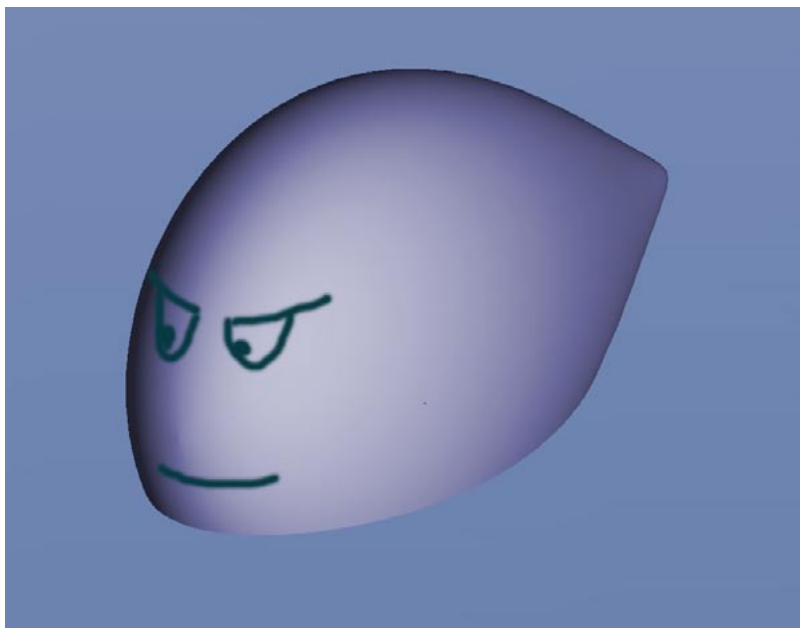
Gernot Hoffmann

Equations of Motion for a Masspoint



This document started as description
of the Lagrangian equations
for a moving masspoint

Later some methods for terrain
interpolation were added

Settings for Acrobat

Edit / Preferences / General / Page Display (since version 6)
Custom Resolution 72 dpi and use zoom 200% for screenshots

Edit / Preferences / General / Color Management (full version only)
sRGB
EuroscaleCoated or ISOCoated or SWOP
GrayGamma 2.2

Equations of Motion for a Masspoint

Contents

Equations of Motion for a Masspoint

The document describes the derivation of the differential equations for the motion of a mass point in a hill landscape, a terrain.

The derivation is based on Lagrangian Dynamics. It includes gravitational forces, viscous drag and Coulomb friction (modeled rather simply). Furtheron, a positive or negative thrust pulse can be applied.

Two computer graphics show trajectories without and with viscous drag or friction. The masspoint has the shape of a mouse and is automatically aligned to the trajectory and the local normal vector, using three Euler angles, the aircraft sequence with z-axis upwards.

Some source code fragments may help to understand the application of the differential equations.

The numerical integration is performed by ’False Euler’, which uses new velocities as inputs for the calculation of the coordinates.
Standard Euler would use only old values on the right side.

In the last chapters the interpolation for terrain data is described and illustrated for the case that height values are given in a grid.

The PDF document is pixel synchronized for zoom 200%.
If Acrobat Reader is called by a browser, then minor disturbances may occur. Please use direct view by Acrobat.

1. Equations of Motion

1.1 Lagrangian equations

Given is a terrain z = z(x,y).

Generalized coordinates are $q_1 = x$ and $q_2 = y$.

Kinetic energy:

$$T = 0.5m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) = 0.5m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2)$$

$$\dot{z} = z_x\dot{x} + z_y\dot{y}$$

$$T = 0.5m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) = 0.5m(\dot{x}^2 + \dot{y}^2 + (z_x\dot{x} + z_y\dot{y})^2)$$

Potential energy:

$$V = mgz = mg(z_x x + z_y y)$$

Dissipation function R see below.

Lagrangian equations:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_k} - \frac{\partial T}{\partial q_k} = -\frac{\partial V}{\partial q_k} - \frac{\partial R}{\partial \dot{q}_k} + F_{q_k}$$

$$m((1+z_x^2)\ddot{x} + z_x z_y \ddot{y}) = -mgz_x + F_x$$

$$m(z_x z_y \ddot{x} + (1+z_y^2)\ddot{y}) = -mgz_y + F_y$$

Expressions with derivatives $\dot{z}_x$ and $\dot{z}_y$ were neglected.

$F_x = mf_x$ and $F_y = mf_y$ are the forces due to Coulomb or viscous friction (next page):

$$\begin{aligned}(1+z_x^2)\ddot{x} + z_x z_y \ddot{y} &= -gz_x + f_x \\ z_x z_y \ddot{x} + (1+z_y^2)\ddot{y} &= -gz_y + f_y\end{aligned}$$

Solving this by Cramer we find the decoupled accelerations:

$$\ddot{x} = \frac{-gz_x + f_x(1+z_x^2) - f_y z_x z_y}{1+z_x^2+z_y^2}$$

$$\ddot{y} = \frac{-gz_y + f_y(1+z_y^2) - f_x z_x z_y}{1+z_x^2+z_y^2}$$

1. Equations of Motion

1.2 Calculation of friction forces

Normal unit vector:

$$\mathbf{n} = \begin{bmatrix} -z_x \\ -z_y \\ 1 \end{bmatrix} \frac{1}{\sqrt{z_x^2 + z_y^2 + 1}}$$

Normal component of gravity force:

$$\mathbf{N} = (\mathbf{G}^T \mathbf{n})\mathbf{n} = ((0, 0, -mg)\,\mathbf{n})\mathbf{n}$$

Absolute value of normal force:

$$N = \frac{mg}{\sqrt{z_x^2 + z_y^2 + 1}}$$

Coulomb friction force:

$$\mathbf{F} = -\mu N \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \frac{1}{\sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}}$$

Singularity for zero velocity (reversed sign).

Replace $\dot{z} = (z_x \dot{x} + z_y \dot{y})$

$$F_{q_k} = F_x \frac{\partial x}{\partial q_k} + F_y \frac{\partial y}{\partial q_k} + F_z \frac{\partial z}{\partial q_k}$$

$q_1 = x$

$$F_x := F_x + 0 + F_z \frac{\partial z}{\partial x} = F_x + F_z z_x$$

$q_2 = y$

$$F_y := 0 + F_y + F_z \frac{\partial z}{\partial y} = F_y + F_z z_y$$

Use new components F_x, F_y for Coulomb friction.

Dissipation function for viscous drag:

$$R = 0.5\,d(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) = 0.5\,d(\dot{x}^2 + \dot{y}^2 + (z_x \dot{x} + z_y \dot{y})^2)$$

Friction force for viscous drag:

$$F_x = -d((1 + z_x^2)\dot{x} \;+\; z_x z_y \dot{y}\;)$$

$$F_y = -d(\; z_x z_y \dot{x} \;+ (1 + z_y^2)\dot{y}\;)$$

1. Equations of Motion

1.3 Calculation of Euler angles

The yaw angle or azimuth is defined by the horizontal components of the velocity:

$$\tan\psi = \frac{V_y}{V_x}$$

The object normal should be aligned to the surface normal.
The rotation matrix for aircraft angles is used.
Index 1 means ground fixed, index 4 means body fixed.
The coordinate transformation $\mathbf{x}_4 = \mathbf{A}_{41}\mathbf{x}_1$ defines the relation between column matrices $\mathbf{x}_4, \mathbf{x}_1$.
for the same physical vector $\mathbf{x}$ in rotated coordinate systems.

$$\mathbf{A}_{41} = \begin{bmatrix} +\cos\theta\cos\psi & +\cos\theta\sin\psi & -\sin\theta \\ -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & +\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & +\sin\phi\cos\theta \\ +\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & +\cos\phi\cos\theta \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \mathbf{A}_{41} \begin{bmatrix} -z_x \\ -z_y \\ 1 \end{bmatrix} \frac{1}{\sqrt{1 + z_x^2 + z_y^2}}$$

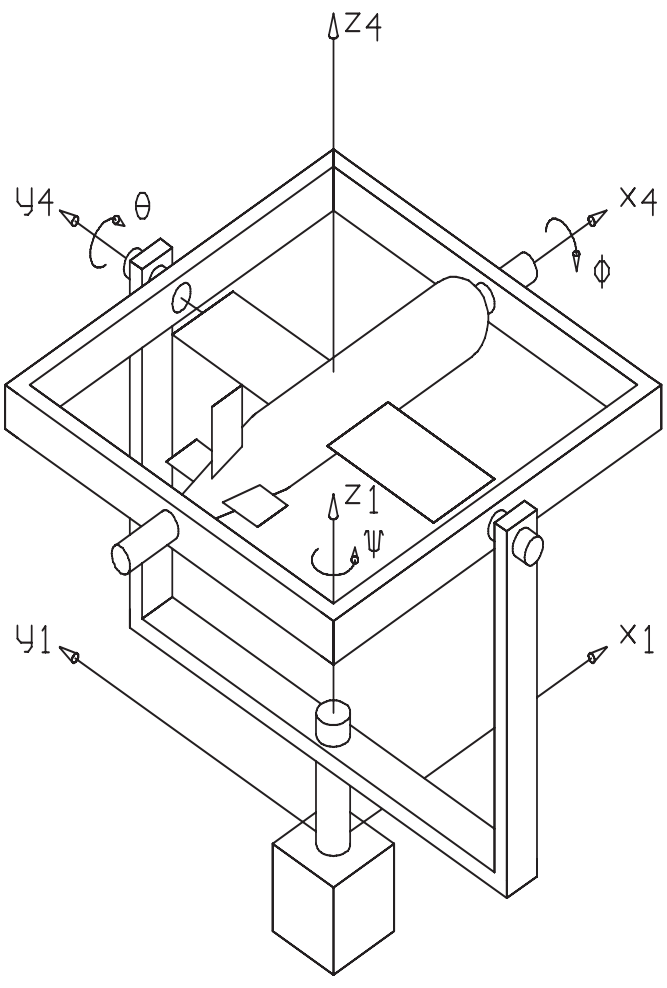
The first and the second equation deliver the pitch angle θ and the roll angle ϕ.

$$\tan\theta = -z_x\cos\psi - z_y\sin\psi$$

$$\tan\phi = (-z_x\sin\psi + z_y\cos\psi)\cos\theta$$

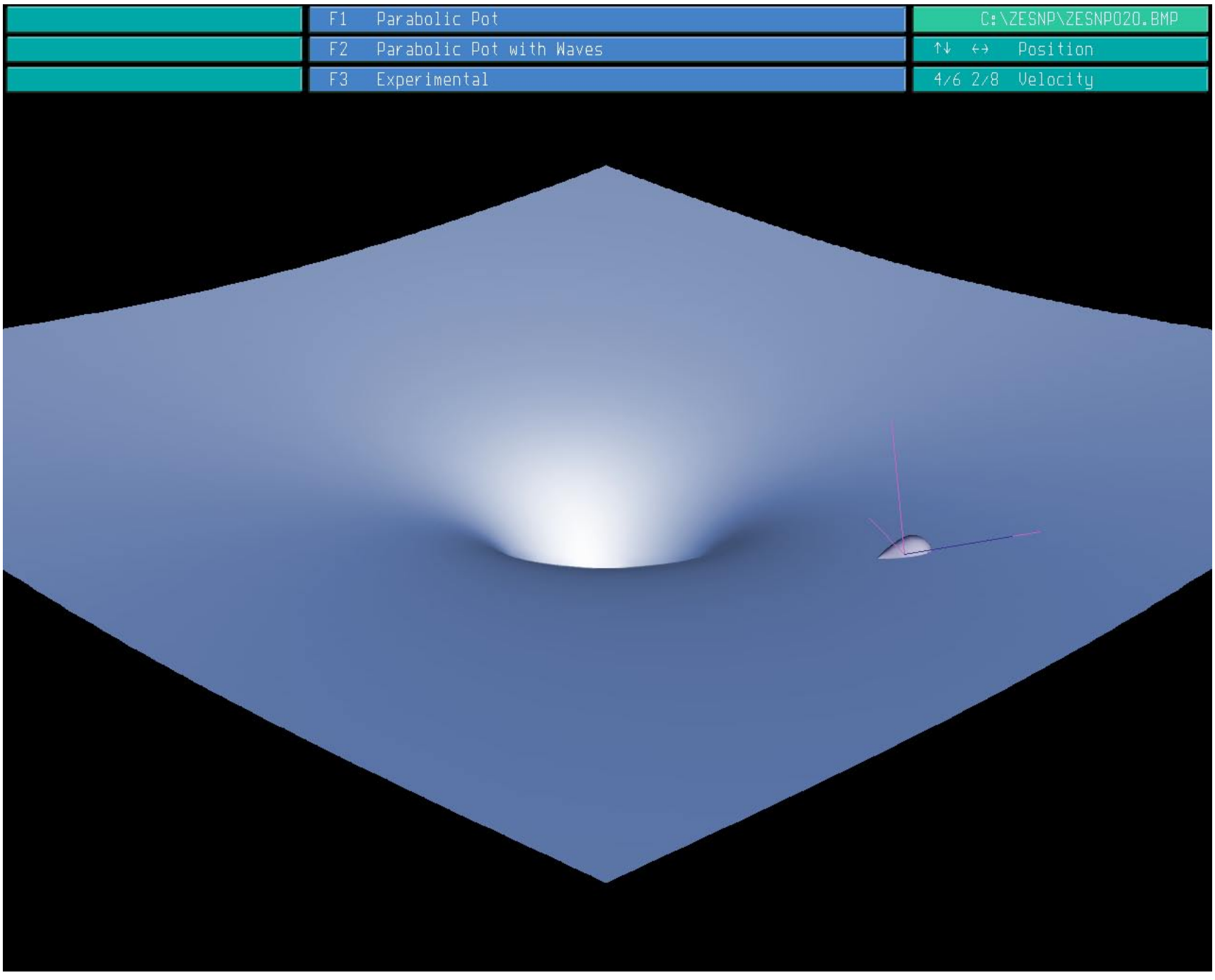
Aircraft standards use x_4 forward, y_4 to the right and z_4 downwards.
If the ground fixed system is aligned similarly, then the same matrix as above can be used.

For convenience we have here the z-axis upwards.



2. Simulation

2.1 Setting initial conditions

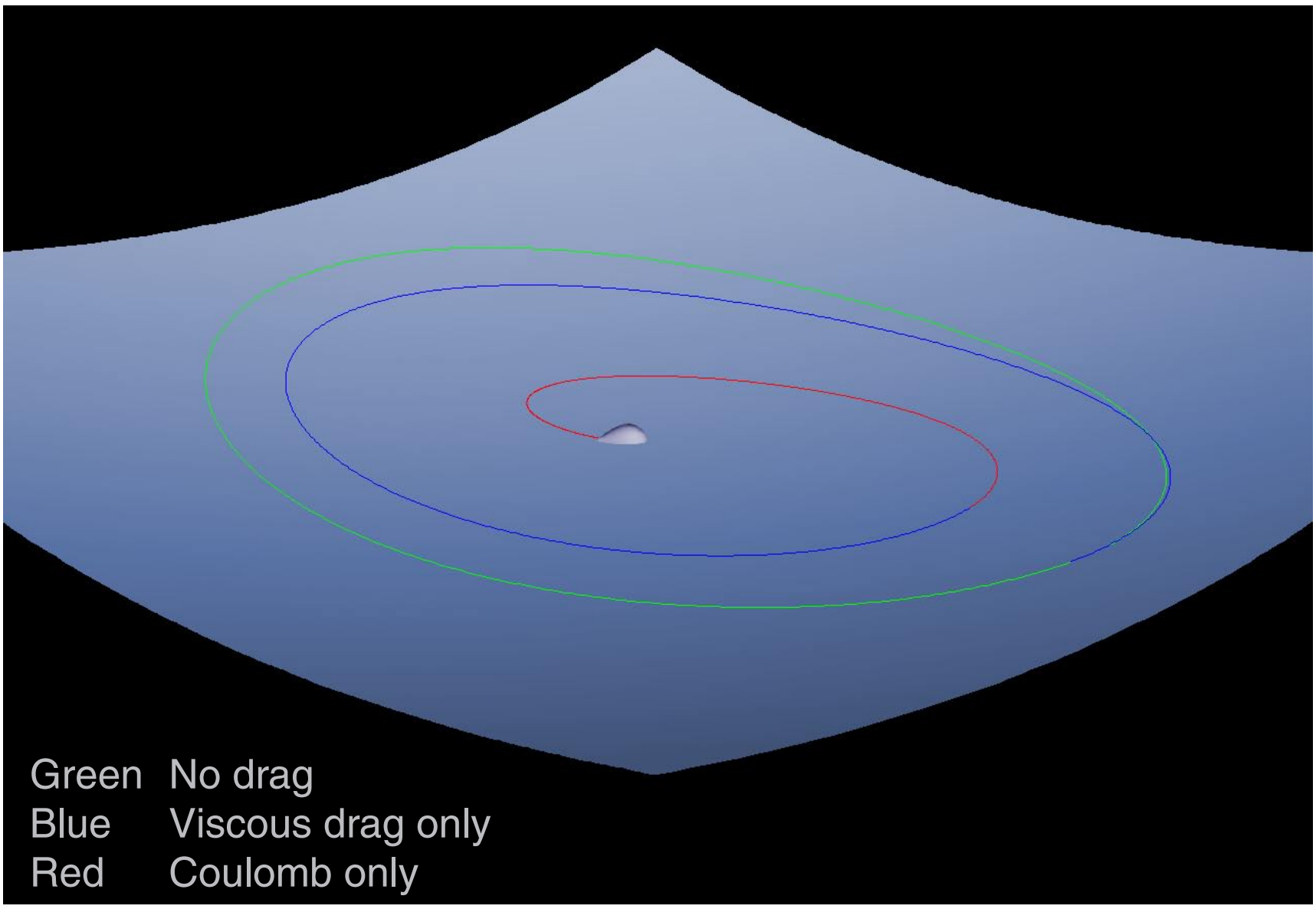


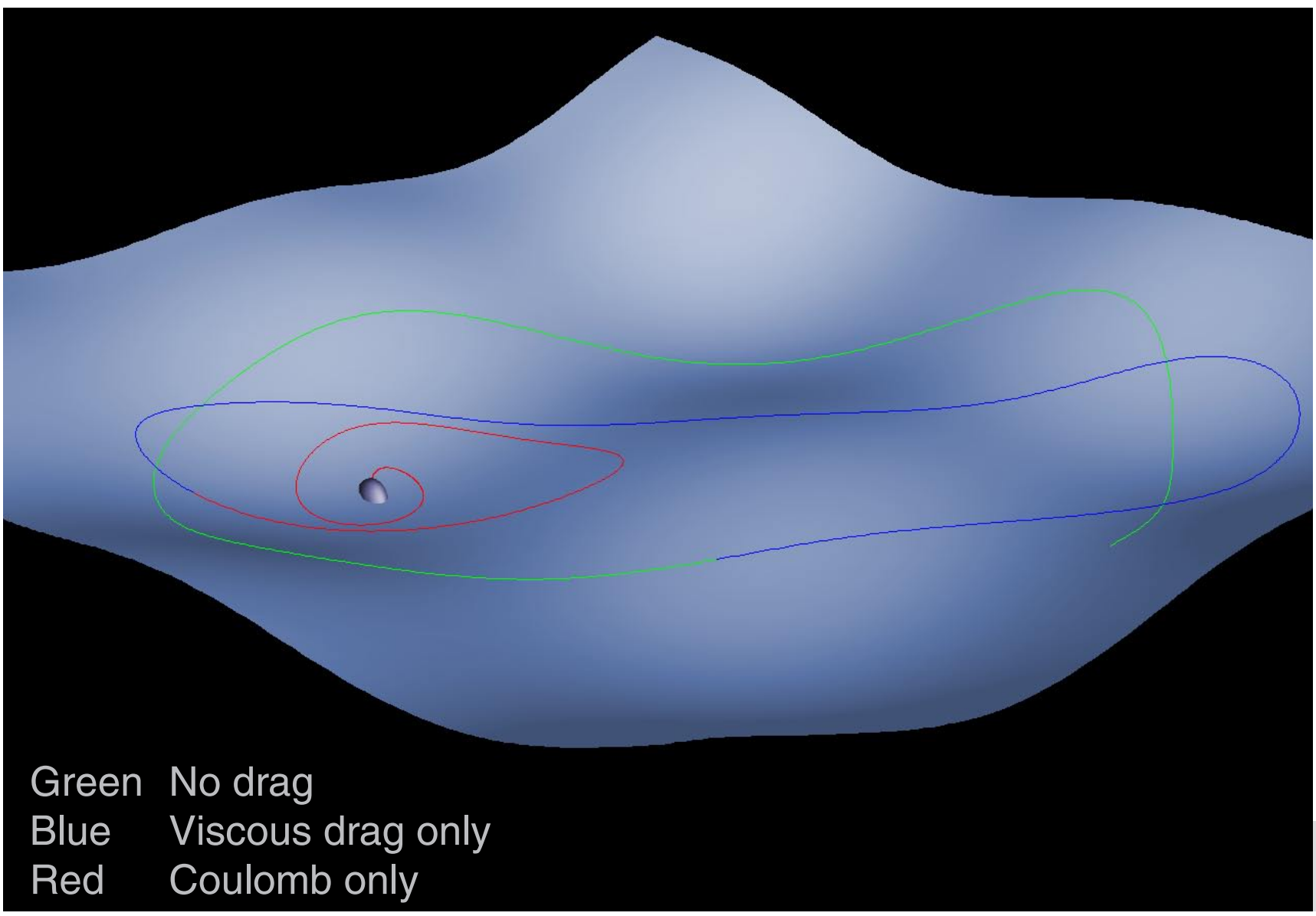
Initial conditions position and velocity

Velocity by heading and value

2. Simulation

2.2 Trajectories in a parabolic pot and in a wavy pot





2. Simulation

2.3 Trajectory of a celestial body

The gravitational attraction force is proportional to the two masses
and inverse proportional to the square of the distance.
For simplicity without masses and gravitational constant:

$$r = \sqrt{x^2+y^2}$$

$$F_g = -1/r^2$$

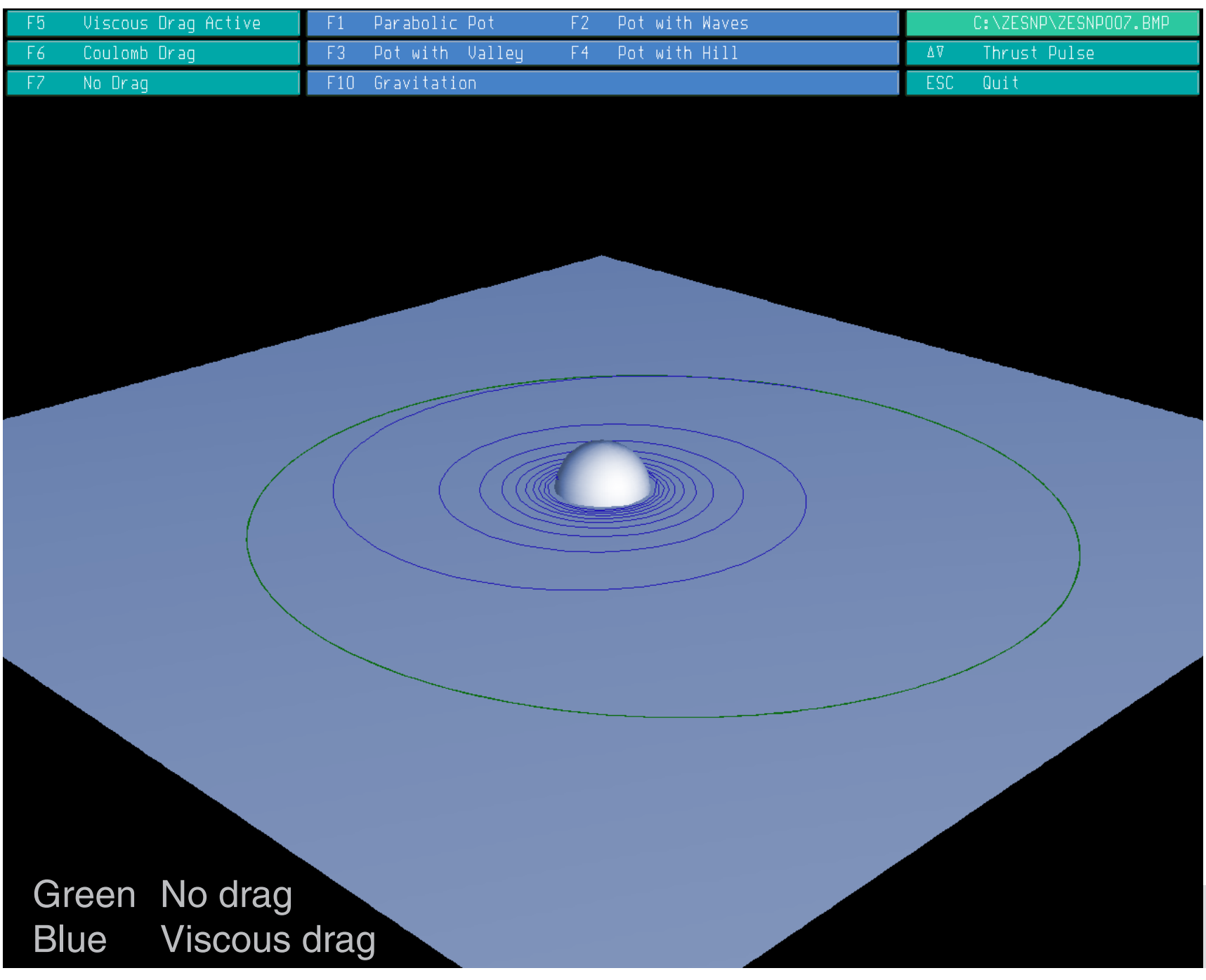
$$F_x = F_g \cos(\alpha) = F_g\, x/r$$

$$F_y = F_g \sin(\alpha) = F_g\, y/r$$

$$F_x = -x/r^3$$

$$F_y = -y/r^3$$

This example is not included in the source code.
It shows the rather good quality of the False Euler integration.



2. Simulation

2.4.1 Source code

```
Procedure FDeriv(LType: Integer; x,y: Double; Var zx,zy: Double);
{ Derivatives }
Const   ds=1E-4; s2=0.5/ds;
Var z1,z2: Double;
Begin
FValue(LType,x+ds,y,z2);
FValue(LType,x-ds,y,z1);
zx:=s2*(z2-z1);
FValue(LType,x,y+ds,z2);
FValue(LType,x,y-ds,z1);
zy:=s2*(z2-z1);
End;

Procedure Angles(Vp: XYZ; zx,zy: Double);
{ Calculate Rotation matrix }
Var Psi,The,Phi             : Double;
       sPsi,cPsi,sThe,cThe    : Double;
Label Ex;
Begin
 If (Abs(Vp.x)>0) Or (Abs(Vp.y)>0) Then
 Begin
 Psi:=atan2(Vp.y,Vp.x);
 Siccoc(Psi,sPsi,cPsi);
 The:=atan2(-zx*cPsi-zy*sPsi,1);
 Siccoc(The,sThe,cThe);
 Phi:=atan2((-zx*sPsi+zy*cPsi)cThe,1);
 MatObj3D(Psi,The,Phi);
 End;
End;

Procedure Integ;
{ Mass point in hill landscape                }
Const   gr =9.810; { Gravity constant         }
        vd =0.200; { Viscous drag coefficient }
        mu =0.050; { Coulomb friction mue     }
        min=0.005; { Stop velocity            }
Var dTn,fx,fy,fz,fcx,fcy,fcz,Nf,Vn,Na             : Double;
        z1,zx,zy,zx2,zy2,zxy,sPsi,cPsi,sThe,cThe  : Double;
Begin
 T:=T+dT;
 fx:=0; fy:=0;
 FDeriv(LType,xp.x,xp.y,zx,zy);
 zx2:=Sqr(zx);
 zy2:=Sqr(zy);
 zxy:=zx*zy;
 dTn:=dT/(1+zx2+zy2);
```

2.4.2 Source code

```
{ Viscous drag        }
If Dcon Then
 Begin
   fx:=fx-vd*((1+zx2)*Vp.x +zxy*Vp.y);
   fy:=fy-vd*((1+zy2)*Vp.y +zxy*Vp.x);
 End;
{ Coulomb friction    }
If Ccon Then
 Begin
   With Vp Do Vn:=Sqrt(Sqr(x)+Sqr(y)+Sqr(zx*x+zy*y)+0.0001);
   Nf := mu*gr/Sqrt(1+zx2+zy2);
   Vn :=-Nf/Vn;
   fcx:= Vn*Vp.x;
   fcy:= Vn*Vp.y;
   fcz:= Vn*Vp.z;
   fx:=fx+fcx+fcz*zx;
   fy:=fy+fcy+fcz*zy;
  End;
{ Thrust }
 If fc<>0  Then
  Begin
   With Vp Do Vn:=Sqrt(Sqr(x)+Sqr(y)+Sqr(zx*x+zy*y)+0.0001);
   Vn :=+fc/Vn;
   fcx:= Vn*Vp.x;
   fcy:= Vn*Vp.y;
   fcz:= Vn*Vp.z;
   fx:=fx+fcx+fcz*zx;
   fy:=fy+fcy+fcz*zy;
  End;
{ dVp/dt = Ap }
  With Vp Do
   Begin
   x:=x+dTn*(-gr*zx+fx*(1+zx2)-fy*zxy);
   y:=y+dTn*(-gr*zy+fy*(1+zy2)-fx*zxy);
   End;
{ dXp/dt = Vp }
  With Xp Do
   Begin
   x:=x+dT*Vp.x;
   y:=y+dT*Vp.y;
   FValue(LType,x,y,z1);
   z:=z1;
   End;
 { Rotation matrix, angles, aligned to velocity and to normal }
   Angles(Vp,zx,zy);
End;
```

3. Interpolation of Terrain Data

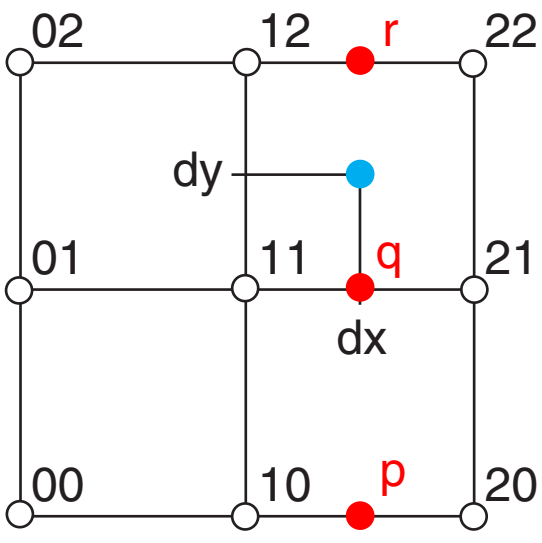
3.1 Biquadratic interpolation

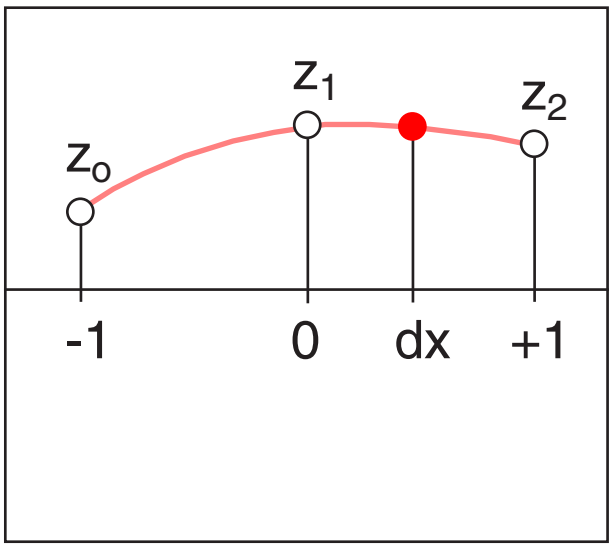
The landscape or terrain $z=z(x,y)$ is given in a grid x_i,y_k by $z_{ik}=z(x_i,y_k)$.
The first step is the normalization for unit steps in each direction.
In order to avoid negative indices we number the points by 00,10,..,22
as shown in the diagram.
We may want to calculate interpolated values and normal vectors by
biquadratic interpolation.
For a paraboloid we need only the center point x_1,y_1 plus five points, but
it is probably better to use all eight neighbours.
dx and dy are the non-integer deviations from the center point.
This interpolation can be performed by two methods:

a) Gauss Least Squares paraboloid
b) Sequential Interpolation in both directions

3.2 Gauss least squares interpolation

$f(dx,dy) = z_{11}+a_1dx^2+a_2dy^2+a_3dx\,dy+a_4dx+a_5dy$
Write down 8 equations for 8 neighbour points.
This results in 8 linear equations for 5 unknowns $\mathbf{a}=(a_1,a_2,...,a_5)^T$.
$\mathbf{Ma} = \mathbf{z}$
The system can be solved by the Gauss Normal Equation:
$\mathbf{M}^T\mathbf{M}\,\mathbf{a} = \mathbf{M}^T\mathbf{z}$
The normal vector is simply:
$n_x = -f_x = -(2a_1dx+a_3dy+a_4)$
$n_y = -f_y = -(2a_2dy+a_3dx+a_5)$
$n_z = 1$





3. Interpolation of Terrain Data

3.3 Sequential interpolation

First we calculate p,q,r in x-direction, figure last page, then we interpolate these values in y-direction.

From the right figure we derive easily three equations for the one-dimensional quadratic function:

$$z(dx) = a_o + a_1 dx + a_2 dx^2$$

$$\begin{aligned} z_o &= a_o - a_1 + a_2 \\ z_1 &= a_o \\ z_2 &= a_o + a_1 + a_2 \end{aligned}$$

$$\begin{aligned} a1 &= 0.5(z_2 - z_o) \\ a2 &= 0.5(z_2 + z_o) - z_1 \end{aligned}$$

Just the same procedure is applied for the normal vector component z_x.
For z_y we have to start the whole process once again, but now in vertical direction.
This is better understandable in the source code example.

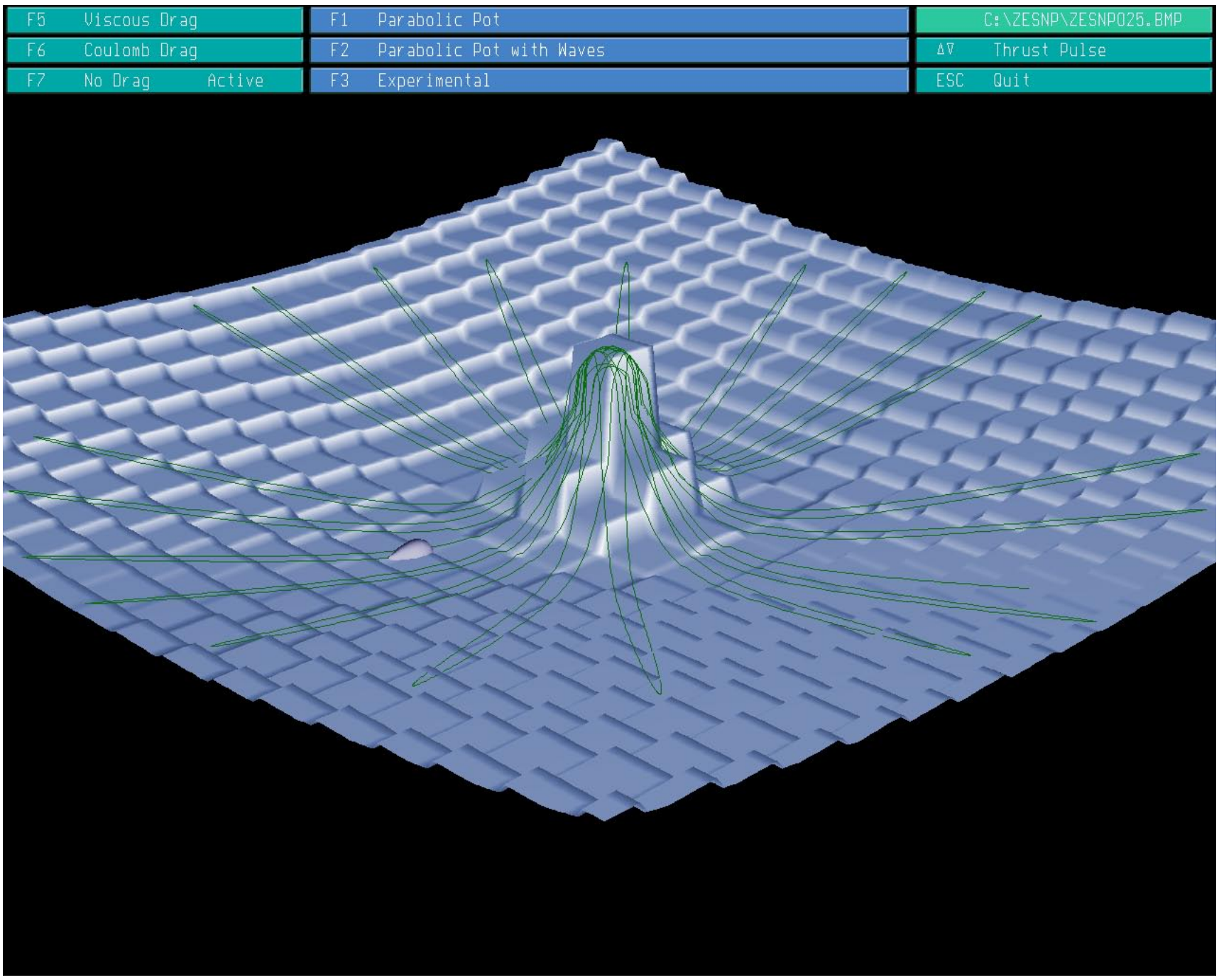
3.4 Comparison of the two methods

By Gauss Least Squares we would fill an array with interpolated normal vectors.
This is time consuming because of the matrix operations and the equation solver.
Then we can interpolate in the actual task these normal vectors very fast linearly.

By Sequential Interpolation we can calculate the value and the normal vector
at any position between the table grid values.
Experiments have shown that this is rather fast, therefore better than Gauss L.S.
Of course we can use Sequential Interpolation also for filling a table of normals.

3. Interpolation of Terrain Data

3.5 Trajectories on interpolated terrain



Shows so far the *not interpolated* terrain
(renderer applies some smoothing graphically)

Shows motion on the *interpolated* surface,
using interpolated height and normal

The green trajectory is here z-buffered

3. Interpolation of Terrain Data

3.6.1 Source code

```
Procedure Biquad(grid: Integer; x,y: Double; Var z,zx,zy: Double);
{   Interpolated Value and Normal for ZIK=f(xi,yk) }
{   E.g. grid=10 divisions in each direction
    x,y = -1..+1 mapped to —> -10..+10    }
Var     x11,y11                              : Integer;
        dx,dy,dx2,dy2,a1,a2,p,q,r            : Double;
        z0,z1,z2,zxp,zxq,zxr,zyp,zyq,zyr     : Double;
Begin
x:=grid*x;
y:=grid*y;
x11:=Round(x);
y11:=Round(y);
dx:=x-x11;
dy:=y-y11;
dx2:=Sqr(dx);
dy2:=Sqr(dy);
{ For z and zx in x-direction }
z0:=ZIK^[x11-1,y11-1];  {Ignore Pointer ^, use ordinary memory ZIK(i,k)}
z1:=ZIK^[x11  ,y11-1];
z2:=ZIK^[x11+1,y11-1];
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
p :=z1+a1*dx+a2*dx2;
zxp:=a1+2*a2*dx;
z0:=ZIK^[x11-1,y11  ];
z1:=ZIK^[x11  ,y11  ];
z2:=ZIK^[x11+1,y11  ];
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
q :=z1+a1*dx+a2*dx2;
zxq:=a1+2*a2*dx;
z0:=ZIK^[x11-1,y11+1];
z1:=ZIK^[x11  ,y11+1];
z2:=ZIK^[x11+1,y11+1];
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
r :=z1+a1*dx+a2*dx2;
zxr:=a1+2*a2*dx;
{ Interpolate values in y-direction }
z0:=p;
z1:=q;
z2:=r;
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
z :=z1+a1*dy+a2*dy2;
```

3. Interpolation of Terrain Data

3.6.2 Source code

```
{ Interpolate zx in y-direction }
z0:=zxp;
z1:=zxq;
z2:=zxr;
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
zx:=(z1+a1*dy+a2*dy2)*grid;   { Scaling of derivative zx by grid }
{ For zy in y-direction }
z0:=ZIK^[x11-1,y11-1];
z1:=ZIK^[x11-1,y11  ];
z2:=ZIK^[x11-1,y11+1];
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
zyp:=a1+2*a2*dy;
z0:=ZIK^[x11  ,y11-1];
z1:=ZIK^[x11  ,y11  ];
z2:=ZIK^[x11  ,y11+1];
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
zyq:=a1+2*a2*dy;
z0:=ZIK^[x11+1,y11-1];
z1:=ZIK^[x11+1,y11  ];
z2:=ZIK^[x11+1,y11+1];
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
zyr:=a1+2*a2*dy;
{ Interpolate zy in x-direction }
z0:=zyp;
z1:=zyq;
z2:=zyr;
a1:=0.5*(z2-z0);
a2:=0.5*(z2+z0)-z1;
zy:=(z1+a1*dx+a2*dx2)*grid;
End;
```

3. Interpolation of Terrain Data

3.6.3 Source code, optimized

```
Procedure Biquad(grid1: Integer; x,y: Double; Var z,zx,zy: Double);
{       Interpolated Value and Normal for ZIK=f(xi,yk)
        E.g. grid1=10 divisions in each direction
        x = xmin..xmax mapped to –> -10..+10
        y = ymin..ymax mapped to –> -10..+10
        Export Double because of other procedures
        Optimized version                                          }
Var     i,k,i1,i2                          : Integer;
        dx,dy,d2x,d2y,a1,a2,p,q,r          : Single;
        zxp,zxq,zxr,zyp,zyq,zyr            : Single;
        z00,z10,z20,z01,z11,z21,z02,z12,z22  : Single;
Begin
x:=grid1*x; i:=Round(x); dx:=x-i; d2x:=2*dx;
y:=grid1*y; k:=Round(y); dy:=y-k; d2y:=2*dy;
Dec(i); Dec(k); i1:=i+1; i2:=i+2;
{ Ignore Pointer ^, use ordinary memory ZIK(i,k) }
z00:=ZIK^[i,k];    z10:=ZIK^[i1,k];   z20:=ZIK^[i2,k]; Inc(k);
z01:=ZIK^[i,k];    z11:=ZIK^[i1,k];   z21:=ZIK^[i2,k]; Inc(k);
z02:=ZIK^[i,k];    z12:=ZIK^[i1,k]; z22:=ZIK^[i2,k];
{ 3 times in x-direction, row 0,1,2    for z and zx }
a1:=0.5*(z20-z00);
a2:=  a1+z00-z10;
p :=z10+(a1+a2*dx)*dx;
zxp:=a1+a2*d2x;
a1:=0.5*(z21-z01);
a2:=  a1+z01-z11;
q :=z11+(a1+a2*dx)*dx;
zxq:= a1+a2*d2x;
a1:=0.5*(z22-z02);
a2:=a1+z02-z12;
r :=z12+(a1+a2*dx)*dx;
zxr:= a1+a2*d2x;
{ In y-direction for z and zx }
a1:=0.5*(r-p);
a2:=a1+p-q;
z :=q+(a1+a2*dy)*dy;
a1:=0.5*(zxr-zxp);
a2:=  a1+zxp-zxq;
zx:=(zxq+(a1+a2*dy)*dy)*grid1;
```

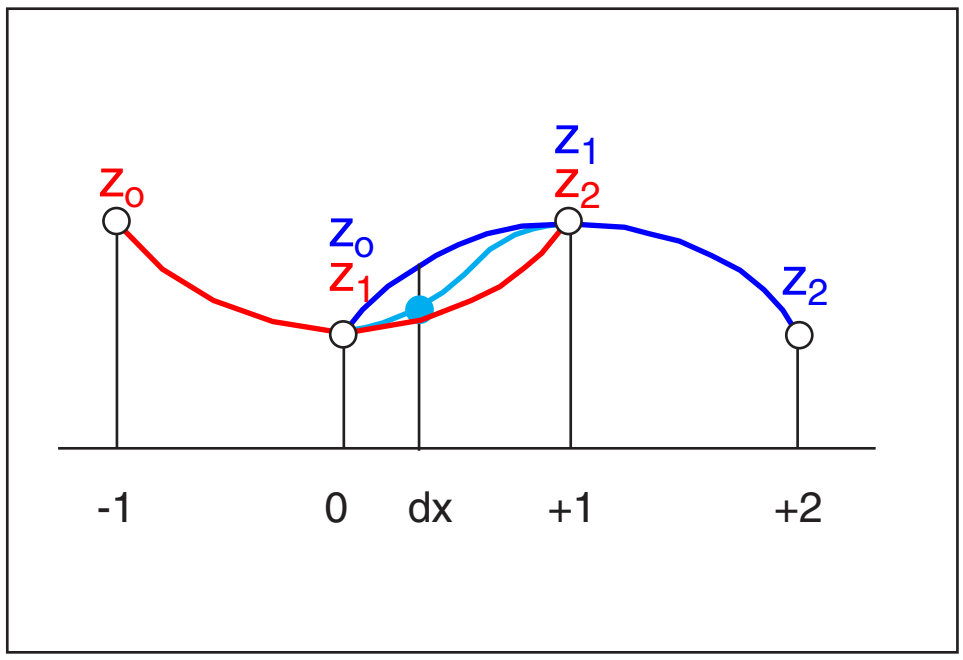
3.6.4 Source code, optimized

```
{ 3 times in y-direction for zy }
a1:=0.5*(z02-z00);
a2:=  a1+z00-z01;
zyp:=a1+a2*d2y;
a1:=0.5*(z12-z10);
a2:=  a1+z10-z11;
zyq:=a1+a2*d2y;
a1:=0.5*(z22-z20);
a2:=  a1+z20-z21;
zyr:=a1+a2*d2y;
{ In x-direction for zy }
a1:=0.5*(zyr-zyp);
a2:=  a1+zyp-zyq;
zy:=(zyq+(a1+a2*dx)*dx)*grid1;
End;
```

4. Biquadratic Interpolation with Blending

4.1 Linear blending

The biquadratic interpolation is by no means perfect.
Discontinuities appear halfway between the gridpoints, as shown by the red and the blue parabola.
For dx<0.5 the red curve is valid.
For dx>0.5 the center is shifted one step to the right and the blue curve is valid.



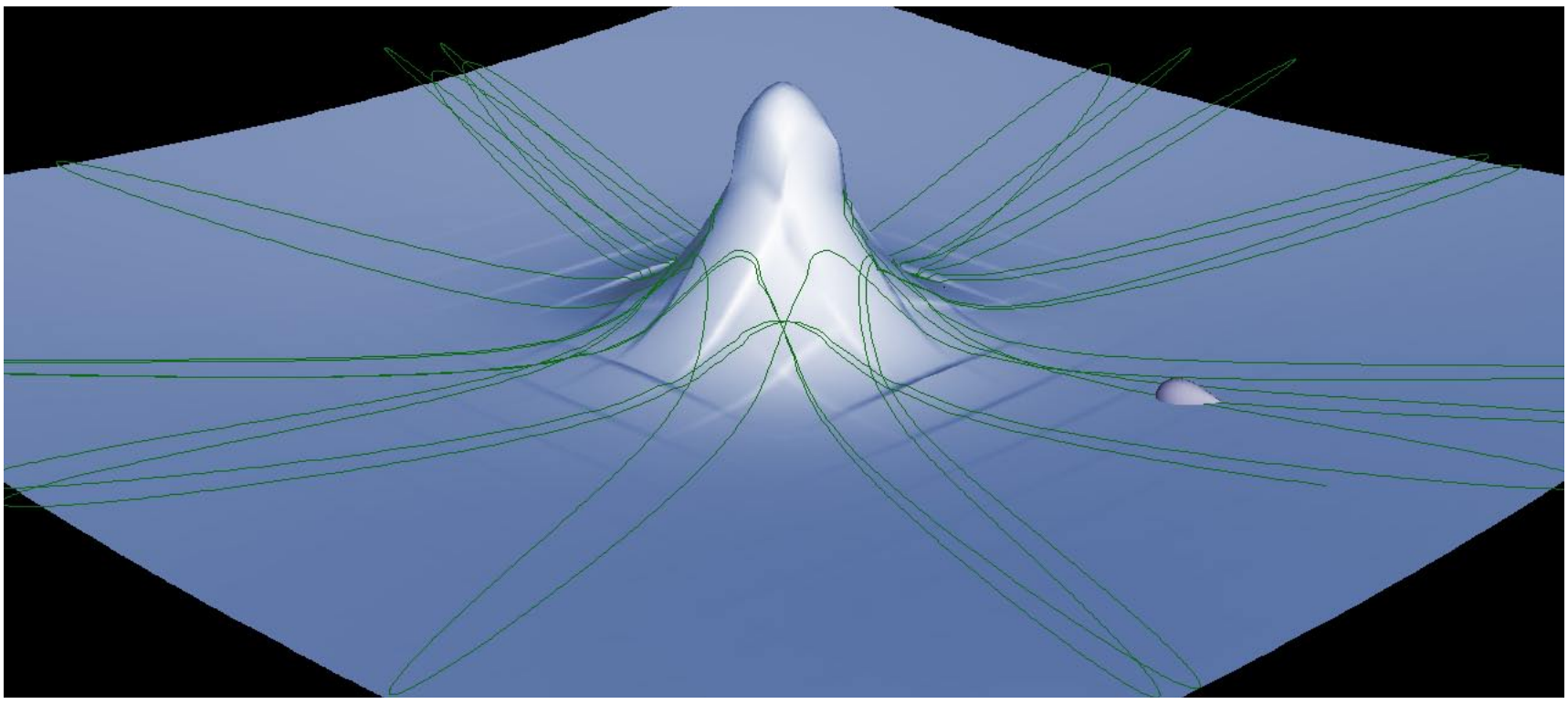
We exclude here the time consuming spline interpolation (chapter 5), which delivers the best results. We apply simply a linear interpolation of adjacent paraboloids. The direction of the shift depends on the sign of dx and dy.
Then we have to calculate z_a as usual, z_b shifted one step to the right (dx>0) or to the left (dx<0) and z_c shifted one step up (dy>0) or down (dy<0) in x,y-coordinates.
These three values are blended according to the absolute values of dx and dy, using a plane equation with three triangle corners z_a, z_b, z_c.
Linear blending blends paraboloids, therefore it does not generate planes, for the general case. The paraboloids can be hyperbolic.
The code example may make the algorithm clear and also the correct shift directions.
On the next page, the straight biquadratic surface and the blended surface is shown.
The true function is smooth, a parabolic or conical pot plus a volcano:
$f(x,y) = a(x^2+y^2)+b/(x^2+y^2+c)$.

The normals are blended as well.

One may prefer a simplified interpolator for the value only and apply then a numerical differentiation (chapter 4.3.5). This code is also better understandable and should be used as a basis for experiments.
The numerical differentiation has a worse quality compared to the direct approach.
This can be shown by graphics of $Sqrt(z_x^2+z_y^2)$

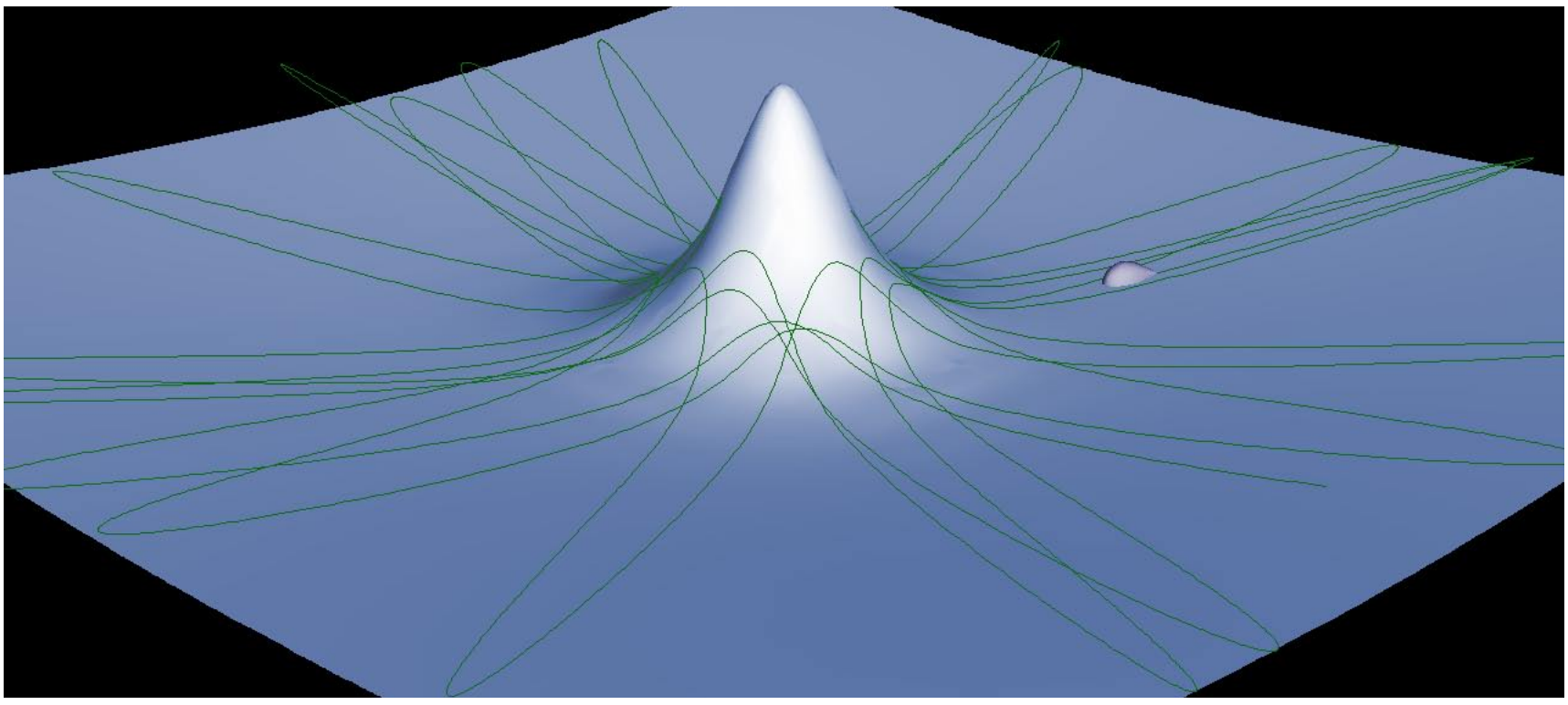
4. Biquadratic Interpolation with Blending

4.2 Graphics without and with linear blending

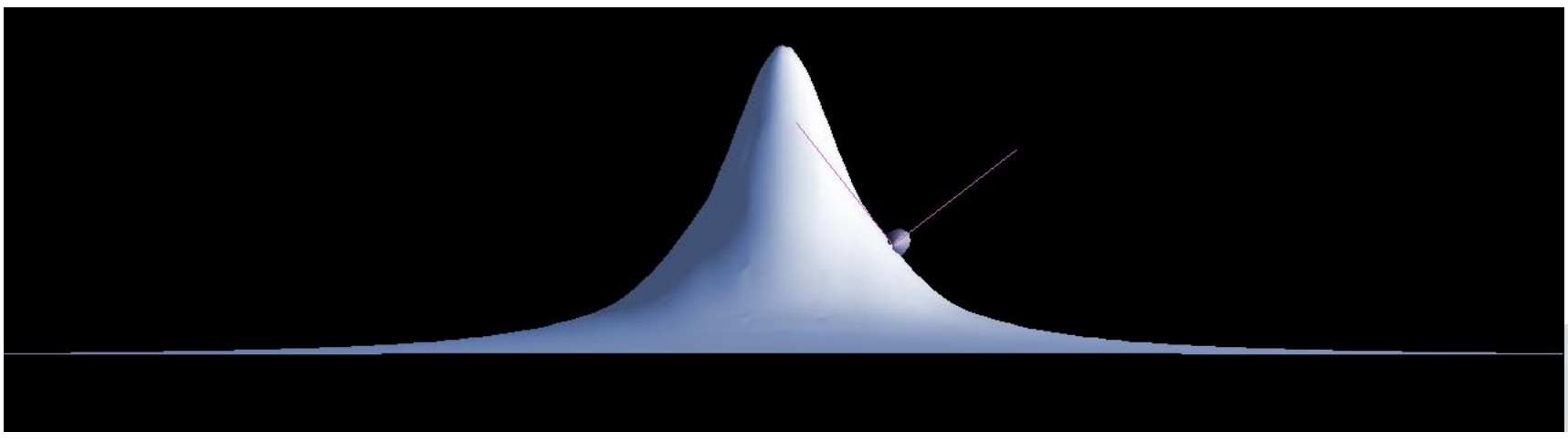


Interpolated surface without blending

With blending



Testing the normal vector



4. Biquadratic Interpolation with Blending

4.3.1 Source code with blending

```
Procedure Biquad(grid1: Integer; x,y: Double; Var z,zx,zy: Double);
{   Interpolated Value and Normal for ZIK=f(xi,yk)
    E.g. grid1=10 divisions in each direction
    x = xmin..xmax mapped to -> -10..+10
    y = ymin..ymax mapped to -> -10..+10
    Export Double because of other procedures
    Optimized version
    With linear blending for Value and Normal}
Var     i,k,i1,i2,ia,ka                         : Integer;
        dx,dy,d2x,d2y,dxa,dya,a1,a2             : Single;
        px,qx,rx,py,qy,ry                       : Single;
        xa,ya,xb,yb,xc,yc                       : Single;
        z00,z10,z20,z01,z11,z21,z02,z12,z22     : Single;
        zp,zq,zr,za,zb,zc                       : Single;
Begin
{ Pass A }
x:=grid1*x; i:=Round(x); dx:=x-i; d2x:=2*dx; dxa:=dx; ia:=i;
y:=grid1*y; k:=Round(y); dy:=y-k; d2y:=2*dy; dya:=dy; ka:=k;
Dec(i); Dec(k); i1:=i+1; i2:=i+2;
{ Ignore Pointer ^, use ordinary memory ZIK(i,k) }
z00:=ZIK^[i,k];    z10:=ZIK^[i1,k];   z20:=ZIK^[i2,k]; Inc(k);
z01:=ZIK^[i,k];    z11:=ZIK^[i1,k];   z21:=ZIK^[i2,k]; Inc(k);
z02:=ZIK^[i,k];    z12:=ZIK^[i1,k];   z22:=ZIK^[i2,k];
{ A: 3 times in x-direction, row 0,1,2 for zp,zq,zr and dz/dx }
a1:=0.5*(z20-z00);
a2:=  a1+z00-z10;
zp:=z10+(a1+a2*dx)*dx;
px:=    a1+a2*d2x;
a1:=0.5*(z21-z01);
a2:=  a1+z01-z11;
zq:=z11+(a1+a2*dx)*dx;
qx:=    a1+a2*d2x;
a1:=0.5*(z22-z02);
a2:=  a1+z02-z12;
zr:=z12+(a1+a2*dx)*dx;
rx:=    a1+a2*d2x;
{ A: In y-direction for z }
a1:=0.5*(zr-zp);
a2:=  a1+zp-zq;
za:=zq+(a1+a2*dy)*dy;
{ A: In y-direction for dz/dx }
a1:=0.5*(rx-px);
a2:=  a1+px-qx;
xa:=qx+(a1+a2*dy)*dy;
```

4. Biquadratic Interpolation with Blending

4.3.2 Source code with blending

```
{ A: 3 times in y-direction for dz/dy }
a1:=0.5*(z02-z00);
a2:=  a1+z00-z01;
py:=a1+a2*d2y;
a1:=0.5*(z12-z10);
a2:=  a1+z10-z11;
qy:=a1+a2*d2y;
a1:=0.5*(z22-z20);
a2:=  a1+z20-z21;
ry:=a1+a2*d2y;
{ A: In x-direction for dz/dy }
a1:=0.5*(ry-py);
a2:=  a1+py-qy;
ya:=qy+(a1+a2*dx)*dx;
{ Pass B, Shift in x-direction }
If dxa>0 Then i:=ia+1 Else i:=ia-1;
dx:=x-i; dy:=dya; k:=ka; d2x:=2*dx; d2y:=2*dy;
Dec(i); Dec(k); i1:=i+1; i2:=i+2;
z00:=ZIK^[i,k];    z10:=ZIK^[i1,k];   z20:=ZIK^[i2,k]; Inc(k);
z01:=ZIK^[i,k];    z11:=ZIK^[i1,k];   z21:=ZIK^[i2,k]; Inc(k);
z02:=ZIK^[i,k];    z12:=ZIK^[i1,k];   z22:=ZIK^[i2,k];
{ B: 3 times in x-direction, row 0,1,2 for zp,zq,zr and dz/dx }
a1:=0.5*(z20-z00);
a2:=  a1+z00-z10;
zp:=z10+(a1+a2*dx)*dx;
px:=   a1+a2*d2x;
a1:=0.5*(z21-z01);
a2:=  a1+z01-z11;
zq:=z11+(a1+a2*dx)*dx;
qx:=   a1+a2*d2x;
a1:=0.5*(z22-z02);
a2:=  a1+z02-z12;
zr:=z12+(a1+a2*dx)*dx;
rx:=   a1+a2*d2x;
{ B: In y-direction for z }
a1:=0.5*(zr-zp);
a2:=  a1+zp-zq;
zb:=zq+(a1+a2*dy)*dy;
{ B: In y-direction for dz/dx }
a1:=0.5*(rx-px);
a2:=  a1+px-qx;
xb:=qx+(a1+a2*dy)*dy;
```

4. Biquadratic Interpolation with Blending

4.3.3 Source code with blending

```
{ B: 3 times in y-direction for zy }
a1:=0.5*(z02-z00);
a2:=  a1+z00-z01;
py:=a1+a2*d2y;
a1:=0.5*(z12-z10);
a2:=  a1+z10-z11;
qy:=a1+a2*d2y;
a1:=0.5*(z22-z20);
a2:=  a1+z20-z21;
ry:=a1+a2*d2y;
{ B: In x-direction for dz/dy }
a1:=0.5*(ry-py);
a2:=  a1+py-qy;
yb:=qy+(a1+a2*dx)*dx;
{ Pass C, Shift in y-direction }
If dya>0 Then k:=ka+1 Else k:=ka-1;
dx:=dxa; dy:=y-k; i:=ia; d2x:=2*dx; d2y:=2*dy;
Dec(i); Dec(k); i1:=i+1; i2:=i+2;
z00:=ZIK^[i,k];    z10:=ZIK^[i1,k];   z20:=ZIK^[i2,k]; Inc(k);
z01:=ZIK^[i,k];    z11:=ZIK^[i1,k];   z21:=ZIK^[i2,k]; Inc(k);
z02:=ZIK^[i,k];    z12:=ZIK^[i1,k];   z22:=ZIK^[i2,k];
{ C: 3 times in x-direction, row 0,1,2 for zp,zq,zr and dz/dx }
a1:=0.5*(z20-z00);
a2:=  a1+z00-z10;
zp:=z10+(a1+a2*dx)*dx;
px:=   a1+a2*d2x;
a1:=0.5*(z21-z01);
a2:=  a1+z01-z11;
zq:=z11+(a1+a2*dx)*dx;
qx:=   a1+a2*d2x;
a1:=0.5*(z22-z02);
a2:=  a1+z02-z12;
zr:=z12+(a1+a2*dx)*dx;
rx:=   a1+a2*d2x;
{ C: In y-direction for z }
a1:=0.5*(zr-zp);
a2:=  a1+zp-zq;
zc:=zq+(a1+a2*dy)*dy;
{ C: In y-direction for dz/dx }
a1:=0.5*(rx-px);
a2:=  a1+px-qx;
xc:=qx+(a1+a2*dy)*dy;
{ C: 3 times in y-direction for dz/dy }
a1:=0.5*(z02-z00);
a2:=  a1+z00-z01;
py:=a1+a2*d2y;
```

4. Biquadratic Interpolation with Blending

4.3.4 Source code with blending

```
a1:=0.5*(z12-z10);
a2:=  a1+z10-z11;
qy:=a1+a2*d2y;
a1:=0.5*(z22-z20);
a2:=  a1+z20-z21;
ry:=a1+a2*d2y;
{ C: In x-direction for dz/dy }
a1:=0.5*(ry-py);
a2:=  a1+py-qy;
yc:=qy+(a1+a2*dx)*dx;
{ Blend A,B,C }
dxa:=Abs(dxa); dya:=Abs(dya);
z :=za+dxa*(zb-za)+dya*(zc-za);
zx:=xa+dxa*(xb-xa)+dxa*(xc-xa);
zy:=ya+dxa*(yb-ya)+dya*(yc-ya);
zx:=zx*grid1;
zy:=zy*grid1;
End;
```

4. Biquadratic Interpolation with Blending

4.3.5 Source code with blending, numerical differentiation

```
Procedure Biquad1(grid1: Integer; x,y: Double; Var z: Double);
{   Interpolated Value for ZIK=f(xi,yk)
    E.g. grid1=10 divisions in each direction
    x = xmin..xmax mapped to -> -10..+10
    y = ymin..ymax mapped to -> -10..+10
    Export Double because of other procedures
    Optimized version
    With linear blending for Value }
Var     i,k,i1,i2,ia,ka    : Integer;
        dx,dy,d2x,d2y,dxa,dya,a1,a2   : Single;
        xa,ya,xb,yb,xc,yc  : Single;
        z00,z10,z20,z01,z11,z21,z02,z12,z22: Single;
        zp,zq,zr,za,zb,zc  : Single;
Begin
{ Pass A }
x:=grid1*x; i:=Round(x); dx:=x-i; d2x:=2*dx; dxa:=dx; ia:=i;
y:=grid1*y; k:=Round(y); dy:=y-k; d2y:=2*dy; dya:=dy; ka:=k;
Dec(i); Dec(k); i1:=i+1; i2:=i+2;
{ Ignore Pointer ^, use ordinary memory ZIK(i,k) }
z00:=ZIK^[i,k];    z10:=ZIK^[i1,k];   z20:=ZIK^[i2,k]; Inc(k);
z01:=ZIK^[i,k];    z11:=ZIK^[i1,k];   z21:=ZIK^[i2,k]; Inc(k);
z02:=ZIK^[i,k];    z12:=ZIK^[i1,k];   z22:=ZIK^[i2,k];
{ A: 3 times in x-direction, row 0,1,2 for zp,zq,zr and dz/dx }
a1:=0.5*(z20-z00);
a2:=  a1+z00-z10;
zp:=z10+(a1+a2*dx)*dx;
a1:=0.5*(z21-z01);
a2:=  a1+z01-z11;
zq:=z11+(a1+a2*dx)*dx;
a1:=0.5*(z22-z02);
a2:=  a1+z02-z12;
zr:=z12+(a1+a2*dx)*dx;
{ A: In y-direction for z }
a1:=0.5*(zr-zp);
a2:=  a1+zp-zq;
za:=zq+(a1+a2*dy)*dy;
{ Pass B, Shift in x-direction }
If dxa>0 Then i:=ia+1 Else i:=ia-1;
dx:=x-i; dy:=dya; k:=ka; d2x:=2*dx; d2y:=2*dy;
Dec(i); Dec(k); i1:=i+1; i2:=i+2;
z00:=ZIK^[i,k];    z10:=ZIK^[i1,k];   z20:=ZIK^[i2,k]; Inc(k);
z01:=ZIK^[i,k];    z11:=ZIK^[i1,k];   z21:=ZIK^[i2,k]; Inc(k);
z02:=ZIK^[i,k];    z12:=ZIK^[i1,k];   z22:=ZIK^[i2,k];
{ B: 3 times in x-direction, row 0,1,2 for zp,zq,zr and dz/dx }
a1:=0.5*(z20-z00);
a2:=  a1+z00-z10;
zp:=z10+(a1+a2*dx)*dx;
```

4. Biquadratic Interpolation with Blending

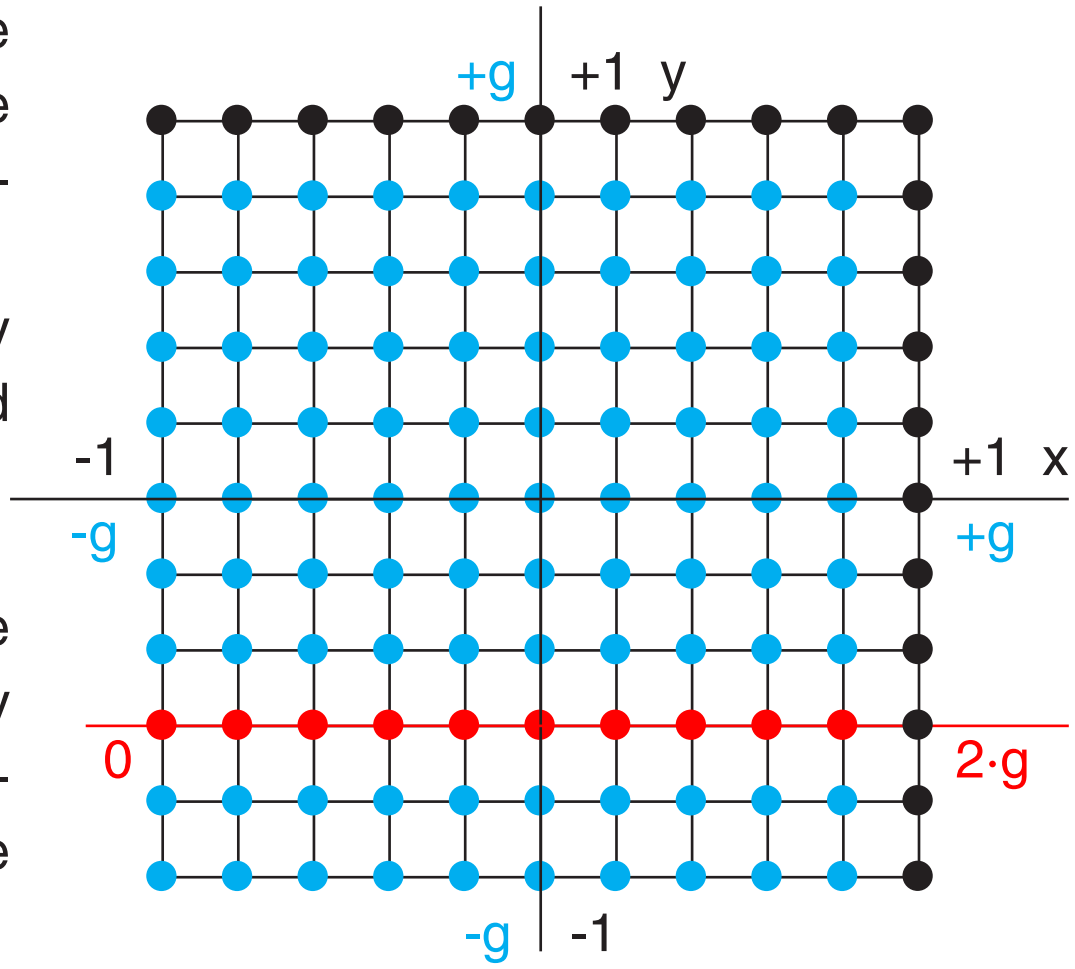
4.3.6 Source code with blending, numerical differentiation

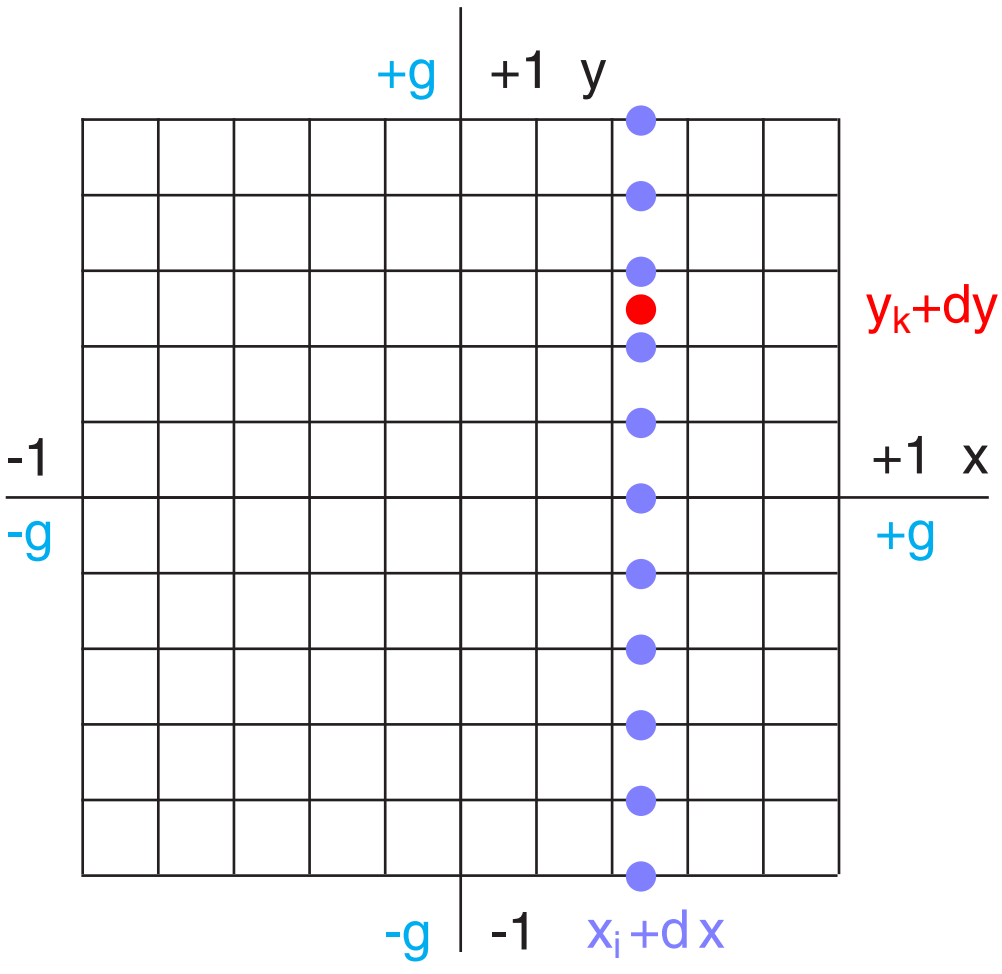
```
a1:=0.5*(z21-z01);
a2:=  a1+z01-z11;
zq:=z11+(a1+a2*dx)*dx;
a1:=0.5*(z22-z02);
a2:=  a1+z02-z12;
zr:=z12+(a1+a2*dx)*dx;
{ B: In y-direction for z }
a1:=0.5*(zr-zp);
a2:=  a1+zp-zq;
zb:=zq+(a1+a2*dy)*dy;
{ Pass C, Shift in y-direction }
If dya>0 Then k:=ka+1 Else k:=ka-1;
dx:=dxa; dy:=y-k; i:=ia; d2x:=2*dx; d2y:=2*dy;
Dec(i); Dec(k); i1:=i+1; i2:=i+2;
z00:=ZIK^[i,k];    z10:=ZIK^[i1,k];   z20:=ZIK^[i2,k]; Inc(k);
z01:=ZIK^[i,k];    z11:=ZIK^[i1,k];   z21:=ZIK^[i2,k]; Inc(k);
z02:=ZIK^[i,k];    z12:=ZIK^[i1,k];   z22:=ZIK^[i2,k];
{ C: 3 times in x-direction, row 0,1,2 for zp,zq,zr and dz/dx }
a1:=0.5*(z20-z00);
a2:=  a1+z00-z10;
zp:=z10+(a1+a2*dx)*dx;
a1:=0.5*(z21-z01);
a2:=  a1+z01-z11;
zq:=z11+(a1+a2*dx)*dx;
a1:=0.5*(z22-z02);
a2:=  a1+z02-z12;
zr:=z12+(a1+a2*dx)*dx;
{ C: In y-direction for z }
a1:=0.5*(zr-zp);
a2:=  a1+zp-zq;
zc:=zq+(a1+a2*dy)*dy;
{ Blend A,B,C }
z :=za+Abs(dxa)*(zb-za)+Abs(dya)*(zc-za);
End;

Procedure Biquad2(grid1: Integer; x,y: Double; Var zx,zy: Double);
{ Numerical Differentiation, assumed input x,y = -1...+1 }
Var ds,dd,z1,z2: Double;
Begin
ds:=0.001*grid1; dd:=0.5/ds;
Biquad1(grid1,x+ds,y,z2);
Biquad1(grid1,x-ds,y,z1);
zx:=dd*(z2-z1);
Biquad1(grid1,x,y+ds,z2);
Biquad1(grid1,x,y-ds,z1);
zy:=dd*(z2-z1);
End;
```

5. Global Spline Interpolation

5.1 Interpolation by global splines

A function z(x,y) is given in the range x=-1 to+1 and y=-1 to +1 by discrete values $z=z(x_i,y_k)$ in a regular non-integer grid.
The independent variables x and y are transformed into an integer grid x=-g to +g and y=-g to +g.
The illustration shows g=5 .
The discrete values $z_{ik}=z(x_i,y_k)$ are interpolated by splines. For arbitrary x,y the value z and the partial derivatives z_x,z_y are calculated, using the two-dimensional spline surface.
The preparation works as follows:
All splines in x-direction are determined. One row is shown red.
For all red knots in a row, which are numbered now from 0 to 2·g, the value z and the four local spline parameters a_3,a_2,a_1,a_0 are calculated and stored.
A local spline value is calculated by $z(x_i+dx)=a_3dx^3+a_2dx^2+a_1dx+a_0$.
The last knot has an assigned value but no spline parameters, because the precedent knot carries the parameters for the interval x_{g-1} to x_g .





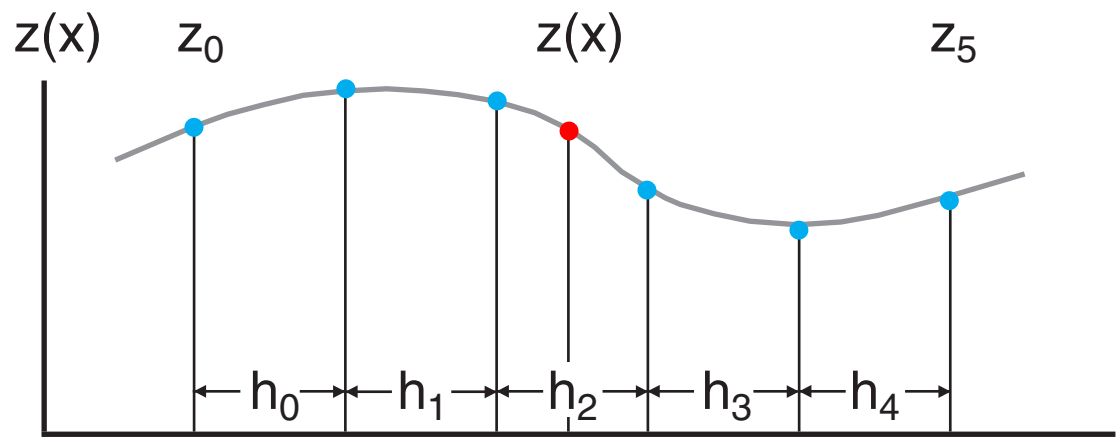
An actual interpolation calculates first all values $z(x_i+dx,y_k)$, marked by blue dots. These values are then interpolated by an explicit new spline for $z(x_i+dx,y_k+dy)$. The final position is shown red.
The kernel is a one-dimensional spline interpolator, as described on the next page.
An actual calculation is of order O(n) and therefore fast. Reference:
H.R.Schwarz, Numerische Mathematik, B.G.Teubner,Stuttgart 1993

5. Global Spline Interpolation

5.2.1 One-dimensional spline interpolation

A function z(x) is given at $i=0...p$ values x_i . An elastic beam, the spline, connects all points by cubic parabolas with second order continuity. Values, slopes and curvatures are equal at transition points.The curvatures at the ends have to be defined.

Each parabola is valid for an intervall x_i to x_{i+1}.
We have p sets of coefficients a_3, a_2, a_1, a_0 for $i=0...p-1$.
This results in $n=p-1$ linear equations.



$$dx_i = x - x_i$$

$$z(dx_i) = a_{3i}\,dx_i^{\,3} + a_{2i}\,dx_i^{\,2} + a_{1i}\,dx_i + a_{0i}$$

$$a_{3i} = \frac{1}{6h_i}(z''_{i+1} - z''_i)$$

$$a_{2i} = \frac{1}{2}\,z''_i$$

$$a_{1i} = \frac{1}{h_i}(z_{i+1} - z_i) - \frac{1}{6h_i}(z''_{i+1} + 2z''_i)$$

$$a_{0i} = z_i$$

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 \\ 0 & 0 & h_3 & 2(h_3+h_4) \end{bmatrix} \begin{bmatrix} z''_1 \\ z''_2 \\ z''_3 \\ z''_4 \end{bmatrix} = \begin{bmatrix} (6/h_1)(z_2 - z_1) - (6/h_0)(z_1 - z_0) - h_0 z''_0 \\ (6/h_2)(z_3 - z_2) - (6/h_1)(z_2 - z_1) \\ (6/h_3)(z_4 - z_3) - (6/h_2)(z_3 - z_2) \\ (6/h_4)(z_5 - z_4) - (6/h_3)(z_4 - z_3) - h_4 z''_5 \end{bmatrix}$$

We assume equal stepsizes and normalize for $h_i=1$. Furtheron, the curvatures at both ends are zero. This type is called a natural spline. The tridiagonal system of equations can be easily solved by O(n) recursion with (5n-4) significant operations.

$$a_{3i} = \frac{1}{6}(z''_{i+1} - z''_i)$$

$$a_{2i} = \frac{1}{2}\,z''_i$$

$$a_{1i} = z_{i+1} - z_i - \frac{1}{6}(z''_{i+1} + 2z''_i)$$

$$a_{0i} = z_i$$

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} z''_1 \\ z''_2 \\ z''_3 \\ z''_4 \end{bmatrix} = \begin{bmatrix} 6(z_2 - 2z_1 + z_0) \\ 6(z_3 - 2z_2 + z_1) \\ 6(z_4 - 2z_3 + z_2) \\ 6(z_5 - 2z_4 + z_3) \end{bmatrix}$$

5. Global Spline Interpolation

5.2.2 Solution of tridiagonal systems

This chapter follows again H.R.Schwarz, Numerische Mathematik.
Correction February 23 / 2007 (previous code was already correct).

A given tridiagonal system with n=4

$$\begin{bmatrix} a_1 & b_1 & & \\ c_1 & a_2 & b_2 & \\ & c_2 & a_3 & b_3 \\ & & c_3 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}$$

Triangle decomposition

$$\begin{bmatrix} a_1 & b_1 & & \\ c_1 & a_2 & b_2 & \\ & c_2 & a_3 & b_3 \\ & & c_3 & a_4 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ u_1 & 1 & & \\ & u_2 & 1 & \\ & & u_3 & 1 \end{bmatrix} \begin{bmatrix} m_1 & v_1 & & \\ & m_2 & v_2 & \\ & & m_3 & v_3 \\ & & & m_4 \end{bmatrix}$$

Decomposition $\mathbf{A} = \mathbf{U}\,\mathbf{V}$, with $v_i = b_i$

$m_1 = a_1$
For $i = 1$ to $n-1$ do
$u_i = c_i / m_i$
$m_{i+1} = a_{i+1} - u_i b_i$

$\mathbf{UVx} = \mathbf{r}$

Solution $\mathbf{Uy} = \mathbf{r}$

$y_1 = r_1$
For $i = 2$ to n do
$y_i = r_i - u_{i-1} y_{i-1}$

Solution $\mathbf{Vx} = \mathbf{y}$

$x_n = y_n / m_n$
For $i = n-1$ downto 1 do
$x_i = (y_i - b_i x_{i+1}) / m_i$

Simplified algorithm for equal stepsizes $h = 1$

$m_1 = 4$
For $i = 1$ to $n-1$ do
$m_{i+1} = 4 - 1/m_i$

$y_1 = 6(z_2 - 2z_1 + z_0)$
For $i = 2$ to n do
$y_i = 6(z_{i+1} - 2z_i + z_{i-1}) - y_{i-1}/m_{i-1}$

$x_n = y_n / m_n$
For $i = n-1$ downto 1 do
$x_i = (y_i - x_{i+1}) / m_i$

Final version, replace m_i by $1/m_i$

$m_1 = 0.25$
For $i = 1$ to $n-1$ do
$m_{i+1} = 1/(4 - m_i)$

$y_1 = 6(z_2 - 2z_1 + z_0)$
For $i = 2$ to n do
$y_i = 6(z_{i+1} - 2z_i + z_{i-1}) - y_{i-1} m_{i-1}$

$x_n = y_n m_n$
For $i = n-1$ downto 1 do
$x_i = (y_i - x_{i+1}) m_i$

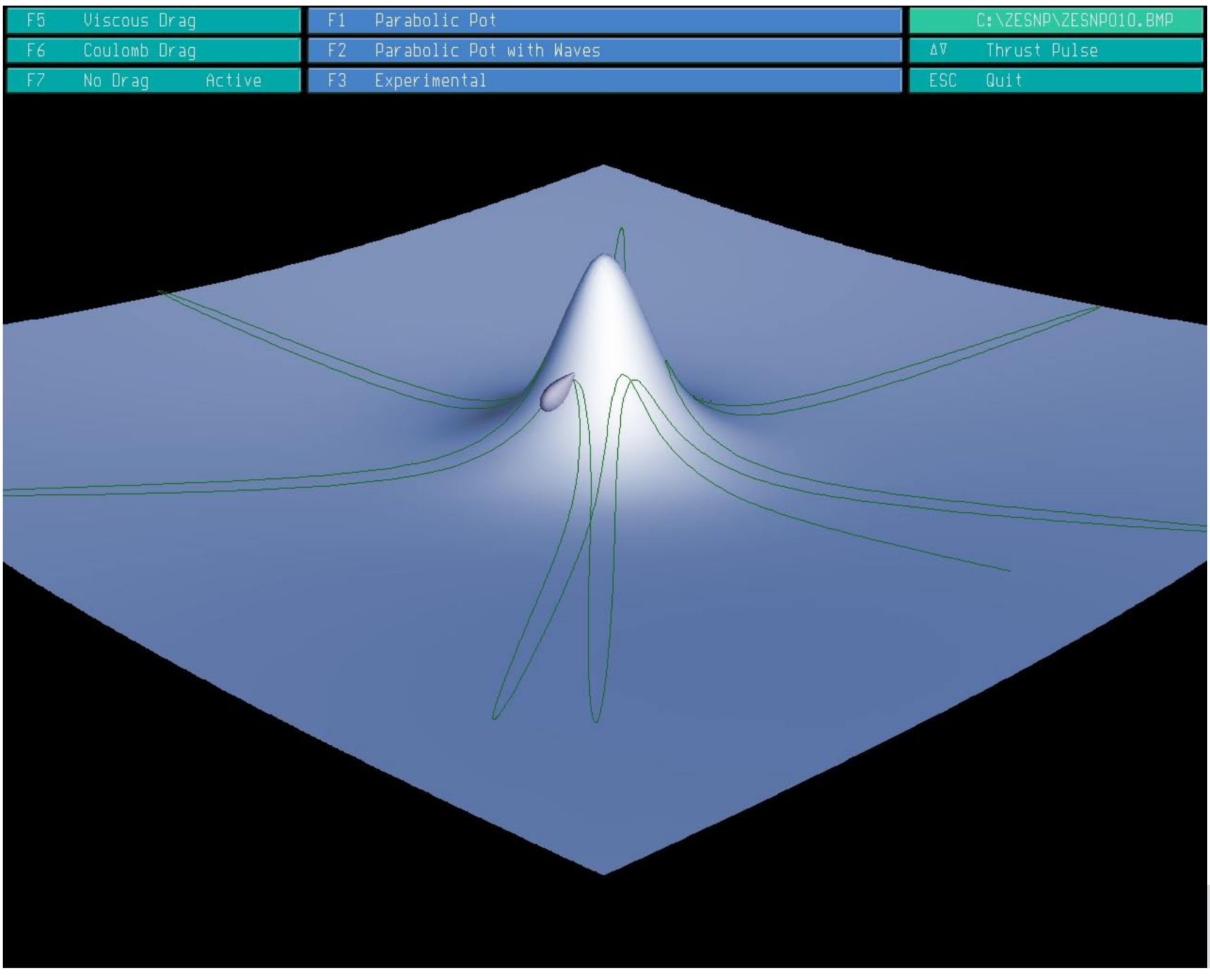
At the ends we use zero curvatures.

5. Global Spline Interpolation

5.3 Further remarks

FillGrid	Assign values $z(x_i, y_k)$ in grid
CofSpline	Calculate parameters for one x-spline for one y_k
AllSpline	Calculate parameters for all x-splines for all y_k
ValSpline	Calculate interpolated value $z(x)$ and derivative $z_x(x)$
ZxySpline	Calculate interpolated values $z(x,y)$ and partial derivatives $z_x(x,y)$, $z_y(x,y)$

As an alternative the storage of parameters for x-splines and y-splines was investigated. It turned out that a straightforward addition of the contributions of both at an actual non-integer position does not deliver correct results.
Speed (P2, 400MHz):
Raw grid g=10 for 21·21 knots. Interpolates for 201·201 values in fine grid in 4 seconds. Actual trajectory, interpolation, derivatives, integration and graphics for the running wood-louse by 85 frames per second.



5. Global Spline Interpolation

5.4.1 Source code

```
{ Interpolation variables}

Const   grid=10;
Type    GridOne = Record a3,a2,a1,a0,z: Double; End;
Type    GridRow = Array[-grid..grid]   Of  GridOne;
Type    ZikTyp  = Array[-grid..grid]   Of ^GridRow;
Var Zik     : ZikTyp;
Const   spl=1000;
Type    VecTyp  = Array[0..spl] Of Double;
Var Vec     : VecTyp;
Type    CoefOne = Record c3,c2,c1,c0: Double; End;
Type    CoefTyp = Array[0..spl] Of CoefOne;

{ Note: Read  Zik[k]^[i] as Zik[i,k]  }

Procedure FillGrid(grid,LType:Integer);
{   e.g. grid=10 for x,y=-1..+1 -> x,y=-10..+10 }
Var     i,k  : Integer;
        m,z1 : Double;
Begin
m:=1/grid;
For k:=-grid to +grid Do
For i:=-grid to +grid Do
Begin
 FValue(LType,i*m,k*m,z1);
 Zik[k]^[i].z:=z1;
End;
End;
```

5. Global Spline Interpolation

5.4.2 Source code

```
Procedure CofSpline(p: Integer; Vec: VecTyp; Var C: CoefTyp;
                    Var flag: Integer);
{ G.Hoffmann October 11, 2002 / February 23, 2007
  One-dimensional Spline-Interpolation, Coefficients
  Input : p     number of gridpoints 0...p
          Vec   Array [0..p  ] of Double, z values
  Output: C     Spline coefficients for each knot
                Array [0..p-1] of Record c3,c2,c1,c0 Double
          flag  0 No error; -1 Array size error
  n     : Number of equations
  mm    : 1/mi
  xx    : Unknown parameters
  yy    : Auxiliary variable                          }
Const spl=1000; { same as in main program }
Const d6: Double=1/6; m6: Double=+6;
Type  SplAux=Array[0..spl+1] Of Double;
Var   mm,yy,xx       : SplAux;
      i,j,n          : Integer;
      c3,c2,c1,c0,dx : Double;
      xxi,xxp,vei,vep: Double;
Label EX;
Begin
flag:=0;
If (p<4) Or (p>spl) Then Begin flag:=-1; Goto Ex; End;
n:=p-1;
mm[1]:=0.25;
For j:=1 to n-1 Do mm[j+1]:=1/(4.0-mm[j]);
yy[1]:=m6*(Vec[2]-2*Vec[1]+Vec[0]);
For j:=2 to n Do yy[j]:=m6*(Vec[j+1]-2*Vec[j]+Vec[j-1])-yy[j-1]*mm[j-1];
xx[n+1]:=0; xx[0]:=0;          xx[n]:= yy[n]*mm[n];
For j:=n-1 Downto 1 Do         xx[j]:=(yy[j]-xx[j+1])*mm[j];
xxi:=xx[0]; vei:=Vec[0];
For i:=0 to n Do
Begin
 With C[i] Do
 Begin
 xxp:=xx[i+1]; vep:=Vec[i+1];
 c3:= (xxp-xxi)*d6;
 c2:= 0.5*xxi;
 c1:= vep-vei-(xxp+2*xxi)*d6;
 c0:= vei;
 xxi:=xxp; vei:=vep;
 End;
End;
EX:
End;
```

5. Global Spline Interpolation

5.4.3 Source code

```
Procedure ValSpline(   p: Integer; Vec: VecTyp; x: Double;
                           Var z,zx: Double; Var flag: Integer);
{   G.Hoffmann October 26, 2002 / February 23, 2007
    One-dimensional Spline-Interpolation, Value and Derivative
    Input :     p       number of gridpoints 0...p
                Vec Array [0..p  ] of Double, z values
                x       Non-integer input
    Output:     z       Interpolated value z(x)
                flag  0 No error; -1 Array size error
    n       :   Number of equations
    mm      :   1/mi
    xx      :   Unknown parameters
    yy      :   Auxiliary variable }
Const spl=1000; { same as in main program }
Const   d6: Double=1/6; m6: Double=+6;
Type    SplAux=Array[0..spl+1] Of Double;
Var mm,yy,xx            : SplAux;
       i,j,n            : Integer;
       c3,c2,c1,c0,dx   : Double;
Label EX;
Begin
flag:=0;
If (p<4) Or (p>spl) Then Begin flag:=-1; Goto Ex; End;
n:=p-1;
mm[1]:=0.25;
For j:=1 to n-1 Do mm[j+1]:=1/(4.0-mm[j]);
yy[1]:=m6*(Vec[2]-2*Vec[1]+Vec[0]);
For j:=2 to n Do yy[j]:=m6*(Vec[j+1]-2*Vec[j]+Vec[j-1])-yy[j-1]*mm[j-1];
xx[n+1]:=0; xx[0]:=0;  xx[n]:= yy[n]*mm[n];
For j:=n-1 Downto 1 Do xx[j]:=(yy[j]-xx[j+1])*mm[j];
i := Round(x-0.5);
If i<0 Then i:=0 Else
If i>n Then i:=n;
c3:= (xx[i+1]-xx[i])*d6;
c2:= 0.5*xx[i];
c1:= Vec[i+1]-Vec[i]-(xx[i+1]+2*xx[i])*d6;
c0:= Vec[i];
dx:= x-i;
{ Direct Value and Derivative:
z := dx*(dx*(dx*c3+c2)+c1)+c0;
zx:= dx*(3*dx*c3+2*dx*c2)+c1;  }
..cont
```

5.4.4 Source code

```
{ Horner Value and Derivative: }
z :=c3;
zx:=c3;
z :=c2+dx*z ;
zx:=z +dx*zx;
z :=c1+dx*z ;
zx:=z +dx*zx;
z :=c0+dx*z ;
EX:
End;

Procedure AllSpline(grid: Integer; Var Zik: ZikTyp);
{   G.Hoffmann, October 05, 2002
    Fill grid knots with Spline coefficients c3,c2,c1,c0
    for all splines in x-direction    }
Var i,j,k,grid2,flag    : Integer;
       Vec              : VecTyp;
       Cof              : CoefTyp;
Begin
grid2:=2*grid;
{   Splines in x-direction }
For k:=0 to grid2 Do
Begin
 j:=k-grid;
 For i:=0 to grid2 Do Vec[i]:=Zik[j]^[i-grid].z;
 CofSpline(grid2,Vec,Cof,flag);
 For i:=0 to grid2-1 Do
  Begin
    With Zik[j]^[i-grid] Do
    With Cof[i] Do
    Begin a3:=c3; a2:=c2; a1:=c1; a0:=c0;
    End;
  End;
End;
```

5. Global Spline Interpolation

5.4.5 Source code

```
Procedure ZxySpline(  grid: Integer; Zik: ZikTyp; x,y: Double;
                          Var z,zx,zy: Double; mode: Integer);
{   G.Hoffmann October 5, 2002
    Spline interpolation
    Source  x,y     -1  ..  +1
    Mapped  to  -grid   .. grid
    mode    1: Value z only and zx=0, zy=0
            2:  Value and Derivatives z,zx,zy
    Result  z(x,y),zx=dz/dx,zy=dz/dy
    Using   Spline coefficients for knots in Zik }
Var     xs,ys,dx,dy,f,d    : Double;
        i,k,xi,grid2       : Integer;
        Vecf,Vecd          : VecTyp;
Begin
grid2:=2*grid;
xs:=grid*x;
ys:=grid*(y+1); { Shift for 0..grid2 }
xi:=Round(xs-0.5);
If xi<-gridThen xi:=-grid Else
If xi> grid-1 Then xi:= grid-1;
dx:=xs-xi;
Case Mode Of
1: Begin
{   y-Interpolation of Values only }
    For k:=0 to grid2 Do
    Begin
    With Zik[k-grid]^[xi] Do
    Begin
    { Direct }
    Vecf[k]:=dx*(dx*(dx*a3+a2)+a1)+a0;
    { Horner
    f:=a3;
    f:=a2+dx*f;
    f:=a1+dx*f;
    Vecf[k]:=a0+dx*f; }
    End;
    End;
    { y-Interpolation of Values z and zy }
    ValSpline(grid2,Vecf,ys,z,zy,flag);
    zx:=0;
    zy:=0;
End;
```

5. Global Spline Interpolation

5.4.6 Source code

```
2: Begin
{   y-Interpolation of Values and Derivatives }
    For k:=0 to grid2 Do
    Begin
    With Zik[k-grid]^[xi] Do
    Begin
    { Direct }
    Vecf[k]:=dx*(dx*(dx*a3+a2)+a1)+a0;
    Vecd[k]:=dx*(3*dx*a3+2*a2)+a1;
    { Horner
    f:=a3;
    d:=a3;
    f:=a2+dx*f;
    d:=f +dx*d;
    f:=a1+dx*f;
    d:=f +dx*d;
    f:=a0+dx*f;
    Vecd[k]:=d;
    Vecf[k]:=f;}
    End;
    End;
    { y-Interpolation of Values z and zy  }
    ValSpline(grid2,Vecf,ys,z,zy,flag);
    { y-Interpolation of Derivatives zx, dummy f }
    ValSpline(grid2,Vecd,ys,zx,f,flag);
    zx:=grid*zx;
    zy:=grid*zy;
    End;
End; { Case }
End;
```

This doc:

http://docs-hoffmann.de/masspoint09022002.pdf

Gernot Hoffmann
September 09 / 2002 – October 26 / 2015

Website
Load browser / Click here