Projective Geometry in Computer Vision: The Mathematics Behind Image Transformations

Shashank Arava

December 5, 2025

Abstract

Projective geometry extends Euclidean geometry by introducing "points at infinity" where parallel lines meet. While Euclidean geometry preserves lengths and angles, projective geometry preserves only incidence and the *cross ratio*—properties essential for modeling camera imaging. This paper explores homogeneous coordinates, the projective plane $\mathbb{P}^2$, and projective transformations, drawing from Birchfield's introduction. We demonstrate how these concepts enable computer vision algorithms including homography-based perspective correction and image stitching. Images from a self-written Python implementation provide visualizations of example applications.

1 Introduction

Anyone who has taken a photograph knows that parallel lines in the real world—like railroad tracks or building edges—appear to converge in the image. This reveals a fundamental truth: the camera does not preserve Euclidean geometry. A rectangular door appears as a quadrilateral; a circular rim becomes an ellipse; parallel railroad tracks meet at the horizon. These are not optical illusions—they are consequences of perspective projection.

Euclidean geometry preserves lengths, angles, and parallelism—precisely what photographs do *not* preserve. To reason mathematically about images, we need a geometry that embraces these transformations. Projective geometry provides exactly this framework, extending Euclidean geometry by adding "ideal points" where parallel lines meet [1].

This connection has enormous practical importance. Computer vision powers applications from autonomous vehicles to augmented reality. At the heart of many applications lie projective algorithms: homography estimation for image stitching, the fundamental matrix for stereo vision, and camera calibration for 3D reconstruction.

This paper provides an accessible introduction to projective geometry with emphasis on computer vision applications. We develop the mathematical foundations—homogeneous

coordinates, the projective plane $\mathbb{P}^2$, and projective transformations—then show how these enable practical algorithms.

2 Mathematical Foundations

2.1 The Hierarchy of Geometries

As Birchfield [1] explains, there are four major geometries, each defined by its allowed transformations and preserved properties:

Table 1: The four geometries and their invariants [1].

Property	**Euclidean**	**Similarity**	**Affine**	**Projective**
Length	✓			
Angle	✓	✓		
Ratio of lengths	✓	✓	✓	
Parallelism	✓	✓	✓	
Incidence	✓	✓	✓	✓
Cross ratio	✓	✓	✓	✓

Euclidean geometry (rotations and translations) preserves the most properties. Similarity adds uniform scaling. Affine adds non-uniform scaling and shear. Projective geometry—the most general—preserves only incidence and the cross ratio. Since camera imaging is a projective process, projective geometry is the natural framework for computer vision.

2.2 Homogeneous Coordinates

The key to computational projective geometry is *homogeneous coordinates*. A Euclidean point (x, y) becomes a triple (X, Y, W) where $x = X/W$ and $y = Y/W$. Crucially, homogeneous coordinates are defined only up to scale: $(X, Y, W) \sim (\lambda X, \lambda Y, \lambda W)$ for any $\lambda \neq 0$.

Definition 2.1 (Homogeneous Coordinates)**.** A point $\mathbf{p} \in \mathbb{P}^2$ is represented by (X, Y, W) where $(X, Y, W) \sim (\lambda X, \lambda Y, \lambda W)$ for all $\lambda \neq 0$, and $(0, 0, 0)$ is excluded.

Lines also gain an elegant representation. The line $ax + by + c = 0$ becomes $\mathbf{u}^T\mathbf{p} = 0$, where $\mathbf{u} = (a, b, c)^T$. Remarkably, *points and lines have the same representation*—both are triples of numbers.

2.2.1 Ideal Points and the Line at Infinity

Points with $W = 0$ are *ideal points* or *points at infinity*. They cannot be converted to Euclidean coordinates but are valid projective points.

Consider two parallel lines $ax + by + c_1 = 0$ and $ax + by + c_2 = 0$. Their intersection is:

$$\mathbf{p} = \mathbf{u}_1 \times \mathbf{u}_2 = (b(c_2 - c_1), a(c_1 - c_2), 0)^T \tag{1}$$

This is an ideal point. In projective geometry, parallel lines *do* meet—at infinity. All ideal points lie on the *line at infinity* $\mathbf{l}_\infty = (0, 0, 1)^T$.

Definition 2.2 (The Projective Plane)**.** $\mathbb{P}^2$ is the affine plane augmented by a single ideal line containing all ideal points, where ideal entities are indistinguishable from ordinary ones.

2.2.2 Operations in Homogeneous Coordinates

Common operations reduce to linear algebra:

- **Line through two points:** $\mathbf{u} = \mathbf{p}_1 \times \mathbf{p}_2$
- **Intersection of two lines:** $\mathbf{p} = \mathbf{u}_1 \times \mathbf{u}_2$
- **Collinearity:** $\det[\mathbf{p}_1\ \mathbf{p}_2\ \mathbf{p}_3] = 0$
- **Concurrence:** $\det[\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3] = 0$

This reveals the *duality* between points and lines: any theorem about points becomes a theorem about lines by swapping these words.

2.3 The Cross Ratio

Since projective geometry preserves neither lengths nor ratios, what *is* preserved? The *cross ratio*—a ratio of ratios that remains invariant under all projective transformations.

Definition 2.3 (Cross Ratio)**.** For four collinear points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$:

$$\text{CR}(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) = \frac{|\mathbf{p}_1\mathbf{p}_3| \cdot |\mathbf{p}_2\mathbf{p}_4|}{|\mathbf{p}_1\mathbf{p}_4| \cdot |\mathbf{p}_2\mathbf{p}_3|} \tag{2}$$

While projective transformations may distort configurations, the cross ratio remains unchanged—making it useful for recognition in perspective-distorted images.

2.4 Projective Transformations

A *projective transformation* (or *homography*) maps $\mathbb{P}^2$ to itself while preserving collinearity:

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \tag{3}$$

where $\mathbf{H}$ is a 3×3 non-singular matrix with 8 degrees of freedom (9 elements minus 1 for scale). Different transformation classes correspond to restrictions on $\mathbf{H}$:

- **Euclidean (3 DOF):** Rotation + translation
- **Similarity (4 DOF):** Adds uniform scale

- **Affine (6 DOF):** Adds non-uniform scale, shear; $(h_{31}, h_{32}) = (0, 0)$
- **Projective (8 DOF):** General 3×3 matrix

When $(h_{31}, h_{32}) \neq (0, 0)$, the transformation includes perspective effects.

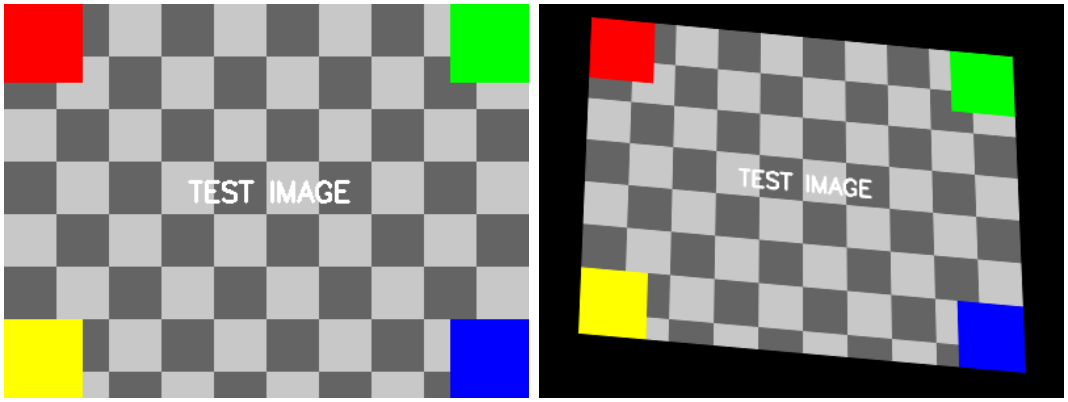


Figure 1: Left: Original test image. Right: After projective transformation. The rectangle becomes a quadrilateral and parallel edges converge.

3 Applications to Computer Vision

3.1 The Homography Matrix

A single 3×3 matrix relates points in two images when: (1) all visible points lie on one plane, or (2) the camera rotates without translation. In both cases: $\mathbf{p}_2 \sim \mathbf{H}\mathbf{p}_1$.

3.1.1 Computing Homography: The DLT Algorithm

Given $n \geq 4$ correspondences, the *Direct Linear Transform (DLT)* algorithm [2] computes $\mathbf{H}$. Each correspondence gives two linear equations, forming a system $\mathbf{A}\mathbf{h} = \mathbf{0}$ solved via SVD. Normalization is critical [3].

3.2 Perspective Correction

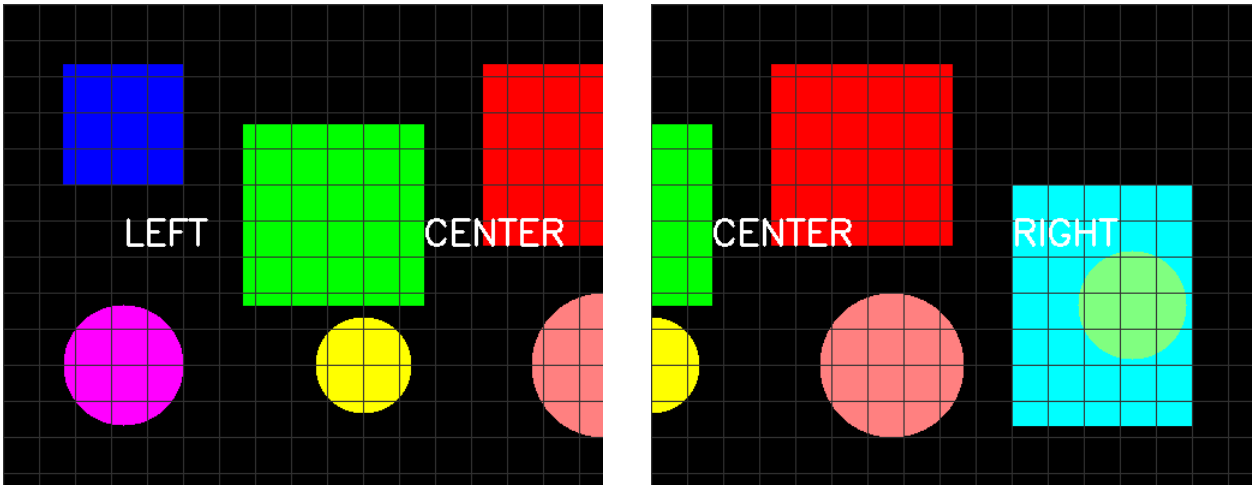
Perspective correction transforms a quadrilateral back to a rectangle—exactly what document scanner apps do. The process: (1) identify four corners, (2) specify desired rectangle corners, (3) compute homography via DLT, (4) apply to all pixels.

3.3 Image Stitching

Image stitching combines overlapping images into a panorama. For a rotating camera, views are related by homographies. The pipeline:

1. **Feature detection:** Find distinctive points (SIFT, ORB, AKAZE)

2. **Feature matching:** Find correspondences between images
3. **Homography estimation:** Compute **H** using RANSAC for robustness
4. **Warping:** Transform one image to align with the other
5. **Blending:** Combine warped images seamlessly



(a) Left input image (b) Right input image

Figure 2: Two overlapping test images for stitching demonstration.

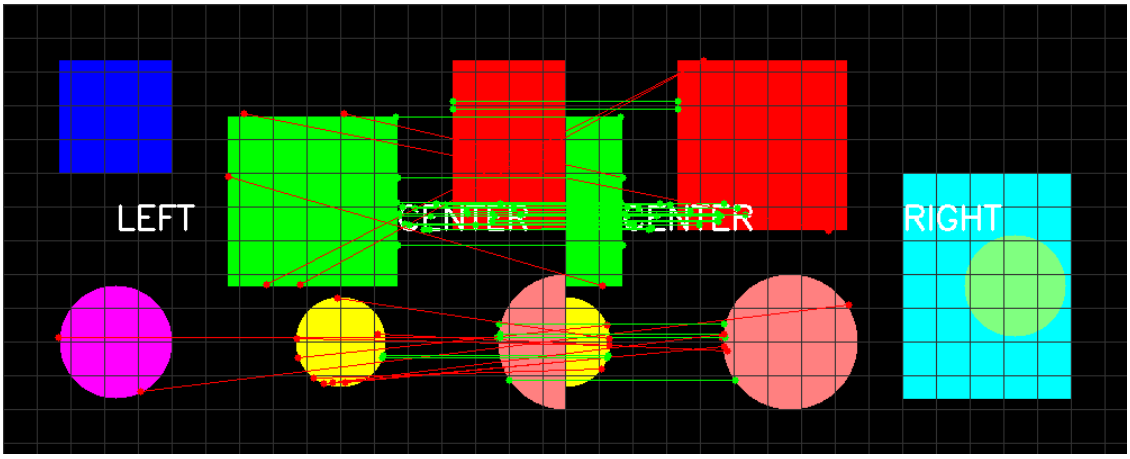


Figure 3: Feature matching between images. Green lines connect inlier correspondences verified by RANSAC; red lines indicate rejected outliers.

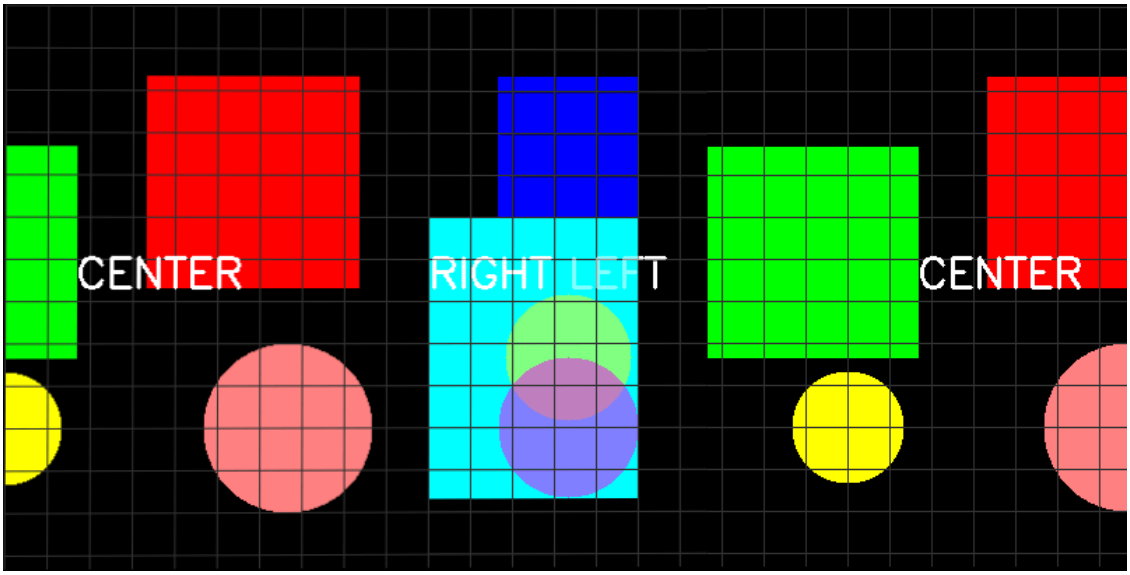


Figure 4: The final stitched panorama, created by warping one image using the homography and blending.

The computed homography was approximately a pure translation ($h_{13} \approx -302$ pixels), which makes sense since the test images were horizontal shifts.

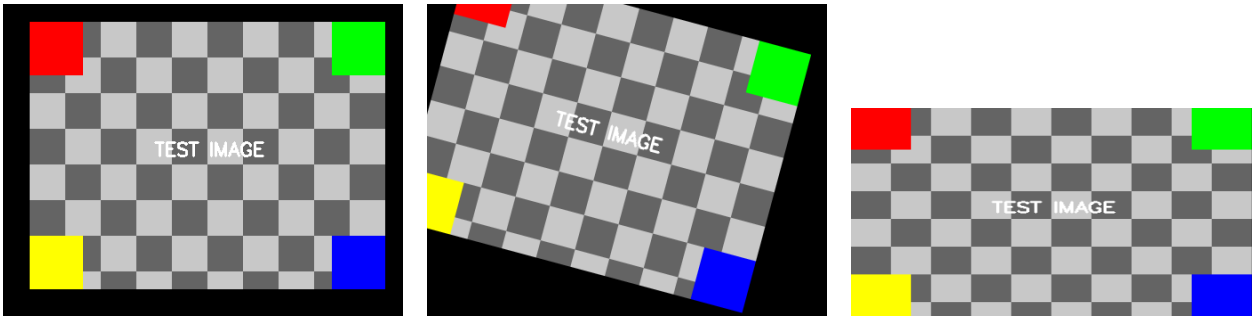
3.4 The Fundamental Matrix

For general camera motion (not just rotation), the *fundamental matrix* $\mathbf{F}$ encodes the epipolar geometry [2]. For the corresponding points: $\mathbf{m}_2^T \mathbf{F} \mathbf{m}_1 = 0$. The fundamental matrix has 7 DOF and enables stereo vision and 3D reconstruction.

4 Python Visualization

A Python implementation demonstrates these algorithms with detailed comments explaining the mathematics:

- `homography.py`: DLT algorithm with normalization
- `image_stitching.py`: Complete stitching pipeline
- `perspective_correction.py`: Interactive perspective correction



(a) Translation (b) Rotation (c) Non-uniform scaling

Figure 5: Special cases of projective transformations, each preserving additional structure.

5 Conclusion

Projective geometry has become indispensable to modern computer vision. The main takeaways are:

1. **Homogeneous coordinates** unify points and lines, elegantly handling infinity.
2. **The projective plane** $\mathbb{P}^2$ is the natural setting for describing images.
3. **Projective transformations** (8 DOF) encompass all simpler transformations.
4. **The cross ratio** is the fundamental projective invariant.
5. **Applications** including stitching, perspective correction, and 3D reconstruction all rely on projective geometry.

From smartphone document scanners to autonomous vehicles, projective geometry provides the mathematical foundation for understanding images.

References

[1] Birchfield, S. (1998). *An introduction to projective geometry (for computer vision)*. Unpublished note, Stanford university, 14.

[2] Hartley, R., Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press.

[3] Hartley, R. I. (1997). *In defense of the eight-point algorithm.* IEEE Transactions on pattern analysis and machine intelligence, 19(6), 580-593.

[4] Szeliski, R. (2022). *Computer vision: algorithms and applications*. Springer Nature.

[5] Stillwell, J., Ribet, K. A., and S.. Axler. (2005). *The four pillars of geometry (Vol. 2)*. New York: Springer.

Python Implementation: https://github.com/ssarava/MATH430-Final-Project