Computational Photography

* Study the basics of computation and its impact on the entire workflow of photography, from capturing, manipulating and collaborating on, and sharing photographs.

Georgia Tech College of Computing

Making a Panorama

(Lords Cricket Ground, London, UK, by I. Essa)

Georgia Tech College of Computing

Lesson Objectives

1. Generate a Panorama
2. Image Re-projection
3. Homography from a pair of images
4. Computing inliers and outliers
5. Details of constructing panoramas

Review: 5 Steps to Make a Panorama

(Lords Cricket Ground, London, UK, by I. Essa)

* ✓ Capture Images
* ✓ Detection and matching
* ✓ Warping → Aligning Images
* ✓ Blending, Fading, Cutting
* Cropping (Optional)

Align Images: Translate??

L

R

L on top

R on top

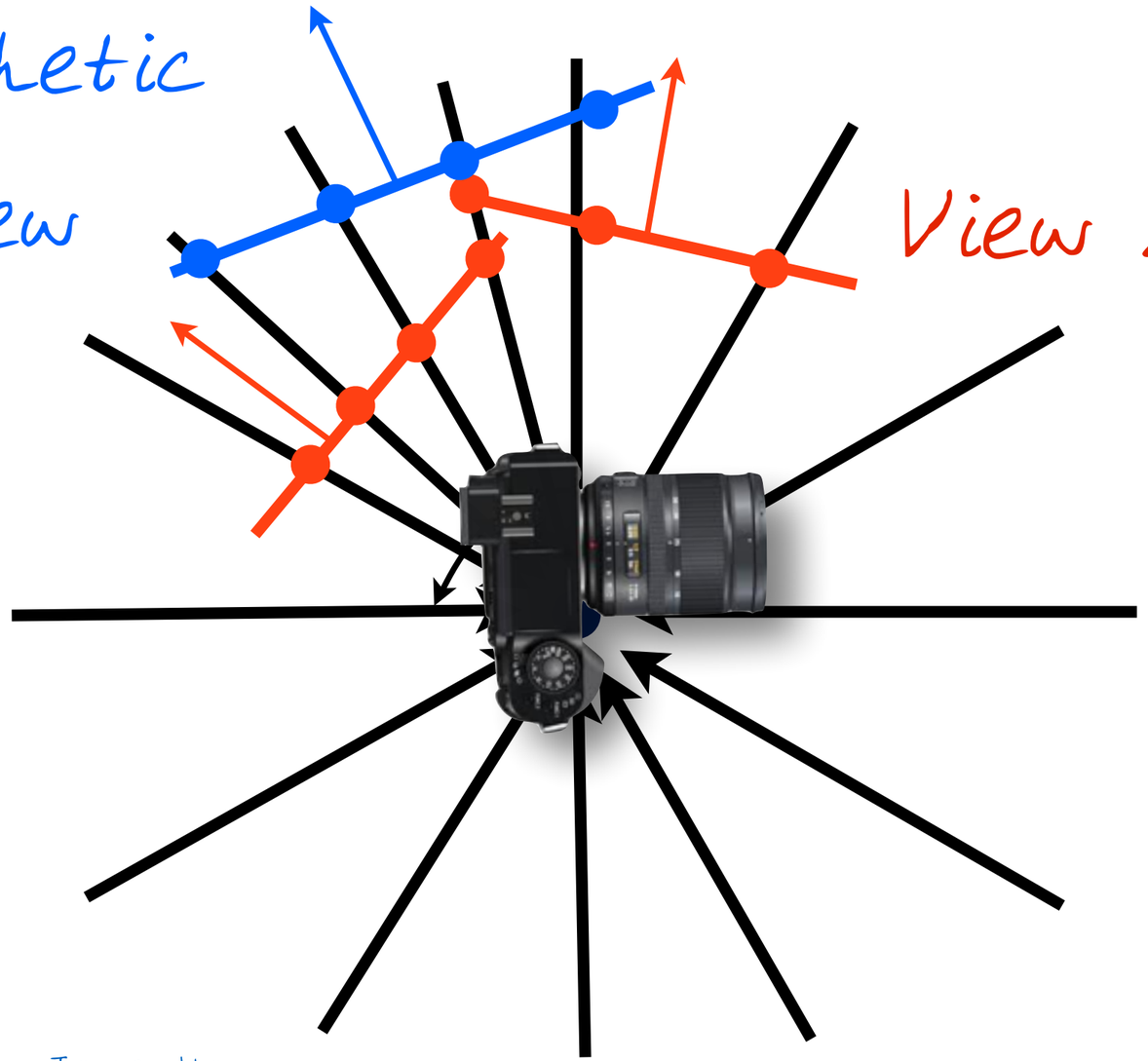
Better: Warp

A Bundle of Rays Contains all Views

Synthetic View

View 2

View 1



Possible to generate any synthetic camera view as long as it has the same center of projection!

Slide motivated by James Hays

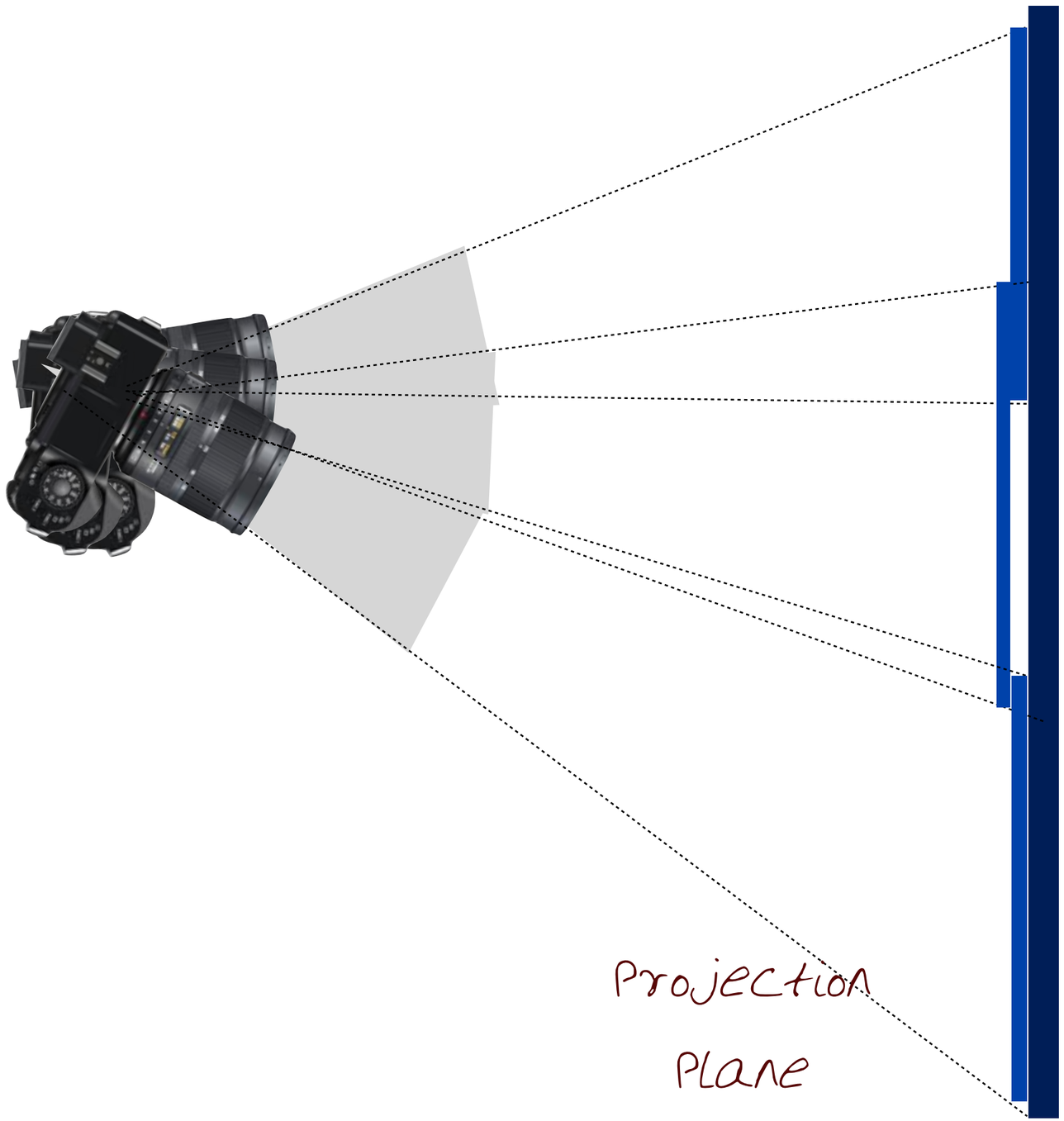


Image Re-projection to Panorama Projection Plane

* The panorama mosaic has a natural interpretation in 3D
* Images are reprojected onto a common plane
* The mosaic is formed on this plane
* Mosaic is a synthetic wide-angle camera

Image Re-Projection (1)

To relate two images from the same camera center and map a pixel from **PP1** to **PP2**:

* Cast a ray through each pixel in **PP1**
* Draw the pixel where that ray intersects **PP2**

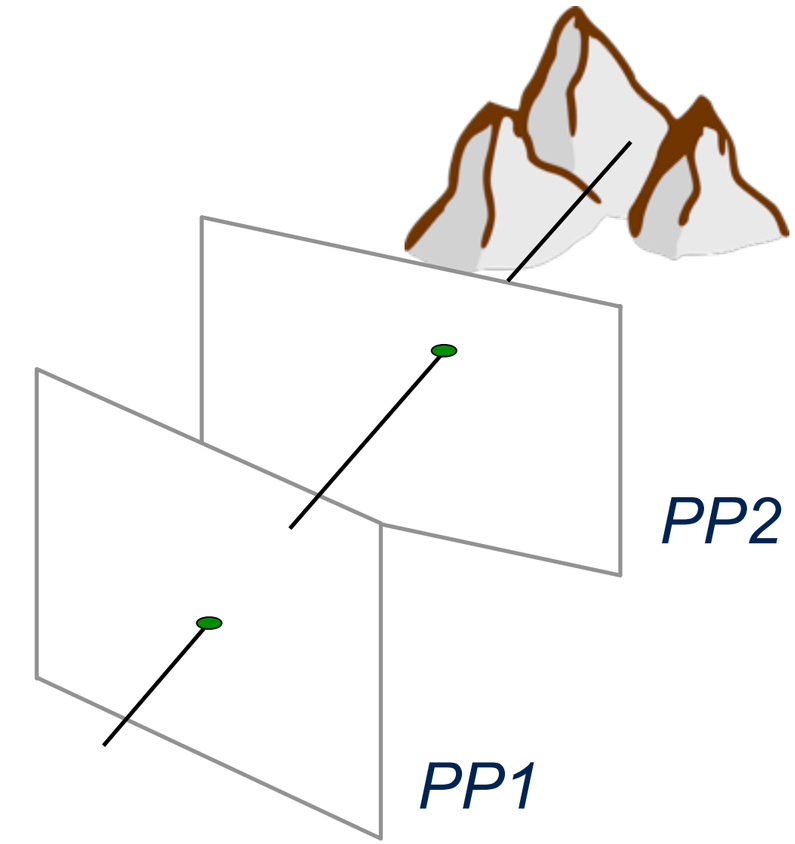
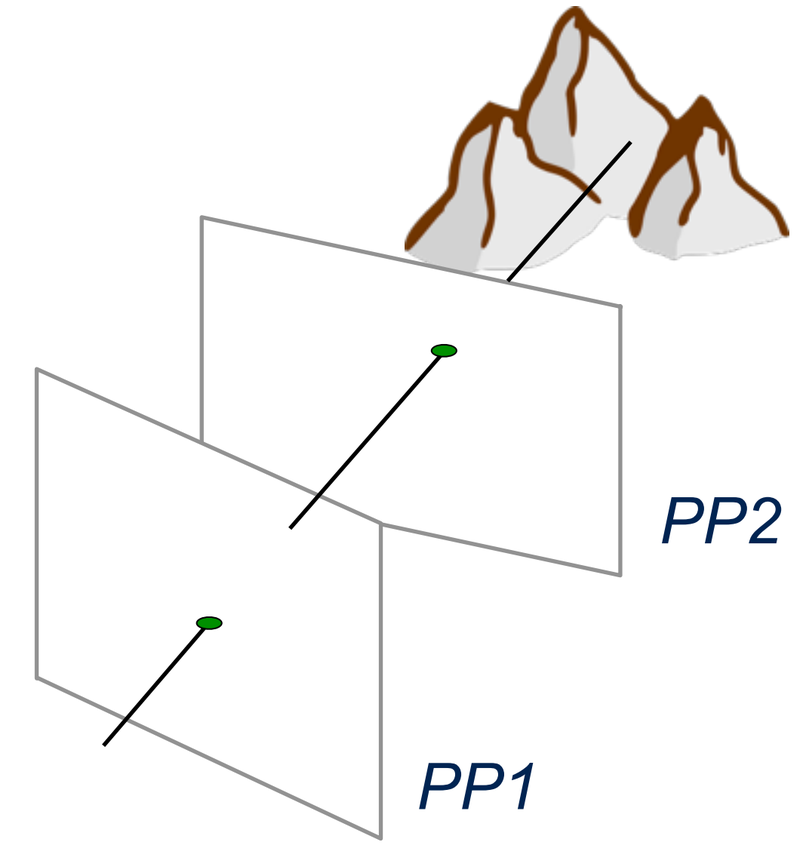


Image Re-Projection (2)

To relate two images from the same camera center and map a pixel from **PP1** to **PP2**:



* Rather than a 3D re-projection,
* Think of it as a 2D image warp from one image to another
* Do not need to know the geometry of the two planes with respect to the eye?

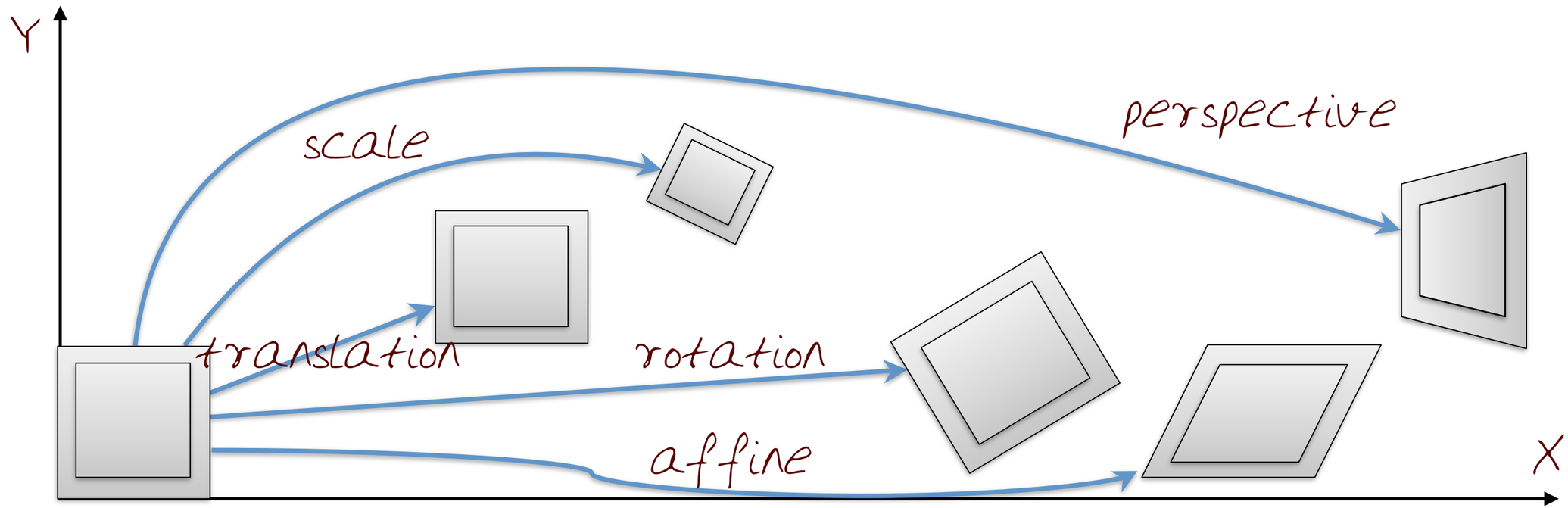
Recall: Image Warping

Which transform is the right one for warping **PP1** into **PP2**?

E.g. translation, Euclidean, affine, projective

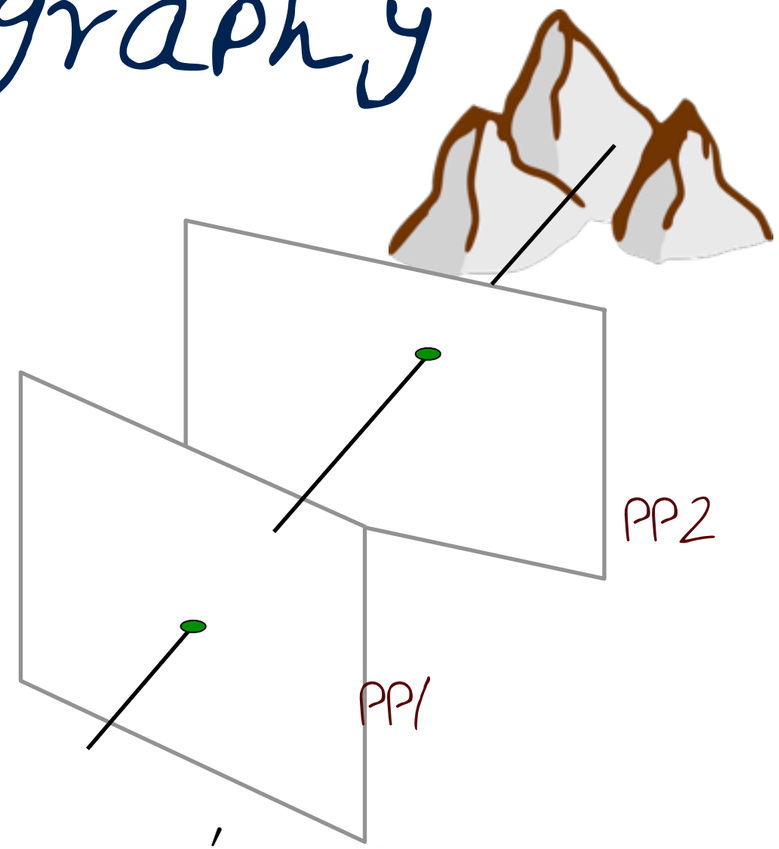
Translation: 2 unknowns, Euclidean: 3 unknowns

Affine: 6 unknowns, Projective: 8 unknowns



Introducing Homography

Relates two images from the same camera center



$$p' = Hp$$

* Rectangle should map to arbitrary quadrilateral
* Parallel lines aren't parallel
* Straight lines must be straight

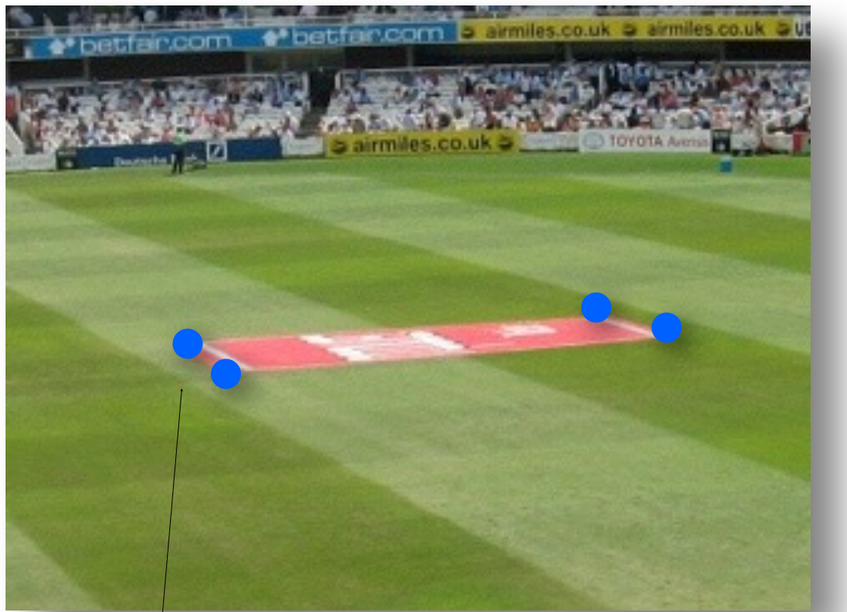
$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

http://en.wikipedia.org/wiki/Homography

Computing Homography

Computing Homography

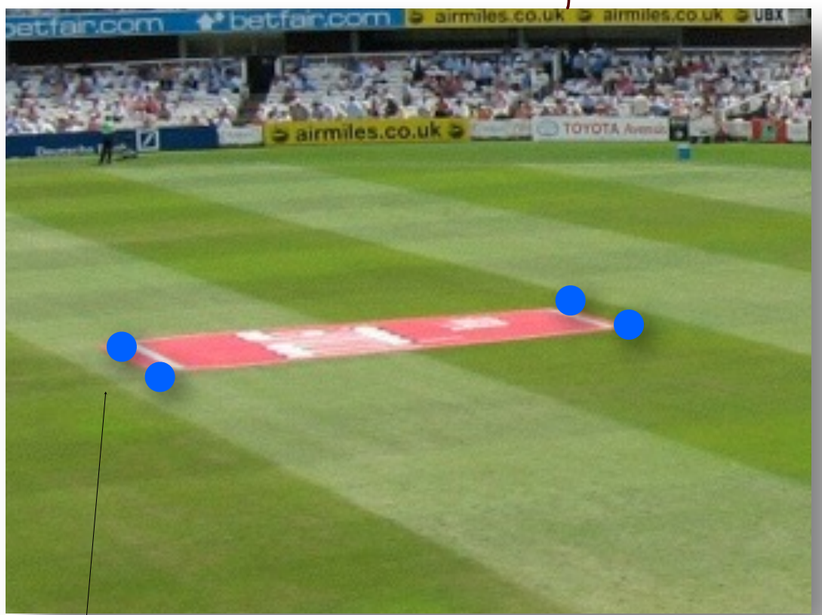
Zoomed



(x, y)

$p_1, p_2, \ldots, p_n$

Zoomed and Left Pan



$(wx^{'}/w, wy^{'}/w) = (x^{'}, y^{'})$

$p_1^{'}, p_2^{'}, \ldots, p_n^{'}$

$$\begin{bmatrix} wx^{'} \\ wy^{'} \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad p^{'} = Hp$$

To compute the homography H, given pairs of corresponding points in two images, we need to set up an equation where the parameters of H are unknowns...

Solving for a Homography

* Set up a system of linear equations:

$$p' = Hp$$

  * $Ah = b$
  * where vector of unknowns

$$h = [a, b, c, d, e, f, g, h]^T$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

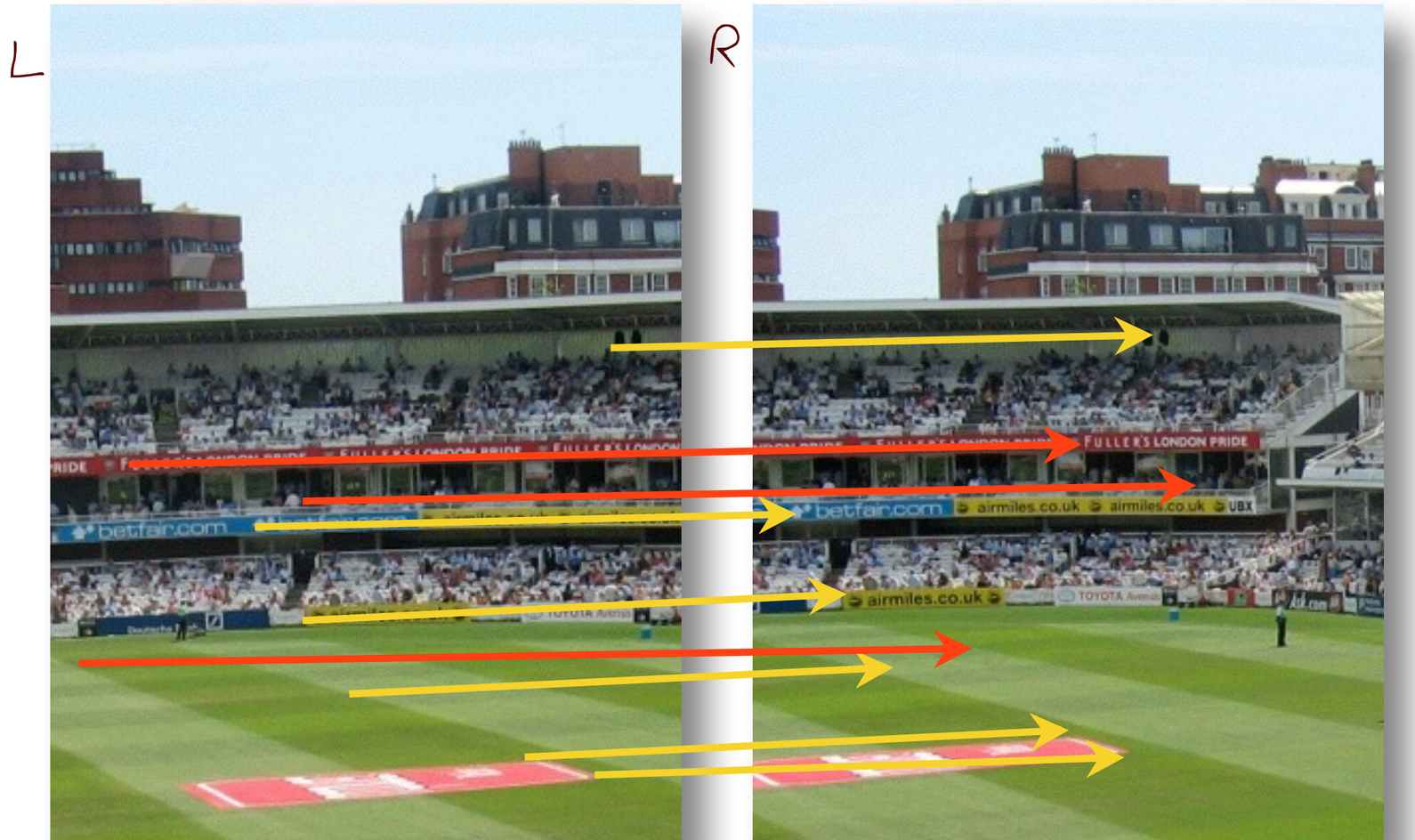
* Need at least 8 equations, but the more the better...
* Solve for h. If over-constrained (i.e. more data), solve using least-squares:

$$\min \|Ah - b\|^2$$

Warp into a Shared Coordinate Space

Warping and Interpolation

Dealing with BAD Matches



RANdom SAmple Consensus (RANSAC)

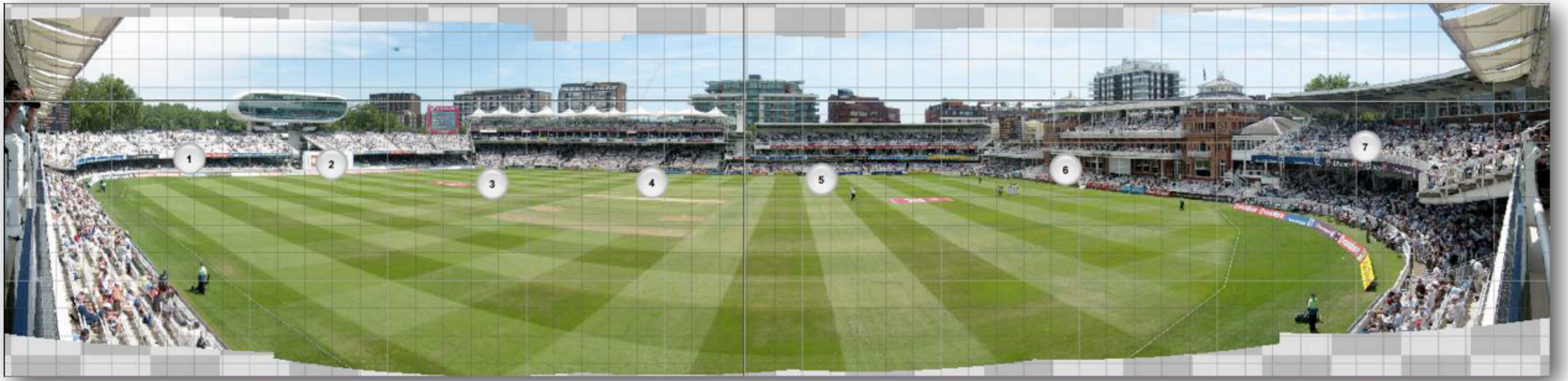
* Select ONE match, Count INLIERS
* Find "average" translation vector

RANdom SAmple Consensus (RANSAC)

Loop till find a convergence/popular $\boldsymbol{H}$:

1. Select four feature pairs (at random)
2. Compute homography $\boldsymbol{H}$ (exact)
3. Compute inliers where: $SSD(p_{in}', \boldsymbol{H}\, p_{in}) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares $\boldsymbol{H}$ estimate on all of the inliers

Key idea: Not that there are more inliers than outliers, but that the outliers are wrong in different ways.



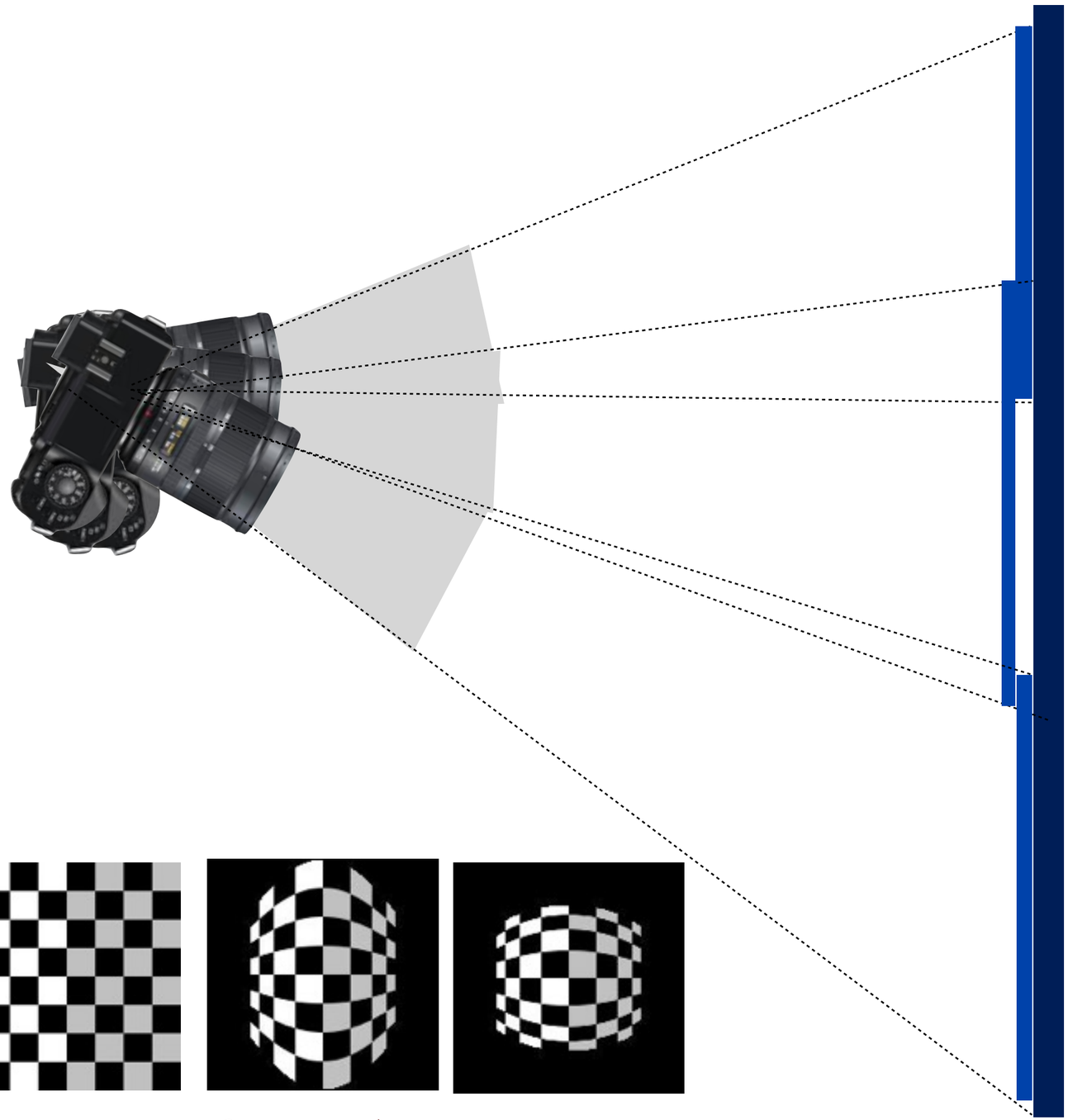
Using kolor autopano giga™ v3

DEMO

```
import numpy as np
import cv2

# Read images
img1 = cv2.imread("einstein.png")  # left image
img2 = cv2.imread("davinci.png")  # right image
print "Image 1 size: {}x{}".format(img1.shape[1], img1.shape[0])
print "Image 2 size: {}x{}".format(img2.shape[1], img2.shape[0])
cv2.imshow("Image 1", img1)
cv2.imshow("Image 2", img2)

# Convert to grayscale
img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# Initialize ORB detector object
orb = cv2.ORB()  # or cv2.SIFT() in OpenCV 2.4.9+

# Find keypoints, compute descriptors and show them on original images (with scale and
orientation)
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)
print "Image 1: {} keypoints found".format(len(kp1))
print "Image 2: {} keypoints found".format(len(kp2))
img1_kp = cv2.drawKeypoints(img1, kp1, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_kp = cv2.drawKeypoints(img2, kp2, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow("Image 1: Keypoints", img1_kp)
cv2.imshow("Image 2: Keypoints", img2_kp)
```

```
# Create two sequences of corresponding (matched) points
pts1 = []
pts2 = []
for mat in matches:
    # Get the matching keypoints for each of the images
    # Note: We could get rid of weak matches (large distance) here, but RANSAC can handle them
    pts1.append(kp1[mat.queryIdx].pt)
    pts2.append(kp2[mat.trainIdx].pt)

pts1 = np.asarray(pts1, dtype=np.float32)
pts2 = np.asarray(pts2, dtype=np.float32)

# Compute homography (since our objects are planar) using RANSAC to reject outliers
M_hom, inliers = cv2.findHomography(pts2, pts1, cv2.RANSAC)  # transform img2 to img1's space
print "Computed homography (perspective transform matrix):"
print M_hom

# Create panorama
pano_size = (int(M_hom[0, 2] + img2.shape[1]), max(img1.shape[0], img2.shape[0]))  # combined size
img_pano = cv2.warpPerspective(img2, M_hom, pano_size)  # apply transform to img2
img_pano[0:img1.shape[0], 0:img1.shape[1], :] = img1  # copy in pixels from img1

# Note: This will produce a result with visible seams; better: blend/cut/fade
cv2.imshow("Panorama", img_pano)
```

Demo



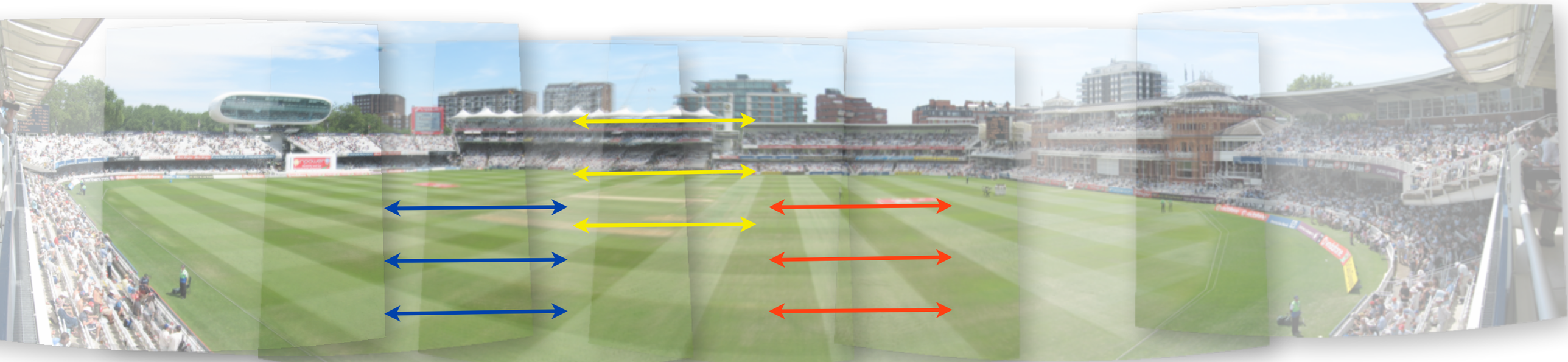
Not Just a Plane

Projection Plane

* Cylinder
* Sphere

Plain Cylinder Sphere

Planar, Spherical, Cylindrical Panoramas



"Finding Panoramas"

Using RANSAC and related matching techniques, we can find images next to each other that form a panorama. So we don't have to take pictures in a sequence.

Brown and Lowe (2003).

Summary

* Five (5) steps to generate a Panorama
* Image Re-projection for Panoramas
* Homography and how it is computed from a pair of images
* RANSAC as an approach to deal with Bad matches across images
* Additional considerations for Panoramas

Further Reading

* Brown and Lowe (2003). "Recognising Panoramas." International Conference on Computer Vision (ICCV2003) (pdf | bib | ppt)
* Microsoft Research Image Composite Editior (ICE)
* Panorama Tools Graphical User Interface (PTGui)
* Hugin Panorama Photo Stitcher

Computational Photography

* Study the basics of computation and its impact on the entire workflow of photography, from capturing, manipulating and collaborating on, and sharing photographs.

Georgia Tech College of Computing