Image warping – introduction

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz
http://cgg.mff.cuni.cz/~pepca/

Warping .. image deformation

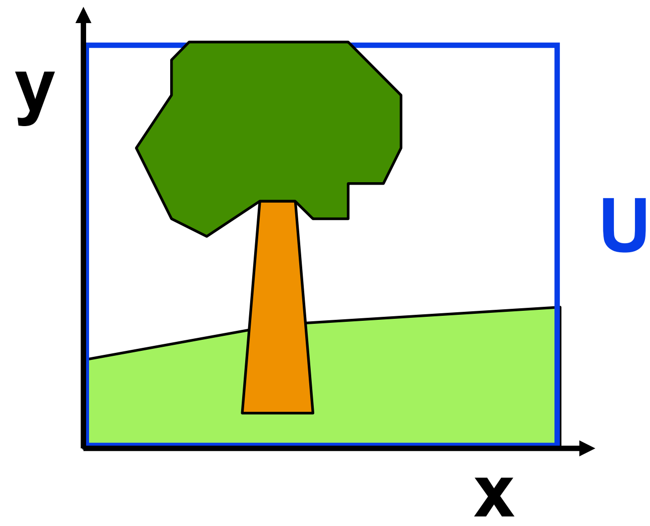
- **texture mapping** in 3D rendering (after rasterization)
  – perspective distortion, mapping textures to arbitrary shapes
- correction of **geometric distortion** (digital image acquirement)
  – satellite and aerial photography
  – scanning of deformed documents
- **special effects** in TV, film and advertisement

Mathematical model

„image function”

$$f: \quad U \subset R^2 \rightarrow R^n$$

$$f: \quad [x, y] \rightarrow [a_1, a_2, \ldots a_n]$$



position on the plane

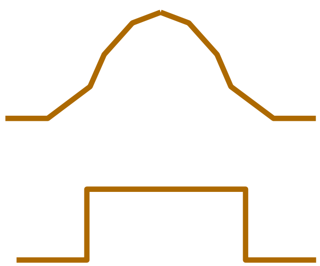
image attributes (<u>color</u>, transparency)

Spatial discretization

Digitized raster image:

$$\mathbf{I}: \quad \langle 0..m-1 \rangle \times \langle 0..n-1 \rangle \rightarrow \mathbf{R}^n$$

Digitization using filter **d**:



$$I_f(i,j) = \iint_{R^2} f(x,y) \cdot d(x-i, y-j) \, dx \, dy$$

d .. device characteristics
(optical system, CCD element)

Digital image reproduction

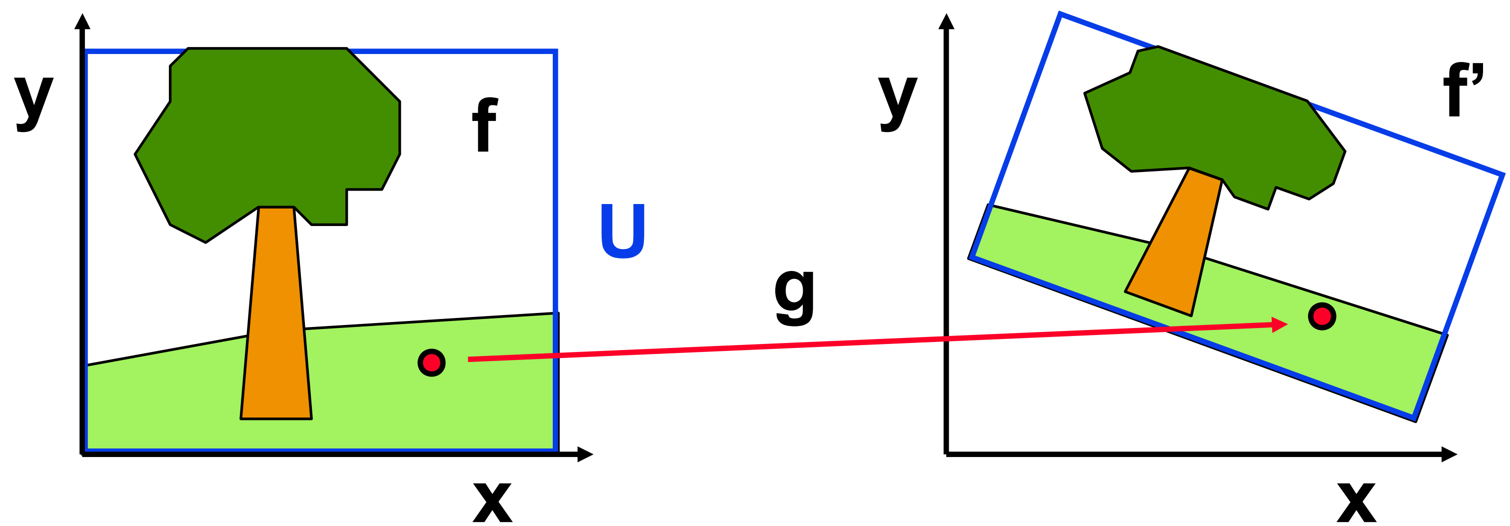
Reconstruction of a discrete image:

$$f^r(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I_f(i,j) \cdot r(i-x, j-y)$$

r .. output device characteristics (impulse response)

.. we need $\mathbf{f^r}$ to be similar to **f** (in frequency range defined by the Nyquist law)

Geometric transform



$$g:\quad R^2 \to R^2$$

$$f(x,y) = f'(g(x,y))$$

$$f(g^{-1}(u,v)) = f'(u,v)$$

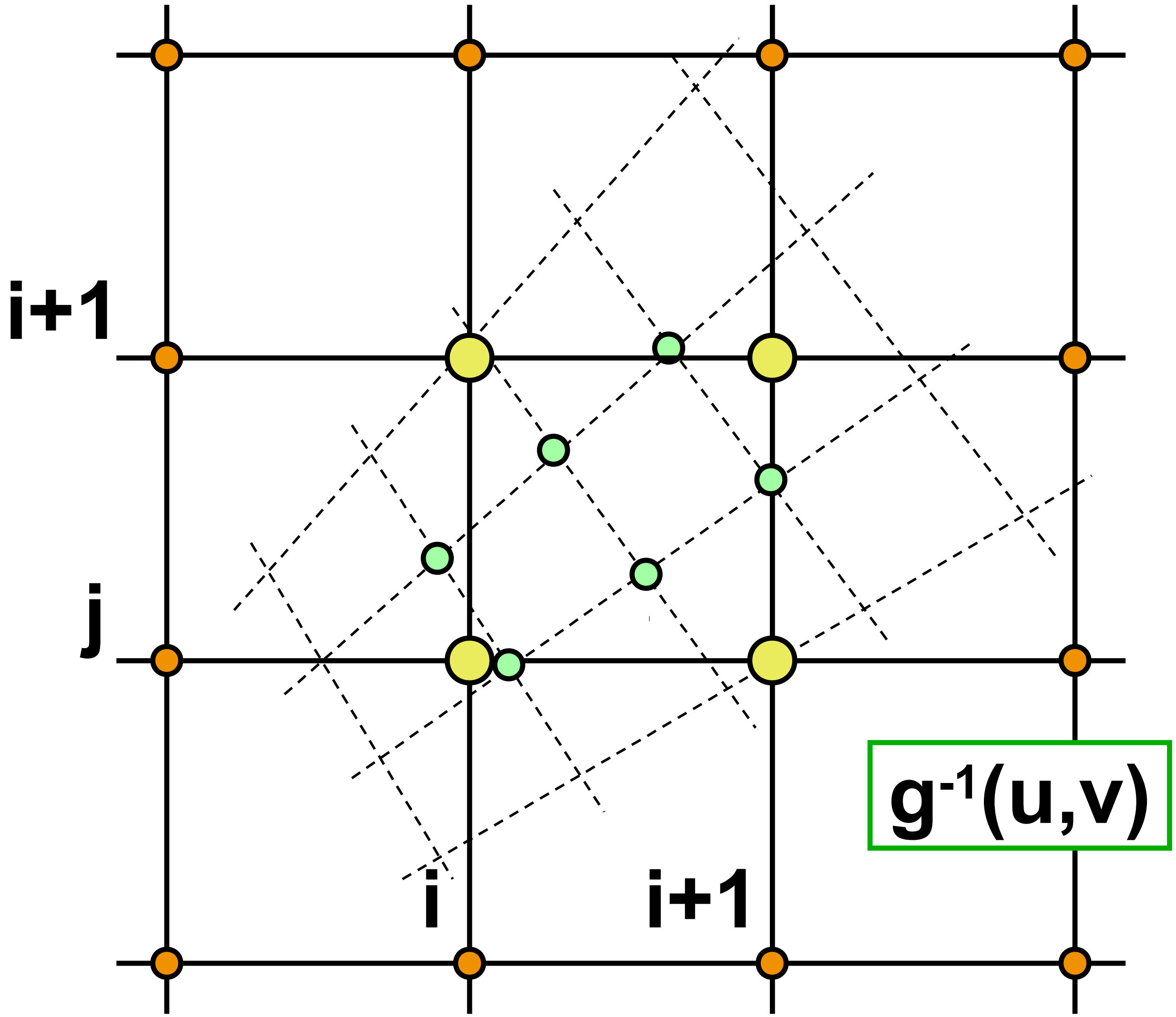
But what about raster images?

$$G(I_f) \cong I_{f'}$$

Transform with interpolation

- digitization = **sampling**
  - digitizing filter = Dirac delta
- attributes (color) of transformed pixel computed by **approximation** or **interpolation**
  - inverse transform function $\mathbf{g}^{-1}$ is needed
- **„rounding“**, **polynomial interpolation**
  - bilinear to bicubic interpolation/approximation is sufficient

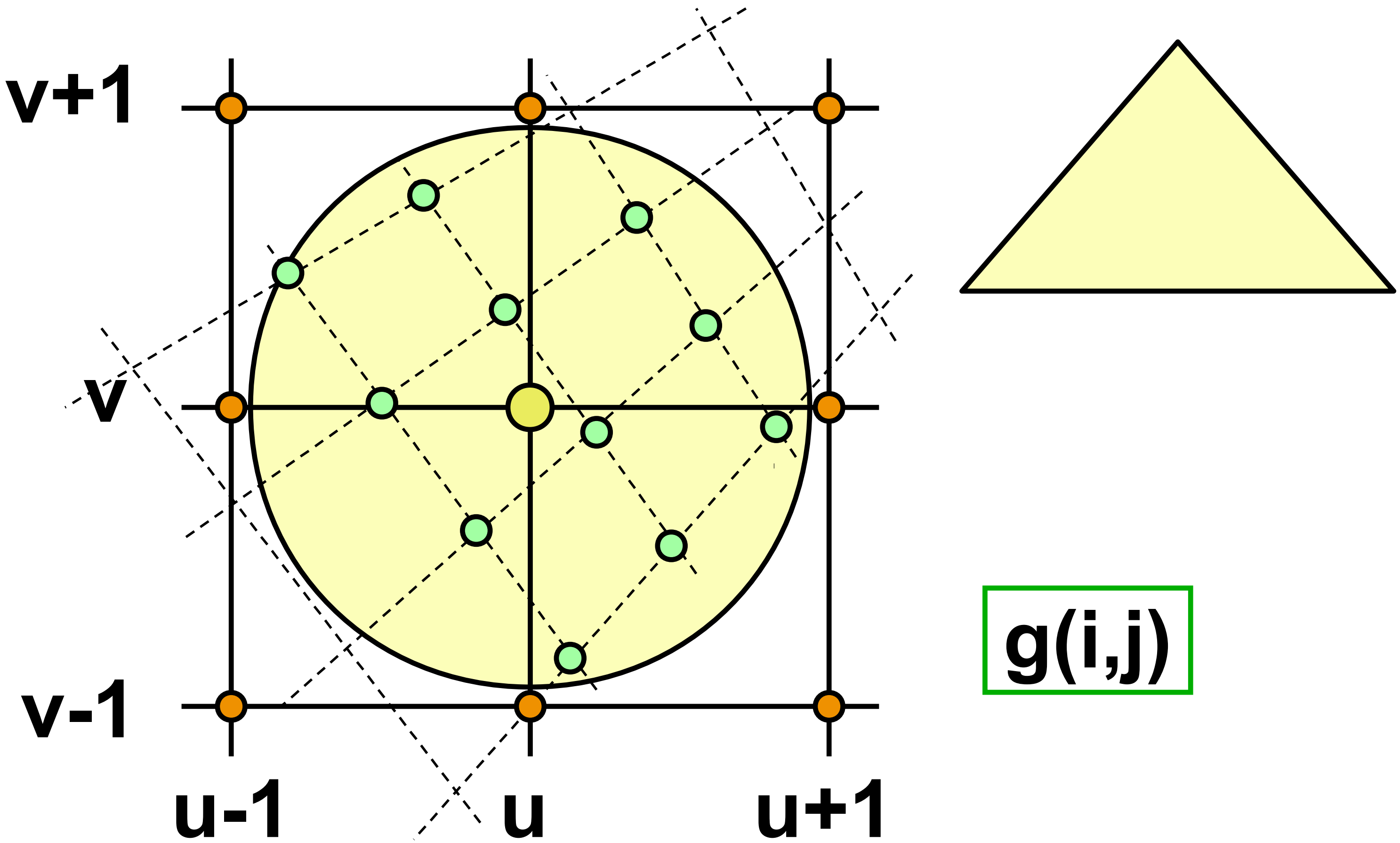
Interpolation in source coord. system



Transform with filtering

- **pixel-area** model
  - digitizing filter has areal support (e.g. box or conical filter)
- **source pixels** are projected to target coordinate system
  - only **g** is needed
- suitable also for **contractive transforms**
  - isometric transform ⇒ image is blurred
  - big contraction ⇒ high computing time (speedup needed)

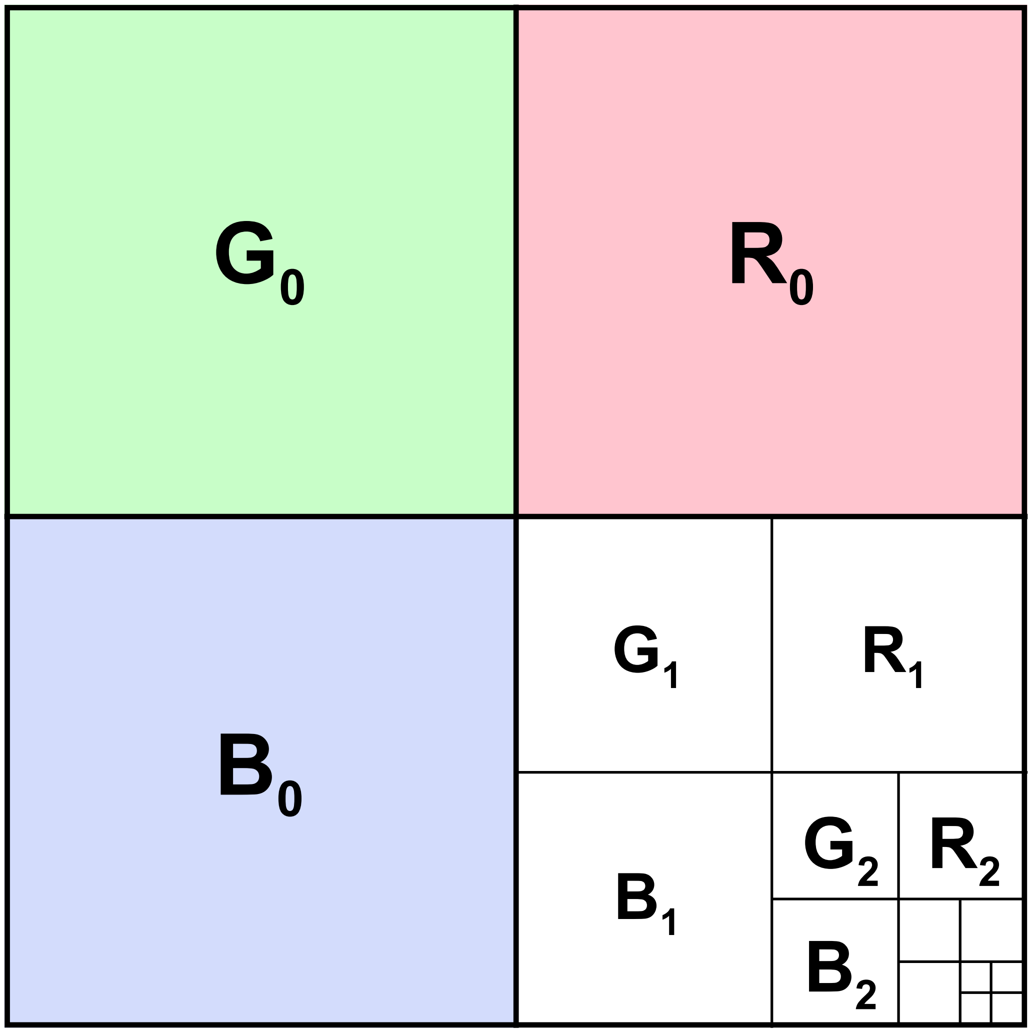
Transform with filtering



„MIP map“ (*multum in parvo*)

- **pyramidal representation** of the source image (hierarchy)
  – preprocessing = pre-filtering
- useful mostly for **transforms with big contraction**
  – substantial speedup
  – used mostly for **texture mapping** on the GPU (distant objects)
- **compact storage**
  – only **4/3** size of original RGB image

„MIP map”



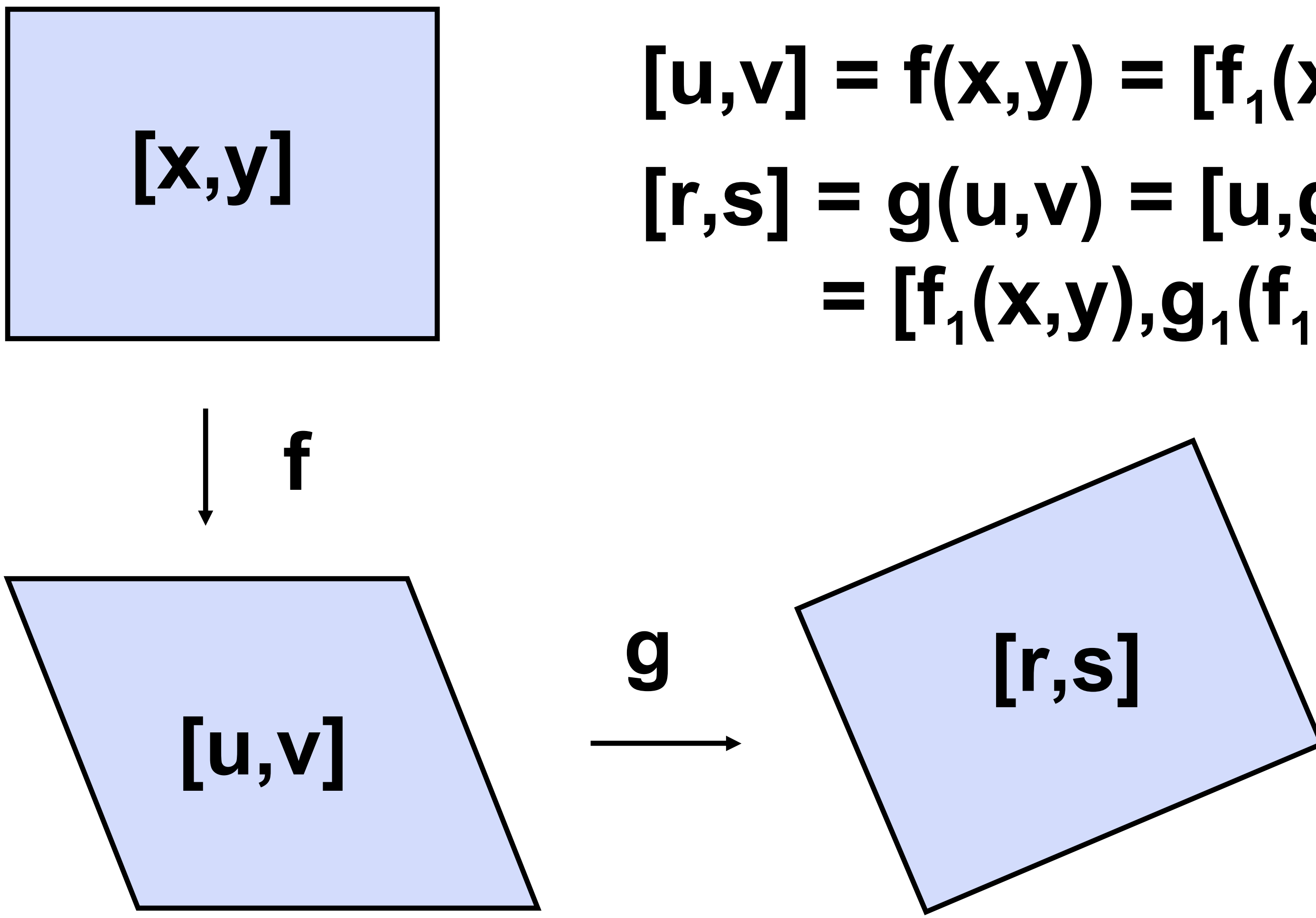
Multi-pass algorithms

- transform is factored into several **consecutive steps**
  – each step works either **on rows** or **on columns** of an image
- **faster computation**
  – simpler filtering
  – two 1D filters are faster than a 2D one
- **bottleneck problem**
  – partial mapping is highly contractive or even non-injective

Two-pass rotation

$$[u,v] = f(x,y) = [f_1(x,y),y]$$

$$[r,s] = g(u,v) = [u,g_1(u,v)]$$
$$= [f_1(x,y),g_1(f_1(x,y),v)]$$



Derivation

Target transform (rotation by **α**):

$$r = x \cdot \cos\alpha - y \cdot \sin\alpha$$
$$s = x \cdot \sin\alpha + y \cdot \cos\alpha$$

First transform **f**:

$$u = f_1(x, y) \qquad v = y$$

Second transform **g**:

$$r = u = f_1(x, y) \qquad s = g_1(u, v) = g_1[f_1(x, y), y]$$

Derivation

Horizontal shear for the first pass:

$$f_1(x,y) = x \cdot \cos\alpha - y \cdot \sin\alpha$$

Vertical shear for the second pass:

$$g_1(u,v) = x \cdot \sin\alpha + y \cdot \cos\alpha = x \cdot \sin\alpha + v \cdot \cos\alpha$$

$$u = f_1(x,v) = x \cdot \cos\alpha - v \cdot \sin\alpha$$

$$x = (u + v \cdot \sin\alpha) / \cos\alpha$$

$$g_1(u,v) = u \cdot \tan\alpha + v \cdot \sec\alpha$$

Three-pass rotation

Rotation matrix factoring:

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

$$= \begin{bmatrix} 1 & -\tan\alpha/2 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -\tan\alpha/2 \\ 0 & 1 \end{bmatrix}$$

sufficient for angles: $-\pi/2 \le \alpha \le \pi/2$

Arbitrary angle

- **multi-pass methods** cannot be used for wide range of angles. Suitable angle ranges:

  $-45° \le \alpha \le 45°$ for two-pass rotation

  $-90° \le \alpha \le 90°$ for three-pass rotation

- **auxiliary rotation** (multiple of **90°**)
  – simple, fast
  – no image degradation

General separable transform

Separable transform:

$$\mathbf{h}(\mathbf{x},\mathbf{y}) = [\mathbf{h}_1(\mathbf{x},\mathbf{y}), \mathbf{h}_2(\mathbf{x},\mathbf{y})] = \mathbf{g}(\mathbf{f}(\mathbf{x},\mathbf{y}))$$

$$\mathbf{f}(\mathbf{x},\mathbf{y}) = [\mathbf{f}_1(\mathbf{x},\mathbf{y}), \mathbf{y}]$$

$$\mathbf{g}(\mathbf{u},\mathbf{v}) = [\mathbf{u}, \mathbf{g}_1(\mathbf{u},\mathbf{v})]$$

$$\mathbf{f}_1(\mathbf{x},\mathbf{y}) = \mathbf{h}_1(\mathbf{x},\mathbf{y})$$

$$\mathbf{g}_1(\mathbf{u},\mathbf{v}) = \mathbf{h}_2(\varphi(\mathbf{u},\mathbf{v}), \mathbf{v})$$

if $\exists\ \varphi(\mathbf{u},\mathbf{v})$ such that

$$\mathbf{x} = \varphi(\mathbf{u},\mathbf{v})$$

Image degradation

Inapplicable areas: low values of the derivative

$$\frac{\partial h_1(x,y)}{\partial x}$$

or high values of the derivative $\frac{\partial g_1(u,v)}{\partial v}$

Sometimes even the inverse function could not be defined $\varphi(u,v)$!

$\Rightarrow$ concurrent processing of images I and I^T

Optimized algorithm

- **concurrent processing** of original **I** and transposed $\mathbf{I}^T$ image
  – both branches use the same method (two-pass algorithm using separable transforms **h** and $\mathbf{h}^T$)
- result pixels are composed from $\mathbf{h(I)}$ and $\mathbf{(h^T(I^T))^T}$
  – comparing degradation pixel by pixel
- even for „non-injective“ deformations!
  – image „folding”..

The End

More info:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 815-832
- **J. Gomes et al.**: ***Warping and Morphing of Graphical Objects***, Course Notes - SIGGRAPH'95