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

Image Transformation

* Transformations applied to images

Lesson Objectives

1. Transform the image
2. Rigid Transformations: Translation, Rotations
3. Affine/Projective Transformation
4. Degrees of Freedom for different transformations

Image Transformations

image filtering:
change range of image

$$g(x) = T(f(x))$$

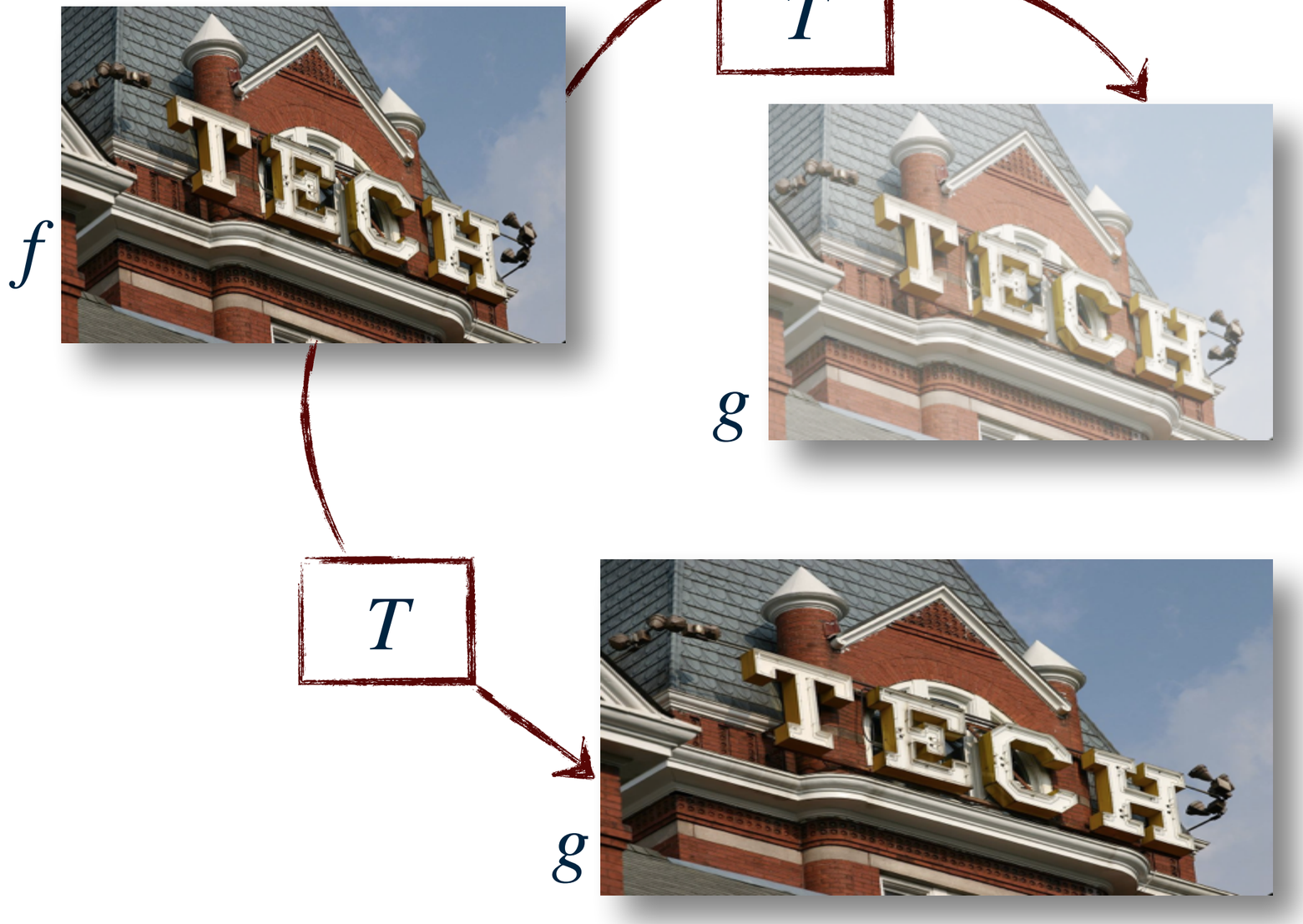


image warping:
change domain of image

$$g(x) = f(T(x))$$

Parametric Global Warping

translation

rotation

Parametric Global Warping

aspect

scale

Parametric Global Warping

affine

perspective

Parametric Global Warping

T

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Transformation T

$$p' = T(p)$$

- A global and parametric T
  - Same for any p
  - A few numbers (parameters)
- As a matrix transform

$$p' = \mathbf{M}p$$

Image Scaling (2D)

* Multiply each components by a scalar
* Uniform scaling: Scalar same for x, y
* Non-uniform: Not same

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Image Transformations

$$\mathbf{M} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

scale around (0,0)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

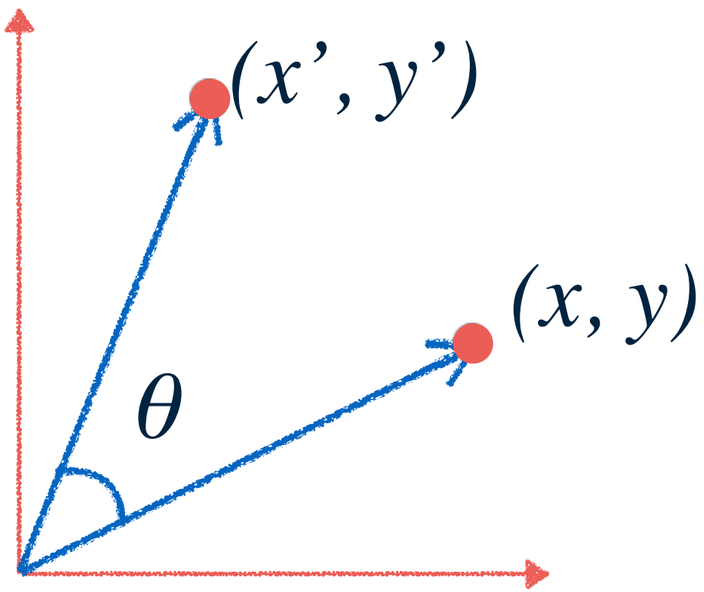
$$\mathbf{M} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

mirror over y-axis

$$\mathbf{M} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix}$$

shear

2D Rotation



$$
\begin{aligned}
x' &= x\cos(\theta) - y\sin(\theta) \\
y' &= x\sin(\theta) + y\cos(\theta)
\end{aligned}
$$

$$
\mathbf{M} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}
$$

2D Linear Transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \Longleftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Only LINEAR COMBINATIONS

Linear transformations are combinations of

Scale,

Rotation,

Shear, and

Mirror

2D Linear Transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \Longleftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Only LINEAR COMBINATIONS

Properties of linear transformations:

Origin maps to origin

Lines map to lines

Parallel lines remain parallel

Ratios are preserved

2D Translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \Longleftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}$$

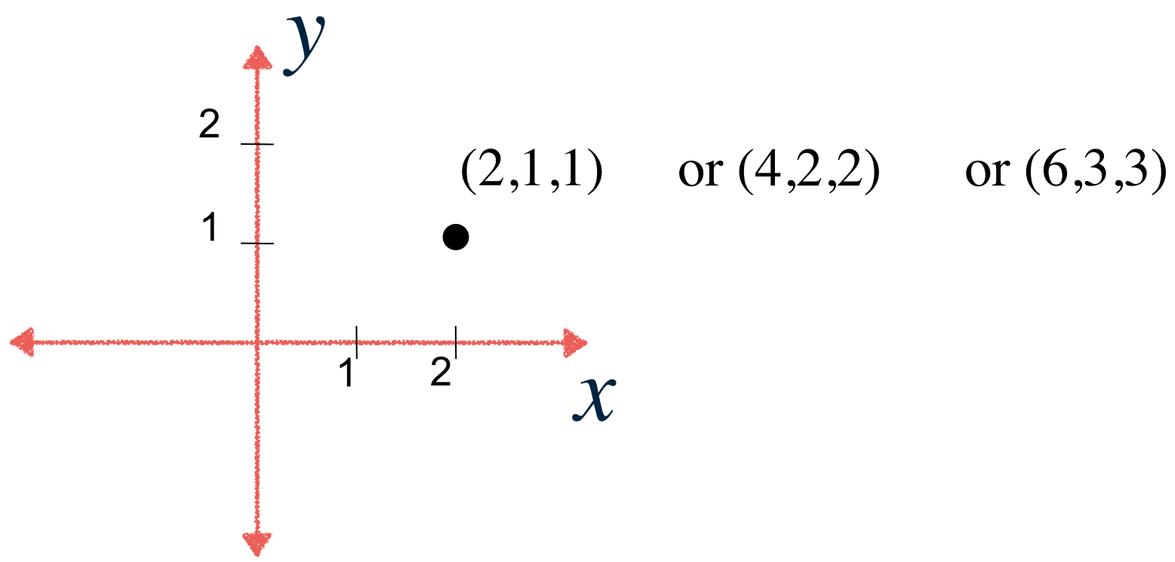
translation in x, y

Homogeneous Coordinates

* Represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

* Add a 3rd coordinate to every 2D point
  * (x, y, w) ⇒ (x/w, y/w)
  * (x, y, 0) ⇒ infinity
  * (0, 0, 0) is not allowed

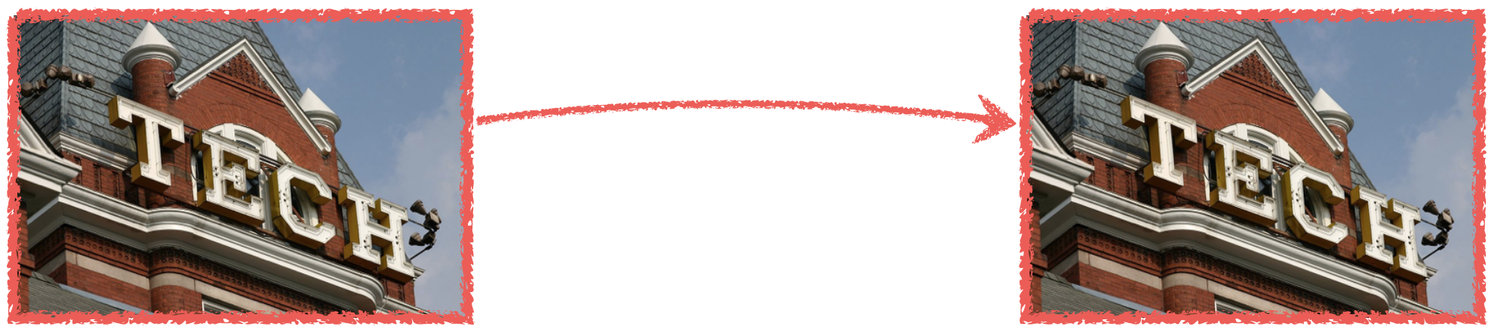


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

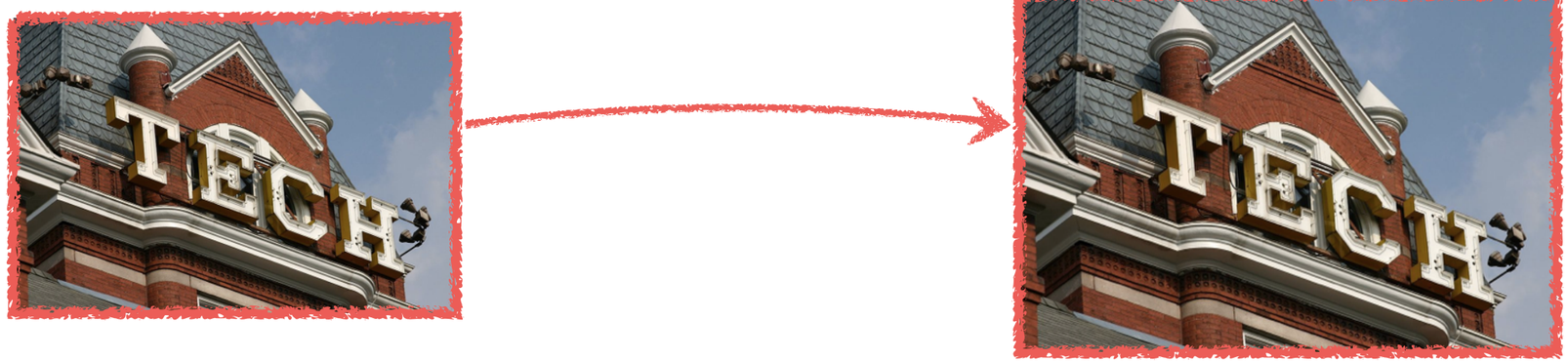


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

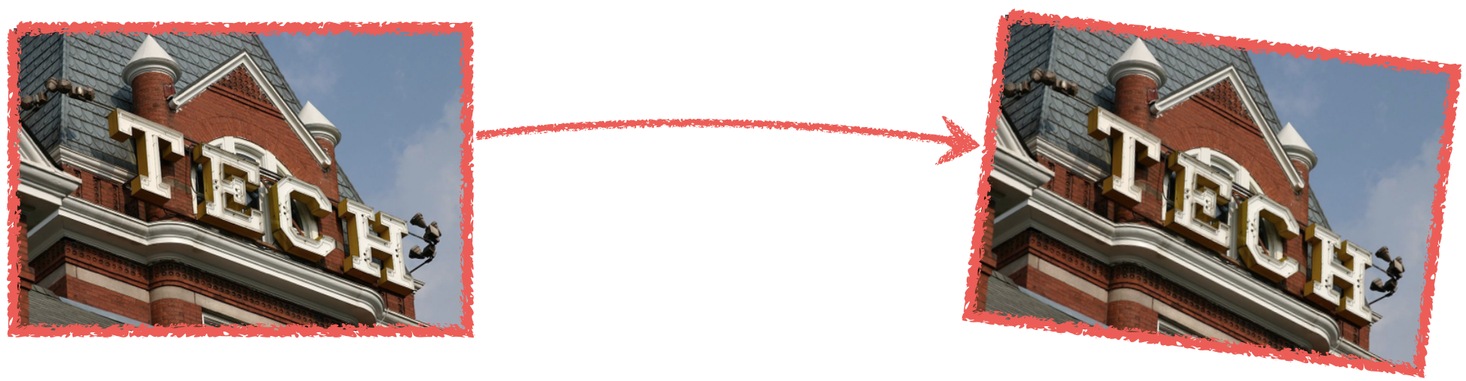


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

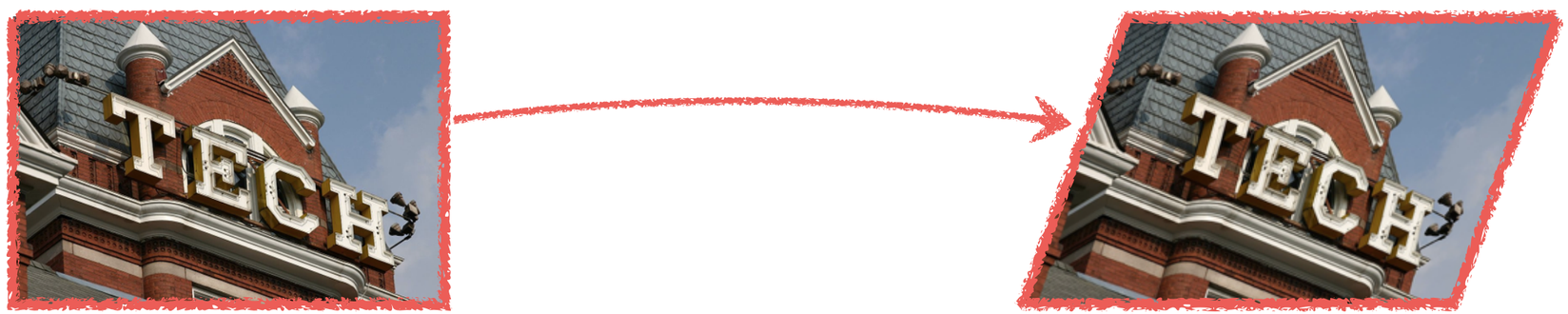


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Shear

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Basic 2D Transformation

Transformations can be combined

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left\{ \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

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

Affine Transformations

- Combines linear transformations, and Translations
- Properties
- Origin does not necessarily map to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved

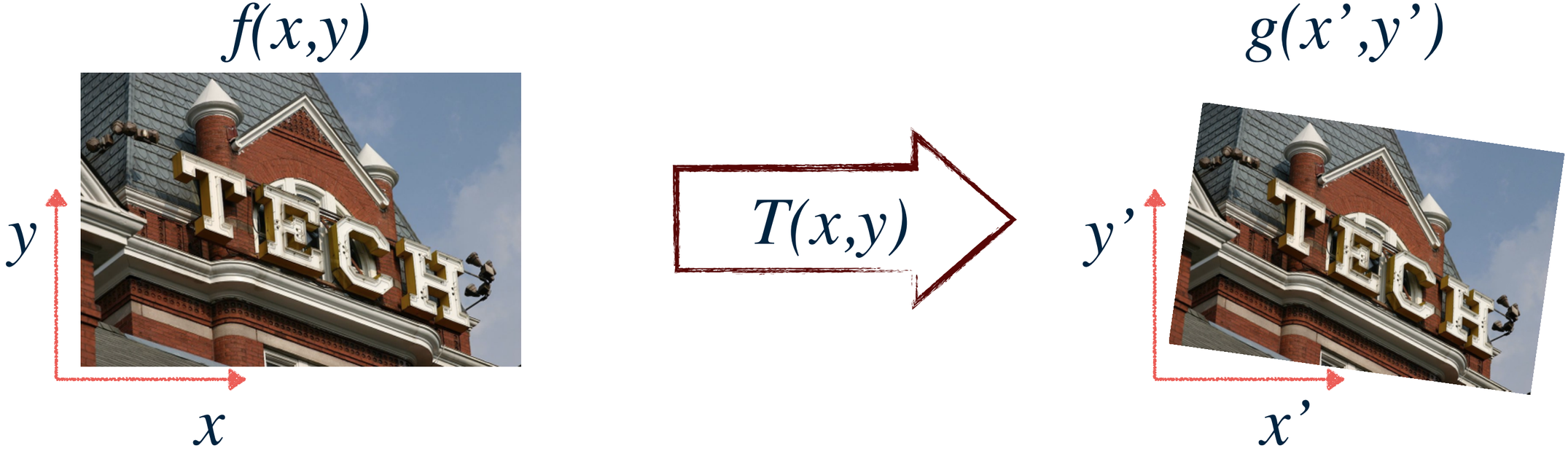
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Projective Transformations

* Combination of Affine transformations, and Projective warps
* Properties:
  * Origin does not necessarily map to origin
  * Lines map to lines
  * Parallel lines do not necessarily remain parallel
  * Ratios are not preserved

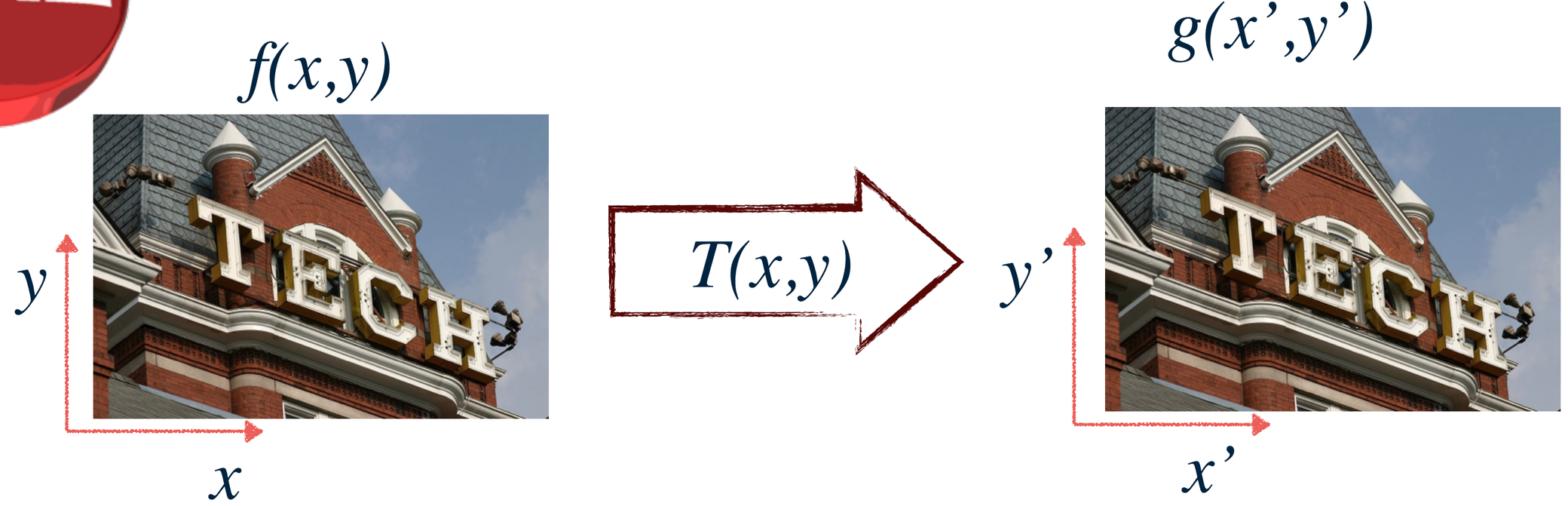
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Recovering Transformations



- What if we know f and g and want to recover the transform T?
- How many points correspondences would we need?
- How many Degrees of Freedom?

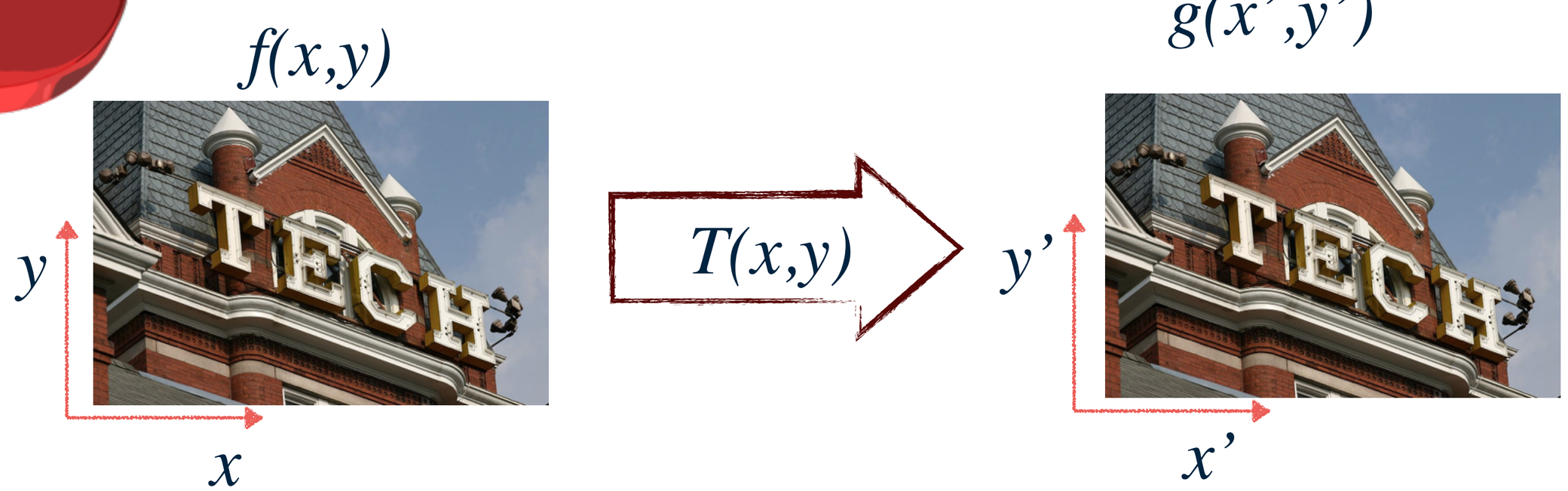
Translation



* How many correspondences needed for translation?

* How many Degrees of Freedom?

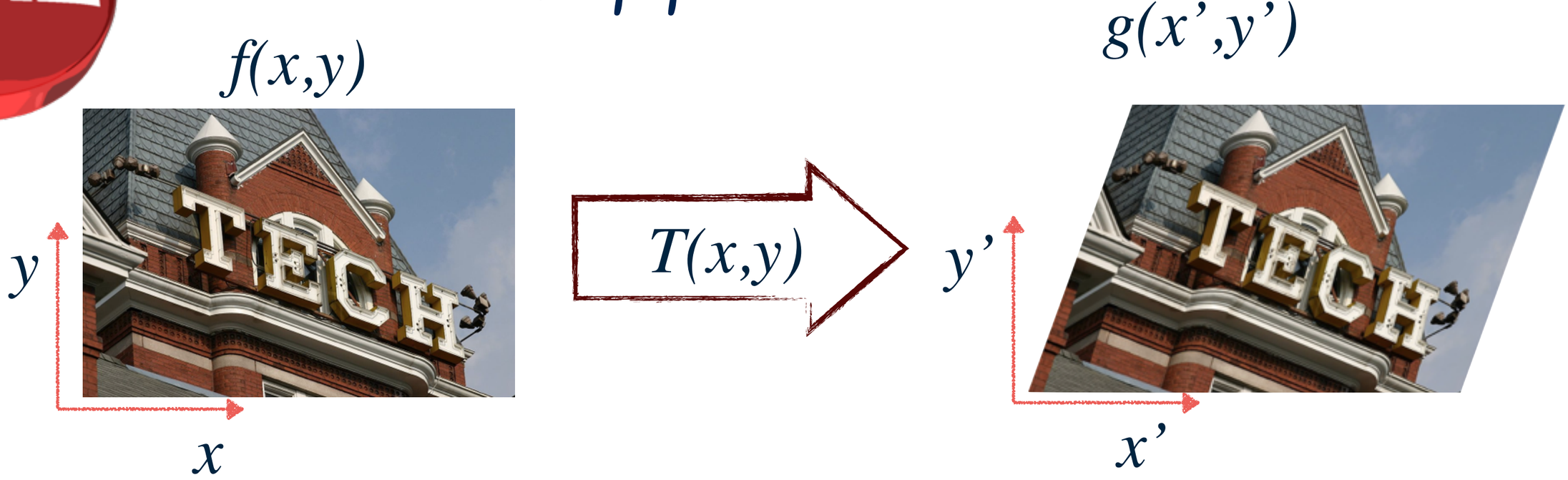
Rotation - Euclidean



* How many correspondences needed for translation?

* How many Degrees of Freedom?

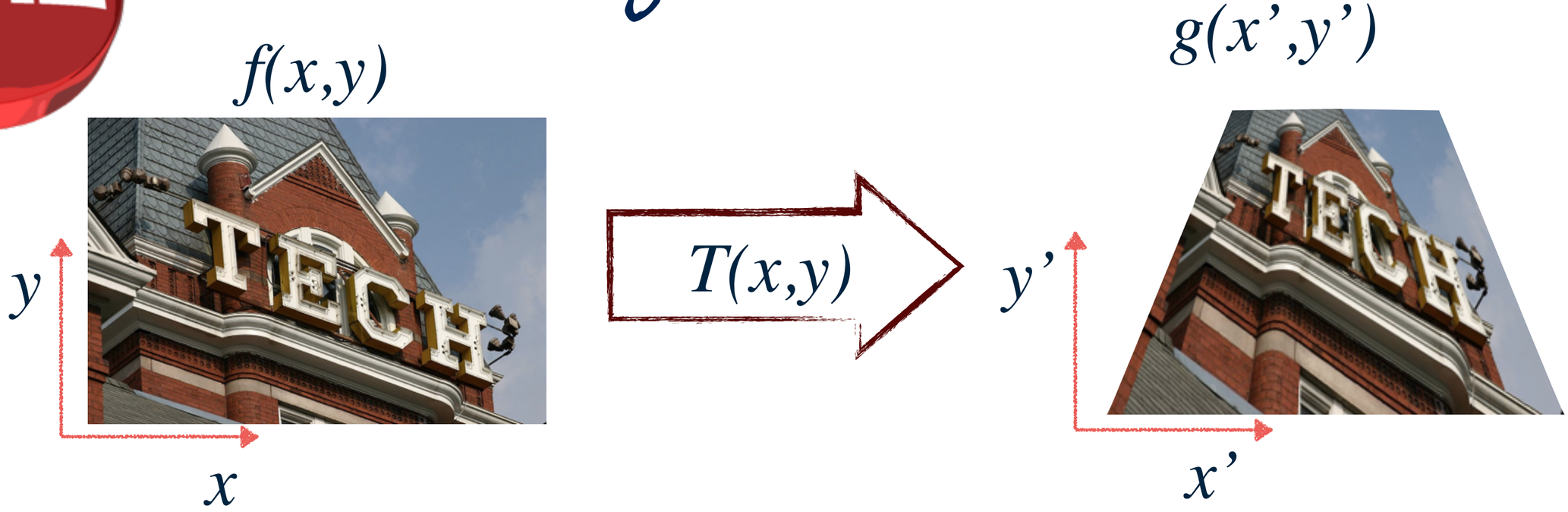
Affine



* How many correspondences needed for translation?

* How many Degrees of Freedom?

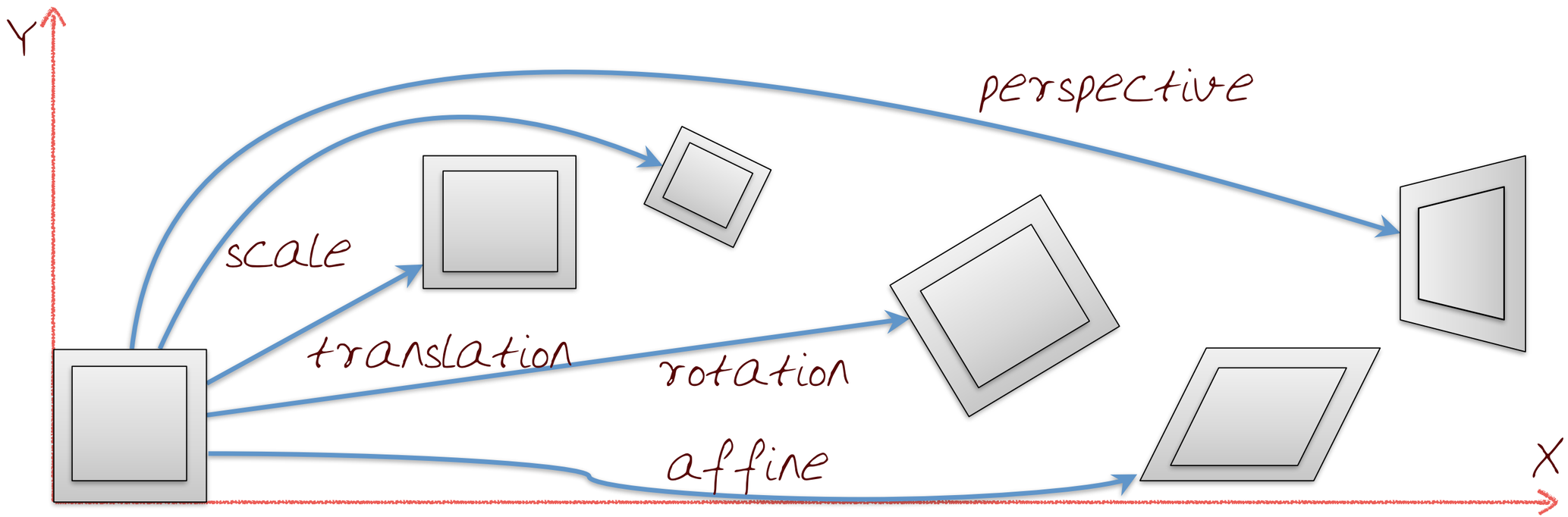
Projective



* How many correspondences needed for translation?

* How many Degrees of Freedom?

2D image transformations



Szeliski 2010

	Icon	DOF	3 X 3	Preserves
Translation		2		orientation
Euclidean		3		lengths
Similarity		4		angles
Affine		6		parallelism
Projective		8		straight lines

Szeliski 2010

DEMO

Translation

```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Translation
M_trans = np.float32(
    [[1, 0, 100],
     [0, 1,  50]])
print "Translation matrix:"
print M_trans
img_trans = cv2.warpAffine(img, M_trans, (width, height))  # note: last arg is size of output image
(width, height)
cv2.imshow("Translation", img_trans)
```

Rotation

```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Rotation around origin (0, 0)
M_rot = cv2.getRotationMatrix2D((0, 0), 45, 1)  # 45 deg, scale = 1
print "Rotation matrix (around origin):"
print M_rot
img_rot = cv2.warpAffine(img, M_rot, (width, height))
cv2.imshow("Rotation around origin", img_rot)

# Rotation around center (i.e. translation + rotation)
M_rot_center = cv2.getRotationMatrix2D((width / 2, height / 2), -45, 1)  # -45 deg, scale = 1
print "\nRotation matrix (around center):"
print M_rot_center
img_rot_center = cv2.warpAffine(img, M_rot_center, (width, height))
cv2.imshow("Rotation around center", img_rot_center)
```

Shear

```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Scale (resize)
img_scaled = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_CUBIC)
# specify scaling factors OR: img_scaled = cv2.resize(img, (int(1.5 * width), int(1.5 * height)),
interpolation=cv2.INTER_CUBIC)  # specify target size
cv2.imshow("Scale", img_scaled)

# Shear or skew (horizontal only)
M_shear = np.float32(
    [[1, 0.5, 0],
     [0,   1, 0]])
print "Shear matrix:"
print M_shear
img_shear = cv2.warpAffine(img, M_shear, (int(width + 0.5 * height), height))  # output image needs
to be wider to accomodate stretched image
cv2.imshow("Shear", img_shear)
```

Affine

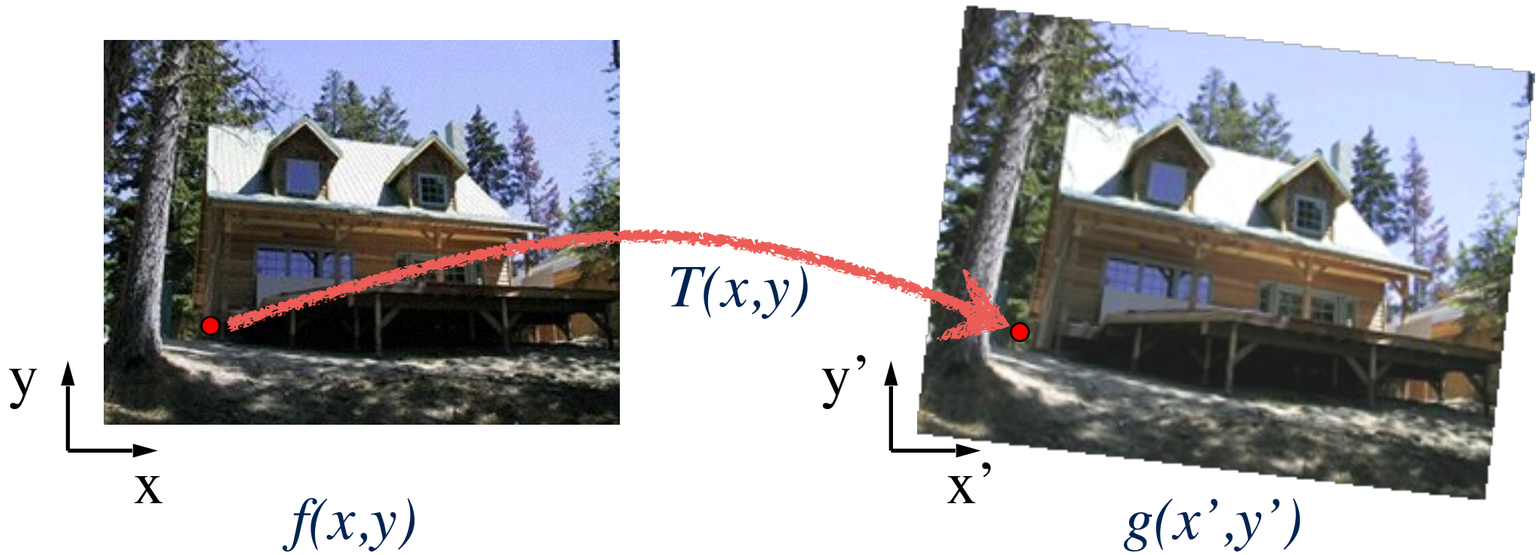
```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Affine transform (may include translation, rotation, scale)
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])  # original point locations
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])  # desired point locations after transform
(from feature matching, e.g.)

M_aff = cv2.getAffineTransform(pts1, pts2)  # compute affine transform (needs exactly 3
corresponding point pairs)
print "Computed affine transform matrix:"
print M_aff

img_aff = cv2.warpAffine(img, M_aff, (width, height))  # warp original image by applying it
cv2.imshow("Warped by affine transform", img_aff)
```

Projective

```
import cv2
import numpy as np

# Read image
img = cv2.imread('berlin-wall-03.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)
```

```
# Perspective transform
pts1 = np.float32([[220, 85], [223, 414], [602, 190], [616, 323]])
pts2 = np.float32([[75, 75], [75, 225], [500, 75], [500, 225]])

M_persp = cv2.getPerspectiveTransform(pts1, pts2)  # compute perspective transform (needs exactly 4
points)
print "Computed perspective transform matrix:"
print M_persp

img_persp = cv2.warpPerspective(img, M_persp, (600, 300))  # apply transform; last arg is output
image size
cv2.imshow("Perspective transform", img_persp)
```

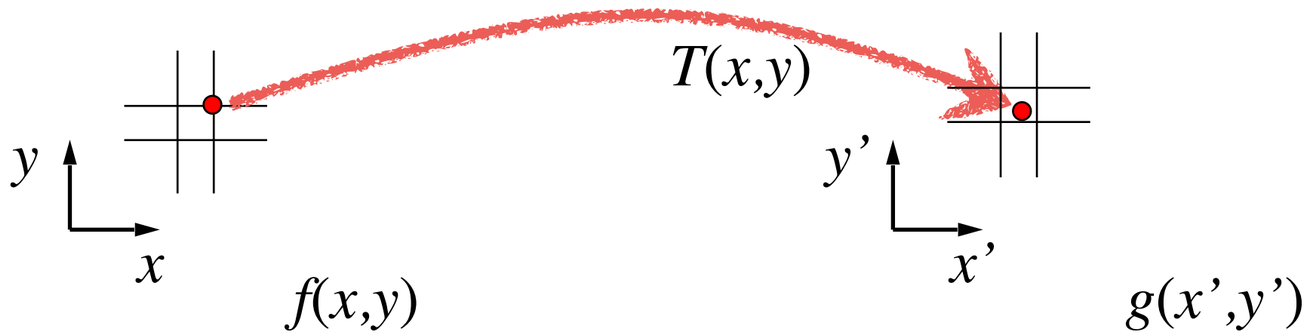
Forward warping



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

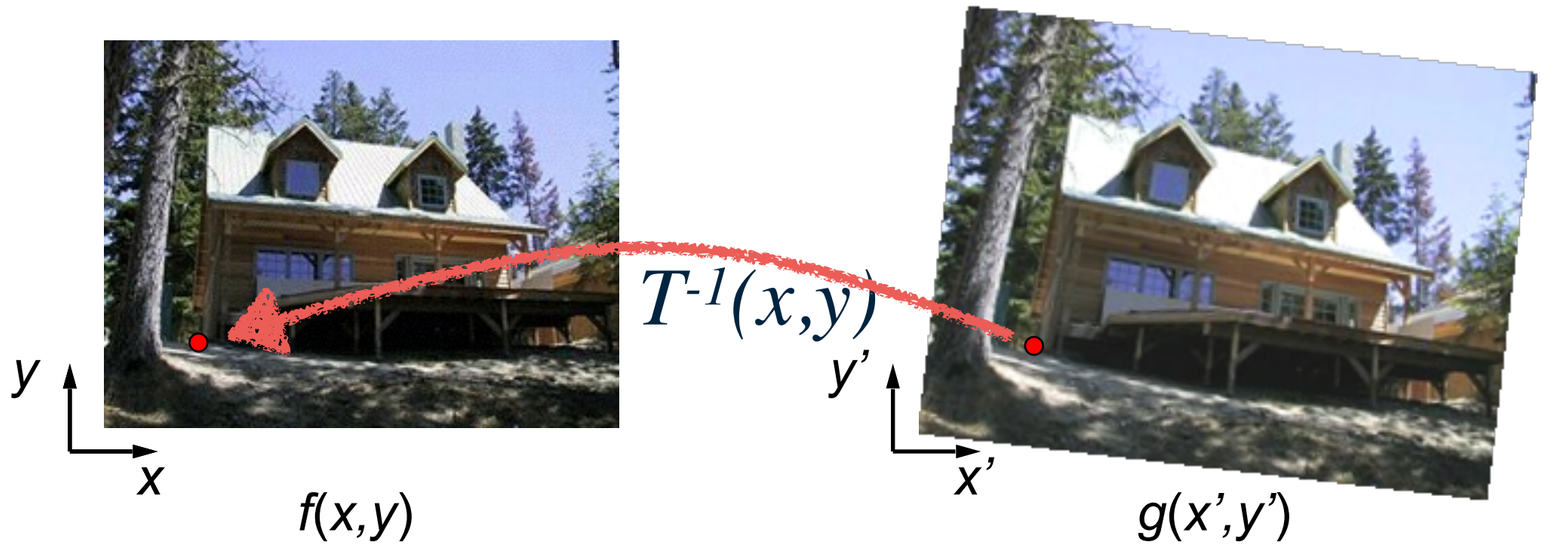
* Send each pixel $f(x,y)$ to its corresponding location
  * $(x',y') = T(x,y)$ in the second image
* Q: what if pixel lands "between" two pixels?

Forward warping



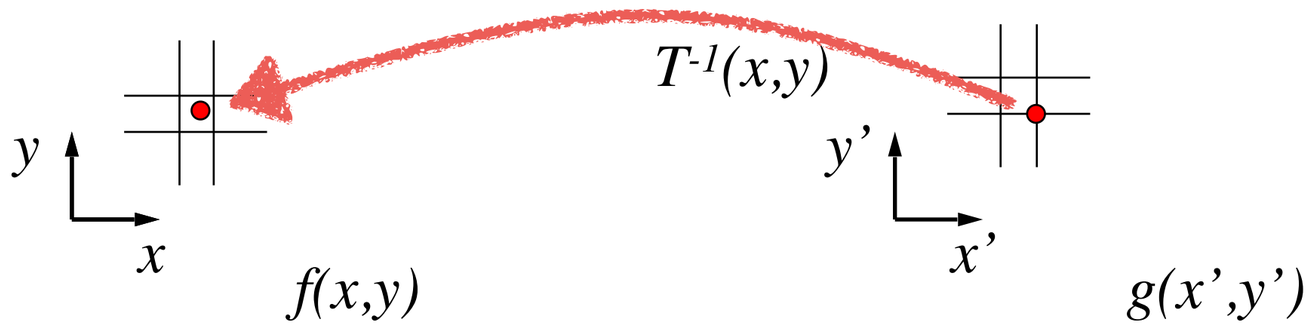
* Send each pixel $f(x,y)$ to its corresponding location
  * $(x',y') = T(x,y)$ in the second image
* Q: what if pixel lands "between" two pixels?
* A: distribute color among neighboring pixels (x',y')

Inverse warping



* Get each pixel $g(x',y')$ from its corresponding location
  * $(x,y) = T^{-1}(x',y')$ in the first image
* Q: what if pixel comes from "between" two pixels?

Inverse warping



* Get each pixel $g(x',y')$ from its corresponding location
  * $(x,y) = T^{-1}(x',y')$ in the first image
* Q: what if pixel comes from "between" two pixels?
* A: Interpolate color value from neighbors

Forward vs. inverse warping

* Q: which is better?

* A: usually inverse ⇒ eliminates holes

  * however, it requires an invertible warp function ⇒ not always possible...

Summary

* Learned about Image Transformations (besides image filtering)
* Studied details of Rigid to Projective Transformation of Images

Credits

* For more information, see:
  * Richard Szeliski (2010) Computer Vision: Algorithms and Applications, Springer (Chapter 2)
* Some concepts in slides motivated by similar slides by Aaron Bobick, James Hays and Greg Turk
* Additional list will be available on website

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

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

Image Warping and Morphing

* Warping and morphing applied to images

Georgia Tech College of Computing

Lesson Objectives

1. Image Warping
2. Forward and Inverse Warping
3. Warping using a mesh
4. Image Morphing
5. Feature-based Image Morphing

Recall: Image Transformations

image filtering:
change range of image

$$g(x) = T(f(x))$$

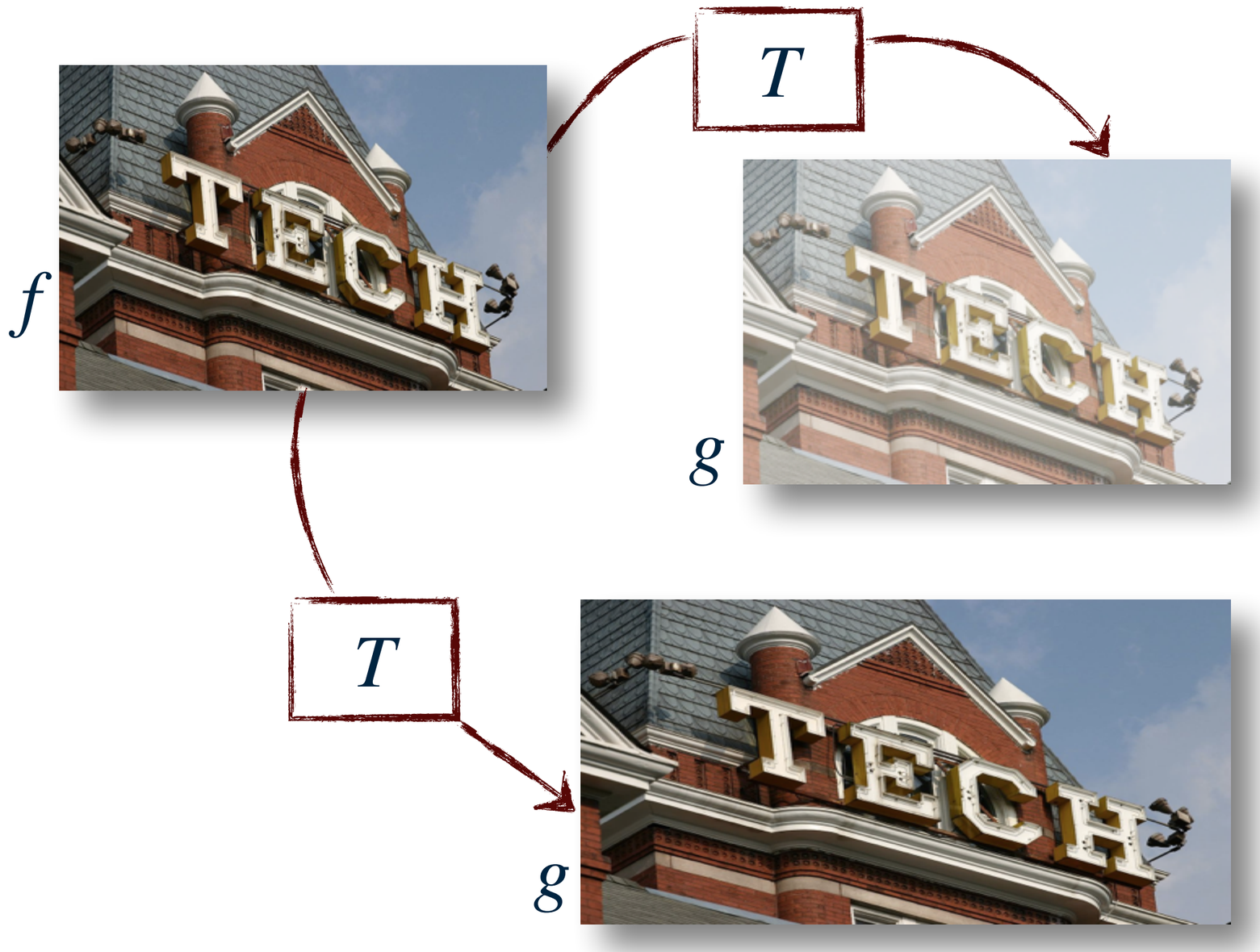


image warping:
change domain of image

$$g(x) = f(T(x))$$

Image Transformation vs. Warping

* Transformation: Lines remain lines
* Warping: Points are mapped to points
* A mathematical function for warping from a plane to the plane

Image Warping

- Distorted through simulation of optical aberrations
- Projected onto a curved or mirrored surface
- Partitioned into polygons and each polygon distorted
- Distorted using morphing

Two methods: Forward/Inverse

* Consider a S and T image
* S has pixel coordinates: (u,v)
* T has pixel coordinates: (x,y)

$S \Rightarrow (u,v)$

Forward $(x, y) = [X(u, v), Y(u, v)]$

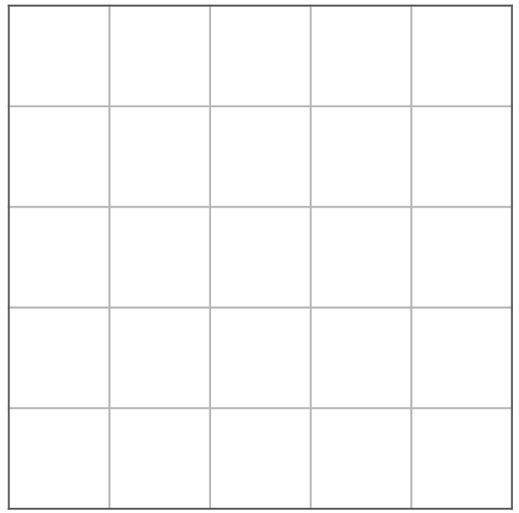
Inverse $(u, v) = [U(x, y), V(x, y)]$

$T \Rightarrow (x,y)$

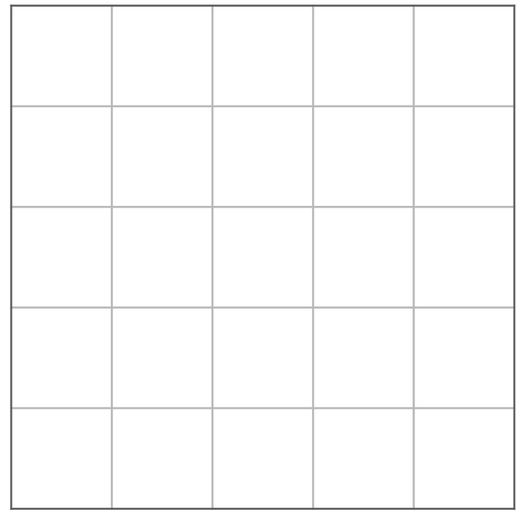
Forward/Inverse Warping

$$(x, y) = [X(u, v), Y(u, v)]$$

Input



(u,v) integer valued

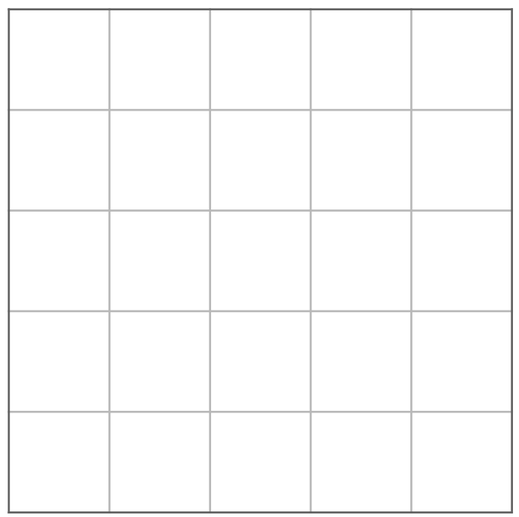


(x,y) real valued

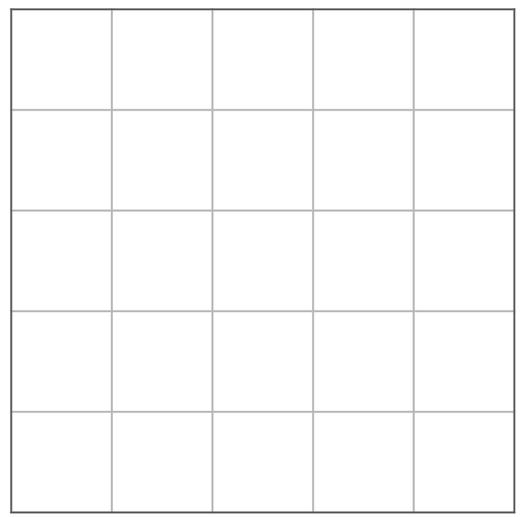
Output

$$(u, v) = [U(x, y), V(x, y)]$$

Input



(u,v) real valued



(x,y) integer valued

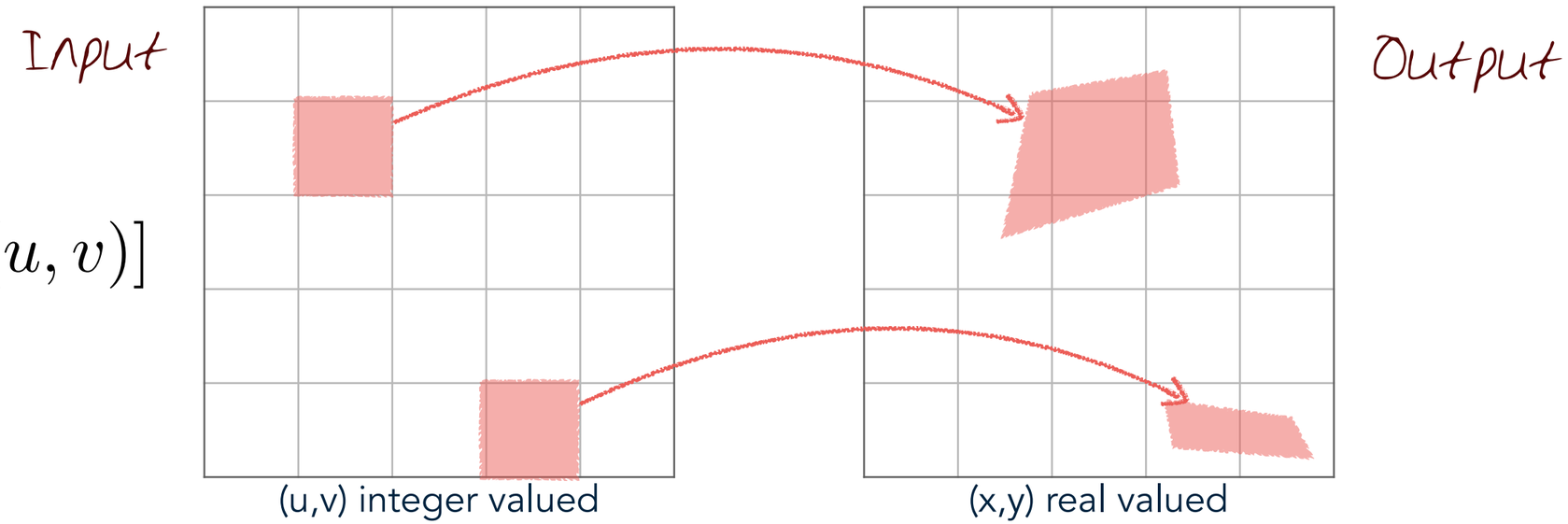
Output

Forward/Inverse Warping

$$(x, y) = [X(u, v), Y(u, v)]$$

Problems:

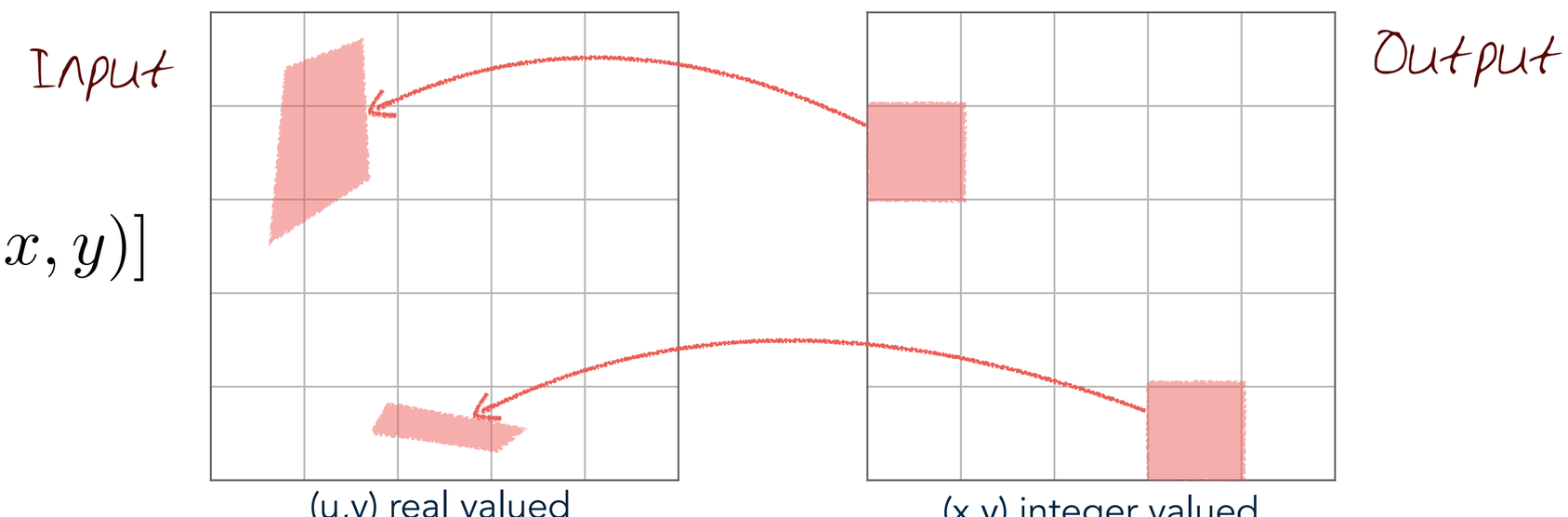
Holes, Overlaps



$$(u, v) = [U(x, y), V(x, y)]$$

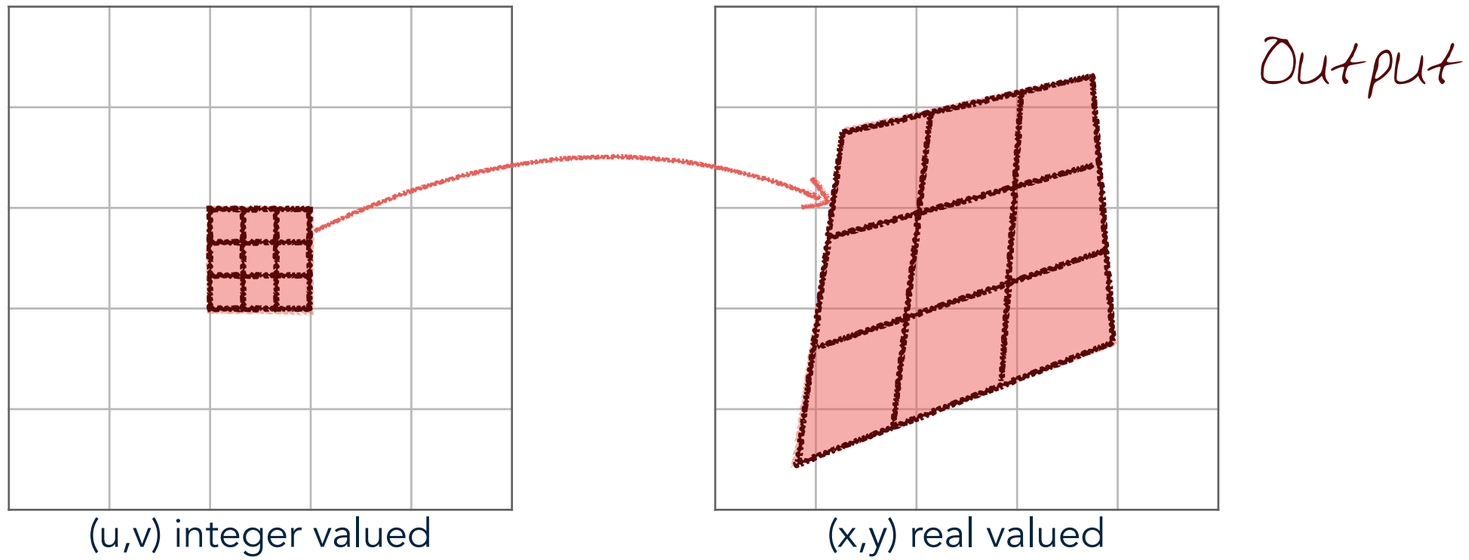
Problems:

Minification

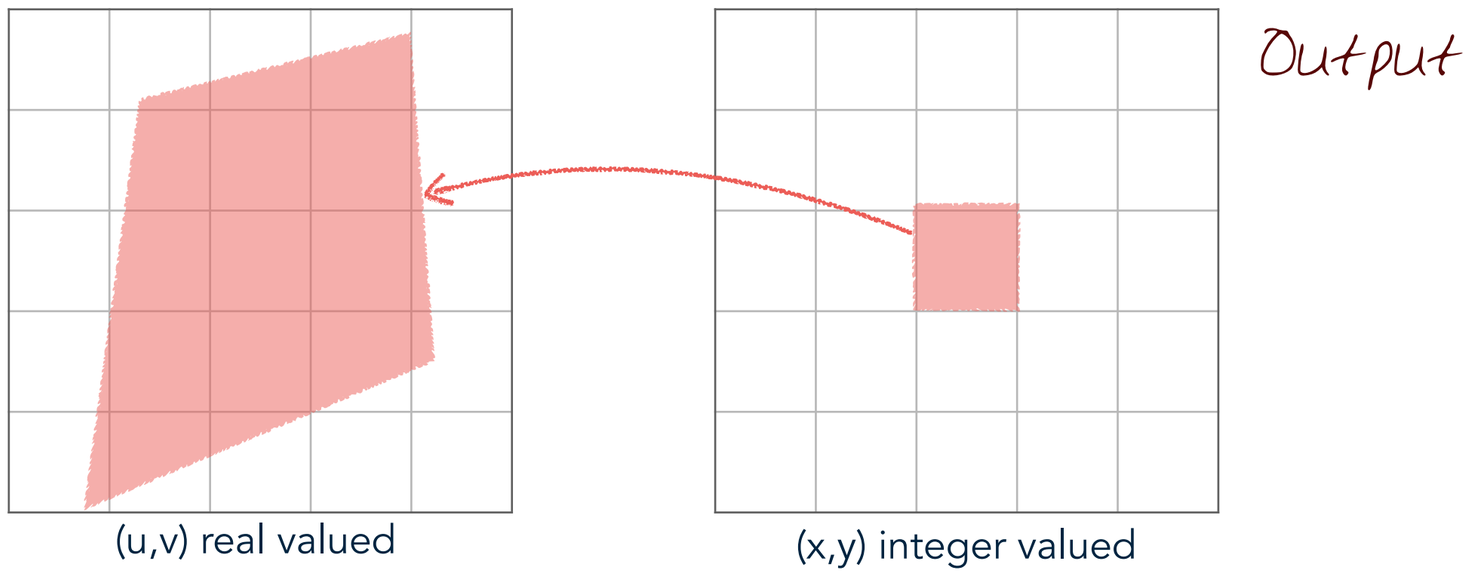


Magnification / Minification

$$(x, y) = [X(u, v), Y(u, v)]$$



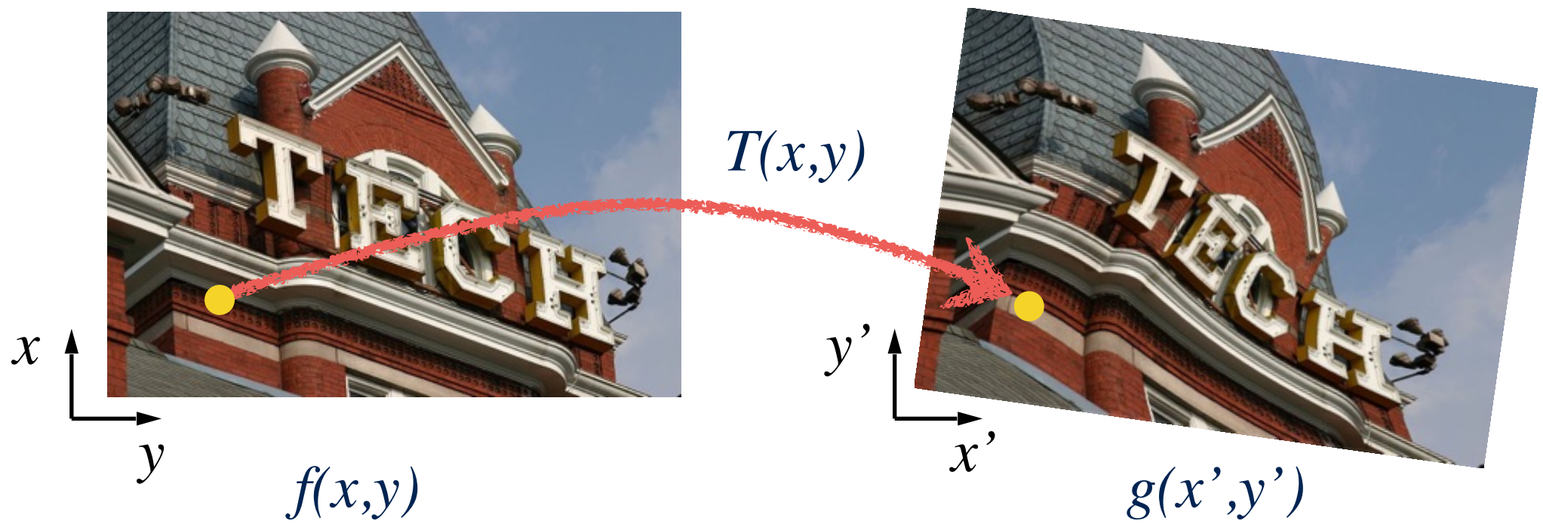
$$(u, v) = [U(x, y), V(x, y)]$$



Additional Info:

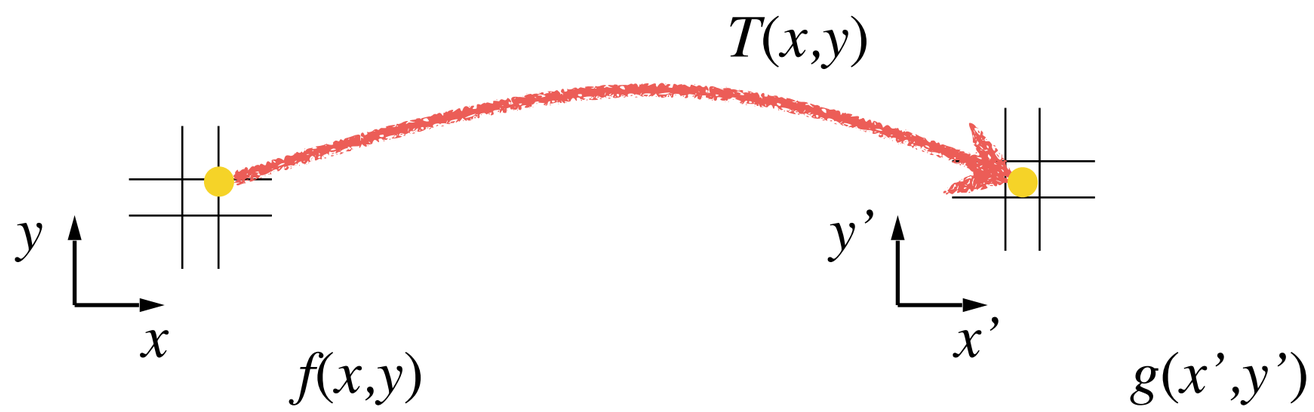
See Two-pass Transforms

Recall: Forward warping



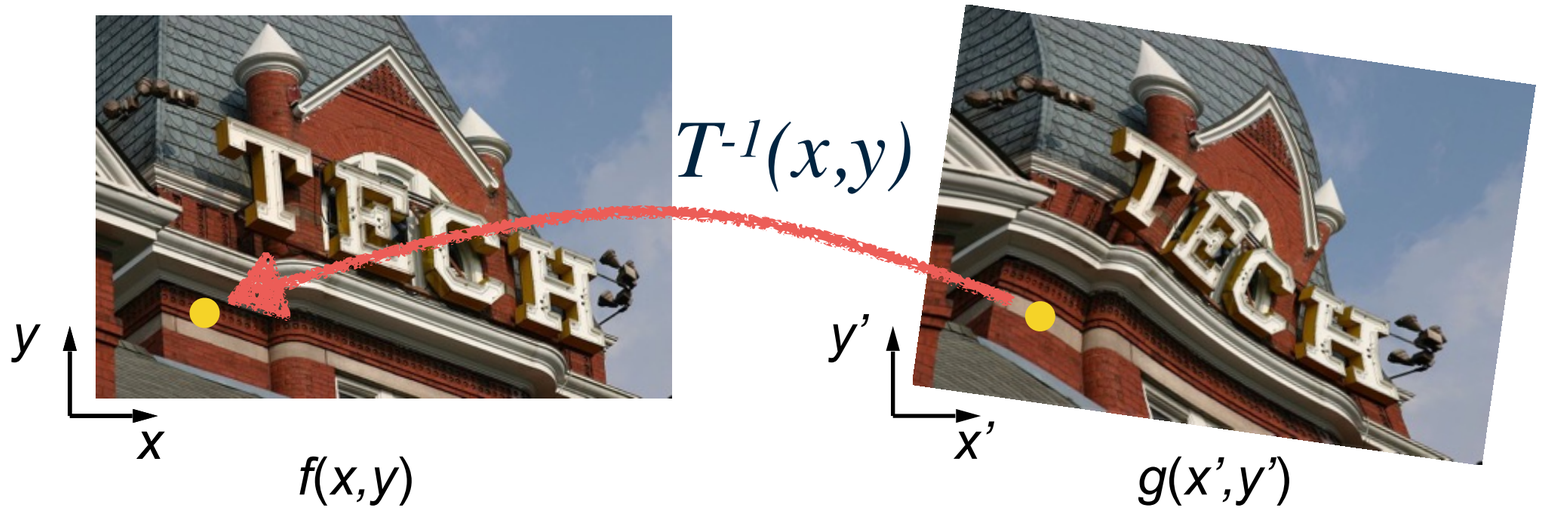
* Send each pixel $f(x,y)$ to its corresponding location
  * $(x',y') = T(x,y)$ in the second image
* Q: what if pixel lands "between" two pixels?

Recall: Forward warping



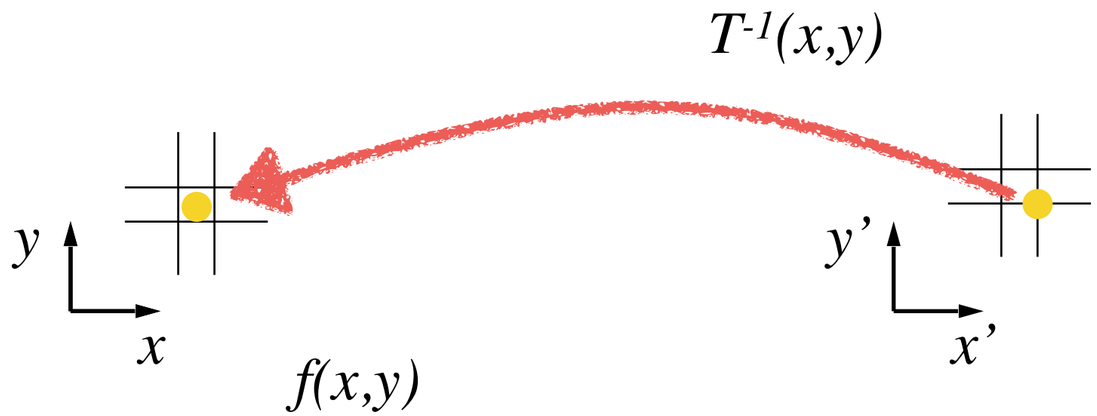
- Send each pixel $f(x,y)$ to its corresponding location
  - $(x',y') = T(x,y)$ in the second image
- Q: what if pixel lands "between" two pixels?
- A: distribute color among neighboring pixels (x',y')

Recall: Inverse warping



* Get each pixel $g(x',y')$ from its corresponding location
  * $(x,y) = T^{-1}(x',y')$ in the first image
* Q: what if pixel comes from "between" two pixels?

Recall: Inverse warping



* Get each pixel $g(x',y')$ from its corresponding location
  * $(x,y) = T^{-1}(x',y')$ in the first image
* Q: what if pixel comes from "between" two pixels?
* A: Interpolate color value from neighbors

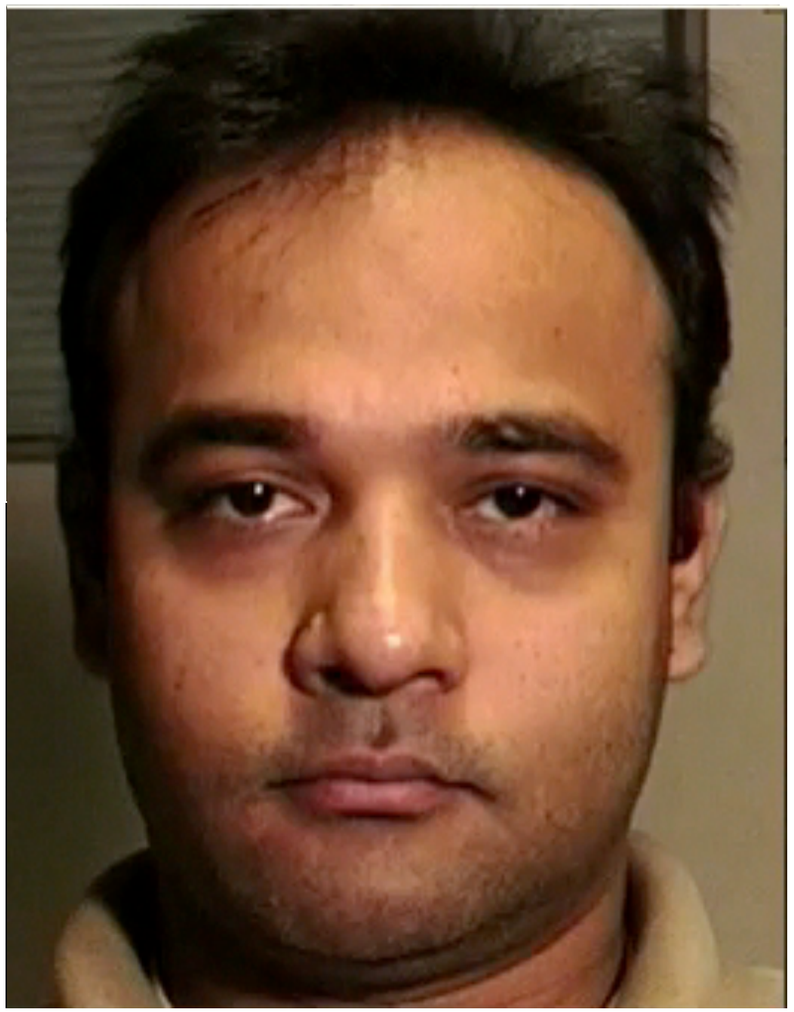
Forward vs. inverse warping

* Q: which is better?

* A: usually inverse ⇒ eliminates holes

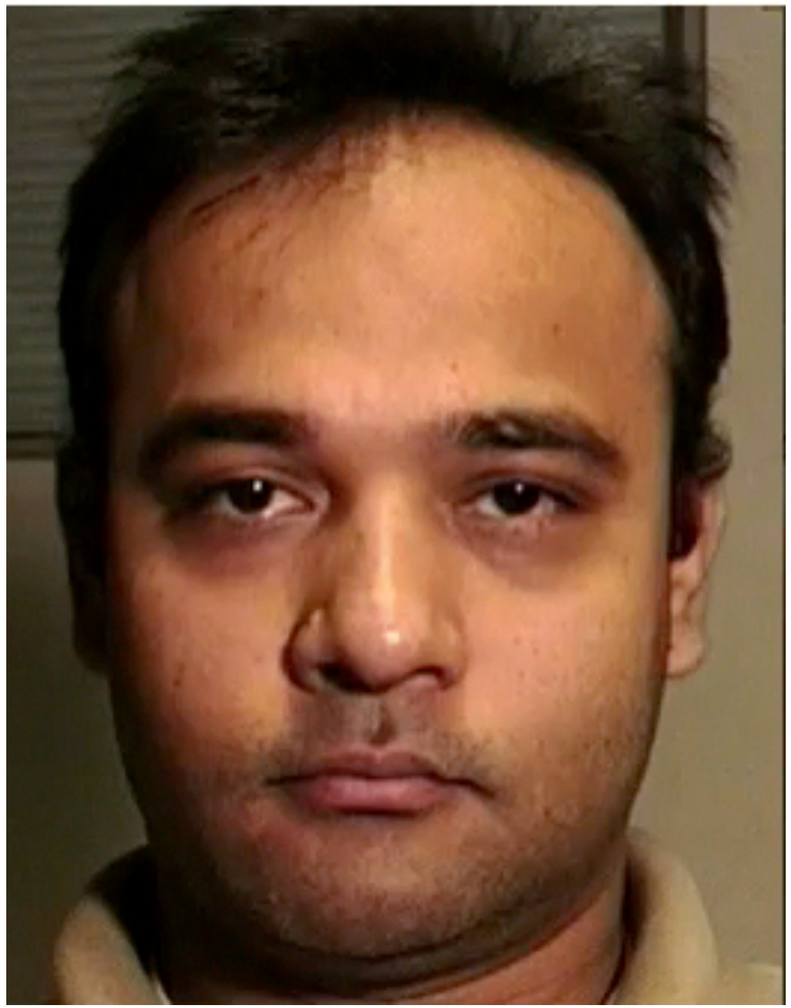
  * however, it requires an invertible warp function ⇒ not always possible...

Mesh-based Warping



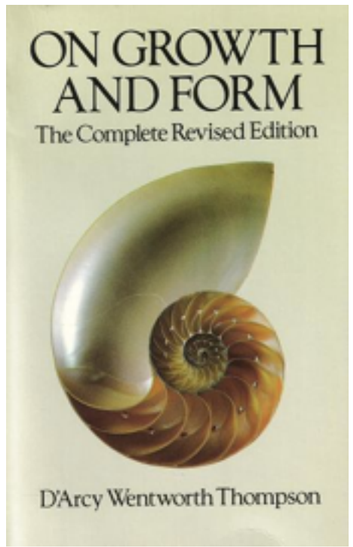
* Use a sparse set of corresponding points and interpolate with a displacement field
* Triangulate the set of points on Source
* Use the affine model for each triangle
* Triangulate Target with displaced Points
* Use inverse mapping

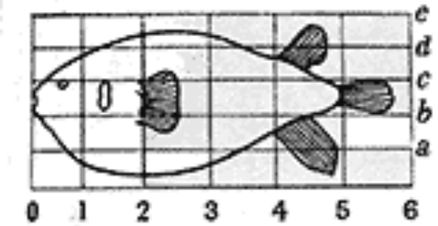
Image Morphing



* Animations that changes (or morphs) one image or shape into another through a seamless transition

* Widely used in movies





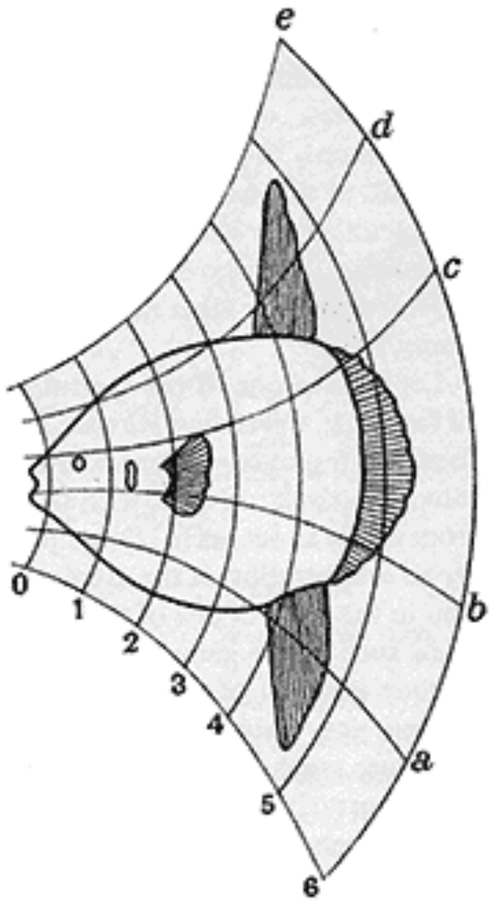
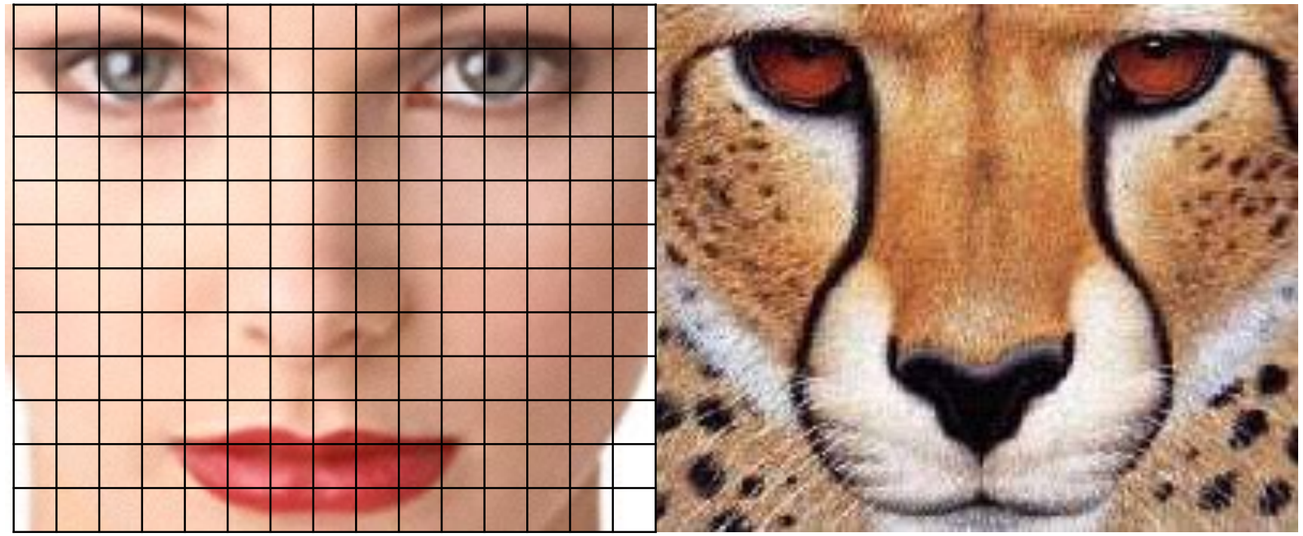


Image Morphing: Approaches

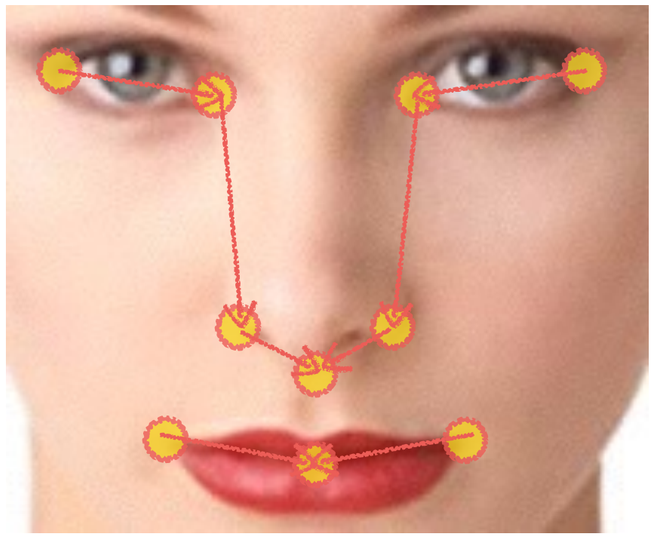
* Quadrilateral mesh displaced with variational interpolation

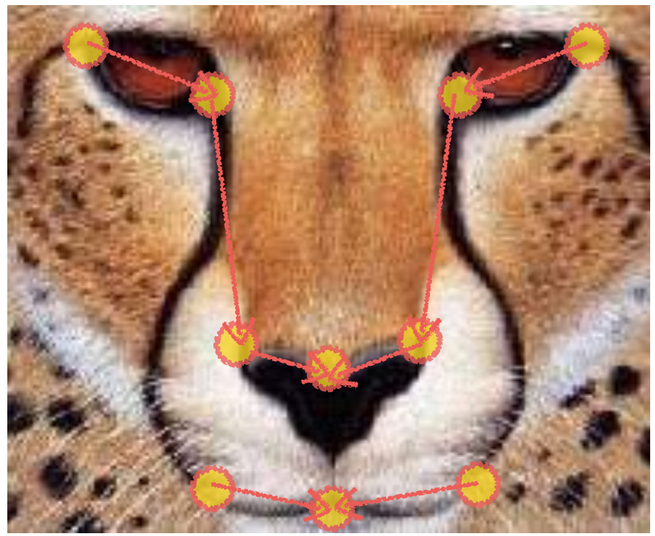


Images on this slide are for demonstration purposes only and may not reflect actual processed images
For more details see Szeliski Chapter 3

Image Morphing: Approaches

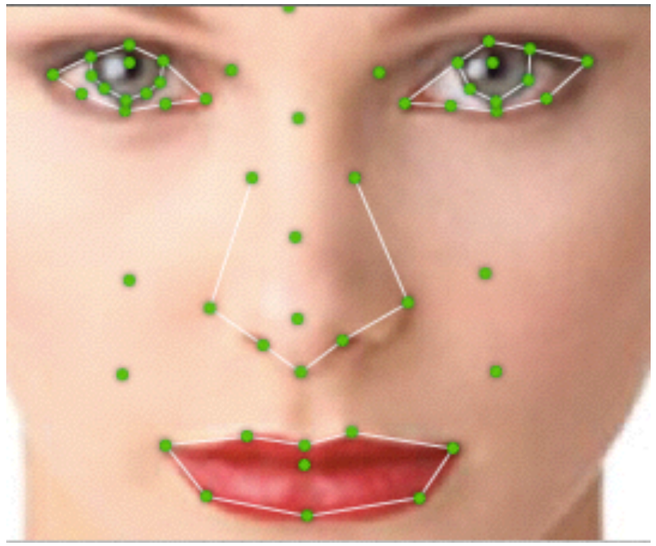
* Quadrilateral mesh displaced with variational interpolation
* Corresponding features/ points
* Corresponding orientated line segments (specifies translation, rotation, scaling)

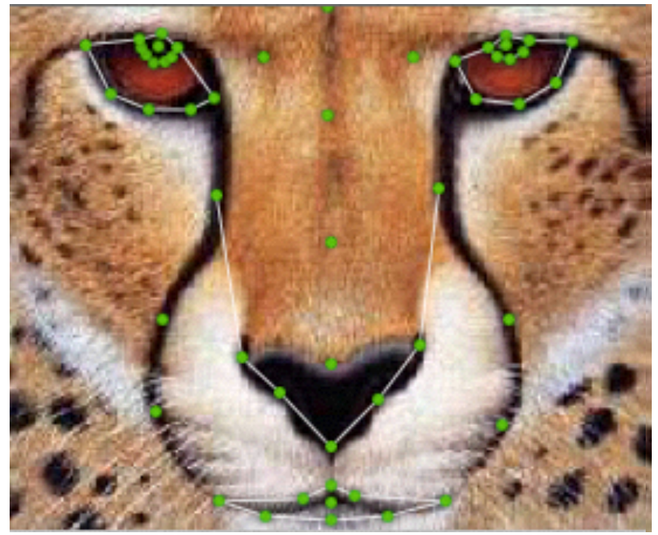




Images on this slide are for demonstration purposes only and may not reflect actual processed images
For more details see Szeliski Chapter 3

Feature-based Morphing





Beier and Neely (1992)

Software used: Fantamorph v5

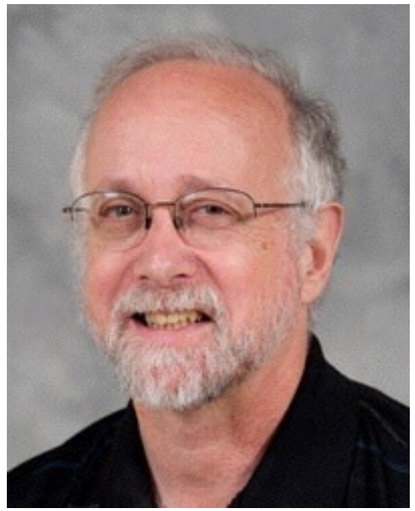
Feature-based Morphing

Michael Jackson - Black Or White

Beier and Neely (1992)

Feature-based Morphing

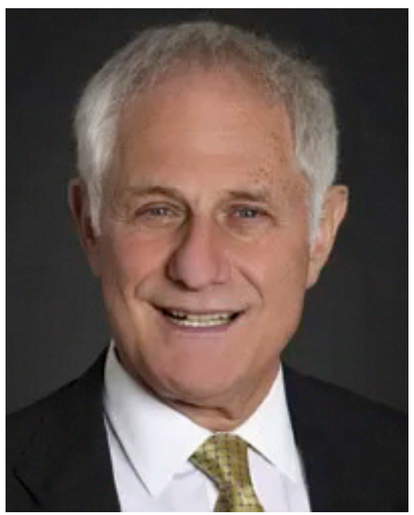
Zvi



Ron

Irfan

Charles



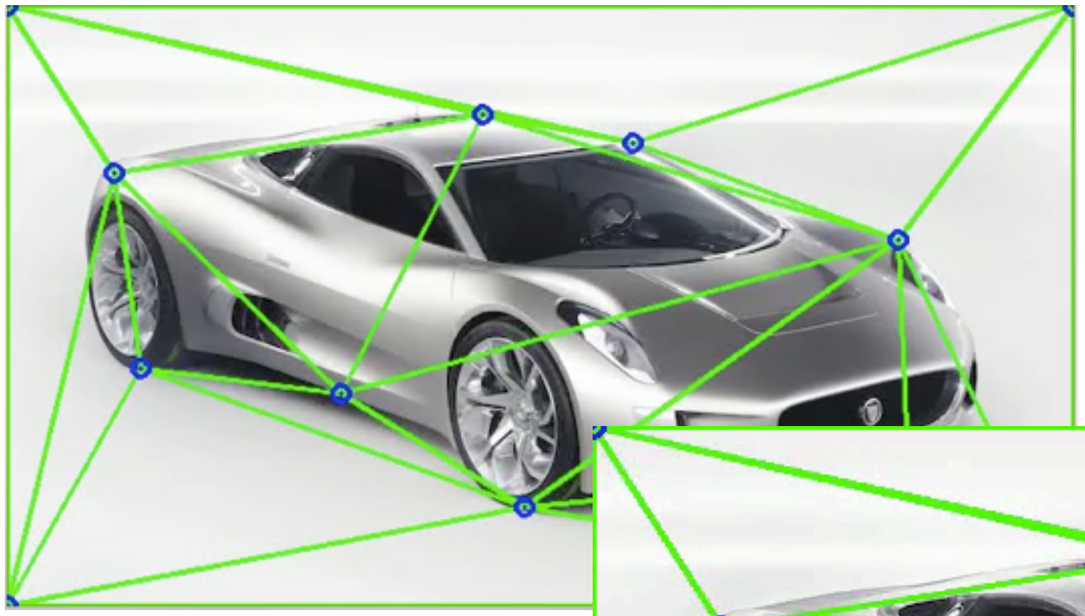
CoC Deans

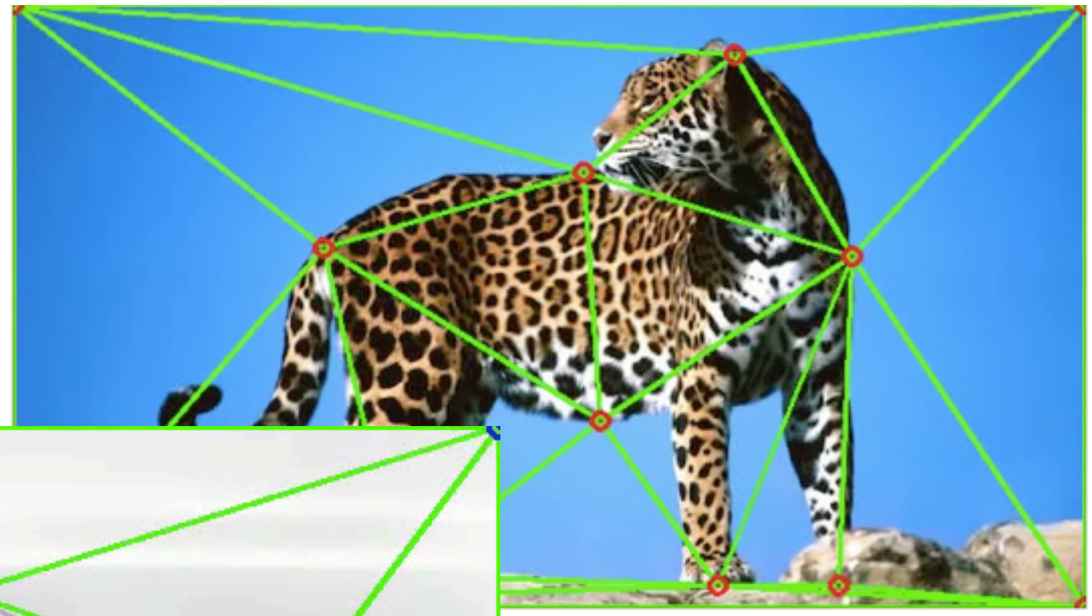
Morphed

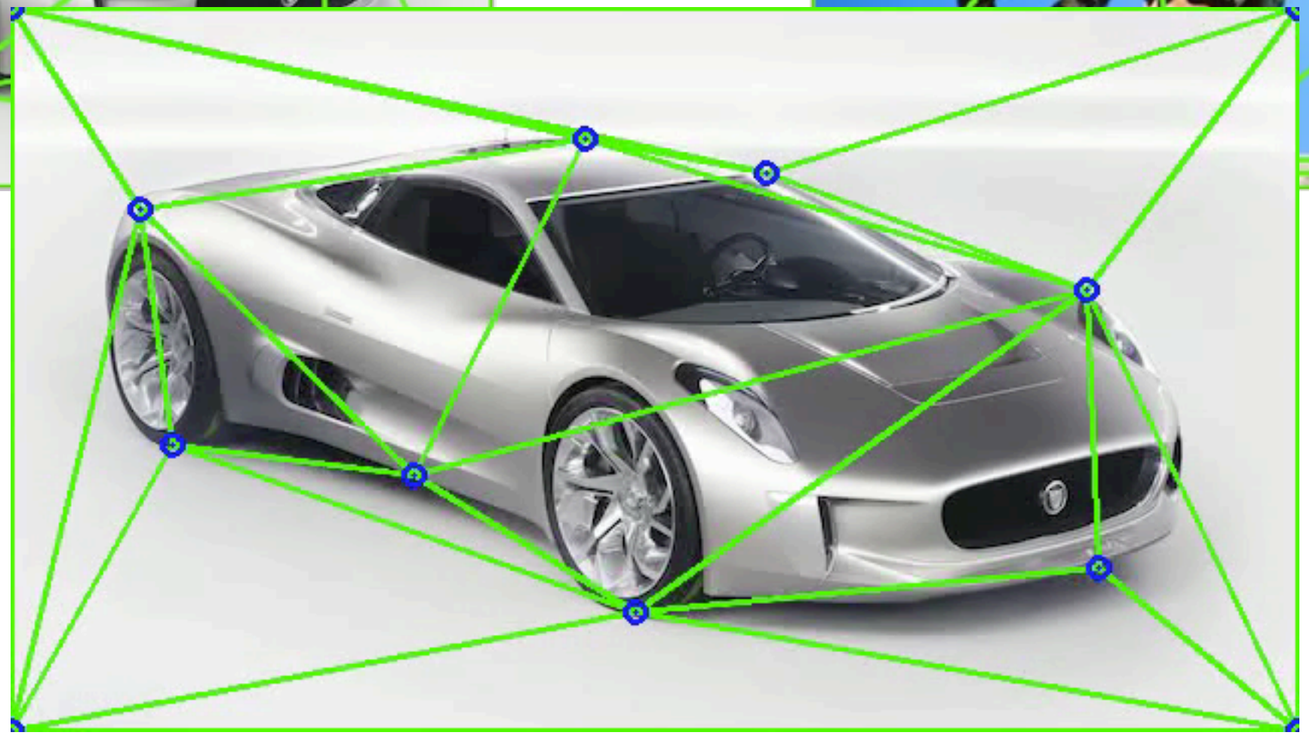
Beier and Neely (1992)

Software used: Fantamorph v5

Demo





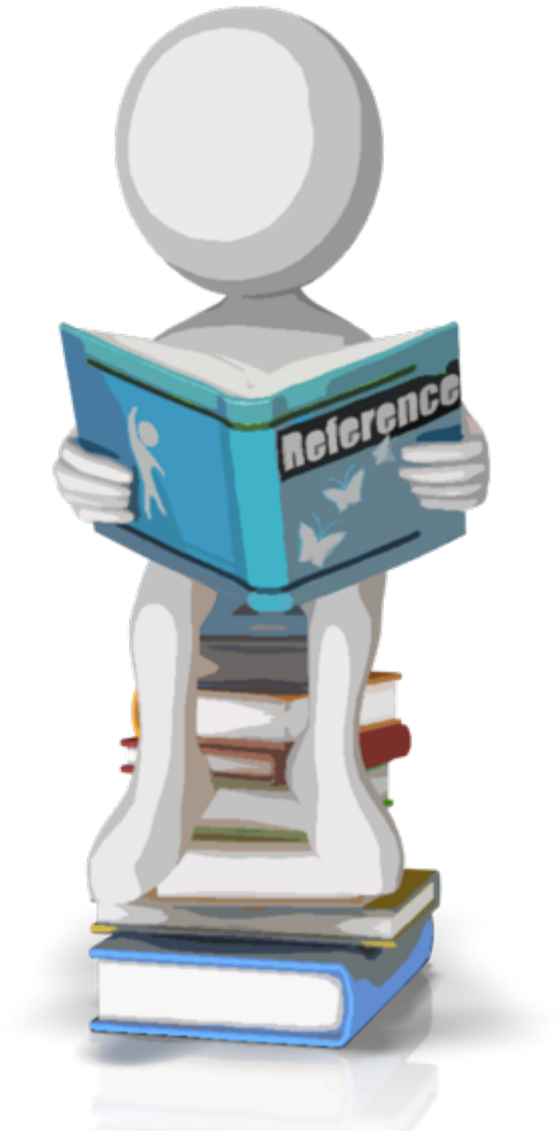


```
# Main script
if __name__ == "__main__":
    # Read two images: We want to morph from src to dst
    img_src = cv2.imread("jaguar-cx75.png")
    img_dst = cv2.imread("jaguar-standing.png")

    # Define corresponding point pairs (x, y)
    pts_src = np.float32(
        [[64, 100], [80, 217], [285, 65], [200, 232],
         [375, 82], [310, 300], [534, 140],[540, 278]])
    pts_dst = np.float32(
        [[185, 145], [230, 340], [340, 100], [350, 248],
         [430, 30], [420, 346], [500, 150], [492, 346]])

    # Visualize points [debug]
    '''
    img_src_out = img_src.copy()
    img_dst_out = img_dst.copy()
    drawPoints(img_src_out, pts_src, (255, 0, 0), (255, 0, 0))
    drawPoints(img_dst_out, pts_dst, (0, 0, 255), (0, 0, 255))
    cv2.imshow("src", img_src_out)
    cv2.imshow("dst", img_dst_out)
    '''

    # Call morphing function with src, dst images and points, and optional parameters
    num_frames = 31
    morph(img_src, img_dst, pts_src, pts_dst, num_frames, save_video=True, show_points=False,
show_triangles=False)
```

Summary

* More details about Image Warping
* Basics of Forward and Inverse Warping
* How to warp an image using a mesh
* Image Morphing
* Feature-based Image Morphing

Further Reading



* Beier and Neely (1992) "Feature-based Image Metamorphosis" ACM SIGGRAPH 1992

* Szeliski (2010) Chapter 3

Credits

* For more information, see:
  * Richard Szeliski (2010) Computer Vision: Algorithms and Applications, Springer (Chapter 3)
* Some concepts in slides motivated by similar slides by James Hays Alyosha Efros and Greg Turk
* Additional list will be available on website

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

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

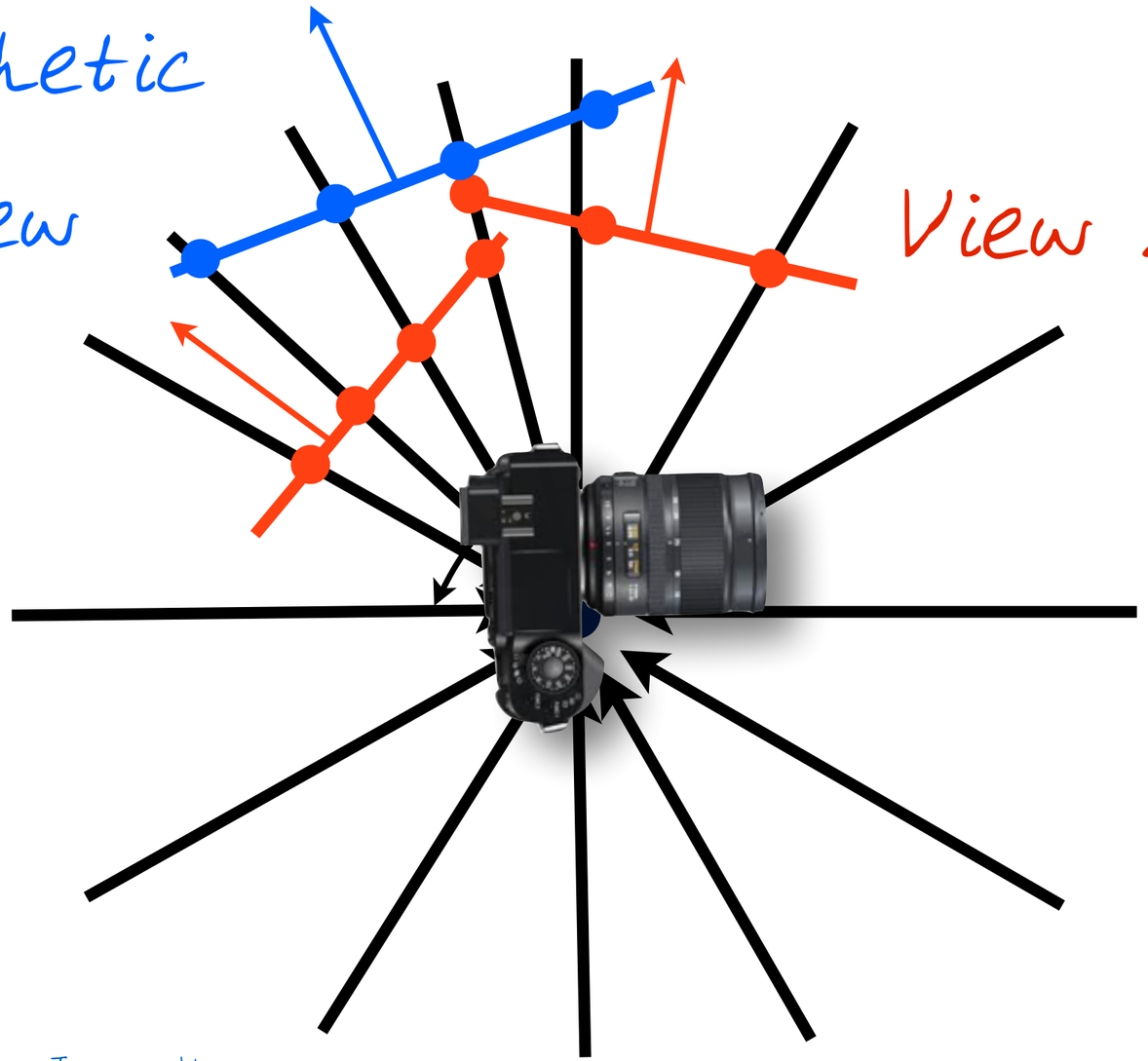
Better: Warp

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

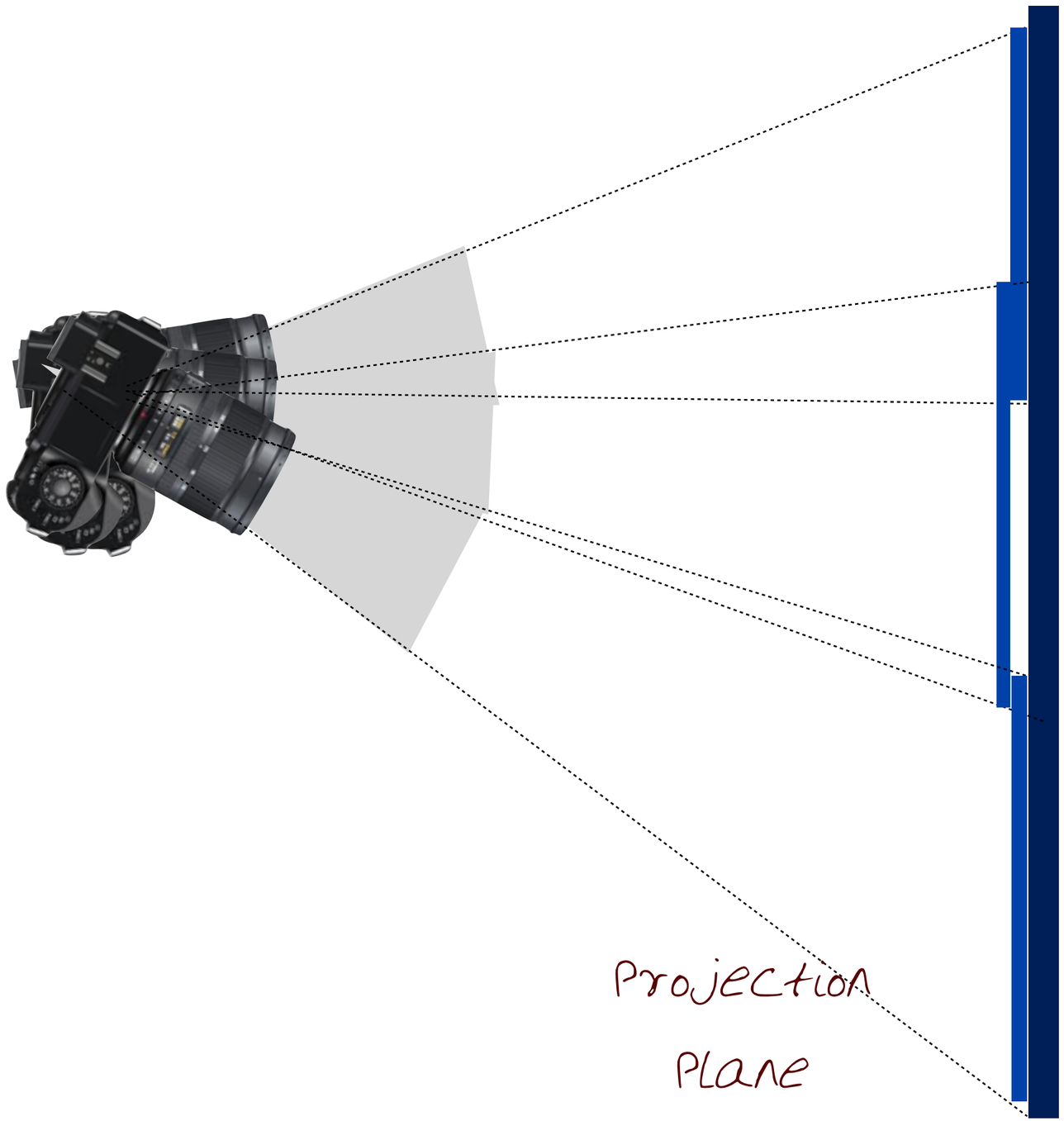


Image Re-projection to Panorama Projection Plane

* 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**

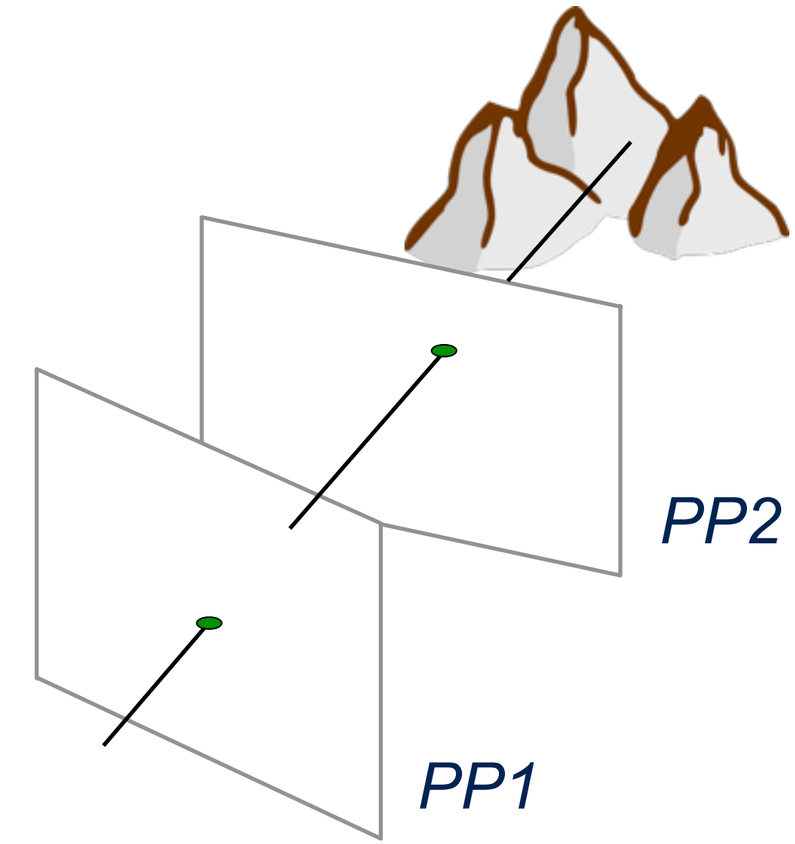
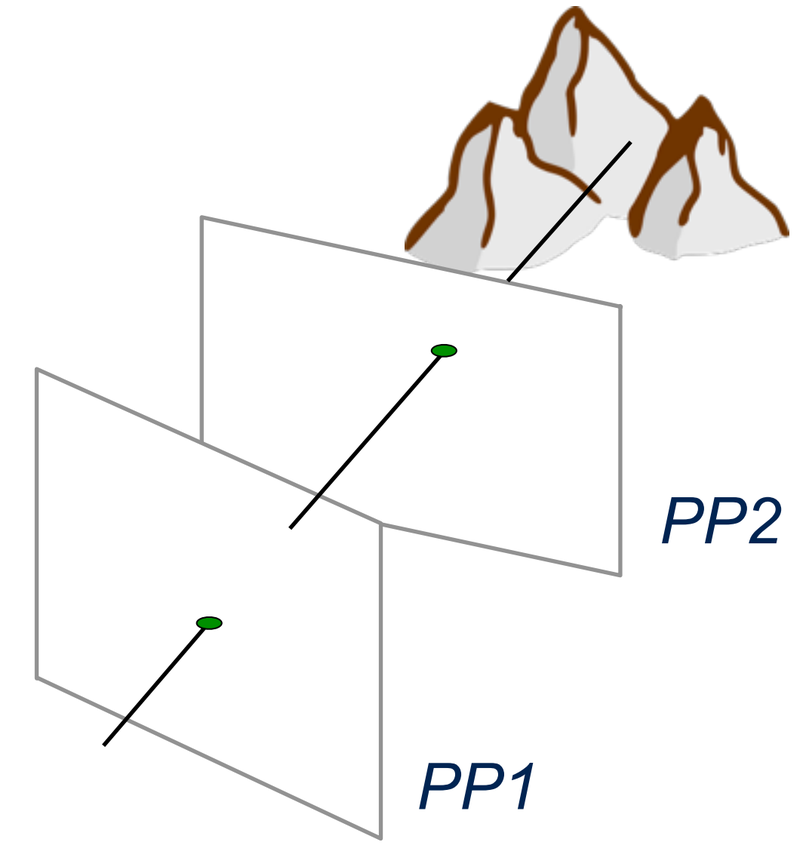


Image Re-Projection (2)

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?

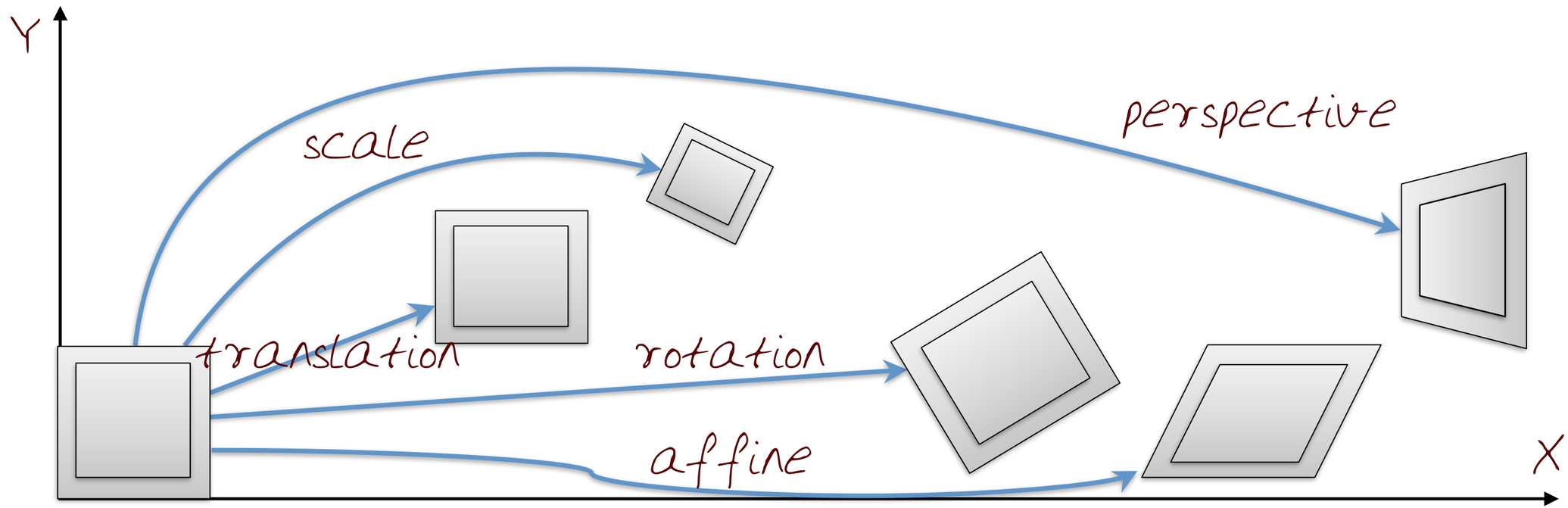
Recall: Image Warping

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

E.g. translation, Euclidean, affine, projective

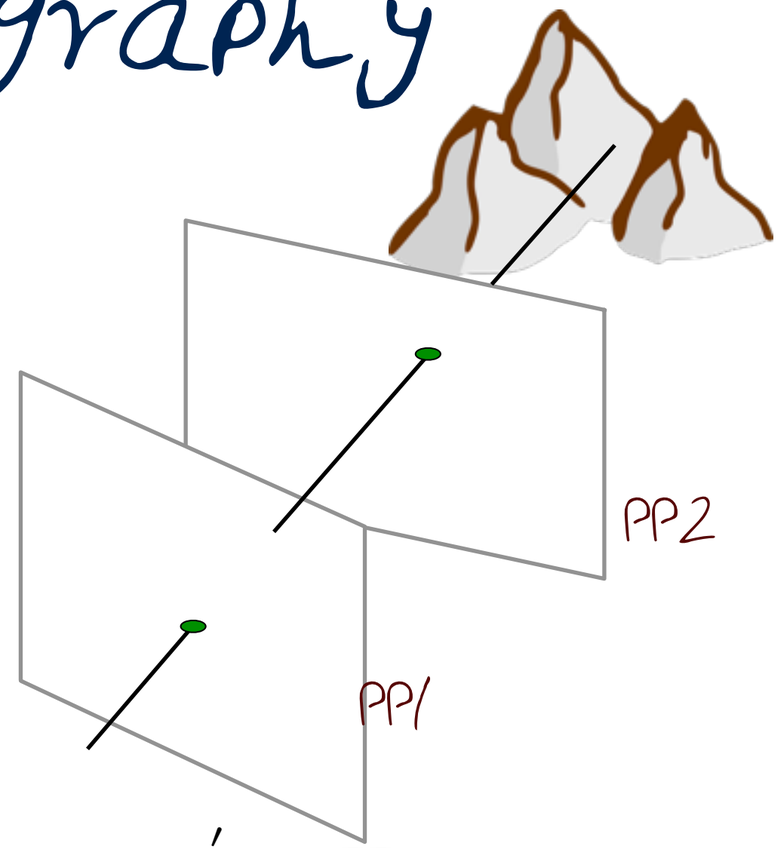
Translation: 2 unknowns, Euclidean: 3 unknowns

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

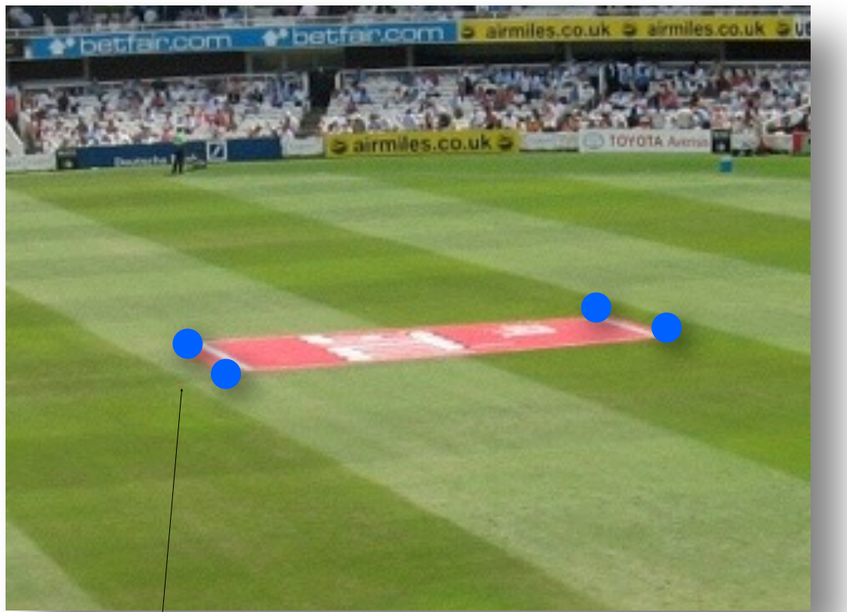
$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

Computing Homography

Computing Homography

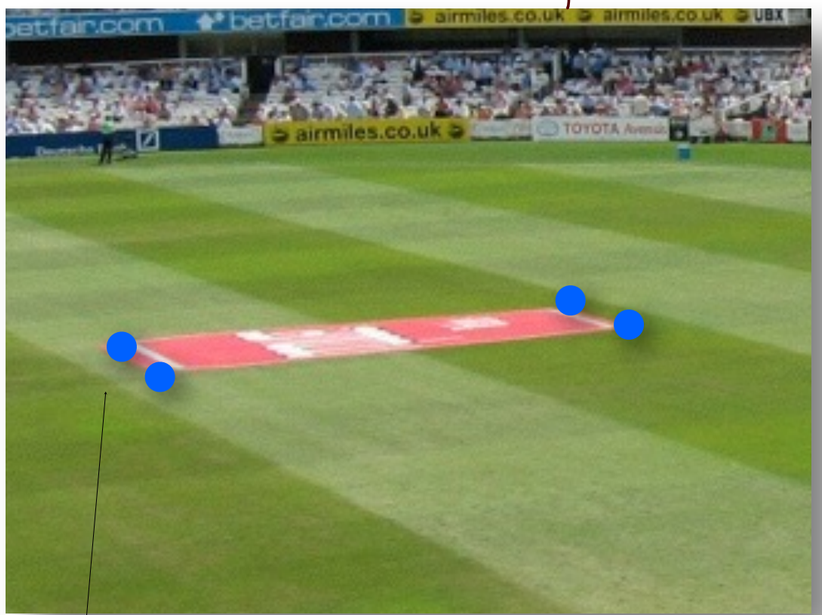
Zoomed



(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

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

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

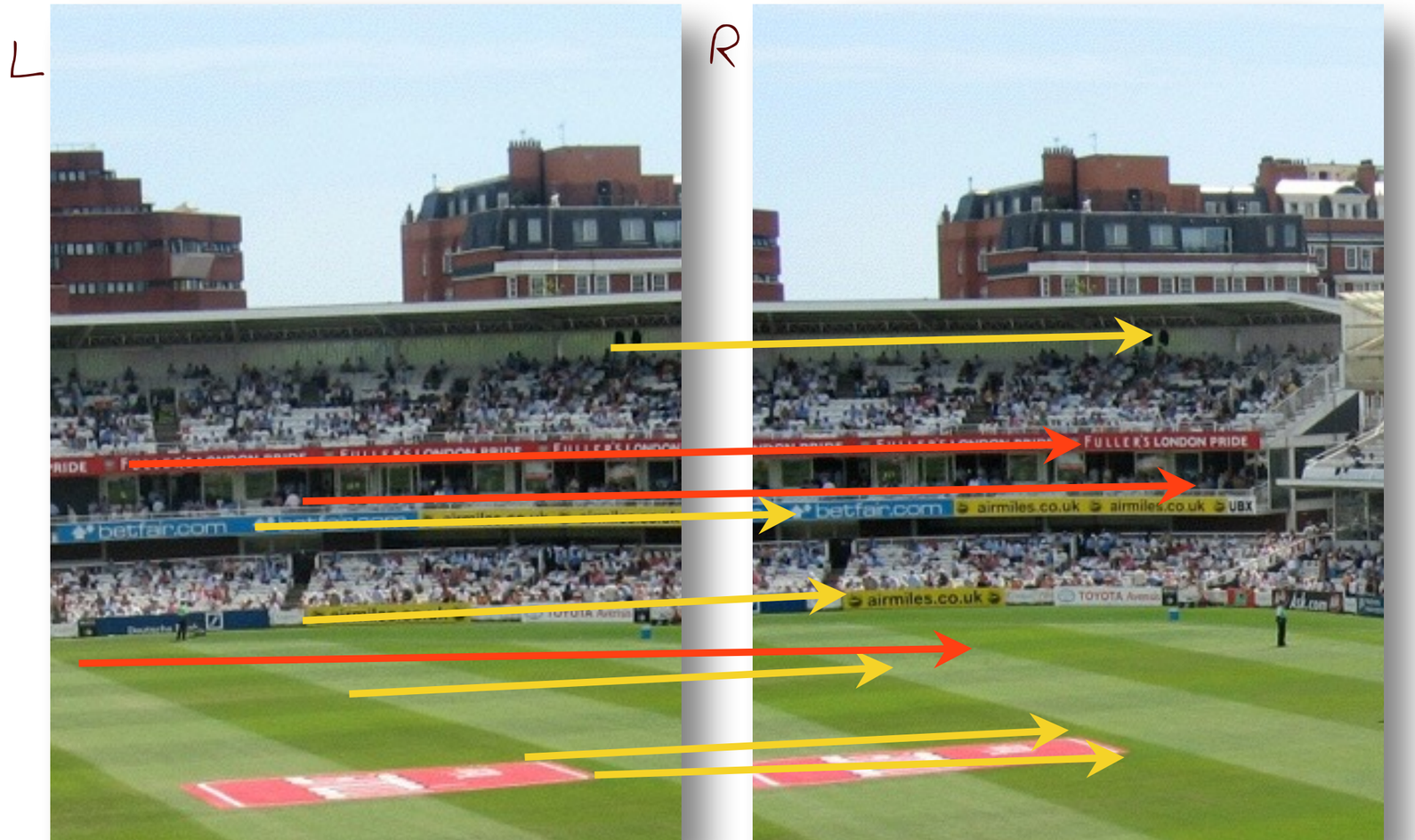
* Need at least 8 equations, but the more the better...
* Solve for h. If over-constrained (i.e. more data), solve using least-squares:

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

Warp into a Shared Coordinate Space

Warping and Interpolation

Dealing with BAD Matches



RANdom SAmple Consensus (RANSAC)

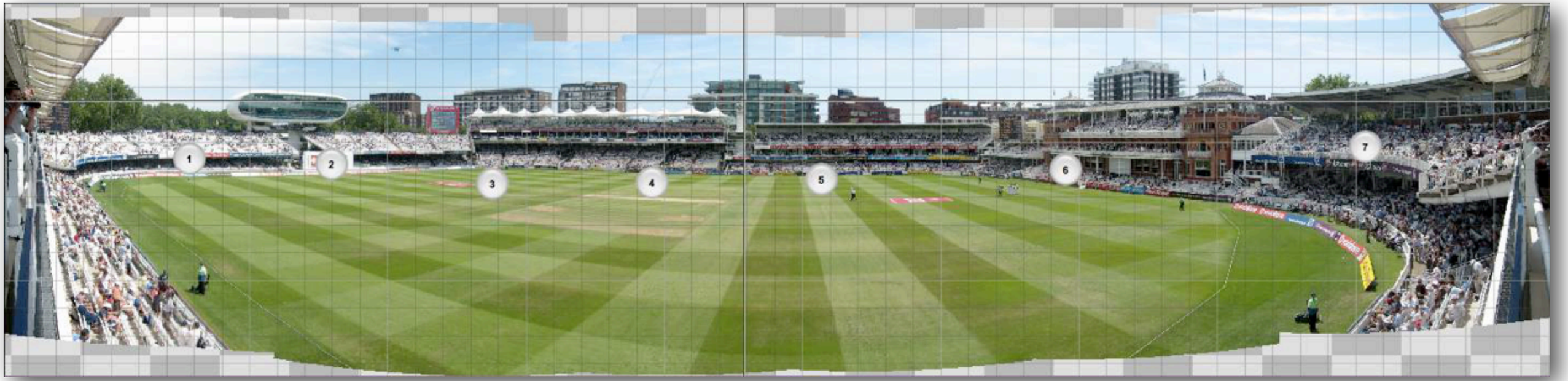
* Select ONE match, Count INLIERS
* Find "average" translation vector

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.



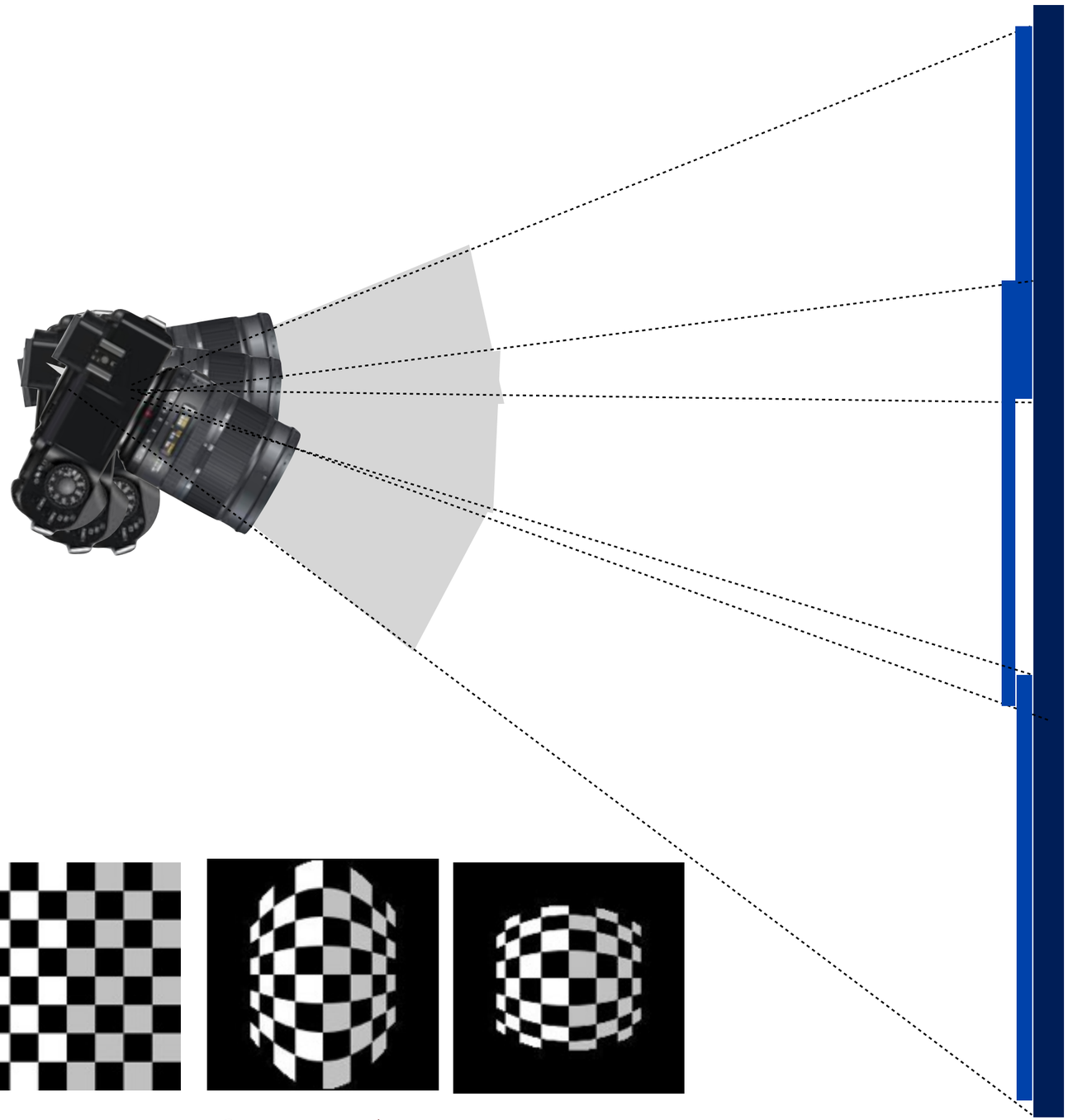
Using kolor autopano giga™ v3

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



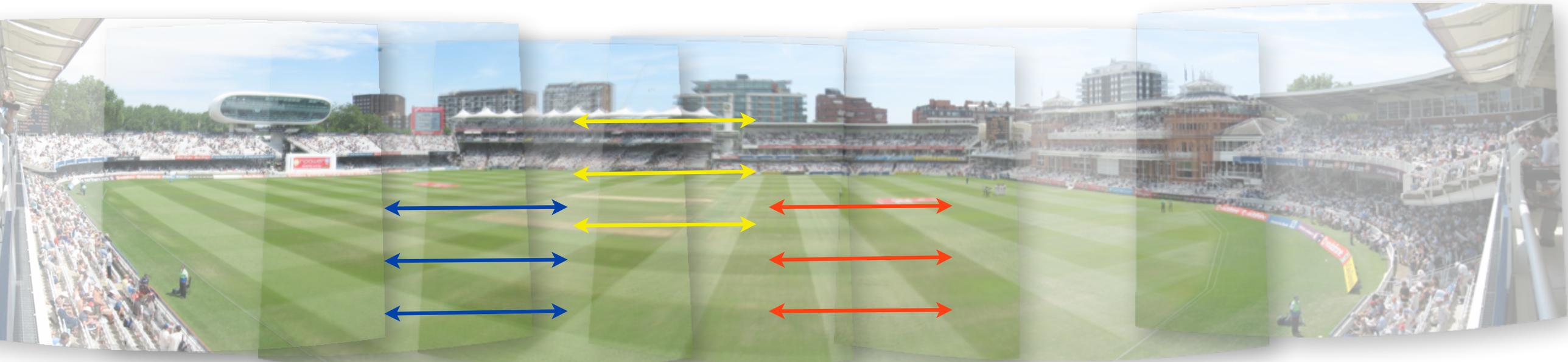
Not Just a Plane

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

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

High Dynamic Range Photography

- Importance and Characteristics of High Dynamic Range Images
- Merging Images to Create a High Dynamic Range Photograph

Georgia Tech College of Computing

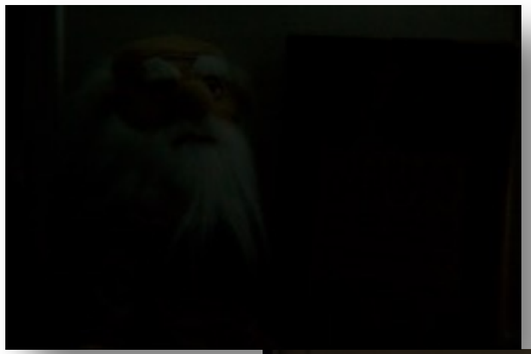
1. Dynamic Range
2. Digital cameras do not encode Dynamic Range very well
3. Image Acquisition Pipeline for Capturing Scene Radiance to Pixel Values
4. Linear and non-linear aspects inherent in the Image Acquisition Pipeline

Lesson Objectives

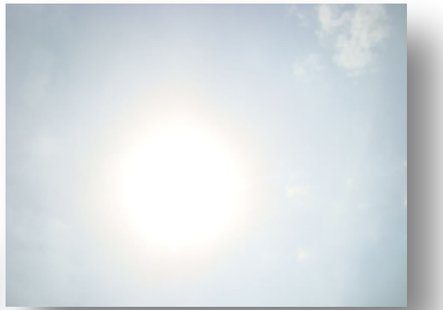
5. Camera Calibration
6. Pixel Values from different Exposure Images are used to render a Radiance Map of Scene
7. Tone Mapping

Dynamic Range in Real World

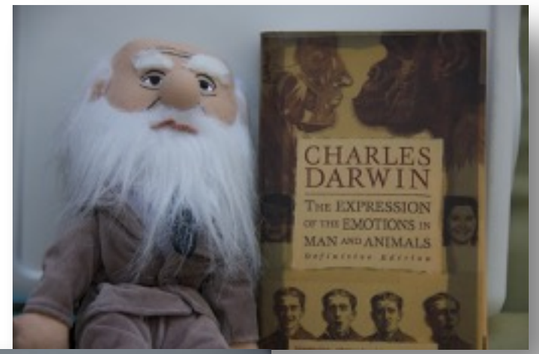
Inside, No Lights
Long Exposure

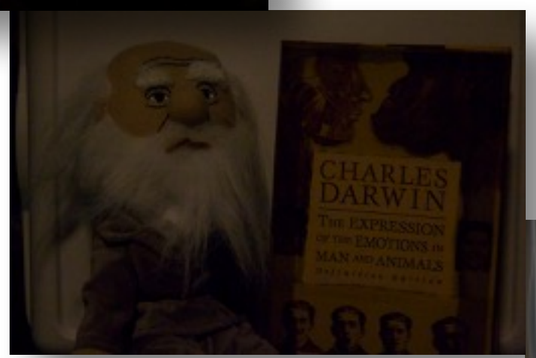


Into the Sun

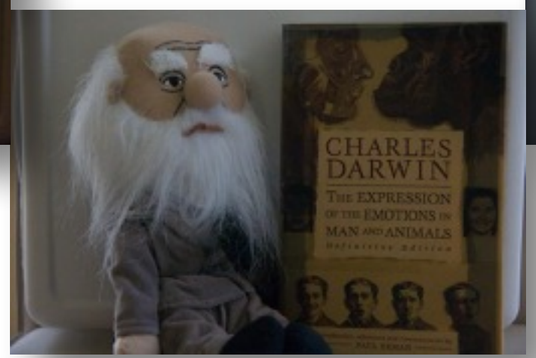


Outside,
in the Sun





Inside,
Incandescent Light

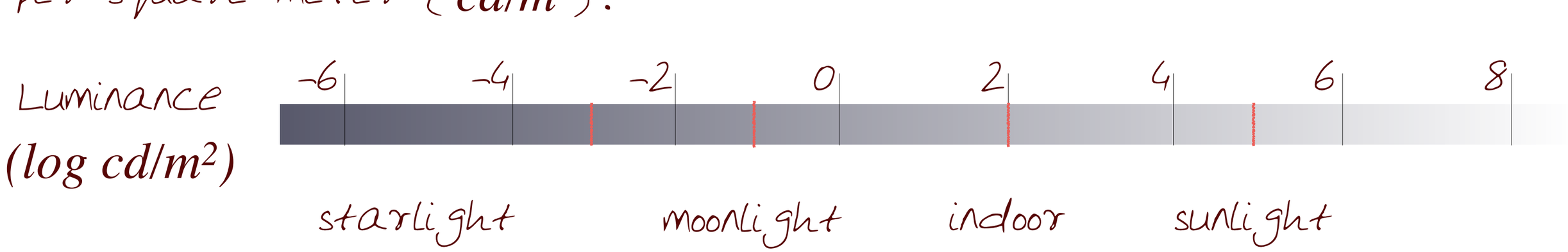


Inside, Near Window
(Natural Light)

Outside,
Under Shade

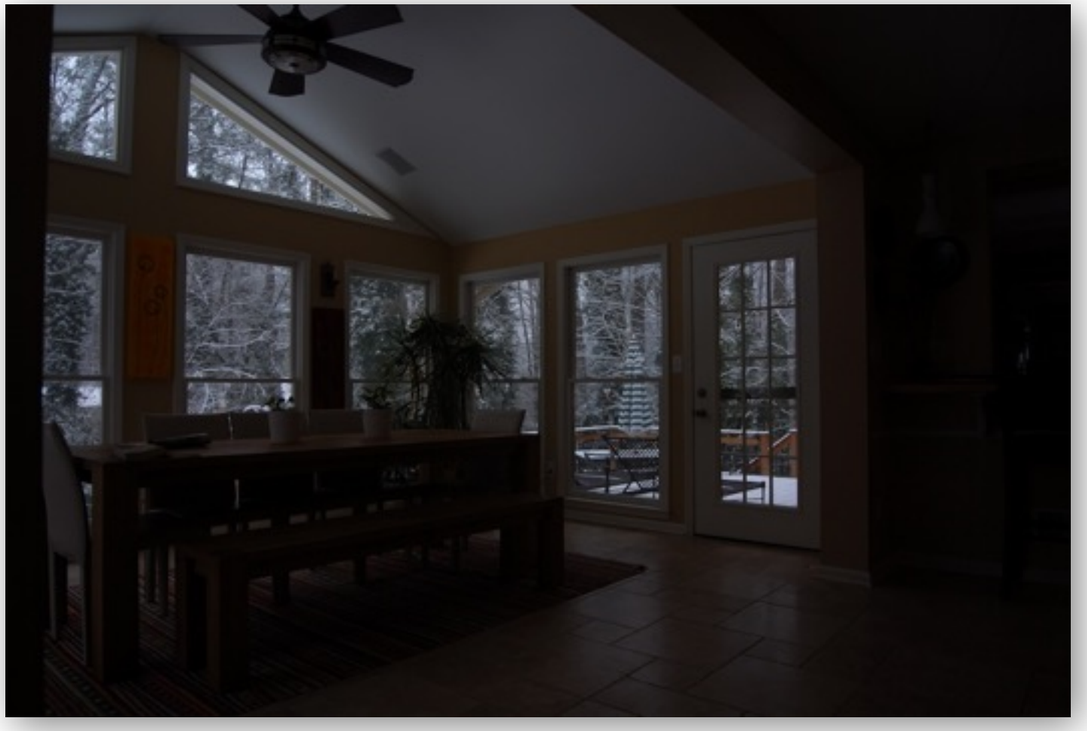
Dynamic Range

Luminance: A photometric measure of the luminous intensity per unit area of light traveling in a given direction. Measured in candela per square meter (cd/m^2).



- Human Static Contrast Ratio: 100:1 (10^2:1) → about 6.5 f-stops
- Human Dynamic Contrast Ratio: 1,000,000:1 (10^6:1) → about 20 f-stops

Limited Dynamic Range of Current Cameras

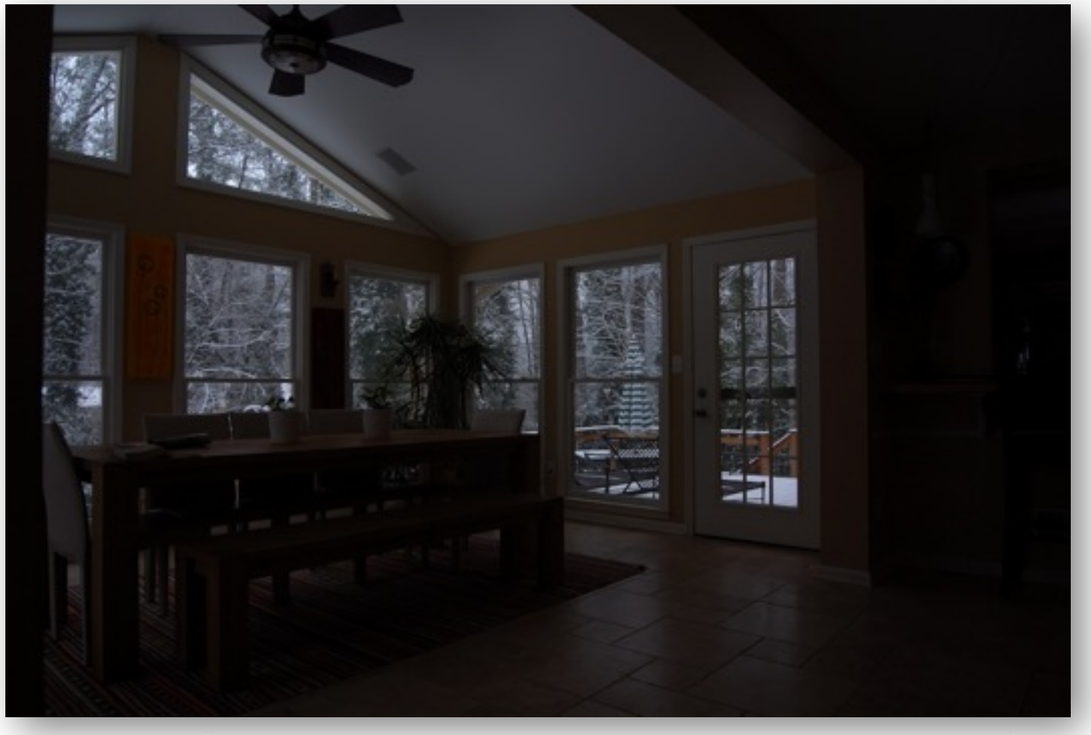


Short Exposure: Snow and Outside Visible

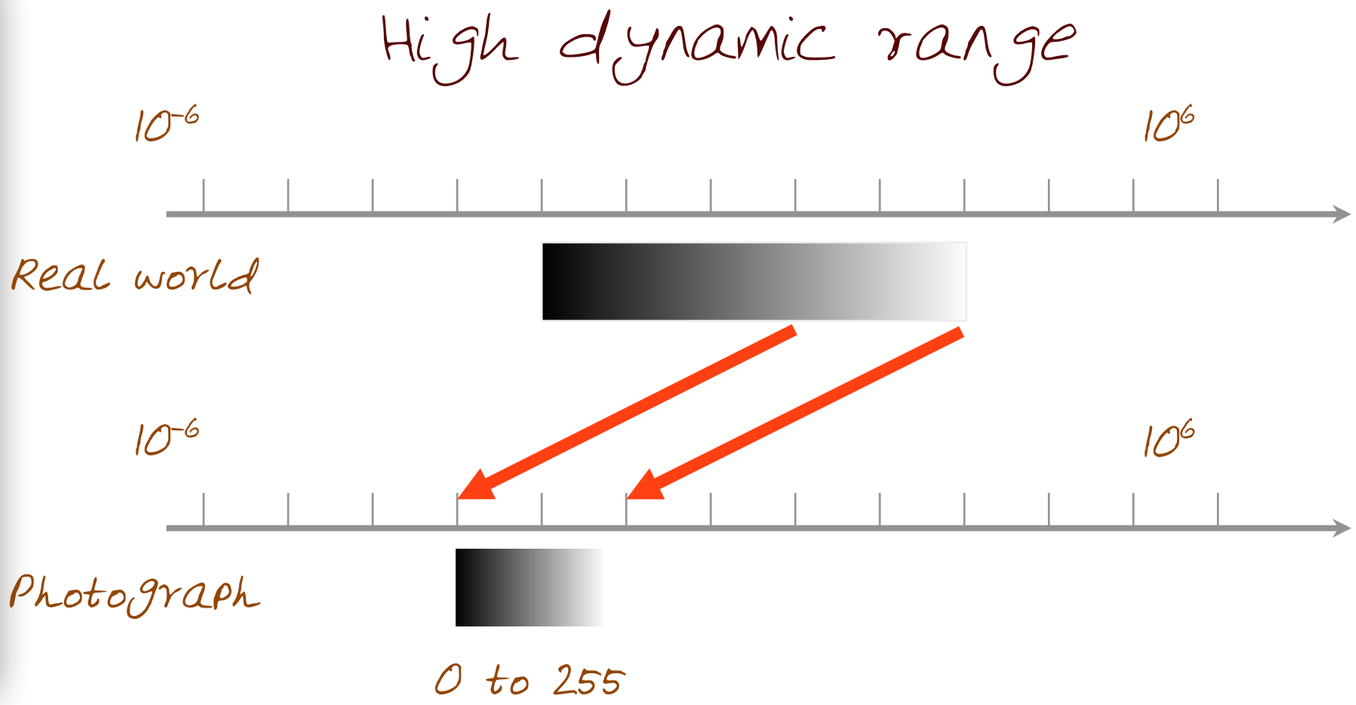
Long Exposure: Inside Visible

* Need about 5-10 million values to store all brightnesses around us
* 8-bit images provide only 256 values!!

Limited Dynamic Range of Current Cameras

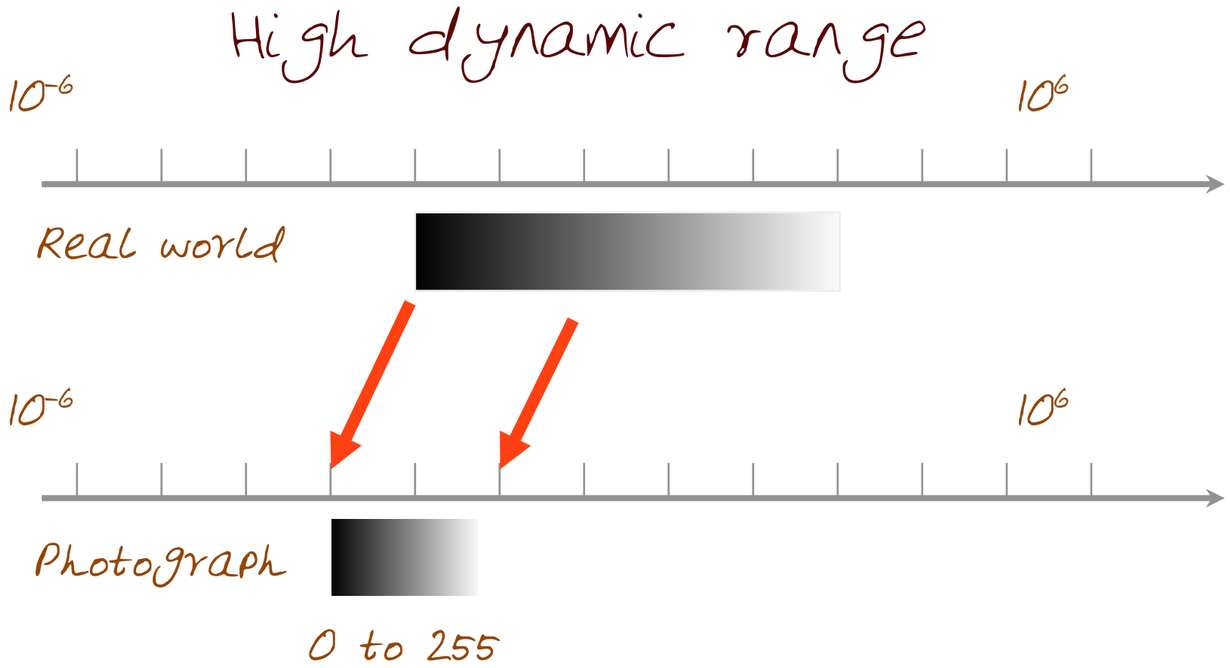


Short Exposure: Snow and Outside Visible



* Need about 5-10 million values to store all brightnesses around us.
* 8-bit images provide only 256 values!!

Limited Dynamic Range of Current Cameras

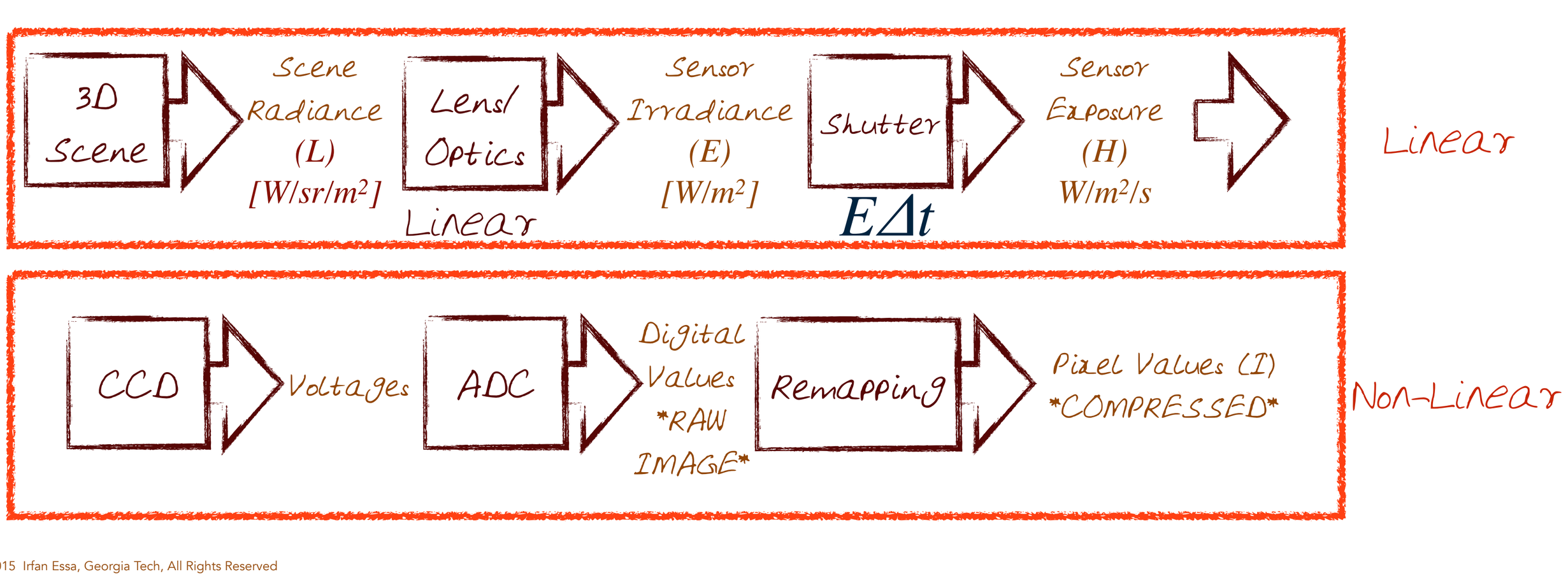


Long Exposure: Inside Visible

* Need about 5-10 million values to store all brightnesses around us.
* 8-bit images provide only 256 values!!

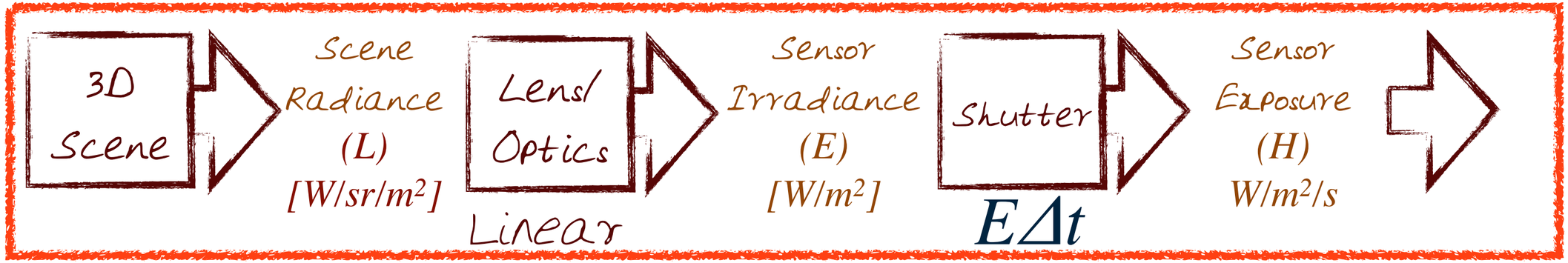
Relationship Between Image and Scene Brightness

The Image Acquisition Pipeline



Relationship Between Image and Scene Brightness

$g: L \rightarrow E \rightarrow H \rightarrow I \quad \Longleftrightarrow \quad g^{-1}: I \rightarrow E \rightarrow H \rightarrow L$

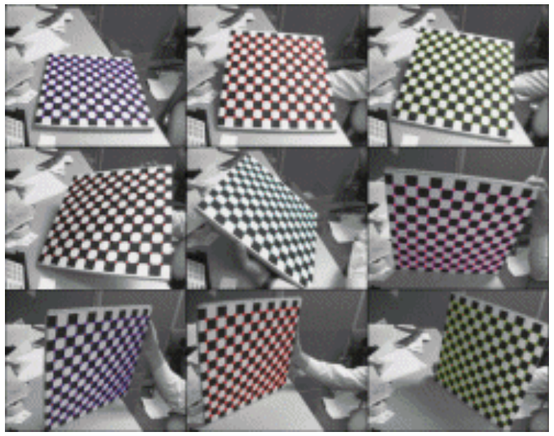


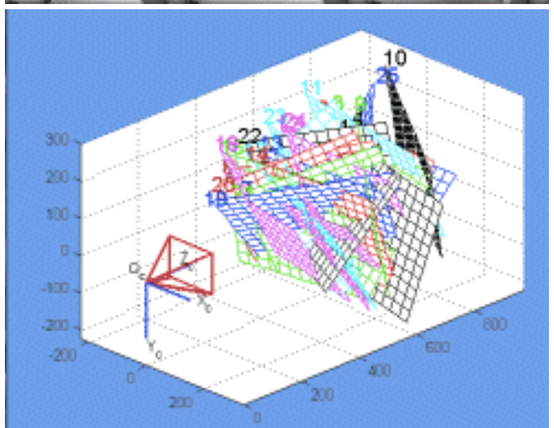
Camera Calibration

* Geometric
  * How pixel coordinates relate to directions in the world
* Radiometric / Photometric
  * How pixel values relate to radiance amounts in the world

Camera Calibration

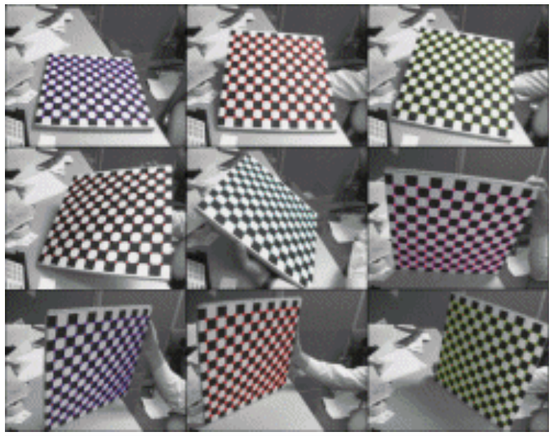
* Geometric
  * How pixel coordinates relate to directions in ~~the world~~ other images
* Radiometric / Photometric
  * How pixel values relate to radiance amounts in the world

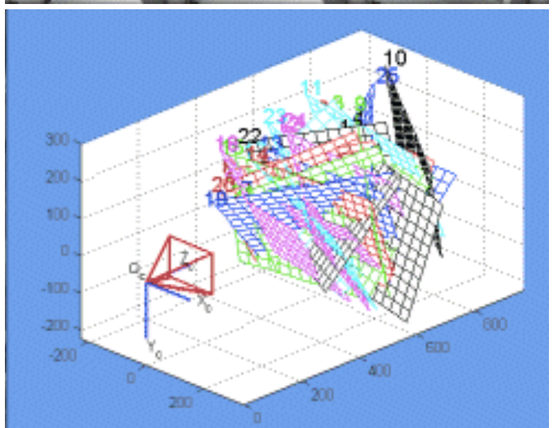




Camera Calibration

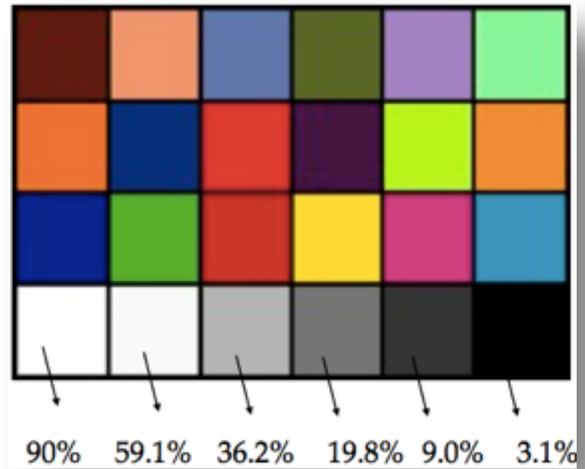
* Geometric
  * How pixel coordinates relate to directions in ~~the world~~ other images
* Radiometric / Photometric
  * How pixel values relate to radiance amounts in ~~the world~~ other images



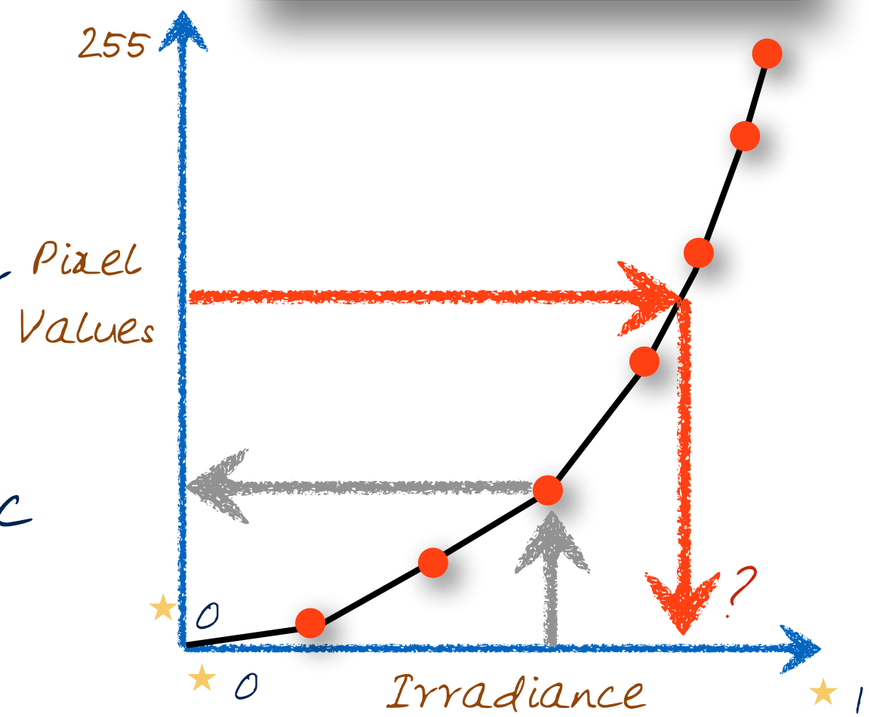


Radiometric Calibration

$g: L \rightarrow E \rightarrow H \rightarrow I \Leftrightarrow g^{-1}: I \rightarrow E \rightarrow H \rightarrow L$



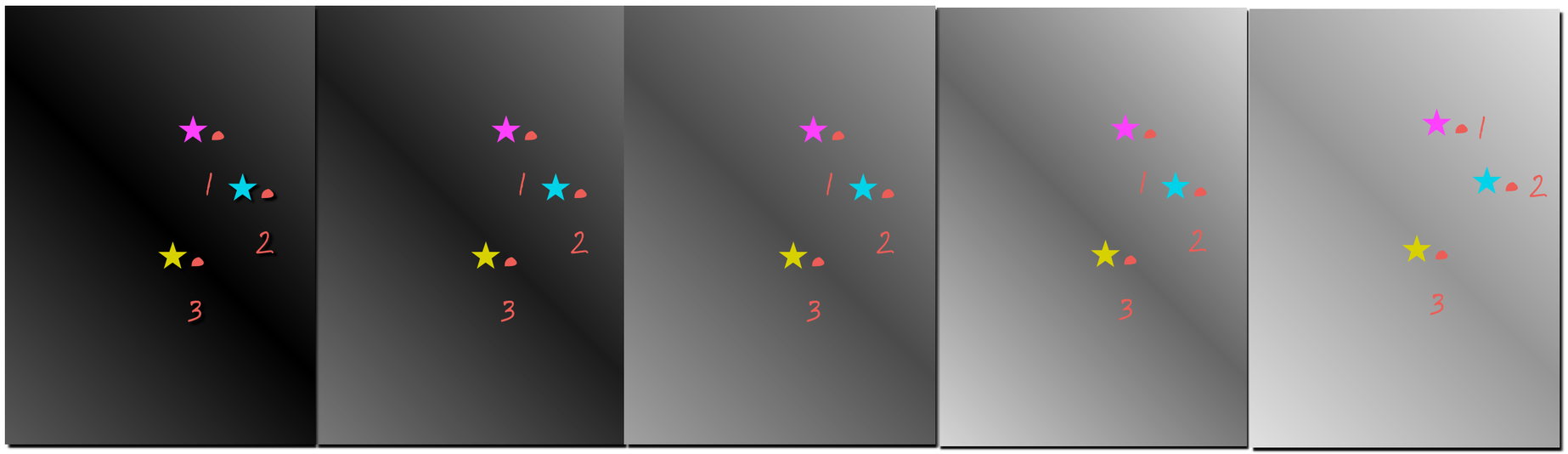
- A Color Chart with know reflectances
- Multiple camera exposures to fill up the curve
- Method assumes constant lighting on all patches and works best when source is far away (example sunlight)
- Unique inverse exists because g is monotonic and smooth for all cameras



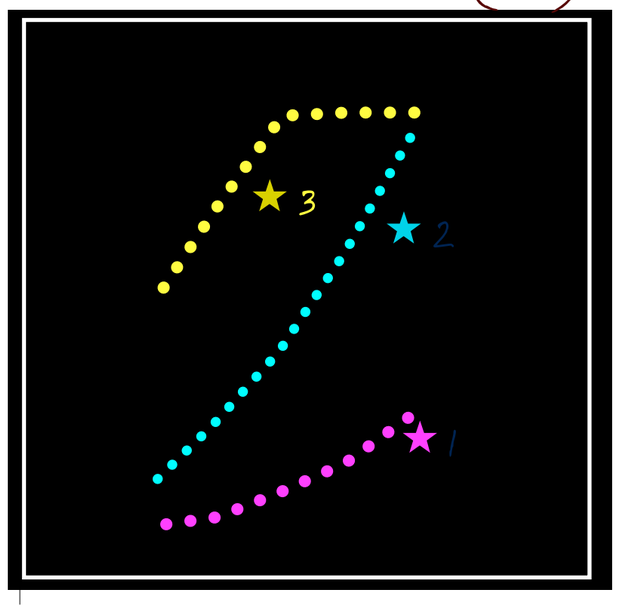
(Grossberg and Nayar 2003)

A Sequence of Images of Different Exposures

Series of Images



$\Delta t = 1/64$ sec $\Delta t = 1/16$ sec $\Delta t = 1/4$ sec $\Delta t = 1$ sec $\Delta t = 4$ sec



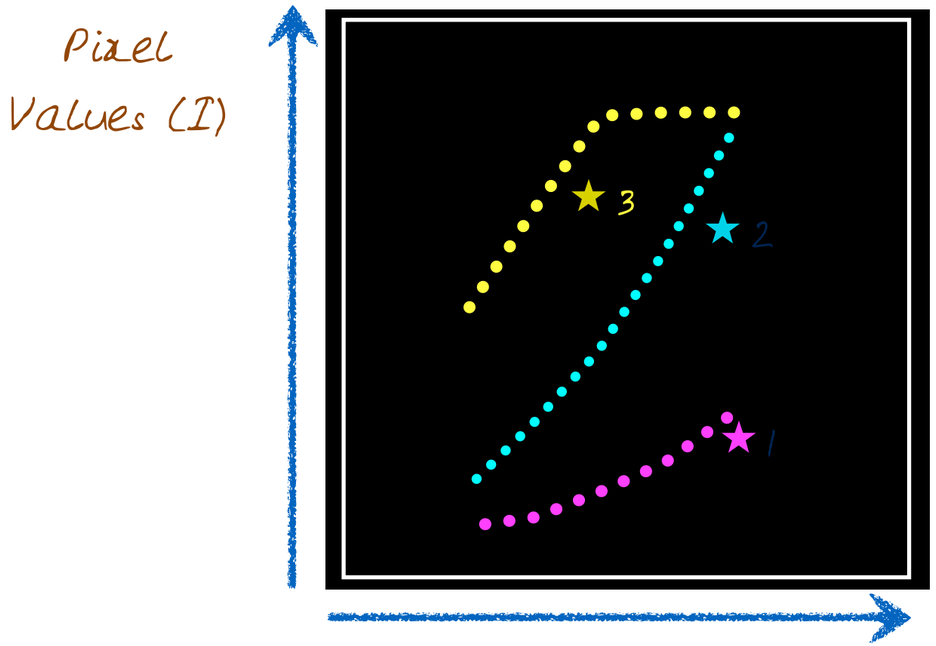
$$\text{Pixel Values } (I) = g(\text{Exposure})$$

$$\text{Exposure } (H) = \text{Irradiance } (E) * \Delta t$$

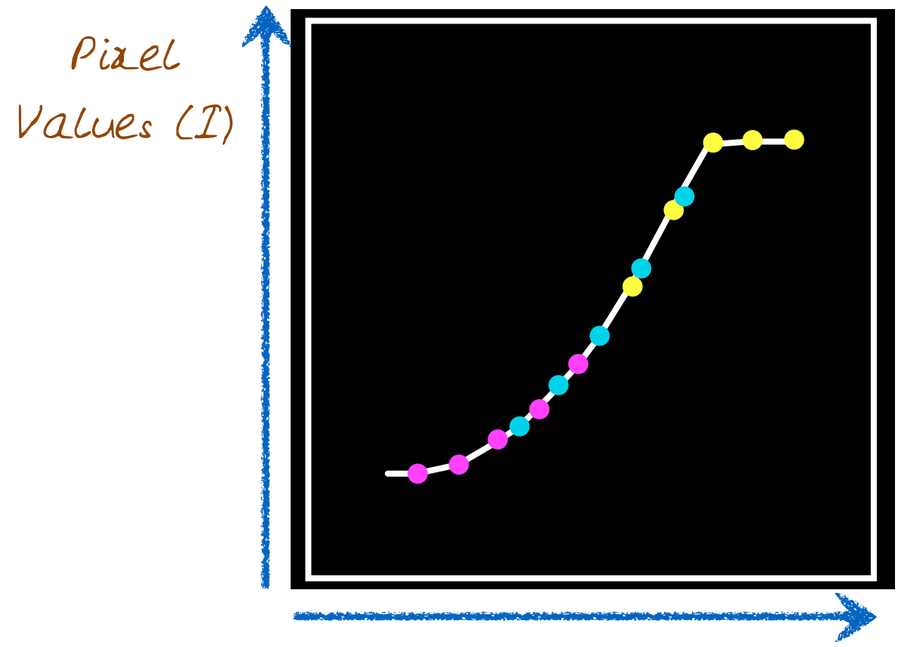
$$\log \text{Exposure } (H) = \log \text{Irradiance } (E) + \log \Delta t$$

Debevec and Malik 1997

Response Curves



Assuming unit radiance for each pixel



After adjusting radiances to obtain a smooth response curve

How to Compute

* Let $g(z)$ be the discrete inverse response function
* For each pixel site i in each image j, compute

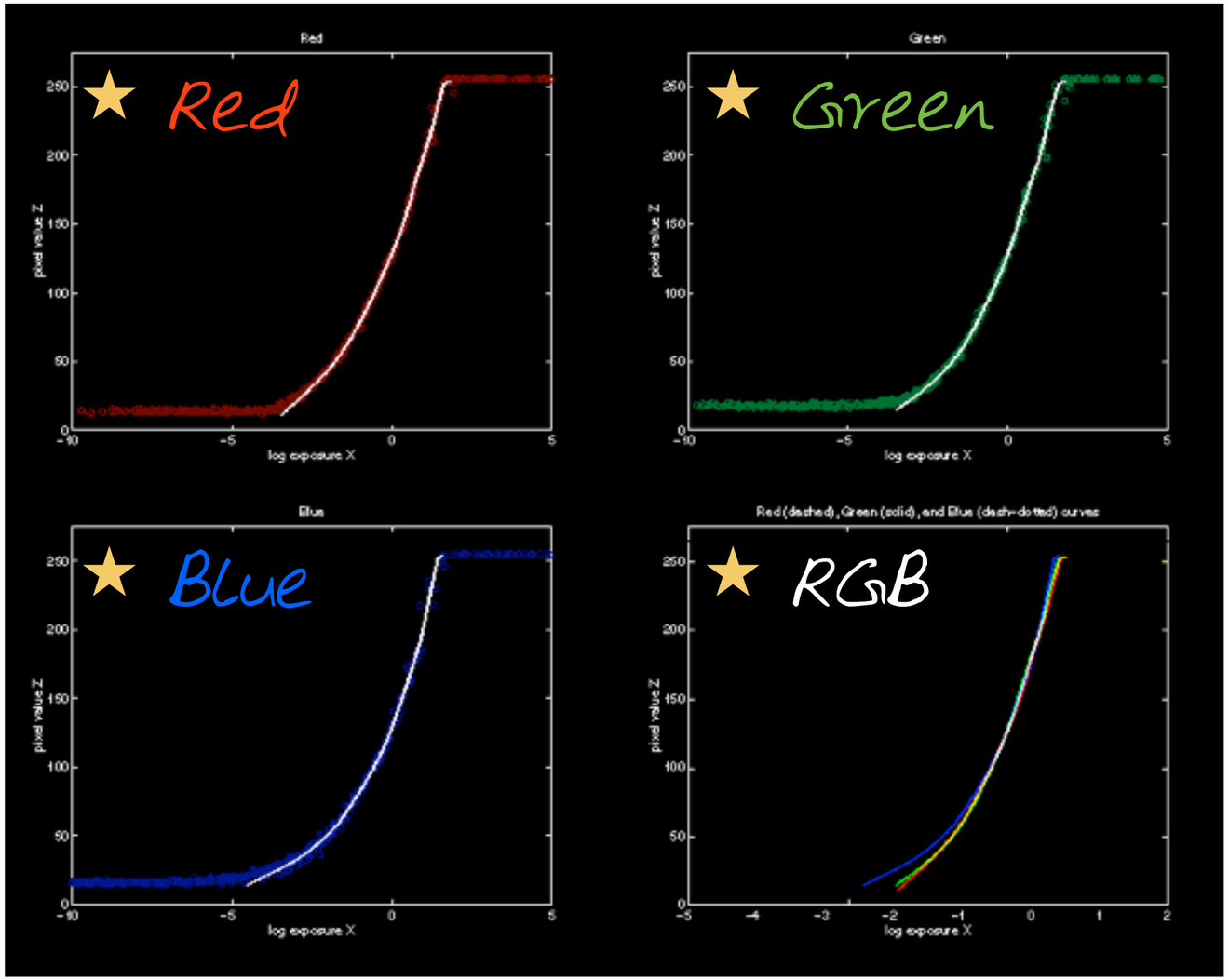
$$\ln(E_i) + \ln(\Delta t_j) = g(Z_{ij})$$

* Solve the overdetermined linear system for N pixels over P different exposure images:

$$\underbrace{\sum_{i=1}^{N}\sum_{i=1}^{P}\left[\ln(E_i) + \ln(\Delta t_j) - g(Z_{ij})\right]^2}_{\text{Fitting Term}} + \underbrace{\lambda \sum_{z=Z_{min}}^{Z_{max}} g''(z)^2}_{\text{Smoothness Term}}$$

See Debevec and Malik (1997) for more details

Response Curves



(Not actual curves for these images, used here just for demonstration)

Radiance Map

Need a New File Format

Radiance Format

32 bits / pixel

Red Green Blue Exponent

★ (145, 215, 87, 149) =

★ (145, 215, 87) * 2^(149-128) =

★ (1190000, 1760000, 713000)

★ (145, 215, 87, 103) =

★ (145, 215, 87) * 2^(103-128) =

★ (0.00000432, 0.00000641, 0.00000259)

Ward (2001), There are many other formats too

Now to Display it!

Tone Mapping

- Map one set of colors to another
- Displaying on a medium that has limited dynamic range
  - Printers, monitors, and projectors all have a limited dynamic range
- Inadequate to reproduce the full range of light intensities present in natural scenes

★ http://commons.wikimedia.org/wiki/File:Kanitz-Kyawsche_Gruft_in_Hainewalde_HDR.jpg

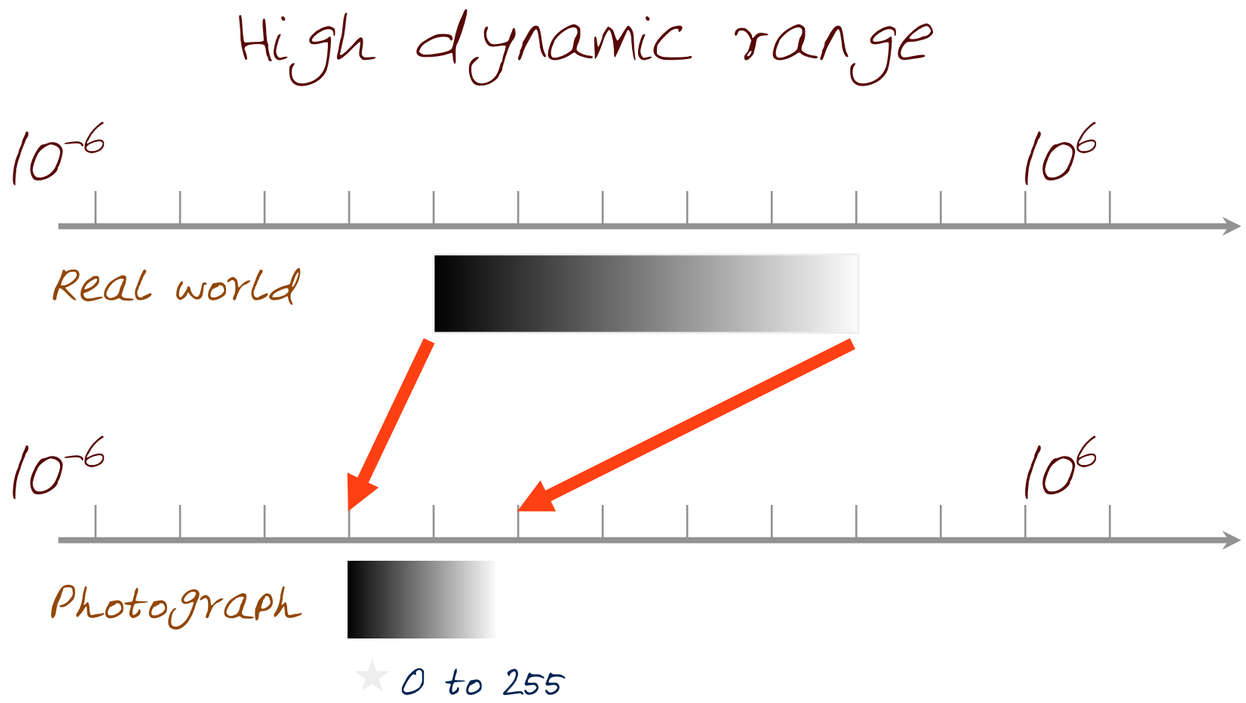
Tone Mapping

- Addresses the problem of
  - strong contrast reduction from the scene radiance to the displayable range
  - preserves the image details and color appearance
- Many well-known Algorithms exist for this
- See Banterle, et al. (2011), Reinhard et al. (2002) and Durand and Dorsey (2002)

★ http://commons.wikimedia.org/wiki/File:Kanitz-Kyawsche_Gruft_in_Hainewalde_HDR.jpg

Tone Mapping

★ http://enwikipedia.org/wiki/File:Dundus_Square.jpg



* Match limited contrast of the medium
* Preserve details

* Use filtering approaches to "compress" locally and globally

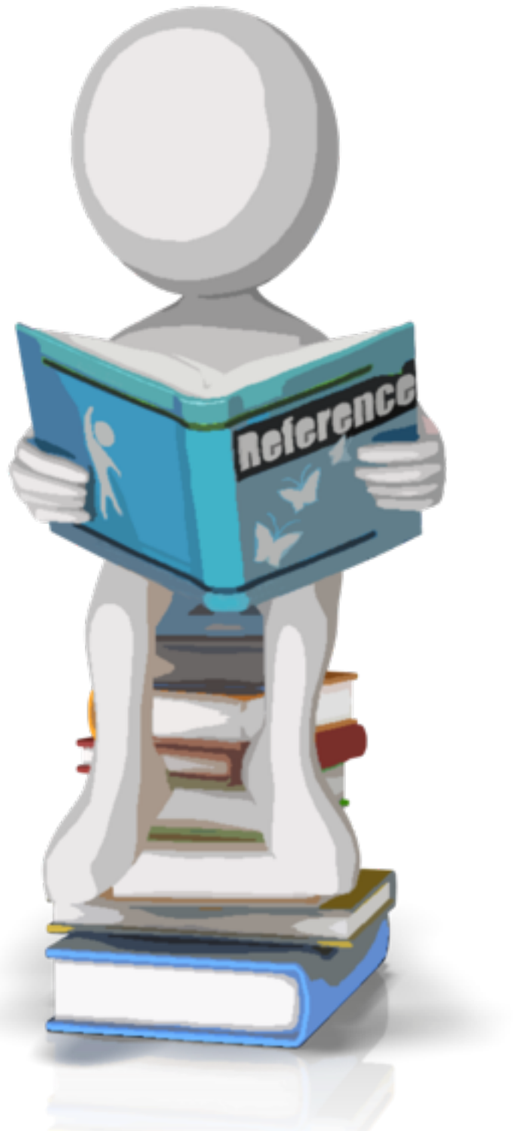
Summary

* Discussed issues of Dynamic Range
* Reviewed the Image Acquisition Pipeline for Capturing Scene Radiance to Pixel Values
* Discussed the linear and non-linear aspects of the Image Acquisition Pipeline for Capturing Scene Radiance to Pixel Values

Summary

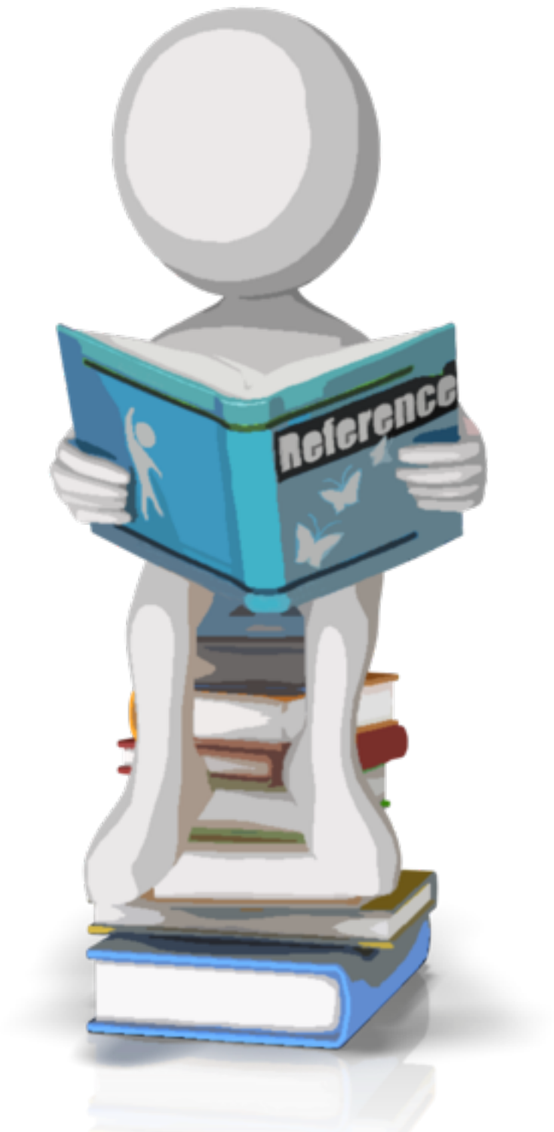
* Introduced the need for Camera Calibration just from other images
* Presented the methods for going Pixel Values from different Exposure Images to render a Radiance Map of a Scene
* Introduced the concept of Tone Mapping

Further Information



* Grossberg and Nayar (2003), "Determining the Camera Response from Images: What is Knowable?," IEEE Transactions on Pattern Analysis and Machine Intelligence,
* Debevec and Malik (1997). "Recovering High Dynamic Range Radiance Maps from Photographs." In SIGGRAPH 1997
* Ward (2001), "High Dynamic Range Imaging," Proceedings of the Ninth Color Imaging Conference, November 2001.

Further Information



- Durand and Dorsey (2002), "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images" In SIGGRAPH 2002.
- Reinhard, Stark, Shirley and Ferwerda (2002), "Photographic Tone Reproduction for Digital Images", In SIGGRAPH 2002.
- Banterle, Artusi, Debattista, and Chalmers (2011) Advanced High Dynamic Range Imaging CRC Press. (with Matlab Code)
- Many Software suites on the Internet.
- Also, look for "Exposure Fusion"

Credits

* Softwares used
  * Matlab by Mathwork's Inc.
* For more information, see
  * Richard Szeliski (2010) Computer Vision: Algorithms and Applications, Springer.
* Some concepts in slides motivated by similar slides by J. Hays.
* Photographs by Irfan Essa

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

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

Stereo Photography / Imagery

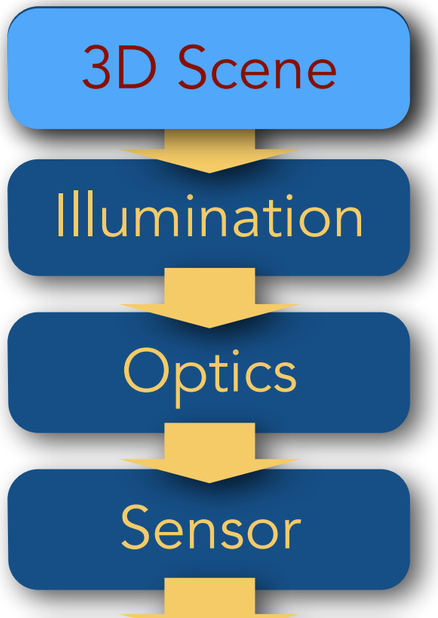
* Depth / Range Imaging
* Structure / 3D Scenes
* Stereo . . Depth from 2 views

Georgia Tech College of Computing

Lesson Objectives

1. Geometry (Depth/ Structure) in a Scene
2. Stereo
3. Parallax
4. Compute Depth from a stereo image pair

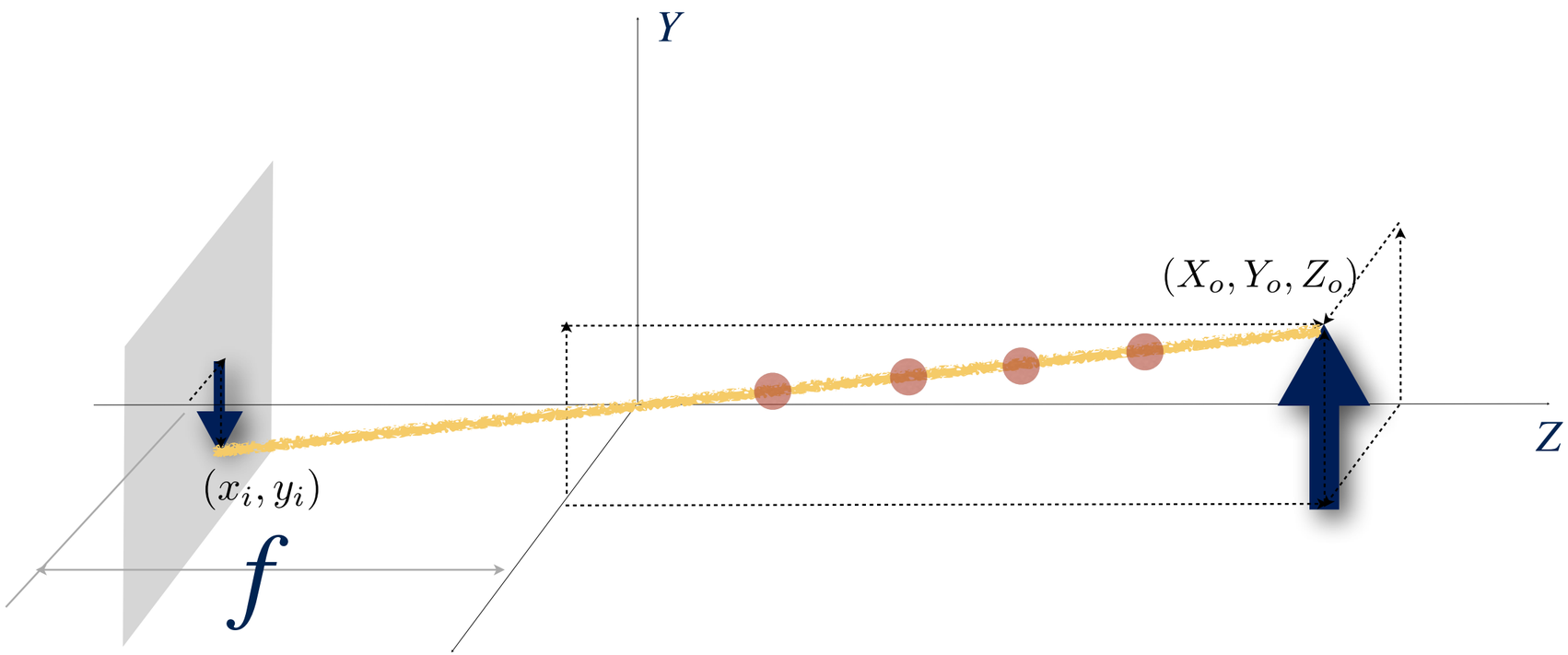
Depth (of a Scene)

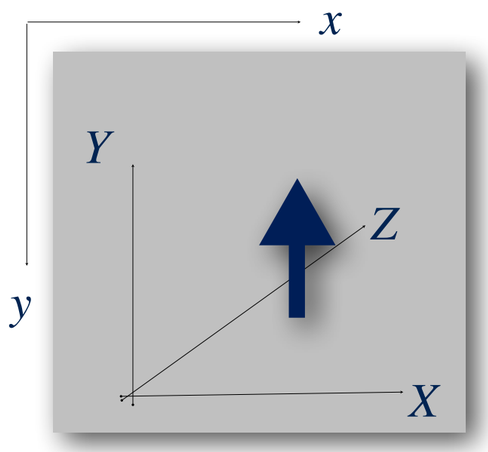


McMillan & Gortler 1999

* Above all, we are interested in capturing a 3D scene with Geometry
* Need Depth, Geometry, 3D

Compute depth / structure





$$x_i = \frac{X_o}{Z_o} f \qquad y_i = \frac{Y_o}{Z_o} f$$

$$x_i = \frac{kX_o}{kZ_o} f \qquad y_i = \frac{kY_o}{kZ_o} f$$

Fundamental Ambiguity: Any points along the same ray map to the same point in the image

Depth ambiguity

Images from S. Lazebnik and I. Essa

Depth Cues

Perspective (Vanishing Lines/points)

Depth Cues

Objects of known sizes

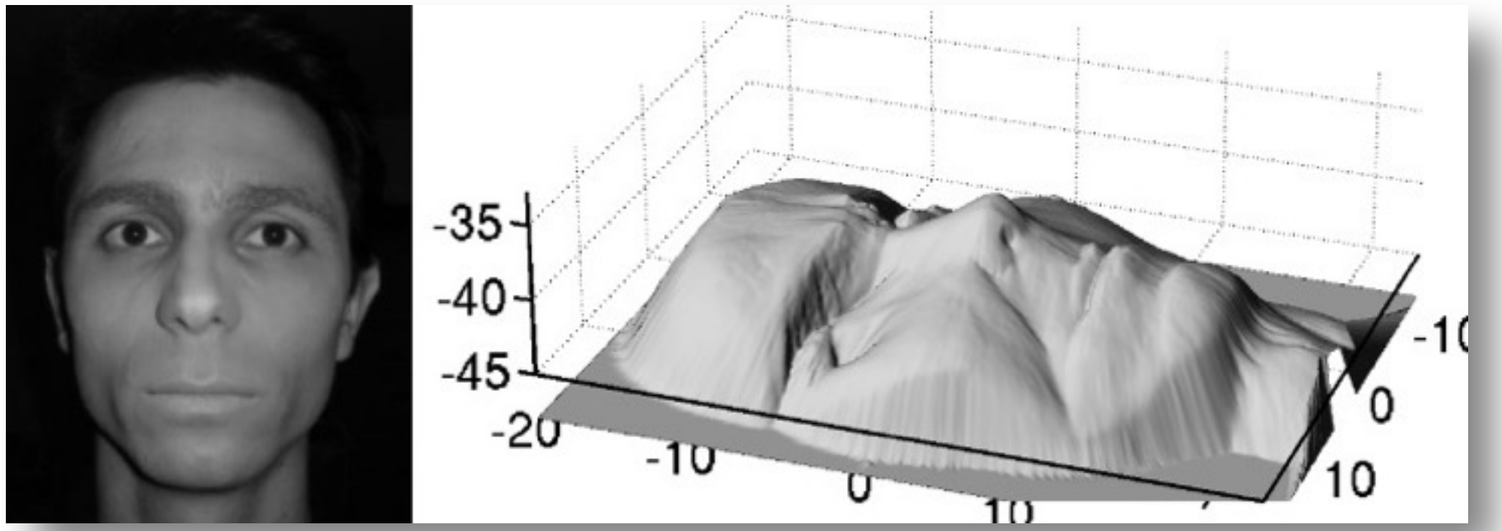
Depth Cues

Occlusions,

etc

Depth Cues

Shape from Shading

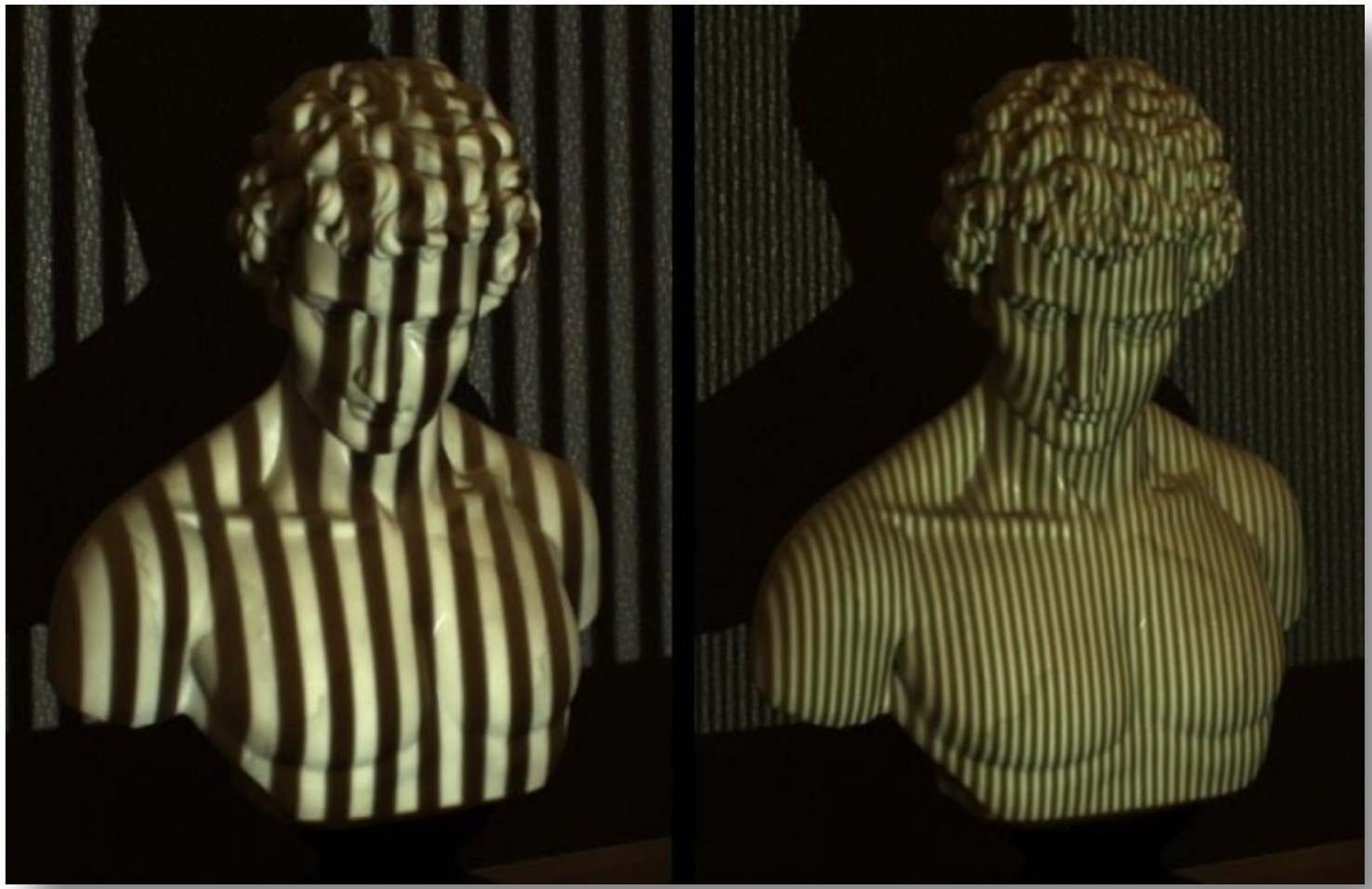


trimensional

3D Scanner for iPhone

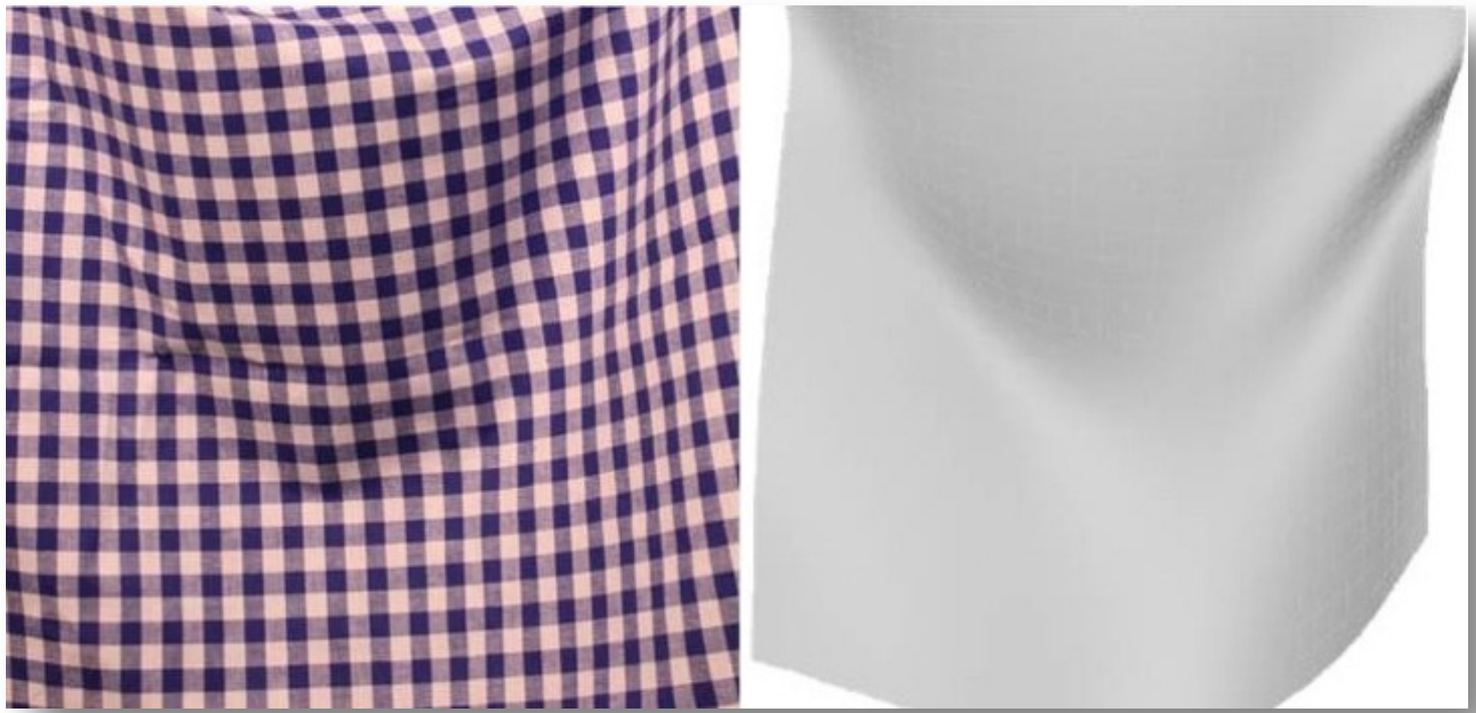
Depth Cues

Shape from Structured Light



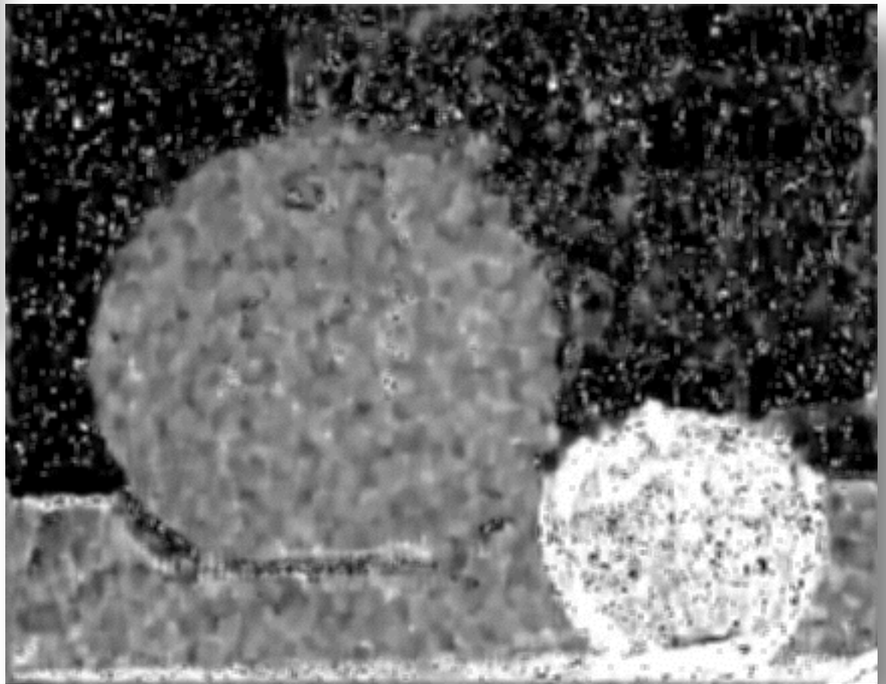
Depth Cues

Shape from Texture



Hilsmann, Schneider, Eisert (2011)

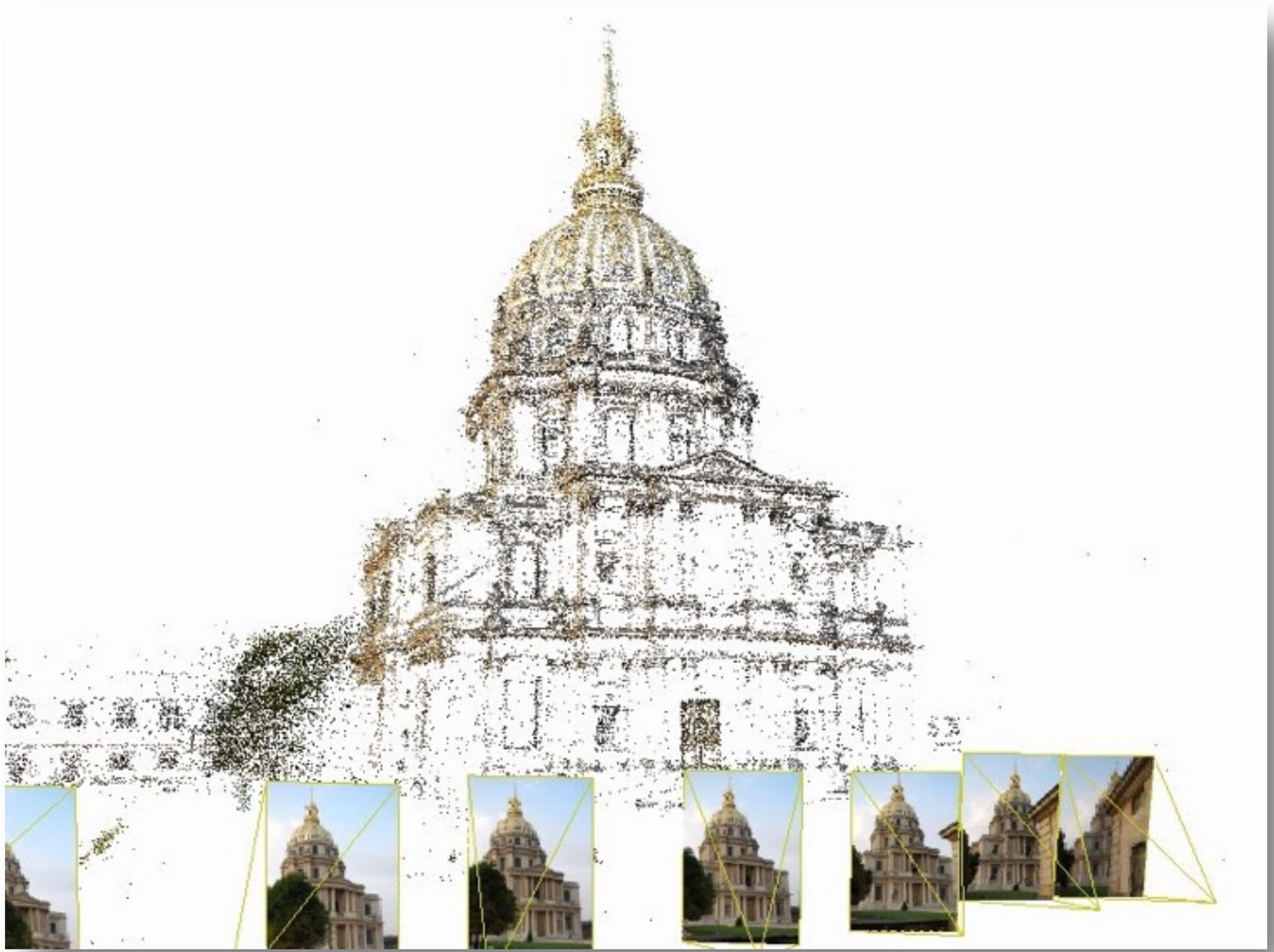
Depth Cues



Shape from De Focus

Favaro, & Soatto (1999)

Depth Cues

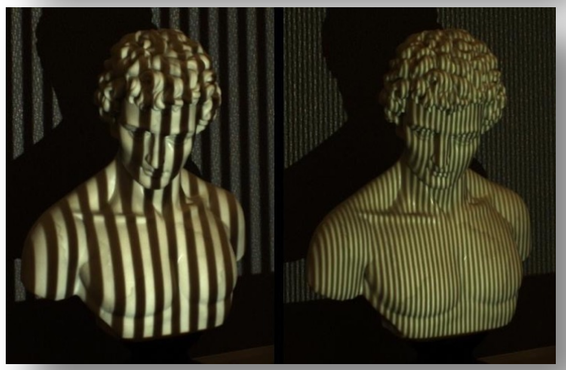


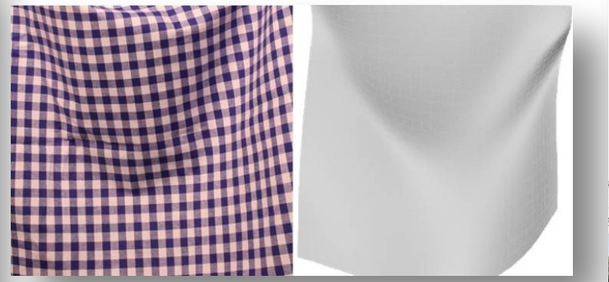
Structure from Motion

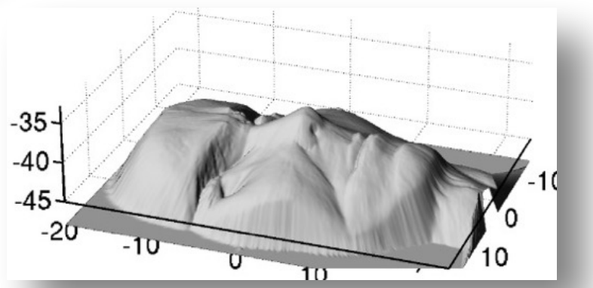
Fredriksson and Olsson (2012)

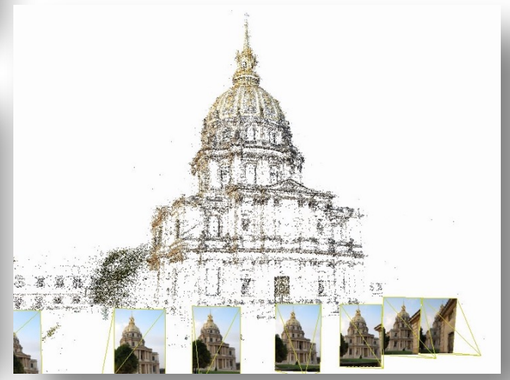
Estimating Depth / Shape From One View

- Shape from X
  - Perspective
  - Shading
  - Motion
  - Focus
  - Occlusions
  - Objects

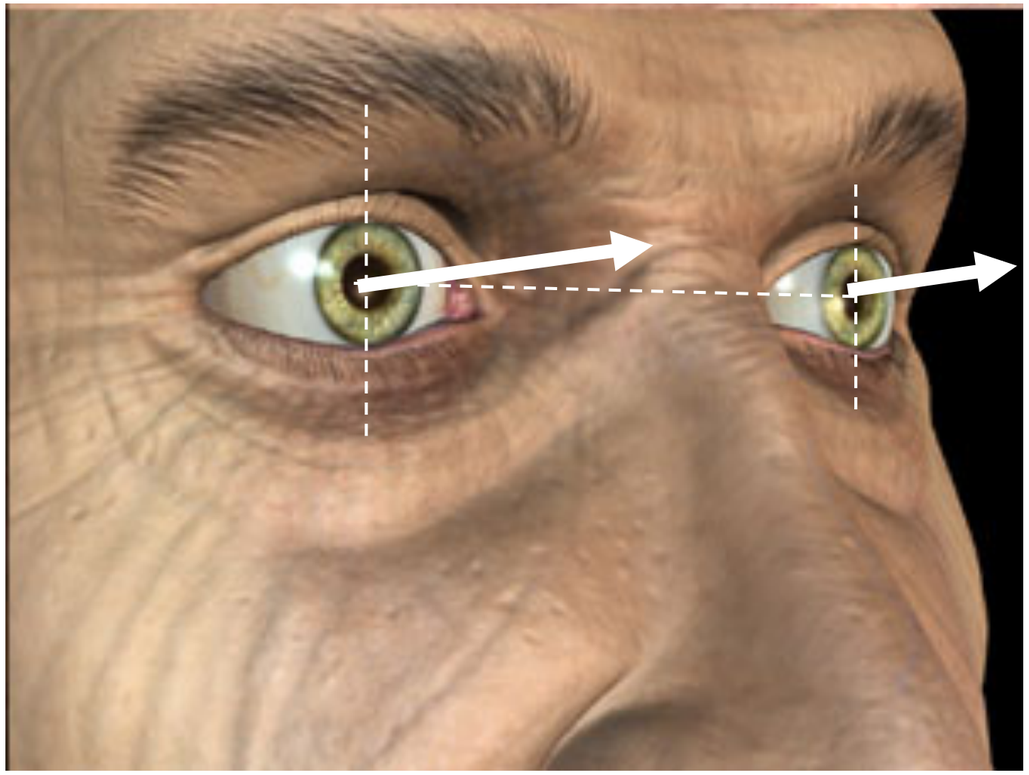


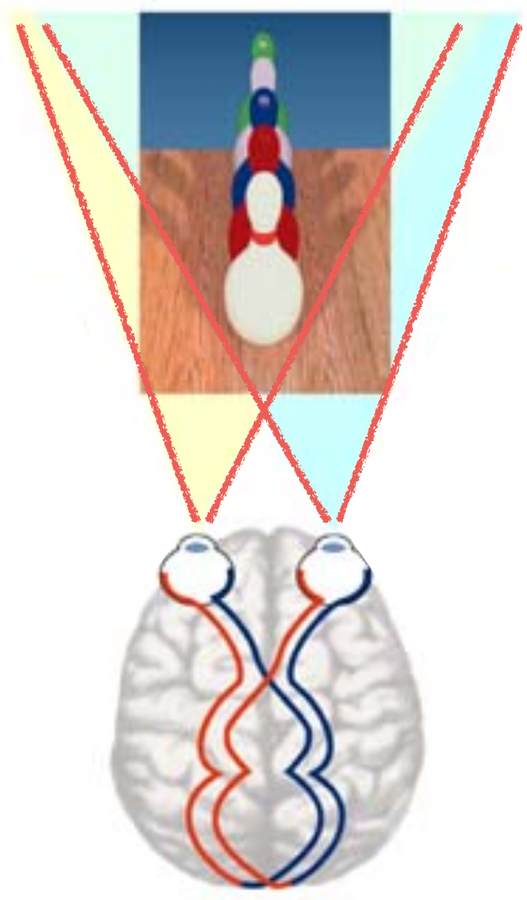






Stereo Vision

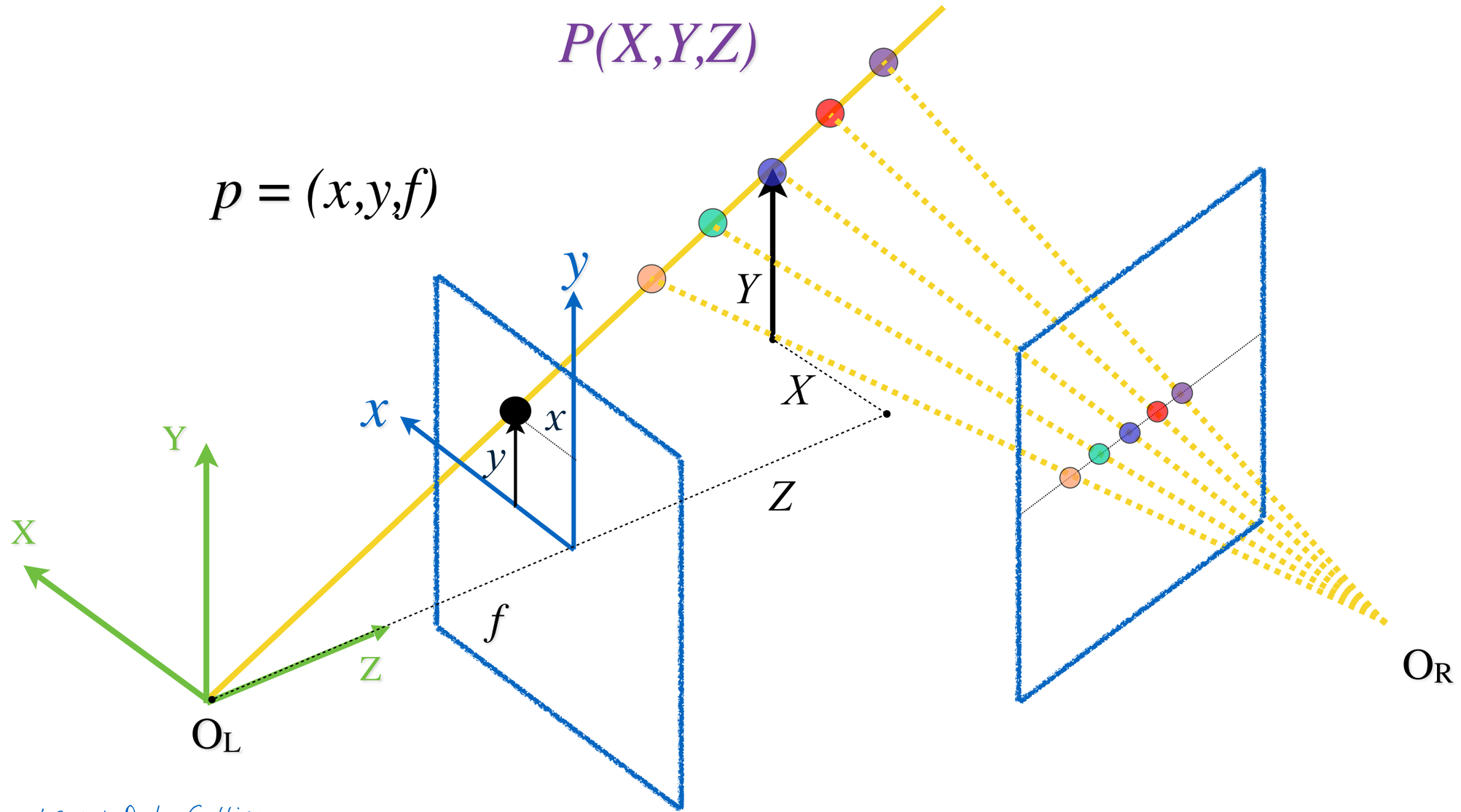




Inferring depth from images captured at the same time by two or more cameras

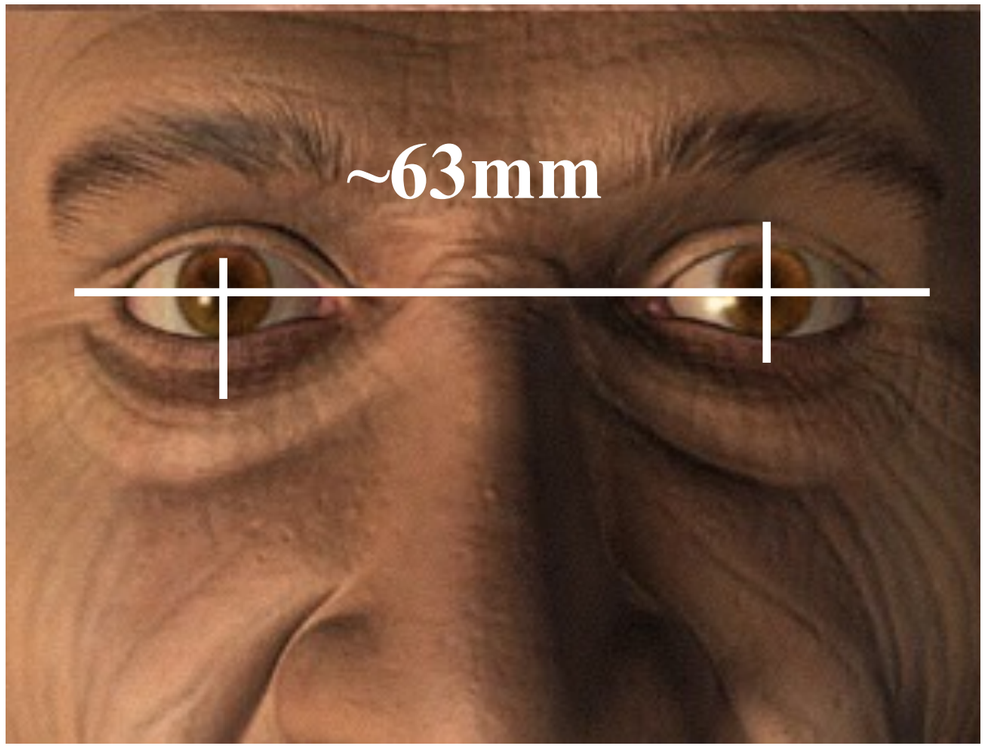
Slide courtesy: Bob Collins

Why Stereo Vision?



Slide courtesy: Bob Collins

Why Stereo Vision?



Eyes form a stereo system

The right and left eyes see the world from slightly shifted vantage points

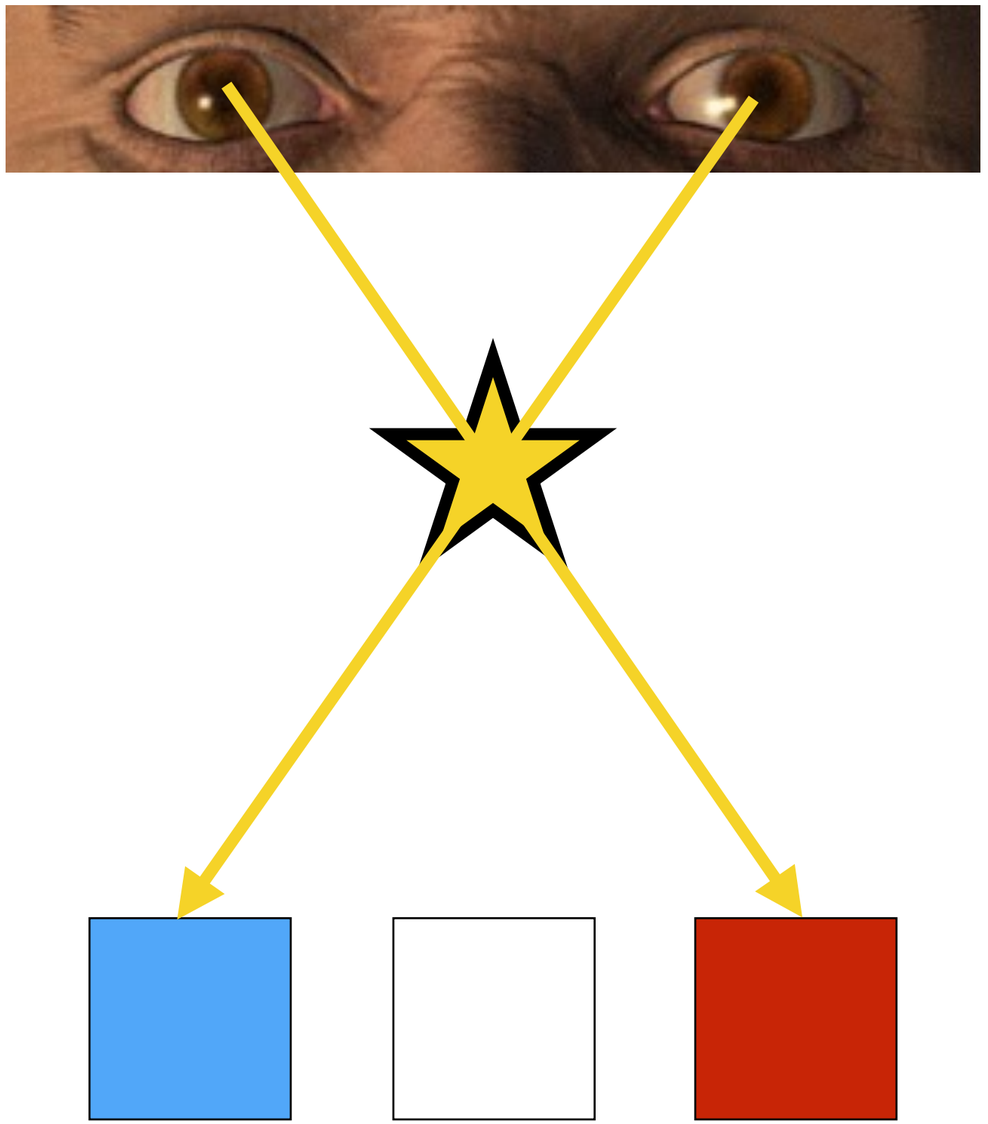
Slide courtesy: Bob Collins

Parallax

Slide courtesy: Bob Collins

Parallax

Points at different depths displace differently



Nearby points displace more than far ones

Depth via Parallax

Parallax = apparent motion of scene features located at different distances

http://4hdwallpapers.com/wp-content/uploads/2013/03/California-Mountain.jpg

Depth via Parallax

Far

Mid

Near

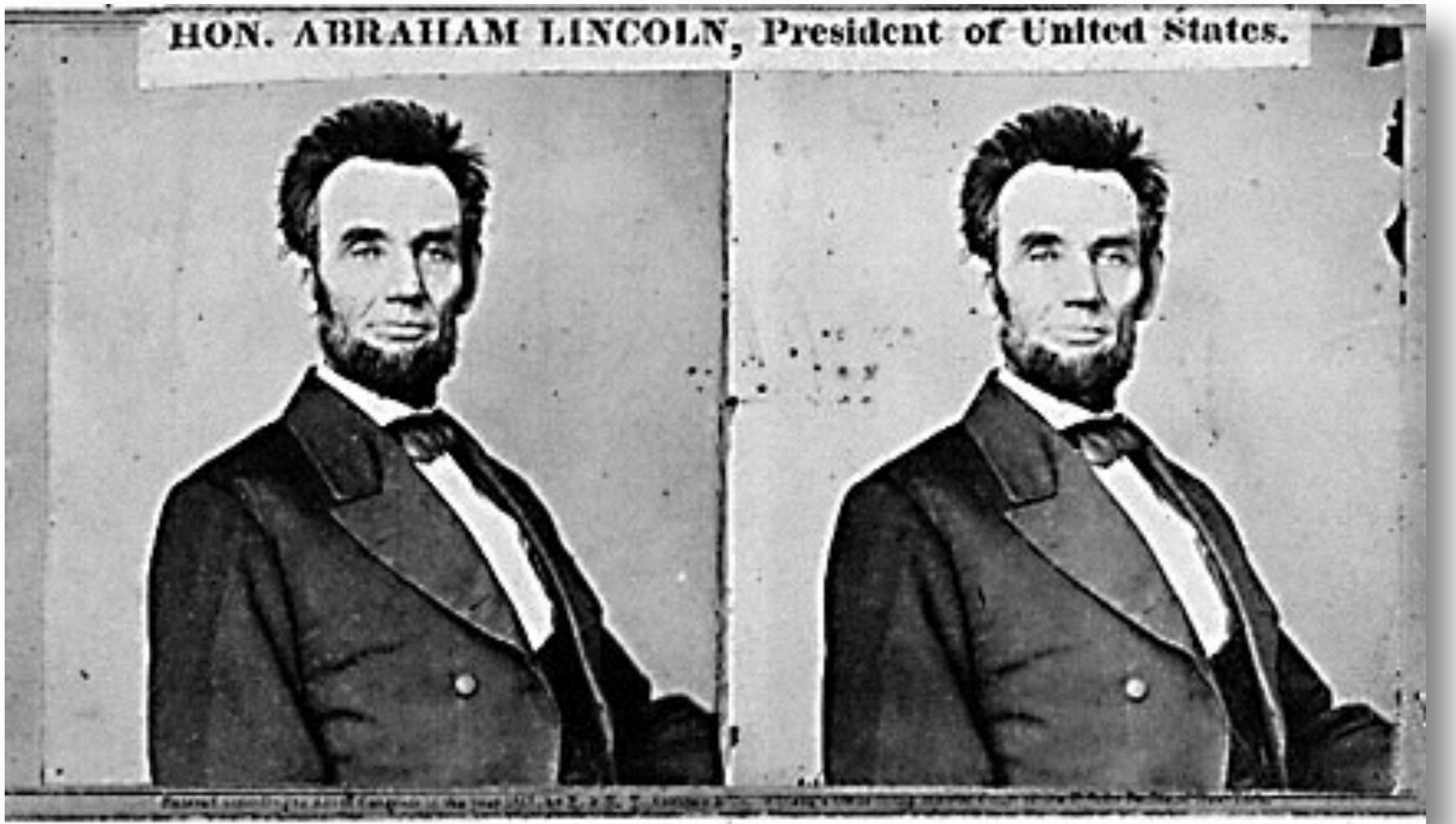
http://4hdwallpapers.com/wp-content/uploads/2013/03/California-Mountain.jpg

Stereo Photography and Stereo Viewers

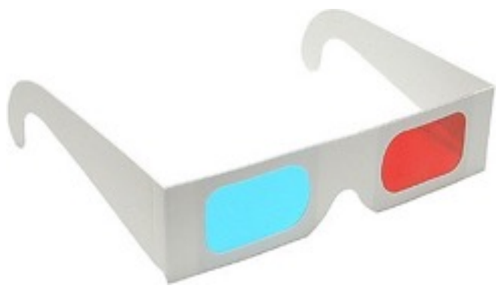
Invented by Sir Charles Wheatstone 1838

Take two pictures of the same subject from two slightly different viewpoints and display so that each eye sees only one of the images

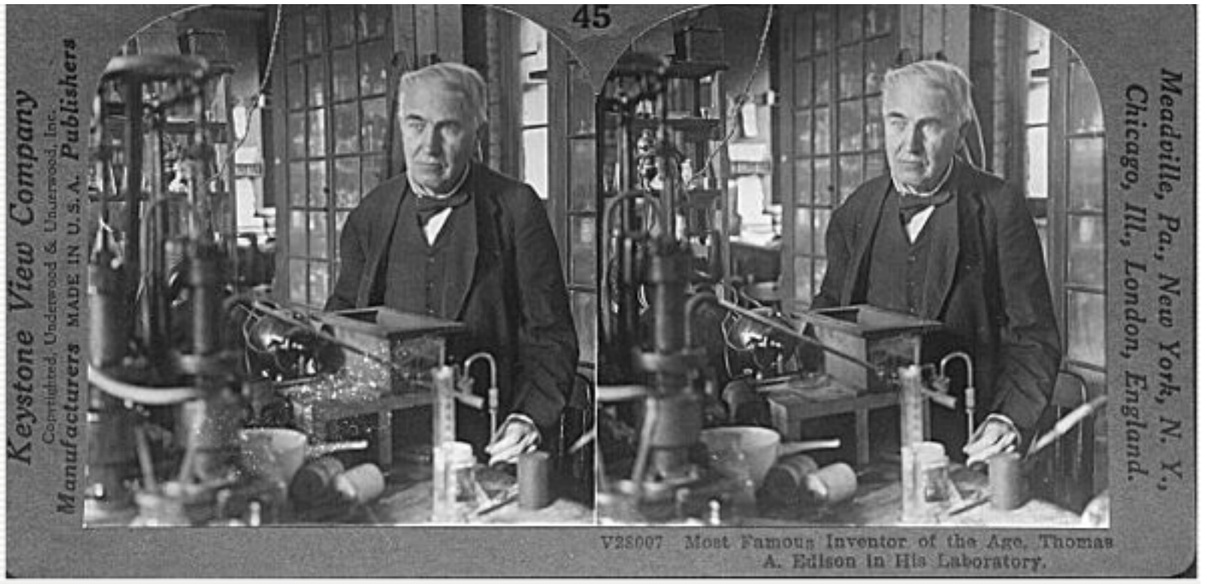
Early 3D photographs



Anaglyph



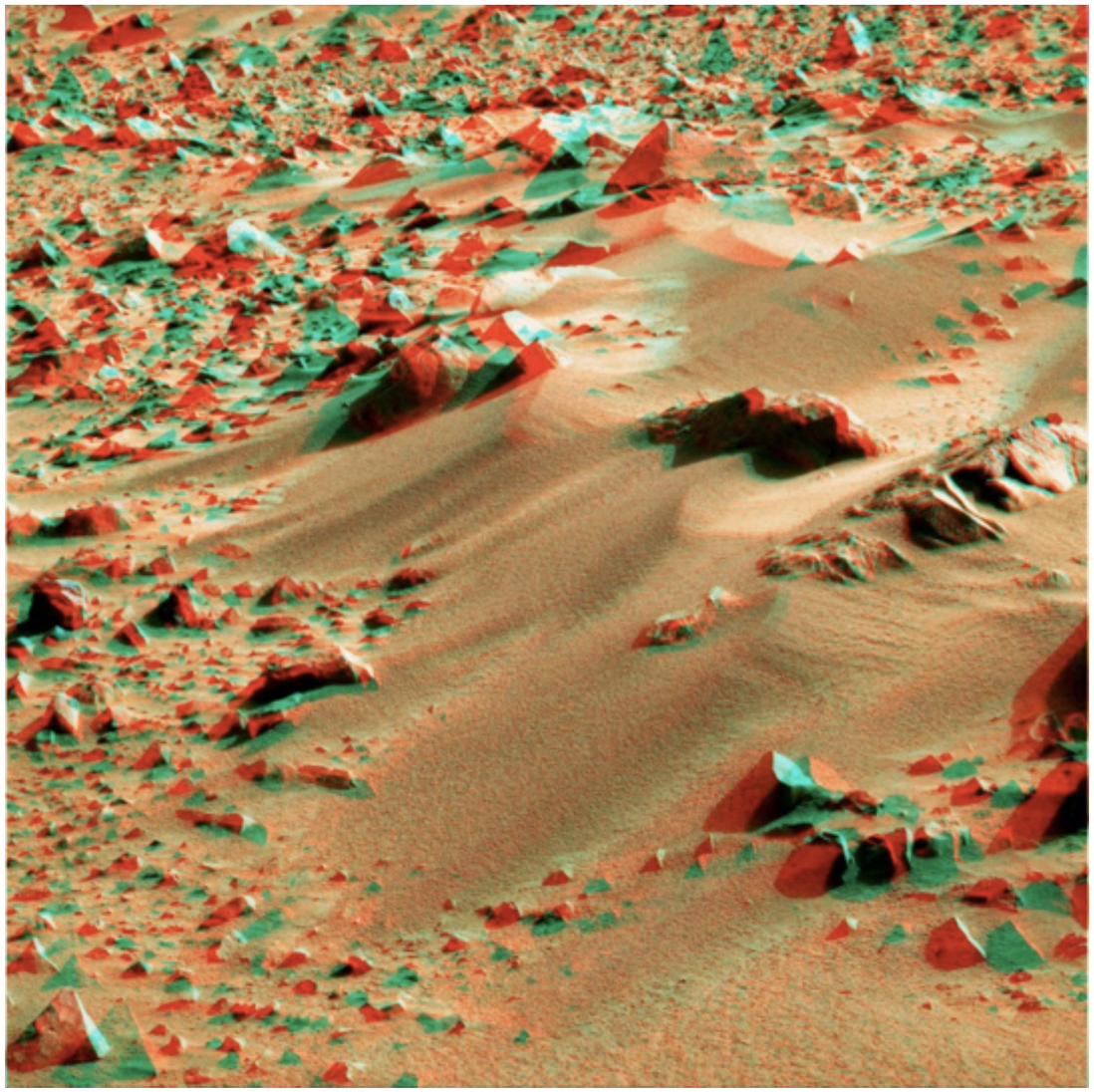
Put red filter over left eye



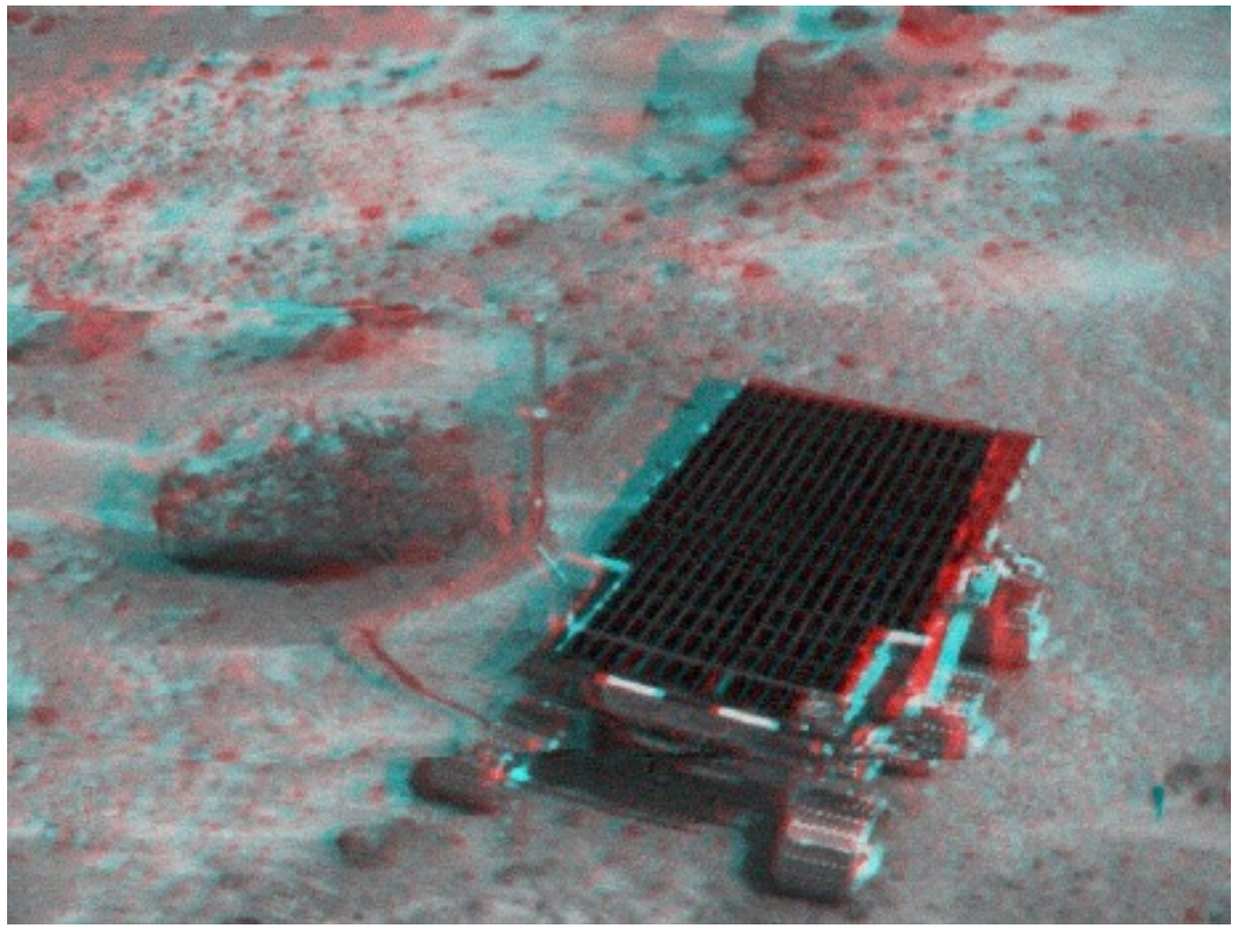
Anaglyphs encode parallax in a single picture.
Two slightly different perspectives of the same subject are superimposed on each other in contrasting colors, producing a three-dimensional effect when viewed through two correspondingly colored filters

http://www.johnsonshawmuseum.org

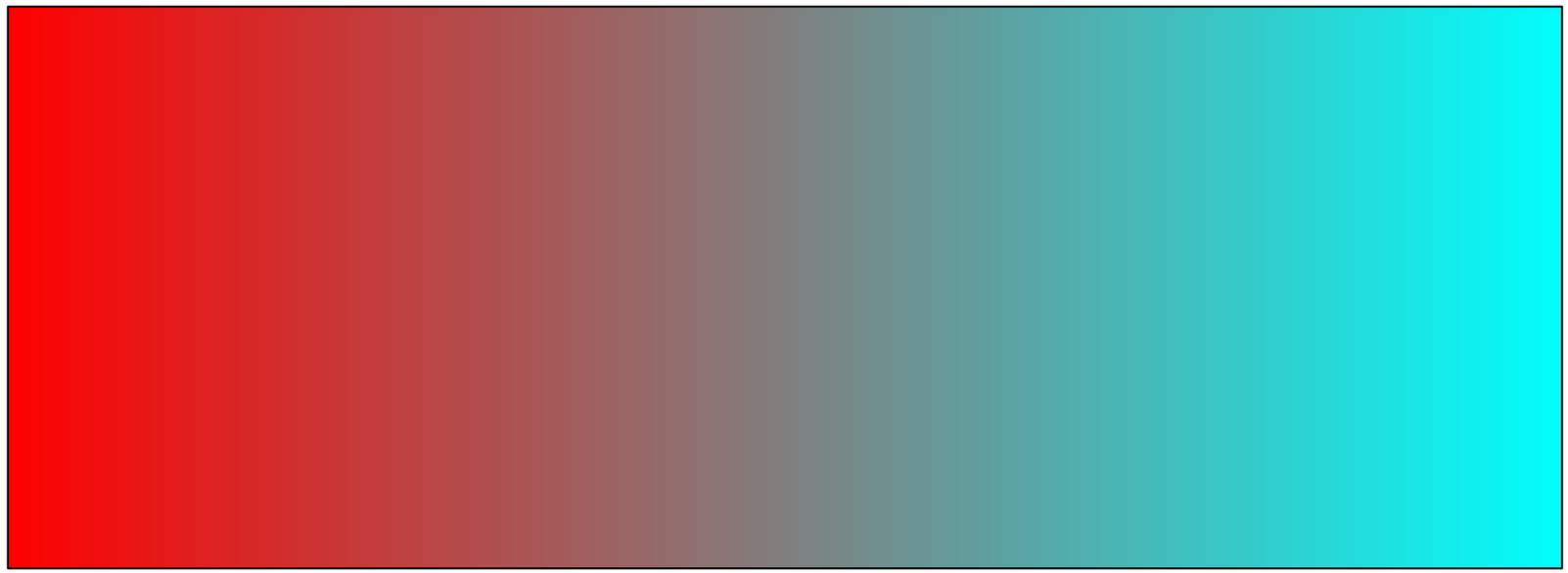
Anaglyph



Anaglyph



How Anaglyphs Work



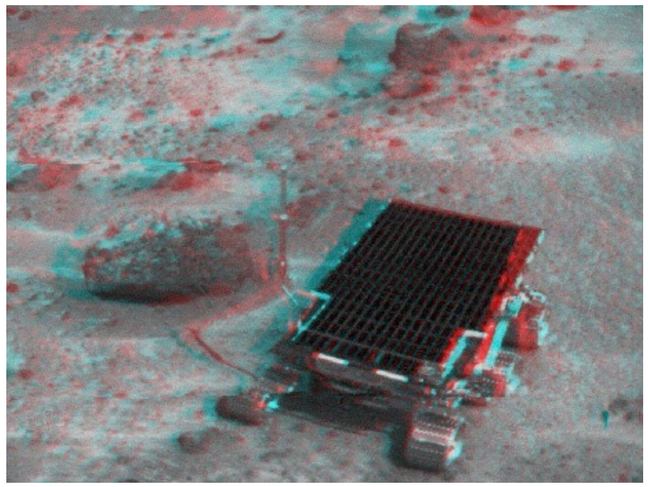
* Close right eye, then close left. What do you observe?
* Red filter selectively passes red color, and similarly for cyan filter and cyan color.

Make an Anaglyph Image

* Given: Grayscale images `img_left`, `img_right`
* Output: Red/cyan anaglyph image

Making an Anaglyph

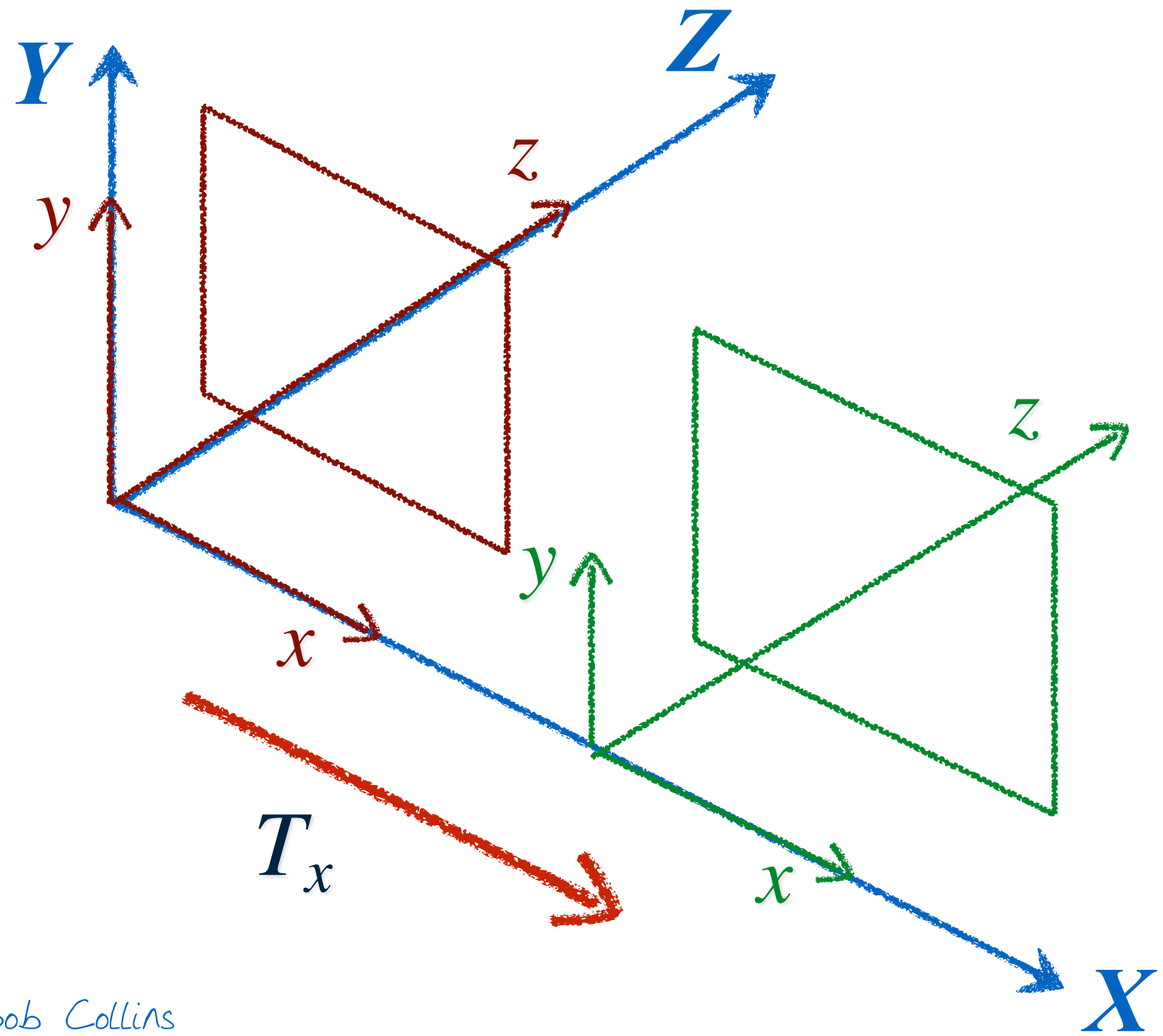
* Take a greyscale stereo pair
* Copy the left image to the red channel of a new image (the anaglyph image)
* Copy the right image to the green and blue channels of the anaglyph image (note: green+blue = cyan)
* View with red-cyan glasses,
  * left eye sees only the left image,
  * and the right eye sees only the right image
  * The brain fuses to form 3D.



A Simple Stereo System

left camera located at $(0,0,0)$

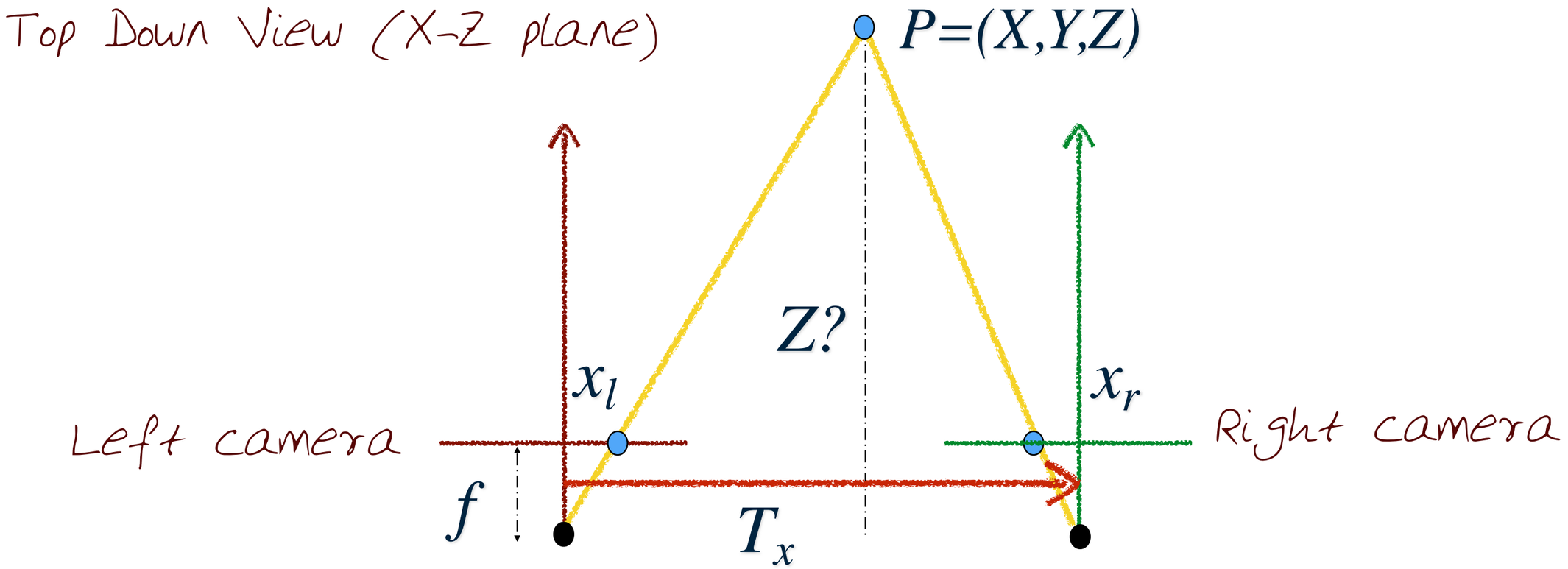
right camera located at $(T_x,0,0)$



Right camera is simply shifted by T_x units along the X axis. Otherwise, the cameras are identical (same orientation / focal lengths)

Slide adapted from Bob Collins

A Simple Stereo System



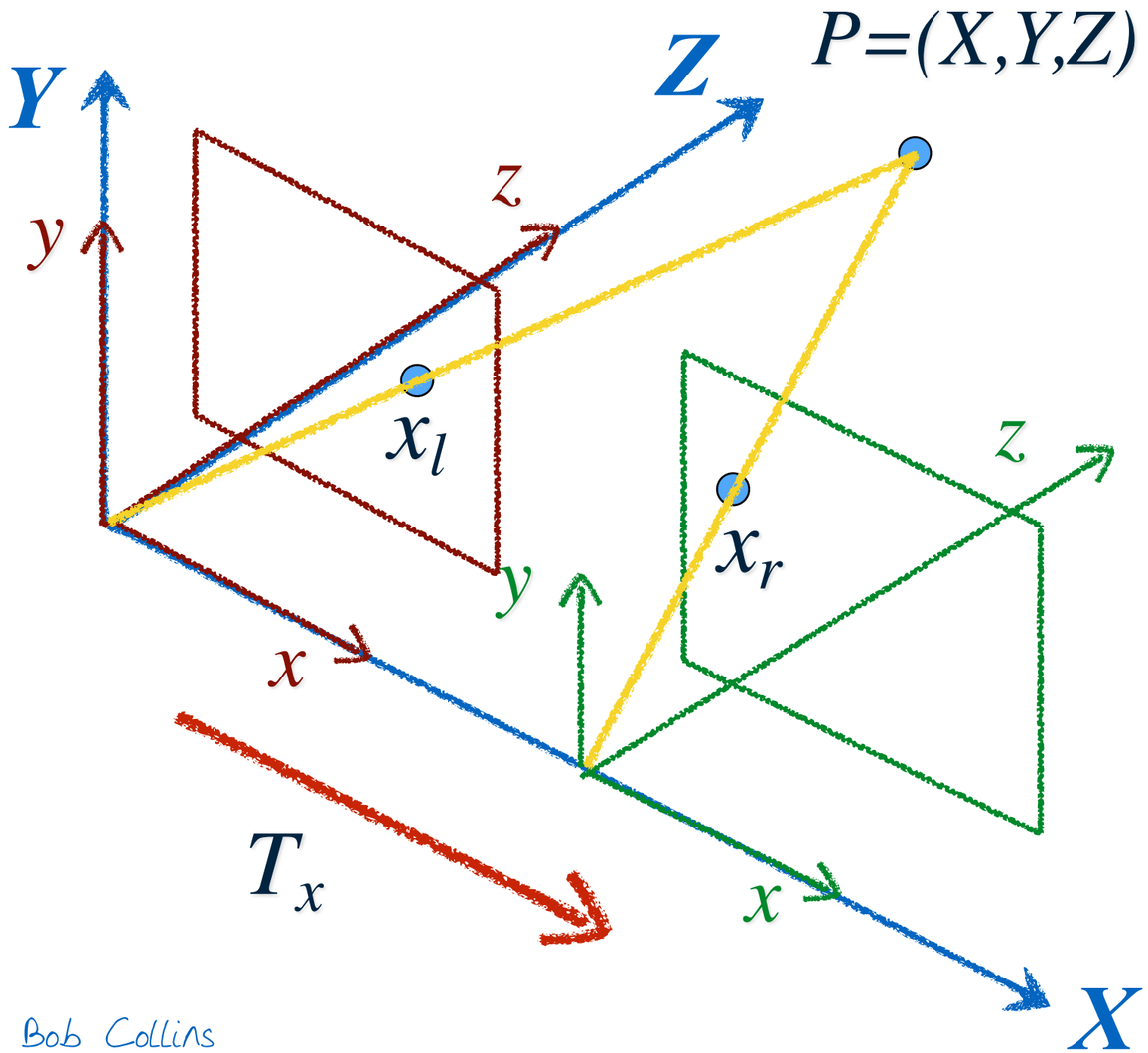
Translated by a distance T_x along X axis

(Tx is also called the stereo "baseline")

A Simple Stereo System

left camera located at $(0,0,0)$

right camera located at $(T_x,0,0)$



Point P in Left Camera:

$$x_l = f\frac{X}{Z} \qquad y_l = f\frac{Y}{Z}$$

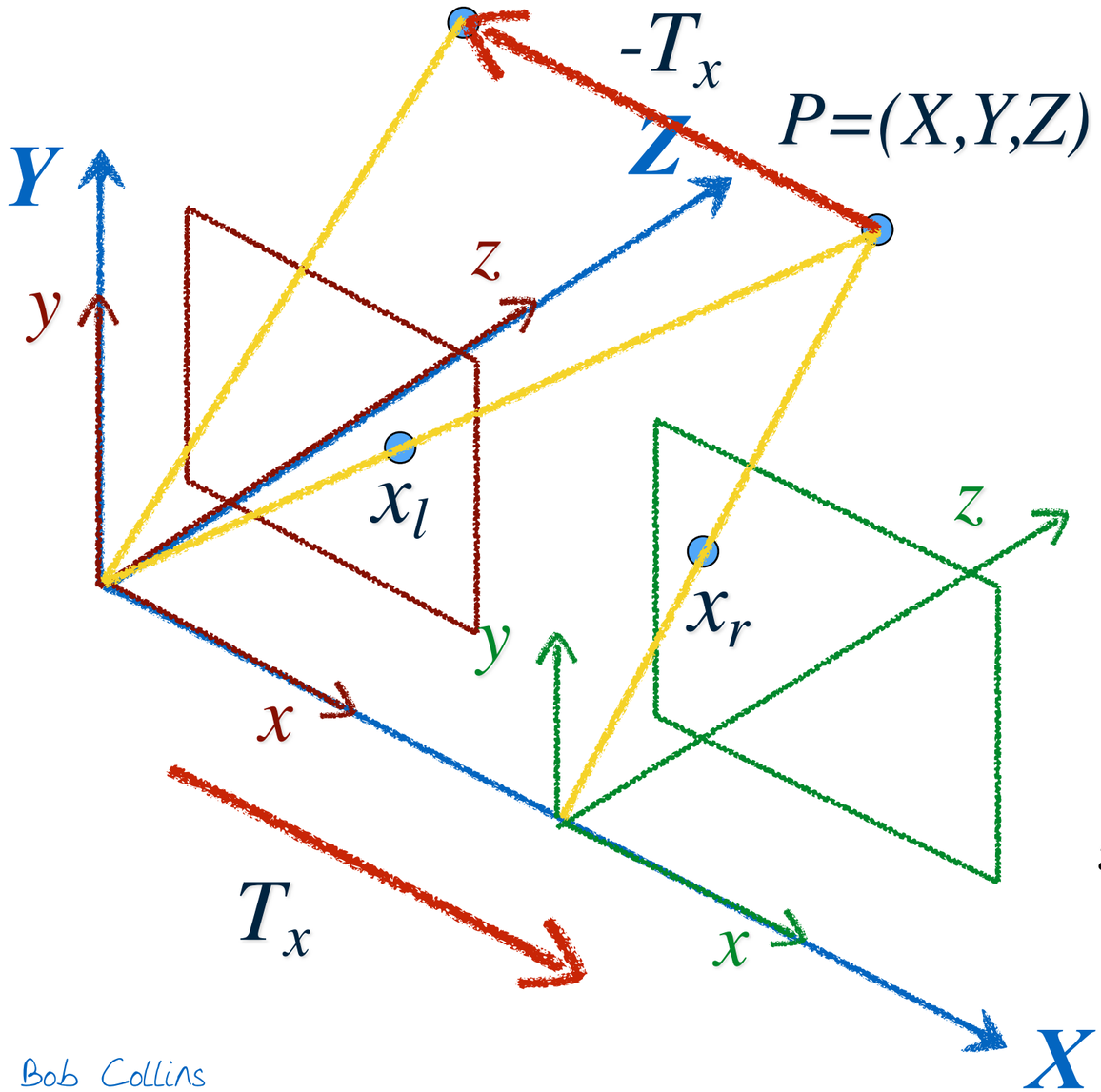
The same point in the Right Camera?

Slide adapted from Bob Collins

A Simple Stereo System

left camera located at $(0,0,0)$

right camera located at $(T_x,0,0)$



Point P in Left Camera:

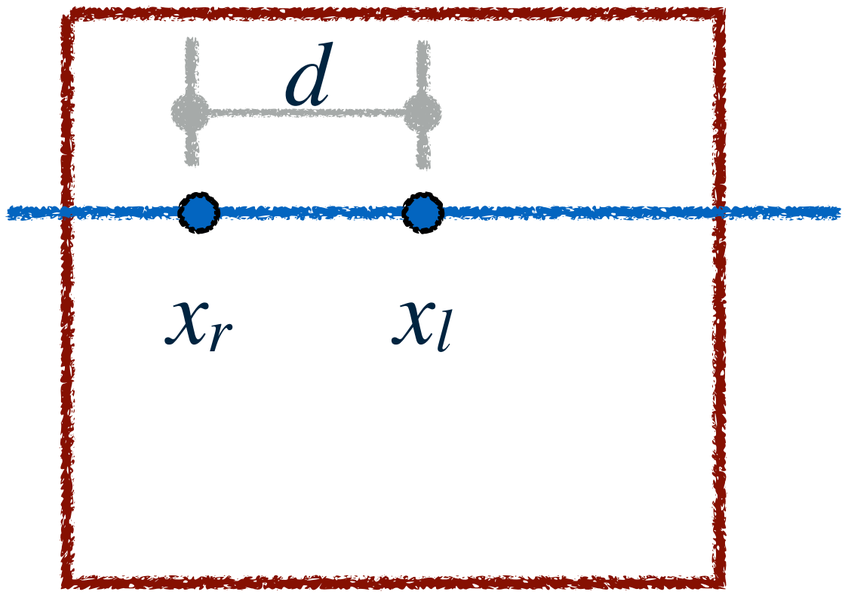
$$x_l = f\frac{X}{Z} \qquad y_l = f\frac{Y}{Z}$$

The same point in the Right Camera?

$$x_r = f\frac{X - T_x}{Z} \qquad y_r = f\frac{Y}{Z}$$

Slide adapted from Bob Collins

Stereo Disparity



$$y_r = y_l$$

left camera

$$x_l = f\frac{X}{Z} \qquad y_l = f\frac{Y}{Z}$$

right camera

$$x_r = f\frac{X - T_x}{Z} \qquad y_r = f\frac{Y}{Z}$$

$$d = x_l - x_r$$

$$d = f\frac{X}{Z} - f\frac{X - T_x}{Z}$$

depth

baseline

$$d = f\frac{T_x}{Z}$$

$$Z = f\frac{T_x}{d}$$

disparity

Stereo Example

Left Image

Right Image

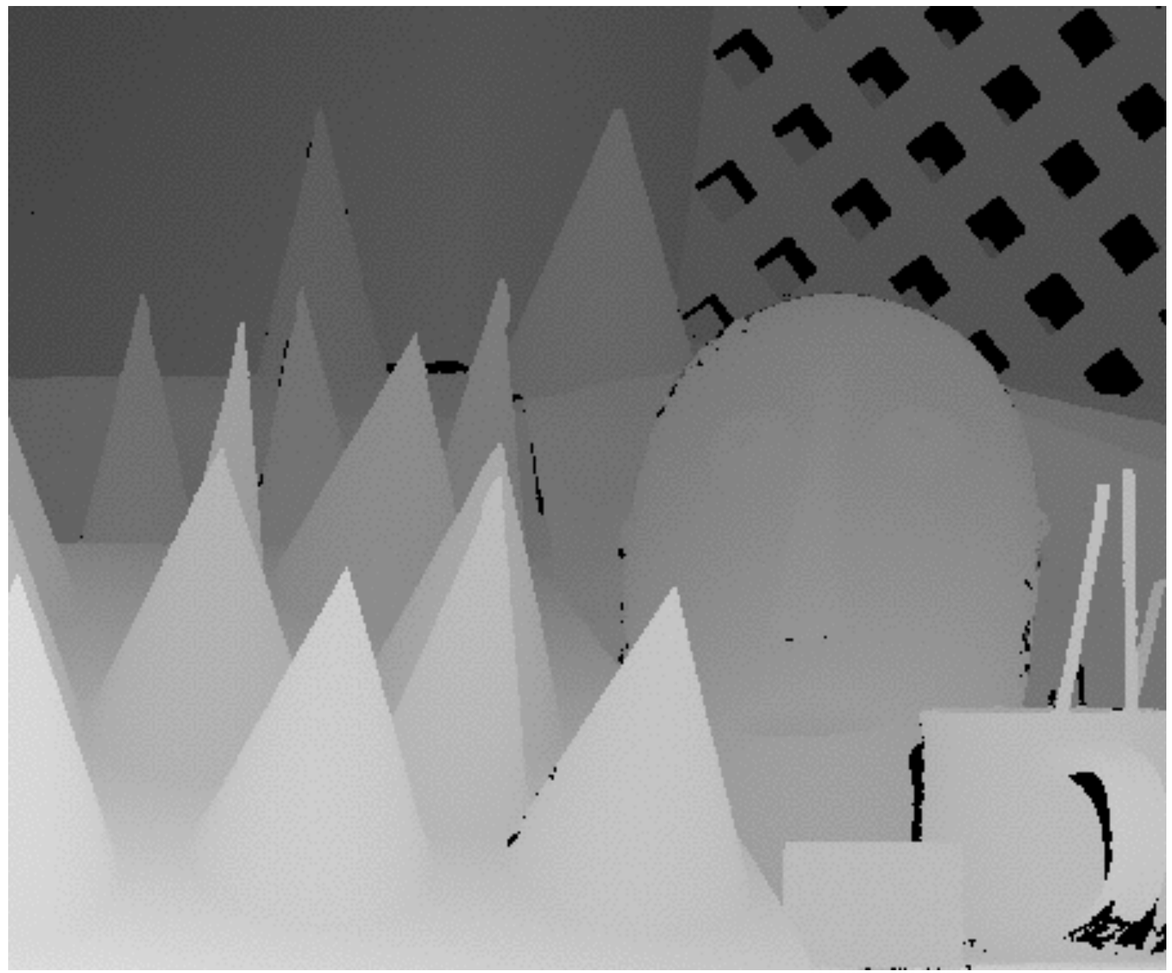
From middlebury stereo evaluation page http://www.middlebury.edu/stereo/

Stereo Example

Left image

Right image

Disparity values (0-64)

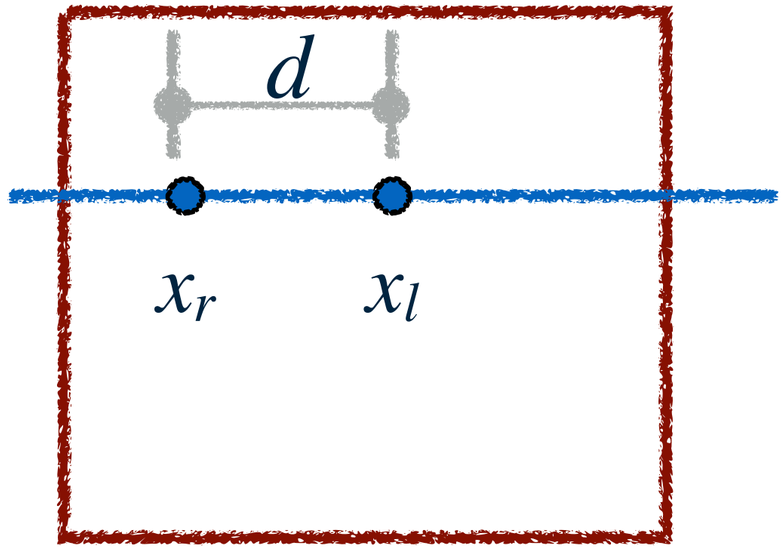


Disparity is larger for closer surfaces

Slide adapted from Bob Collins

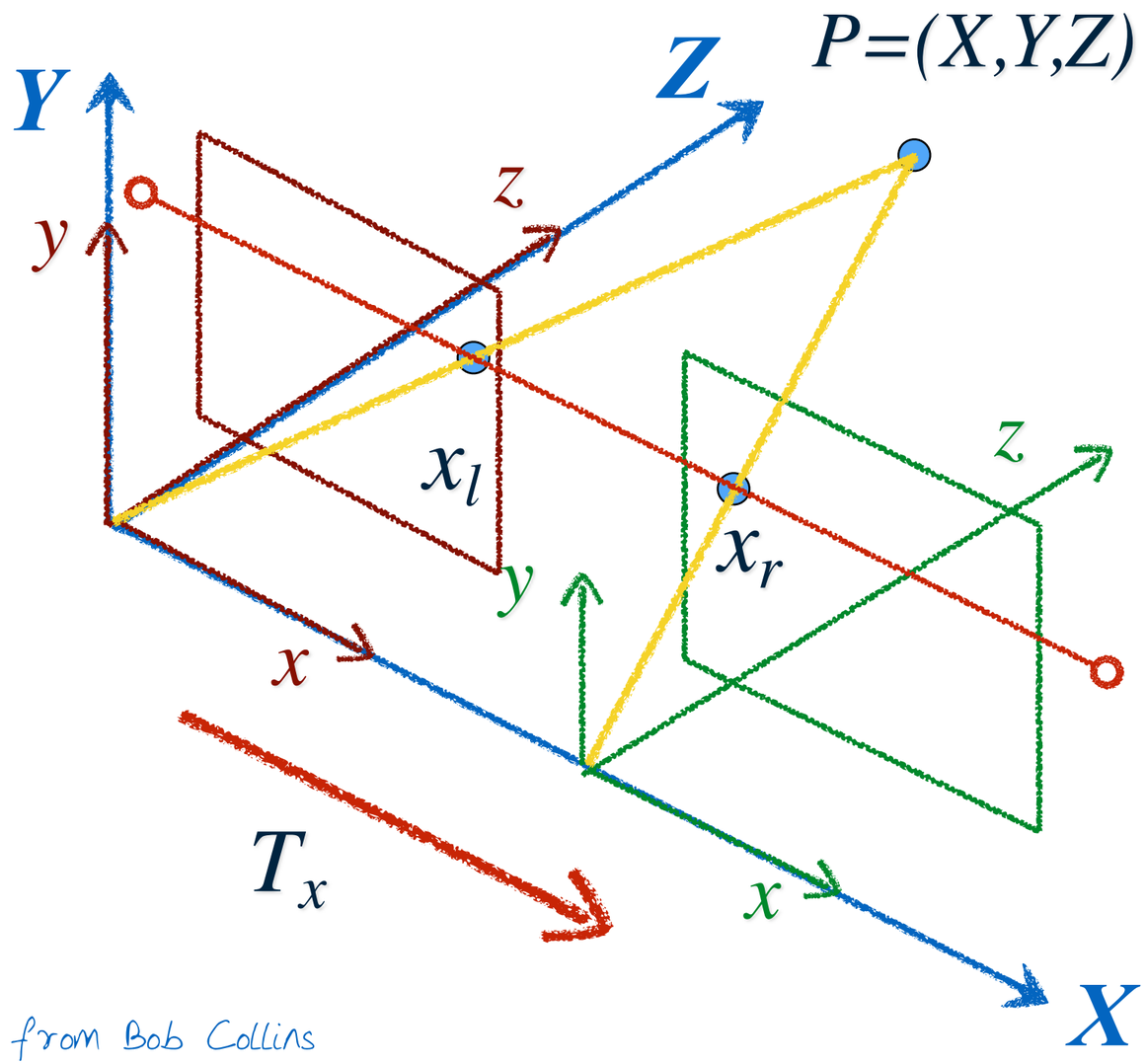
Computing Disparity

$$d = x_l - x_r$$



* Correspondence
  * Which pixel in the right image corresponds to each pixel in the left
* Epipolar Constraint
  * No need to search the whole 2D right image
  * The epipolar constraint reduces the search space to a one-dimensional line

Recall: A Simple Stereo System



left camera

$$x_l = f\frac{X}{Z} \qquad y_l = f\frac{Y}{Z}$$

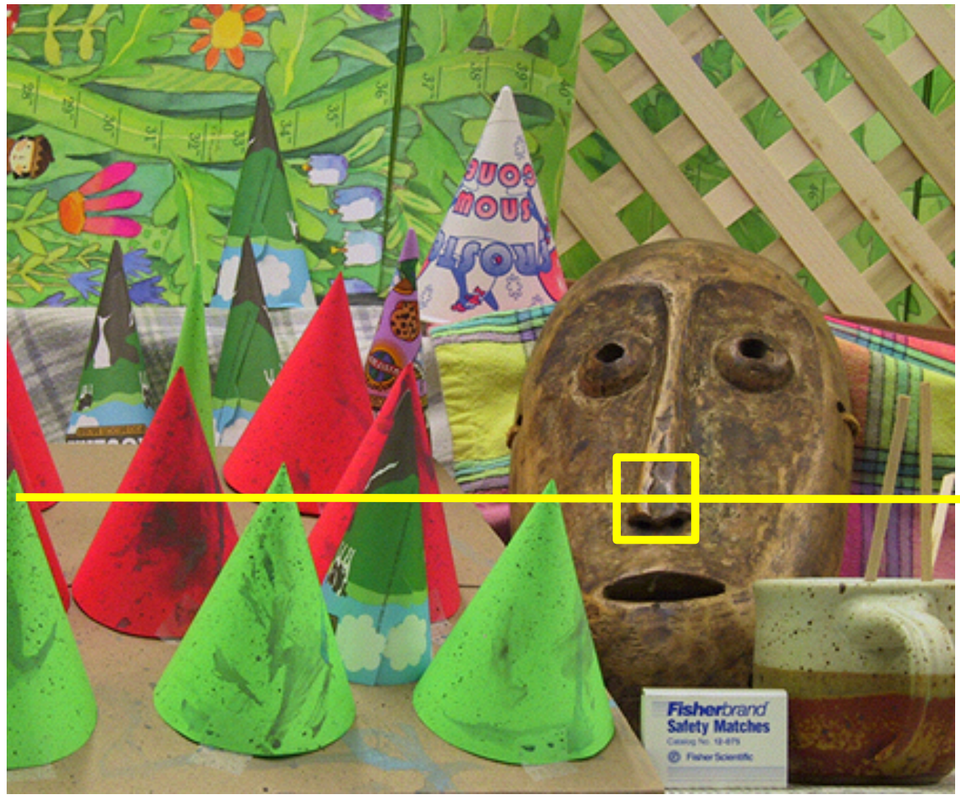
right camera

$$x_r = f\frac{X - T_x}{Z} \qquad y_r = f\frac{Y}{Z}$$

Slide adapted from Bob Collins

Matching Using Epipolar Lines

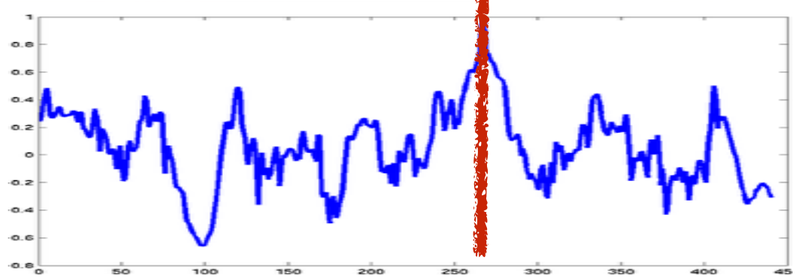
Left Image



Right Image

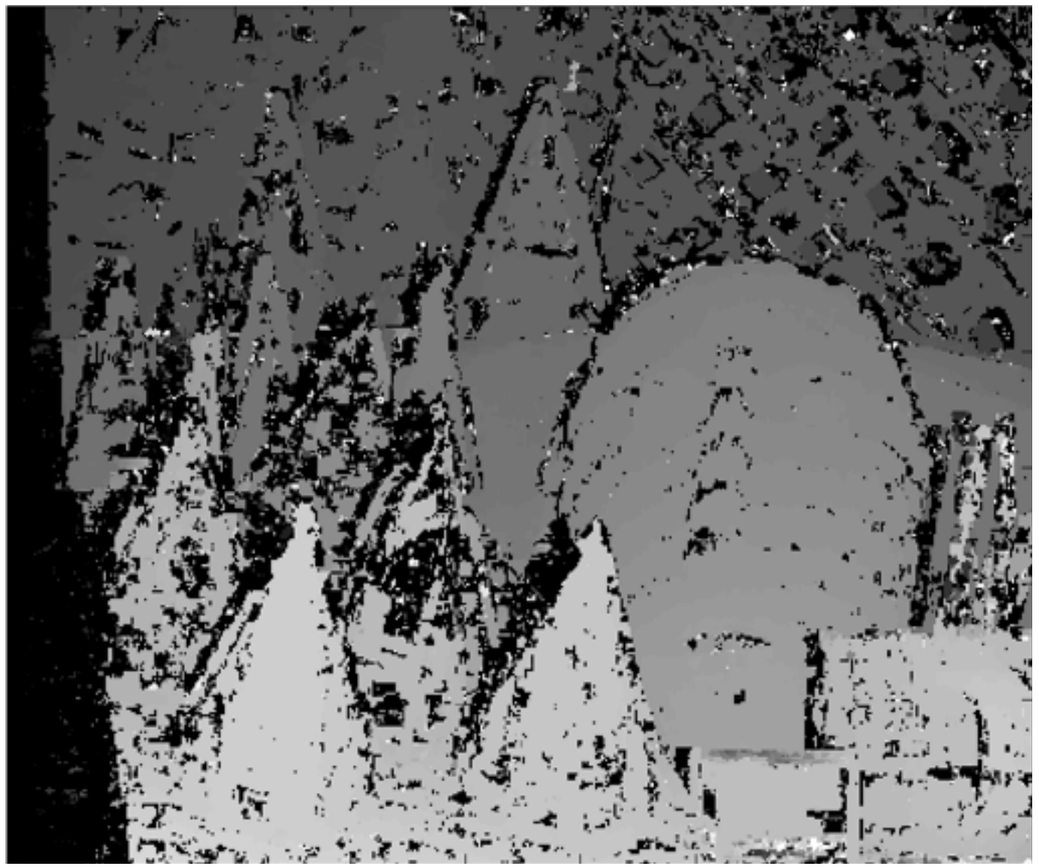
For a patch in left image

Compare with patches along same row in right image

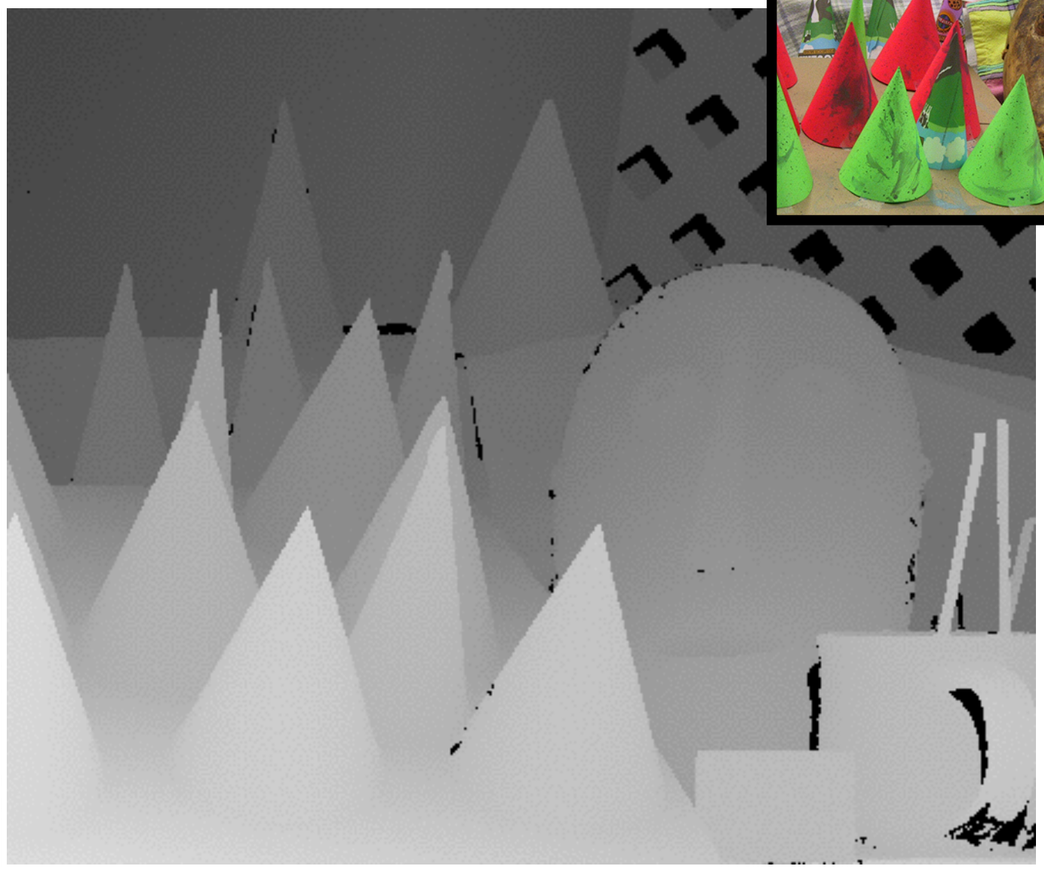


Match Score Values

No Matches



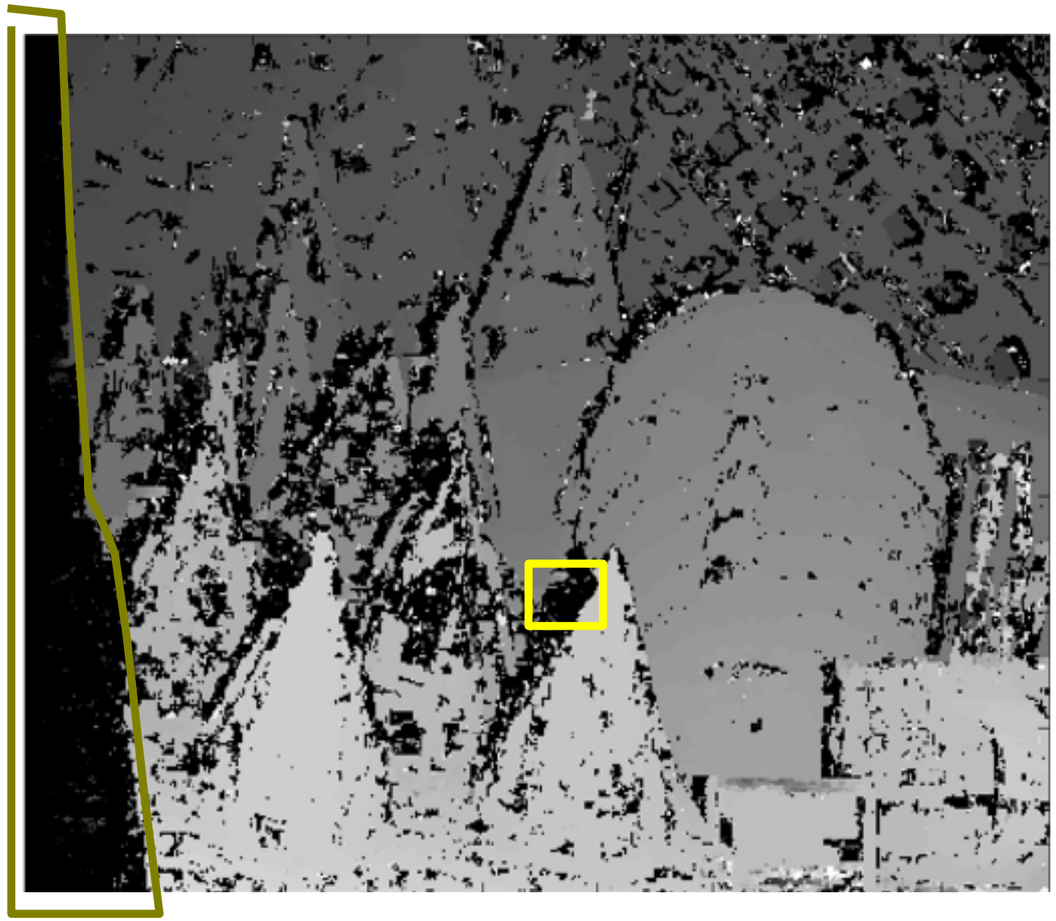
Computed disparities



Ground truth

Black pixels: bad disparity values, or no matching patch in right image

Occlusions: No Matches



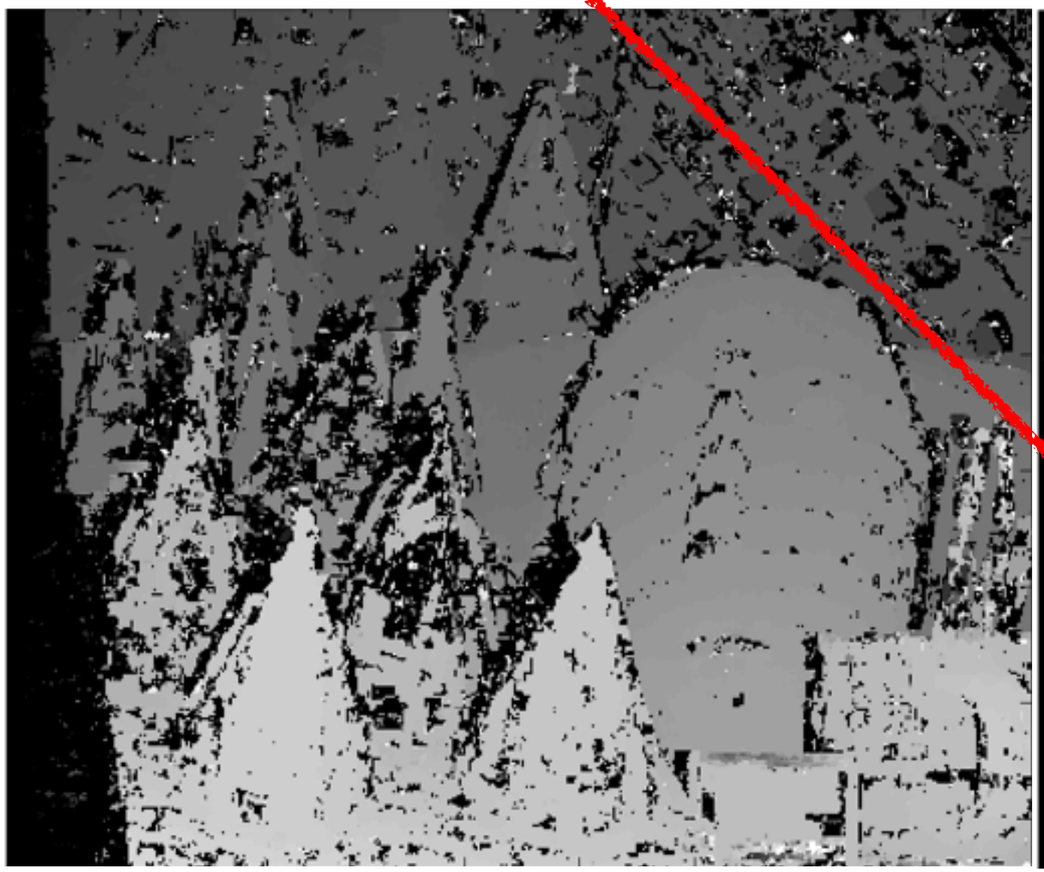
Left Image

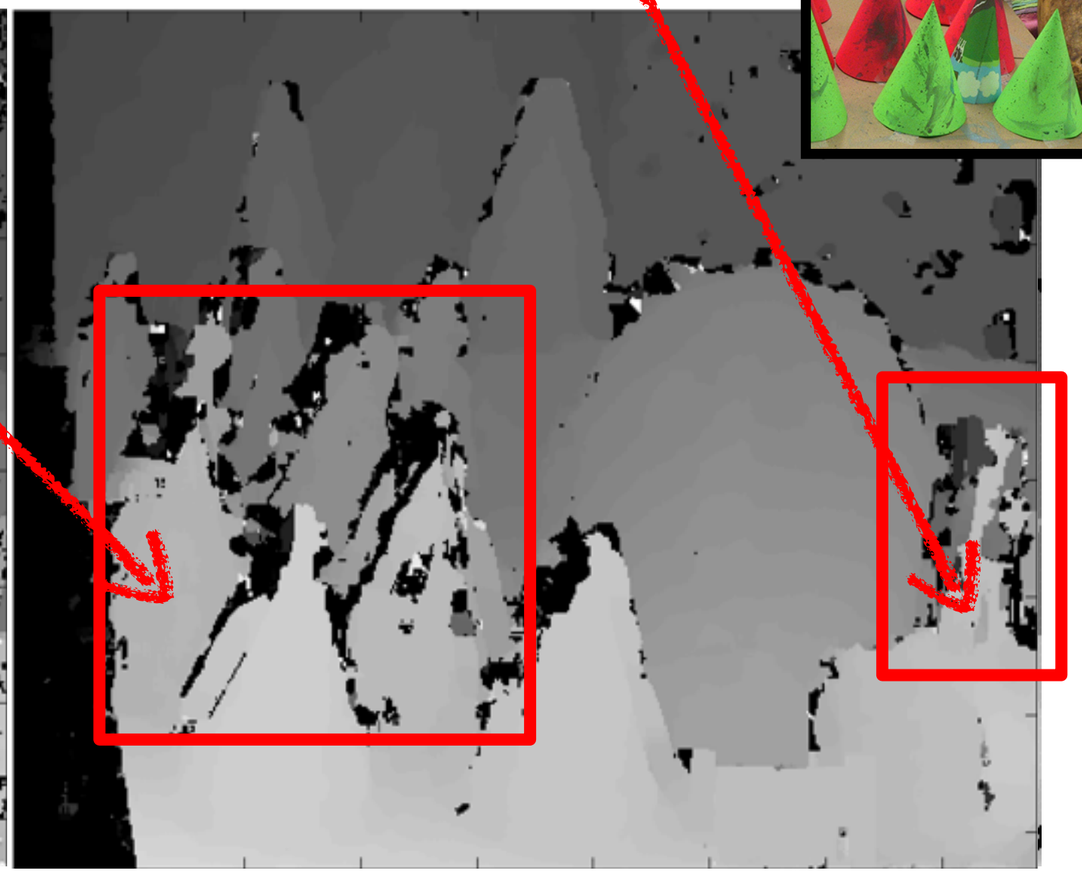
Right Image

Effects of Patch Size

Smoother in some areas

Loss of finer details

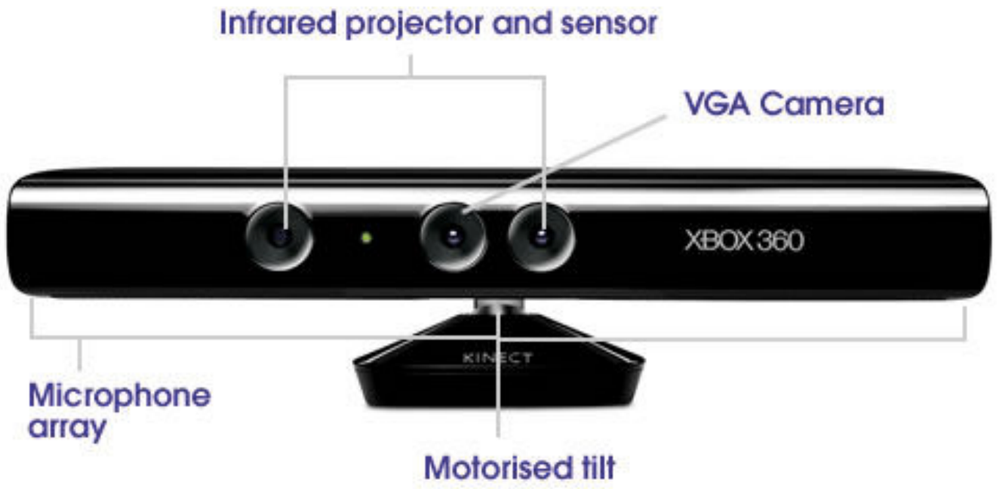


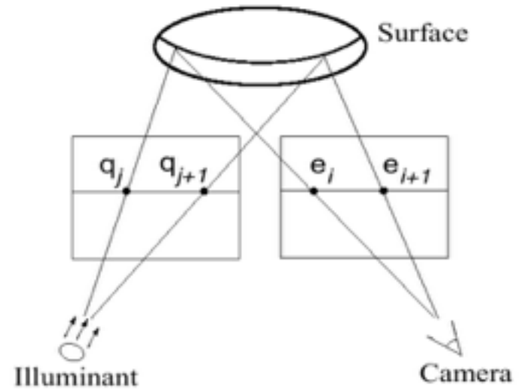


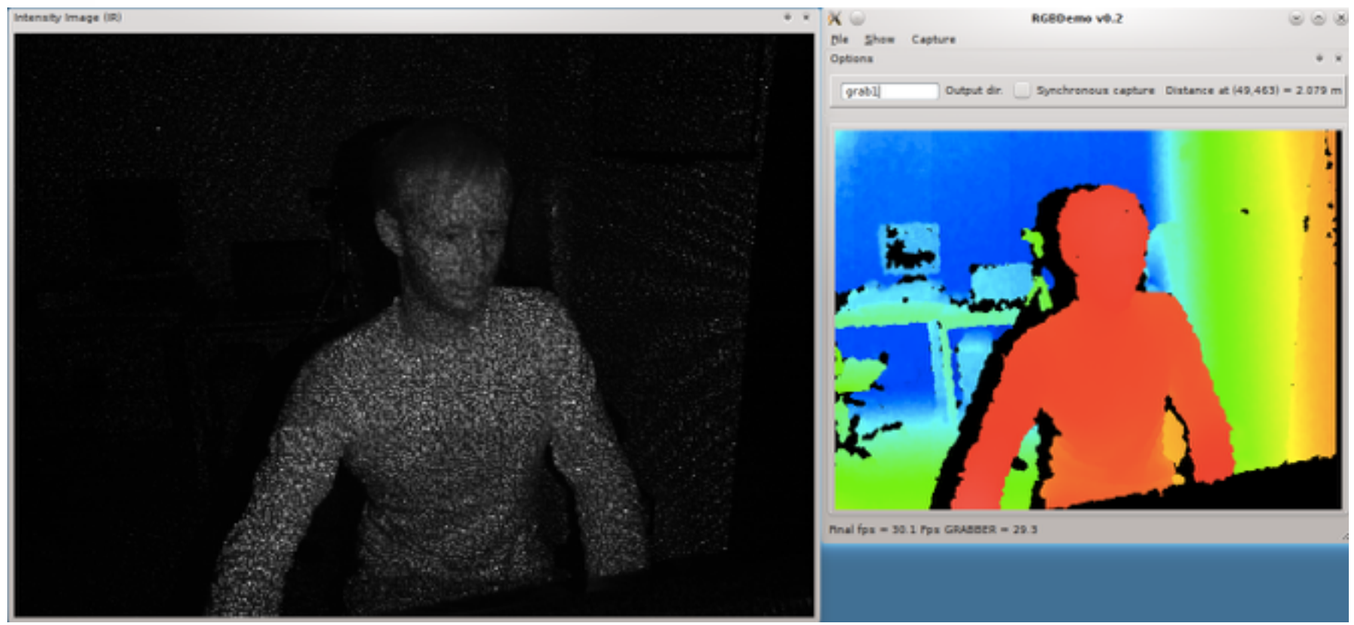
5 X 5 patches

11 X 11 patches

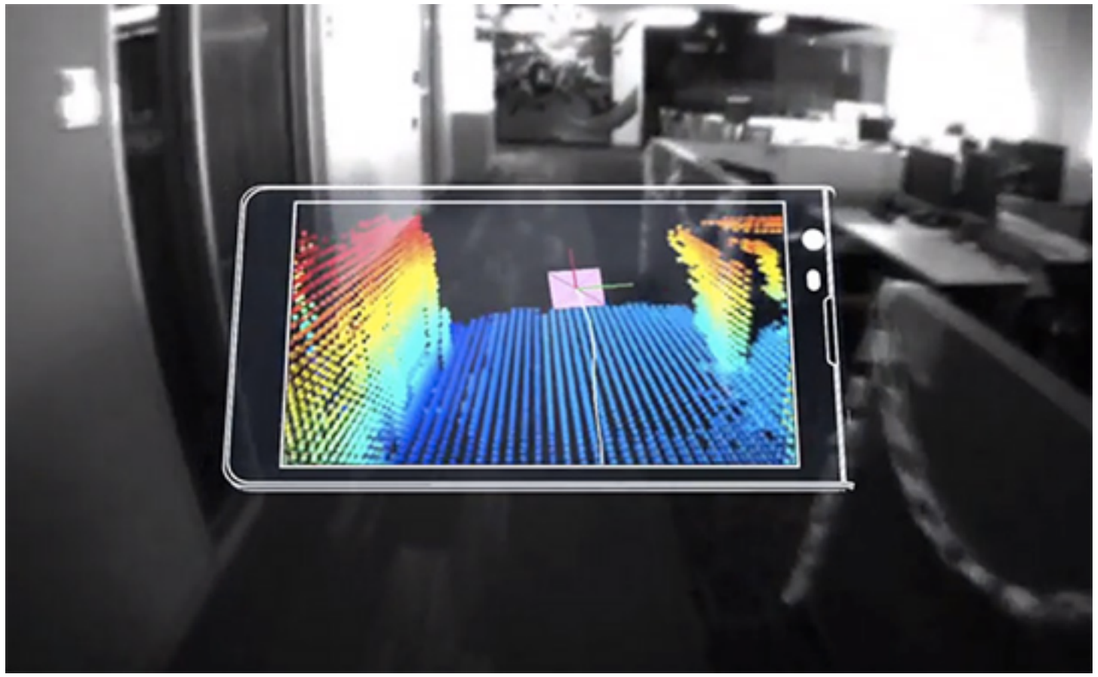
Some well known RGBD Cameras







http://nicolas.burrus.name/



Google Project Tango

Amazon Fire

5 Cameras

Kinect For Windows 1

Processed Image From Kinect

Kinect For Windows 2

Image via http://blogs.msdn.com

zugara

Kinect 1 to Kinect 2: Time of Flight Camera

Summary

* Shape/Structure/Geometry from X methods
* Stereo Imaging
* Compute depth from a Stereo image pair

Further Reading

* McMillan & Gortler (1999) "Image-Based Rendering: A New Interface Between Computer Vision and Computer Graphics" Applications of Computer Vision to Computer Graphics Vol 33, No 2 1999 [Link]
* Szeliski (2010)
* Hartley & Zisserman (2004)

Credits

- For more information, see:
  - Richard Szeliski (2010) Computer Vision: Algorithms and Applications, Springer (Chapter 3)
- Some concepts in slides motivated by similar slides by James Hays Alyosha Efros, Svetlana Lazebnik, Bob Collins, and Greg Turk
- Grant Schindler,
  - http://www.trimensional.com/
- Additional list will be available on website

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

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

Photo-synth and other Panorama Systems

* Exploring space with Photographs
* Photosynth, Photomaps, etc.

Georgia Tech College of Computing

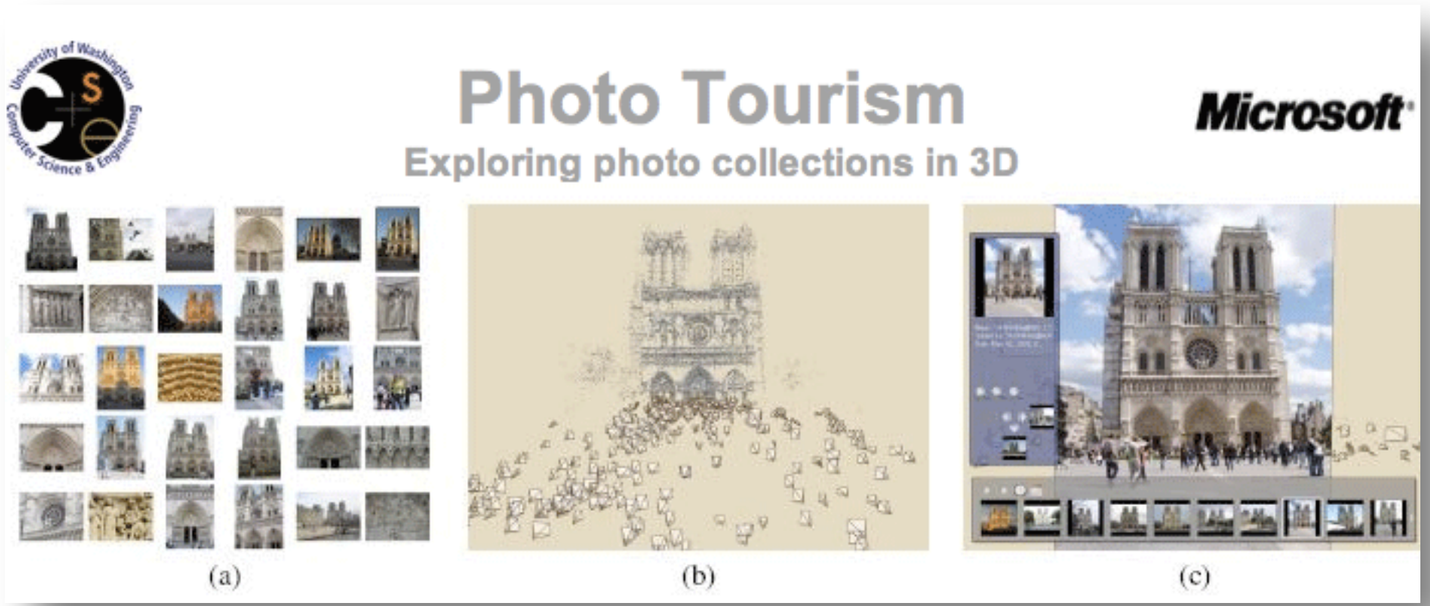
Lesson Objectives

1. Going beyond Panoramas
2. Photo Tourism
3. Photo maps, Street Views, etc.

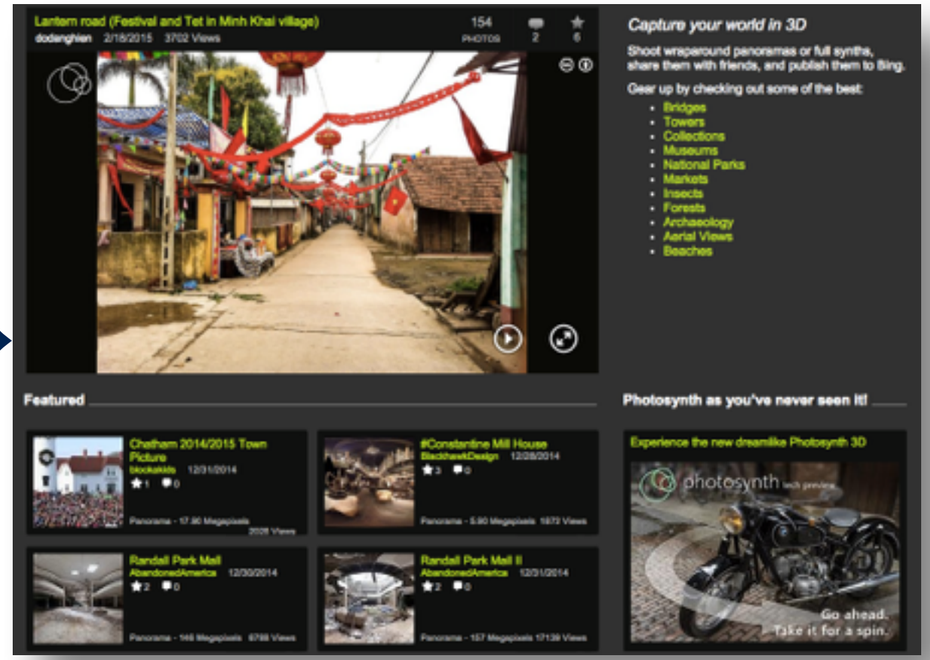
Recall: Panoramas

Recall:
Planar,
Spherical,
Cylindrical
Panoramas

Photo Tourism ⇒ Photo Synth



⇒



* Snavely, Seitz, Szeliski, "Photo tourism: Exploring photo collections in 3D," ACM SIGGRAPH, 2006

* photosynth.net Technology Preview (2008 - 2013)

http://phototour.cs.washington.edu/

Photo Tourism

Exploring photo collections in 3D

Noah Snavely Steven M. Seitz Richard Szeliski

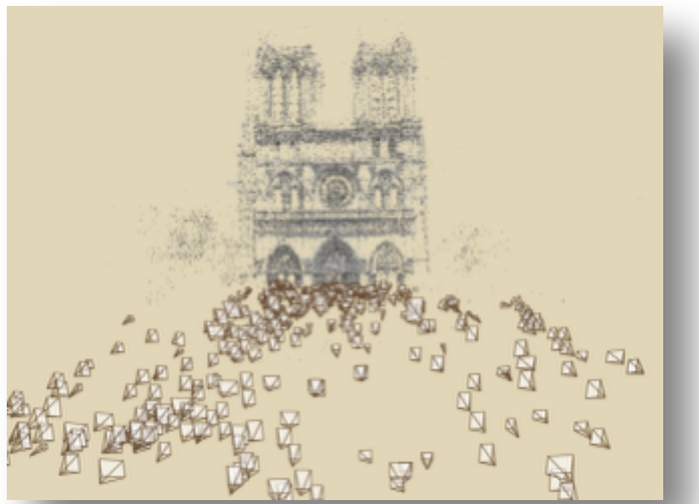
University of Washington *Microsoft Research*

SIGGRAPH 2006

Photo Tourism overview

Input photographs

Scene reconstruction



Relative camera positions and orientations

Point cloud

Sparse correspondence

Photo Explorer

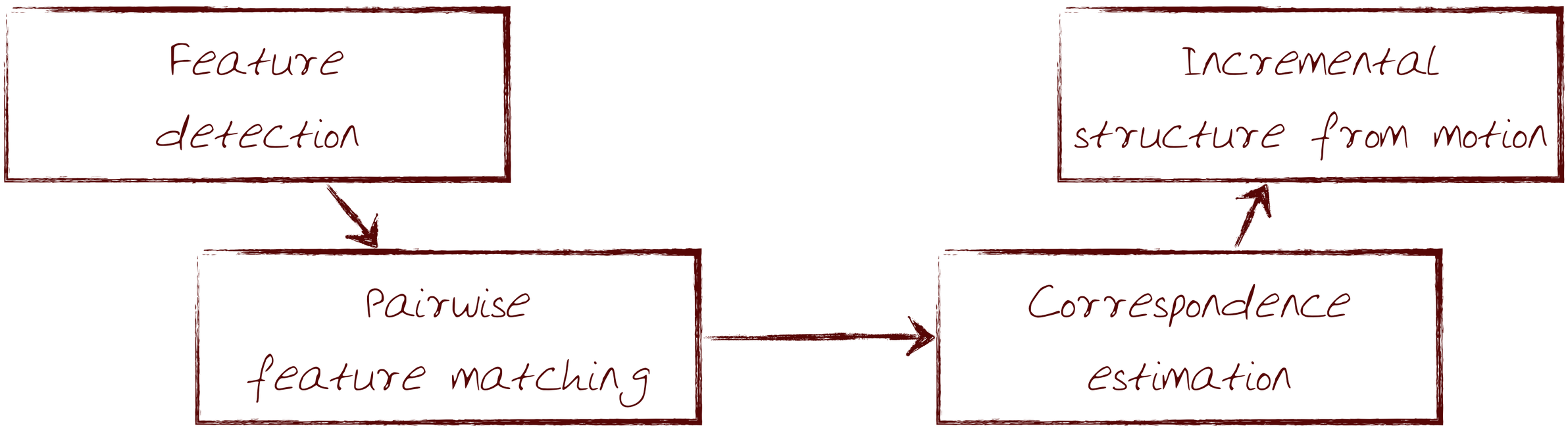
Adapted from Noah Snavely

http://phototour.cs.washington.edu/

Scene reconstruction

Automatically estimate

- position, orientation, and focal length of cameras
- 3D positions of feature points



Adapted from Noah Snavely

Feature Detection

Feature detection

↓

Pairwise feature matching

↓

Correspondence estimation

↓

Incremental structure from motion

Detect features using SIFT [Lowe, 2004]

Adapted from Noah Snavely

Feature Detection

Feature detection

Pairwise feature matching

Correspondence estimation

Incremental structure from motion

Detect features using SIFT [Lowe, 2004]

Adapted from Noah Snavely

Feature detection

Pairwise feature matching

Correspondence estimation

Incremental structure from motion

Feature Detection

Detect features using SIFT [Lowe, 2004]

Adapted from Noah Snavely

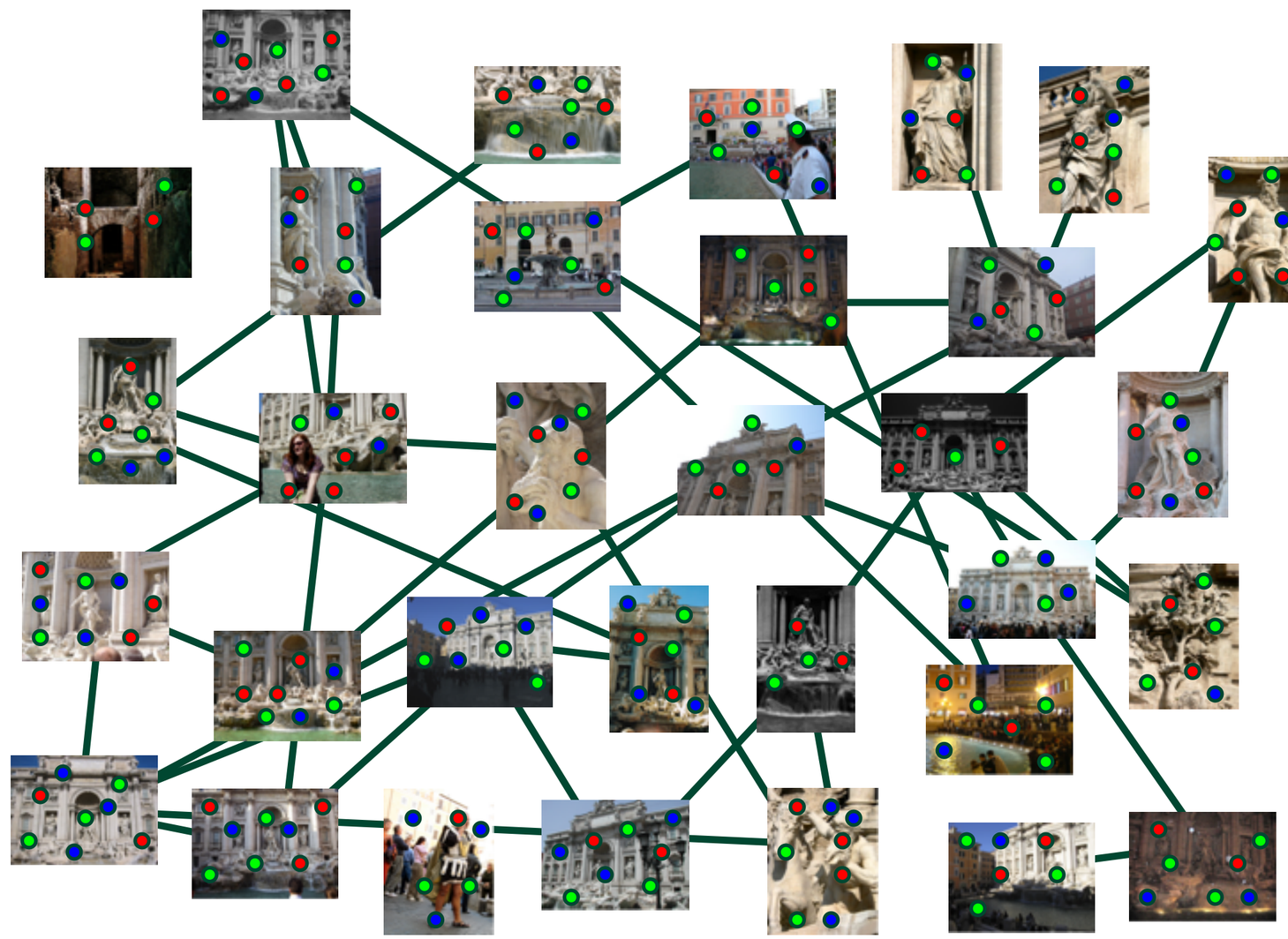
Feature detection

Pairwise feature matching

Correspondence estimation

Incremental structure from motion

Pairwise feature matching



Match features between each pair of images

Adapted from Noah Snavely

Feature detection

↓

Pairwise feature matching

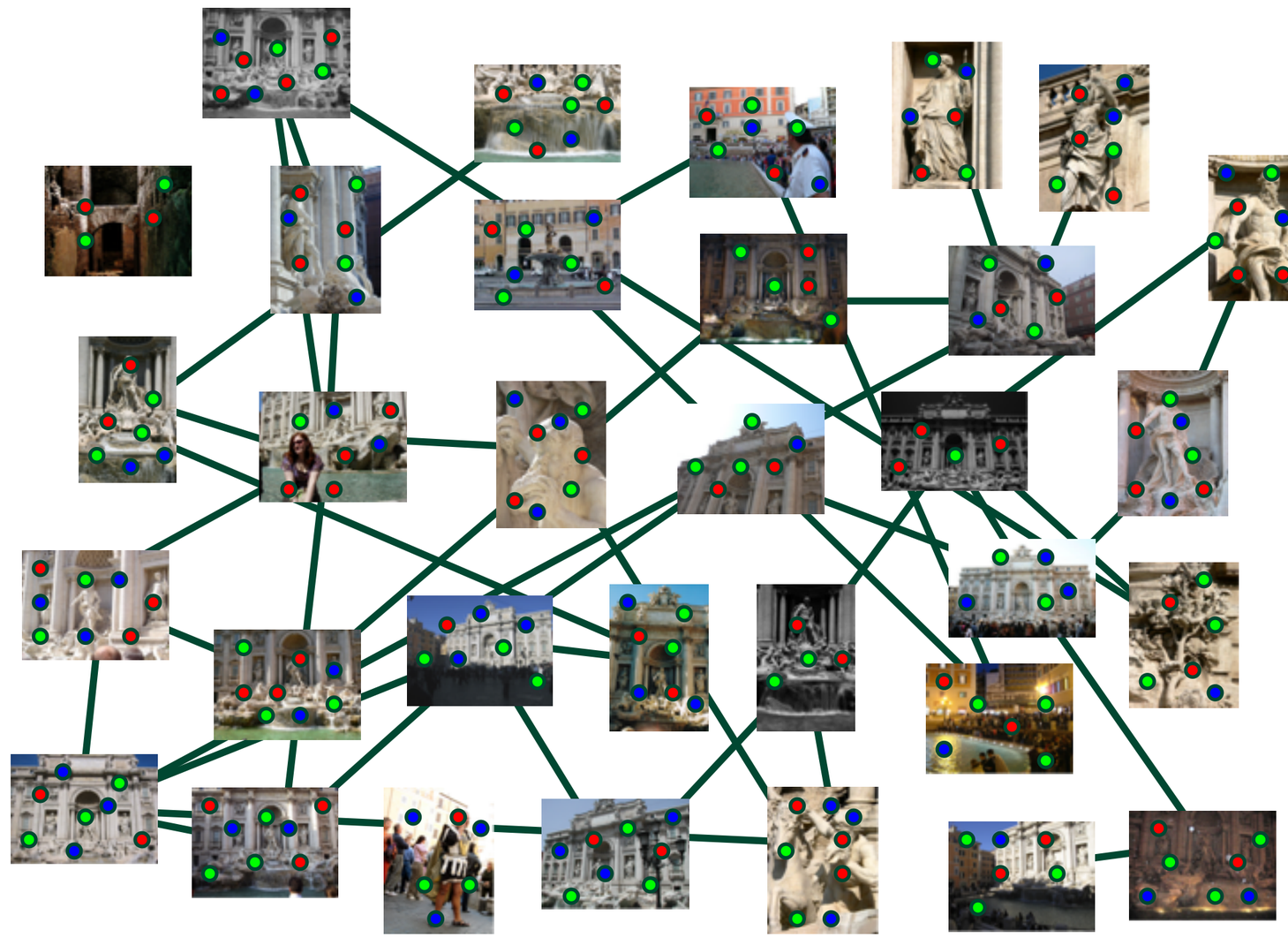
↓

Correspondence estimation

↓

Incremental structure from motion

Pairwise feature matching



Refine matching using RANSAC between pairs

Adapted from Noah Snavely

Feature detection

↓

Pairwise feature matching

↓

Correspondence estimation

↓

Incremental structure from motion

Correspondence Estimation

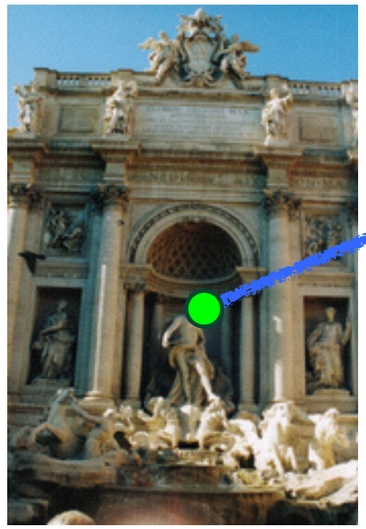


Image 1

Image 2

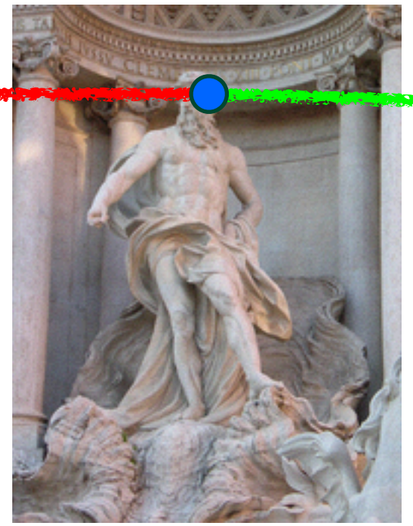


Image 3

Image 4

Link up pairwise matches to form connected components of matches across several images

Adapted from Noah Snavely

Structure from Motion

Feature detection

↓

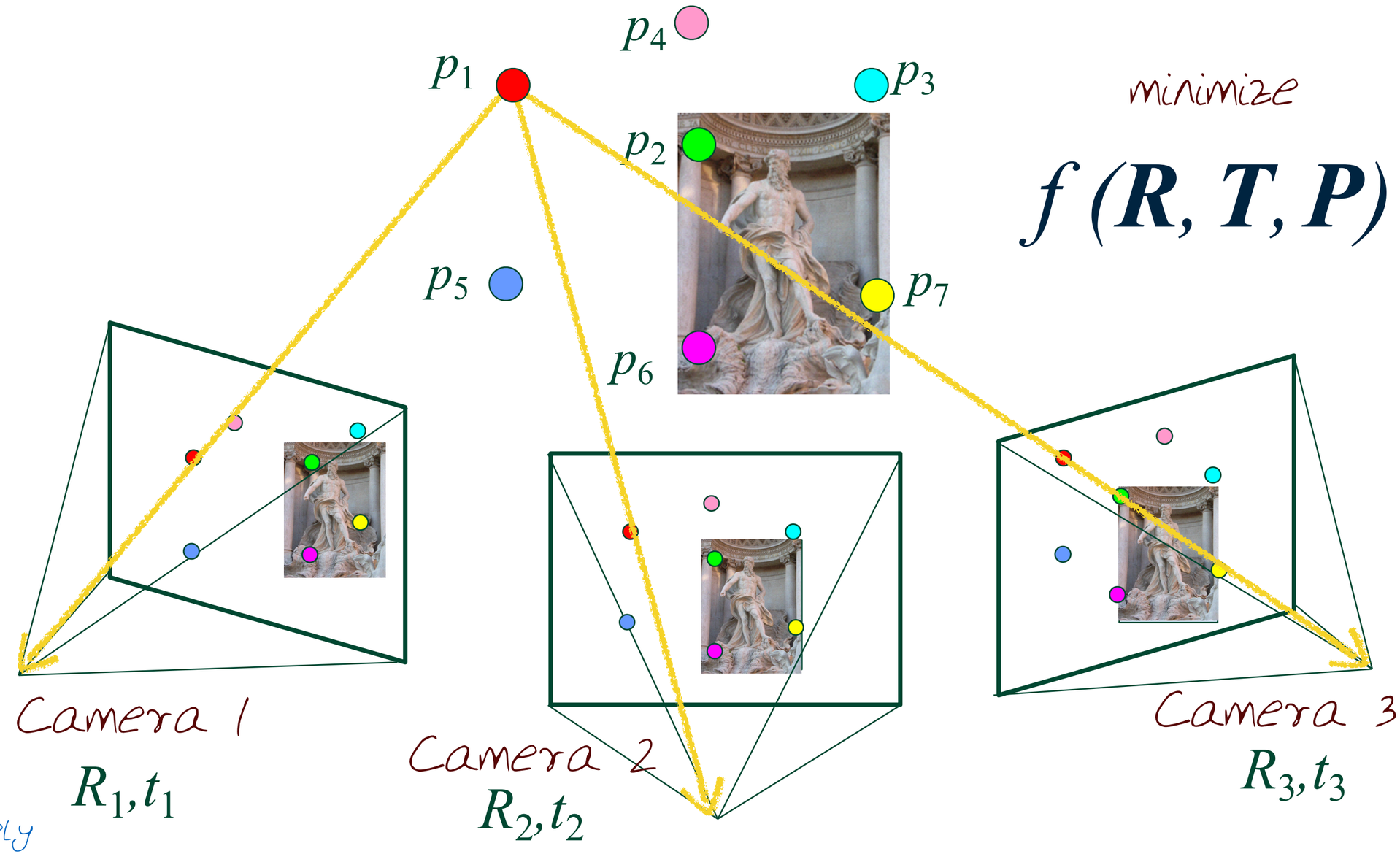
Pairwise feature matching

↓

Correspondence estimation

↓

Incremental structure from motion



minimize

$$f(\boldsymbol{R}, \boldsymbol{T}, \boldsymbol{P})$$

Adapted from Noah Snavely

Incremental Structure from Motion

Feature detection

↓

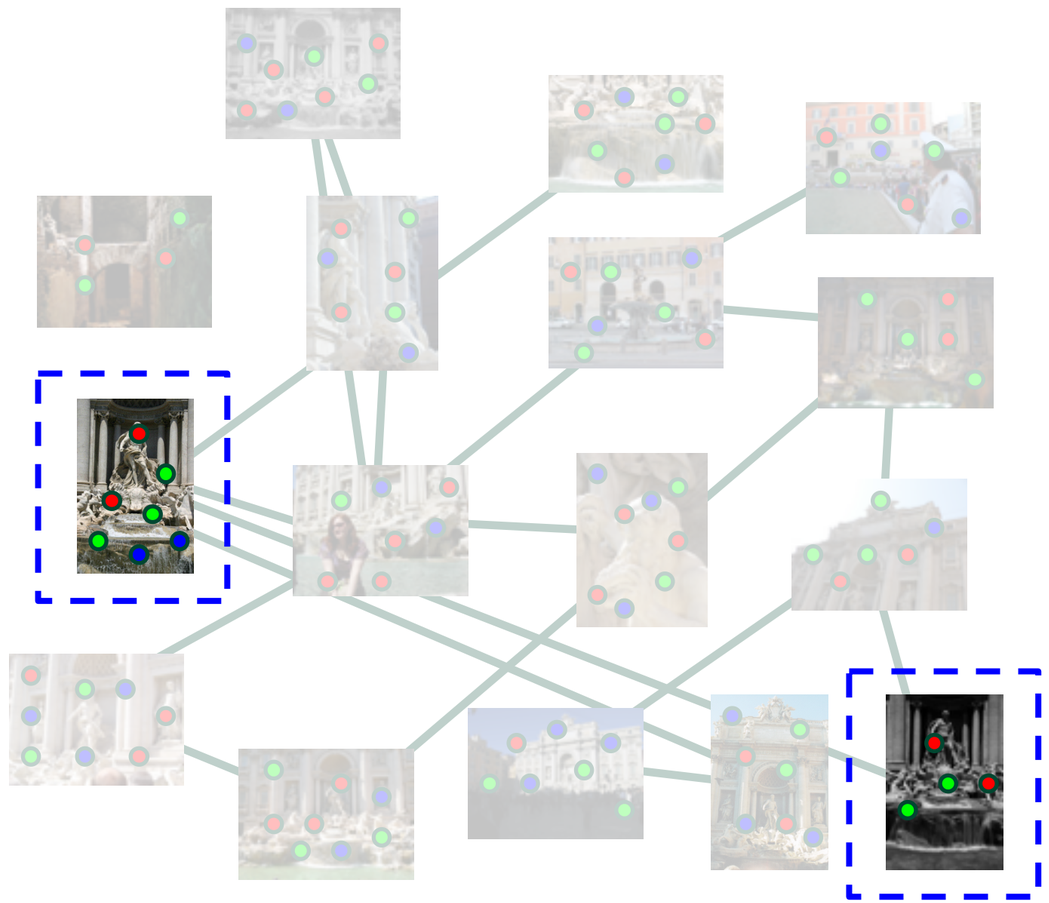
Pairwise feature matching

↓

Correspondence estimation

↓

Incremental structure from motion



Adapted from Noah Snavely

Incremental Structure from Motion

Feature detection

↓

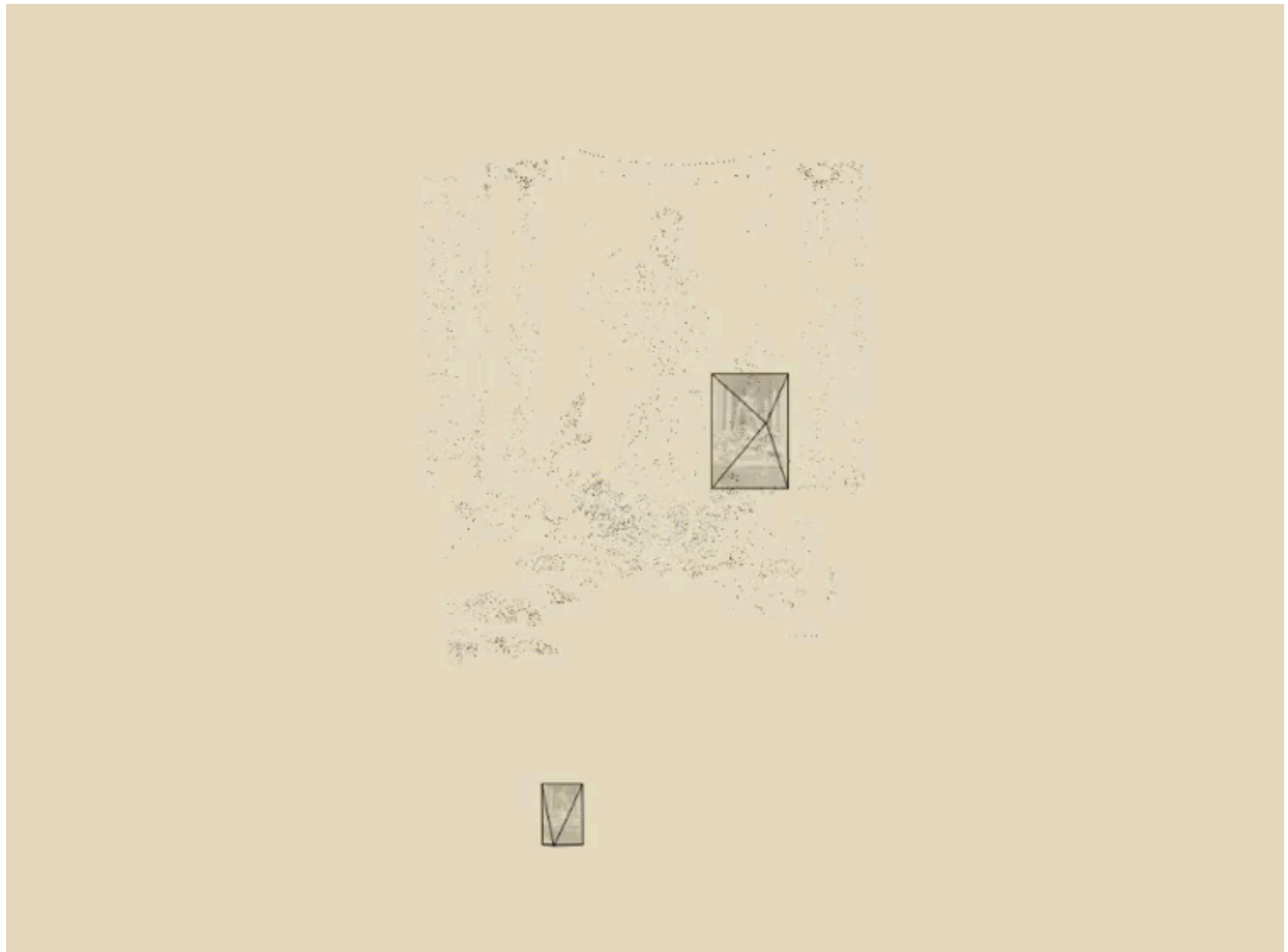
Pairwise feature matching

↓

Correspondence estimation

↓

Incremental structure from motion



Incremental Structure from Motion

Feature detection

↓

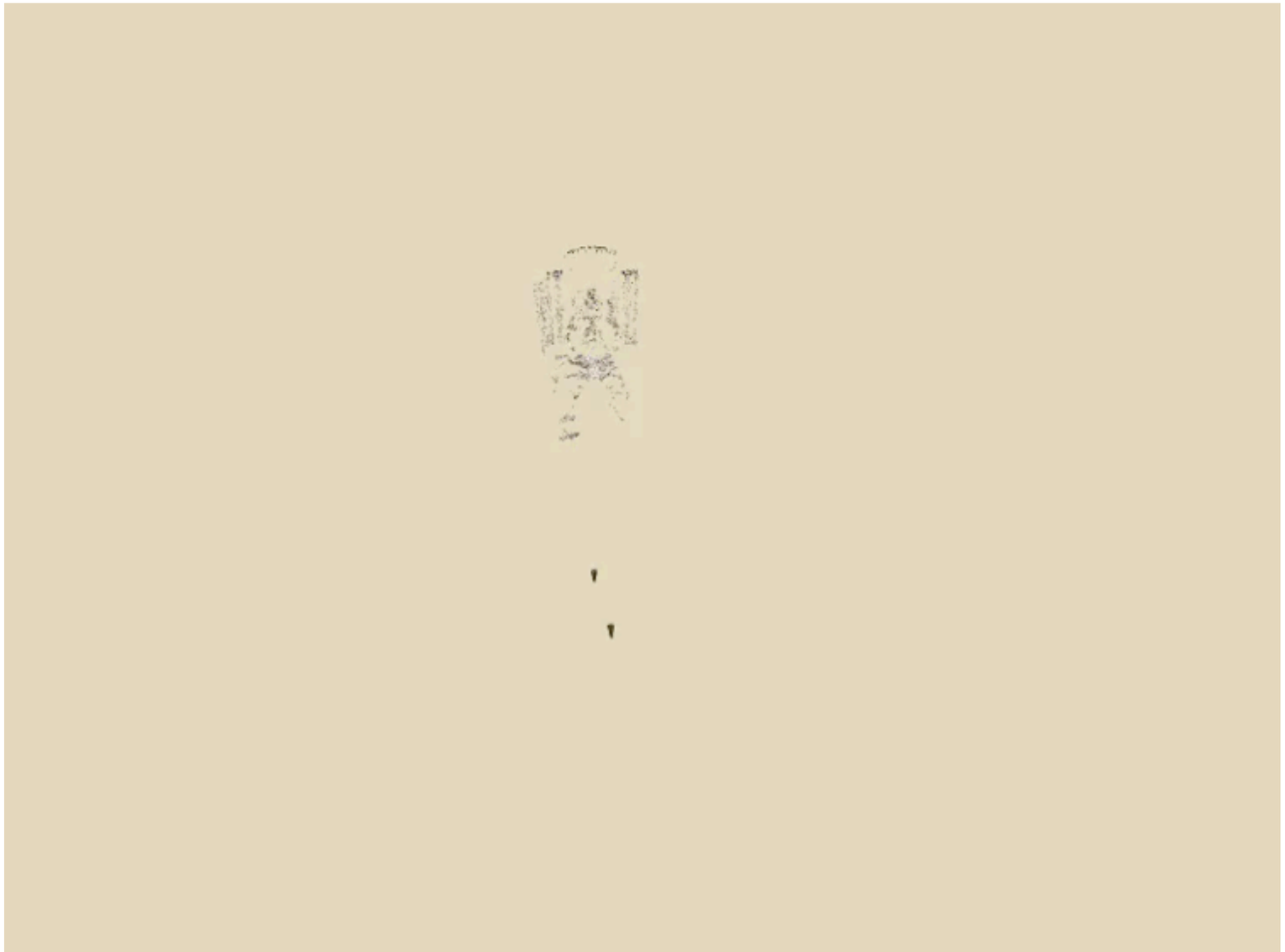
Pairwise feature matching

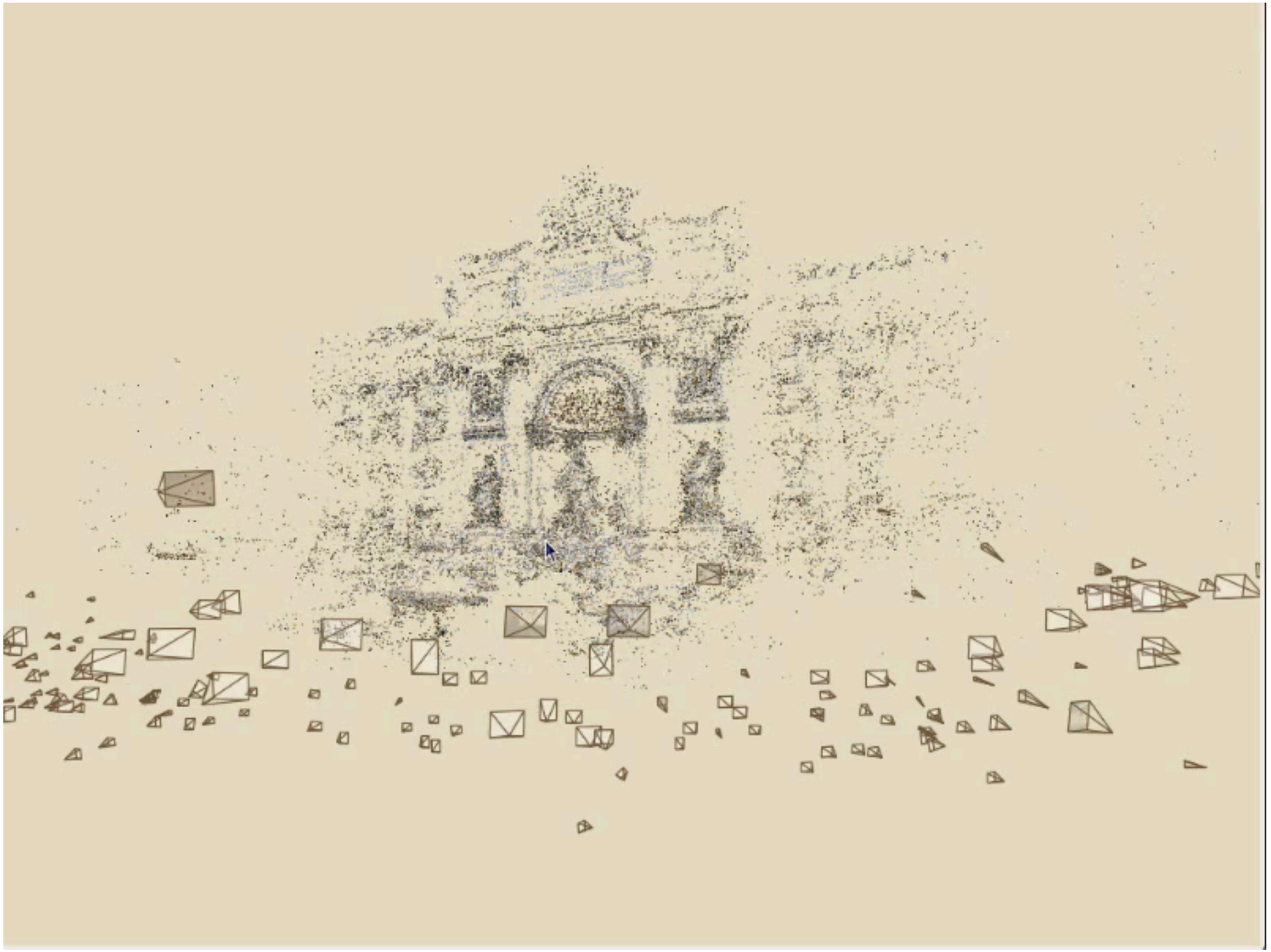
↓

Correspondence estimation

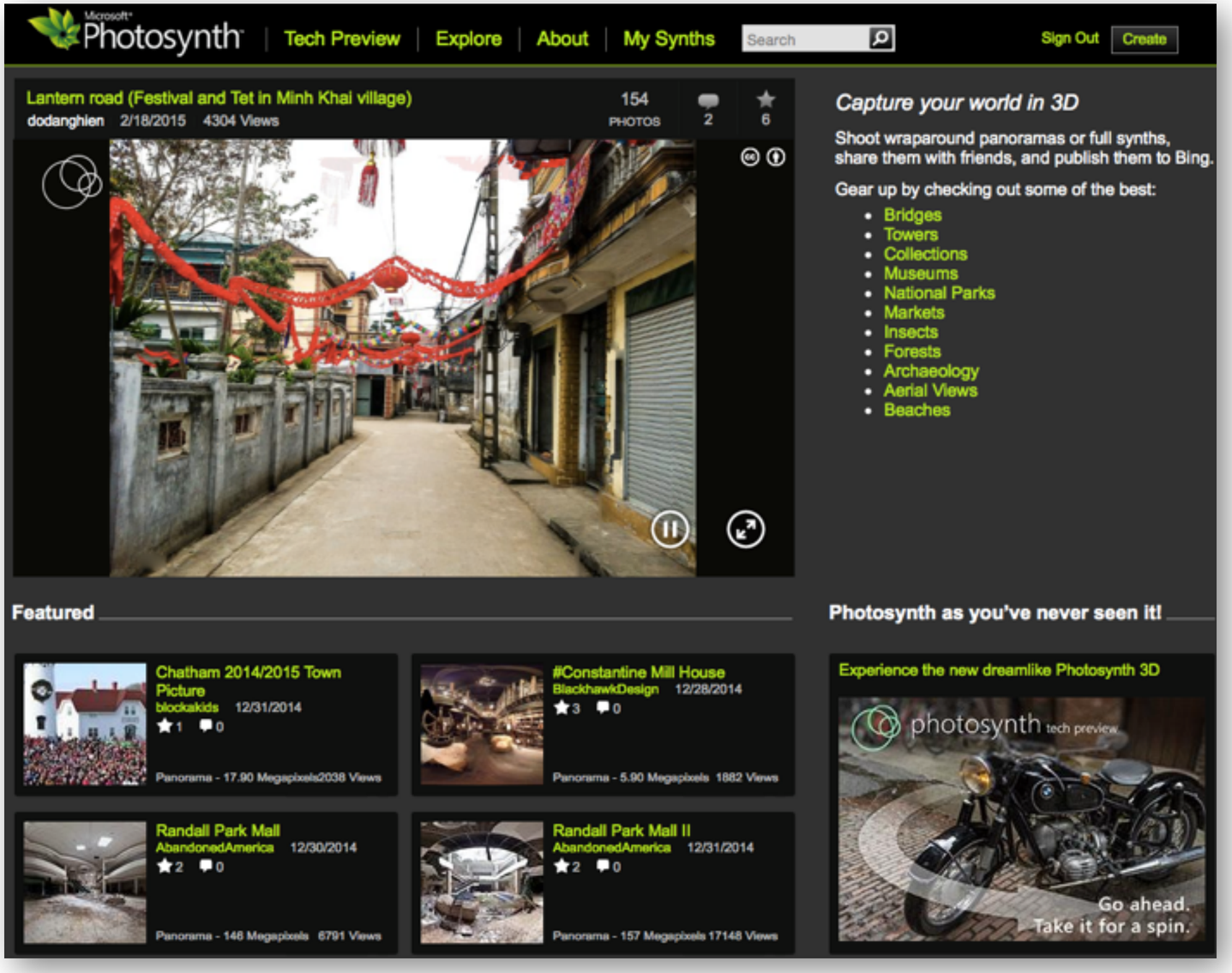
↓

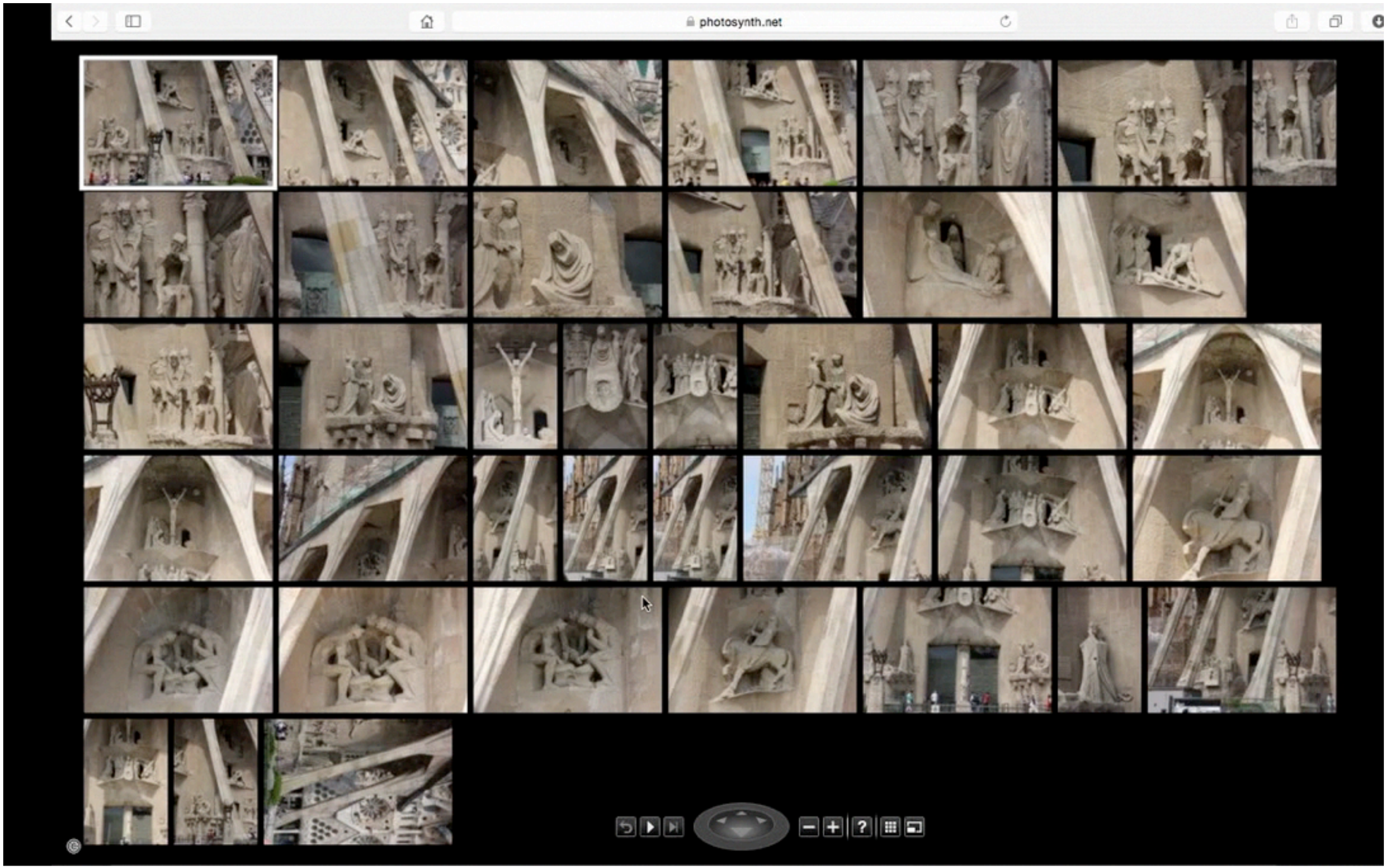
Incremental structure from motion

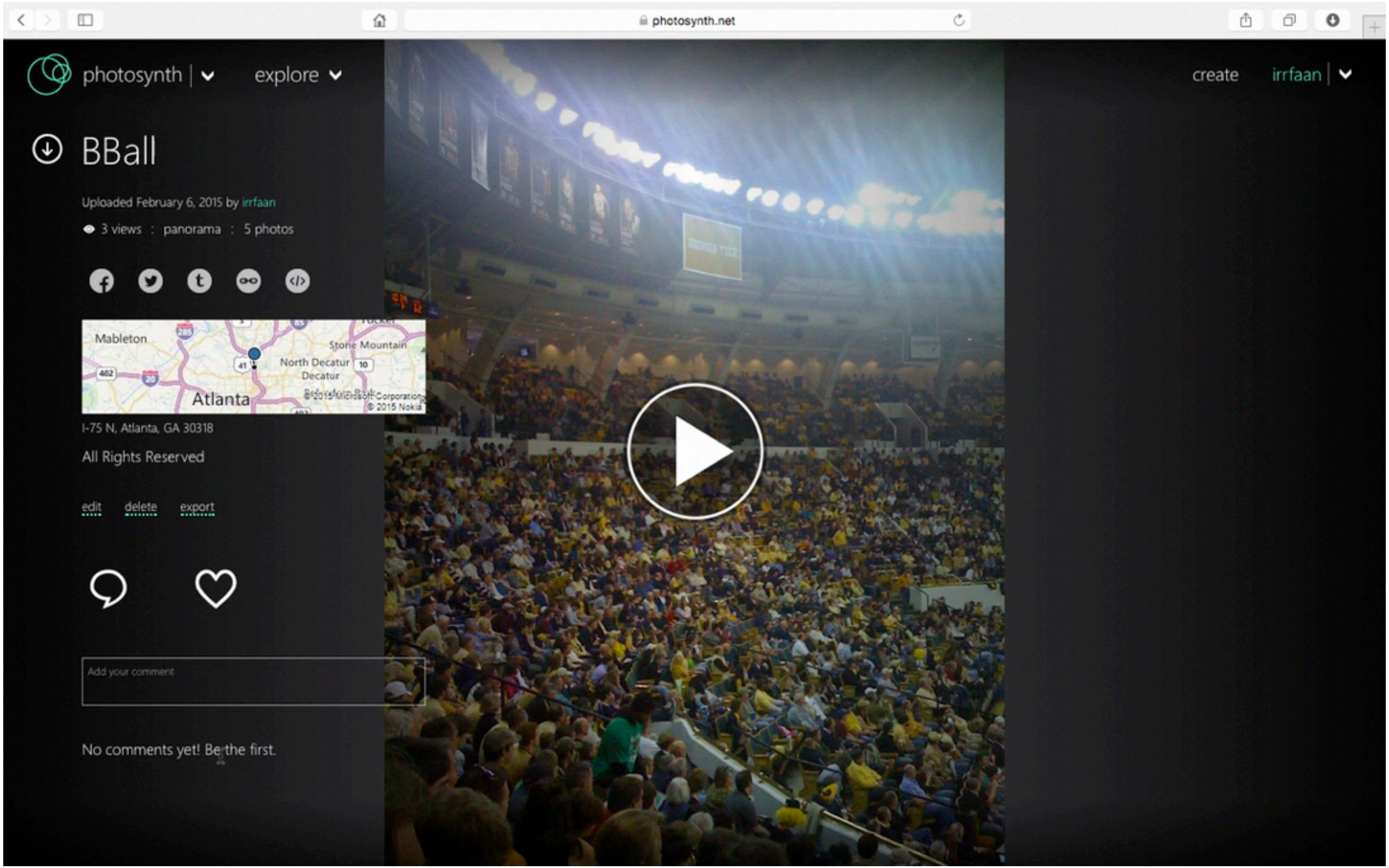


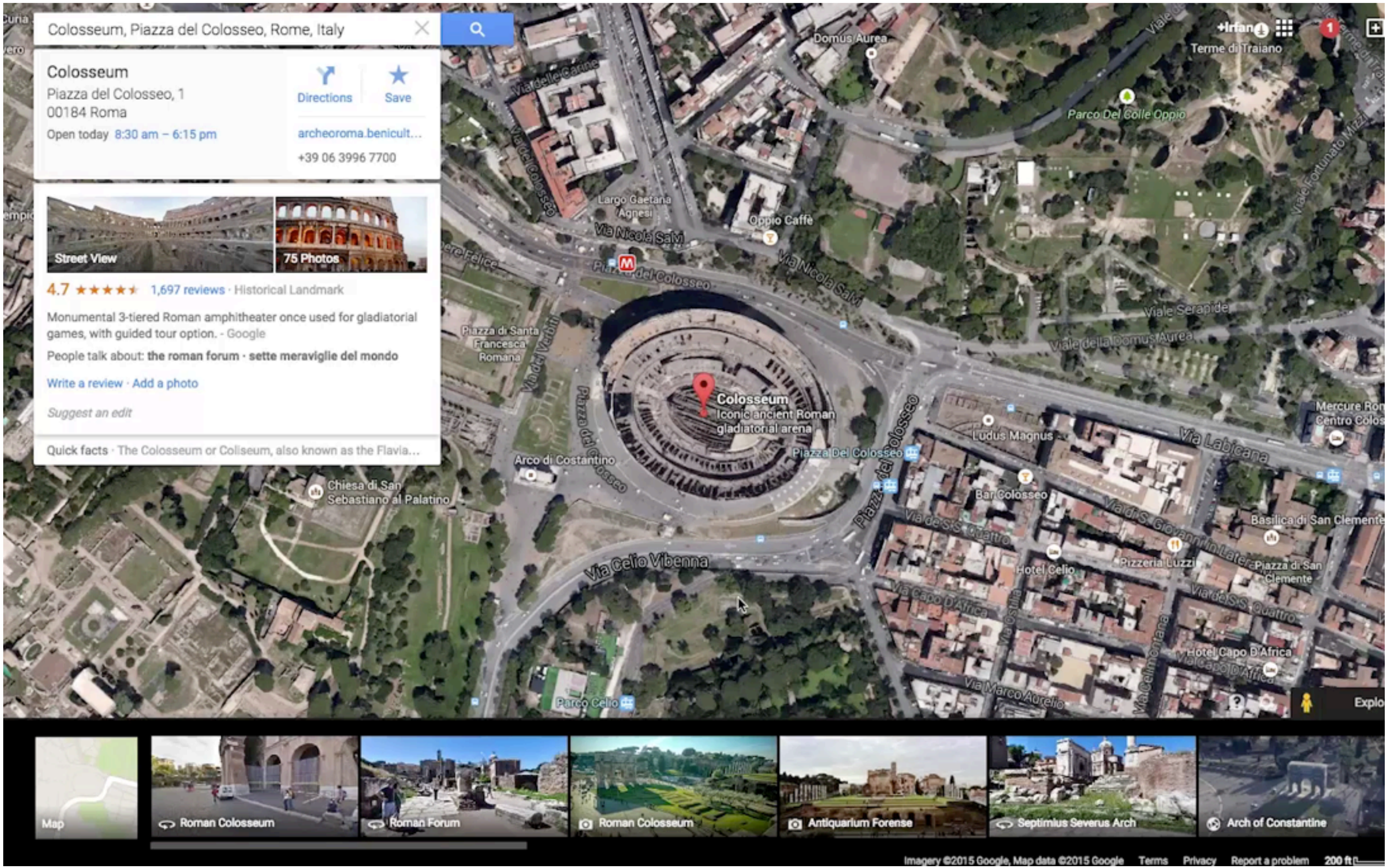


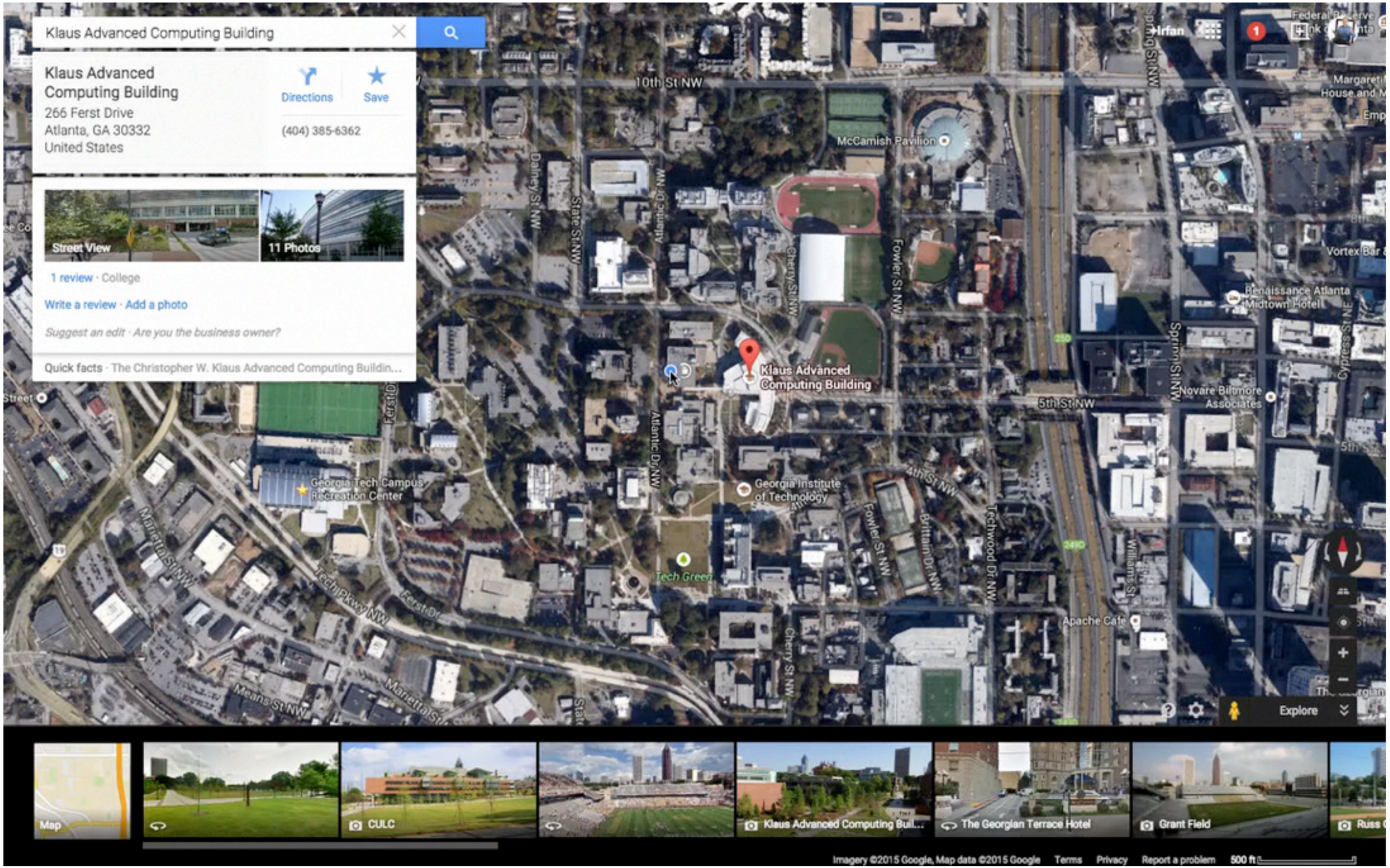
photosynth.net

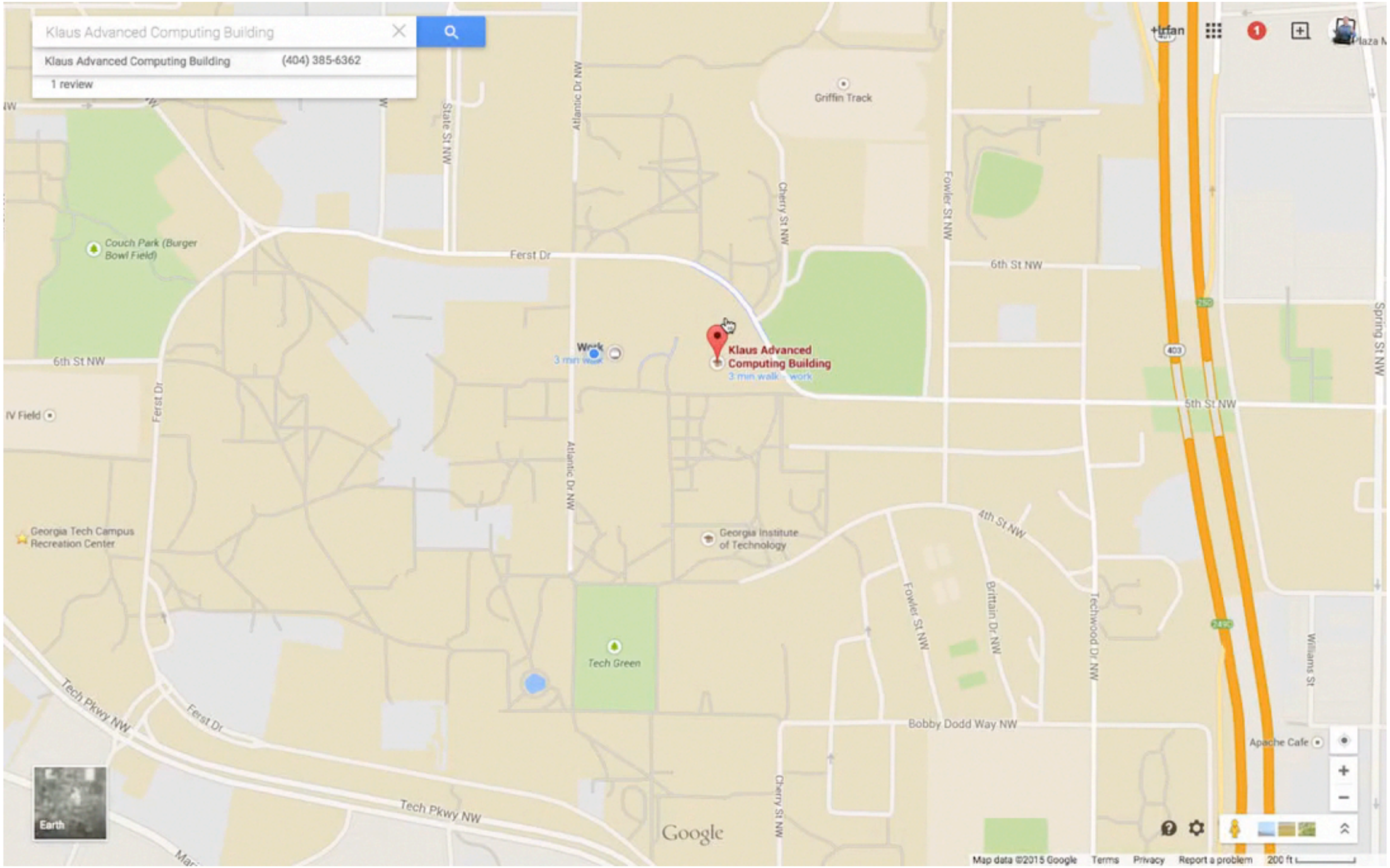












Google Streetview

Collecting Imagery ›

First off we need to actually drive around and photograph the locations to show in Street View. We pay close attention to many factors, including the weather and and the population density of various areas, to determine when and where can collect the best possible imagery.

Aligning imagery ›

To match each image to its geographic location on the map, we combine signals from sensors on the car that measure GPS, speed and direction. This helps us reconstruct the car's exact route, and even tilt and realign images as needed.

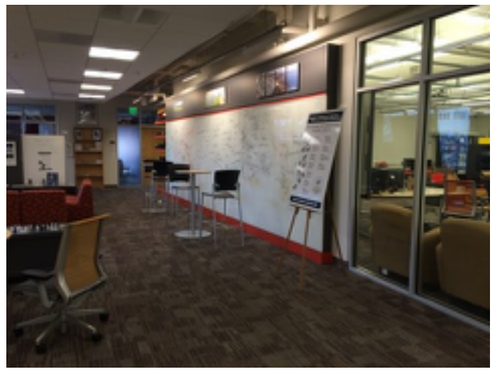
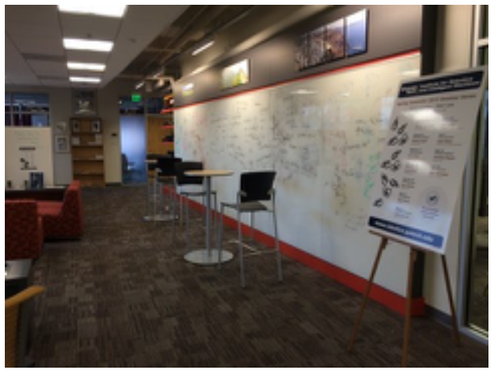
Turning photos into 360-degree panoramas ›

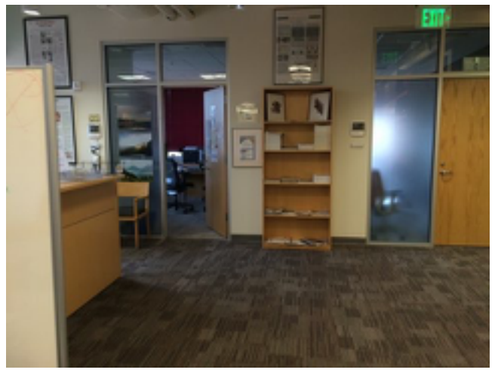
To avoid gaps in the panoramas, adjacent cameras take slightly overlapping pictures, and then we "stitch" the photos together into a single 360-degree image. We then apply special image processing algorithms to lessen "seams" and create smooth transitions.

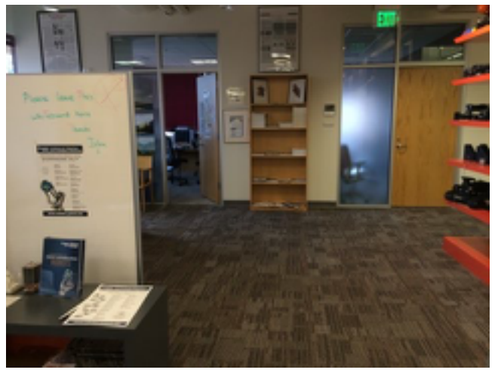
Showing you the right image ›

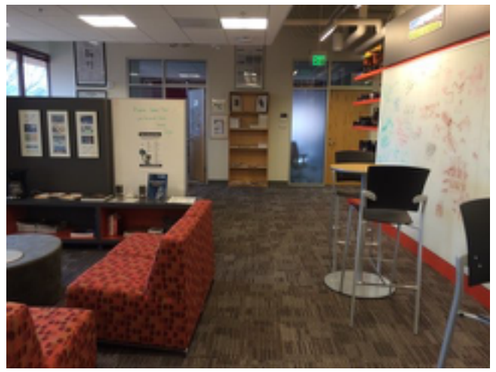
How quickly the car's three lasers reflect off surfaces tells us how far a building or object is, and enables us to construct 3D models. When you move to an area in the distance, the 3D model determines the best panorama to show you for that location.

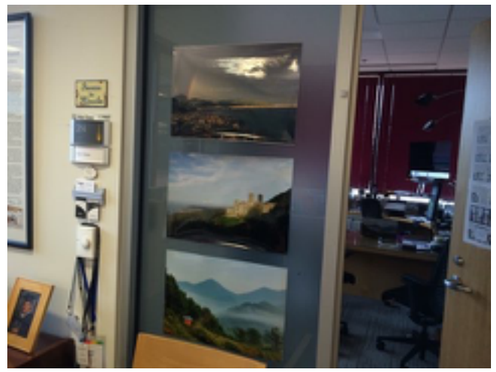
http://www.google.com/maps/about/behind-the-scenes/streetview/

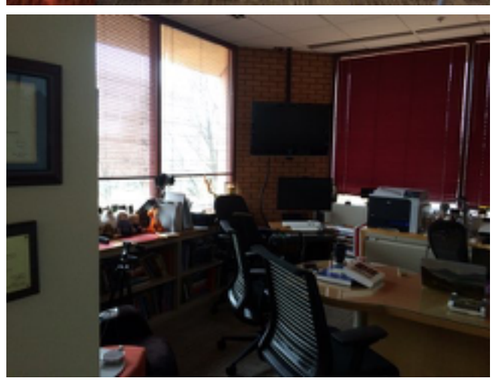



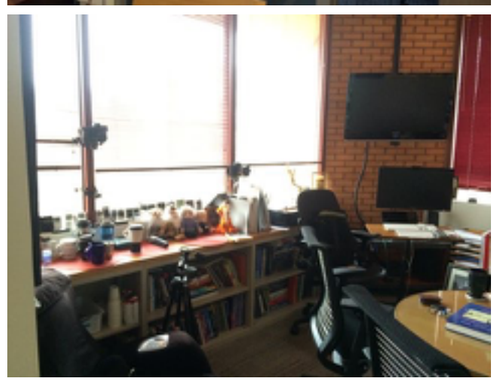
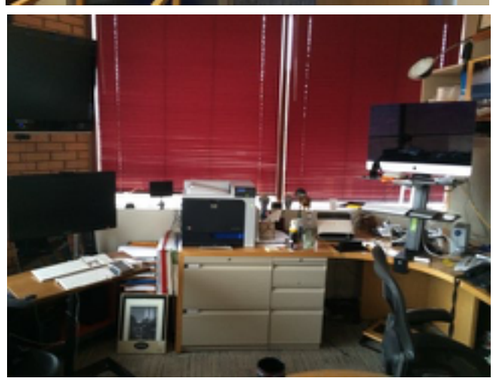
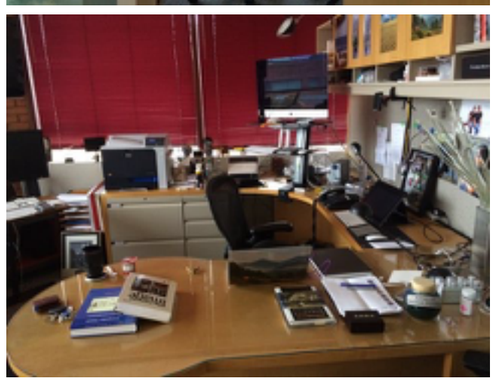


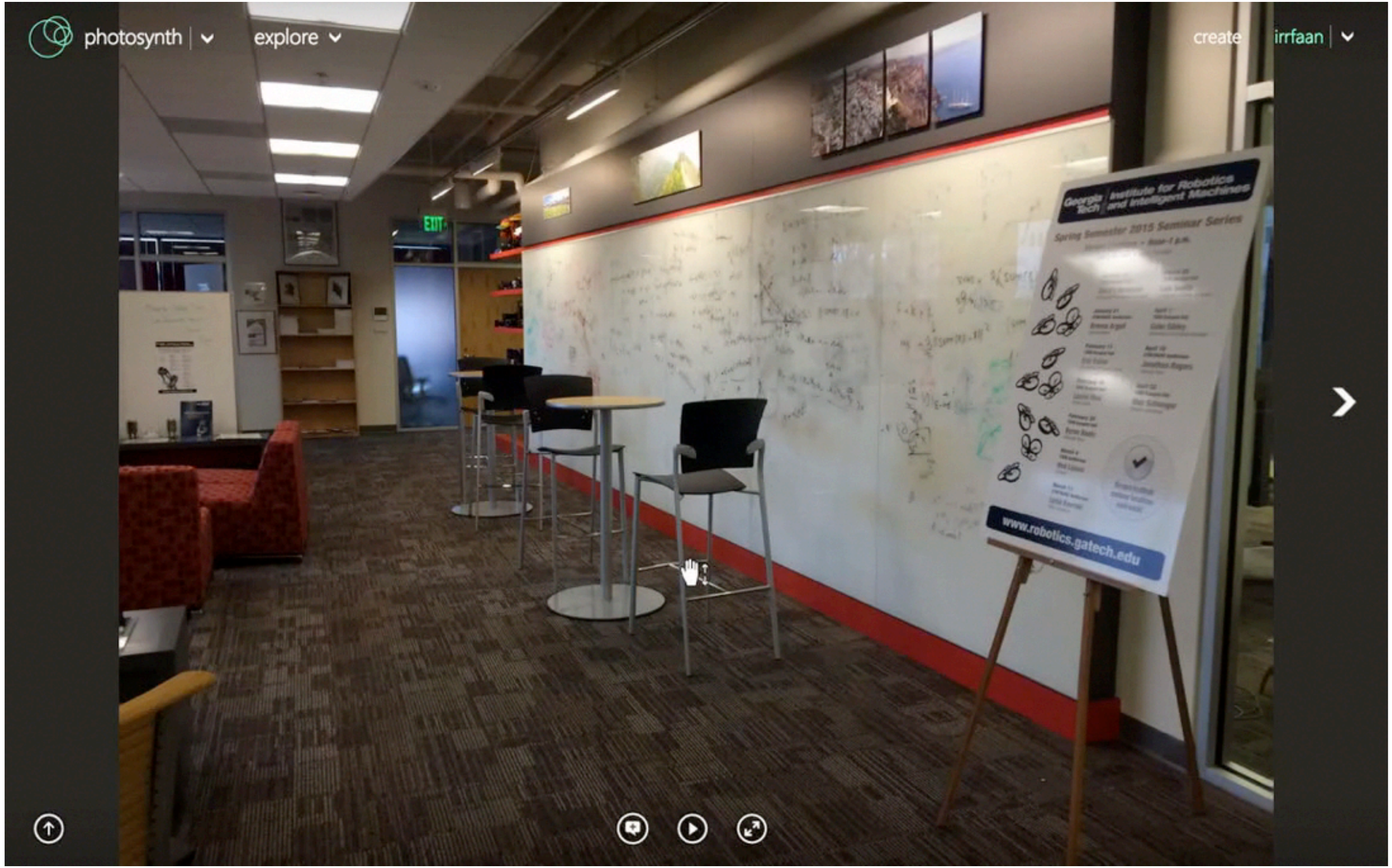








Summary

* Photos to show space/ environments
* Photo tourism / photosynth / street views

Further Reading

* Snavely, Seitz, Szeliski (2010) "Photo tourism: Exploring photo collections in 3D," ACM SIGGRAPH 2006
* Kushal, Self, Furukawa, Gallup, Hernandez, Curless, Seitz (2012) "Photo tours", 3DPTV 2012
* Szeliski (2010)

Credits

* For more information, see:
  * Richard Szeliski (2010) Computer Vision: Algorithms and Applications, Springer (Chapter 3)
* Some concepts in slides motivated by similar slides from Noah Snavely
* www.photosynth.net
* maps.google.com/phototours

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