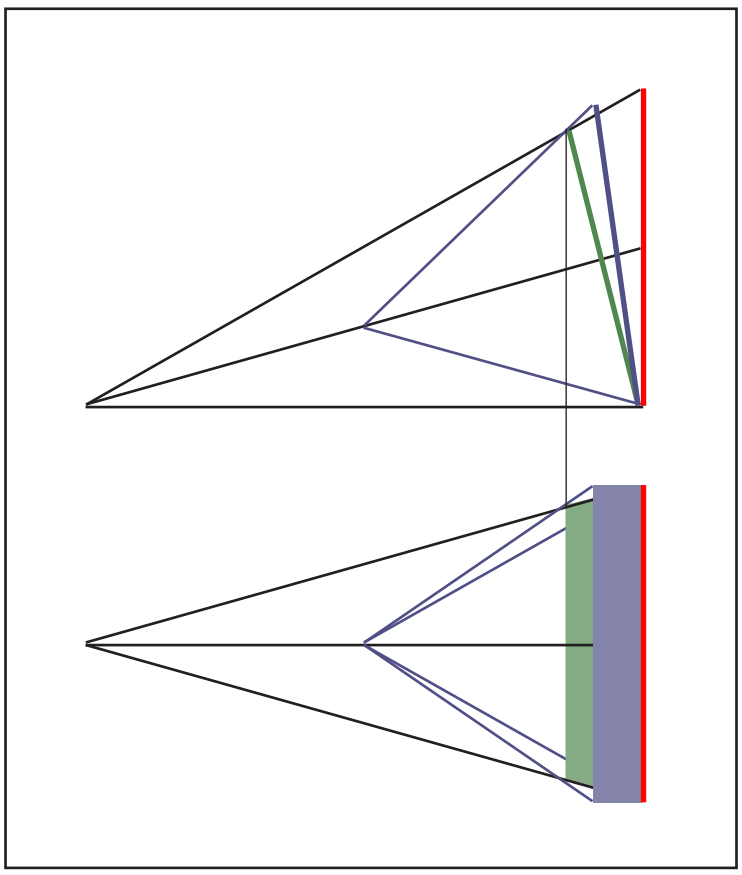
Gernot Hoffmann

# Rectification by Photogrammetry



## Contents



Settings for Acrobat

Edit / Preferences / General / Page Display (since version 6)
Custom Resolution 72 dpi / View by zoom 100% or 200%

Edit / Preferences / General / Color Management (full version only)
sRGB
Euroscale Coated or ISO Coated or SWOP
Gray Gamma 2.2

# 1. Introduction

Four mouselines define a quadrilateral, which should be rectified to a regular rectangle. Methods of photogrammetry are applied: projective mapping, which includes also translation, rotation, scaling and shear.

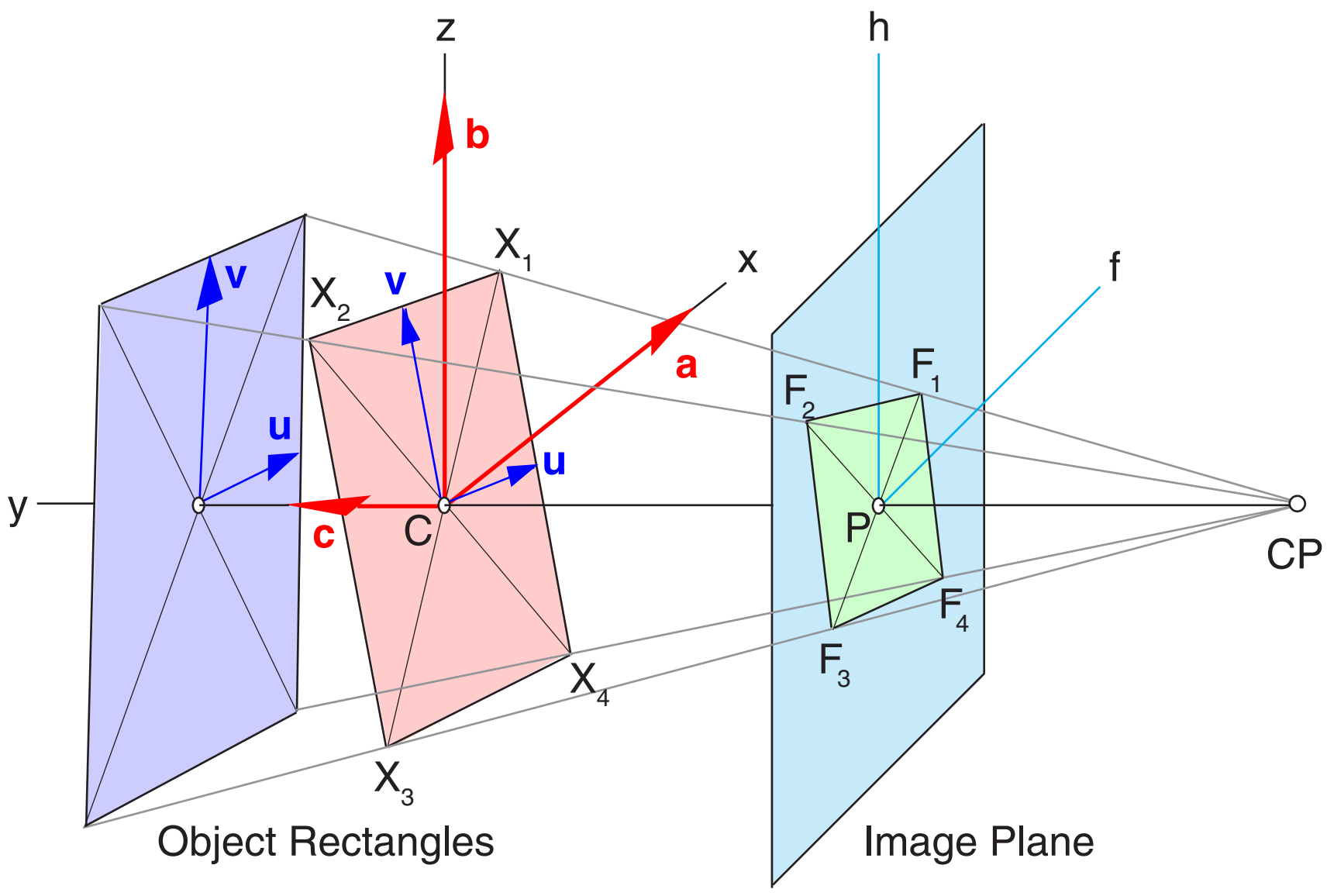
Now the façade is rectified. The aspect ratio is not accurate because no landmarks are available. Missing parts were reconstructed, e.g. the top of the roof and parts of the right house. The image was sharpened and the entrance made lighter. All applications by ZEBRA, a program by the author.
The photo shows the Old Hospital in Stonetown, Zanzibar, East Africa.

# 2. Reconstruction of the Aspect Ratio

Is it possible to calculate the aspect ratio of an arbitrary object rectangle by rectification of an image quadrilateral ?

It is assumed that the photo was taken by a standard camera: the image plane is orthogonal to the viewline. Both scalefactors for f and h are equal and lens distortions may be neglectable.
The view line shall hit the center C of the red rectangle which represents e.g. a house façade. The principal point P is in the center of the film negative, therefore well known if the image is not cropped.

Can we assign to the image $F_1 ... F_4$ another object rectangle (blue) which has a different aspect ratio ? The center of projection CP is allowed to move along the view line.
This looks unlikely and therefore we assume, that the aspect ratio is unique for the image. The next step is an attempt to calculate the aspect ratio.




How is this problem solved in ZEBRA applications, previous page ?
The width of the rectified rectangle is estimated by the average of the upper and lower width of the quadrilateral. The height by averaging the quadrilateral heights.
An aspect ratio correction may be applied, if the image does not look correct, but here it was not necessary.

# 3.1 Calculation of the Aspect Ratio

We assume a very general synthetical camera without lens distortions. The image plane can be tilted relative to the optical axis. It is allowed that the scales in both directions are different. The image can be sheared. Altogether, besides the perspective mapping, a general affine distortion is allowed.
The camera is located somewhere in the object space and is arbitrarily rotated. Object coordinates are $\mathbf{x}=(x,y,z)^T$. Image coordinates are $\mathbf{f} = (f,h)^T$.

The general perspective mapping is performed by these equations:

$$f = \frac{a_o + \mathbf{a}^T\mathbf{x}}{1 + \mathbf{c}^T\mathbf{x}}$$

$$h = \frac{b_o + \mathbf{b}^T\mathbf{x}}{1 + \mathbf{c}^T\mathbf{x}}$$

We have 11 unknowns $a_o$, $b_o$, $\mathbf{a} = (a_x, a_y, a_z)^T$, $\mathbf{b} = (b_x, b_y, b_z)^T$, $\mathbf{c} = (c_x, c_y, c_z)^T$

The vector **c** is generally orthogonal to the image plane. For an ideal camera without affine distortions, the lenghts of **a** and **b** are equal and **a** and **b** are orthogonal to each other. This can be proved mathematically and it explains why we need only 7 informations to recover the relevant information for a camera identification: 3 positions , 3 angles, 1 scale factor. The knowledge of the focal length is totally useless, by the way.

The special case (previous page) results in the simplification $a_o=0$, $b_o=0$ .

$$f = \frac{\mathbf{a}^T\mathbf{x}}{1 + \mathbf{c}^T\mathbf{x}}$$

$$h = \frac{\mathbf{b}^T\mathbf{x}}{1 + \mathbf{c}^T\mathbf{x}}$$

The plane $\mathbf{a}^T\mathbf{x}=0$ delivers f=0. The plane $\mathbf{b}^T\mathbf{x}=0$ delivers h=0. f and h are orthogonal, therefore, **a** and **b** are orthogonal. $1+\mathbf{c}^T\mathbf{x}=0$ is a plane parallel to the image plane through the center of projection. **c** is orthogonal to the image plane, in view line direction.

# 3.2 Calculation of the Aspect Ratio

Because we are not interested in the general task of photogrammetry (calculate object points by image points), we choose an arbitrary object coordinate system x, y, z as shown in the drawing.

The rectangle is described by two unknown base vectors **u** and **v**. Then we have four corners:

$$\mathbf{x}_1 = +\mathbf{u} + \mathbf{v}$$
$$\mathbf{x}_2 = -\mathbf{u} + \mathbf{v}$$
$$\mathbf{x}_3 = -\mathbf{u} - \mathbf{v}$$
$$\mathbf{x}_4 = +\mathbf{u} - \mathbf{v}$$

Especially we choose $\mathbf{a}=(1, 0, 0)^T$ and $\mathbf{b}=(0, 0, 1)^T$ because the scalefactor is already included in **u** and **v**.
Because of the orthogonality we have $\mathbf{c}=(0, c_y, 0)^T$ with unknown $c_y$.

Multiplying the equations for f and h by the denominator and rearranging, we get this set of nonlinear equations for the 7 unknowns $u_x, u_y, u_z, v_x, v_y, v_z, c_y$:

$$(1)\ fu_1 = u_x + v_x - f_1 ( 1 + c_y( u_y + v_y) ) = 0$$
$$(2)\ fu_2 = -u_x + v_x - f_2 ( 1 + c_y(-u_y + v_y) ) = 0$$
$$(3)\ fu_3 = -u_x - v_x - f_3 ( 1 + c_y(-u_y - v_y) ) = 0$$
$$(4)\ fu_4 = u_x - v_x - f_4 ( 1 + c_y( u_y - v_y) ) = 0$$

$$(5)\ fu_5 = u_z + v_z - h_1 ( 1 + c_y( u_y + v_y) ) = 0$$
$$(6)\ fu_6 = -u_z + v_z - h_2 ( 1 + c_y(-u_y + v_y) ) = 0$$
$$(7)\ fu_7 = -u_z - v_z - h_3 ( 1 + c_y(-u_y - v_y) ) = 0$$
$$(8)\ fu_8 = u_z - v_z - h_4 ( 1 + c_y( u_y - v_y) ) = 0$$

**u** and **v** are orthogonal:

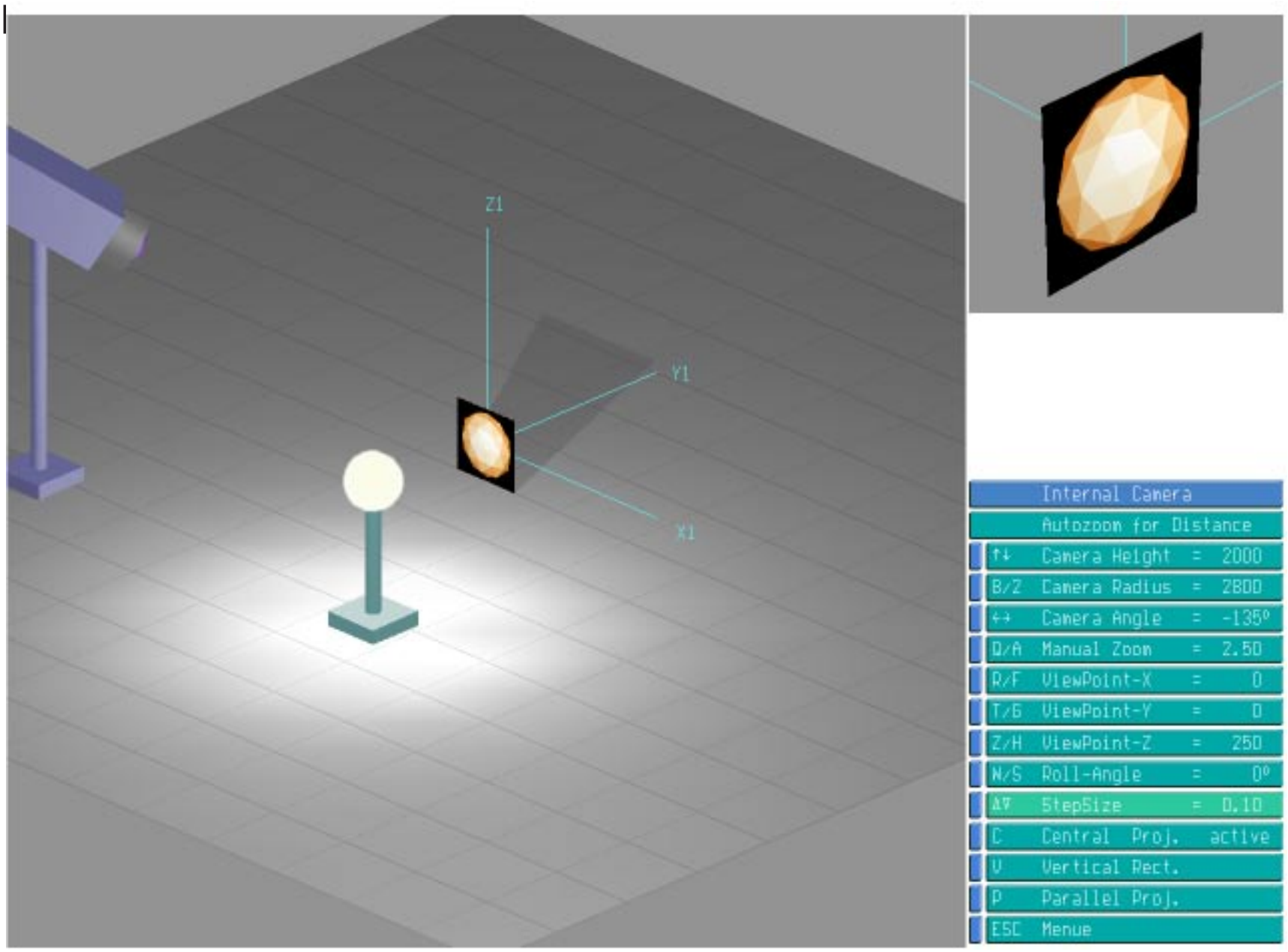
$$(9)\ fu_9 = u_x v_x + u_y v_y + u_z v_z = 0$$

Obviously we have 9 equations for 7 unknowns. Because we solve the system by the method of Steepest Descent (gradient downhill) we can take all equations. For Newton-Raphson-Jacobi we can omit (4) and (8), because the equations contain redundant information. But this was not tested.
After solving the equations numerically, the aspect ratio $\lambda$ is found by

$$(10)\ \lambda = \text{length}(\mathbf{v}) / \text{length}(\mathbf{u}) = \text{Sqrt} ( (v_x^2 + v_y^2 + v_z^2 ) / (u_x^2 + u_y^2 + u_z^2 )$$

# 4.1 Test Results for the Calculation

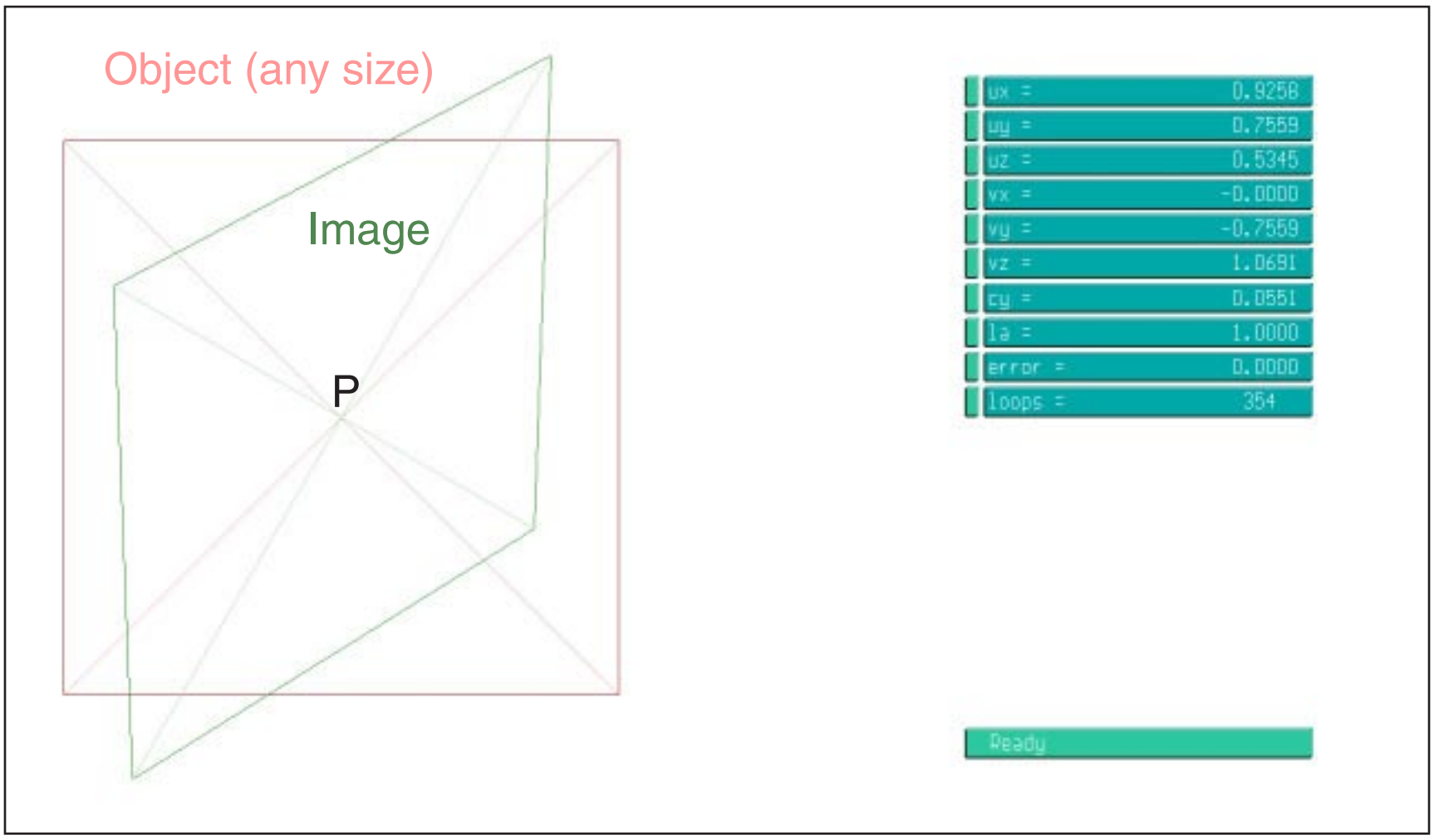
This image shows the test situation approximately:
Object plane 500 x 500 units
Camera Position x = -2000, y = -2000, z = +2000 units
The view point is at the origin of the cyan object coordinate system.
The small image top right shows the view through the camera.




```
Function ErrorFun (n: Integer; pn: Array Of Double):
         Double;
{    Sum of squared equation errors
     Global f1,f2,f3,f4; ux,uy,uz,vx,vy,vz,cy }
Var fu1,fu2,fu3,fu4,fu5,fu6,fu7,fu8,fu9,er: Double;
Begin
ux:=pn[1]; uy:=pn[2]; uz:=pn[3];
vx:=pn[4]; vy:=pn[5]; vz:=pn[6];
cy:=pn[7];
fu1  :=+ux+vx-f1.f*(1+cy*(+uy+vy));
fu2  :=-ux+vx-f2.f*(1+cy*(-uy+vy));
fu3  :=-ux-vx-f3.f*(1+cy*(-uy-vy));
fu4  :=+ux-vx-f4.f*(1+cy*(+uy-vy));
fu5  :=+uz+vz-f1.h*(1+cy*(+uy+vy));
fu6  :=-uz+vz-f2.h*(1+cy*(-uy+vy));
fu7  :=-uz-vz-f3.h*(1+cy*(-uy-vy));
fu8  :=+uz-vz-f4.h*(1+cy*(+uy-vy));
fu9  := ux*vx+uy*vy+uz*vz;
Errorfun:= Sqr(fu1)+Sqr(fu2)+Sqr(fu3)+Sqr(fu4)+
           Sqr(fu5)+Sqr(fu6)+Sqr(fu7)+Sqr(fu8)+Sqr(fu9);
End;
```

# 4.2 Test Results for the Calculation

This image shows the test result:
The aspect ratio is correctly identified after 354 iterations for ultimate accuracy. In fact much less iterations are necessary for practical accuracy. It takes anyway only about 0.1 seconds on a 400 MHz PC.



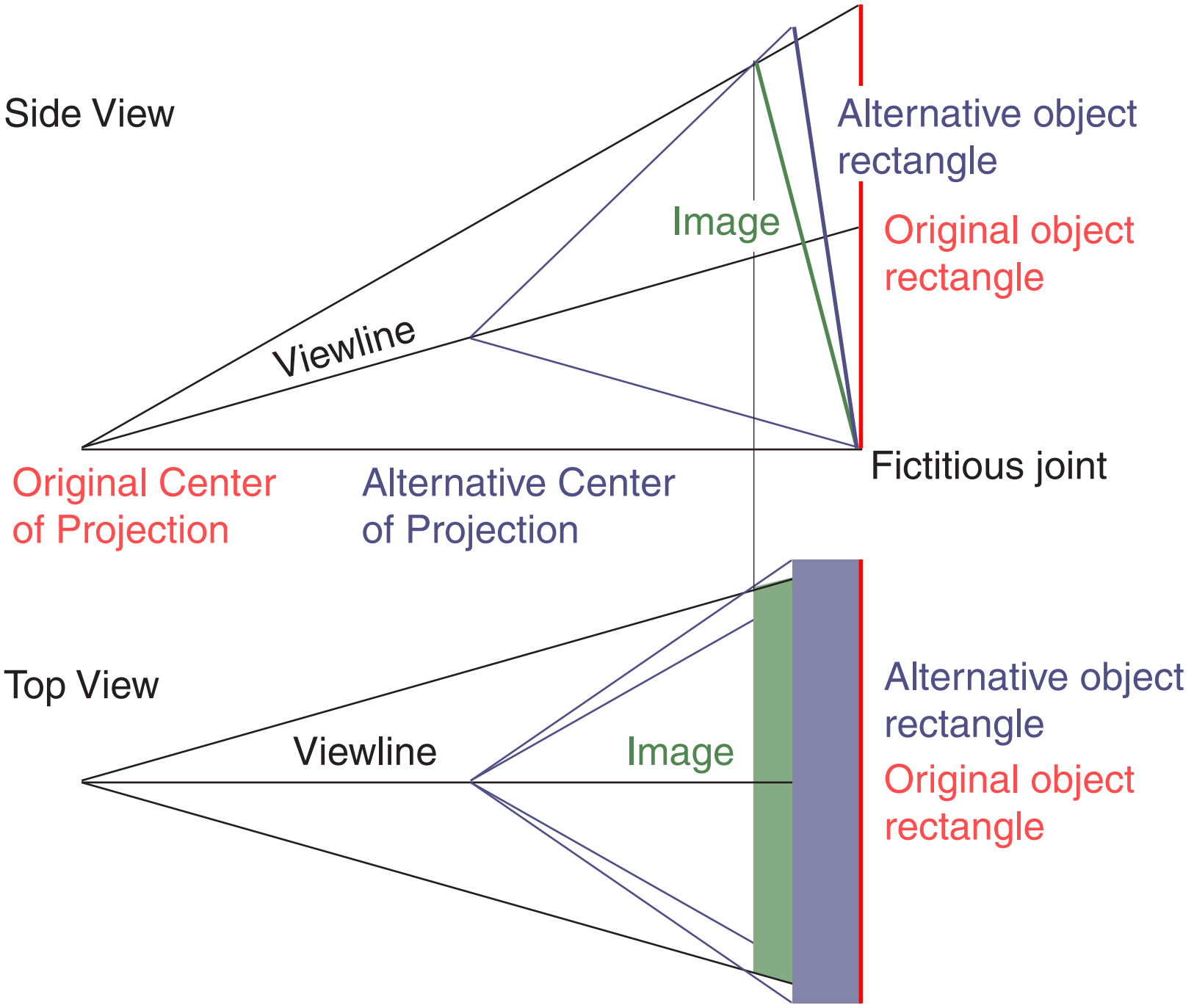

The calculation is based on floating point image data. If the source plane is not a square but a rectangle then the aspect ratio is calculated correctly as well. The accuracy for the aspect ratio is about 1% if we use pixel data. The object rectangle has the edge lenght 500 pixels. The accuracy deteriorates if the green image quadrilateral is considerably smaller.

Now we encounter a very odd problem: for camera positions where one vanishing point is at infinity (e.g. horizontal or vertical parallel edges), the calculation delivers wrong results.
But if we add the additional equation error $fu_{10} = 1 - \lambda$, using the true aspect ratio 1 and the actually calculated value $\lambda$, then the total squared equation error sum is zero - the equations are correct but the solution is not unique.

Final result: the aspect ration can be calculated, if the view line differs in both directions from the object rectangle normal. In other cases the result is ambiguous.
On the next page it is shown that projections with only one tilt angle can be generated by rectangles with different aspect ratios.

# 5. One tilt angle

Now we show, that for a single tilt angle several object rectangles can produce the same image.
The red square is the original rectangle, a line in both figures. The view line points from the center of projection to the center of the rectangle and the green image plane is orthogonal to the view line.
Now we move the center of projection along the view line. The blue rectangle is constructed by a rotation about the fictitious joint. The width (height in the lower figure) is the same as before. The rotation angle is found by the intersection of two rays through the upper image corners. Obviously the blue rectangle is not a square.




On the next pages are some procedures for perspective rectification, so far without further explanations, as used for the example on page 2.
The algorithm handles additionally translations, rotations, scaling and mirroring.
The aspect ratio is calculated by averaging image edge lengths.
Especially the counterclockwise sorting of the four corners is essential.

# 6.1 Procedures for Perspective Mapping

```
Here are some auxiliary procedures for Perspective Rectification,
so far without further explanations. No automatic Aspect Ratio !
```

```
Procedure Sort(Var x0,y0,x1,y1,x2,y2,x3,y3: Integer;
               Var N03: Array Of Integer);
  { Sort   x0...x3 counterclockwise
           x1    x0
           x2    x3   }
  Var      da,alf0,alf1,alf2,alf3: Single;
           xm,ym,dx,dy,flag,dn   : Integer;
  Begin
    xm:=(x0+x1+x2+x3) Div 4; { preliminary center, mean value }
    ym:=(y0+y1+y2+y3) Div 4;
    x0:=x0-xm; x1:=x1-xm; x2:=x2-xm; x3:=x3-xm;
    y0:=y0-ym; y1:=y1-ym; y2:=y2-ym; y3:=y3-ym;
    Atangens(-y0,x0,alf0,flag); If alf0<0 Then alf0:=alf0+2*pi;
    Atangens(-y1,x1,alf1,flag); If alf1<0 Then alf1:=alf1+2*pi;
    Atangens(-y2,x2,alf2,flag); If alf2<0 Then alf2:=alf2+2*pi;
    Atangens(-y3,x3,alf3,flag); If alf3<0 Then alf3:=alf3+2*pi;
    {   N03[0]: new number point 0
        N03[1]: new number point 1
        N03[2]: new number point 2
        N03[3]: new number point 3 }
        N03[0]:=0; N03[1]:=1; N03[2]:=2; N03[3]:=3;
    If alf0>alf1 Then
      Begin
        da  :=alf0; dx:=x0; dy:=y0;
        alf0:=alf1; x0:=x1; y0:=y1;
        alf1:=da;   x1:=dx; y1:=dy;
        dn:=N03[0]; N03[0]:=N03[1]; N03[1]:=dn;
      End;
    If alf0>alf2 Then
      Begin
        da  :=alf0; dx:=x0; dy:=y0;
        alf0:=alf2; x0:=x2; y0:=y2;
        alf2:=da;   x2:=dx; y2:=dy;
        dn:=N03[0]; N03[0]:=N03[2]; N03[2]:=dn;
      End;
    If alf0>alf3 Then
      Begin
        da  :=alf0; dx:=x0; dy:=y0;
        alf0:=alf3; x0:=x3; y0:=y3;
        alf3:=da;   x3:=dx; y3:=dy;
        dn:=N03[0]; N03[0]:=N03[3]; N03[3]:=dn;
      End;
    If alf1>alf2 Then
      Begin
        da  :=alf1; dx:=x1; dy:=y1;
        alf1:=alf2; x1:=x2; y1:=y2;
```

# 6.2 Procedures for Perspective Mapping

```
        alf2:=da;   x2:=dx; y2:=dy;
        dn:=N03[1]; N03[1]:=N03[2]; N03[2]:=dn;
  End;
If alf1>alf3 Then
  Begin
        da  :=alf1; dx:=x1; dy:=y1;
        alf1:=alf3; x1:=x3; y1:=y3;
        alf3:=da;   x3:=dx; y3:=dy;
        dn:=N03[1]; N03[1]:=N03[3]; N03[3]:=dn;
  End;
If alf2>alf3 Then
  Begin
        da  :=alf2; dx:=x2; dy:=y2;
        alf2:=alf3; x2:=x3; y2:=y3;
        alf3:=da;   x3:=dx; y3:=dy;
        dn:=N03[2]; N03[2]:=N03[3]; N03[3]:=dn;
    End;
   x0:=x0+xm; x1:=x1+xm; x2:=x2+xm; x3:=x3+xm;
   y0:=y0+ym; y1:=y1+ym; y2:=y2+ym; y3:=y3+ym;
End;
```

```
Procedure Collin (xof,yof: Integer; scx,scy,ang,mir: Single;
                  x0,y0,x1,y1,x2,y2,x3,y3: Integer; Var p8: BN);
{   Collinear Coefficients for Perspective Rectification
    Destination points defined in X1 = Screen
    Source points      defined in X8 = FMem   }
Var i,j,flag
: Integer;
    sx0,sy0,sx1,sy1,sx2,sy2,sx3,sy3          : Extended;
    rx0,ry0,rx1,ry1,rx2,ry2,rx3,ry3,lam,det  : Extended;
    x10,x31,x20,y10,y31,y20,xc,yc,a,b        : Extended;
    sx,sy,cp,sp,gx,gy,fx,fy                  : Single;
    N03                            Array[0..3] Of Integer;
    B8: BN; A88: ANN;   { Arrays for linear equations }
 Procedure Transf (Var x,y: Extended);
  Var xs,ys: Single;
  Begin
   xs:=sx*(x-gx-xof);
   ys:=sy*(y-gy-yof);
   x:= cp*xs+sp*ys+fx;
   y:=-sp*xs+cp*ys+fy;
  End;
Begin
 Sort(x0,y0,x1,y1,x2,y2,x3,y3,N03);{ Sort ccw                  }
 sx:=mir/scx; sy:=1/scy;          { mir=+1,-1                  }
 SicCoc(wrad*mir*ang,sp,cp);      { Fast sine, cosine          }
 gx:=xpx /2;                      { Center of Screen           }
 gy:=ypx /2;
 fx:=mxpix/2;                     { Center of Image in FMem    }
 fy:=mypix/2;
```

# 6.3 Procedures for Perspective Mapping

```
 sx1:=x1; sy1:=y1;
 sx2:=x2; sy2:=y2;
 sx3:=x3; sy3:=y3;
 a  :=(sx0-sx1+sx3-sx2)/4;    { Half edge length in X1           }
 b  :=(sy3-sy0+sy2-sy1)/4;    { mean values of two opposite edges}
{ Center of Squares in X1  }
 x10:=sx1-sx0; x31:=sx3-sx1; x20:=sx2-sx0;
 y10:=sy1-sy0; y31:=sy3-sy1; y20:=sy2-sy0;
 lam:=x10*y31-y10*x31;
 det:=x20*y31-y20*x31;
 lam:=lam/det;
 xc :=sx0+lam*x20;
 yc :=sy0+lam*y20;
{ Distorted Square in X8   }
 Transf(sx0,sy0);Transf(sx1,sy1);Transf(sx2,sy2);Transf(sx3,sy3);
{ Undistorted Square in X1 }
 rx0:=xc+a; ry0:=yc-b;
 rx1:=xc-a; ry1:=yc-b;
 rx2:=xc-a; ry2:=yc+b;
 rx3:=xc+a; ry3:=yc+b;
{ Undistorted Square in X8 }
 Transf(rx0,ry0);Transf(rx1,ry1);Transf(rx2,ry2);Transf(rx3,ry3);
{ Collinear Parameters     }
 For i:=1 to 8 Do For j:=1 to 8 Do A88[i,j]:=0.0;
 A88[1,1]:=rx0; A88[1,2]:=ry0; A88[1,3]:=1.0;
 A88[2,1]:=rx1; A88[2,2]:=ry1; A88[2,3]:=1.0;
 A88[3,1]:=rx2; A88[3,2]:=ry2; A88[3,3]:=1.0;
 A88[4,1]:=rx3; A88[4,2]:=ry3; A88[4,3]:=1.0;
 A88[5,4]:=rx0; A88[5,5]:=ry0; A88[5,6]:=1.0;
 A88[6,4]:=rx1; A88[6,5]:=ry1; A88[6,6]:=1.0;
 A88[7,4]:=rx2; A88[7,5]:=ry2; A88[7,6]:=1.0;
 A88[8,4]:=rx3; A88[8,5]:=ry3; A88[8,6]:=1.0;
 A88[1,7]:=-sx0*rx0; A88[1,8]:=-sx0*ry0;
 A88[2,7]:=-sx1*rx1; A88[2,8]:=-sx1*ry1;
 A88[3,7]:=-sx2*rx2; A88[3,8]:=-sx2*ry2;
 A88[4,7]:=-sx3*rx3; A88[4,8]:=-sx3*ry3;
 A88[5,7]:=-sy0*rx0; A88[5,8]:=-sy0*ry0;
 A88[6,7]:=-sy1*rx1; A88[6,8]:=-sy1*ry1;
 A88[7,7]:=-sy2*rx2; A88[7,8]:=-sy2*ry2;
 A88[8,7]:=-sy3*rx3; A88[8,8]:=-sy3*ry3;
 B8[1]:=sx0; B8[2]:=sx1; B8[3]:=sx2; B8[4]:=sx3;
 B8[5]:=sy0; B8[6]:=sy1; B8[7]:=sy2; B8[8]:=sy3;
{ Solve Equations A88*p8=B8 }
 HoGaussP(8,A88,p8,B8,det,flag);
End;
```

Gernot Hoffmann,
January 04 / 2001 — February 11 / 2013
Website
Load browser / Click here