Image stitching

IN4393 – Computer Vision

Image stitching

- We are given a bunch of photographs; how do we *stitch* them together?

- But first, we need to understand *non-linear least squares* and *homographies*

Non-linear least squares problems

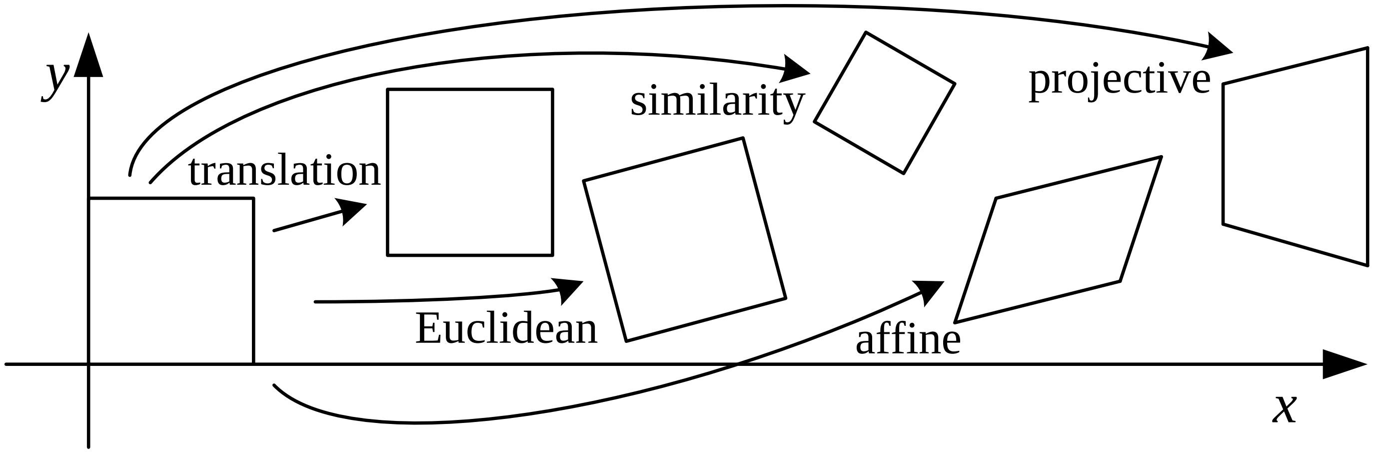
Non-linear least squares

- Fitting a panography was *linear* in the transformation parameters:

$$E = \sum_i \|f(\mathbf{x}_i;\mathbf{p}) - \mathbf{x}'_i\|^2 = \sum_i \|\mathbf{x}_i + J(\mathbf{x}_i)\mathbf{p} - \mathbf{x}'_i\|^2$$

$$= \sum_i \|J(\mathbf{x}_i)\mathbf{p} - \Delta\mathbf{x}_i\|^2$$

- For more complex motion models, the transformation is *non-linear*:

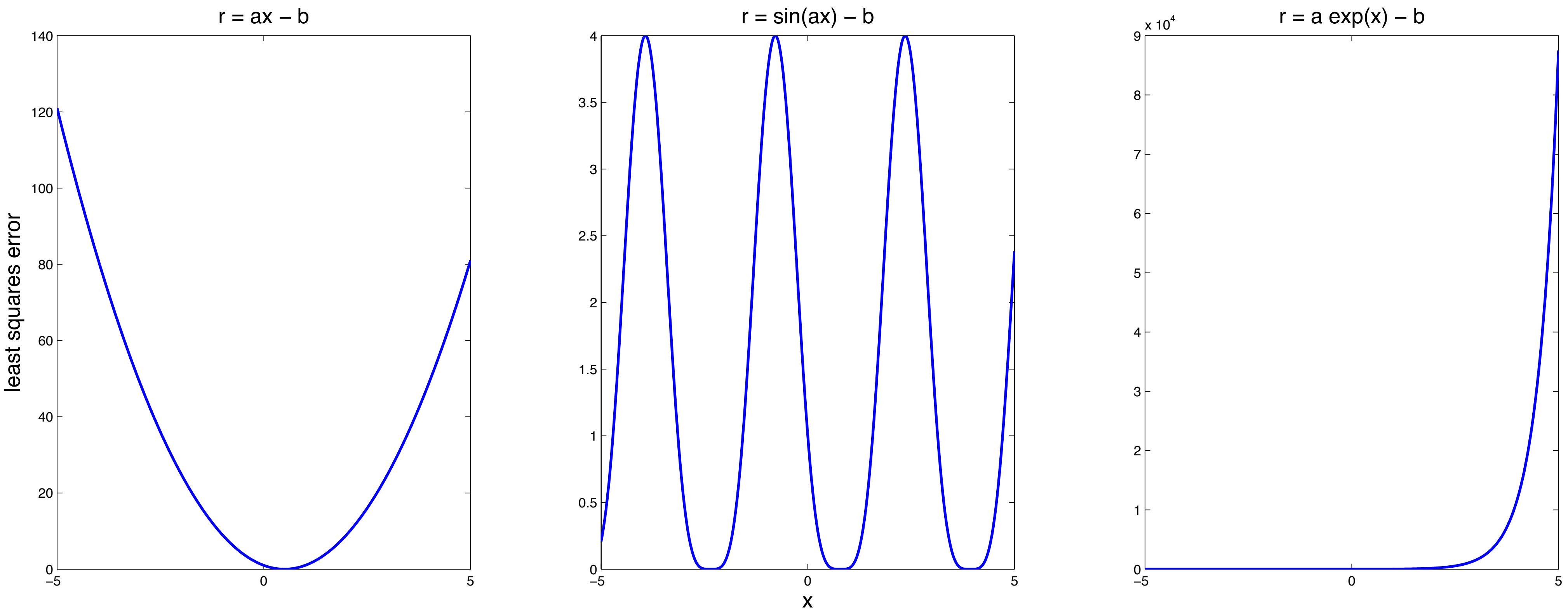


Non-linear least squares

- Consider the *non-linear least squares problem*:

$$g(\mathbf{x}) = \|f(\mathbf{x}; \mathbf{A}) - \mathbf{b}\|^2$$

- This problem is in general not *convex*; it may have multiple *local minima*:

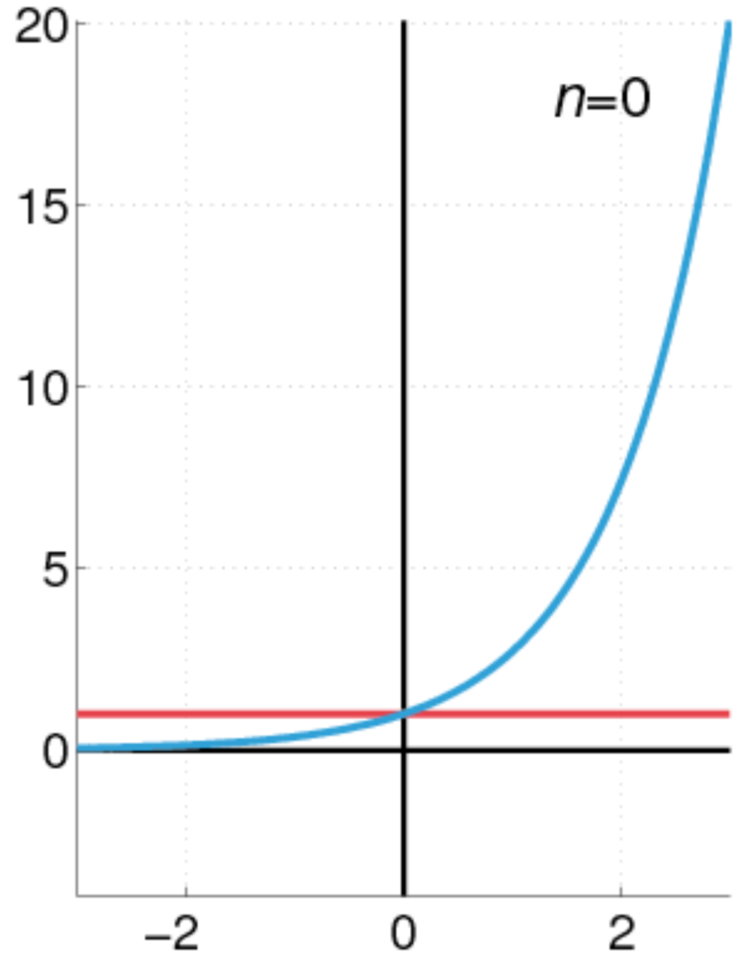


Taylor expansion

- The *Taylor expansion* of the function $f(x)$ around a is given by:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

- Herein, $f^{(n)}$ denotes the *n*-th derivative



Gauss-Newton method

- Iteratively find parameter updates $\Delta\mathbf{x}$

Gauss-Newton method

- Iteratively find parameter updates $\Delta\mathbf{x}$
- Perform a *first-order Taylor expansion* of the residual around the current $\mathbf{x}$:

$$\|f(\mathbf{x} - \Delta\mathbf{x}; \mathbf{A}) - \mathbf{b}\|^2 \approx \|f(\mathbf{x}; \mathbf{A}) + J(\mathbf{x})\Delta\mathbf{x} - \mathbf{b}\|^2$$

Jacobian

parameter update objective

approximation around *x*

Gauss-Newton method

- Iteratively find parameter updates $\Delta\mathbf{x}$

- Perform a *first-order Taylor expansion* of the residual around the current $\mathbf{x}$:

$$\underbrace{\|f(\mathbf{x} - \Delta\mathbf{x}; \mathbf{A}) - \mathbf{b}\|^2}_{\text{parameter update objective}} \approx \|\underbrace{f(\mathbf{x}; \mathbf{A})}_{\text{approximation around } x} + \underbrace{J(\mathbf{x})}_{\text{Jacobian}}\Delta\mathbf{x} - \mathbf{b}\|^2$$

- Note that the resulting residual approximation is linear in $\Delta\mathbf{x}$:

  - The parameter update $\Delta\mathbf{x}$ may be obtained via linear least squares

Gauss-Newton method

- Writing down the linear least-squares solution for $\Delta \mathbf{x}$, we obtain:

$$\Delta \mathbf{x} = \left(J(\mathbf{x})^\top J(\mathbf{x}) \right)^{-1} J(\mathbf{x})^\top r(\mathbf{x})$$

"Gauss-Newton approximation to Hessian"

- Gauss-Newton iteratively performs this update: $\mathbf{x} \leftarrow \mathbf{x} - \Delta \mathbf{x}$

- The Taylor expansion just became inaccurate! So iterate the whole process...

Gauss-Newton method

- Writing down the linear least-squares solution for $\Delta\mathbf{x}$, we obtain:

$$\Delta\mathbf{x} = \left(J(\mathbf{x})^\top J(\mathbf{x})\right)^{-1} J(\mathbf{x})^\top r(\mathbf{x})$$

"Gauss-Newton approximation to Hessian"

- Gauss-Newton iteratively performs this update: $\mathbf{x} \leftarrow \mathbf{x} - \Delta\mathbf{x}$

- The Taylor expansion just became inaccurate! So iterate the whole process...

- To implement, you only need to derive *Jacobian*: $J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$

Newton's method

- Perform a *second-order Taylor expansion* of $g(\mathbf{x})$ around $\mathbf{x}$:

$$g(\mathbf{x}) \approx \|r(\mathbf{x})\|^2 - 2J(\mathbf{x})r(\mathbf{x})\Delta\mathbf{x} + \left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^2\right](\Delta\mathbf{x})^2$$

with residuals: $r(\mathbf{x}) = f(\mathbf{x}; \mathbf{A}) - \mathbf{b}$

Newton's method

- Perform a *second-order Taylor expansion* of $g(\mathbf{x})$ around $\mathbf{x}$:

$$g(\mathbf{x}) \approx \|r(\mathbf{x})\|^2 - 2J(\mathbf{x})r(\mathbf{x})\Delta\mathbf{x} + \left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^2\right](\Delta\mathbf{x})^2$$

with residuals: $r(\mathbf{x}) = f(\mathbf{x}; \mathbf{A}) - \mathbf{b}$

- This looks a lot like a linear least-squares problem; set gradient to zero:

$$-2J(\mathbf{x})^{\mathrm{T}} r(\mathbf{x}) + 2\left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^{\mathrm{T}} J(\mathbf{x})\right]\Delta\mathbf{x} = 0$$

$$\left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^{\mathrm{T}} J(\mathbf{x})\right]\Delta\mathbf{x} = J(\mathbf{x})^{\mathrm{T}} r(\mathbf{x})$$

Newton's method

- Perform a *second-order Taylor expansion* of $g(\mathbf{x})$ around $\mathbf{x}$:

$$g(\mathbf{x}) \approx \|r(\mathbf{x})\|^2 - 2J(\mathbf{x})r(\mathbf{x})\Delta\mathbf{x} + \left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^2\right](\Delta\mathbf{x})^2$$

with residuals: $r(\mathbf{x}) = f(\mathbf{x};\mathbf{A}) - \mathbf{b}$

- This looks a lot like a linear least-squares problem; set gradient to zero:

$$-2J(\mathbf{x})^{\mathrm{T}}r(\mathbf{x}) + 2\left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^{\mathrm{T}}J(\mathbf{x})\right]\Delta\mathbf{x} = 0$$

$$\left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^{\mathrm{T}}J(\mathbf{x})\right]\Delta\mathbf{x} = J(\mathbf{x})^{\mathrm{T}}r(\mathbf{x})$$

- Note the similarity of the Newton update with the Gauss-Newton update:

$$\Delta\mathbf{x} = \left[H(\mathbf{x})r(\mathbf{x}) + J(\mathbf{x})^{\mathrm{T}}J(\mathbf{x})\right]^{-1} J(\mathbf{x})^{\mathrm{T}}r(\mathbf{x})$$

Homographies

Homogeneous coordinates

- We are used to describing a location in *Cartesian coordinates*:

$$\mathbf{x} = [x \quad y]^{\mathrm{T}} \qquad\qquad \mathbf{x} = [x \quad y \quad z]^{\mathrm{T}}$$

Homogeneous coordinates

- We are used to describing a location in *Cartesian coordinates*:

$$\mathbf{x} = [x \quad y]^{\mathrm{T}} \qquad \mathbf{x} = [x \quad y \quad z]^{\mathrm{T}}$$

- Alternatively, we can describe locations in *homogeneous coordinates*:

$$\tilde{\mathbf{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{w}]^{\mathrm{T}} \qquad \tilde{\mathbf{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{z} \quad \tilde{w}]^{\mathrm{T}}$$

Homogeneous coordinates

- We are used to describing a location in *Cartesian coordinates*:

$$\mathbf{x} = [x \quad y]^{\mathrm{T}} \qquad \mathbf{x} = [x \quad y \quad z]^{\mathrm{T}}$$

- Alternatively, we can describe locations in *homogeneous coordinates*:

$$\tilde{\mathbf{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{w}]^{\mathrm{T}} \qquad \tilde{\mathbf{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{z} \quad \tilde{w}]^{\mathrm{T}}$$

- The corresponding Cartesian coordinates are given by:

$$\mathbf{x} = [\tilde{x}/\tilde{w} \quad \tilde{y}/\tilde{w}]^{\mathrm{T}} \qquad \mathbf{x} = [\tilde{x}/\tilde{w} \quad \tilde{y}/\tilde{w} \quad \tilde{z}/\tilde{w}]^{\mathrm{T}}$$

Homogeneous coordinates

- We are used to describing a location in *Cartesian coordinates*:

$$\mathbf{x} = [x \quad y]^{\mathrm{T}} \qquad\qquad \mathbf{x} = [x \quad y \quad z]^{\mathrm{T}}$$

- Alternatively, we can describe locations in *homogeneous coordinates*:

$$\tilde{\mathbf{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{w}]^{\mathrm{T}} \qquad\qquad \tilde{\mathbf{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{z} \quad \tilde{w}]^{\mathrm{T}}$$

- The corresponding Cartesian coordinates are given by:

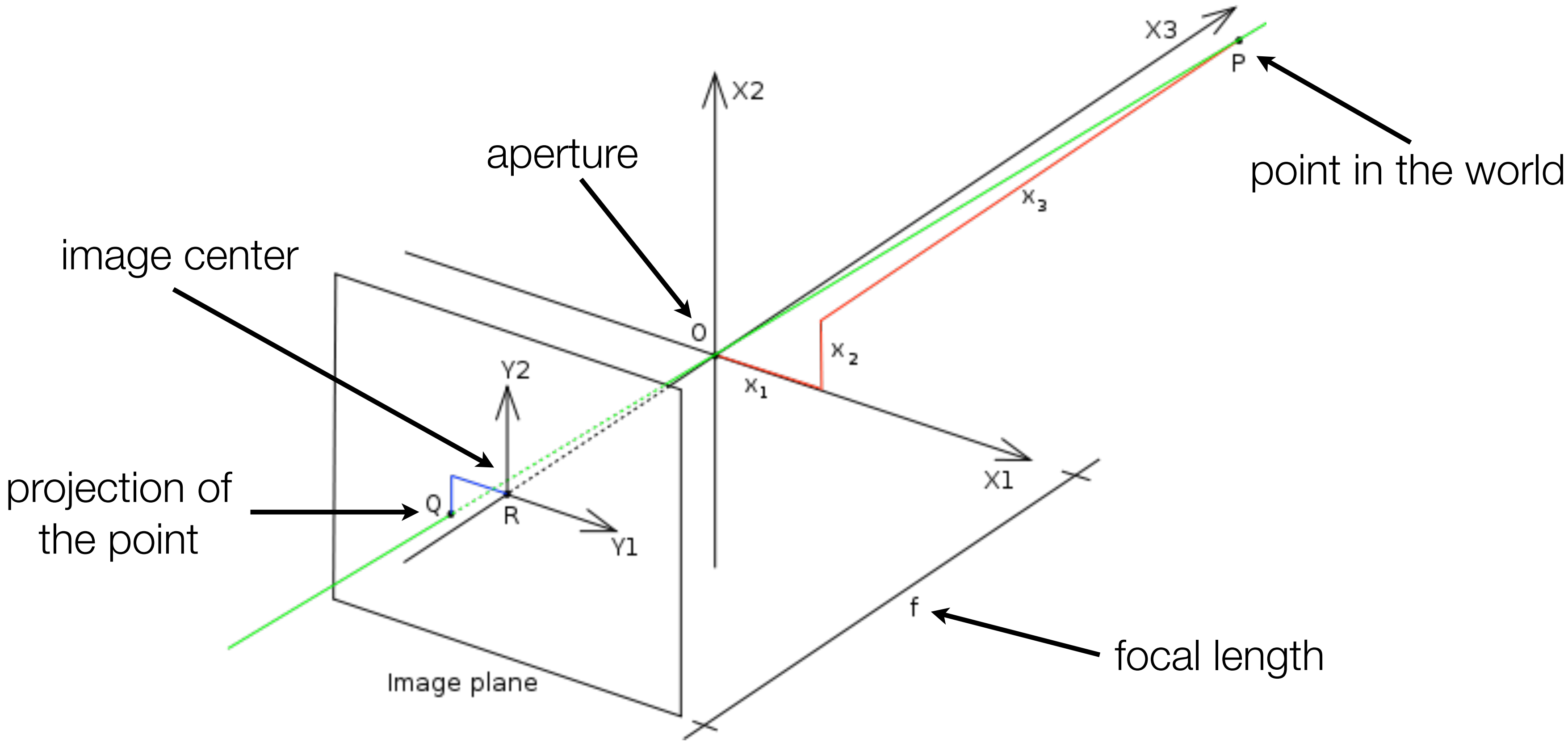
$$\mathbf{x} = [\tilde{x}/\tilde{w} \quad \tilde{y}/\tilde{w}]^{\mathrm{T}} \qquad\qquad \mathbf{x} = [\tilde{x}/\tilde{w} \quad \tilde{y}/\tilde{w} \quad \tilde{z}/\tilde{w}]^{\mathrm{T}}$$

- Essentially, you can think of $\tilde{w}$ as a way to deal with object scale (*"disparity"*)

- Homogeneous coordinates are very useful when working with *perspective transformations* (*homographies*)

Pinhole camera

- *Pinhole camera* is a model for how an ideal camera works:



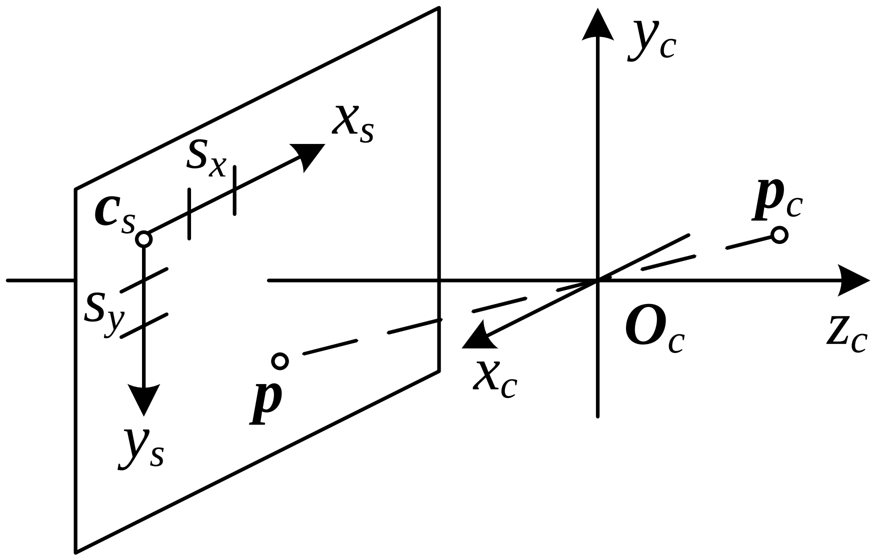
Pinhole camera

- *Focal length* of the camera influences what is captured on the image plane:

- A large focal length implies small *field of view*, and vice versa

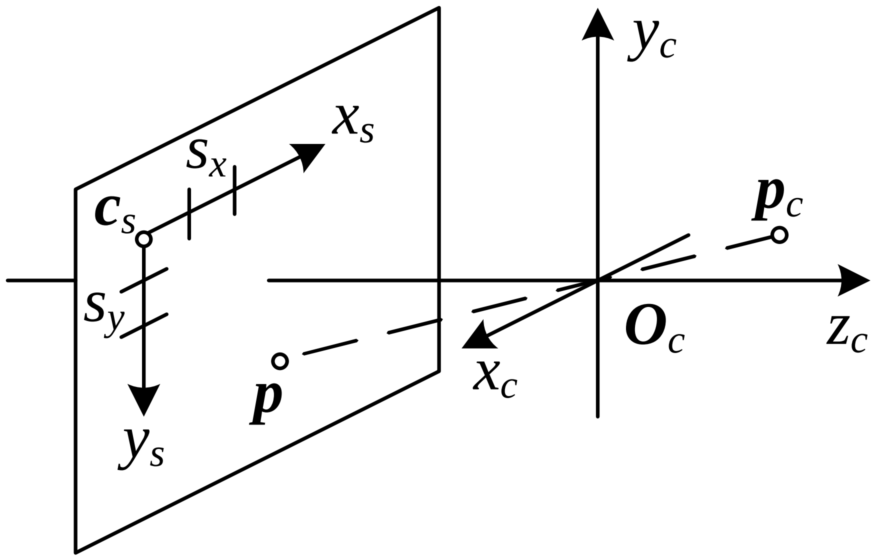
How does a camera see a 3D point?

- Suppose we observe a 2D point, what is the corresponding *3D ray*?



How does a camera see a 3D point?

- Suppose we observe a 2D point, what is the corresponding *3D ray*?



$$\mathbf{p} = \left[\mathbf{R}_s | \mathbf{c}_s\right] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \mathbf{M}_s \bar{\mathbf{x}}_s$$

3D orientation of image plane (relative to nodal point)

3D location of image plane (relative to nodal point)

pixel spacing

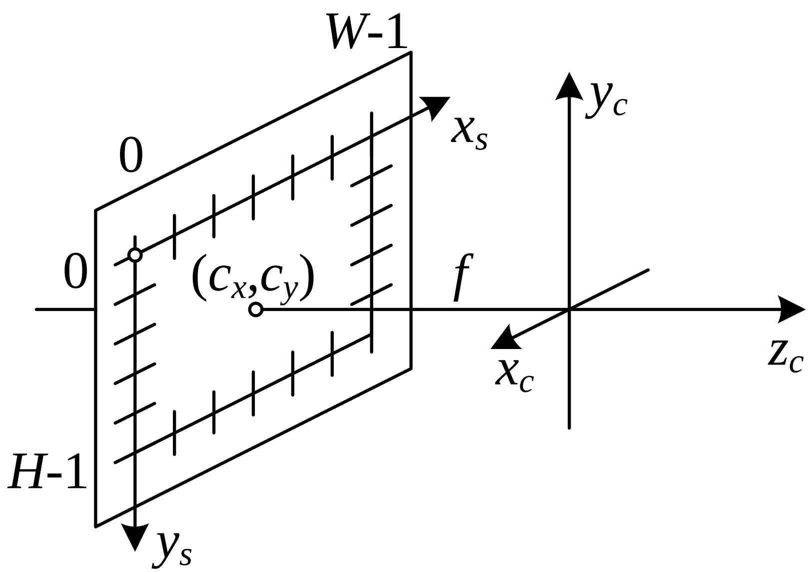
2D pixel (augmented notation)

How does a camera see a 3D point?

- The reverse operation has an unknown scaling (since we don't know the depth)

- For a *camera-centered* point, the 3D-to-2D projection can therefore be written in terms of the *calibration matrix* (note the homogeneous coordinates!):

$$\tilde{\mathbf{x}}_s = \alpha \mathbf{M}_s^{-1} \mathbf{p}_c = \mathbf{K}\mathbf{p}_c \approx \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_c$$



How does a camera see a 3D point?

- The reverse operation has an unknown scaling (since we don't know the depth)
- For a *camera-centered* point, the 3D-to-2D projection can therefore be written in terms of the *calibration matrix* (note the homogeneous coordinates!):

$$\tilde{\mathbf{x}}_s = \alpha \mathbf{M}_s^{-1} \mathbf{p}_c = \mathbf{K} \mathbf{p}_c \approx \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_c$$

- Including 3D rotation and location of the camera, we obtain:

$$\tilde{\mathbf{x}} = \underbrace{\mathbf{K}}_{\substack{\text{camera intrinsics} \\ \text{(calibration matrix)}}} \underbrace{\left[\mathbf{R}|\mathbf{t}\right]}_{\substack{\text{camera extrinsics} \\ \text{(rotation + translation)}}} \mathbf{p} = \underbrace{\mathbf{P}}_{\text{camera matrix}} \mathbf{p}$$

How does a camera see a 3D point?

- The camera matrix is an example of a *projective transformation*:

How do two cameras see a place?

- Assume we have two cameras with projection matrices $\tilde{\mathbf{P}}_0$ and $\tilde{\mathbf{P}}_1$
- Where does camera 1 see the point that camera 0 sees at $\tilde{\mathbf{x}}_0$?

$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{P}}_1 \tilde{\mathbf{P}}_0^{-1} \tilde{\mathbf{x}}_0 = \mathbf{M}_{10} \tilde{\mathbf{x}}_0$$

How do two cameras see a place?

- Assume we have two cameras with projection matrices $\tilde{\mathbf{P}}_0$ and $\tilde{\mathbf{P}}_1$
- Where does camera 1 see the point that camera 0 sees at $\tilde{\mathbf{x}}_0$?

$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{P}}_1 \tilde{\mathbf{P}}_0^{-1} \tilde{\mathbf{x}}_0 = \mathbf{M}_{10} \tilde{\mathbf{x}}_0$$

- *Disparity* of point is irrelevant, so we can take the 3x3 sub-matrix of $\mathbf{M}_{10}$:

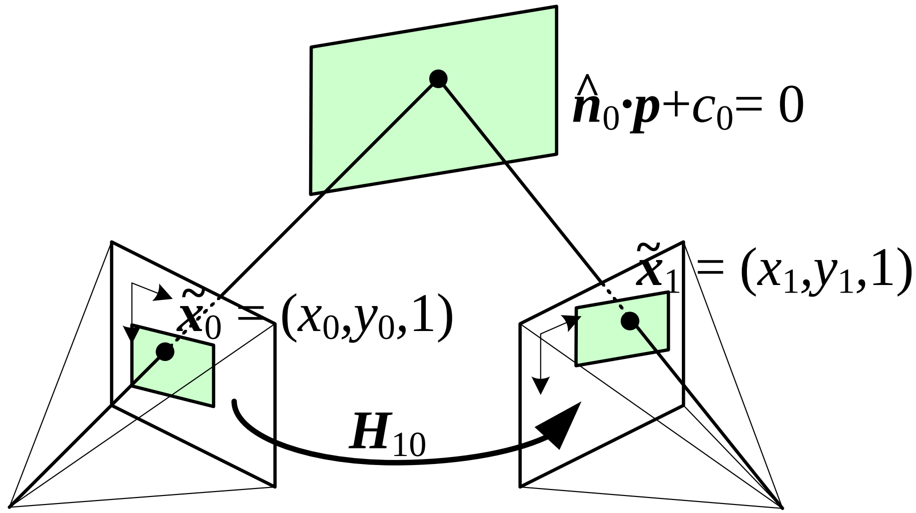
$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{H}}_{10} \tilde{\mathbf{x}}_0$$

- This is known as a *homography* $\tilde{\mathbf{H}}_{10}$ between the two cameras

* Notation in book is a bit sloppy here: page 56

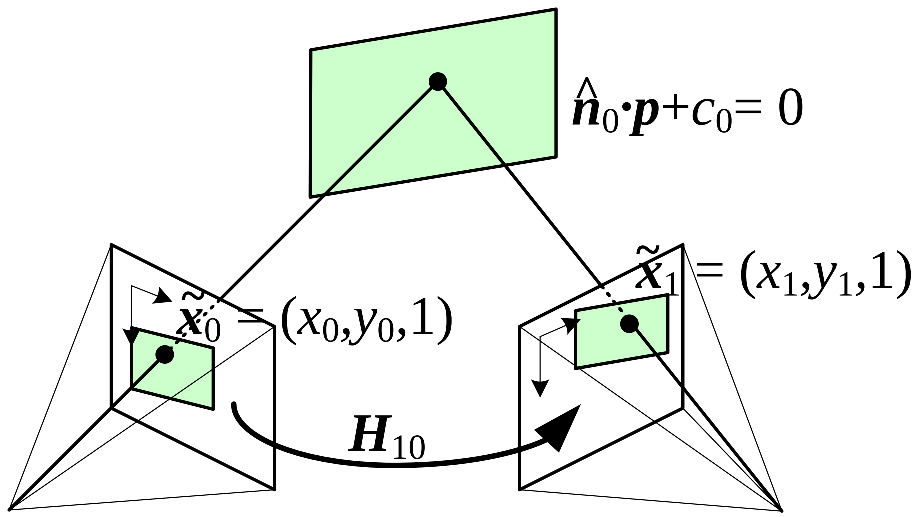
How do two cameras see a place?

- Illustration of homography between two camera (for point and plane):



How do two cameras see a place?

- Illustration of homography between two camera (for point and plane):



$$\boldsymbol{H}_{10} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$$

- In Cartesian coordinates, the homography is given by:

$$x_1 = \frac{h_{00}x_0 + h_{01}y_0 + h_{02}}{h_{20}x_0 + h_{21}y_0 + 1} \qquad y_1 = \frac{h_{10}x_0 + h_{11}y_0 + h_{12}}{h_{20}x_0 + h_{21}y_0 + 1}$$

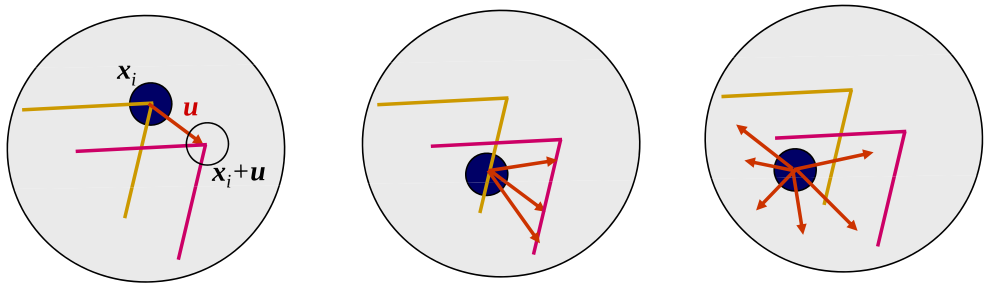
Image stitching

Image stitching

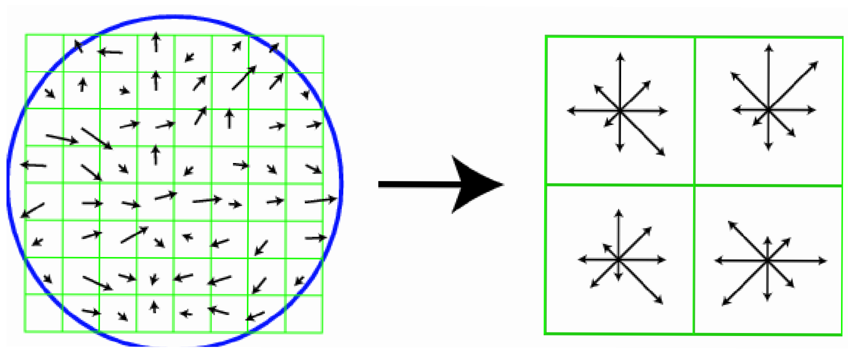
- Stitching algorithms typically have four main ingredients:
  - Method to determine *correspondences* between images
  - Model describing the set of possible *motions* between images (homography)
  - Algorithm to perform *alignment* of the images (bundle adjustment)
  - Algorithm that *composites* the images after alignment (blending; seam finding)

Determining correspondences

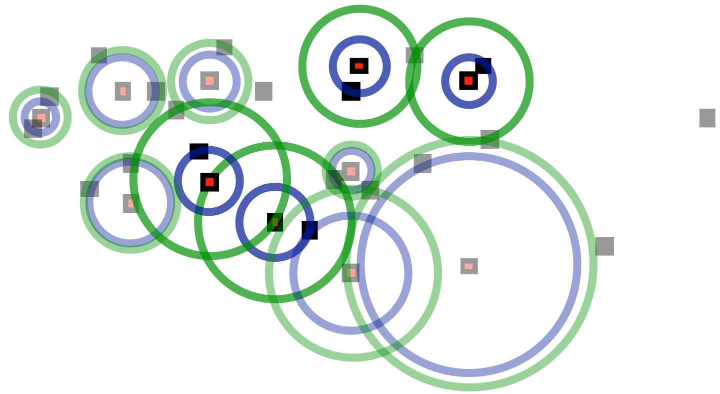
- Feature detection:



- Feature description:



- Feature matching:



Motion model

- We will consider *homographies* because of their generality:

Fitting a homography

- Recall the definition of a homography in Cartesian coordinates:

$$x' = f(x, y) = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \qquad y' = g(x, y) = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + 1}$$

- This makes the alignment objective a non-linear least squares problem:

$$\sum_i \| \begin{bmatrix} f(x_i, y_i; \mathbf{H}) \\ g(x_i, y_i; \mathbf{H}) \end{bmatrix} - \begin{bmatrix} x'_i \\ y'_i \end{bmatrix} \|^2$$

Fitting a homography

- Recall the definition of a homography in Cartesian coordinates:

$$x' = f(x, y) = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \qquad y' = g(x, y) = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + 1}$$

- This makes the alignment objective a non-linear least squares problem:

$$\sum_i \| \begin{bmatrix} f(x_i, y_i; \mathbf{H}) \\ g(x_i, y_i; \mathbf{H}) \end{bmatrix} - \begin{bmatrix} x'_i \\ y'_i \end{bmatrix} \|^2$$

- Simply use Gauss-Newton's (or Newton's) method to solve this problem:
  - You may have to use RANSAC to deal with outliers!

Remark on RANSAC

- The treatment of RANSAC we saw last week was somewhat simplified
- It was not random at all!

Remark on RANSAC

- The treatment of RANSAC we saw last week was somewhat simplified
- It was not random at all! Full RANSAC actually works as follows:

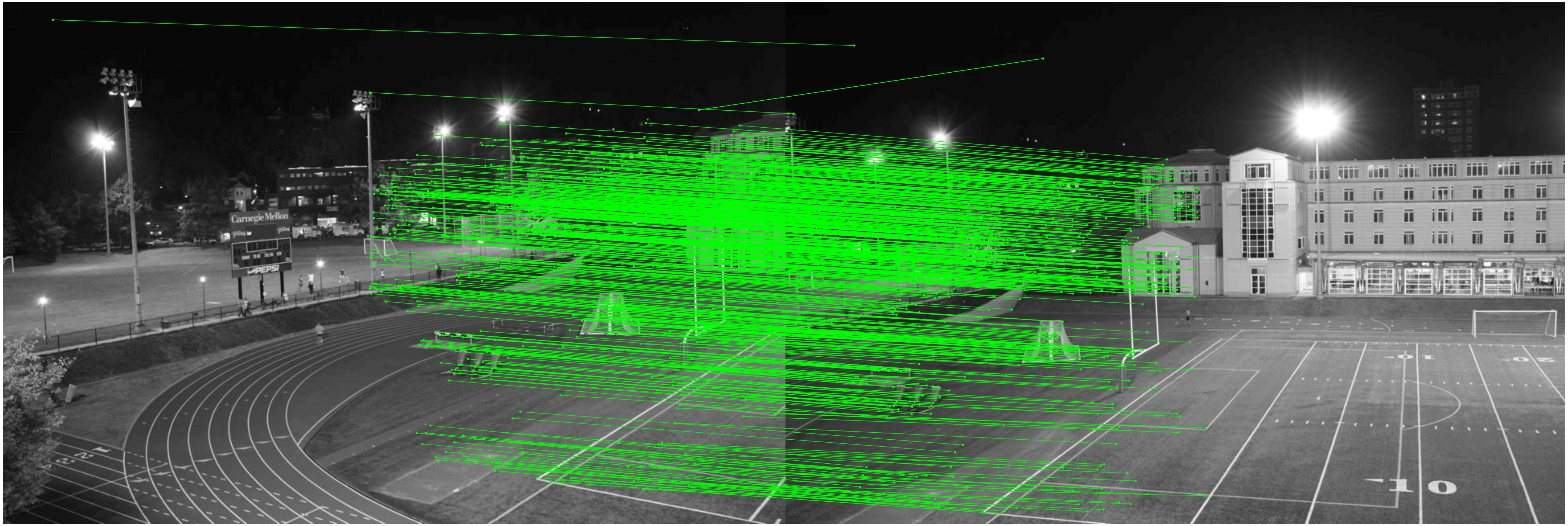
1) Select random subset from 50% of “best” matches as initial inliers
2) Model is fitted to the *hypothetical inliers*
3) Data are tested against the fitted model to determine hypothetical inliers
4) Return to step 2) until sufficient points are classified as inliers (or fixed number of times)
5) Return to step 1) a fixed number of times

Fitting a homography

- Consider the following two images and SIFT matches between them:

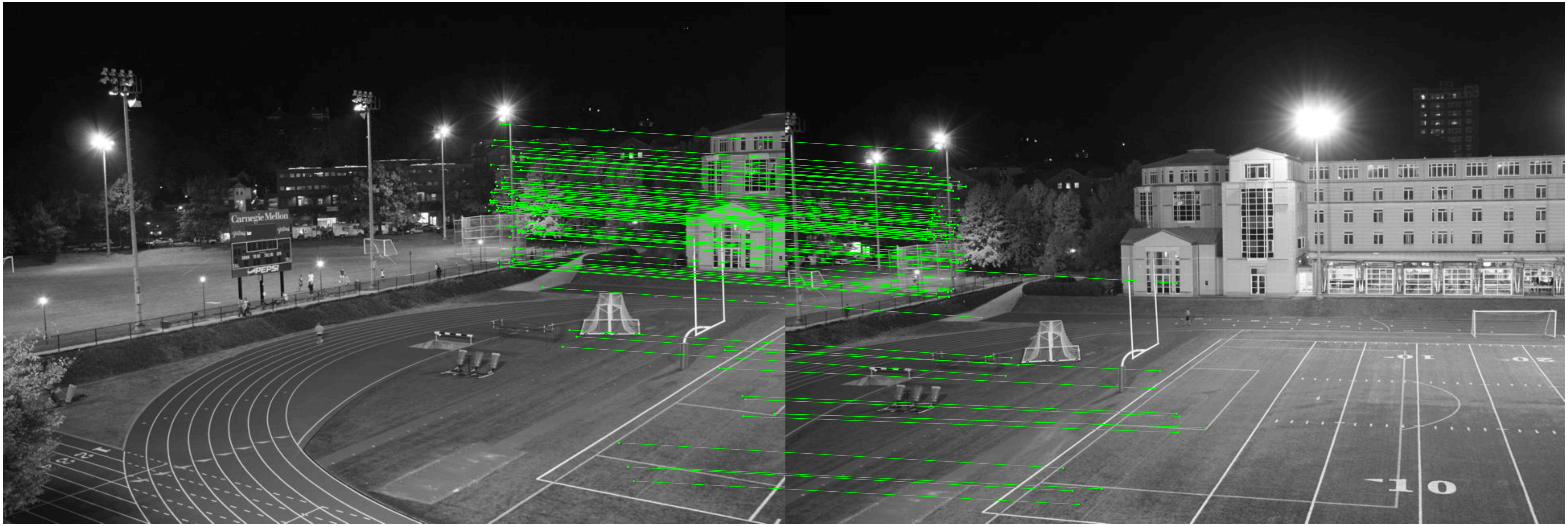
Fitting a homography

- Consider the following two images and SIFT matches between them:



Fitting a homography

- Visualization of the inliers found by RANSAC:



Fitting a homography

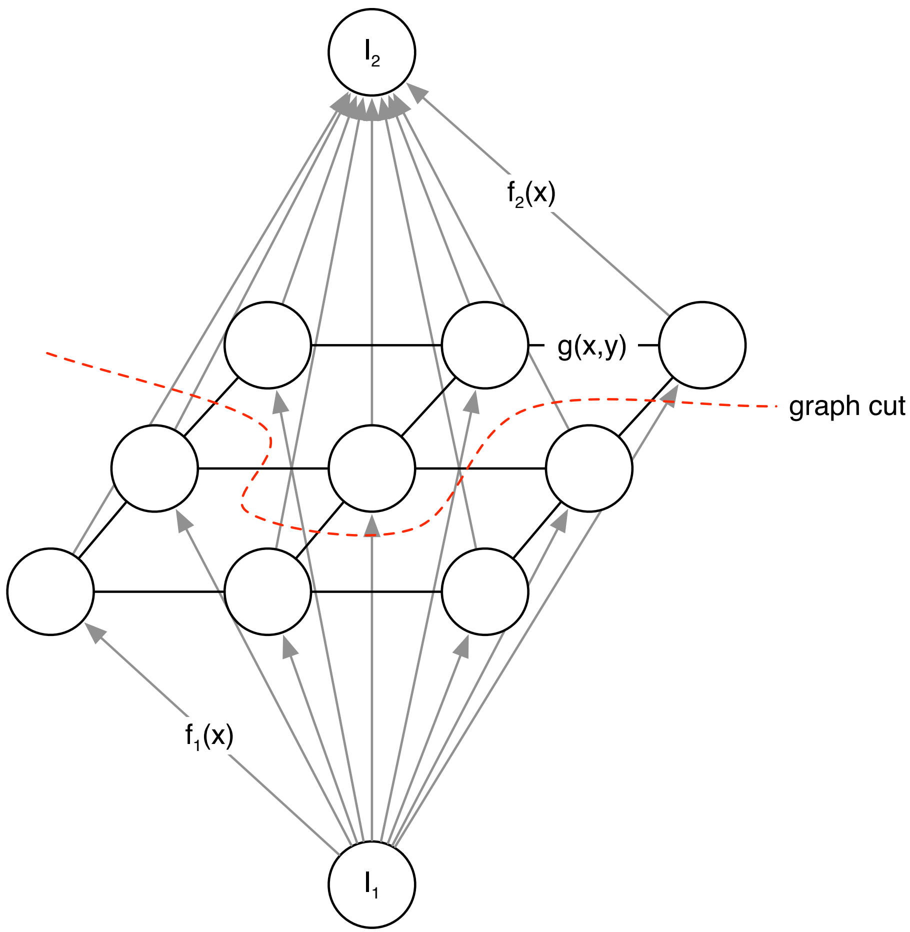
- Visualization of the homography found between the two images:

Alpha blending

- *Alpha blending* averages the images; leads to *“ghosting”* and visible *seams*:

Finding an optimal seam

- We would like to find a *seam* that minimizes the difference between images
- This can be formulated as a *graph min-cut* problem, as follows:



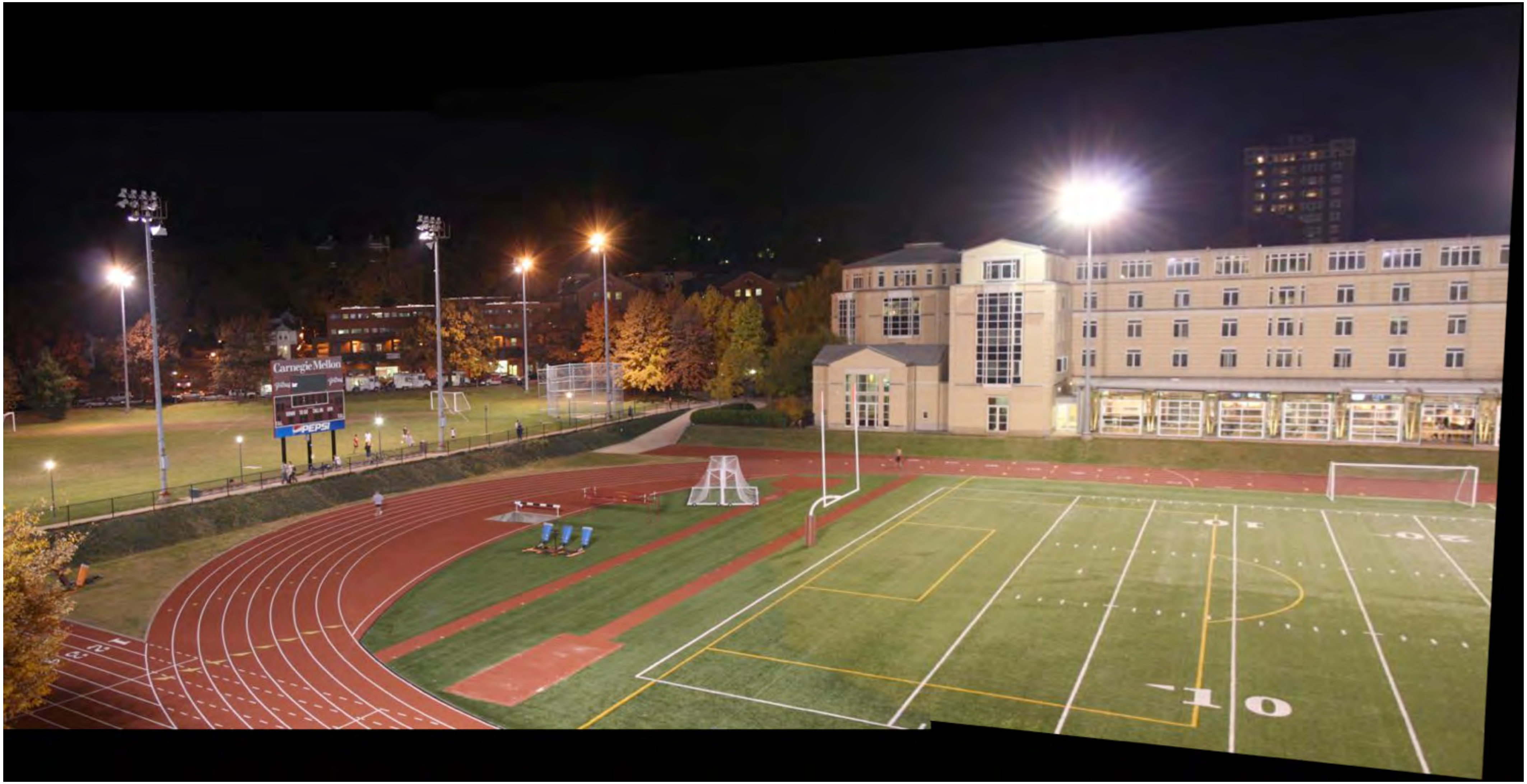
- When you want pixel $\mathbf{x}$ to be taken from I_1, set $f_2(\mathbf{x}) = \infty$ (and vice versa)
- In all other cases: $f_1(\mathbf{x}) = f_2(\mathbf{x}) = 0$
- Set, *e.g.*, $g(x,y) = |I_1(\mathbf{x}) - I_2(\mathbf{y})| + |I_2(\mathbf{x}) - I_1(\mathbf{y})|$
- Efficient graph-cut algorithms exist

Finding an optimal seam

- Ghosting can be prevented by finding a *seam* at which to switch images:

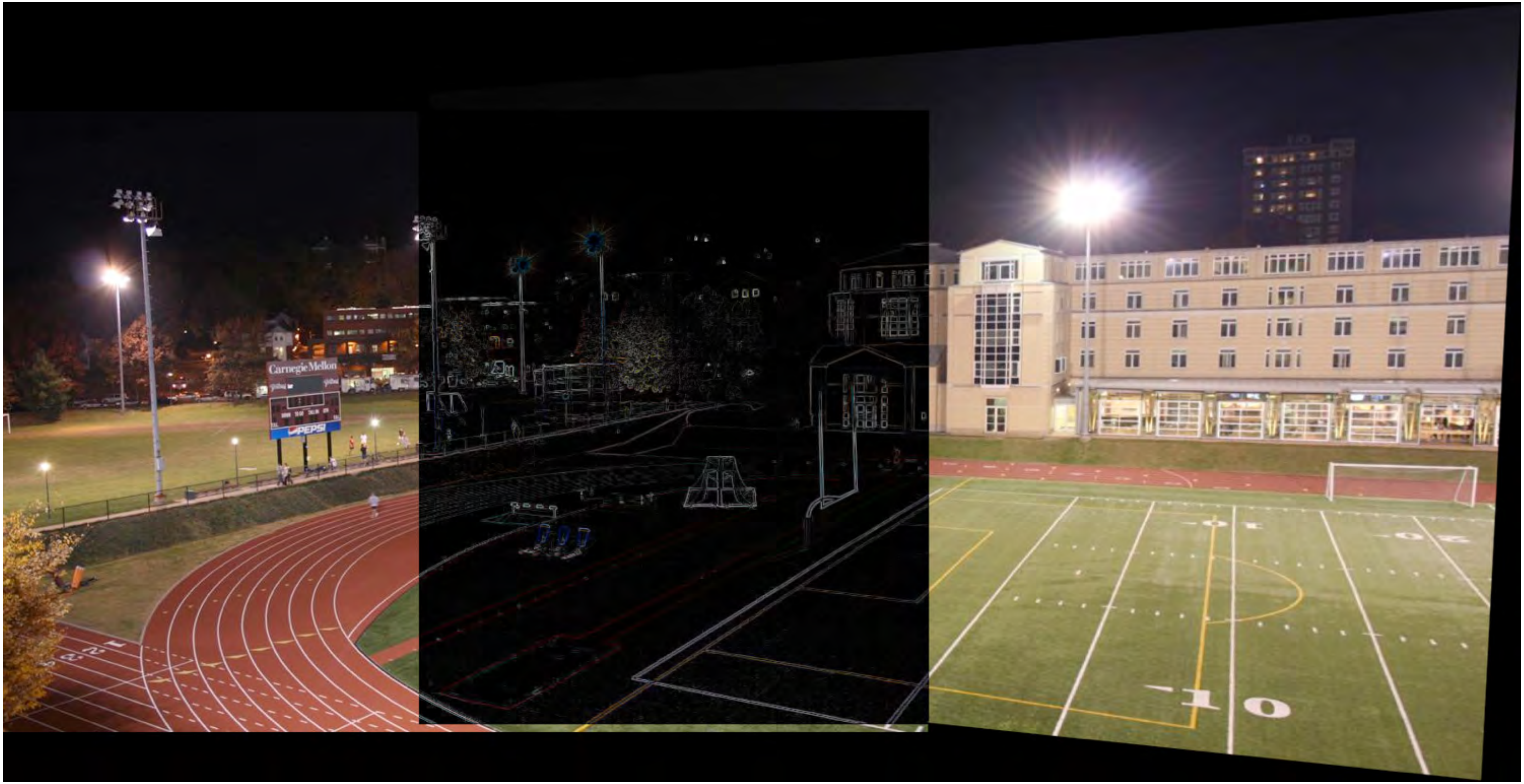
Feathering

- Visibility of seam may be reduced by *“blurring”* weights of both images:



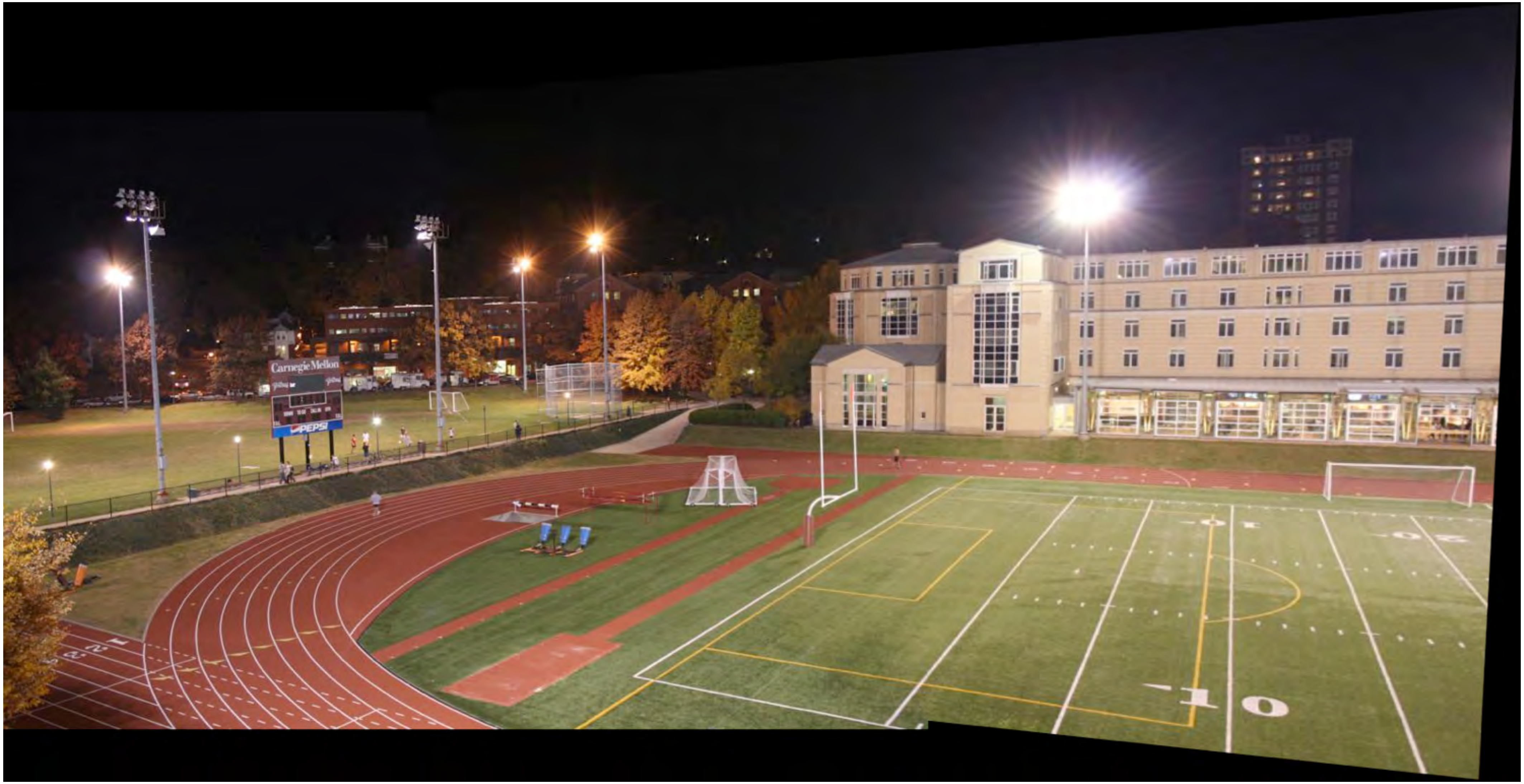
Correcting for exposure differences

- Instead of using seam to select image, we use it to select the *image gradient*:



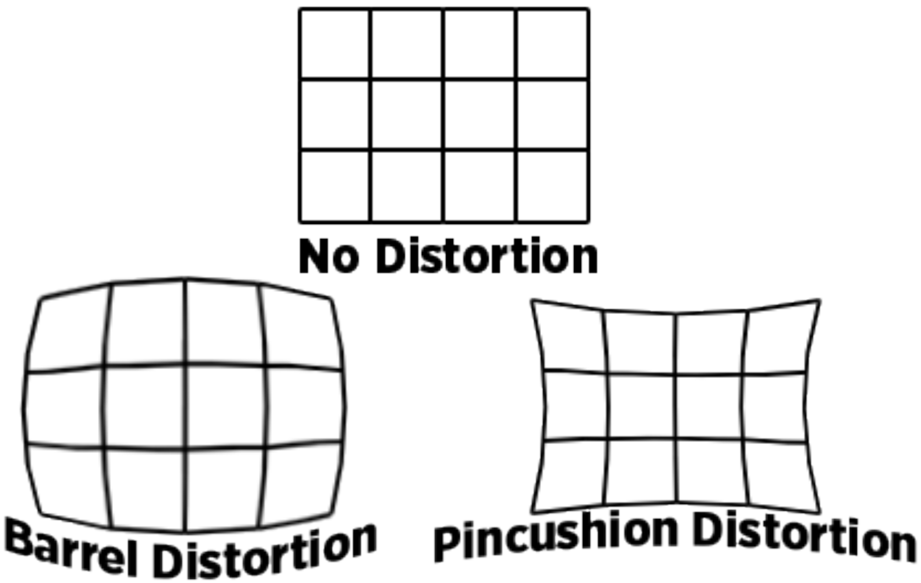
Correcting for exposure differences

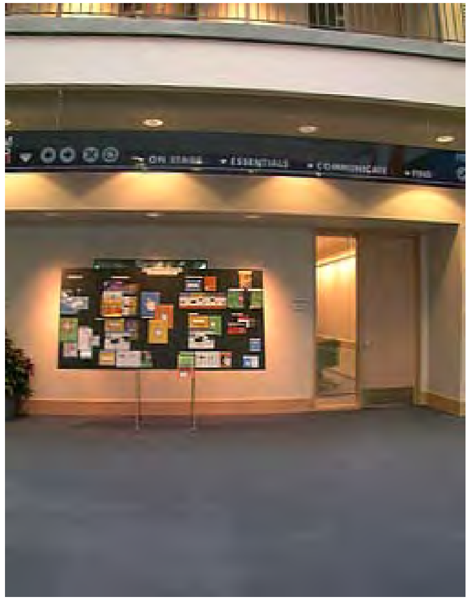
- The stitched image can then be recovered from this gradient:

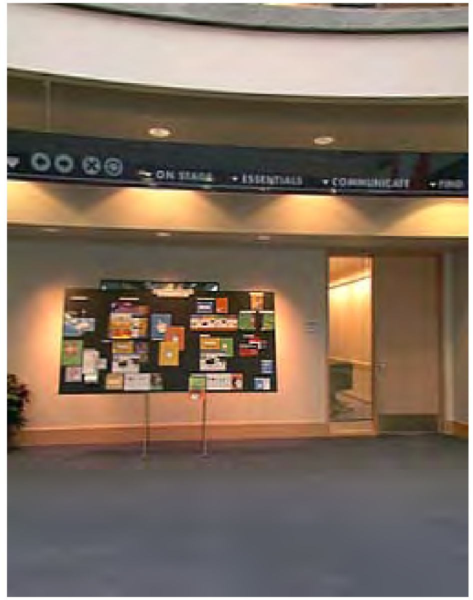


Lens distortion

- Homography model may be too simple when lens distortion is present:







- A common way of dealing with this is by incorporating a distortion model in the objective, and also minimizing w.r.t. the parameters of that model

Other applications of homographies

Reading material: Section 2 and 9