The Perspective Transform in Embedded Vision

Aditya Joshi, Lead Design Engineer

Shrinivas Gadkari, Design Engineering Director

05/22/2018

cadence™

Outline of the Presentation

- What is perspective transform
- Why It is important – Applications
- How it is estimated and applied
- Challenges in implementation on DSP
- Compute Efficient Solution

Understanding Perspective Transform

Image Capture from Perspective of the Camera

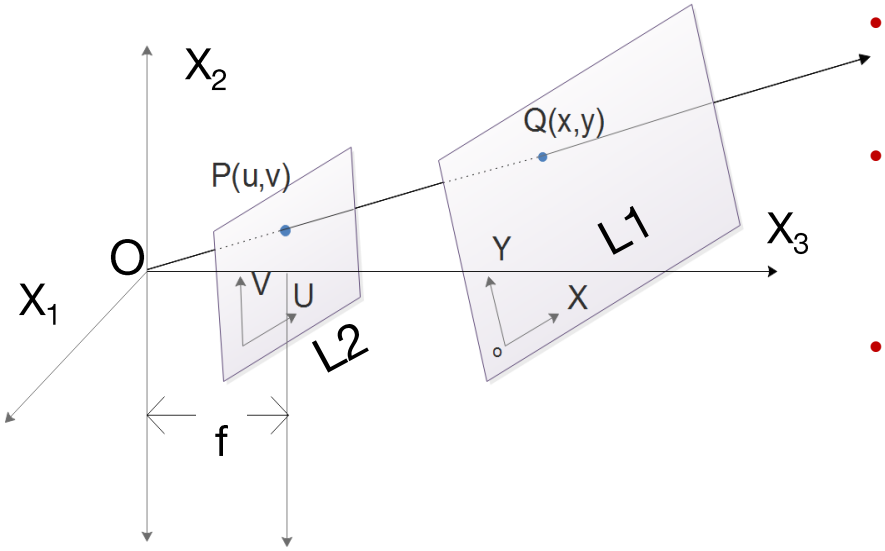
View from Left

Front View

View from Right

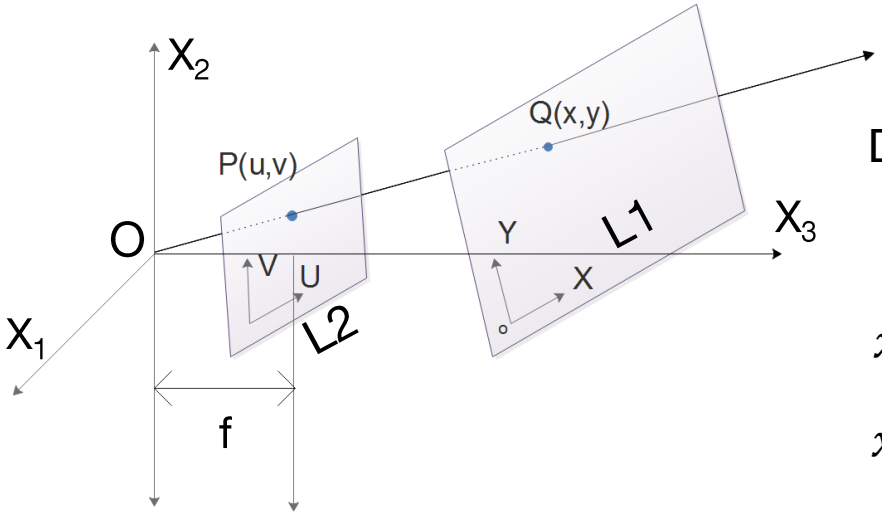
Captured Image depends on relative position – Perspective of the camera

Mathematical Model of Perspective Projection - 1



- Q : Point in physical plane L1
  - (x, y) – coordinates of Q in physical plane
- P : Projection of Q on image plane L2
  - (u, v) – coordinates of P in image plane
  - f – camera focal length
- Projected (u,v) are related to Physical (x,y) as:
  - $u = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}$
  - $v = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$
- This is the **Perspective Transform** between destination coordinates (u,v) and source coordinates (x,y)

Mathematical Model of Perspective Projection -2



- $\overline{\mathbf{v_0}}$ is vector representation of the origin of plane L1
- $\hat{e_X}$, $\hat{e_Y}$ unit vectors along X, Y axes
- $\hat{e_1}$, $\hat{e_2}$, $\hat{e_3}$ unit vectors along X_1, X_2, X_3 axes

Deriving the Perspective Transform Equations:

$$\overline{Q} = \hat{e_X}.x + \hat{e_Y}.y + \overline{v_0}$$

$$x_1 = \overline{Q}.\hat{e_1} = h_{11}x + h_{12}y + h_{13}$$

$$x_2 = \overline{Q}.\hat{e_2} = h_{21}x + h_{22}y + h_{23}$$

$$x_3 = \overline{Q}.\hat{e_3} = h_{31}x + h_{32}y + h_{33}$$

$$h_{i1} = \hat{e_X} \cdot \hat{e_i}$$
$$h_{i2} = \hat{e_Y} \cdot \hat{e_i}$$
$$h_{i3} = \overline{\mathbf{v}_0} \cdot \hat{e_i}$$

$$u = \frac{x_1}{x_3} * f \qquad v = \frac{x_2}{x_3} * f$$

Mathematical Model of Perspective Transform - 3

- In camera 1 perspective:
  - Coordinates (X, Y) of point Q related to coordinates (x_s, y_s) of projected point, P_s, via Perspective transform with coefficients

    $H = \{h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33}\}$

- In camera 2 perspective:
  - Coordinates (X, Y) of point Q related to coordinates (x_d, y_d) of projected point, P_d, via Perspective transform with coefficients:

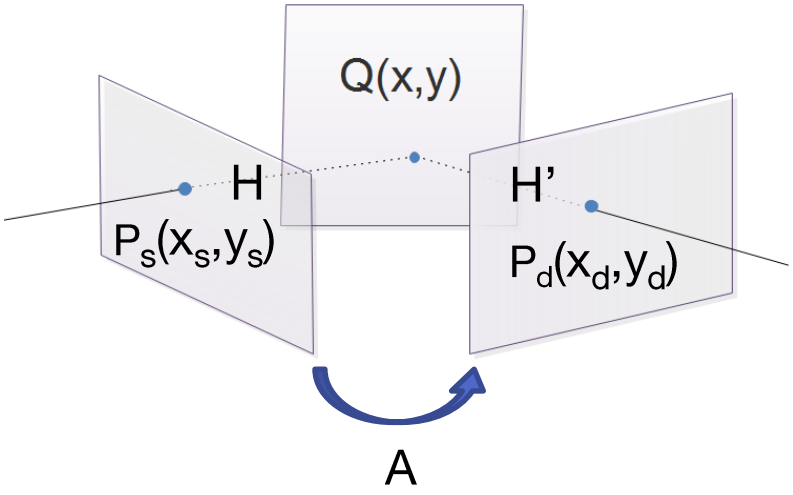
    $H' = \{h'_{11}, h'_{12}, h'_{13}, h'_{21}, h'_{22}, h'_{23}, h'_{31}, h'_{32}, h'_{33}\}$

- It can be also shown that coordinates of $P_s = (x_s, y_s)$ are related coordinates of $P_d = (x_d, y_d)$ via a Perspective Transform:

  $A=\{a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, 1.0\}$

$$x_s = \frac{a_{11}x_d + a_{12}y_d + a_{13}}{a_{31}x_d + a_{32}y_d + 1.0}, \qquad y_s = \frac{a_{21}x_d + a_{22}y_d + a_{23}}{a_{31}x_d + a_{32}y_d + 1.0}$$

- This mapping of coordinates from one perspective (x_s, y_s) to another perspective (x_d, y_d) is central to many imaging and vision applications.



Applications of Perspective Transform

Application Example 1: Image Stitching

- To create a panoramic image of a scene using two (or more) sub images
- Use overlapping area in sub images to estimate homography
- Apply homography to align all images in one perspective
- Combine (stitch) images after perspective alignment.

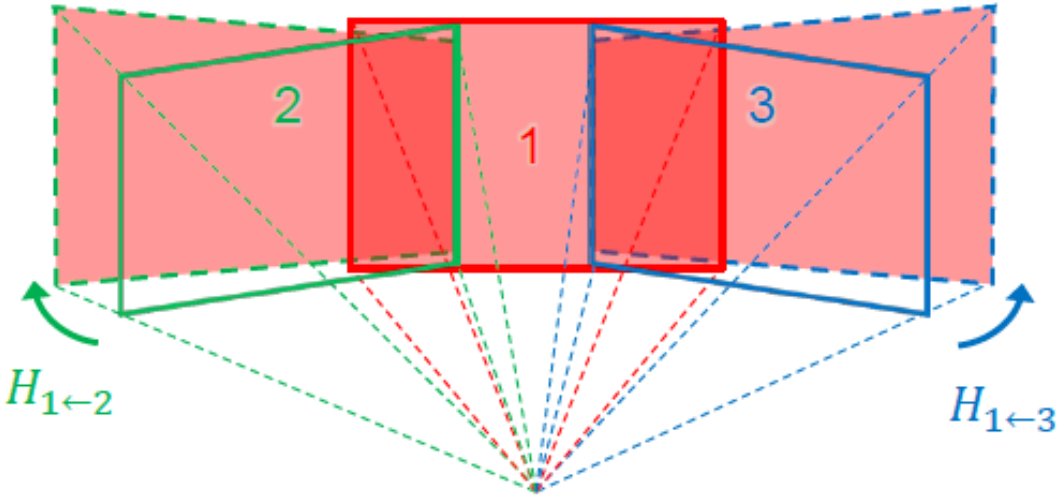


Image Stitching

Left Image

Right Image

Perspective align + combine

Application Example 2: Sensor Noise Filtering

- Basic Idea: Improve image resolution by temporal averaging of random sensor noise across multiple captured frames
- Practical Issue: Each image is captured from a slightly different camera position – (see circled area in images below)
  - Solution: Align the images to perspective of any one frame + temporal averaging

Noisy Image -1

+

Noisy Image -2

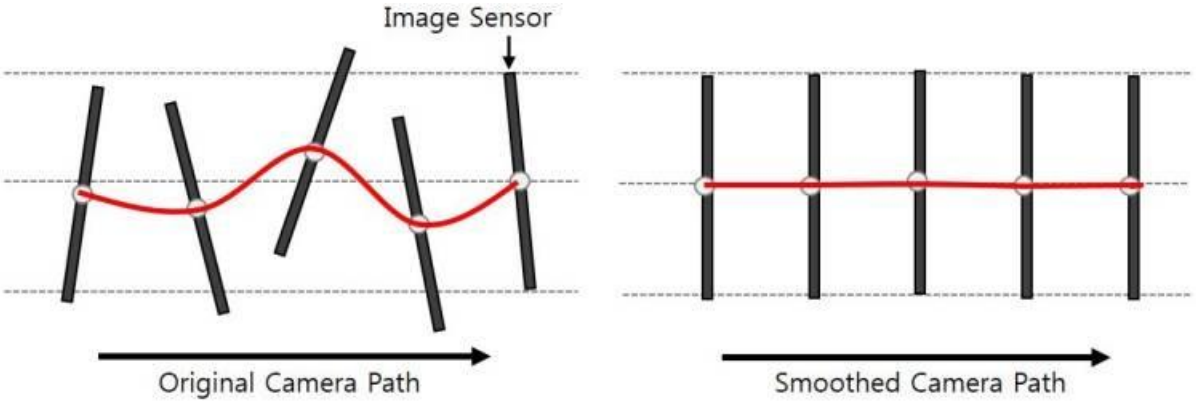
+ …

=

Realign + Temporal Filter

Application Example 3: Video Stabilization

- Basic Idea: Reduce jerkiness in captured image sequence by choosing a smooth change in camera perspective across frames



Video Stabilization

Frame-1

Frame-2: With Jerkiness

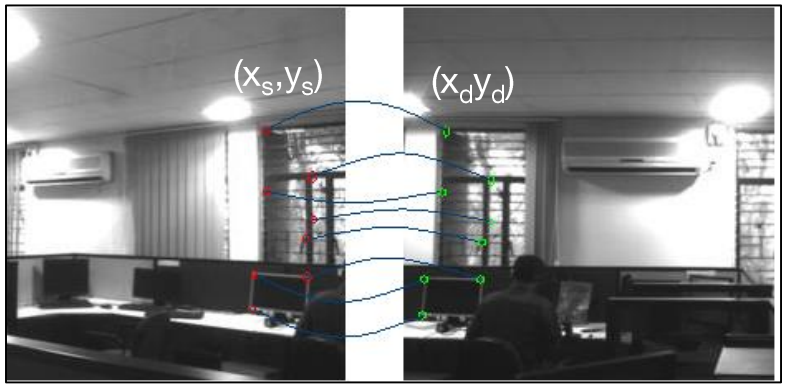
Frame-3

Frame-2 : Smooth perspective change

Estimation and Application of Perspective Transform

Estimating the Perspective Transform

- Problem: Given two images – two perspectives of the same physical image, estimate transform coefficients: A = $\{a_{ij}\}$
- Step 1: Find matching key points $(x_s,y_s) \leftrightarrow (x_d,y_d)$ in the two images
  - example Harris Corner Detection, BRIEF descriptor matching
- Step 2: Create the following matrix equation and solve for the coefficient vector.



Matching Key Points (x_d, y_d) and (x_s, y_s)

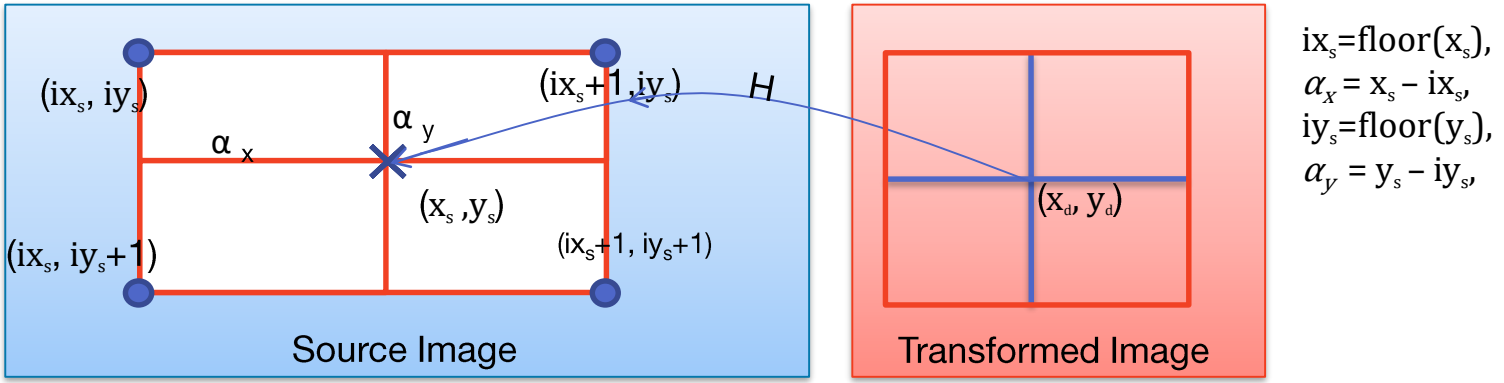
$$\begin{bmatrix} x_{d1} & y_{d1} & 1 & 0 & 0 & 0 & -x_{s1}x_{d1} & -x_{s1}y_{d1} \\ 0 & 0 & 0 & x_{d1} & y_{d1} & 1 & -y_{s1}x_{d1} & -y_{s1}y_{d1} \\ & & & & \vdots & & & \\ x_{dn} & y_{dn} & 1 & 0 & 0 & 0 & -x_{sn}x_{dn} & -x_{sn}y_{dn} \\ 0 & 0 & 0 & x_{dn} & y_{dn} & 1 & -y_{sn}x_{dn} & -y_{sn}y_{dn} \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} x_{s1} \\ y_{s1} \\ \vdots \\ x_{sn} \\ y_{sn} \end{bmatrix}$$

Applying the Perspective Transform

- For each pixel in transformed (destination) image at location (x_d, y_d):
  - Compute the Source co-ordinates (x_s ,y_s)

$$x_s = \frac{a_{11}x_d + a_{12}y_d + a_{13}}{a_{31}x_d + a_{32}y_d + 1} \qquad y_s = \frac{a_{21}x_d + a_{22}y_d + a_{23}}{a_{31}x_d + a_{32}y_d + 1}$$

  - Estimate the source pixel value at (x_s ,y_s) using bilinear interpolation:



$ix_s = floor(x_s)$,
$\alpha_x = x_s - ix_s$,
$iy_s = floor(y_s)$,
$\alpha_y = y_s - iy_s$,

Porting Challenges on DSP

Implementation challenges on DSP

- Computational Challenge:
  - Division tends to be an expensive operation on a DSP (in fact on all embedded processors)
  - Per pixel division needed to compute x_s and y_s creates computational bottleneck
- Example of computational requirements on a Vision DSP
  - Total processing cycles for Perspective correction of 2Kx1K image - 2.8 M cycles
  - 2 M cycles are consumed by divide operations alone
- This is the main motivation to explore minimal division implementation of the Perspective transform

Compute efficient Perspective Transform -1

Key Idea:

- Frame-to-frame change in camera perspective is minimal
- Resulting value of coefficients (a_{31},a_{32}) are very small
- Full division can be applied for coordinate estimation only for a small subset of pixels
- For other pixels use bilinear interpolation of these exactly computed coordinates
- Dynamically determine the subset size based on actual values (a_{31},a_{32})
  - Limit the resulting coordinate estimation error < Err_{Max}

Compute efficient Perspective Transform - 2

- Details:
  - Use modest tile sizes : (8x8), (16x16), (32x32)
  - Use full divide based Perspective equation to compute only the source coordinates of four tile corner pixels
  - For all other pixels in tile use bilinear interpolation of computed tile corner source coordinates to estimate the source coordinates
  - Dynamically determine the tile size based on the coefficient values (a_{31},a_{32}) to limit the coordinate estimation error < Err_{Max}
    - Err_{Max} : maximum allowed coordinate error
    - Err_{Max} range : 0.01 – 0.05 (depending on application architect specification)

Division Free Perspective Transform : Results

Dynamically Determined Tile Size	Computation Cycles on Tensilica® Vision DSP on (2Kx1K image) (Mega Cycles)	Max Coordinate Error	Computational Savings
Full Perspective Transform	2.77	0.0	1 X
4x4	1.03	< 0.01	2.7 X
8x8	0.86	< 0.01	3.2 X
16x16	0.81	< 0.05	3.4 X
32x32	0.80	< 0.05	3.45 X

- Observation: By dynamically adjusting the tile size based on perspective coefficient values and the prescribed limit on coordinate error, performance speed up in the range of 3x can be achieved.

Summary

- Perspective transform accurately computes an image as seen from a different camera position (perspective).
- Used in the imaging applications which involve joint processing of multiple pictures (frames) taken from slightly different camera positions.
- Per-cycle-divide operation involved in a canonical implementation poses a large computational complexity problem.
- We described a minimal division modification of Perspective transform - achieves DSP cycles reduction by a factor of 3x for typical use cases.

References

1. C. Vancea and S. Nedevschi, "LUT-based image rectification module implemented in FPGA," in Proc. 2007 IEEE International Conference on Intelligent Computer Communication and Processing, pp. 147-154, Sept. 2007.
2. Yu-Hsi Chen and Hsueh-Yi Sean Lin, “Full-frame Video Stabilization via SIFT Feature Matching” in International Conference on Intelligent Information Hiding and Multimedia Signal Processing 2014
3. B. Barenbrug, F. J. Peters, C. W. A. M. van Overveld, “Algorithmsfor division free perspective correct rendering,” Proc. ACM SIGGRAPH/EUROGRAPHICS workshop Graph. Hardware, pp. 7-13, Aug. 2000.

THANK YOU