000	8.000000	10.000000	168
169	6	48.000000	8.000000	−18.000000	8.000000	10.000000	169
170	6	50.000000	8.000000	−18.000000	8.000000	10.000000	170
171	6	52.000000	8.000000	−18.000000	8.000000	10.000000	171
172	6	54.000000	8.000000	−18.000000	8.000000	10.000000	172
173	6	56.000000	8.000000	−18.000000	8.000000	10.000000	173
174	6	58.000000	8.000000	−18.000000	8.000000	10.000000	174
175	6	60.000000	8.000000	−18.000000	8.000000	10.000000	175
176	6	62.000000	8.000000	−18.000000	8.000000	10.000000	176
177	6	64.000000	8.000000	−18.000000	8.000000	10.000000	177
178	6	66.000000	8.000000	−18.000000	8.000000	10.000000	178
179	7	68.000000	8.000000	−18.000000	8.000000	10.000000	179
180	7	0.000000	8.000000	−18.000000	8.000000	10.000000	180
181	7	2.000000	8.000000	−18.000000	8.000000	10.000000	181
182	7	4.000000	8.000000	−18.000000	8.000000	10.000000	182
183	7	6.000000	8.000000	−18.000000	8.000000	10.000000	183
184	7	8.000000	8.000000	−18.000000	8.000000	10.000000	184
185	7	10.000000	8.000000	−18.000000	8.000000	10.000000	185
186	7	12.000000	8.000000	−18.000000	8.000000	10.000000	186
187	7	14.000000	8.000000	−18.000000	8.000000	10.000000	187
188	7	16.000000	8.000000	−18.000000	8.000000	10.000000	188
189	7	18.000000	8.000000	−18.000000	8.000000	10.000000	189
190	7	20.000000	8.000000	−18.000000	8.000000	10.000000	190
191	7	22.000000	8.000000	−18.000000	8.000000	10.000000	191
192	7	24.000000	8.000000	−18.000000	8.000000	10.000000	192
193	7	26.000000	8.000000	−18.000000	8.000000	10.000000	193
194	7	28.000000	8.000000	−18.000000	8.000000	10.000000	194
195	7	30.000000	8.000000	−18.000000	8.000000	10.000000	195
196	7	32.000000	8.000000	−18.000000	8.000000	10.000000	196

197	7	34.000000	8.000000	−18.000000	8.000000	10.000000	197
198	7	36.000000	8.000000	−18.000000	8.000000	10.000000	198
199	7	38.000000	8.000000	−18.000000	8.000000	10.000000	199
200	7	40.000000	8.000000	−18.000000	8.000000	10.000000	200
201	7	42.000000	8.000000	−18.000000	8.000000	10.000000	201
202	7	44.000000	8.000000	−18.000000	8.000000	10.000000	202
203	7	46.000000	8.000000	−18.000000	8.000000	10.000000	203
204	7	48.000000	8.000000	−18.000000	8.000000	10.000000	204
205	7	50.000000	8.000000	−18.000000	8.000000	10.000000	205
206	7	52.000000	8.000000	−18.000000	8.000000	10.000000	206
207	7	54.000000	8.000000	−18.000000	8.000000	10.000000	207
208	7	56.000000	8.000000	−18.000000	8.000000	10.000000	208
209	7	58.000000	8.000000	−18.000000	8.000000	10.000000	209
210	7	60.000000	8.000000	−18.000000	8.000000	10.000000	210
211	7	62.000000	8.000000	−18.000000	8.000000	10.000000	211
212	7	64.000000	8.000000	−18.000000	8.000000	10.000000	212
213	7	66.000000	8.000000	−18.000000	8.000000	10.000000	213
214	8	68.000000	8.000000	−18.000000	8.000000	10.000000	214
215	8	0.000000	8.000000	−18.000000	8.000000	10.000000	215
216	8	2.000000	8.000000	−18.000000	8.000000	10.000000	216
217	8	4.000000	8.000000	−18.000000	8.000000	10.000000	217
218	8	6.000000	8.000000	−18.000000	8.000000	10.000000	218
219	8	8.000000	8.000000	−18.000000	8.000000	10.000000	219
220	8	10.000000	8.000000	−18.000000	8.000000	10.000000	220
221	8	12.000000	8.000000	−18.000000	8.000000	10.000000	221
222	8	14.000000	8.000000	−18.000000	8.000000	10.000000	222
223	8	16.000000	8.000000	−18.000000	8.000000	10.000000	223
224	8	18.000000	8.000000	−18.000000	8.000000	10.000000	224
225	8	20.000000	8.000000	−18.000000	8.000000	10.000000	225
226	8	22.000000	8.000000	−18.000000	8.000000	10.000000	226
227	8	24.000000	8.000000	−18.000000	8.000000	10.000000	227
228	8	26.000000	8.000000	−18.000000	8.000000	10.000000	228
229	8	28.000000	8.000000	−18.000000	8.000000	10.000000	229
230	8	30.000000	8.000000	−18.000000	8.000000	10.000000	230
231	8	32.000000	8.000000	−18.000000	8.000000	10.000000	231
232	8	34.000000	8.000000	−18.000000	8.000000	10.000000	232
233	8	36.000000	8.000000	−18.000000	8.000000	10.000000	233
234	8	38.000000	8.000000	−18.000000	8.000000	10.000000	234
235	8	40.000000	8.000000	−18.000000	8.000000	10.000000	235
236	8	42.000000	8.000000	−18.000000	8.000000	10.000000	236
237	8	44.000000	8.000000	−18.000000	8.000000	10.000000	237
238	8	46.000000	8.000000	−18.000000	8.000000	10.000000	238
239	8	48.000000	8.000000	−18.000000	8.000000	10.000000	239
240	8	50.000000	8.000000	−18.000000	8.000000	10.000000	240
241	8	52.000000	8.000000	−18.000000	8.000000	10.000000	241
242	8	54.000000	8.000000	−18.000000	8.000000	10.000000	242
243	8	56.000000	8.000000	−18.000000	8.000000	10.000000	243
244	8	58.000000	8.000000	−18.000000	8.000000	10.000000	244
245	8	60.000000	8.000000	−18.000000	8.000000	10.000000	245
246	8	62.000000	8.000000	−18.000000	8.000000	10.000000	246
247	8	64.000000	8.000000	−18.000000	8.000000	10.000000	247
248	8	66.000000	8.000000	−18.000000	8.000000	10.000000	248
249	9	68.000000	8.000000	−18.000000	8.000000	10.000000	249
250	9	0.000000	8.000000	−18.000000	8.000000	10.000000	250
251	9	2.000000	8.000000	−18.000000	8.000000	10.000000	251
252	9	4.000000	8.000000	−18.000000	8.000000	10.000000	252
253	9	6.000000	8.000000	−18.000000	8.000000	10.000000	253
254	9	8.000000	8.000000	−18.000000	8.000000	10.000000	254
255	9	10.000000	8.000000	−18.000000	8.000000	10.000000	255
256	9	12.000000	8.000000	−18.000000	8.000000	10.000000	256
257	9	14.000000	8.000000	−18.000000	8.000000	10.000000	257
258	9	16.000000	8.000000	−18.000000	8.000000	10.000000	258

```
259 9 18.000000 8.000000 -18.000000 8.000000 10.000000 259
260 9 20.000000 8.000000 -18.000000 8.000000 10.000000 260
261 9 22.000000 8.000000 -18.000000 8.000000 10.000000 261
262 9 24.000000 8.000000 -18.000000 8.000000 10.000000 262
263 9 26.000000 8.000000 -18.000000 8.000000 10.000000 263
264 9 28.000000 8.000000 -18.000000 8.000000 10.000000 264
265 9 30.000000 8.000000 -18.000000 8.000000 10.000000 265
266 9 32.000000 8.000000 -18.000000 8.000000 10.000000 266
267 9 34.000000 8.000000 -18.000000 8.000000 10.000000 267
268 9 36.000000 8.000000 -18.000000 8.000000 10.000000 268
269 9 38.000000 8.000000 -18.000000 8.000000 10.000000 269
270 9 40.000000 8.000000 -18.000000 8.000000 10.000000 270
271 9 42.000000 8.000000 -18.000000 8.000000 10.000000 271
272 9 44.000000 8.000000 -18.000000 8.000000 10.000000 272
273 9 46.000000 8.000000 -18.000000 8.000000 10.000000 273
274 9 48.000000 8.000000 -18.000000 8.000000 10.000000 274
275 9 50.000000 8.000000 -18.000000 8.000000 10.000000 275
276 9 52.000000 8.000000 -18.000000 8.000000 10.000000 276
277 9 54.000000 8.000000 -18.000000 8.000000 10.000000 277
278 9 56.000000 8.000000 -18.000000 8.000000 10.000000 278
279 9 58.000000 8.000000 -18.000000 8.000000 10.000000 279
280 9 60.000000 8.000000 -18.000000 8.000000 10.000000 280
281 9 62.000000 8.000000 -18.000000 8.000000 10.000000 281
282 9 64.000000 8.000000 -18.000000 8.000000 10.000000 282
283 9 66.000000 8.000000 -18.000000 8.000000 10.000000 283
284 10 68.000000 8.000000 -18.000000 8.000000 10.000000 284
```

4.7 FileFormats.txt

```
1 File Formats:
2
3 DispFile : test.txt
4 opCode x0 y0 x y r obID
```

4.8 Main File - OpeningScreen.c

```
1 #include<GL/glut.h>
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include "ScreenSaver.c"
5 #define pi 3.14159263
6 #include<string.h>
7
8 //Global declarations and prototypes
9 int count = 0;
10 int xa,ya,xb,yb,xc,yc,xd,yd;
11
12 void motion(int x, int y);  // a rubberbanding procedure
13
14 int glbx,glby,id;//id is id number of a window where we 'play with
       Mr.smiles'
15 float smred=1,smgreen=1,smblue=0;
16
17 float b_red=25, b_green=25, b_blue=25;
18 int prtw1=0;
19
```

```
20 struct pos{   // data structure used for rubber banding
21 float x;
22 float y;
23 int s;
24 };
25 typedef struct pos pos;
26
27 pos strt, fin;
28
29 struct color{
30
31 float r;
32 float g;
33 float b;
34 };
35
36 typedef struct color color;
37
38 color red, green, blue;
39
40 void colorInit()
41 {
42 red.r = asin(1);  // a red colour
43 red.g = acos(0);
44 red.b = asin(0);
45
46 green.r = asin(0);//a green colour
47 green.g = acos(1);
48 green.b = asin(0);
49
50 blue.r = asin(0); // a blue colour
51 blue.g = acos(0);
52 blue.b = asin(1);
53
54 }
55
56 struct button{ // buttons made
57
58 int w1;//button width
59 int w2;
60 int h1;//button height
61 int h2;
62 };
63
64 typedef struct button button;
65
66 button b1, b2, b3,b4;
67 //j is the obId
68
69
70
71 void triangularBoundary(int j,int prt)//to draw the triangluar
       boundary
72 {
73  glColor3f(b_green,b_blue,b_red);
74  if(j==2)
75  {
76  //for(prt=0; prt < 70; prt=prt+2)//loop from left to right at top
77  //{
78  glBegin(GL_POLYGON);
79  glVertex2f(-35+prt,35);
80  glVertex2f(-33+prt,33);
```

```
81   glVertex2f(-35+prt,31);
82   glEnd();
83   //}
84
85   }
86   if(j==3)
87   {
88   //for(prt=0; prt < 70; prt=prt+2)//loop from top to bottom at
         right
89   //{
90   glBegin(GL_POLYGON);
91   glVertex2f(35,-35+prt);
92   glVertex2f(33,-33+prt);
93   glVertex2f(31,-35+prt);
94   glEnd();
95   //}
96   }
97   if(j==4)
98  {
99   //for(prt=0; prt < 70; prt=prt+2)//loop from left to right at top
100  //{
101  glBegin(GL_POLYGON);
102  glVertex2f(-35+prt,-35);
103  glVertex2f(-33+prt,-33);
104  glVertex2f(-35+prt,-31);
105  glEnd();
106  //}
107  }
108  if(j==5)
109 {
110  //for(prt=0; prt < 70; prt=prt+2)//loop from left to right at top
111  //{
112  glBegin(GL_POLYGON);
113  glVertex2f(-35,-35+prt);
114  glVertex2f(-33,-33+prt);
115  glVertex2f(-31,-35+prt);
116  glEnd();
117  //}
118 }
119  glFlush();
120
121 }
122
123
124
125
126 void buttonInit()
127 {
128 //Initialize Buttons
129 b1.w1=-10;
130 b1.w2=10;
131 b1.h1=-28;
132 b1.h2=-24;
133
134
135 b2.w1=-10;
136 b2.w2=10;
137 b2.h1=-23;
138 b2.h2=-19;
139
140 b3.w1 = -10;
141 b3.w2 = 10;
```

```
142 b3.h1 = -33;
143 b3.h2 = -29;
144
145
146 b4.w1=-10;
147 b4.w2=10;
148 b4.h1=-28;
149 b4.h2=-24;
150
151 //Buttons are initialized
152 }
153
154
155 int i;
156
157 struct dispFile
158 {
159     int op; //tells whether circle(0) triangle(2) or arc(1) is to
        be drawn (OpCodes)
160     float x0; // center in case of circle endpt in case of a line
161     float y0;
162     float x;
163     float y;
164     float r;  //radius
165     int obID; //object ID
166
167 };
168
169
170 void drawarc(float xa,float ya,float rd,int callfrom)//function to
    draw an arc for given centre (xa,ya)
171 {
172   float k=rd-2;
173   float j,rad,r;
174
175   if(rd>13&&callfrom==1)
176   ya=ya-22;
177   if((ya>0||callfrom==0)||ya<0&&rd<13)//happy smiley face
178   {
179   for(r=k;r<=k+rd/20;r+=0.01)
180   {
181   glBegin(GL_LINES);
182   for(j=210;j<=330;j+=0.1)
183   {
184    rad=j*3.14159263/180;
185    glVertex2f(xa+r*cos(rad),ya+r*sin(rad));
186   }
187  glEnd();
188  }
189  }
190  else if(ya<0&&callfrom==1)//sad smiley face
191  {
192   for(r=k;r<=k+rd/20;r+=0.01)
193   {
194   glBegin(GL_LINES);
195   for(j=50;j<=130;j+=0.1)
196   {
197    rad=j*3.14159263/180;
198    glVertex2f(xa+r*cos(rad),ya+r*sin(rad));
199   }
200  glEnd();
201  }
```

```
202   }
203  glFlush();
204  }
205
206
207  void dispWrite()
208  {
209      int prt=0; //For the triangular boundary
210      int i,k=5;
211      FILE* fd=NULL;
212      struct dispFile d;
213      fd = fopen("smidisplay.txt","w");
214      for(i=1;i<=284;i++) //4 + 70*(4)
215       {
216        if(i<4)
217        d.op=0;
218
219        if(i==1)//face of the smiley
220        {
221        d.x0=glbx;
222        d.y0=glby;
223        d.x=glbx;
224        d.y=glby;
225        d.r=10;
226        d.obID=1;
227        }
228        if(i==2)//left eye
229        {
230        d.x0=glbx-5;
231        d.y0=glby+8-5;
232        d.x=glbx-5;
233        d.y=glby+8-5;
234        d.r=1;
235        d.obID=2;
236        }
237        if(i==3)//right eye
238        {
239        d.x0=glbx+5;
240        d.y0=glby+8-5;
241        d.x=glbx+5;
242        d.y=glby+8-5;
243        d.r=1;
244        d.obID=3;
245        }
246        if(i==4)//mouth of the smiley
247        {
248        d.op=1;
249        d.x0=glbx;
250        d.y0=glby;
251        d.x=glbx;
252        d.y=glby;
253        d.r=10;
254        d.obID=4;
255        }
256
257        if(i>4)//triangular boundary
258        {
259        if(i==5)
260        d.op=2;
261
262        d.x0=prt;
263        d.y0=glby;
```

```
264         d.x=glbx;
265         d.y=glby;
266         d.r=10;
267
268         prt+=2;
269         if(prt>=70)
270         {
271           prt=0;
272           d.op+=1;
273         }
274
275         d.obID=k;
276         k++;
277         }
278         fprintf(fd,"%d %f %f %f %f %f %d\n",d.op,d.x0,d.y0,d.x,d.y,d.
                r,d.obID);//write into the file
279       }
280    fclose(fd);
281 }
282
283 void button2Draw()
284 {
285
286  glColor3f(b_red,b_green,b_blue);
287  glBegin(GL_POLYGON);
288  glVertex2f(b2.w1,b2.h1);
289  glVertex2f(b2.w2,b2.h1);
290  glVertex2f(b2.w2,b2.h2);
291  glVertex2f(b2.w1,b2.h2);
292  glEnd();
293
294
295 //Writing Text
296 glColor3f(0,0,0);
297 char text2[]="Smiley Color";
298 glRasterPos2i(-8,-21);
299 for(i = 0; i < strlen(text2); i++)
300 {
301  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text2[i]);
302 }
303 glFlush();
304 }
305
306
307
308
309 void startscreen()
310 {
311
312   FILE* fp;
313  struct dispFile d;
314  fp=fopen("smidisplay.txt","r");
315  if(fp==NULL)
316  printf("\nError in reading the file\n");
317  else
318  {
319  glClear(GL_COLOR_BUFFER_BIT);
320  while(fscanf(fp,"%d%f%f%f%f%f%d\n",&d.op,&d.x0,&d.y0,&d.x,&d.y,&d.
         r,&d.obID)!=EOF)//read until end of file
321  {
322  switch(d.op)
323  {
```

```
 case 0:
 radius=d.r;
 if(radius>2)
 drawCircle(d.x0,d.y0,smred,smgreen,smblue,0);  //This is the face
     of the smiley
 else
 drawCircle(d.x0,d.y0,0,pi/2,0,0);  // This draws the 2 eyes
 break;
 case 1:  //for drawing an arc opcode is 1
 drawarc(d.x0,d.y0,10,0);
 break;
 default :
 triangularBoundary(d.op,d.x0); //d.x0 contains prt
 break;
 }
 }

//Button 1
 glColor3f(1,1,1);//start screen saver button
 glBegin(GL_POLYGON);
 glVertex2f(b1.w1,b1.h1);
 glVertex2f(b1.w2,b1.h1);
 glVertex2f(b1.w2,b1.h2);
 glVertex2f(b1.w1,b1.h2);
 glEnd();


//Writing Text
glColor3f(0,0,0);
glRasterPos2i(-9,-26);
char text[]="Start Screen Saver";
 for(i = 0; i < strlen(text); i++)
{
 glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
}

//Button 2
button2Draw();//drawn separetely to show smiley color
    instantaneosly when the button is clicked

//Button 3
 glColor3f(1,1,1);//play with Mr.smiles
 glBegin(GL_POLYGON);
 glVertex2f(b3.w1,b3.h1);
 glVertex2f(b3.w2,b3.h1);
 glVertex2f(b3.w2,b3.h2);
 glVertex2f(b3.w1,b3.h2);
 glEnd();

 //Writing Text
glColor3f(0,0,0);
glRasterPos2i(-9,(b3.h1+b3.h2)/2);
char text3[]="Play With Mr.Smiles";

 for(i = 0; i < strlen(text3); i++)
{
 glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text3[i]);
}

glFlush();
 }
}
```

```
384
385 void smileyMouse(int button,int state,int x,int y)//mouse callback
        for smiley play window
386 {
387
388
389  if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
390  {
391
392
393   y=600-y;  //Converting from Computer perspective to Human
          Perspective
394
395
396 //Push each value based on its Quadrant
397    if(x==300)
398    x=0;
399
400    else if(x>300)
401    x=x-300;
402
403    else
404    x=-(300-x);
405
406    if(y==300)
407    y=0;
408
409    else if(y>300)
410    y=y-300;
411
412    else
413    y=-(300-y);
414
415    glbx=x/8.57;
416    glby=y/8.57;
417
418    if((glbx>=b4.w1&&glbx<=b4.w2)&&(glby>=b4.h1&&glby<=b4.h2))//exit
          play
419    {
420     if(button==GLUT_LEFT_BUTTON && state ==GLUT_DOWN)
421       glutDestroyWindow(id);
422    }
423
424    else{
425    glEnable(GL_COLOR_LOGIC_OP);
426    glLogicOp(GL_XOR);
427    strt.x=glbx;
428    strt.y=glby;
429    fin.x=glbx;
430    fin.y=glby;
431    }
432    }
433  }
434
435  void smileyDisplay()
436  {
437  int i;
438   glColor3f(1,1,1);
439  glBegin(GL_POLYGON);
440  glVertex2f(b4.w1,b4.h1);
441  glVertex2f(b4.w2,b4.h1);
442  glVertex2f(b4.w2,b4.h2);
```

```
443   glVertex2f(b4.w1,b4.h2);
444   glEnd();
445   //Writing Text
446  glColor3f(0,0,0);
447  glRasterPos2i(-9,(b4.h1+b4.h2)/2);
448  char text4[]="Exit play";
449   for(i = 0; i < strlen(text4); i++)
450   {
451    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text4[i]);
452   }
453   glFlush();
454   }
455
456  void clipWindow(int xa, int ya, int xb, int yb, int xc, int yc, int
         xd, int yd)
457  {
458
459  glColor3f(0,0,0);
460  glBegin(GL_POLYGON);
461  glVertex2f(xa,ya);
462  glVertex2f(xb,yb);
463  glVertex2f(xc,yc);
464  glVertex2f(xd,yd);
465  glEnd();
466  glFlush();
467
468  }
469
470
471  void mymouse(int button,int state,int x, int y) //Mouse handler for
         positioning smiley on opening screen
472  {
473
474  int k;
475  static int flag=0;
476
477   if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
478   {
479
480
481    y=600-y;  //Converting from Computer perspective to Human
         Perspective
482
483
484  //Push each value based on its Quadrant
485     if(x==300)
486     x=0;
487
488     else if(x>300)
489     x=x-300;
490
491     else
492     x=-(300-x);
493
494     if(y==300)
495     y=0;
496
497     else if(y>300)
498     y=y-300;
499
500     else
501     y=-(300-y);
```

```
502
503     glbx=x/8.57;
504     glby=y/8.57;
505
506
507     if((glbx>=b1.w1&&glbx<=b1.w2)&&(glby>=b1.h1&&glby<=b1.h2))
508     {
509     glutInitWindowSize(600,600);
510     glutCreateWindow("Screen saver");
511     Init();
512     glutDisplayFunc(display);
513     //glutKeyboardFunc(keyboard);
514     }
515
516     else if((glbx>=b3.w1&&glbx<=b3.w2)&&(glby>=b3.h1&&glby<=b3.h2))
517     {
518     glutInitWindowSize(600,600);
519     id=glutCreateWindow("Smiley Play");
520     Init();
521     glutDisplayFunc(smileyDisplay);
522     glutMouseFunc(smileyMouse);
523     //glutKeyboardFunc(keyboard);
524     glutMotionFunc(motion);
525     }
526
527     else if((glbx>=b1.w1&&glbx<=b2.w2)&&(glby>=b2.h1&&glby<=b2.h2))
528     {
529
530     srand(time(0)+rand());
531     k = rand()%50;
532
533
534   if(k%3==0)
535  {
536   b_red = red.r;
537   b_green = red.g;
538   b_blue = red.b;
539  }
540
541  else if(k%3==1)
542  {
543   b_red = green.r;
544   b_green = green.g;
545   b_blue = green.b;
546  }
547
548  else
549  {
550   b_red = blue.r;
551   b_green = blue.g;
552   b_blue = blue.b;
553  }
554
555
556  smred = b_red;
557  smgreen = b_green;
558  smblue = b_blue;
559
560
561     b_red = sin(b_red);
562     b_green = cos(b_green);
563     b_blue = sin(b_blue);
```

```
564
565 //printf("r : %f\t g : %f\t b : %f\n",b_red, b_green, b_blue);
566
567    button2Draw();
568
569    }
570    else
571    {
572    dispWrite();
573    startscreen();
574    }
575  }
576
577    else if(button==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
578 {
579
580    y=600-y;  //Converting from Computer perspective to Human
          Perspective
581
582
583 //Push each value based on its Quadrant
584    if(x==300)
585    x=0;
586
587    else if(x>300)
588    x=x-300;
589
590    else
591    x=-(300-x);
592
593    if(y==300)
594    y=0;
595
596    else if(y>300)
597    y=y-300;
598
599    else
600    y=-(300-y);
601
602    x=x/8.57;
603    y=y/8.57;
604
605  if(count==0)
606 {
607    if(flag==1)
608 {
609    startscreen();
610    flag=0;
611 }
612
613    xa = x;
614    ya = y;
615    count++;
616    radius = 0.5;
617    drawCircle(xa,ya,asin(1),acos(1),asin(1),0);
618 }
619
620  else if(count==1)
621 {
622    xb = x;
623    yb = y;
624    count++;
```

```
625     radius = 0.5;
626     ddaline(xa,ya,xb,yb,asin(1),acos(0),asin(0));
627     drawCircle(xb,yb,asin(1),acos(1),asin(1),0);
628 }
629
630  else if(count==2)
631 {
632     xc = x;
633     yc = y;
634     count++;
635     radius = 0.5;
636     ddaline(xb,yb,xc,yc,asin(1),acos(0),asin(0));
637     drawCircle(xc,yc,asin(1),acos(1),asin(1),0);
638 }
639
640  else if(count==3)
641 {
642     xd = x;
643     yd = y;
644
645     radius = 0.5;
646     ddaline(xc,yc,xd,yd,asin(1),acos(0),asin(0));
647     ddaline(xd,yd,xa,ya,asin(1),acos(0),asin(0));
648     drawCircle(xd,yd,asin(1),acos(1),asin(1),0);
649     flag = 1;
650     count = 0;
651
652 }
653
654
655 }
656
657 }
658
659 void keyboard(unsigned char key, int x, int y)
660 {
661    if(key=='c')
662 {
663  startscreen();
664  clipWindow(xa,ya,xb,yb,xc,yc,xd,yd);
665 }
666
667 }
668
669
670 void drawSmiley(float xc, float yc, float r)
671 {
672 drawCircle(xc,yc,pi/2,0,0,0);
673 radius = r/10;
674 drawCircle(xc-r/2,yc+r/3,pi/2,0,pi/2,0);
675 radius=r/10;
676 drawCircle(xc+r/2,yc+r/3,pi/2,0,pi/2,0);
677 drawarc(xc,yc,r,1);
678
679 glFlush();
680 }
681
682 void motion(int x, int y)
683 {
684 float xc, yc, myradius;
685 //Repeat the coordinate conversions
686
```

```
687   y=600-y;  //Converting from Computer perspective to Human
          Perspective
688
689
690 //Push each value based on its Quadrant
691     if(x==300)
692     x=0;
693
694     else if(x>300)
695     x=x-300;
696
697     else
698     x=-(300-x);
699
700     if(y==300)
701     y=0;
702
703     else if(y>300)
704     y=y-300;
705
706     else
707     y=-(300-y);
708
709     x=x/8.57;
710     y=y/8.57;
711 //Conversions done
712
713
714 //Euclid's Distance Formula
715 myradius = sqrt((fin.x-strt.x)*(fin.x-strt.x)+(fin.y-strt.y)*(fin.y
        -strt.y))/2;
716 xc = (fin.x+strt.x)/2;
717 yc = (fin.y+strt.y)/2;
718 //We now have centre and radius
719
720 radius = myradius;
721 drawSmiley(xc, yc, myradius);
722
723 fin.x = x;
724 fin.y = y;
725
726 myradius = sqrt((fin.x-strt.x)*(fin.x-strt.x)+(fin.y-strt.y)*(fin.y
        -strt.y))/2;
727 xc = (fin.x+strt.x)/2;
728 yc = (fin.y+strt.y)/2;
729
730 radius = myradius;
731
732 drawSmiley(xc,yc,myradius);
733 glFlush();
734 }
735
736
737 int main(int argc,char **argv)
738 {
739
740  buttonInit();
741  colorInit();
742  glutInit(&argc,argv);
743  glutInitWindowSize(600,600);
744  glutCreateWindow("Screen saver");
745  Init();
```

```
746   glutDisplayFunc(startscreen);
747   glutMouseFunc(mymouse);
748   glutKeyboardFunc(keyboard);
749   glutMainLoop();
750 }
```

Acknowledgements

We would like to express our gratitude towards Prof. Ananthanarayana V.S for providing us an oppurtunity to demonstrate and utilize the skills we learnt in the theory course. We would also like to thank Mr Sridhar and Pushpalatha Ma'am for guiding us and assisting us on every step of our Project.