

Appendix

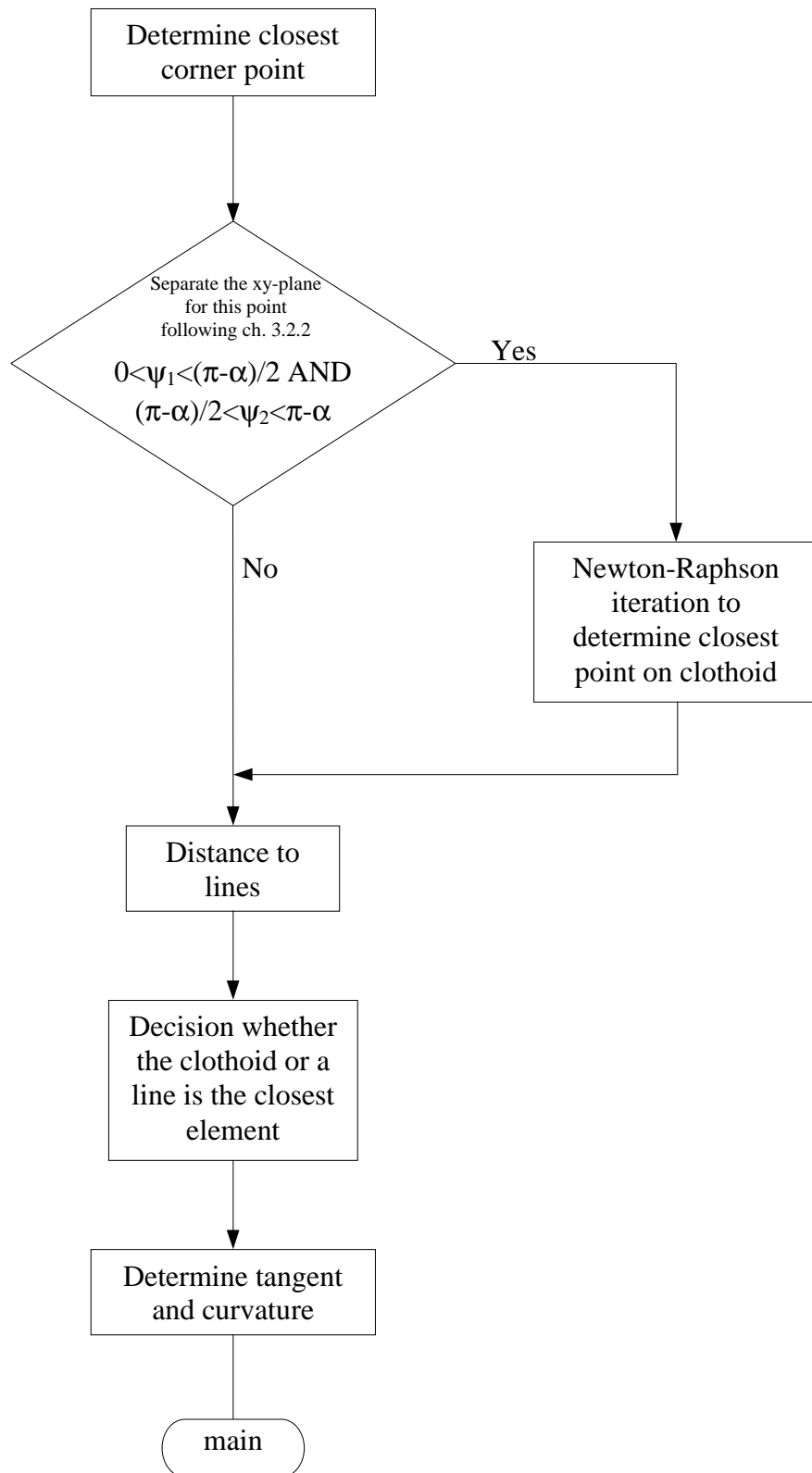


Figure 1: Algorithm for collision detection

1. Main program

```
/* This program allows the user to input an arbitrary amount (<10) of points with the mouse.
After that the set up polygon is separated in elements existing of two subsequent lines, which
intersect at the corner point. For these elements the coordinate frames to compute the clothoid
are set up and 50 points for both the proximal and the distal element are computed (in order to
plot it).
Then the user can define arbitrary points in the xy plain and for these points the closest
element is computed and shown graphically on the screen.
The used function are the following:
1) fresnel: computes the values for the approximation of the fresnel integrals
2) derivativezero: determines the point on the clothoid if the clothoid is the closest element
3) closestpoint: determines the closest point (needed to know in which corner the arbitrary
point is)
4) closestline: computes the closest line if the point is not close to the clothoid
5) relative_coordinate: computes the coordinates relative to a chosen point or the global coord.
of a point that is given in relative coord.
6) closest_element: determines the closest element by dividing the area in sub-areas in which
the point is closest to the clothoid, thus only the necessary computation is
done

The used variables:
rhotext: string for input querie of rho

i, j, e, ii, iii: count variables for loops
m: is set to 1 one if last defined point equals the first point
n: number of points (-> n-2 number of corners/elements)

prod: variable used to compute the scalar product
C, S: values of fresnel integrals
norm_c2, norm_d2: length of c2 and d2 in order to make them unit length
s_vec: values needed to compute the coordinates of the clothoid
eye[2][2]: 2x2-identity matrix
rhoold: variable needed to store corner.rho
help: help variable to store a temporary value

point_on_elem: stores the coordinates of the closest point, the curvature of the corresponding
closest element and its tangent
corner: stores all the information of a corner: direction of the lines, coordinate frames,
the parameters determining the clothoid, the values of the clothoid and the point of
intersection of the perpendicular through the starting and ending point
*/

#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <graph.h>
#include <stdlib.h>
#include <stddef.h>
#include <time.h>
#include <dos.h>
#include <i86.h>
#include <time.h>
#include <sys\timeb.h>
#include "mousesupport.h"

/* -----*/
/* ----- used functions -----*/
/* -----*/

void fresnel(float x, float *C, float *S);
float derivativezero( Pairs corner, Lines position, float *pp, int numberofpoint, int si);
float closestpoint(Lines position, int n, int *numberofpoint, Lines l,
float *vec_to_point);
float closestline(Pairs corner[numofpoints], Lines position,
Lines l, float *s, float *r, int g, int h);
void relative_coordinate(int cx, int cy, Lines *l, int n, int sig);
void closest_element(Lines position, int n, int m, Lines l, Pairs corner[numofpoints],
pointOnElem *point_on_elem);

/* -----*/

main()
{
char rhotext[80], msg[80];
```

```

int i, j, e, n, m, ii, iii, ee/*, flag*/;
float prod, C, S, help, norm_c2, norm_d2, eye[2][2], rhoold[numofpoints], s_vec[50];
Lines l, c, position;
pointOnElem point_on_elem;

static Pairs corner[numofpoints];

struct timeb start1_time, end1_time;

i=0;

/*----- input of points that build up the walls -----*/

i=mouse(&l, &c); //return number of points

// in order to get a closed polygon add first two points as last points

m=1;

l.points[i][0]=l.points[0][0];
l.points[i][1]=l.points[0][1];
i++;
l.points[i][0]=l.points[1][0];
l.points[i][1]=l.points[1][1];
i++;

n=i;

for (ii=0; ii<n-2; ii++)
{
    sprintf(rhotext,"rho for %d. corner: ", ii+1);
    _settextposition(1,1);
    _outtext(rhotext);
    scanf("%f", &corner[ii].rho);
    sprintf(rhotext,"                ");
    _settextposition(1,1);
    _outtext(rhotext);
}

/*----- defining the directions of the lines -----*/
/*-----*/

for (e=0;e<n-2;e++) // loop over all elements (corner)
{
    corner[e].point[0]=l.points[e+1][0]; // point of intersection (corner point)
    corner[e].point[1]=l.points[e+1][1];

    corner[e].t_prime[0]=l.points[e+1][0]-l.points[e][0]; // direction of first line
    corner[e].t_prime[1]=l.points[e+1][1]-l.points[e][1];
    corner[e].norm_tpr=sqrt(corner[e].t_prime[0]*corner[e].t_prime[0]+
        corner[e].t_prime[1]*corner[e].t_prime[1]);
    corner[e].t_prime[0]=corner[e].t_prime[0]/corner[e].norm_tpr;
    corner[e].t_prime[1]=corner[e].t_prime[1]/corner[e].norm_tpr;

    corner[e].t_g[0]=l.points[e+2][0]-l.points[e+1][0]; // direction of second line
    corner[e].t_g[1]=l.points[e+2][1]-l.points[e+1][1];
    corner[e].norm_tg=sqrt(corner[e].t_g[0]*corner[e].t_g[0]+
        corner[e].t_g[1]*corner[e].t_g[1]);
    corner[e].t_g[0]=corner[e].t_g[0]/corner[e].norm_tg;
    corner[e].t_g[1]=corner[e].t_g[1]/corner[e].norm_tg;

    // define 2x2 identity matrix
    for (i=0;i<2;i++)
    {
        for (j=0;j<2;j++)
        {
            eye[i][j] = (i==j) ? 1 : 0;
        }
    }
}

/* --- set coordinate frames for proximal and distal segment of the clothoid --- */

```

```

/* proximal c1, c2      distal d1, d2*/
corner[e].c1[0]=corner[e].t_prime[0];
corner[e].c1[1]=corner[e].t_prime[1];

corner[e].d1[0]=corner[e].t_g[0];
corner[e].d1[1]=corner[e].t_g[1];

for (i=0;i<2;i++)
{
    corner[e].c2[i]=0.0;
    corner[e].d2[i]=0.0;
    for (j=0;j<2;j++)
    {
        help=corner[e].t_prime[i]*corner[e].t_prime[j];
        corner[e].c2[i]=corner[e].c2[i]+(eye[i][j]-help)*
            (corner[e].t_g[j]-corner[e].t_prime[j]);
        help=corner[e].t_g[i]*corner[e].t_g[j];
        corner[e].d2[i]=corner[e].d2[i]+(eye[i][j]-help)*
            (corner[e].t_g[j]-corner[e].t_prime[j]);
    }
}
norm_c2=sqrt(corner[e].c2[0]*corner[e].c2[0]+corner[e].c2[1]*corner[e].c2[1]);
corner[e].c2[0]=corner[e].c2[0]/norm_c2;
corner[e].c2[1]=corner[e].c2[1]/norm_c2;

norm_d2=sqrt(corner[e].d2[0]*corner[e].d2[0]+corner[e].d2[1]*corner[e].d2[1]);
corner[e].d2[0]=corner[e].d2[0]/norm_d2;
corner[e].d2[1]=corner[e].d2[1]/norm_d2;

/*----- set parameters -----*/
corner[e].alpha=0.0;
corner[e].p0=0.0;
corner[e].a=0.0;
corner[e].D=0.0;
prod=0.0;
for (i=0;i<2;i++) //scalar product t_g*t_prime
{
    prod=prod+corner[e].t_g[i]*corner[e].t_prime[i];
}
corner[e].alpha=pi-acos(prod); // alpha is angle between the two lines
corner[e].p0=sqrt(1-cos(alpha/pi));
fresnel(corner[e].p0, &C, &S);
corner[e].a=pi*corner[e].rho*corner[e].p0;
corner[e].D=corner[e].a*(C+S*1/tan(corner[e].alpha/2));

if ( (corner[e].D>(corner[e].norm_tpr/2)) || (corner[e].D>(corner[e].norm_tg/2)) )
// check if D is longer than half of one of the adjacent lines and if positive
// then reduce to the minimum of both
{
    corner[e].D=min(corner[e].norm_tpr/2,corner[e].norm_tg/2);
    corner[e].a=corner[e].D/(C+S*1/tan(corner[e].alpha/2));
    corner[e].rho=corner[e].a/(pi*corner[e].p0);
}
} //end of loop over elements

// ----- END OF LOOP OVER ELEMENTS -----
/*-----*/

/* -----*/
/* ----- Starting point of the cobot -----*/
/* -----*/

// choose startingpoint as ending-point of distant clothoid of first(last) corner
c.points[0][0]=c.points[0][0]+(int)(corner[n-3].d1[0]*corner[n-3].D);
c.points[0][1]=c.points[0][1]+(int)(corner[n-3].d1[1]*corner[n-3].D);

// transform all points to the coordinate system with starting point as origin
relative_coordinate(c.points[0][0], c.points[0][1], &l, n, 1);

for (ee=0; ee<n-2; ee++) // needed because the following loop somehow changes the value of
corner.rho
{
    rhoold[ee]=corner[ee].rho;
}

```

```

    }

/* ----- define proximal segment ----- */
for (e=0; e<n-2; e++)
{
    corner[e].point[0]=l.points[e+1][0]; // point of intersection (corner point)
    corner[e].point[1]=l.points[e+1][1];

    for (j=0;j<=50;j++)
    {
        s_vec[j]=j*corner[e].p0/50;
        fresnel(s_vec[j], &C, &S);
        corner[e].sigma_prox[j][0]=corner[e].point[0]+
            (corner[e].a*C-corner[e].D)*corner[e].c1[0]+corner[e].a*S*corner[e].c2[0];
        corner[e].sigma_prox[j][1]=corner[e].point[1]+
            (corner[e].a*C-corner[e].D)*corner[e].c1[1]+corner[e].a*S*corner[e].c2[1];
    }
}

/* ----- define distal segment ----- */

for (j=0;j<=50;j++)
{
    s_vec[j]=(j+50)*corner[e].p0/50;
    fresnel(2*corner[e].p0-s_vec[j], &C, &S);
    corner[e].sigma_dist[j][0]=corner[e].point[0]+
        (corner[e].D-corner[e].a*C)*corner[e].d1[0]+corner[e].a*S*corner[e].d2[0];
    corner[e].sigma_dist[j][1]=corner[e].point[1]+
        (corner[e].D-corner[e].a*C)*corner[e].d1[1]+corner[e].a*S*corner[e].d2[1];
}
}
for (ee=0; ee<n-2; ee++)
{
    corner[ee].rho=rhoold[ee];
}

/* ----- graphical output ----- */

_setvideomode(_VRES16COLOR);
_moveto(c.points[0][0]-10,c.points[0][1]);
_lineto(c.points[0][0]+10,c.points[0][1]);

_moveto(c.points[0][0],c.points[0][1]-10);
_lineto(c.points[0][0],c.points[0][1]+10);

relative_coordinate(c.points[0][0], c.points[0][1], &l, n, -1);
if (m!=1) // last point is not first point
{
    _moveto(l.points[0][0],l.points[0][1]);
}
else
{
    _moveto(corner[0].sigma_prox[0][0]+c.points[0][0],
corner[0].sigma_prox[0][1]+c.points[0][1]);
}
for (e=0; e<n-2; e++)
{
    for (iii=0; iii<=49; iii++)
    {
        _lineto(corner[e].sigma_prox[iii][0]+c.points[0][0],
corner[e].sigma_prox[iii][1]+c.points[0][1]);
    }
    for (iii=0;iii<=49; iii++)
    {
        _lineto(corner[e].sigma_dist[iii][0]+c.points[0][0],
corner[e].sigma_dist[iii][1]+c.points[0][1]);
    }
}

if (m!=1) // last point is not first point
{
    _lineto(l.points[n-1][0], l.points[n-1][1]);
}
else
{
    _lineto(corner[0].sigma_prox[0][0]+c.points[0][0],
corner[0].sigma_prox[0][1]+c.points[0][1]);
}
}

```

```

// ----- input of arbitray points to check the right computation of the closest point -----
relative_coordinate(c.points[0][0], c.points[0][1], &l, n, 1);
for (i=1; i<10; i++)
{
    mouse1(&position, c, 1);
    relative_coordinate(c.points[0][0], c.points[0][1], &position, 1, 1);

    closest_element(position, n, m, 1, corner, &point_on_elem);

    relative_coordinate(c.points[0][0], c.points[0][1], &position, 1, -1);
    _moveto(position.points[0][0], position.points[0][1]);
    _lineto(point_on_elem.point[0]+c.points[0][0], point_on_elem.point[1]+c.points[0][1]);
}

getch();
_setvideomode(_DEFAULTMODE);

return(0); //end of main
}

```

```

void relative_coordinate(int cx, int cy, Lines *l, int n, int sig)
{
    int i;

    if (sig==1)
    // points relative to chosen origin c
    for (i=0; i<n; i++)
    {
        l->points[i][0] -= cx;
        l->points[i][1] -= cy;
    }

    if (sig==-1)
    // points in global coordinate system
    for (i=0; i<n; i++)
    {
        l->points[i][0] += cx;
        l->points[i][1] += cy;
    }
    return;
}

```

2. Closest_element

```
/* This program determines which element (clothoid, line or point) is closest to an arbitrary
point in the yy-plane. First it computes the closest point. Starting from this we know which
corner is closest. Then the area in this corner is considered: We separate the plane in to sub-
planes by the two perpendicular lines through the starting and ending point of the clothoid and
their intersection. If the point is outside the area which is closest to the clothoid the distance
to all lines is computed, otherwise it is considered to be closest to the clothoid. In the latter
case we further have to decide which part of the clothoid is closest, the proximal or the distal.
Knowing this we can do binary search.
*/

#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "mousesupport.h"

/* -----*/
/* ----- computation of the closest element -----*/
/* -----*/

void closest_element(Lines position, int n, int m, Lines l, Pairs corner[numofpoints],
pointOnElem *point_on_elem)
{
/* ----- USED FUNCTIONS -----*/

void fresnel(float x, float *C, float *S);
float derivativezero( Pairs corner, Lines position, float *pp, int numberofpoint, int si);
float closestpoint(Lines position, int n, int *numberofpoint, Lines l,
float *vec_to_point);
float closestline(Pairs corner[numofpoints], Lines position,
Lines l, float *s, float *r, int g, int h);
/* -----*/

int numberofpoint, i, ii, closest_line;
float vec_to_point[2], pp, s[2], r, sss[2], rrr, C, S,
distance_to_clothoid, distance_to_point, distance_to_line, distance_to_lastline,
minimum, s0, s1, cospsi2, cospsi3, dd2, psi2, psi3,
p1[2], p2[2], ll;

point_on_elem->point[0]=0;
point_on_elem->point[1]=0;
numberofpoint=0;

// closest point
distance_to_point=closestpoint(position, n,
&numberofpoint, l, &vec_to_point);

distance_to_clothoid=0;

/* decisionmaking whether the proximal or the distal segment is closest or
none of both
psi is angle between line after closest point and vector pq
cos(psi)=(p*q)/(|p|*|q|)
the area which is bounded by the two perpendicular lines
through its starting and ending point has to be considered */

if ((n>=3 && numberofpoint != 1 && numberofpoint != n) ||
(m==1 && (numberofpoint==1 || numberofpoint==n-1))) // m=1 means that last point=first point
{
if (numberofpoint==1 || numberofpoint==n-1)
{
numberofpoint=n-1;
}
// p1: starting point p2: ending point
p1[0]=corner[numberofpoint-2].point[0]-
corner[numberofpoint-2].D*corner[numberofpoint-2].c1[0];
p1[1]=corner[numberofpoint-2].point[1]-
corner[numberofpoint-2].D*corner[numberofpoint-2].c1[1];
p2[0]=corner[numberofpoint-2].point[0]+
corner[numberofpoint-2].D*corner[numberofpoint-2].d1[0];
p2[1]=corner[numberofpoint-2].point[1]+
corner[numberofpoint-2].D*corner[numberofpoint-2].d1[1];
}
```

```

ll=(p2[0]-p1[0])/(corner[numberOfpoint-2].c2[0]-corner[numberOfpoint-2].d2[0]);
// parameter of line: x=p1+ll*c2 (or x=p2+ll*d2)

//s: point on bisector where perpendicular lines through starting and
//ending point of c1. intersect
corner[numberOfpoint-2].s[0]=p1[0]+ll*corner[numberOfpoint-2].c2[0];
corner[numberOfpoint-2].s[1]=p1[1]+ll*corner[numberOfpoint-2].c2[1];

dd2=sqrt( (position.points[0][0]-corner[numberOfpoint-2].s[0])*
          (position.points[0][0]-corner[numberOfpoint-2].s[0])+
          (position.points[0][1]-corner[numberOfpoint-2].s[1])*
          (position.points[0][1]-corner[numberOfpoint-2].s[1]) ); //length of vector from
//position to s
cospsi2=0; /*psi2 angle between vector from intersecton to arbitray point and perpendicular
through beginning point */
for (i=0; i<=1; i++) /* scalar (vector from intersecton to arbitray point)*
(perpendicular through beginning point) */
{
    cospsi2=cospsi2+corner[numberOfpoint-2].c2[i]*
            (corner[numberOfpoint-2].s[i]-position.points[0][i]);
}
psi2=acos(cospsi2/dd2); // |c2|=1

cospsi3=0; /*psi3 angle between vector from intersecton to arbitray point and perpendicular
through ending point */
for (i=0; i<=1; i++) /* scalar product (vector from intersecton to arbitray point)*
(perpendicular through ending point) */
{
    cospsi3=cospsi3+corner[numberOfpoint-2].d2[i]*
            (corner[numberOfpoint-2].s[i]-position.points[0][i]);
}
psi3=acos(cospsi3/dd2); // |d2|=1

if (psi2>=0 && psi2<(pi/2-corner[numberOfpoint-2].alpha/2)
    && psi3>=(pi/2-corner[numberOfpoint-2].alpha/2) && psi3<(pi-corner[numberOfpoint-
2].alpha))
{
    // proximal element is closest
    distance_to_clothoid=derivativezero(corner[numberOfpoint-2], position, &pp,
                                        numberOfpoint, 1);

    if (pp != 0)
    {
        fresnel(pp, &C, &S);
        point_on_elem->point[0]=corner[numberOfpoint-2].point[0]+
            (corner[numberOfpoint-2].a*C-
             corner[numberOfpoint-2].D)*corner[numberOfpoint-2].c1[0]+
            corner[numberOfpoint-2].a*S*corner[numberOfpoint-2].c2[0];
        point_on_elem->point[1]=corner[numberOfpoint-2].point[1]+
            (corner[numberOfpoint-2].a*C-
             corner[numberOfpoint-2].D)*corner[numberOfpoint-2].c1[1]+
            corner[numberOfpoint-2].a*S*corner[numberOfpoint-2].c2[1];
        s0=corner[numberOfpoint-2].rho*(pi-corner[numberOfpoint-2].alpha);
        s1=corner[numberOfpoint-2].a*(pp);
        point_on_elem->curvature=s1/(corner[numberOfpoint-2].rho*s0);
        point_on_elem->tangent[0]=cos((s1*s1)/(2*corner[numberOfpoint-2].rho*s0))*
            corner[numberOfpoint-2].c1[0]+
            sin((s1*s1)/(2*corner[numberOfpoint-2].rho*s0))*
            corner[numberOfpoint-2].c2[0];
        point_on_elem->tangent[1]=cos((s1*s1)/(2*corner[numberOfpoint-2].rho*s0))*
            corner[numberOfpoint-2].c1[1]+
            sin((s1*s1)/(2*corner[numberOfpoint-2].rho*s0))*
            corner[numberOfpoint-2].c2[1];
    }
}

else if (psi3>=0 && psi3<(pi/2-corner[numberOfpoint-2].alpha/2)
    && psi2>=(pi/2-corner[numberOfpoint-2].alpha/2) && psi2<(pi-corner[numberOfpoint-
2].alpha))
{
    // distal element is closest
    distance_to_clothoid=derivativezero(corner[numberOfpoint-2], position, &pp,
                                        numberOfpoint, -1);

    if(pp != 0)
    {
        fresnel(2*corner[numberOfpoint-2].p0-pp, &C, &S);
        point_on_elem->point[0]=corner[numberOfpoint-2].point[0]+
            (corner[numberOfpoint-2].D-
             corner[numberOfpoint-2].a*C)*corner[numberOfpoint-2].d1[0]+
            corner[numberOfpoint-2].a*S*corner[numberOfpoint-2].d2[0];
    }
}

```

```

        point_on_elem->point[1]=corner[numberofpoint-2].point[1]+
            (corner[numberofpoint-2].D-
             corner[numberofpoint-2].a*C)*corner[numberofpoint-2].d1[1]+
             corner[numberofpoint-2].a*S*corner[numberofpoint-2].d2[1];
        s0=corner[numberofpoint-2].rho*(pi-corner[numberofpoint-2].alpha);
        s1=corner[numberofpoint-2].a*(pp);
        point_on_elem->curvature=(2*s0-s1)/(corner[numberofpoint-2].rho*s0);
        point_on_elem->tangent[0]=cos( (2*s0-s1)*(2*s0-s1)/
            (2*corner[numberofpoint-2].rho*s0) ) *
            corner[numberofpoint-2].d1[0]-
            sin( (2*s0-s1)*(2*s0-s1)/
            (2*corner[numberofpoint-2].rho*s0) ) *
            corner[numberofpoint-2].d2[0];
        point_on_elem->tangent[1]=cos( (2*s0-s1)*(2*s0-s1)/
            (2*corner[numberofpoint-2].rho*s0) ) *
            corner[numberofpoint-2].d1[1]-
            sin( (2*s0-s1)*(2*s0-s1)/
            (2*corner[numberofpoint-2].rho*s0) ) *
            corner[numberofpoint-2].d2[1];
    }
} // end of if

if (n>=2)
{
    // find the closest line
    minimum=1000;
    distance_to_line=closestline(corner, position, l, &s, &r, 0, 0);
    if (fabs(distance_to_line)<minimum && r>corner[n-3].D && r<corner[n-3].norm_tg-corner[0].D)
    {
        minimum=fabs(distance_to_line);
        closest_line=1;
        for (ii=0; ii<=1; ii++)
        {
            sss[ii]=s[ii];
            rrr=r;
        }
    }
    for (i=1; i<n-2; i++)
    {
        // Distance to all lines except the first one which was computed before
        distance_to_line=closestline(corner, position,
            l, &s, &r, i, i);
        if (fabs(distance_to_line)<minimum && r>corner[i-1].D && r<corner[i-1].norm_tg-corner[i].D)
        {
            minimum=fabs(distance_to_line);
            closest_line=i+1;
            for (ii=0; ii<=1; ii++)
            {
                sss[ii]=s[ii];
                rrr=r;
            }
        }
    }
}

if (minimum<distance_to_clothoid || distance_to_clothoid == 0)
{
    point_on_elem->point[0]=sss[0]; // line is closest
    point_on_elem->point[1]=sss[1];
    for (ii=0; ii<=1; ii++)
    {
        if (closest_line == 1)
            point_on_elem->tangent[ii]=corner[0].c1[ii];
        else
            point_on_elem->tangent[ii]=corner[closest_line-2].c1[ii];
    }
    point_on_elem->curvature=0.0;
}
else if (minimum>distance_to_point && distance_to_clothoid == 0)
{
    point_on_elem->point[0]=l.points[numberofpoint-1][0]; // point is closest
    point_on_elem->point[1]=l.points[numberofpoint-1][1];
} // end of if
return;
}

```

3. Closestline

```
#include "mousesupport.h"

/* ----- function to compute the distance to a line ----- */

float closestline(Pairs corner[numofpoints], Lines position,
                 Lines l, float *s, float *r, int g, int h)
/* position: arbitrary point
   g and h are needed to decide whether the distance to the last line or to any
   other line is computed (g==h for any line, h!=g for last line)
   s: coordinate of the point on the line
   r: distance from starting-point of line to intersection with perpendicular
   through arbitrary point*/
{
    float distance=0.0;
    if (g==h)
    {
        distance=((l.points[h][0]-position.points[0][0])*corner[g].c1[1]+
                 (position.points[0][1]-l.points[h][1])*corner[g].c1[0])/
                 (corner[g].c2[0]*corner[g].c1[1]-corner[g].c2[1]*corner[g].c1[0]);
        s[0]=position.points[0][0]+distance*corner[g].c2[0];
        s[1]=position.points[0][1]+distance*corner[g].c2[1];
        *r=((position.points[0][0]-l.points[h][0])*corner[g].c2[1]-
            (position.points[0][1]-l.points[h][1])*corner[g].c2[0])/
            (corner[g].c1[0]*corner[g].c2[1]-corner[g].c1[1]*corner[g].c2[0]);
    }
    else
    {
        distance=((l.points[h][0]-position.points[0][0])*corner[g].d1[1]+
                 (position.points[0][1]-l.points[h][1])*corner[g].d1[0])/
                 (corner[g].d2[0]*corner[g].d1[1]-corner[g].d2[1]*corner[g].d1[0]);
        s[0]=position.points[0][0]+distance*corner[g].d2[0];
        s[1]=position.points[0][1]+distance*corner[g].d2[1];
        *r=((position.points[0][0]-l.points[h][0])*corner[g].d2[1]-
            (position.points[0][1]-l.points[h][1])*corner[g].d2[0])/
            (corner[g].d1[0]*corner[g].d2[1]-corner[g].d1[1]*corner[g].d2[0]);
    }
    return(distance);
}
```

4. Closestpoint

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "mousesupport.h"

float closestpoint(Lines position, int n, int *numberofpoint, Lines l,
                  float *vec_to_point )
/* numberofpoint: number of the corner-point that is closest to arbitrary point
   vec_to_point: vector from closest corner-point to arbitrary point
   min: minimal distance */
{
    int i;
    float min, distance;

    min=1000.0;
    //vec_to_point[0]=0.0;
    //vec_to_point[1]=0.0;
    for (i=0; i<=n-1; i++)
    {
        distance=sqrt((position.points[0][0]-l.points[i][0])*
                     (position.points[0][0]-l.points[i][0])+(position.points[0][1]-l.points[i][1])*
                     (position.points[0][1]-l.points[i][1]));
        if (distance<min)
        {
            min=distance;
            vec_to_point[0]=position.points[0][0]-l.points[i][0];
            vec_to_point[1]=position.points[0][1]-l.points[i][1];
            *numberofpoint=i+1;
        }
    }
    return(min);
}
```

5. Derivativezero

```

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include "mousesupport.h"
#define iter "iteration.txt"

void fresnel(float x, float *C, float *S);

float derivativezero( Pairs corner, Lines position, float *pp, int numberofpoint, int si)
/* This function computes the extremum of the distance of an arbitrary point in space
to the clothoid by using binary search
dCdp, dSdp: derivatives of fresnel integrals with respect to p
derd2 means derivative of distance squared
si = 1 for point is closest to proximal segment
si = -1 for point is closest to distal segment
a1, a2: starting and end point
numberofpoint: specifies the corner-point that is closest to the arbitrary point*/
{
    int i, xx;
    float C, S, k1, k2, dCdp, dSdp;
    float z, sigma[2], ppp, para, f, fprime;
    float c11[2], c21[2], a1, a2;

    FILE*output_file;
    output_file=fopen(iter,"w");

    if (si==1)
    {
        for (i=0; i<=1; i++)
        {
            c11[i]=corner.c1[i];
            c21[i]=corner.c2[i];
        }
        a1=0;
        a2=corner.p0;
    }
    else if (si==-1)
    {
        for (i=0; i<=1; i++)
        {
            c11[i]=corner.d1[i];
            c21[i]=corner.d2[i];
        }
        a1=corner.p0;
        a2=2*corner.p0;
    }

    //-----

    if (si == 1)
    {
        ppp=(a2+a1)/2;
        fprintf(output_file,"anfang: %f p0= %f 2*p0= %f\n", ppp, a1, a2);
        xx=0;
        do
        {
            xx++;
            para=pi*ppp*ppp/2;
            fresnel(ppp, &C, &S);
            f=(corner.a*C-corner.D)*cos(para)+corner.a*S*sin(para)+
                (corner.point[0]-position.points[0][0])*(cos(para)*c11[0] + sin(para)*c21[0])+
                (corner.point[1]-position.points[0][1])*(cos(para)*c11[1] + sin(para)*c21[1]);
            dCdp=cos(pi*ppp*ppp/2);
            dSdp=sin(pi*ppp*ppp/2);
            fprime=corner.a*dCdp*cos(para)-(corner.a*C-
corner.D)*pi*sin(para)+corner.a*dSdp*sin(para)+
                corner.a*S*pi*cos(para)+

```

```

sin(para)*c11[0])+      pi*ppp*(corner.point[0]-position.points[0][0])*(cos(para)*c21[0]-
sin(para)*c11[1]);      pi*ppp*(corner.point[1]-position.points[0][1])*(cos(para)*c21[1]-
if (fabs(fprime)>2*1e-16)
{
    ppp=f/fprime;
}
else
{
    printf("\nDivide by zero!!\n");
}
fprintf(output_file,"p: %f iteratationen: %d corrector: %f\n", ppp, xx, f/fprime);
}
while(fabs(f/fprime)>1e-3);
}

if (si == -1)
{
    ppp=(a2+a1)/2;
    fprintf(output_file,"anfang: %f p0= %f 2*p0= %f\n", ppp, a1, a2);
    xx=0;
    do
    {
        xx++;
        para=pi/2*(a2-ppp)*(a2-ppp);
        fresnel(a2-ppp,&C, &S);
        f=(corner.D-corner.a*C)*cos(para)-corner.a*S*sin(para)+
          (corner.point[0]-position.points[0][0])*(cos(para)*c11[0]-sin(para)*c21[0])+
          (corner.point[1]-position.points[0][1])*(cos(para)*c11[1]-sin(para)*c21[1]);
        dCdp=cos(para);
        dSdp=sin(para);
        fprime=corner.a+(corner.point[0]-position.points[0][0])*pi*(a2-ppp)*
          (sin(para)*c11[0]+cos(para)*c21[0])+
          (corner.point[1]-position.points[0][1])*pi*(a2-ppp)*
          (sin(para)*c11[1]+cos(para)*c21[1])+
          (corner.D-corner.a*C)*pi*(a2-ppp)*sin(pi/2*(a2-ppp)*(a2-ppp))+
          corner.a*S*pi*(a2-ppp)*cos(pi/2*(a2-ppp)*(a2-ppp));

        if (fabs(fprime)>2*1e-16)
        {
            ppp=f/fprime;
        }
        else
        {
            printf("\nDivide by zero!!\n");
        }
        fprintf(output_file,"p: %f iteratationen: %d corrector: %f\n", ppp, xx, f/fprime);
    }

    while (fabs(f/fprime)>1e-3);
}
fresnel(ppp, &C, &S);
for (i=0; i<=1; i++)
{
    sigma[i]=(corner.point[i]+si*(corner.a*C-corner.D)*c11[i]+
              corner.a*S*c21[i]);
}
k1=position.points[0][0]-sigma[0];
k2=position.points[0][1]-sigma[1];
z=0;
z=sqrt(k1*k1+k2*k2);
fprintf(output_file,"\nznew: %f\n", z);

*pp=ppp;

fclose(output_file);
return(z);
}

```

6. Fresnel

```
#include <math.h>
#include "mousesupport.h"

void fresnel(float x, float *C, float *S)
{
    float r_numerator, r_denominator, r12;
    float a_denominator, a03;
    float helpC, helpS;

    r_numerator=0.506*x+1;
    r_denominator=1.79*x*x+2.054*x+sqrt(2);
    r12=r_numerator/r_denominator;

    a_denominator=0.803*x*x*x+1.866*x*x+2.524*x+2;
    a03=1/a_denominator;

    helpC=sin(0.5*pi*(a03-x*x));
    helpS=cos(0.5*pi*(a03-x*x));
    *C=0.5-r12*helpC;
    *S=0.5-r12*helpS;
}
```

7. Mousesupport

```
/*Bitfields for mouse call mask:
Bit(s)  Description      (Table 2426)
 0      call if mouse moves
 1      call if left button pressed
 2      call if left button released
 3      call if right button pressed
 4      call if right button released
 5      call if middle button pressed (Mouse Systems/Logitech/Genius mouse)
 6      call if middle button released (Mouse Systems/Logitech/Genius mouse)
 7-15   unused
Note:   some versions of the Microsoft documentation incorrectly state that CX
        bit 0 means call if mouse cursor moves
```

(Table 2427)

```
Values interrupt routine is called with:
  AX = condition mask (same bit assignments as call mask)
  BX = button state
  CX = cursor column
  DX = cursor row
  SI = horizontal mickey count
  DI = vertical mickey count*/
#include <graph.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <i86.h>
#include <math.h>
#include "mousesupport.h"

#ifndef FALSE
#define FALSE 0
#define TRUE !FALSE
#endif

int MouseEvent = 0;
int MouseMoved = 0;
int RPressed = 0;
int RReleased = 0;
int LPressed = 0;
int LReleased = 0;
int BPressed = 0;
int MouseAX = 0;
int MouseBX = 0;
int MouseCX = 0;
int MouseDX = 0;

#pragma off (check_stack);
void _loadds far MouseHandler (int max, int mbx, int mcx, int mdx)
{
#pragma aux MouseHandler parm [EAX] [EBX] [ECX] [EDX];
    MouseEvent = 1;
    MouseAX = max;
    MouseBX = mbx;
    MouseCX = mcx;
    MouseDX = mdx;

    if(MouseBX == 0x3)
        BPressed = 1;
    else if(MouseAX == 0x8)
        RPressed = 1;
    else if(MouseAX == 0x10)
        RReleased = 1;
    else if(MouseAX == 0x4)
        LReleased = 1;
    else if(MouseAX == 0x2)
        LPressed = 1;
    else if(MouseAX == 0x1)
        MouseMoved = 1;
}

#pragma on (check_stack);

short InitMouse(void)
```

```

{
    struct SREGS sregs;
    union REGS inregs, outregs;
    struct videoconfig vc;
    int (far *function_ptr)();

    _getvideoconfig( &vc );

    /* check for mouse driver */
    inregs.w.ax = 0;
    int386(0x33, &inregs, &outregs);
    segread(&sregs);

    if(outregs.w.ax == 0x0000)
    {
        _outgtxt( "Mouse NOT installed..." );
        return FALSE;
    }
    else
    {
        /*set the x range of mouse*/
        inregs.w.ax = 0x0007;
        inregs.w.cx = 0x0000;
        inregs.w.dx = (int) vc.numxpixels;
        int386(0x33,&inregs,&outregs);
        /*set the y range of mouse*/
        inregs.w.ax = 0x0008;
        inregs.w.cx = 0x0000;
        inregs.w.dx = (int) vc.numypixels;
        int386(0x33,&inregs,&outregs);

        inregs.w.ax = 0x0001; //show mouse cursor
        int386(0x33,&inregs,&outregs);

        /* install mouse watcher */
        inregs.w.ax = 0xC;
        inregs.w.cx = 0x0014; //call mask: call if anything happens with the mouse
        inregs.w.si = 0xF;
        inregs.w.di = 0xF;
        function_ptr = MouseHandler;
        inregs.x.edx = FP_OFF( function_ptr );
        sregs.es = FP_SEG( function_ptr );
        int386x( 0x33, &inregs, &outregs, &sregs );
        return TRUE;
    }
}

void ExitMouse(void)
{
    union REGS inregs, outregs;

    /* check installation again (to clear watcher) */
    inregs.w.ax = 0;
    int386( 0x33, &inregs, &outregs );
}

int mouse(Lines *l, Lines *c)
{
    char msg[80];
    union REGS inregs, outregs;
    int i, MouseCXold, MouseDXold;
    int rbreleasedold;

    _setvideomode(_VRES16COLOR);

    InitMouse();
    rbreleasedold=RReleased;
    i=0;
    while(!RReleased) //quit with right mousebutton
    {
        show_position(i, *c, *l);

        if(MouseEvent)
        {
            if(LBReleased)
            {
                inregs.w.ax = 0x0002; //hide mouse cursor
                int386(0x33,&inregs,&outregs);

                if (i==0)
                {

```

```

        c->points[0][0]=MouseCX;
        c->points[0][1]=MouseDX;

        l->points[i][0]=MouseCX;
        l->points[i][1]=MouseDX;
        sprintf(msg,"X = %d Y = %d\n", MouseCX, MouseDX);
        _moveto(MouseCX, MouseDX);
    }

    else
    {
        l->points[i][0]=MouseCX;
        l->points[i][1]=MouseDX;
        sprintf(msg,"(%d, %d)", MouseCX, MouseDX);
        _moveto(MouseCXold, MouseDXold);
        _lineto(MouseCX, MouseDX);
    }
    MouseCXold=MouseCX;
    MouseDXold=MouseDX;
    _moveto(MouseCX+20, MouseDX);
    _outgtext(msg);
    LBreleased = 0;
    i++;
}
MouseEvent = 0;
inregs.w.ax = 0x0001; //show mouse cursor
int386(0x33,&inregs,&outregs);
}
}
RBreleased=rbreleasedold;
inregs.w.ax = 0x0002; //hide mouse cursor
int386(0x33,&inregs,&outregs);

ExitMouse();

return(i);
}

void mouse1(Lines *a, Lines c, Lines l)
{
    char msg[80];
    union REGS inregs, outregs;
    short flag;
    //Lines l;

InitMouse();
    flag=TRUE;

    while(flag)
    {
        //show_position(l, c, l);
        if(MouseEvent)
        {
            if(LBreleased)
            {
                inregs.w.ax = 0x0002; //hide mouse cursor
                int386(0x33,&inregs,&outregs);
                a->points[0][0]=MouseCX;
                a->points[0][1]=MouseDX;
                sprintf(msg,"(%d, %d)", MouseCX, MouseDX);
                flag=FALSE;
            }
            _moveto(MouseCX+20, MouseDX);
            _outgtext(msg);
            MouseEvent=0;
        }
    }

ExitMouse();
}

void show_position(int i, Lines c, Lines l)
{
    // shows current global position of the mouse in the upper right corner of the screen
    char msg[80];
    union REGS inregs, outregs;
    int oldx, oldy;
    float length;

```

```

inregs.w.ax=0x0003; // get current mouse position
int386(0x33, &inregs, &outregs);

if (i==0)
{
    sprintf(msg,"x:      y:      ");
    sprintf(msg,"x: %d    y: %d", outregs.w.cx, outregs.w.dx);
    _settextposition(1,60);
    _outtext(msg);
}
else
{
    length=sqrt( (outregs.w.cx-l.points[i-1][0])*(outregs.w.cx-l.points[i-1][0])+
                (outregs.w.dx-l.points[i-1][1])*(outregs.w.dx-l.points[i-1][1]) );
    sprintf(msg,"x:      y:      length:      ");
    sprintf(msg,"x: %d    y: %d    length: %f", outregs.w.cx, outregs.w.dx, length);
    _settextposition(1,40);
    _outtext(msg);
}
}

```

8. Mousesupport.h

```
#ifndef MOUSESAMPLE_H
#define MOUSESAMPLE_H

#define numofpoints 10
#define pi 3.14159265

typedef struct {
    int points[numofpoints][2];
} Lines;
typedef struct {
    int point[2];
    float t_prime[2], t_g[2], norm_tpr, norm_tg, alpha, p0, a, D, c1[2], c2[2],
        d1[2], d2[2], sigma_prox[50][2],
        sigma_dist[50][2], rho, s[2];
} Pairs;
typedef struct {
    int point[2];
    float tangent[2], curvature;
} pointOnElem;

void _loadds far MouseHandler (int max, int mbx, int mcx, int mdx);
short InitMouse(void);
void ExitMouse(void);
int mouse(Lines *l, Lines *c);
void mouse1(Lines *a, Lines c, Lines l);
void show_position(int i, Lines c, Lines l);

#endif
```