**Detailed Instructions on how to use the Program  
  
Aiming and Firing:**  
  
To aim your shot you use the left and right arrow keys to move the aiming device left and right respectively, you can also move your disc left and right. To increase/decrease the power of your shot you use the up and down keys respectively. Once you’ve got your shot lined up you press the ‘spacebar’ key to fire your disc. If your disc inadvertently comes back down the table and crosses back beneath the firing line then the disc will return to its initial position but you will not be able to fire it again.

**Modifying the Environment:**  
  
You can change aspects of the game, aspects such as the camera, surface friction, table side’s friction, elasticity between the discs and the table side, and whether obstructions are enabled/disabled.  
  
**Scoring:**  
  
If your disc lands on the innermost circle you are awarded 500 points, 200 points for the inner circle and 100 points for the outermost circle. Scoring is calculated at the end of each turn, so if your disc is knocked off one of the targets or onto the target then your score will be affected at the beginning of the next turn.  
  
**Controls:**  
 Q/A – Increase/Decrease the friction between the surface and each disc.  
 W/S – Increase/Decrease the friction between the table sides and each disc.  
 E/D – Increase/Decrease the elasticity between the table sides and each disc.  
 T – Toggle on/off the obstructions.  
 Up Arrow/Down Arrow – Increase/Decrease the power of the shot.  
 Left Arrow/Right Arrow – Increase/Decrease the firing angle.  
 Spacebar – Fire the disc.  
 N/M – Move the disc left/right.

**Summary about the Techniques and Algorithms Used  
  
Techniques:**Lighting:  
  
I wanted to create a spotlight effect on my table (3 spotlights placed evenly apart above the table).  
GLfloat LightAmbient[]= { 0.0f, 0.0f, 0.0f, 1.0f };

GLfloat LightDiffuse[]= { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat LightSpecular[]= { 1.0f, 1.0f, 1.0f, 1.0f };   
float spotlight1Direction[] = {0, 0, -1};

glEnable(GL\_LIGHT0);  
float spotlight1Position[] = {600, 0, 2000, 1};  
glLightfv(GL\_LIGHT0, GL\_AMBIENT, LightAmbient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, LightDiffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR,LightSpecular);

glLightfv(GL\_LIGHT0, GL\_POSITION, spotlight1Position);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, spotlight1Direction);

glLightf(GL\_LIGHT0, GL\_SPOT\_EXPONENT, 30.0f);

glLightf(GL\_LIGHT0, GL\_SPOT\_CUTOFF,45.0);  
Here I set the ambience of the light to 0 (no ambience) and the colour of the diffuse and specular light to 1 (bright white). Then the position is set accordingly, the following lines then enable the light and set the diffuse, specular and position of the light, the spot\_direction sets the direction of the spotlight, spot\_exponent sets the intensity (focused/spread out) of the light, and the cut-off specifies the angle of light (cone shaped).  
  
Textures  
  
I needed a way to load textures, so I took a TGA loading routine from nehe’s website, it was straightforward to implement.  
glEnable(GL\_TEXTURE\_2D);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

// Draw specular highlights on top of textures

glLightModeli(GL\_LIGHT\_MODEL\_COLOR\_CONTROL, GL\_SEPARATE\_SPECULAR\_COLOR );  
This enables textures, the TexEnvi is set to GL\_Modulate so the colour of the texture is affected by the colour of object it is been applied to.  
LightModeli’s parameters ensure the specular highlights of the lighting are applied correctly on textures.  
  
**Algorithms:**  
  
Collision Detection  
  
Sphere-Sphere collision:  
 if (((radius1 + radius2)\*(radius1 + radius2)) >=

((sphere2.X()-sphere1.X())\*(sphere2.X()-sphere1.X())) +

((sphere2.Y()-sphere1.Y())\*(sphere2.Y()-sphere1.Y())))

{

return true; // Collision

}

return false; // No collision  
The squareroot is avoided because it is up to 70 times slower than a multiplication.  
  
Sphere – Plane collision:  
 double distance;

for (int i = 0; i < edge.size(); i++)

{

distance = edge[i].dotProduct(edgeNorm[i]); // Check for collision against each edge

distance = sphere1.dotProduct(edgeNorm[i]) - distance;

if (abs(distance) < radius) // Outside is touching

{

normHit.SetVector(edgeNorm[i].X(), edgeNorm[i].Y(), edgeNorm[i].Z());

return true; // Collision

}

if (abs(distance) <= 0) // Moved too fast and it's now intersecting at 2 points

{

return true; // Collision

}

}

return false; // No collision  
  
Sphere-Target collision is essentially the same as sphere-sphere collision.  
  
Collision Response  
  
The collision response code for a sphere-plane collision is fairly straightforward:  
 // 0,1,0 = top, 1,0,0 = left/right, 0,1,0 = bottom

Vector3d vectorOriginal = player[i].GetVelocityVector(j);

// VR = VI - 2(VI.N)N

// VR = VI - (1+e)(VI.N)N - Elasticity taken into account

Vector3d vectorNew;

double dotProduct = vectorOriginal.dotProduct(normHit); // (VI.N)

Vector3d vectorTemp = normHit \* vectorOriginal; // (VI.N)N

vectorTemp = vectorTemp \* (1+myTable.GetDiscToSideElasticity()); // 1+e(VI.N)N

vectorTemp.SetVector(vectorOriginal.X() - vectorTemp.X(),

vectorOriginal.Y() - vectorTemp.Y(), vectorOriginal.Z() - vectorTemp.Z());

player[i].SetVelocityVector(j, vectorTemp);  
However the collision response code for a sphere-sphere collision is not as straightforward and the code is fairly lengthy. Essentially the velocity vectors and an x axes is needed between the two centres of the spheres.