```
////////////////////////////////////////////////////
// An Algorithm for Drawing General Undirected Graphs //
// Tomihisa KAMADA & Satoru KAWAI, 1989               //
////////////////////////////////////////////////////
// GLOBALE VARIABLEN:
// path[i,j]: Entfernungsmatrix
// euclid: 0=hyperbolisch, 1=euclidisch
// diameter: längster kürzester Pfad im Netzwerk
// sumXm[m], sumYm[m]: Kräfte auf einzelnen Knoten

function l(i,j: INTEGER): SINGLE;
//berechnet die optimale Länge zwischen zwei Knoten in Abhängigkeit von der Pfaddistanz
//im Hyperbolischen Fall (globale Variable euclid=0) ist diese Länge von der Position der beiden Knoten abhängig
VAR entfernung, max, min, faktor: SINGLE;
begin
  //L=1/diameter
  //l(ij)=L x path(i,j)
  if euclid=1 then begin
    if path[i,j]<0 then
      l:=0.5
    else
      l:=1/diameter*path[i,j];
  end else begin
    max:=StrToFloat(RightStr(Form1.Factor31.Caption,Length(Form1.Factor31.Caption)-8));
    min:=0;
    entfernung:=Sqrt(Power((X[i]-0.5),2)+Power((Y[i]-0.5),2));
    entfernung:=entfernung+Sqrt(Power((X[j]-0.5),2)+Power((Y[j]-0.5),2));
    faktor:=(1.41-entfernung)*(max-min)/1.41+min;
    if path[i,j]<0 then
      l:=0.5
    else
      l:=1/diameter*path[i,j]*faktor;
  end;
end;

function k(i,j: INTEGER): SINGLE;
//k=die Federstärke welche zwischen zwei Knoten gilt
VAR Kgross: SINGLE;
begin
  Kgross:=1;
  //k(ij)=K / d(i,j)^2
  if path[i,j]<0 then
    k:=Kgross/Power(diameter,2)
  else
    k:=Kgross/Power(path[i,j],2);
end;
```

```
function DeltaMAll(m: INTEGER): SINGLE;
//DeltaM = die Kraft, die auf einem Knoten lastet
//Berechne DeltaM mit Xm und Ym
VAR i: INTEGER;
begin
  sumXm[m]:=0; sumYm[m]:=0;
  for i:=1 to nodeanz do if i<>m then begin
    if (X[i]=X[m]) AND (Y[i]=Y[m]) then X[i]:=X[i]+0.00001;
    sumXm[m]:=sumXm[m]+k(m,i)*( (X[m]-X[i])- (l(m,i)*(X[m]-X[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
    sumYm[m]:=sumYm[m]+k(m,i)*( (Y[m]-Y[i])- (l(m,i)*(Y[m]-Y[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
  end;
  DeltaMAll:=Sqrt((Power(sumXm[m],2)+Power(sumYm[m],2)));
end;

function DeltaMOne(m: INTEGER): SINGLE;
//DeltaM = die Kraft, die auf einem Knoten lastet
//DeltaMOne ist eine Optimierung, berechnet nur die Kraft neu zum Knoten, der gerade verschoben wurde
VAR i: INTEGER;
begin
  DeltaMOne:=Sqrt((Power(sumXm[m],2)+Power(sumYm[m],2)));
end;

procedure DeltaMMinus(m: INTEGER; verschiebenode: INTEGER);
//rechnet aus der Kraftsumme eines Knoten die Kraft des zu verschiebenden Knotens raus
VAR i: INTEGER;
begin
  i:=verschiebenode;
  if (X[i]=X[m]) AND (Y[i]=Y[m]) then X[m]:=X[m]+0.00001;
  sumXm[m]:=sumXm[m]-k(m,i)*( (X[m]-X[i])- (l(m,i)*(X[m]-X[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
  sumYm[m]:=sumYm[m]-k(m,i)*( (Y[m]-Y[i])- (l(m,i)*(Y[m]-Y[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
end;


procedure DeltaMPlus(m: INTEGER; verschiebenode: INTEGER);
//rechnet in die Kraftsumme eines Knoten die Kraft des gerade verschobenen Knotens wieder rein
VAR i: INTEGER;
begin
  i:=verschiebenode;
  if (X[i]=X[m]) AND (Y[i]=Y[m]) then X[m]:=X[m]+0.00001;
  sumXm[m]:=sumXm[m]+k(m,i)*( (X[m]-X[i])- (l(m,i)*(X[m]-X[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
  sumYm[m]:=sumYm[m]+k(m,i)*( (Y[m]-Y[i])- (l(m,i)*(Y[m]-Y[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
end;
```

```
//DIE EINZELNEN TEILE DER GROSSEN GLEICHUNG ZUR BESTIMMUNG VON DELTAx und DELTAy
//Zur berechnung der Kräfte wird die Kräftegleichung partiell angeleitet
//die Zeilennummerierung bezieht sich auf das Originalpaper

//Funktionszeile 7
function f5(m: INTEGER): SINGLE;
VAR i: INTEGER;
  sum: SINGLE;
begin
  sum:=0;
  for i:=1 to nodeanz do if i<>m then begin
    sum:=sum+k(m,i)*( (X[m]-X[i])- (l(m,i)*(X[m]-X[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
  end;
  f5:=sum;
end;

//Funktionszeile 8
function f6(m: INTEGER): SINGLE;
VAR i: INTEGER;
  sum: SINGLE;
begin
  sum:=0;
  for i:=1 to nodeanz do if i<>m then begin
    sum:=sum+k(m,i)*( (Y[m]-Y[i])- (l(m,i)*(Y[m]-Y[i]))/ Sqrt(Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) );
  end;
  f6:=sum;
end;

//Divisor (der untere Teil) der kommenden 4 Formeln
function unten(m,i: INTEGER): SINGLE;
begin
  unten:= Power( (Power((X[m]-X[i]),2)+Power((Y[m]-Y[i]),2)) , 3/2 );
end;

//Formeln 13 - 16 aus dem Paper Seite 10
function f1(m:INTEGER):SINGLE;
VAR i: INTEGER;
  sum: SINGLE;
begin
  sum:=0;
  for i:=1 to nodeanz do if i<>m then
    sum:=sum+k(m,i)*(1-l(m,i)*Power((Y[m]-Y[i]),2) / unten(m,i));
  f1:=sum;
end;
```

```
function f2(m:INTEGER):SINGLE;
VAR i: INTEGER;
  sum: SINGLE;
begin
  sum:=0;
  for i:=1 to nodeanz do if i<>m then
    sum:=sum+k(m,i)*(l(m,i)*((X[m]-X[i])*(Y[m]-Y[i])) / unten(m,i));
  f2:=sum;
end;

function f3(m:INTEGER):SINGLE;
VAR i: INTEGER;
  sum: SINGLE;
begin
  sum:=0;
  for i:=1 to nodeanz do if i<>m then
    sum:=sum+k(m,i)*(l(m,i)*((X[m]-X[i])*(Y[m]-Y[i])) / unten(m,i));
  f3:=sum;
end;

function f4(m:INTEGER):SINGLE;
VAR i: INTEGER;
  sum: SINGLE;
begin
  sum:=0;
  for i:=1 to nodeanz do if i<>m then
    sum:=sum+k(m,i)*(1-l(m,i)*Power((X[m]-X[i]),2) / unten(m,i));
  f4:=sum;
end;


//Die eigentlichen Kräftegleichungen, welche die Teilrechnungen von oben übernehmen
function fDeltaY(m: INTEGER): SINGLE;
begin
  fDeltaY:=(f1(m)*f6(m)-f5(m)*f3(m))/(f2(m)*f3(m)-f1(m)*f4(m));
end;
function fDeltaX(m: INTEGER; deltaY: SINGLE): SINGLE;
begin
  fDeltaX:=(-f5(m)-f2(m)*deltaY)/f1(m);
end;
```

```
procedure Distortion;
//Verändert die Größe der Knoten je nachdem, wie weit weg sie von Bildschirmmittelpunkt sind
//Soll den Verzerrungseffekt unterstützen
//globale Variable vector[i] speichert die Knotengrößen
VAR
  anzahl, i, durchlauf: INTEGER;
  bMax, alpha, beta, b, bNeu, xQuer, yQuer: SINGLE;
begin
  for i:=1 to nodeanz do if nodedispose[i]=0 then begin
    //entfernung des punkts vom Zentrum
    b:=Sqrt(Power((X[i]-0.5),2) + Power((Y[i]-0.5),2));
    //Projektionswinkel auf die Halbkugel (wie weit am rand/unten liegt ein Punkt)
    alpha:=0.5*b*Pi/2;
    bNeu:=Sin(alpha)*0.5;
    vector[i]:=2*Cos(Pi/4+alpha);
  end;
  vectorautosize(10); //automatische Anpassung der Größe der Knoten
  AreaXYFit(1);   //Einpassung auf den Bildschirm
end;


procedure OptimizeKamadaKawai(show: INTEGER);
//Der Optimierungsalgorithmus
VAR min, maxDeltaM: SINGLE;
  hilf: SINGLE;
  gesamtanz,maxDeltanrALT, maxDeltanr,i,j,durchlauf,anz: INTEGER;
  deltaX, deltaY: SINGLE;
Label Weiter;
begin
  Zentralitaeten(5,0,0,0,0);

  deltaX:=0; deltaY:=0;
  min:=0.01; anz:=0;
  gesamtanz:=0;

  maxdeltaM:=0;
  for i:=1 to nodeanz do begin
    hilf:=deltaMAll(i);
    if hilf>maxdeltaM then begin
      maxdeltaM:=hilf;
      maxdeltanr:=i;
    end;
  end;

  anz:=0;
  Repeat
    if maxdeltaM<min then Goto Weiter;
```

```
    maxDeltanrALT:=maxDeltanr;

    for i:=1 to nodeanz do if i<>maxdeltanr then
      deltaMMinus(i,maxdeltanr);

    durchlauf:=0;
    Repeat
      Inc(durchlauf);
      Inc(gesamtanz);
      deltaY:=fDeltaY(maxdeltanr);
      deltaX:=fDeltaX(maxdeltanr,deltaY);
      if show=1 then NodeUndBeziehungenLoeschen(maxdeltanr);
      X[maxdeltanr]:=X[maxdeltanr]+deltaX;
      Y[maxdeltanr]:=Y[maxdeltanr]+deltaY;
      if show=1 then NodeUndBeziehungenZeichnen(maxdeltanr);
    //Solange bis die Kraft unter ein Minimum fällt oder ein Knoten 50 Mal in einem Durchlauf optimiert wird
    Until (DeltaMAll(maxDeltanr)<min) OR (durchlauf>=50);

    for i:=1 to nodeanz do if i<>maxdeltanr then
      deltaMPlus(i,maxdeltanr);

    maxdeltaM:=0;
    for i:=1 to nodeanz do begin
      hilf:=deltaMOne(i);
      if hilf>maxdeltaM then begin
        maxdeltaM:=hilf;
        maxdeltanr:=i;
      end;
    end;
    if maxDeltanr=maxDeltanrALT then Inc(anz);
    //Sollte 20 x der gleiche Knoten zu verschieben sein, dann Abbruch!
    if anz>20 then begin
      Goto Weiter;
    end;
  Until gesamtanz>nodeanz*40;

Weiter:
  if (euclid=0) AND (Form1.VertexDistortion2.checked) then Distortion;
end;
```