Perspective Projection Transformation

Where does a point of a scene appear in an image?

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \xrightarrow{?} \begin{bmatrix} x_p'' \\ y_p'' \end{bmatrix}$$

Transformation in 3 steps:

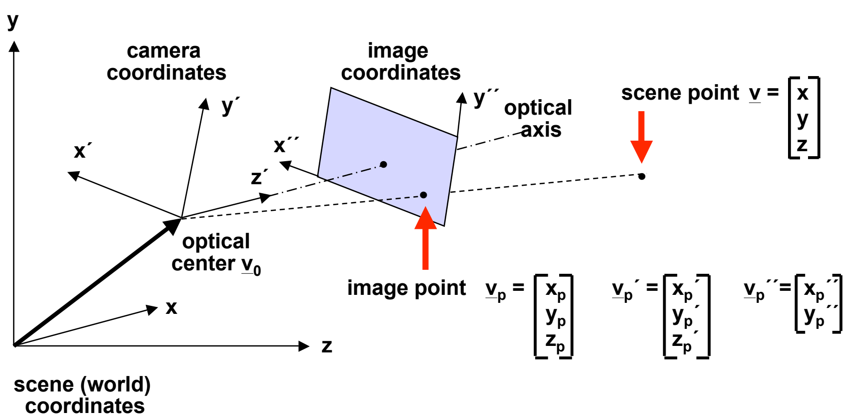
1. **scene coordinates => camera coordinates**
2. **projection of camera coordinates into image plane**
3. **camera coordinates => image coordinates**

Perspective projection equations are essential for Computer Graphics. For Image Understanding we will need the inverse: What are possible scene coordinates of a point visible in the image? This will follow later.

1

Perspective Projection in Independent Coordinate Systems

It is often useful to describe real-world points, camera geometry and image points in separate coordinate systems. The formal description of projection involves transformations between these coordinate systems.



2

3D Coordinate Transformation (1)

The new coordinate system is specified by a translation and rotation with respect to the old coordinate system:

$\underline{v}' = R(\underline{v} - \underline{v}_0)$ $\underline{v}_0$ is displacement vector; R is rotation matrix

Note that these matrices describe coo transforms for positive rotations of the coo system.

R may be decomposed into 3 rotations about the coordinate axes:
$R = R_x R_y R_z$

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}$$

If rotations are performed in the above order:
1) γ = rotation angle about z-axis
2) β = rotation angle about (new) y-axis
3) α = rotation angle about (new) x-axis

("tilt angle", "pan angle", and "nick angle" for the camera coordinate assignment shown before)

$$R_y = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3

3D Coordinate Transformation (2)

By multiplying the 3 matrices R_x, R_y and R_z, one gets

$$R = \begin{bmatrix} \cos\beta\cos\gamma & \cos\beta\sin\gamma & -\sin\beta \\ \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\cos\beta \\ \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\cos\beta \end{bmatrix}$$

For formula manipulations, one tries to avoid the trigonometric functions and takes

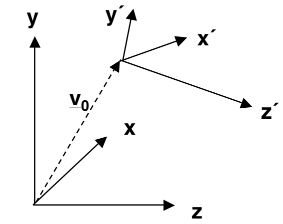
$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Note that the coefficients of R are constrained:
A rotation matrix is orthonormal:

$R R^T = I$ (unit matrix)

4

Example for Coordinate Transformation



camera coo system:

- displacement by $\underline{v}_0$
- rotation by pan angle $\beta = -30^0$
- rotation by nick angle $\alpha = 45^0$

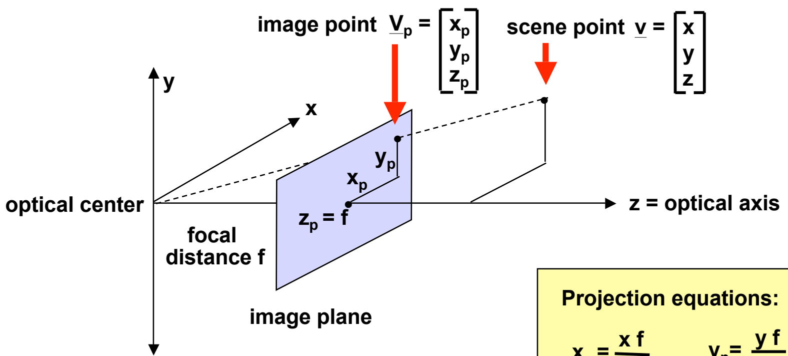
$\underline{v}' = R\,(\underline{v} - \underline{v}_0)$ with R = Rx Ry

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ 0 & -\frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \end{bmatrix} \qquad R_y = \begin{bmatrix} \frac{1}{2}\sqrt{3} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2}\sqrt{3} \end{bmatrix}$$

Perspective Projection Geometry

Projective geometry relates the coordinates of a point in a scene to the coordinates of its projection onto an image plane.

Perspective projection is an adequate model for most cameras.



Projection equations:

$x_p = \frac{x\,f}{z}$ $\qquad y_p = \frac{y\,f}{z}$

Perspective and Orthographic Projection

Within the camera coordinate system the perspective projection of a scene point onto the image plane is described by

$x_p' = \frac{x'f}{z'}$ $y_p' = \frac{y'f}{z'}$ $z_p' = f$ (f = focal distance)

- nonlinear transformation
- loss of information

If all objects are far away (large z'), f/z' is approximately constant

=> orthographic projection

$x_p' = s\,x'$ $y_p' = s\,y'$ (s = scaling factor)

Orthographic projection can be viewed as projection with parallel rays + scaling

7

From Camera Coordinates to Image Coordinates

Transform may be necessary because

- optical axis may not penetrate image plane at origin of desired coordinate system
- transition to discrete coordinates may require scaling

$x_p'' = (x_p' - x_{p0}')\,a$ a, b scaling parameters

$y_p'' = (y_p' - y_{p0}')\,b$ x_{p0}', y_{p0}' origin of image coordinate system

Example:

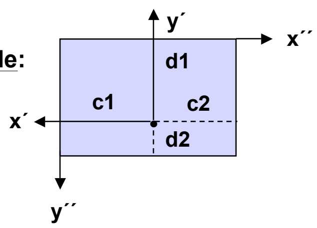


Image boundaries in camera coordinates:
$x'_{max} = c1$ $x'_{min} = c2$
$y'_{max} = d1$ $y'_{min} = d2$

Discrete image coordinates:
$x'' = 0 .. 511$ $y'' = 0 .. 575$

Transformation parameters:
$x_{p0}' = c1$ $y_{p0}' = d1$ $a = 512 / (c2 - c1)$ $b = 576 / (d2 - d1)$

8

Complete Perspective Projection Equation

We combine the 3 transformation steps:

1. scene coordinates => camera coordinates
2. projection of camera coordinates into image plane
3. camera coordinates => image coordinates

$x_p'' = \{ f/z' [\cos\beta \cos\gamma \ (x - x_0) + \cos\beta \sin\gamma \ (y - y_0) + \sin\beta \ (z - z_0)] - x_{p0} \} \ a$

$y_p'' = \{ f/z' [(-\sin\alpha \sin\beta \cos\gamma - \cos\alpha \sin\gamma) \ (x - x_0) +$
$(-\sin\alpha \sin\beta \sin\gamma + \cos\alpha \cos\gamma) \ (y - y_0) +$
$\sin\alpha \cos\beta \ (z - z_0)] - y_{p0} \} \ b$

with $z' = (-\cos\alpha \sin\beta \cos\gamma + \sin\alpha \sin\gamma)(x - x_0) +$
$(-\cos\alpha \sin\beta \sin\gamma - \sin\alpha \cos\gamma)(y - y_0) +$
$\cos\alpha \cos\beta (z - z_0)$

9

Homogeneous Coordinates (1)

4D notation for 3D coordinates which allows to express nonlinear 3D transformations as linear 4D transformations.

Normal: $\underline{v}' = R \ (\underline{v} - \underline{v}_0)$

Homogeneous coordinates: $\underline{\mathit{v}}' = \mathit{A} \ \underline{\mathit{v}}$

(note italics for homogeneous coordinates)

$$A = R\,T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transition to homogeneous coordinates:

$\underline{v}^T = [x \ y \ z]$ => $\underline{\mathit{v}}^T = [wx \ wy \ wz \ w]$ $\quad w \neq 0$ is arbitrary constant

Return to normal coordinates:

1. Divide components 1- 3 by 4th component
2. Omit 4th component

10

Homogeneous Coordinates (2)

Perspective projection in homogeneous coordinates:

$$\underline{v}_p{}' = P\,\underline{v}' \quad \text{with } P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \quad \text{and } \underline{v}' = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \quad \text{gives } \underline{v}_p{}' = \begin{bmatrix} wx \\ wy \\ wz \\ wz/f \end{bmatrix}$$

Returning to normal coordinates gives $\underline{v}_p{}' = \begin{bmatrix} xf/z \\ yf/z \\ f \end{bmatrix}$

compare with earlier slide

Transformation from camera into image coordinates:

$$\underline{v}_p{}'' = B\,\underline{v}_p{}' \quad \text{with } B = \begin{bmatrix} a & 0 & 0 & -x_0 a \\ 0 & b & 0 & -y_0 b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } \underline{v}_p{}' = \begin{bmatrix} wx_p \\ wy_p \\ 0 \\ w \end{bmatrix} \quad \text{gives } \underline{v}_p{}'' = \begin{bmatrix} wa(x_p-x_0) \\ wb(y_p-y_0) \\ 0 \\ w \end{bmatrix}$$

11

Homogeneous Coordinates (3)

Perspective projection can be completely described in terms of a linear transformation in homogeneous coordinates:

$$\underline{v}_p{}'' = B\,P\,R\,T\,\underline{v}$$

B P R T may be combined into a single 4 x 4 matrix *C* :

$$\underline{v}_p{}'' = C\,\underline{v}$$

In the literature the parameters of these equations may vary because of different choices of coordinate systems, different order of translation and rotation, different camera models, etc.

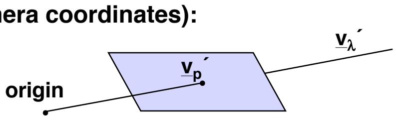
12

Inverse Perspective Equations

Which points in a scene correspond to a point in the image?

$$\begin{bmatrix} x_p'' \\ y_p'' \end{bmatrix} \xrightarrow{?} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Each image point defines a projection ray as the locus of possible scene points (for simplicity in camera coordinates):



$\underline{v}_p{}' \;\Rightarrow\; \underline{v}_\lambda{}' = \lambda\, \underline{v}_p{}'$

$\underline{v} = \underline{v}_0 + R^T \lambda\, \underline{v}_p{}'$

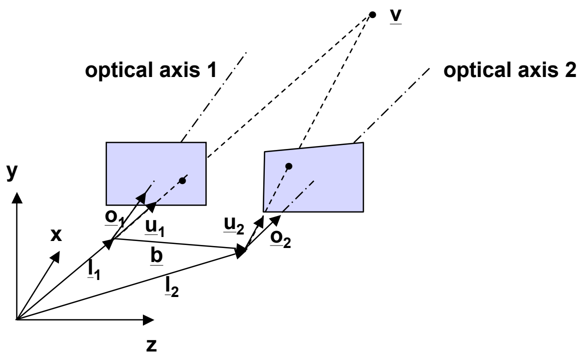
3 equations with the 4 unknowns x, y, z, λ and camera parameters R and $\underline{v}_0$

Applications of inverse perspective mapping for e.g.

- distance measurements
- binocular stereo
- camera calibration
- motion stereo

13

Binocular Stereo (1)



$\underline{l}_1$, $\underline{l}_2$	camera positions (optical center)
$\underline{b}$	stereo base
$\underline{o}_1$, $\underline{o}_2$	camera orientations (unit vectors)
f_1, f_2	focal distances
$\underline{v}$	scene point
$\underline{u}_1$, $\underline{u}_2$	projection rays of scene point (unit vectors)

14

Binocular Stereo (2)

Determine distance to $\underline{v}$ by measuring $\underline{u}_1$ and $\underline{u}_2$

Formally: $\alpha \underline{u}_1 = \underline{b} + \beta \underline{u}_2 \Rightarrow \underline{v} = \alpha \underline{u}_1 + \underline{l}_1$

α and β are overconstrained by the vector equation. In practice, measurements are inexact, no exact solution exists (rays do not intersect).

Better approach: Solve for the point of closest approximation of both rays:

$$\underline{v} = \frac{\alpha_0 \underline{u}_1 + (\underline{b} + \beta_0 \underline{u}_2)}{2} + \underline{l}_1 \quad \Rightarrow \quad \text{minimize} \quad \| \alpha \underline{u}_1 - (\underline{b} + \beta \underline{u}_2) \|^2$$

Solution:

$$\alpha_0 = \frac{\underline{u}_1^T \underline{b} - (\underline{u}_1^T \underline{u}_2)(\underline{u}_2^T \underline{b})}{1 - (\underline{u}_1^T \underline{u}_2)^2}$$

$$\beta_0 = \frac{(\underline{u}_1^T \underline{u}_2)(\underline{u}_1^T \underline{b}) - (\underline{u}_2^T \underline{b})}{1 - (\underline{u}_1^T \underline{u}_2)^2}$$

Distance in Digital Images

Intuitive concepts of continuous images do not always carry over to digital images.

Several methods for measuring distance between pixels:

Eucledian distance

$D_E((i,j),(h,k)) = \sqrt{(i-h)^2 + (j-k)^2}$ — costly computation of square root, can be avoided for distance comparisons

City-block distance

$D_4((i, j)(h, k)) = |i - h| + |j - k|$ — number of horizontal and vertical steps in a rectangular grid

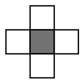
Chessboard distance

$D_8((i, j)(h, k)) = \max \{ |i - h|, |j - k| \}$ — number of steps in a rectangular grid if diagonal steps are allowed (number of moves of a king on a chessboard)

Connectivity in Digital Images

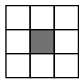
Connectivity is an important property of subsets of pixels. It is based on adjacency (or neighbourhood):

Pixels are 4-neighbours if their distance is $D_4 = 1$



all 4-neighbours of center pixel

Pixels are 8-neighbours if their distance is $D_8 = 1$



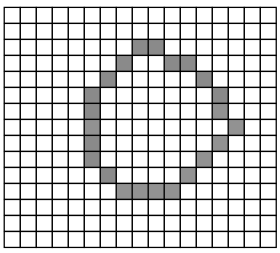
all 8-neighbours of center pixel

A path from pixel P to pixel Q is a sequence of pixels beginning at Q and ending at P, where consecutive pixels are neighbours.

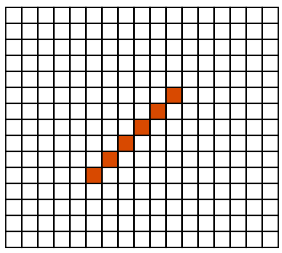
In a set of pixels, two pixels P and Q are connected, if there is a path between P and Q with pixels belonging to the set.

A region is a set of pixels where each pair of pixels is connected.

Closed Curve Paradoxon



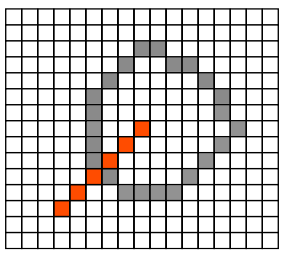
line 1



line 2

solid lines if 8-neighbourhood is used

a similar paradoxon arises if 4-neighbourhoods are used



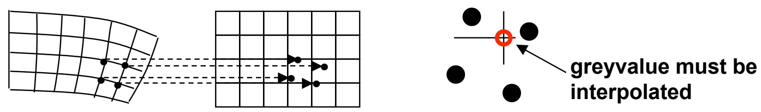
line 2 does not intersect line 1 although it crosses from the outside to the inside

Geometric Transformations

Various applications:

- change of view point
- elimination of geometric distortions from image capturing
- registration of corresponding images
- artificial distortions, Computer Graphics applications

Step 1: Determine mapping $\underline{T}$(x, y) from old to new coordinate system
Step 2: Compute new coordinates (x´, y´) for (x, y)
Step 3: Interpolate greyvalues at grid positions from greyvalues at transformed positions



19

Polynomial Coordinate Transformations

General format of transformation:

$$x' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} a_{rk} x^r y^k$$

$$y' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} b_{rk} x^r y^k$$

- Assume polynomial mapping between (x, y) and (x´, y´) of degree m
- Determine corresponding points
- a) Solve linear equations for a_{rk}, b_{rk} (r, k = 1 ... m)
  b) Minimize mean square error (MSE) for point correspondences

Approximation by biquadratic transformation:

$x' = a_{00} + a_{10}x + a_{01}y + a_{11}xy + a_{20}x^2 + a_{02}y^2$
$y' = b_{00} + b_{10}x + b_{01}y + b_{11}xy + b_{20}x^2 + b_{02}y^2$

at least 6 corresponding pairs needed

Approximation by affine transformation:

$x' = a_{00} + a_{10}x + a_{01}y$
$y' = b_{00} + b_{10}x + b_{01}y$

at least 3 corresponding pairs needed

20

Translation, Rotation, Scaling, Skewing

Translation by vector $\underline{t}$:

$\underline{v}' = \underline{v} + \underline{t}$ with $\underline{v}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ $\underline{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ $\underline{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

Rotation of image coordinates by angle α:

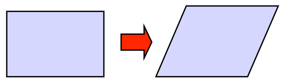
$\underline{v}' = R\,\underline{v}$ with $R = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$

Scaling by factor a in x-direction and factor b in y-direction:

$\underline{v}' = S\,\underline{v}$ with $S = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$

Skewing by angle β:

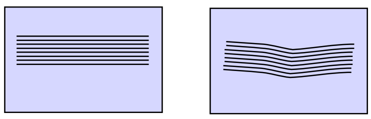
$\underline{v}' = W\,\underline{v}$ with $W = \begin{bmatrix} 1 & \tan\beta \\ 0 & 1 \end{bmatrix}$



Example of Geometry Correction by Scaling

Distortions of electron-tube cameras may be

1 - 2 % => more than 5 lines for TV images



ideal image actual image

Correction procedure may be based on

- fiducial marks engraved into optical system
- a test image with regularly spaced marks

Ideal mark positions:

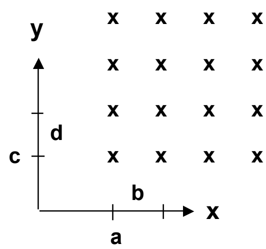
$x_{mn} = a + mb, \quad y_{mn} = c + nd$

$m = 0 \ldots M\text{-}1$

$n = 0 \ldots N\text{-}1$

Actual mark positions:

x'_{mn}, y'_{mn}



Determine a, b, c, d such that MSE (mean square error) of deviations is minimized

Minimizing the MSE

Minimize
$$E = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (x_{mn} - x'_{mn})^2 + (y_{mn} - y'_{mn})^2$$
$$= \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (a + mb - x'_{mn})^2 + (c + nd - y'_{mn})^2$$

From dE/da = dE/db = dE/dc = dE/dd = 0 we get:

$$a = \frac{2}{MN(M+1)} \sum_m \sum_n (2M - 1 - 3m)\ x'_{mn}$$

$$b = \frac{6}{MN(M^2 - 1)} \sum_m \sum_n (2m - M + 1) x'_{mn}$$

$$c = \frac{2}{MN(N+1)} \sum_m \sum_n (2N - 1 - 3n)\ y'_{mn}$$

$$d = \frac{6}{MN(N^2 - 1)} \sum_m \sum_n (2n - N + 1) y'_{mn}$$

Special case M=N=2:

$a = 1/2\ (x'_{00} + x'_{01})$

$b = 1/2\ (x'_{10} - x'_{00} + x'_{11} - x'_{01})$

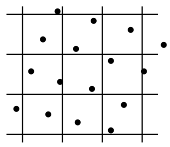
$c = 1/2\ (y'_{00} + y'_{01})$

$d = 1/2\ (y'_{01} - y'_{00} + y'_{11} - y'_{10})$

23

Principle of Greyvalue Interpolation

Greyvalue interpolation = computation of unknown greyvalues at locations (u´v´) from known greyvalues at locations (x´y´)



Two ways of viewing interpolation in the context of geometric transformations:

A Greyvalues at grid locations (x y) in old image are placed at corresponding locations (x´y´) in new image: g(x´y´) = g(T(x y))

=> interpolation in new image

B Grid locations (u´v´) in new image are transformed into corresponding locations (u v) in old image: g(u v) = g(T^{-1}(u´v´))

=> interpolation in old image

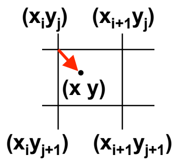
We will take view B:

Compute greyvalues between grid from greyvalues at grid locations.

24

Nearest Neighbour Greyvalue Interpolation

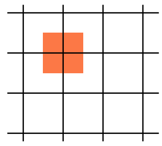
Assign to (x y) greyvalue of nearest grid location



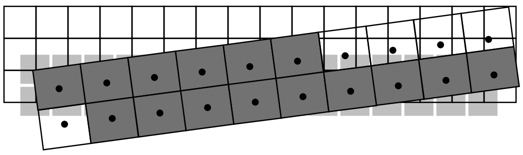
$(x_i y_j)$ $(x_{i+1} y_j)$ $(x_i y_{j+1})$ $(x_{i+1} y_{j+1})$ grid locations

(x y) location between grid with $x_i \le x \le x_{i+1}$, $y_j \le y \le y_{j+1}$

Each grid location represents the greyvalues in a rectangle centered around this location:



Straight lines or edges may appear step-like after this transformation:

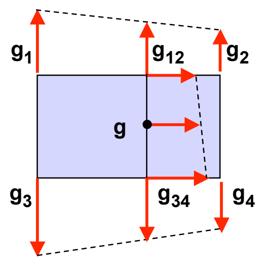


Bilinear Greyvalue Interpolation

The greyvalue at location (x y) between 4 grid points $(x_i y_j)$ $(x_{i+1} y_j)$ $(x_i y_{j+1})$ $(x_{i+1} y_{j+1})$ is computed by linear interpolation in both directions:

$$g(x,y) = \frac{1}{(x_{i+1} - x_i)(y_{j+1} - y_i)} \{ (x_{i+1} - x)(y_{j+1} - y) g(x_i y_j) + (x - x_i)(y_{j+1} - y) g(x_{i+1} y_j) + (x_{i+1} - x)(y - y_j) g(x_i y_{j+1}) + (x - x_i)(y - y_j) g(x_{i+1} y_{j+1}) \}$$

Simple idea behind long formula:
1. Compute g_{12} = linear interpolation of g_1 and g_2
2. Compute g_{34} = linear interpolation of g_3 and g_4
3. Compute g = linear interpolation of g_{12} and g_{34}

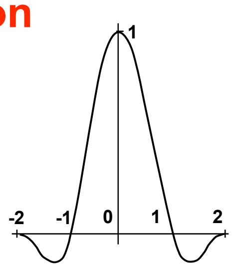


The step-like boundary effect is reduced.
But bilear interpolation may blur sharp edges.

Bicubic Interpolation

Each greyvalue at a grid point is taken to represent the center value of a local bicubic interpolation surface with cross section h_3.

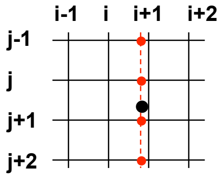
$$h_3 = \begin{cases} 1 - 2|x|^2 + |x|^3 & \text{for } 0 < |x| < 1 \\ 4 - 8|x| + 5|x|^2 - |x|^3 & \text{for } 1 < |x| < 2 \\ 0 & \text{otherwise} \end{cases}$$



cross section of interpolation kernel

The greyvalue at an arbitrary point [u, v] (black dot in figure) can be computed by

- 4 horizontal interpolations to obtain greyvalues at points [u, j-1] ... [u, j+2] (red dots), followed by
- 1 vertical interpolation (between red dots) to obtain greyvalue at [u, v].



Note:
For an image with constant geyvalues g_0 the interpolated greyvalues at all points between the grid lines are also g_0.

27