```
program Sheet_Stretching_4;       {Albert Harris      1997}

uses memtypes, quickdraw, OSIntf,ToolIntf;        {these are "libraries"}

const  PI  =3.1415926;
       MM=5;                      {Sets the number of points to pull on each other as 5 times 5}
       MN=MM;

       STARTPRESSURE = -0.5;   {This determines the pressure to push upward on each point}
       SX = 0.1;       {I forget what this does; it may be unused, left over from an earleir version}
       SY = 0.15;       {Also may be lef over, and not really used}
       STRENGTH = 0.3;  [This sets an initial force pulling points toward each other}

type ary = array [0..MM,0..MN] of real;   {Creates memory space for the point locations}

 var                              {Pascal requires naming all your variables at the beginning}
   x, y, z, nx, ny, nz,original_x,original_y,original_z :ary;
   aniso, dx, dy, dz, dist: real;
   MAG, DISX, DISY,sq: real;
   tryx, tryy, tryz, factor, anfactor, movex, movey, movez, dist_Real : real;
   a,q,qq,m,n,i,dist_Int : integer;
   IX, IY, IZ, TM, TN, COL, count, cycle:integer;

   volume, startvolume, pressure :real;
   mem : longint;
         mouseLoc: Point;

 procedure set_rectangle; {This sets the initial locations of all the points, that will pull on each
other}
   begin;
     for m:= 1 to mm do
      begin
        for n:= 1 to mn do
        begin
          x[m,n]:=400 * (m)/mm;
          y[m,n]:=400 * (n)/mn;
          z[m,n]:=m;

          nx[m,n]:=x[m,n];      {a new location will be recalculated for every point, during each cycle}
          ny[m,n]:=y[m,n];
          nz[m,n]:=z[m,n];

         original_x[m,n]:=x[m,n];
         original_y[m,n]:=y[m,n];
         original_z[m,n]:=z[m,n];


        end;
      end;
 end;  {set_rectangle}
```

{If you change each point's location as soon as it is recalculated, then this will cause the shape changes to vary according to what order you do the recalculations, which is not

good}

```
 procedure new_old; {After all the points have been recalculated, then they are all changed}
  begin
    for m:= 0 to mm+1 do
     begin
       for n:= 0 to mn+1 do
       begin
         x[m,n]:=nx[m,n];
         y[m,n]:=ny[m,n];
         z[m,n]:=nz[m,n];
       end;
     end;
   end; {new_old}

 procedure slant_plot (MAG, DISX, DISY: real);    {Makes it look like a perspective view}
  begin
   for m:= 1 to mm do begin  {This draws horixontal lines}
     moveto (round((x[m,1]+SY*y[m,1])*MAG+DISX),round((z[m,1]+SY*y[m,1])*MAG+DISY));
    for n:= 1 to mn do begin     {this will draw vertical lines}
     lineto (round((x[m,n]+SY*y[m,n])*MAG+DISX),round((z[m,n]+SY*y[m,n])*MAG+DISY));
    end;
  end;

   for n:= 1 to mn do begin
     moveto (round((x[1,n]+SY*y[1,n])*MAG+DISX),round((z[1,n]+SY*y[1,n])*MAG+DISY));
      for m:= 1 to mm do begin
       lineto (round((x[m,n]+SY*y[m,n])*MAG+DISX),round((z[m,n]+SY*y[m,n])*MAG+DISY));
      end;
     end;
  end; {slant_plot}

procedure top_plot (MAG, DISX, DISY: real);
  begin
   for m:= 1 to mm do begin
     moveto (round(x[m,1]*MAG+DISX),round(y[m,1]*MAG+DISY));
    for n:= 1 to mn do begin
     lineto (round(x[m,n]*MAG+DISX),round(y[m,n]*MAG+DISY));
    end;
  end;

   for n:= 1 to mn do begin
     moveto (round(x[1,n]*MAG+DISX),round(y[1,n]*MAG+DISY));
      for m:= 1 to mm do begin
       lineto (round(x[m,n]*MAG+DISX),round(y[m,n]*MAG+DISY));
      end;
     end;
  end; {top_plot}

procedure vectors_plot (MAG, DISX, DISY: real);
  begin
   for m:= 1 to mm do begin
     for n:= 1 to mn do begin
     moveto(round(original_x[m,n]*MAG+DISX),round(original_y[m,n]*MAG+DISY));
```

```
    lineto(round(x[m,n]*MAG+DISX),round(y[m,n]*MAG+DISY));
   end;
  end;
  end; {vectors_plot}


procedure pole_plot (MAG, DISX, DISY: real);   {Draws view from straight above}
 begin
   for m:= 1 to mm do begin
    moveto (round(y[m,1]*MAG+DISX),round(z[m,1]*MAG+DISY));
   for n:= 1 to mn do begin
    lineto (round(y[m,n]*MAG+DISX),round(z[m,n]*MAG+DISY));
   end;
  end;

   for n:= 1 to mn do begin
    moveto (round(y[1,n]*MAG+DISX),round(z[1,n]*MAG+DISY));
     for m:= 1 to mm do begin
      lineto (round(y[m,n]*MAG+DISX),round(z[m,n]*MAG+DISY));
     end;
    end;
  end;   {pole_plot}


procedure equator_plot (MAG, DISX, DISY: real);  {Draws view from the side, which isn't
needed}
begin
   for m:= 1 to mm do begin
    moveto (round(x[m,1]*MAG+DISX),round(z[m,1]*MAG+DISY));
   for n:= 1 to mn do begin
    lineto (round(x[m,n]*MAG+DISX),round(z[m,n]*MAG+DISY));
   end;
  end;

   for n:= 1 to mn do begin
    moveto (round(x[1,n]*MAG+DISX),round(z[1,n]*MAG+DISY));
     for m:= 1 to mm do begin
      lineto (round(x[m,n]*MAG+DISX),round(z[m,n]*MAG+DISY));
     end;
    end;
  end;     {equator_plot}



procedure push;  {This is my best effort to change pressure effects with distance between points}
    var area : real;        {And it is only an approximation}
  begin
   area:=SQRT(SQR(x[m-1,n]  -x[m+1,n]) +
            SQR(y[m-1,n]  -y[m+1,n]) +
            SQR(z[m-1,n]  -z[m+1,n])) *
         SQRT(SQR(x[m  ,n-1]-x[m  ,n+1]) +
            SQR(y[m  ,n-1]-y[m  ,n+1]) +
            SQR(z[m  ,n-1]-z[m  ,n+1])) ;
```

```
 { writeln('area=',area);}  {Left over from when I was de-bugging the program, to find errors_
 if area = 0 then area:=1000;          {Effects of pressure across the sheet of points}
                                                          {Isn't really needed}
    pressure:= STARTPRESSURE;

    movex:= movex - pressure * (z[m+1,n] - z[m-1,n]) *
                          (y[m,n+1] - y[m,n-1]) / area ;

    movey:= movey - pressure * (z[m,n+1] - z[m,n-1]) *
                          (x[m+1,n] - x[m-1,n])  / area ;

    movez:= movez + pressure * (x[m+1,n] - x[m-1,n]) *
                          (y[m,n+1] - y[m,n-1])  / area ;

 end; {push}


function force (dist : real): real;     {Defines the equation by which pulling force varies with
distance}
   begin

    force := -STRENGTH -0.2 * (dist-200/MM)*STRENGTH;

    {force :=  - 5;}                {Left over from de-bugging; it's an alternative force equation}

   { force := 5  - dist/10;}        {Also left over from de-bugging; it's an alternative force
equation}

   end;


      {The following procedure draws the graph of how much forces vary with distance}

 procedure Force_Distance_plot (MAG, DISX, DISY: real);
begin
   moveto (round(1*MAG + DISX),round((0)*MAG+DISY));
   lineto (round(200*MAG + DISX),round((0)*MAG+DISY));


   moveto (round(1*MAG + DISX),round(force(0)*MAG+DISY));

    for dist_Int:= 0 to 200 do begin

    lineto  (round(Dist_Int*MAG + DISX),round(force(Dist_Int)*MAG+DISY));

   end;
end;    {Force_Distance_plot}


procedure sub_pull;
   var dx,dy,dz,dist,f:real;
 begin
   dx := tryx - x[tm,tn];            {Calculates distances and direction to neighboring points}
```

```
   dy := tryy - y[tm,tn];
   dz := tryz - z[tm,tn];

   dist:= SQRT (dx*dx + dy*dy + dz*dz) ;   {Calculates point to point distances}  Pythagorean

 if dist=0 then dist:=1;            {Prevents points from landing exactly on top of each other}
   f:= force(dist)*aniso;

   movex:= movex + f*dx/dist ;
   movey:= movey + f*dy/dist ;
   movez:= movez + f*dz/dist ;
 end; {subpull}


procedure pull;
     var count: integer;
    begin
     for m:=2 to mm-1 do begin
       for n:= 2 to mn-1 do begin
         count :=0;
          tryx:=x[m,n];
          tryy:=y[m,n];
          tryz:=z[m,n];
            repeat
               movex:=0;movey:=0;movez:=0;            {This decides which neighboring points will
pull}
             tm:=m-1 ; tn := n ; aniso := 1 ; sub_pull;   {These are the orthagonal neighbors}
             tm:=m+1 ; tn := n ; aniso := 1 ; sub_pull;

             tm:=m ; tn := n-1 ; aniso := 1 ; sub_pull;
             tm:=m ; tn := n+1 ; aniso := 1 ; sub_pull;

             tm:=m-1 ; tn := n-1 ; aniso := 1 ; sub_pull;  {These are the diagonal neighbors}
             tm:=m-1 ; tn := n+1 ; aniso := 1 ; sub_pull;

             tm:=m+1 ; tn := n-1 ; aniso := 1 ; sub_pull;
             tm:=m+1 ; tn := n+1 ; aniso := 1 ; sub_pull;

      push;
             tryx:=tryx + movex;      {This finds the effect of each cycle of pulling}
             tryy:=tryy + movey;
             tryz:=tryz + movez;
            count := count+1;

{This allows you to recalculate each points new position many as many times as you want, or
need}

      until (sqrt (sqr(movex)+sqr(movey)+sqr(movez)) <0.2) or (count>2);  {Or just calculate once}

          nx[m,n] :=tryx;
          ny[m,n] :=tryy;
          nz[m,n] :=tryz;

       end;
```

```
    end;
end;     {pull}


begin  {main program};
 writeln('Be patient. The computer is calculating initial positions');
 writeln('for points in a spherical surface.');

   set_rectangle;
   pressure:= STARTPRESSURE;
   clearscreen;
    {slant_plot(1.5,50,190);}  cycle:=0;

          slant_plot    (0.5, -0 , 100);
          top_plot      (0.5, 250,  50);

         Force_Distance_plot(1,20,260);


                { pole_plot     (0.5,    20,  270);
           equator_plot  (0.5,   250,  130);}

   writeln('cycle number ',cycle, '              To QUIT, just type the "return" key, and wait.');

  while not keypressed do
     begin;
                 getmouse(mouseLoc);

        {z[3,3]:=100; }

        z[3,3]:=(mouseLoc.v-100);
        x[3,3]:=(mouseLoc.h-100);
        y[3,3]:=(mouseLoc.v-100);

        {The following were used in de-bugging and testing the program}

        {x[5,8]:= x[5,8]+2*sq;    x[11,8]:= x[11,8]-2*sq;

        y[5,8]:= y[5,8]+0.5*sq;  y[11,8]:= y[11,8]+0.5*sq;


        y[8,12]:= y[8,10]-1*sq;*}



       {writeln( mouseLoc.v-100);            {Used in trial runs}
       writeln( mouseLoc.h-100); }

        pull;                          {THIS IS THE HEART OF THE PROGRAM}
        new_old;
        clearscreen;    cycle:=cycle+1;
```

```
          writeln('cycle number ',cycle, '         To QUIT, just type the "return" key, and wait.');


        {The following two lines were used in trial runs, and are not needed}

        { writeln('furrow is ',anfactor:2:2, ' times stronger.');}
         {writeln('pressure is ',pressure:2:2, ' arbitrary units');}

           slant_plot    (0.5, -0 , 50);              {Draws the perspective view}
           top_plot      (0.5, 250,  50);             {Draws the view from straight above}
           vectors_plot  (0.4, 0,  90);    {Draws the direction and distance of each points
movement}

    Force_Distance_plot(1,20,260)   {Draws the graph of pulling force as a function of distance}


      end;  {This is the end of the loop, in the sense of one set of re-calculations}
                {Notice the ;             " end;  "  is the end of any kind of loop}
     readln;   {This you don't need}

   end.   {This is the end of the whole program.
           The word "end" followed by a period ends the program.}
```