
The Ancient
Secrets
Computer Vision

Logistics:

- Homework 2 out! Due a week from Thursday
  - Little longer than past ones, start early
- Google group
-

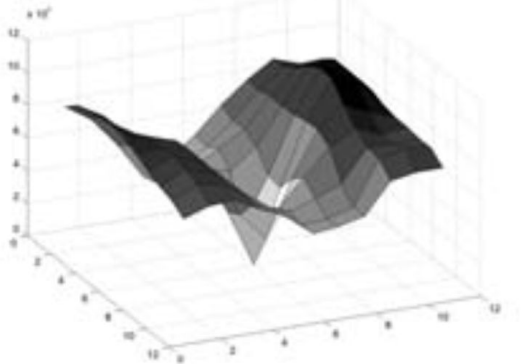

Previously
On
the
Ancient Secrets
of Computer Vision

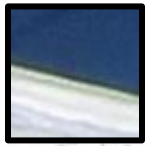
Self-difference

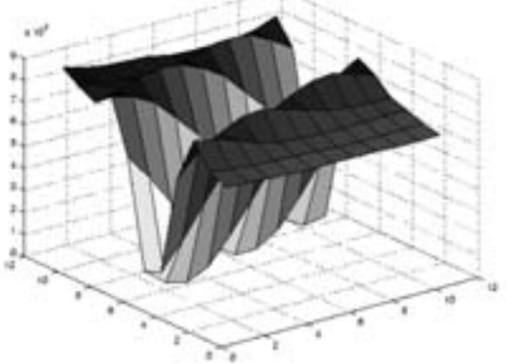
Sky: low everywhere

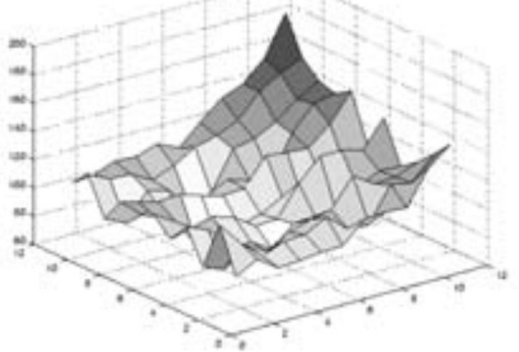
Edge: low along edge

Corner: mostly high









Self-difference is still expensive

- $\Sigma_{\mathbf{d}}\Sigma_{x,y}\ (I(x,y) - I(x+\mathbf{d}_x, y+\mathbf{d}_y))^2$
- Lots of summing
- Need an approximation
  - If you want the mathy details, Szeliski pg 212

Structure matrix

- Weighted sum of gradient information
  - $| \Sigma_i w_i I_x(i) I_x(i) \quad \Sigma_i w_i I_x(i) I_y(i) |$
  - $| \Sigma_i w_i I_x(i) I_y(i) \quad \Sigma_i w_i I_y(i) I_y(i) |$
- Can use Gaussian weighting (so many gaussians)
- Eigen vectors/values of this matrix summarize the distribution of the gradients nearby
- λ_1 and λ_2 are eigenvalues
  - λ_1 and λ_2 both small: no gradient
  - $\lambda_1 >> \lambda_2$: gradient in one direction
  - λ_1 and λ_2 similar: multiple gradient directions, corner

Estimating smallest eigen value

- A few methods:
  - $\det(S) = \lambda_1 * \lambda_2$
  - $\text{trace}(S) = \lambda_1 + \lambda_2$
- $\det(S) - \alpha\ \text{trace}(S)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1+\lambda_2)^2$
- $\det(S) / \text{trace}(S) = \lambda_1\lambda_2/(\lambda_1+\lambda_2)$
- If these estimates are large, λ_2 is large

Harris Corner Detector

- Calculate derivatives I_x and I_y
- Calculate 3 measures I_xI_x, I_yI_y, I_xI_y
- Calculate weighted sums
  - Want a weighted sum of nearby pixels, guess what this is?
  - Gaussian!
- Estimate response based on smallest eigen value
- Non-max suppression (just like canny)

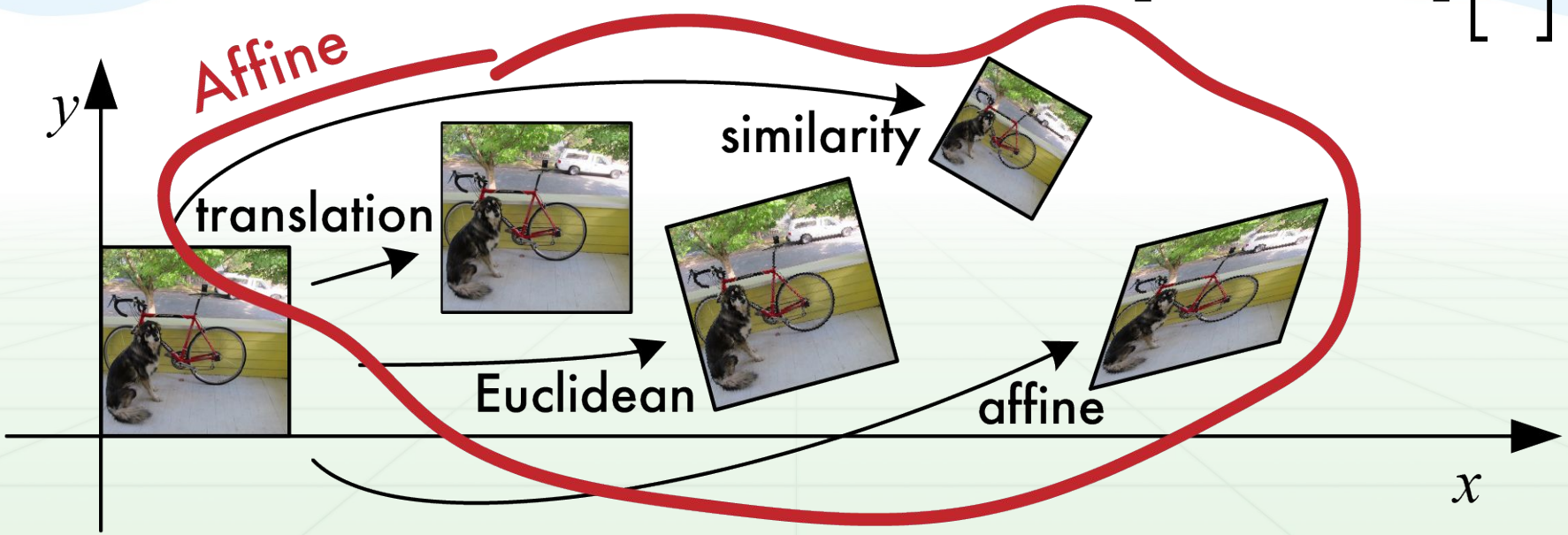
Matching patches: descriptors!

- We want a way to represent an image patch
- Can be very simple, just pixels!
- Finding matching patch is easy, distance metric:
  - $\Sigma_{x,y}\ (I(x,y) - J(x,y))^2$
  - What problems are there with just using pixels?
- Descriptors can be more complex
  - Gradient information
  - How much context?
  - Edges, etc. we'll talk more about descriptors later!

Affine: scale, rotate, translate, shear

General case of 2x3 matrix

$$\mathbf{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Combinations are still affine

Say you want to translate, then rotate, then translate back, then scale.

$\mathbf{x'} = \mathbf{S}\ \mathbf{t}\ \mathbf{R}\ \mathbf{t}\ \bar{\mathbf{x}} = \mathbf{M}\ \bar{\mathbf{x}}$,

If $\mathbf{M} = (\mathbf{S}\ \mathbf{t}\ \mathbf{R}\ \mathbf{t})$

$\mathbf{M}$ is still affine transformation

Wait, but these are all 2x3, how to we multiply them together?

Added row to transforms

$$\bar{\mathbf{x}}' = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

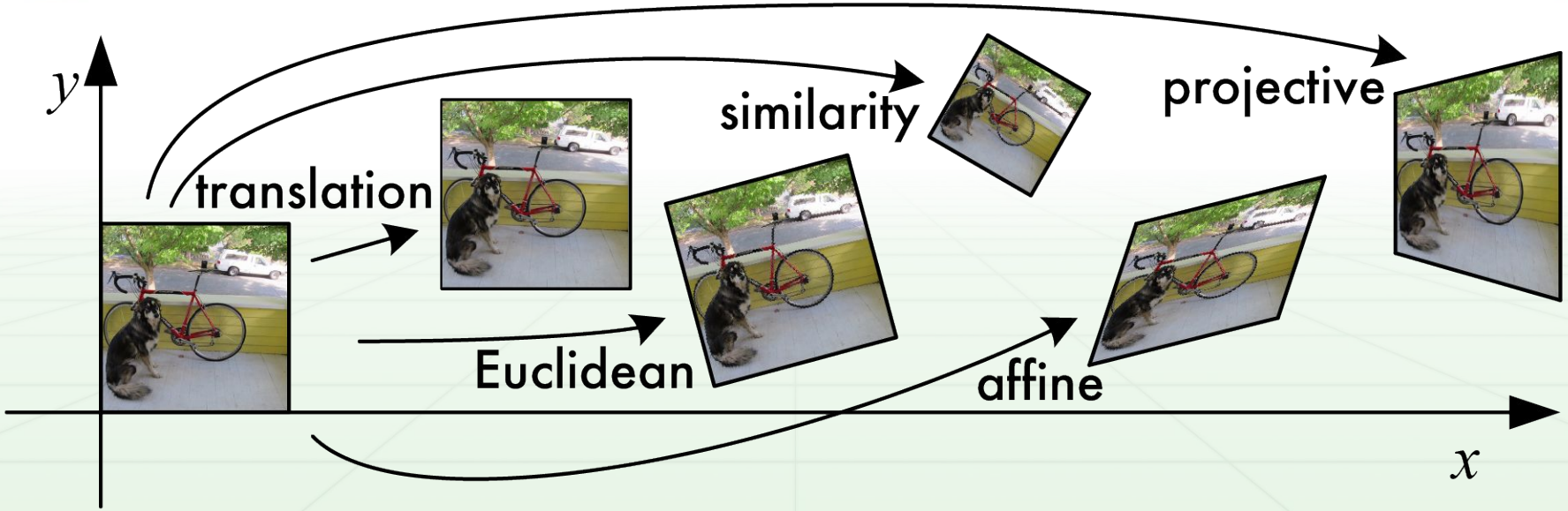
$$\bar{\mathbf{x}}' = \begin{bmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\bar{\mathbf{x}}' = \begin{bmatrix} a & -b & dx \\ b & a & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\bar{\mathbf{x}}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Projective transform

- AKA perspective transform or homography
- Wait but affine was any 2x3 matrix...



Projective transform

- AKA perspective transform or homography
- Wait but affine was any 2x3 matrix...
- Homography is general 3x3 matrix
- Multiplication by scalar: equivalent projection
  - 3*H ~ H

$$\tilde{\mathbf{x}}' = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$$

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\,\tilde{\mathbf{x}}$$

Using homography to project point

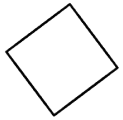
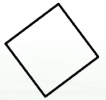
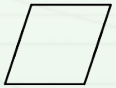
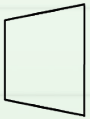
- Multiply **x~** by **H~** to get **x'~**
- Convert to **x'** by dividing by **w'~**

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\,\tilde{\mathbf{x}}$$

$$\begin{bmatrix} \tilde{x}' \\ \tilde{y}' \\ \tilde{w}' \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$$

$$\bar{\mathbf{x}} = \tilde{\mathbf{x}} / \tilde{w}$$

Lots to choose from

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} \mid \mathbf{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} \mid \mathbf{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} \mid \mathbf{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

Say we want affine transformation

- Have our matched points
- Want to estimate **A** that maps from **x** to **x’**
- **xA = x’**
- How many degrees of freedom?
  - 6
- How many knowns do we get with one match? **mA = n**
  - 2
  - $n_x = a_{00}*m_x + a_{01}*m_y + a_{02}*1$
  - $n_y = a_{10}*m_x + a_{11}*m_y + a_{12}*1$

$$\mathbf{x'} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$


Chapter
Seven
Stitching, RANSAC
SIFT, and HOG

Say we want affine transformation

- How many knowns do we get with one match? **mA = n**
  - $n_x = a_{00} * m_x + a_{01} * m_y + a_{02} * 1$
  - $n_y = a_{10} * m_x + a_{11} * m_y + a_{12} * 1$
  - Solve **M a = b**
    - $M^T M a = M^T b$
    - $a = (M^T M)^{-1} M^T b$
  - Still works if overdetermined
    - WHY???

$$\underset{\mathbf{M}}{\begin{bmatrix} m_{x1} & m_{y1} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x1} & m_{y1} & 1 \\ m_{x2} & m_{y2} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x2} & m_{y2} & 1 \\ m_{x3} & m_{y3} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x3} & m_{y3} & 1 \\ & & & \cdots & & \\ & & & \cdots & & \end{bmatrix}} \underset{\mathbf{a}}{\begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix}} = \underset{\mathbf{b}}{\begin{bmatrix} n_{x1} \\ n_{y1} \\ n_{x2} \\ n_{y2} \\ n_{x3} \\ n_{y3} \end{bmatrix}}$$

Linear least squares

Want to solve overdetermined linear system:

- $M\ a = b$

Want to minimize squared error:

$|| b - M\ a ||^2 =$

Linear least squares

Want to solve overdetermined linear system:

$$- M a = b$$

Want to minimize squared error:

$$|| b - M a ||^2 =$$

$$(b - M a)^T (b - M a)$$

Linear least squares

Want to solve overdetermined linear system:

$$- M a = b$$

Want to minimize squared error:

$$|| b - M a ||^2 =$$

$$(b - M a)^T(b - M a) =$$

$$b^Tb - a^TM^Tb - b^TMa + a^TM^TMa$$

Linear least squares

Want to solve overdetermined linear system:

$$M a = b$$

Want to minimize squared error:

$$|| b - M a ||^2 =$$

$$(b - M a)^T(b - M a) =$$

$$b^Tb - a^TM^Tb - b^TMa + a^TM^TMa =$$

$$b^Tb - 2a^TM^Tb + a^TM^TMa$$

Linear least squares

Want to minimize squared error: $|| b - M a ||^2 =$

$b^T b - 2a^T M^T b + a^T M^T M a$

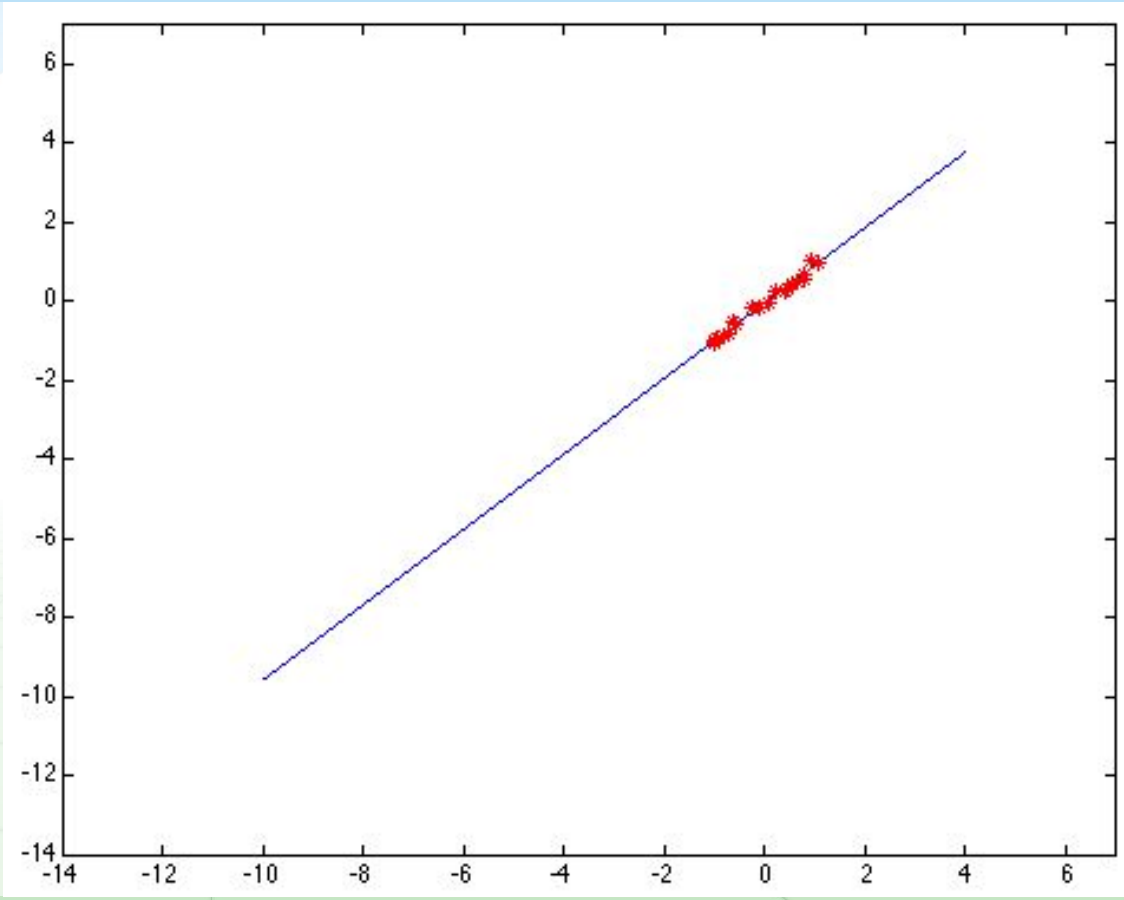
This is convex and minimized when gradient = 0. So we take the derivative wrt a and set = 0.

$-M^T b + (M^T M)a = 0$

$(M^T M)a = M^T b$

$a = (M^T M)^{-1} M^T b$ yay!!!!

So how does least squares do?

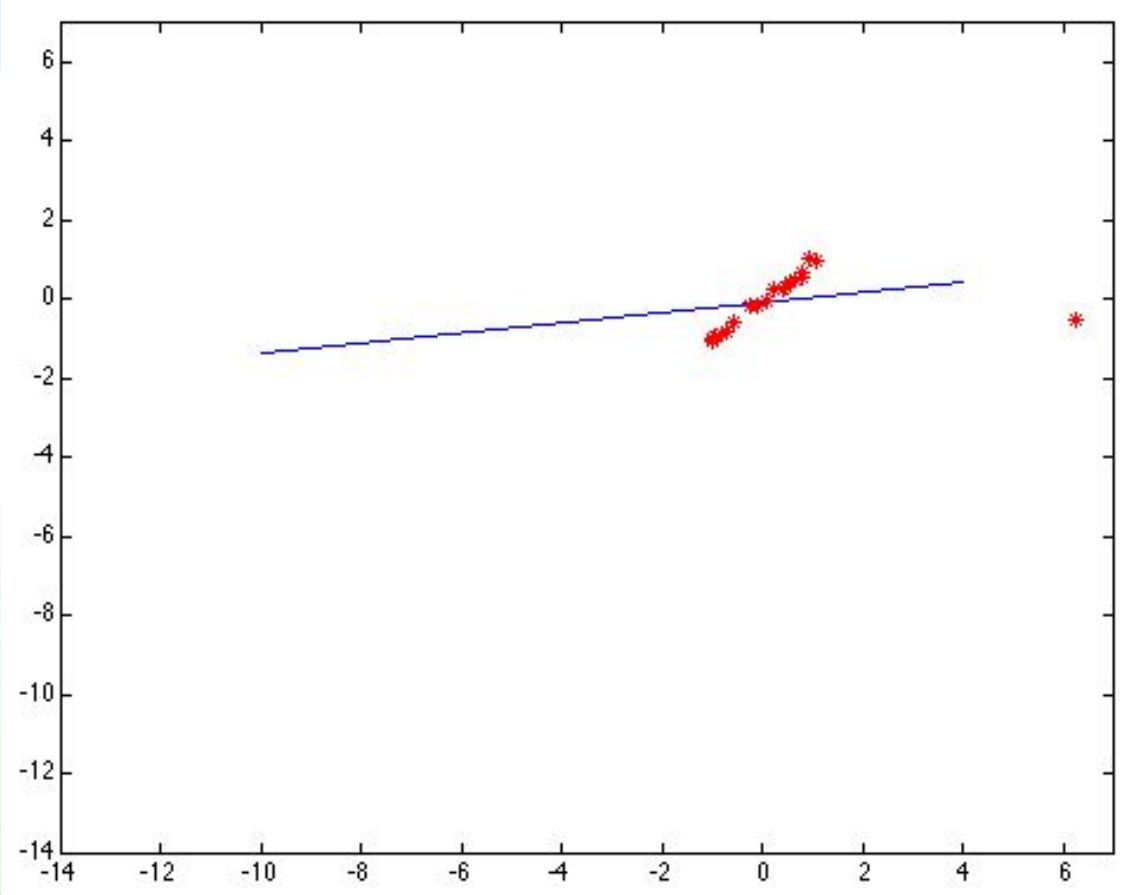


So how does least squares do?

Error based on **squared** residual

Very scared of being wrong, even for just one point

Very bad at handling outliers in data

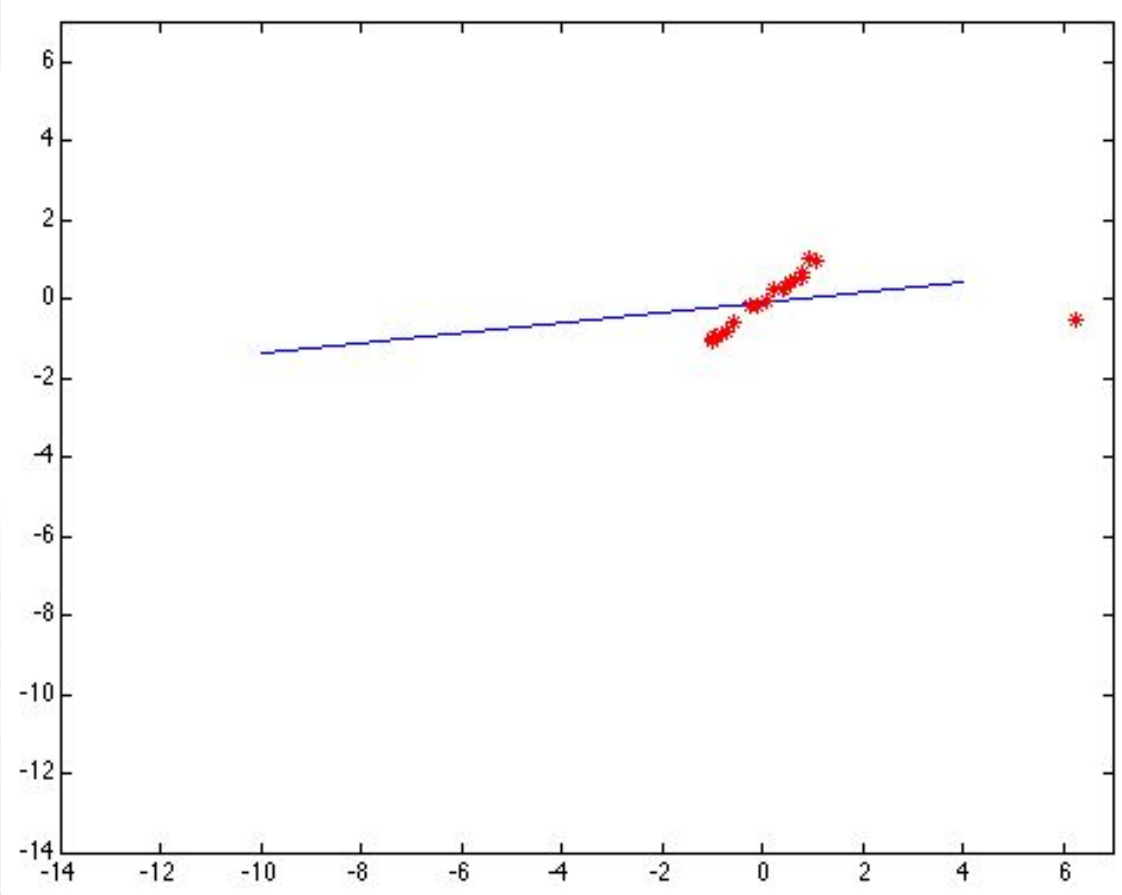


So how does least squares do?

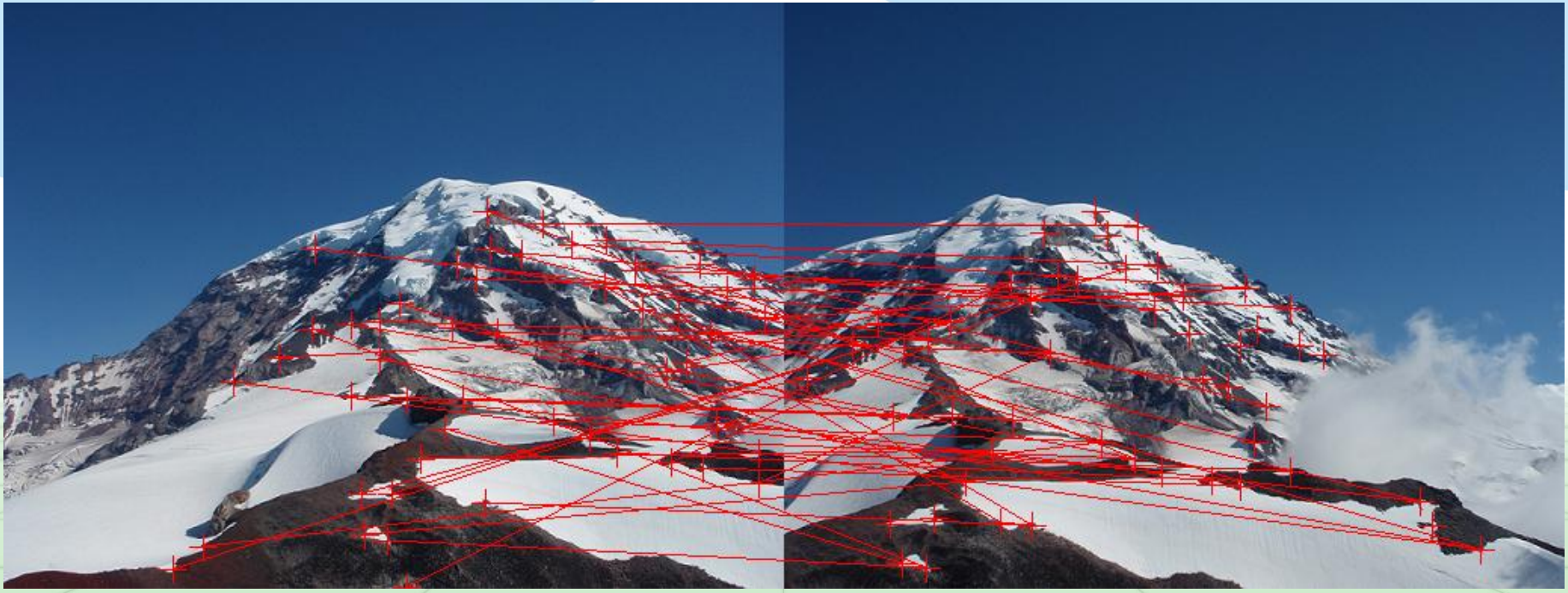
Error based on **squared** residual

Very scared of being wrong, even for just one point

Very bad at handling outliers in data



Not a problem for us, our data is perfect…

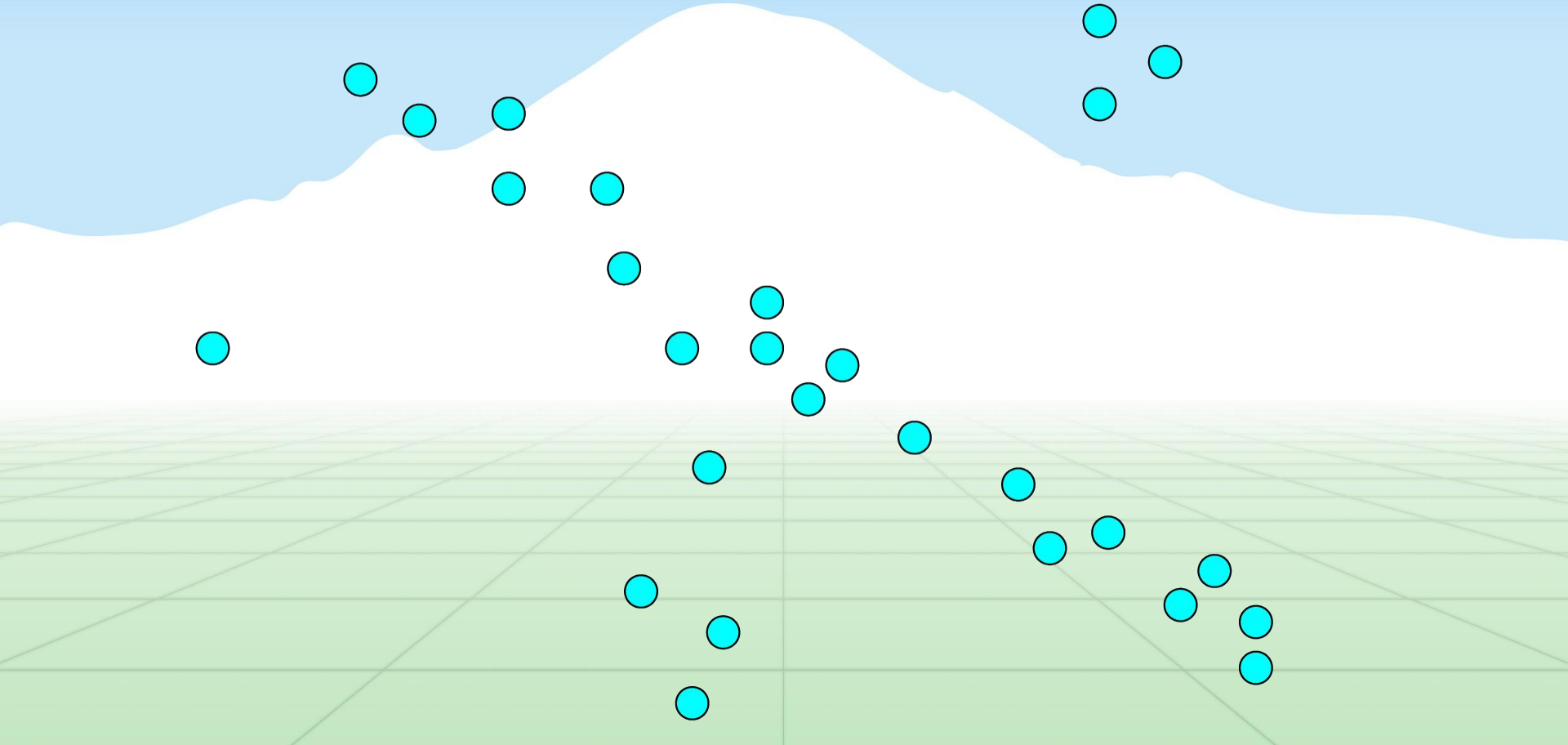


Oh wait…

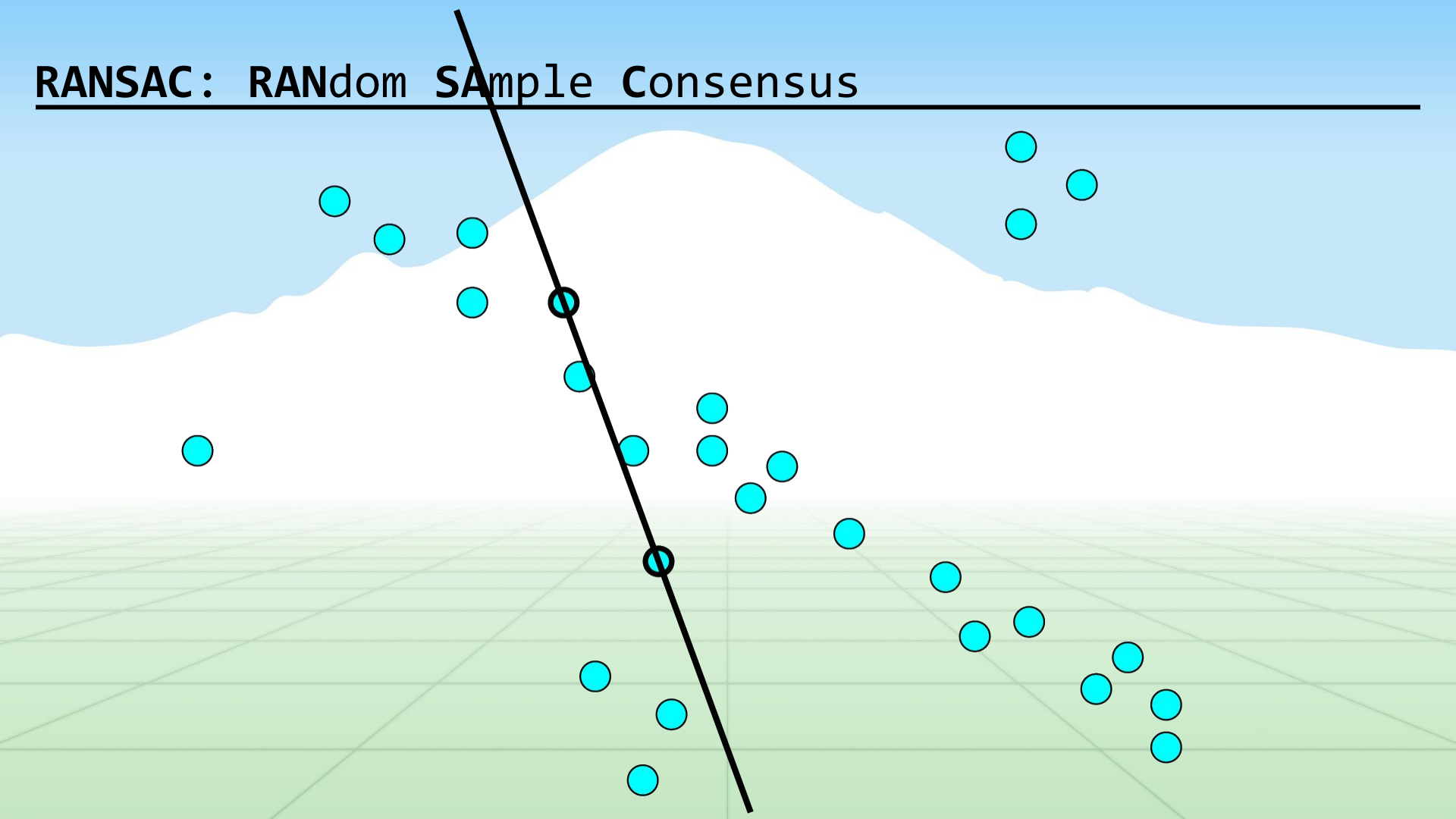
RANSAC: RANdom **SA**mple **C**onsensus

- How can we fit model to inliers but ignore outliers?
- Try a bunch of models, see which ones are best!
- Inliers will all agree on a model
- Outliers are basically random, will not agree

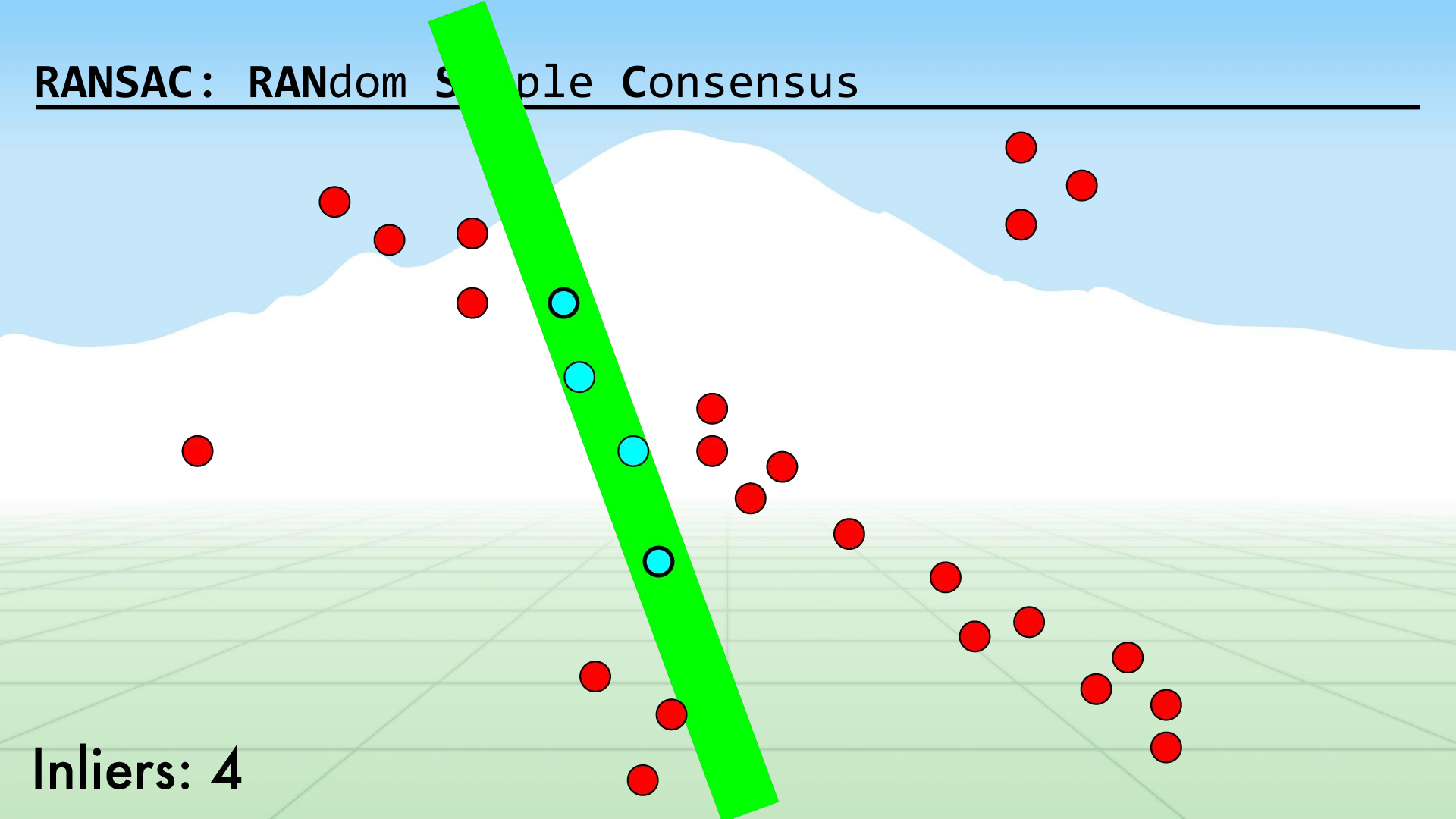
RANSAC: RANdom SAmple Consensus



RANSAC: RANdom SAmple Consensus

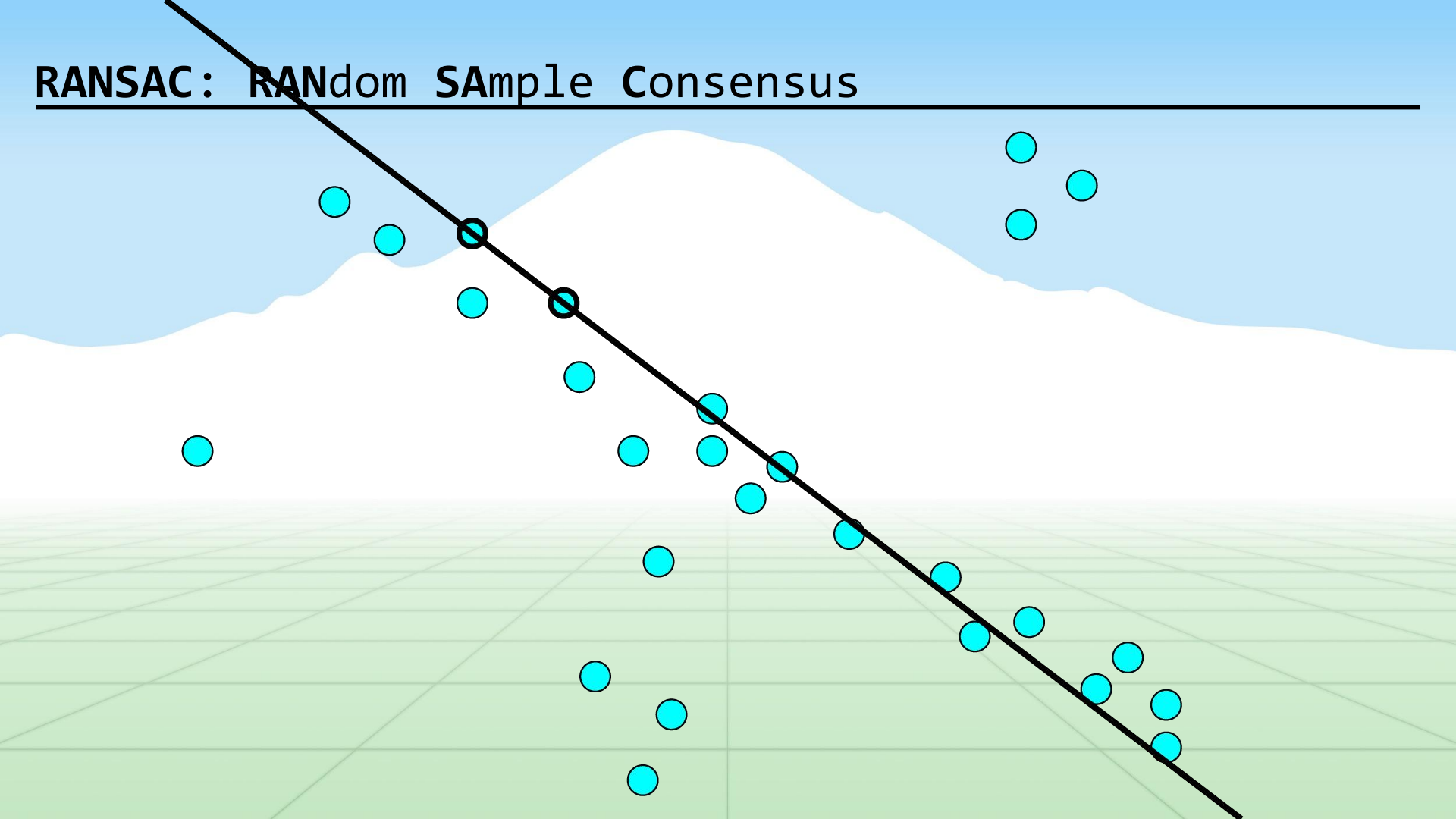


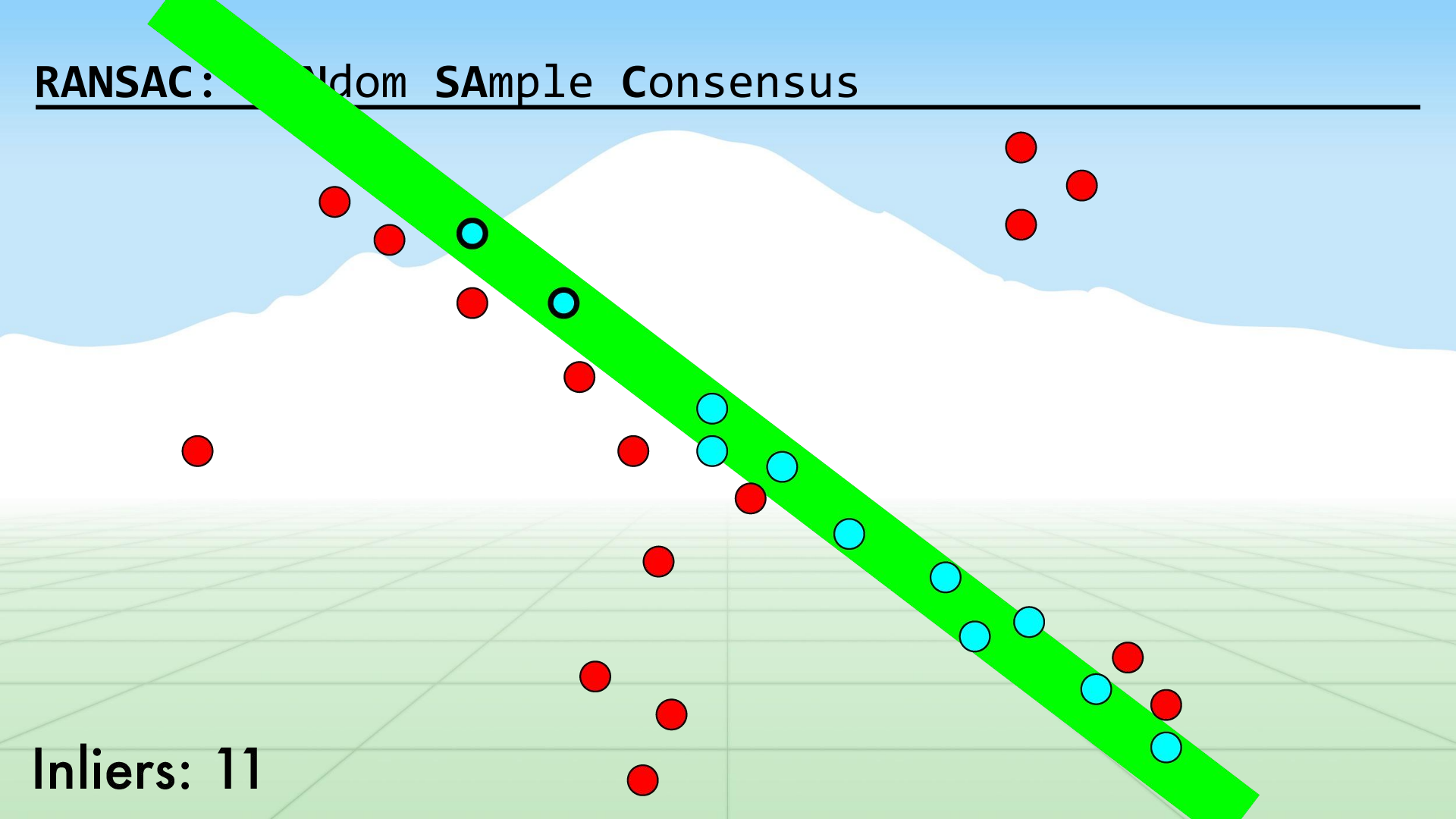
RANSAC: RANdom Sample Consensus



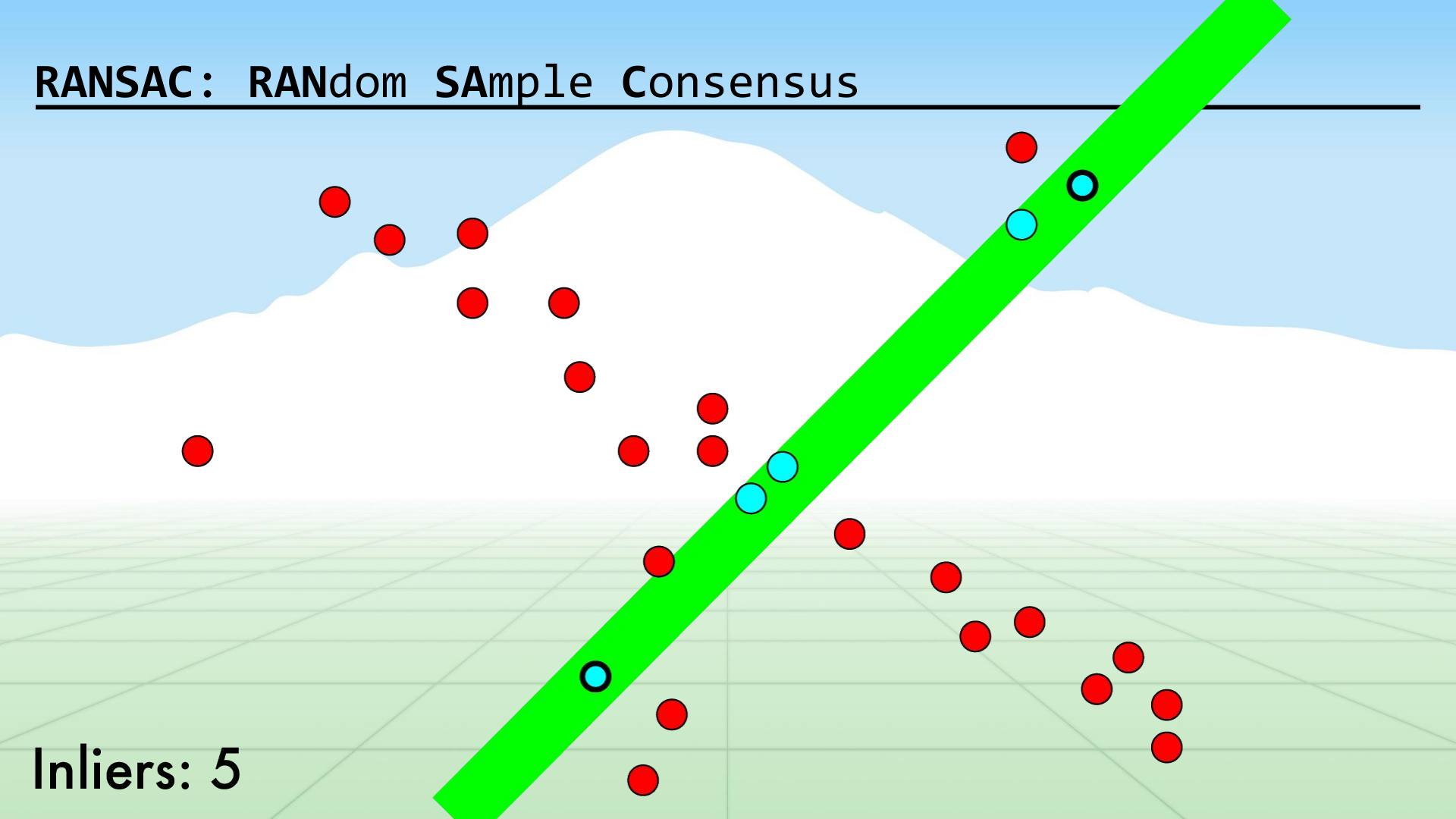
Inliers: 4

RANSAC: RANdom SAmple Consensus



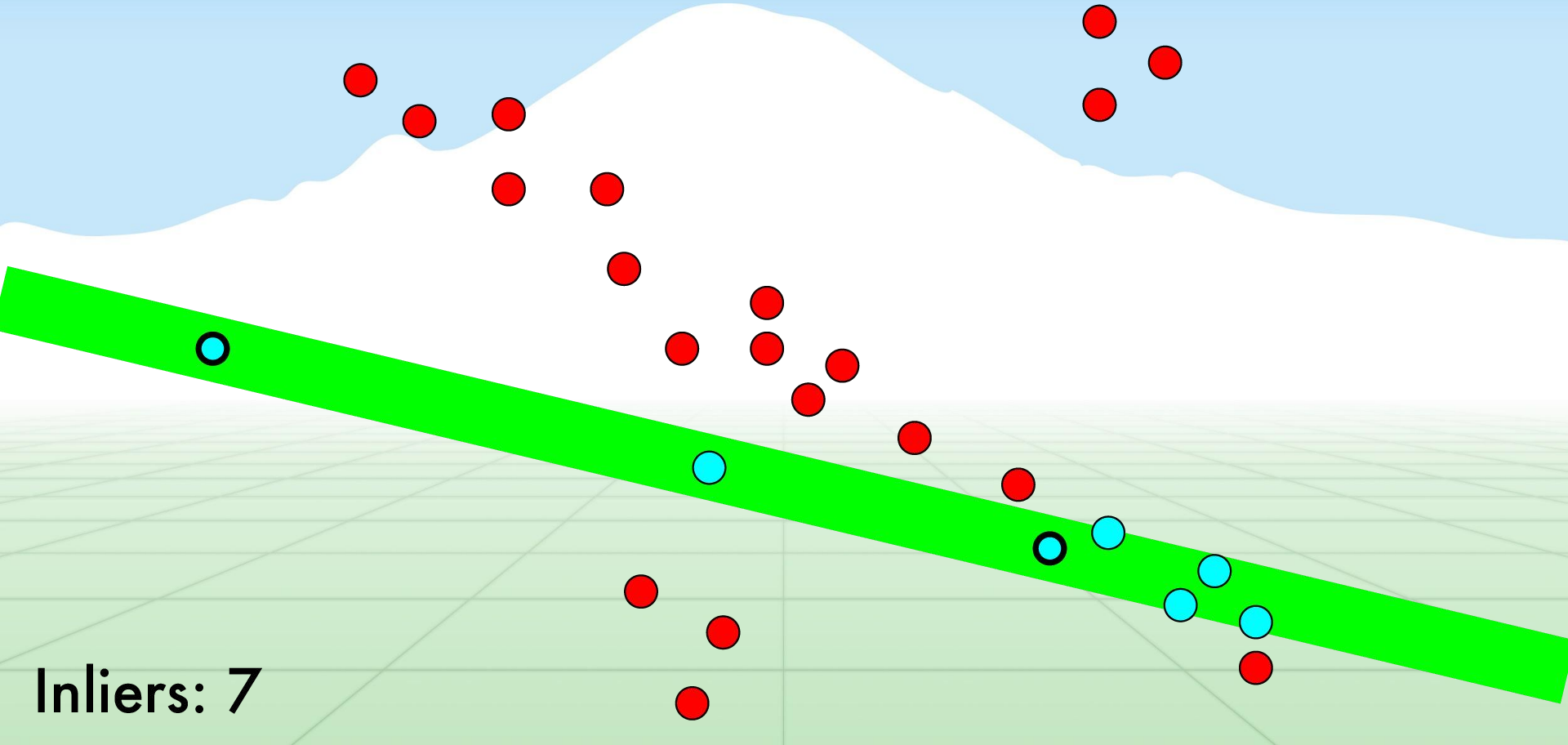

RANSAC: RANdom SAmple Consensus
Inliers: 11

RANSAC: RANdom SAmple Consensus



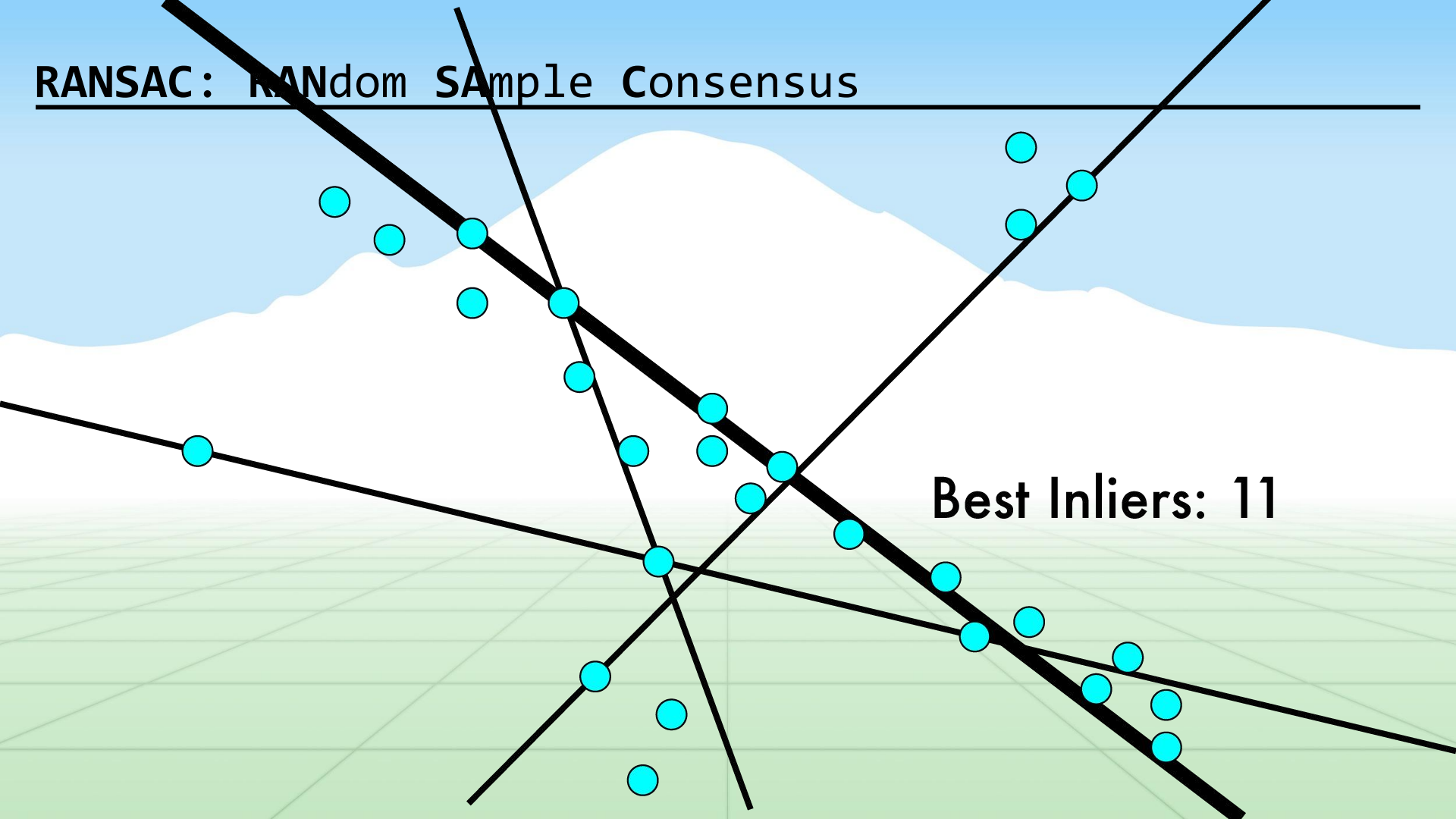
Inliers: 5

RANSAC: RANdom SAmple Consensus



Inliers: 7

RANSAC: RANdom SAmple Consensus



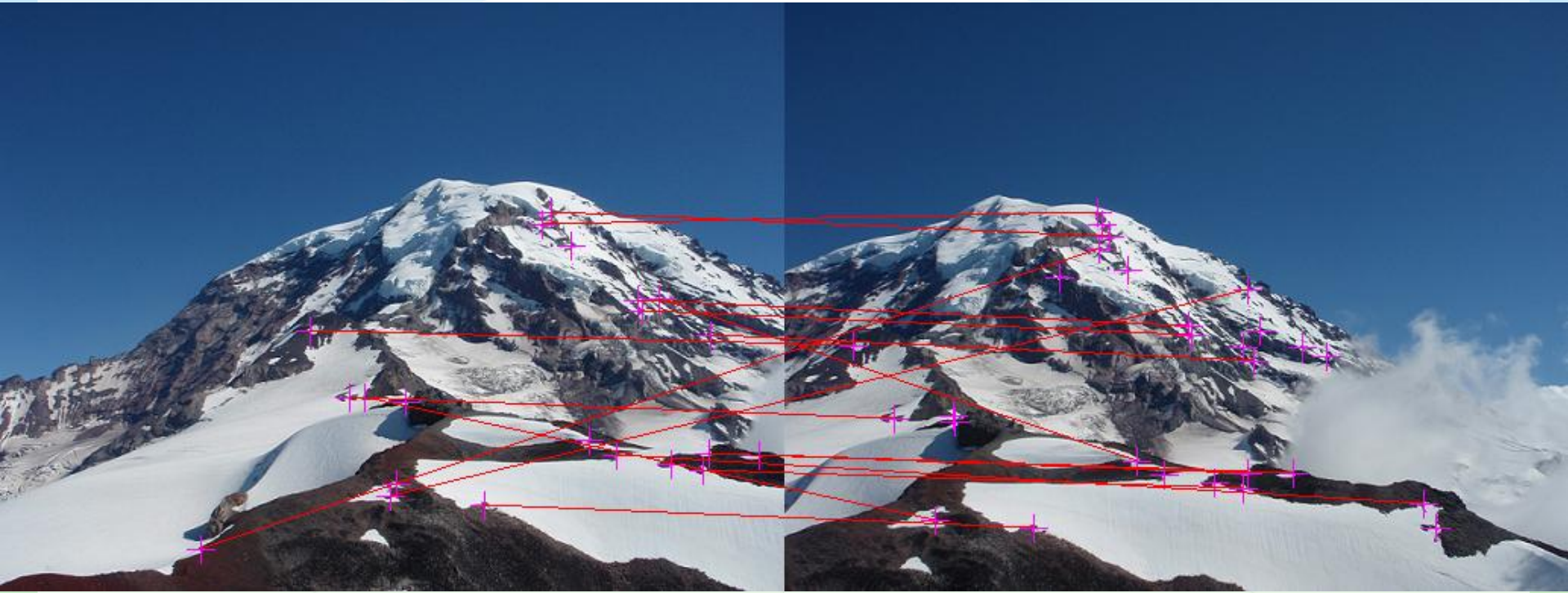
RANSAC: RANdom SAmple Consensus

- Parameters: data, model, n points to fit model, k iterations, t threshold, d “good” fit cutoff

```
bestmodel = None
bestfit = INF
While i < k:
     sample = draw n random points from data
     Fit model to sample
     inliers = data within t of model
     if inliers > bestfit:
          Fit model to all inliers
          bestfit = fit
          bestmodel = model
          if inliers > d:
               return model
return bestmodel
```

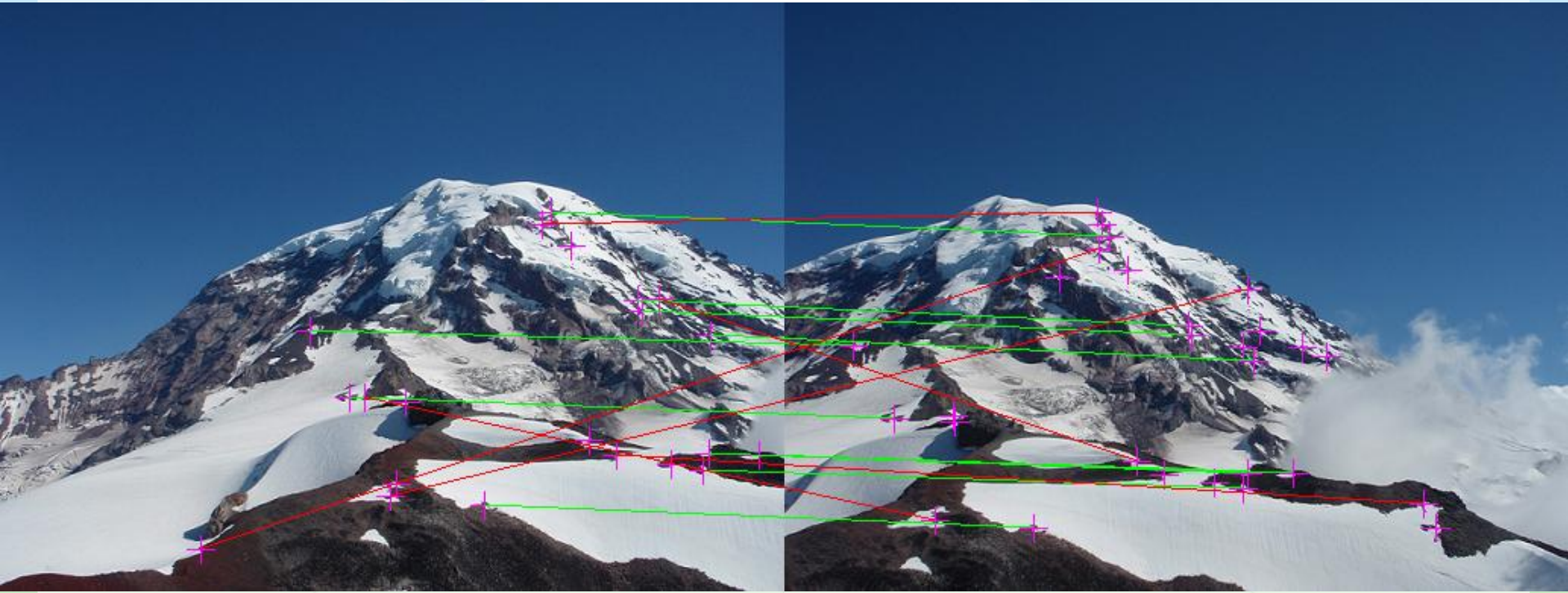
RANSAC: RANdom **SA**mple **C**onsensus

- Works well even with extreme noise.



RANSAC: RANdom **SA**mple **C**onsensus

- Works well even with extreme noise.



RANSAC: RANdom SAmple Consensus

- Parameters: data, model, n points to fit model, k iterations, t threshold, d “good” fit cutoff
- Lots of tunable parameters
- Want high probability of recovering “right” model
- t: often quite small, assume “good” inliers
- n: should be just enough to fit model, no extra
- k: can be very high
- d: should be >> n

We can estimate affine...

- How many knowns do we get with one match? **mA = n**
  - $n_x = a_{00} * m_x + a_{01} * m_y + a_{02} * 1$
  - $n_y = a_{10} * m_x + a_{11} * m_y + a_{12} * 1$
  - Solve **M a = b**
    - $M^T M\ a = M^T\ b$
    - $a = (M^T M)^{-1} M^T\ b$
  - Still works if overdetermined
    - WHY???

$$\underset{\mathbf{M}}{\begin{bmatrix} m_{x1} & m_{y1} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x1} & m_{y1} & 1 \\ m_{x2} & m_{y2} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x2} & m_{y2} & 1 \\ m_{x3} & m_{y3} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x3} & m_{y3} & 1 \\ & & & \dots & & \\ & & & \dots & & \end{bmatrix}} \underset{\mathbf{a}}{\begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix}} = \underset{\mathbf{b}}{\begin{bmatrix} n_{x1} \\ n_{y1} \\ n_{x2} \\ n_{y2} \\ n_{x3} \\ n_{y3} \end{bmatrix}}$$

We want projective (homography)

- What are our equations now?
  - $n_x = (h_{00}*m_x + h_{01}*m_y + h_{02}*m_w) / (h_{20}*m_x + h_{21}*m_y + h_{22}*m_w)$
  - $n_y = (h_{10}*m_x + h_{11}*m_y + h_{12}*m_w) / (h_{20}*m_x + h_{21}*m_y + h_{22}*m_w)$

$$\begin{bmatrix} \tilde{x}' \\ \tilde{y}' \\ \tilde{w}' \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$$

We want projective (homography)

- What are our equations now?
  - $n_x = (h_{00}*m_x + h_{01}*m_y + h_{02}*m_w) / (h_{20}*m_x + h_{21}*m_y + h_{22}*m_w)$
  - $n_y = (h_{10}*m_x + h_{11}*m_y + h_{12}*m_w) / (h_{20}*m_x + h_{21}*m_y + h_{22}*m_w)$
- Assume h_{22} and m_w are 1, now 8 DOF
  - $n_x = (h_{00}*m_x + h_{01}*m_y + h_{02}) / (h_{20}*m_x + h_{21}*m_y + 1)$
  - $n_y = (h_{10}*m_x + h_{11}*m_y + h_{12}) / (h_{20}*m_x + h_{21}*m_y + 1)$

$$\begin{bmatrix} \tilde{x}' \\ \tilde{y}' \\ \tilde{w}' \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$$

We want projective (homography)

- What are our equations now?
  - $n_x = (h_{00}*m_x + h_{01}*m_y + h_{02}*m_w) / (h_{20}*m_x + h_{21}*m_y + h_{22}*m_w)$
  - $n_y = (h_{10}*m_x + h_{11}*m_y + h_{12}*m_w) / (h_{20}*m_x + h_{21}*m_y + h_{22}*m_w)$
- Assume h_{22} and m_w are 1, now 8 DOF
  - $n_x = (h_{00}*m_x + h_{01}*m_y + h_{02}) / (h_{20}*m_x + h_{21}*m_y + 1)$
  - $n_y = (h_{10}*m_x + h_{11}*m_y + h_{12}) / (h_{20}*m_x + h_{21}*m_y + 1)$
- More algebra on n_x
  - $n_x * (h_{20}*m_x + h_{21}*m_y + 1) = (h_{00}*m_x + h_{01}*m_y + h_{02})$
  - $n_x*h_{20}*m_x + n_x*h_{21}*m_y + n_x = h_{00}*m_x + h_{01}*m_y + h_{02}$
  - $n_x = h_{00}*m_x + h_{01}*m_y + h_{02} - n_x*h_{20}*m_x - n_x*h_{21}*m_y$
- Similar for n_y

We want projective (homography)

- What are our equations now?
  - $n_x = h_{00}*m_x + h_{01}*m_y + h_{02} - n_x*h_{20}*m_x - n_x*h_{21}*m_y$
  - $n_y = h_{10}*m_x + h_{11}*m_y + h_{12} - n_x*h_{20}*m_x - n_x*h_{21}*m_y$
- New matrix equations:

$$\underset{\mathbf{M}}{\begin{bmatrix} m_{x1} & m_{y1} & 1 & 0 & 0 & 0 & -m_{x1}n_{x1} & -m_{y1}n_{x1} \\ 0 & 0 & 0 & m_{x1} & m_{y1} & 1 & -m_{x1}n_{y1} & -m_{y1}n_{y1} \\ m_{x2} & m_{y2} & 1 & 0 & 0 & 0 & -m_{x2}n_{x2} & -m_{y2}n_{x2} \\ 0 & 0 & 0 & m_{x2} & m_{y2} & 1 & -m_{x2}n_{y2} & -m_{y2}n_{y2} \\ m_{x3} & m_{y3} & 1 & 0 & 0 & 0 & -m_{x3}n_{x3} & -m_{y3}n_{x3} \\ 0 & 0 & 0 & m_{x3} & m_{y3} & 1 & -m_{x3}n_{y3} & -m_{y3}n_{y3} \\ m_{x4} & m_{y4} & 1 & 0 & 0 & 0 & -m_{x4}n_{x4} & -m_{y4}n_{x4} \\ 0 & 0 & 0 & m_{x4} & m_{y4} & 1 & -m_{x4}n_{y4} & -m_{y4}n_{y4} \end{bmatrix}} \underset{\mathbf{a}}{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix}} = \underset{\mathbf{b}}{\begin{bmatrix} n_{x1} \\ n_{y1} \\ n_{x2} \\ n_{y2} \\ n_{x3} \\ n_{y3} \\ n_{x4} \\ n_{y4} \end{bmatrix}}$$

We want projective (homography)

- New matrix equations:

$$\underbrace{\begin{bmatrix} m_{x1} & m_{y1} & 1 & 0 & 0 & 0 & -m_{x1}n_{x1} & -m_{y1}n_{x1} \\ 0 & 0 & 0 & m_{x1} & m_{y1} & 1 & -m_{x1}n_{y1} & -m_{y1}n_{y1} \\ m_{x2} & m_{y2} & 1 & 0 & 0 & 0 & -m_{x2}n_{x2} & -m_{y2}n_{x2} \\ 0 & 0 & 0 & m_{x2} & m_{y2} & 1 & -m_{x2}n_{y2} & -m_{y2}n_{y2} \\ m_{x3} & m_{y3} & 1 & 0 & 0 & 0 & -m_{x3}n_{x3} & -m_{y3}n_{x3} \\ 0 & 0 & 0 & m_{x3} & m_{y3} & 1 & -m_{x3}n_{y3} & -m_{y3}n_{y3} \\ m_{x4} & m_{y4} & 1 & 0 & 0 & 0 & -m_{x4}n_{x4} & -m_{y4}n_{x4} \\ 0 & 0 & 0 & m_{x4} & m_{y4} & 1 & -m_{x4}n_{y4} & -m_{y4}n_{y4} \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix}}_{\mathbf{a}} = \underbrace{\begin{bmatrix} n_{x1} \\ n_{y1} \\ n_{x2} \\ n_{y2} \\ n_{x3} \\ n_{y3} \\ n_{x4} \\ n_{y4} \end{bmatrix}}_{\mathbf{b}}$$

- Same procedure, Solve **M a = b**
  - Exact if #rows of M = 8
  - Least squares if #rows of M > 8

Are there any problems with this??

- New matrix equations:

$$\underset{\mathbf{M}}{\begin{bmatrix} m_{x1} & m_{y1} & 1 & 0 & 0 & 0 & -m_{x1}n_{x1} & -m_{y1}n_{x1} \\ 0 & 0 & 0 & m_{x1} & m_{y1} & 1 & -m_{x1}n_{y1} & -m_{y1}n_{y1} \\ m_{x2} & m_{y2} & 1 & 0 & 0 & 0 & -m_{x2}n_{x2} & -m_{y2}n_{x2} \\ 0 & 0 & 0 & m_{x2} & m_{y2} & 1 & -m_{x2}n_{y2} & -m_{y2}n_{y2} \\ m_{x3} & m_{y3} & 1 & 0 & 0 & 0 & -m_{x3}n_{x3} & -m_{y3}n_{x3} \\ 0 & 0 & 0 & m_{x3} & m_{y3} & 1 & -m_{x3}n_{y3} & -m_{y3}n_{y3} \\ m_{x4} & m_{y4} & 1 & 0 & 0 & 0 & -m_{x4}n_{x4} & -m_{y4}n_{x4} \\ 0 & 0 & 0 & m_{x4} & m_{y4} & 1 & -m_{x4}n_{y4} & -m_{y4}n_{y4} \end{bmatrix}} \underset{\mathbf{a}}{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix}} = \underset{\mathbf{b}}{\begin{bmatrix} n_{x1} \\ n_{y1} \\ n_{x2} \\ n_{y2} \\ n_{x3} \\ n_{y3} \\ n_{x4} \\ n_{y4} \end{bmatrix}}$$

- Same procedure, Solve **M a = b**
  - Exact if #rows of M = 8
  - Least squares if #rows of M > 8

Panorama algorithm:

Find corners in both images

Calculate descriptors

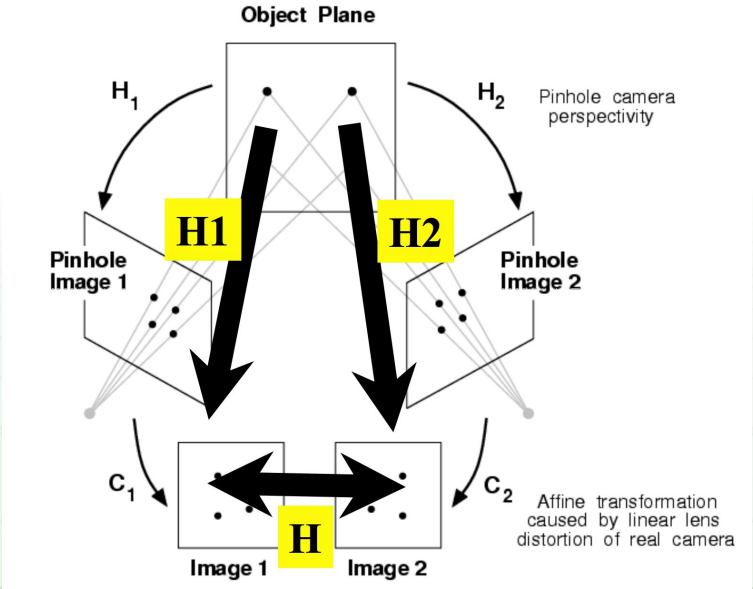
Match descriptors

RANSAC to find homography

Stitch together images with homography

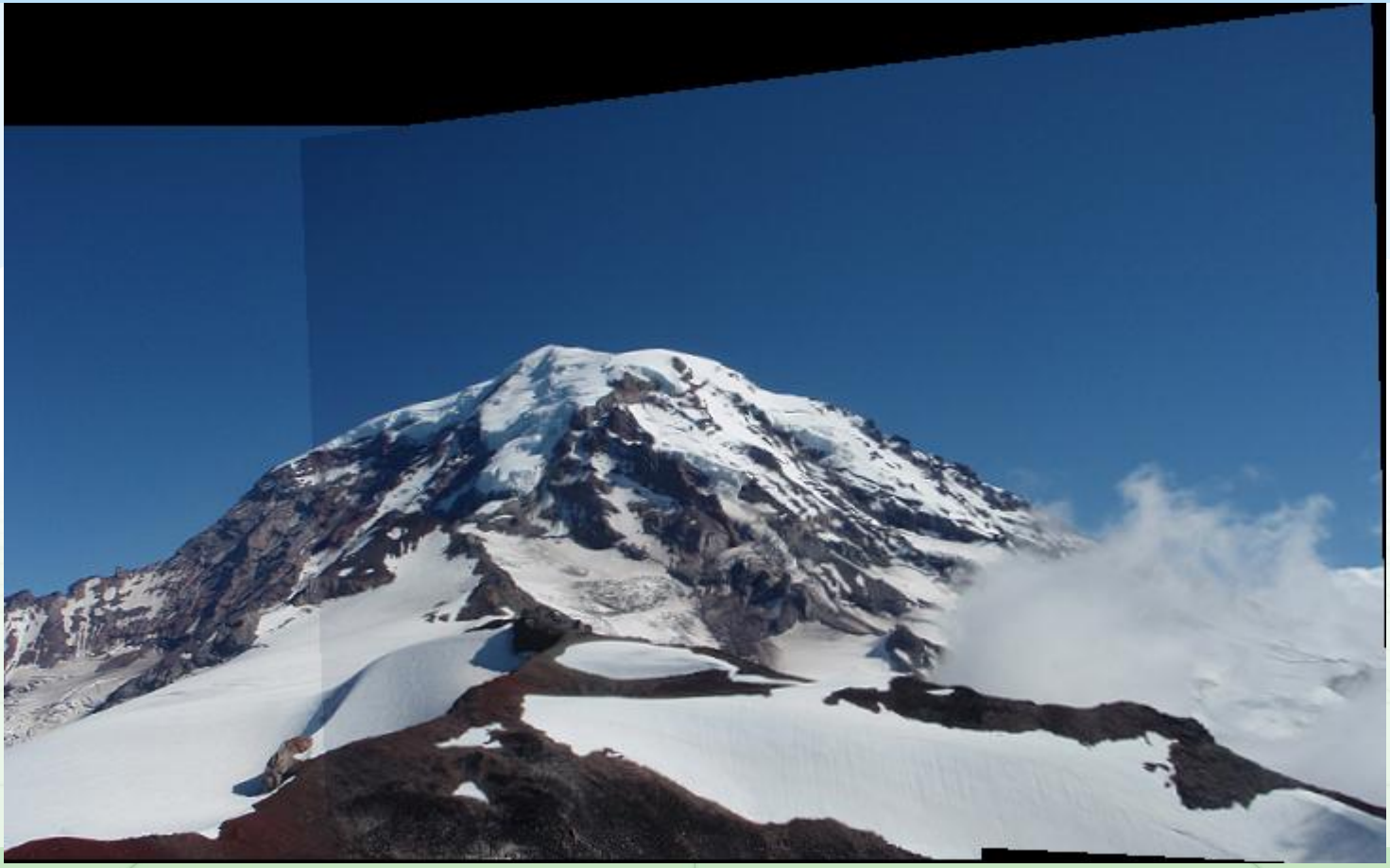
Stitching panoramas:

- We know homography is right choice under certain assumption:
  - Assume we are taking multiple images of planar object

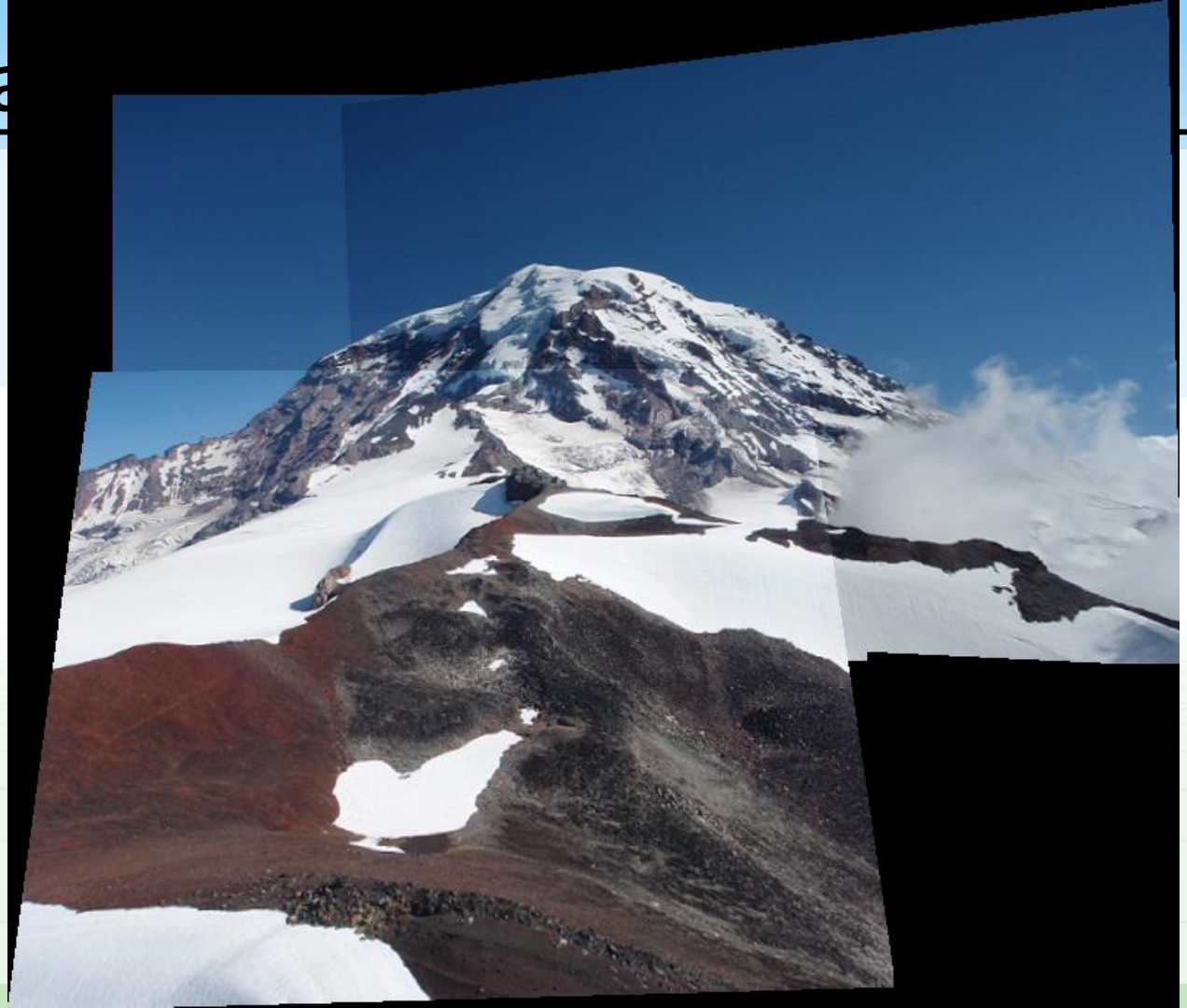


In practice:

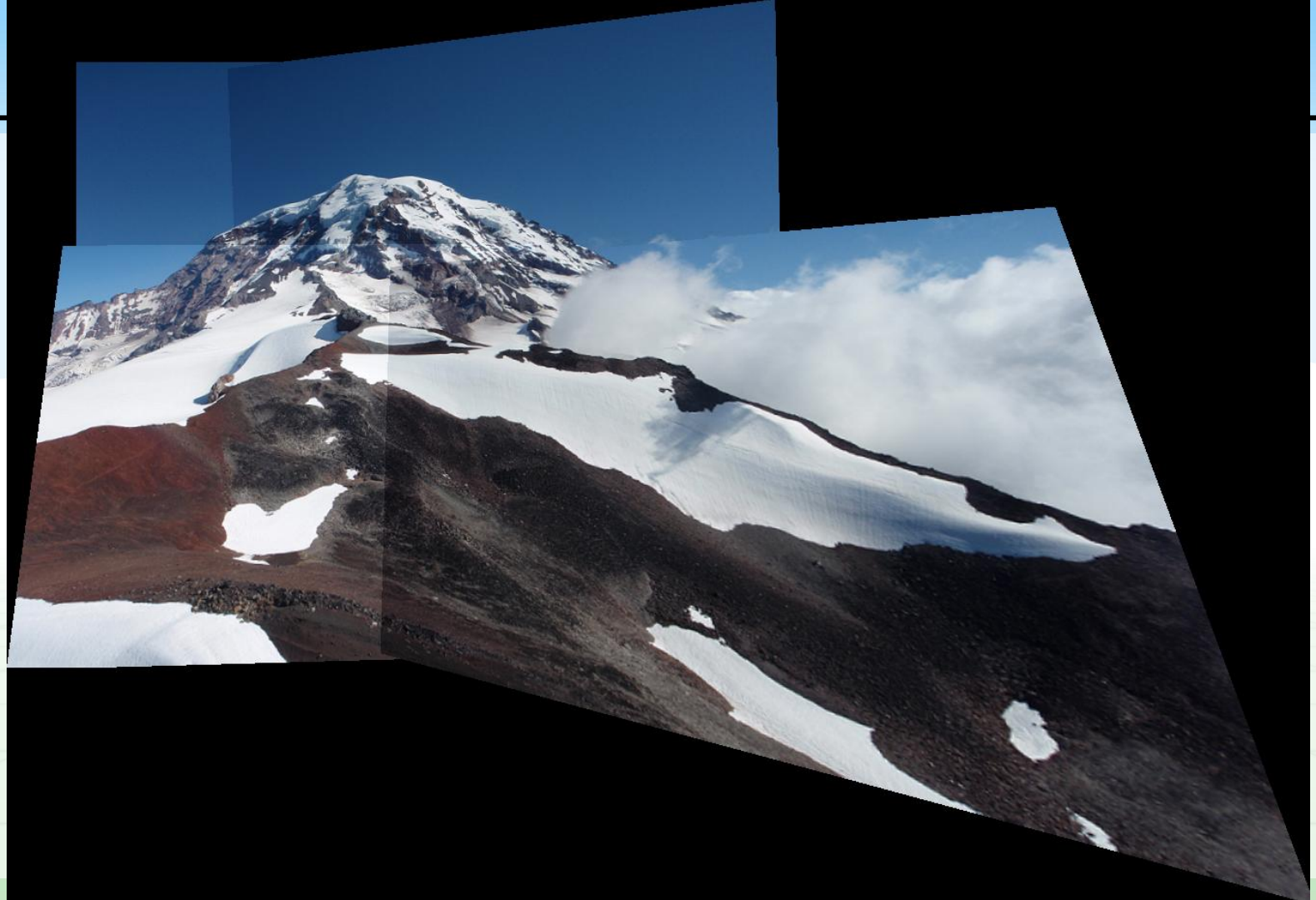
In practice:

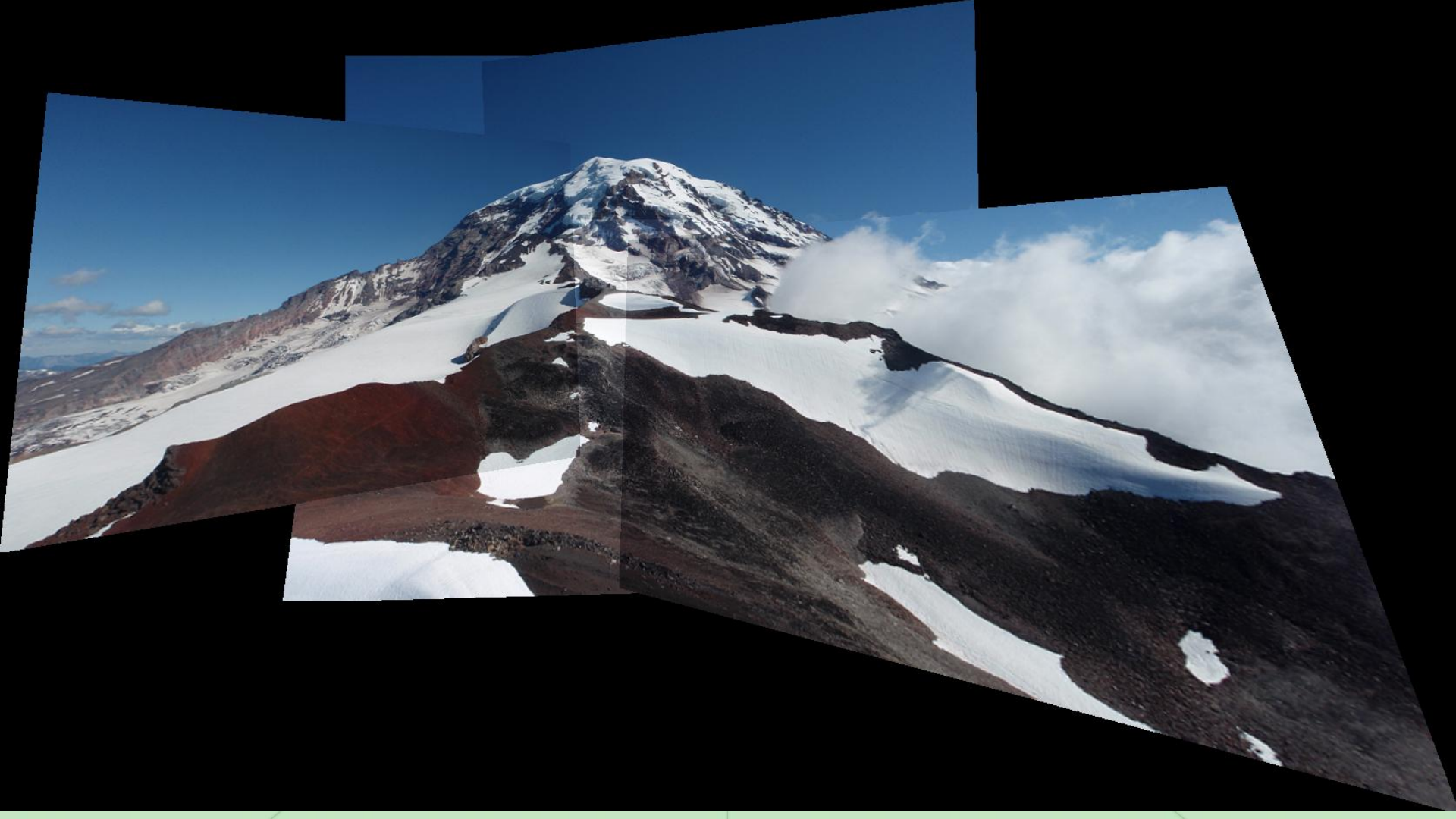


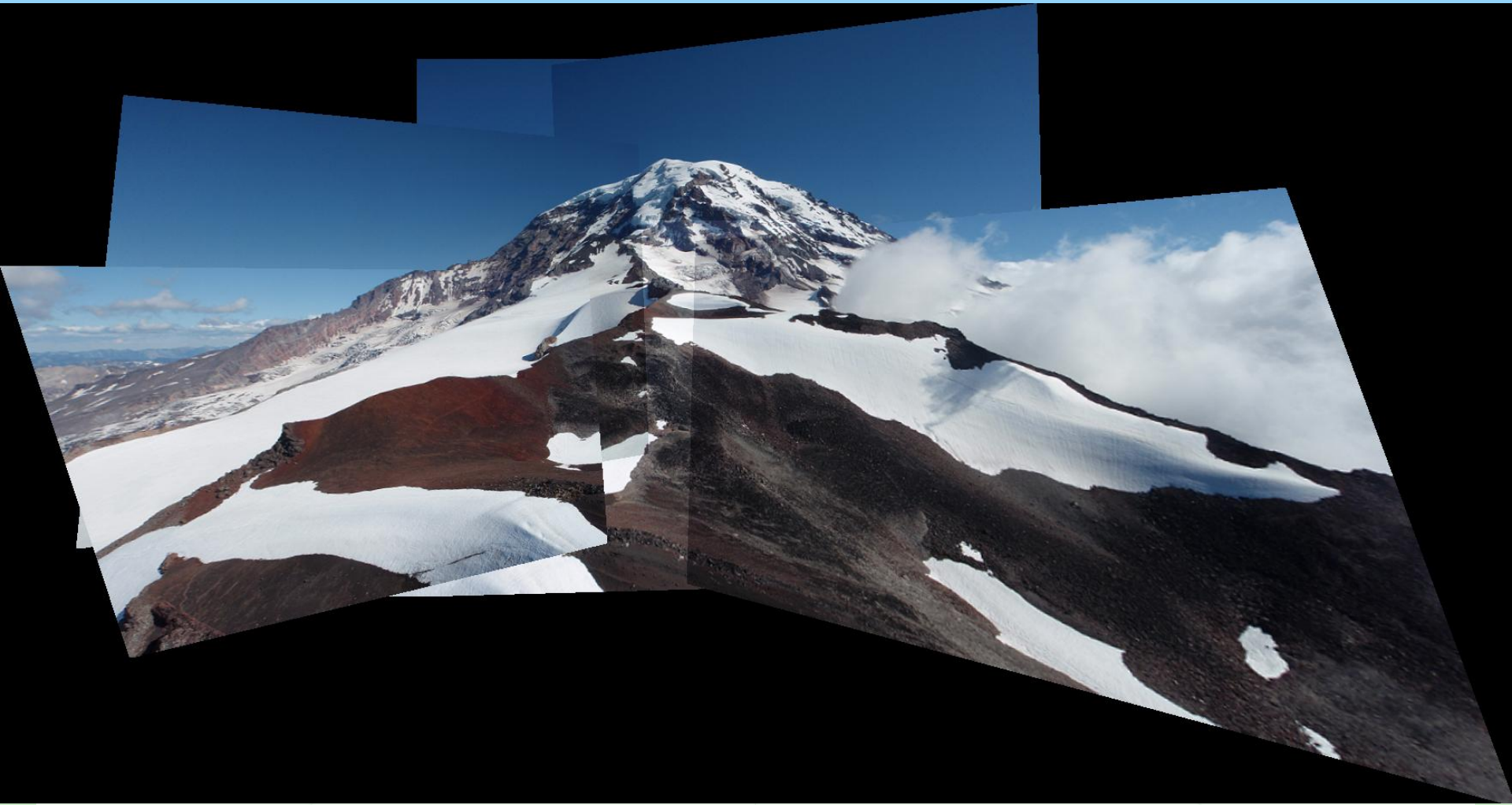
In pra



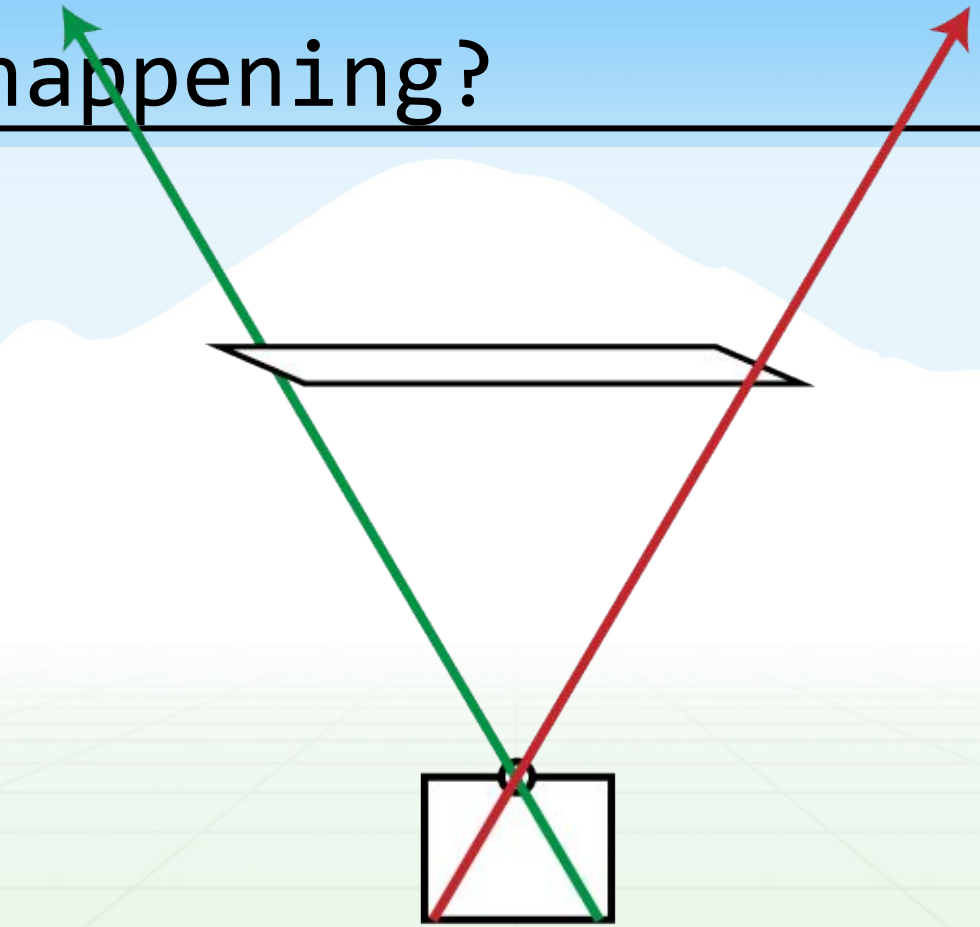
In



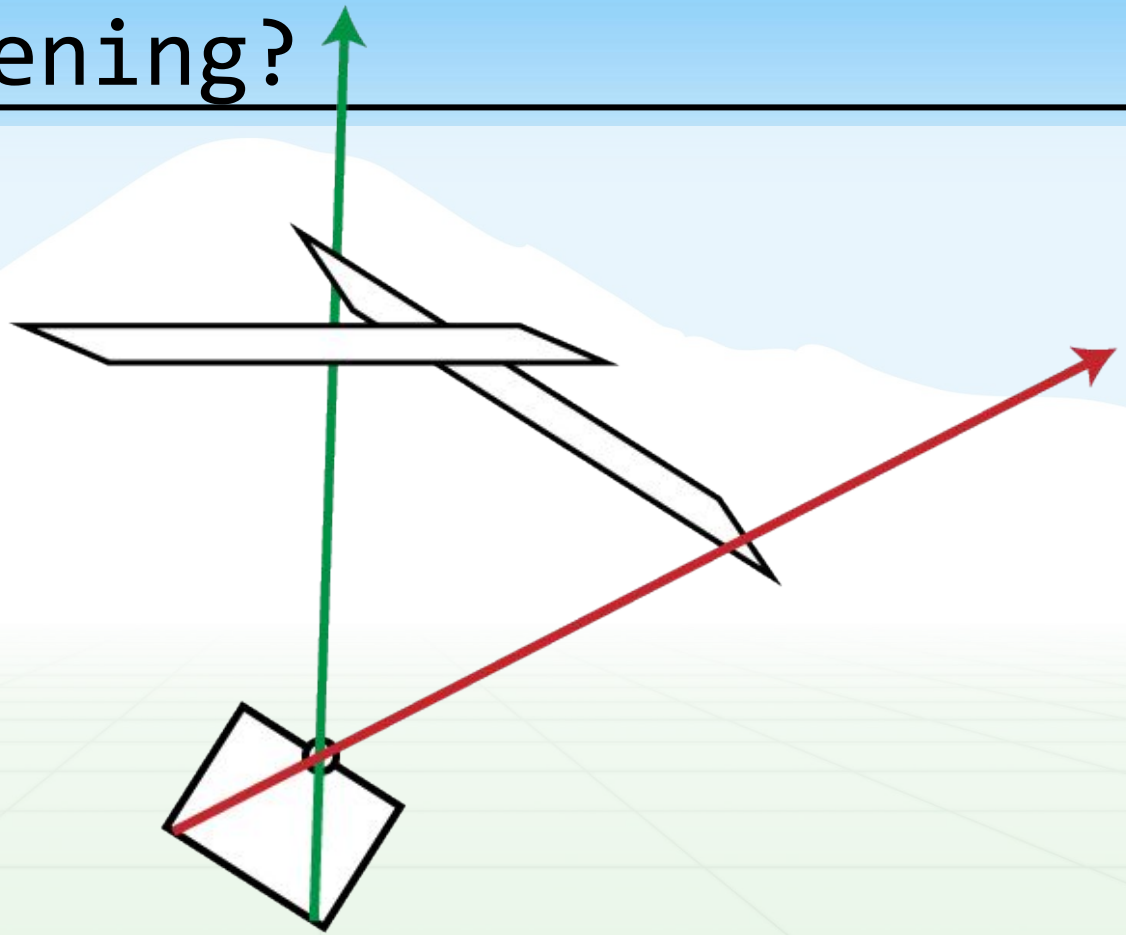




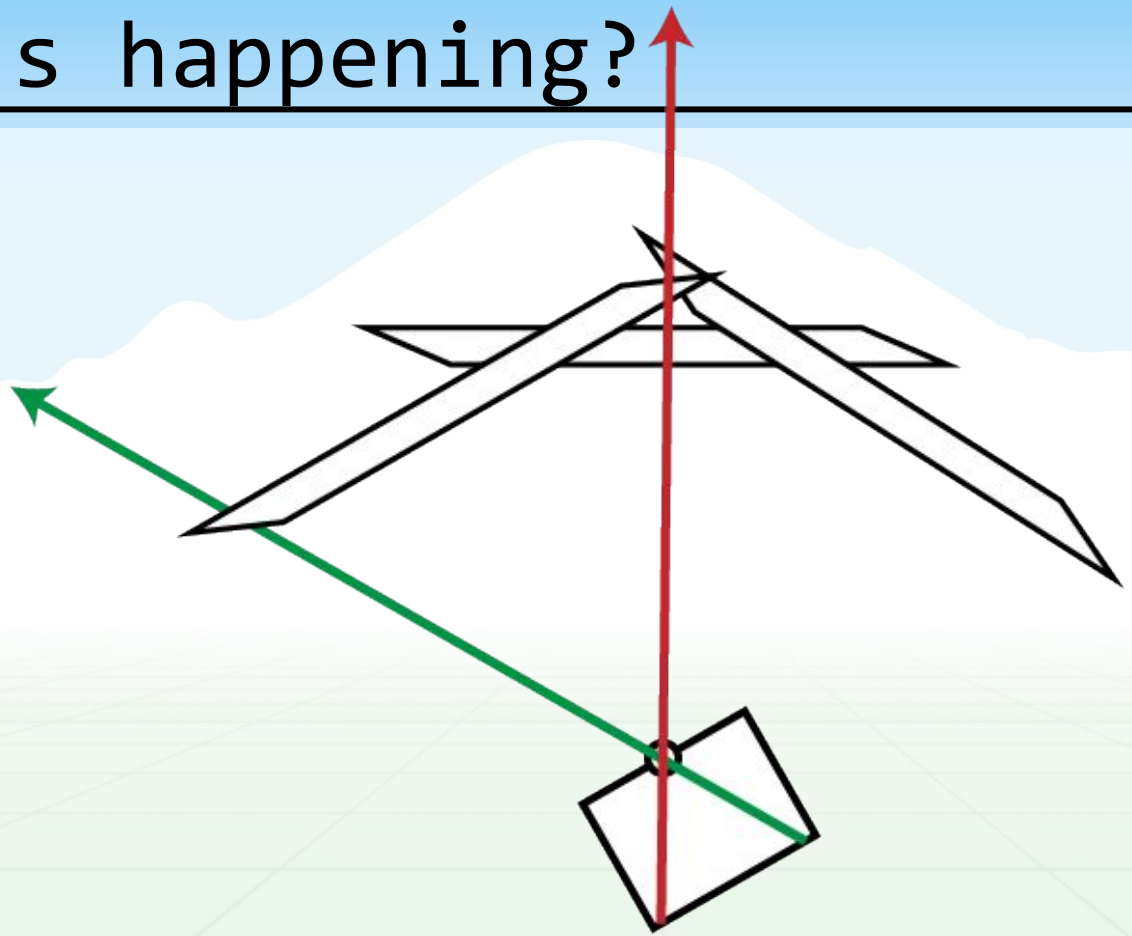
What’s happening?



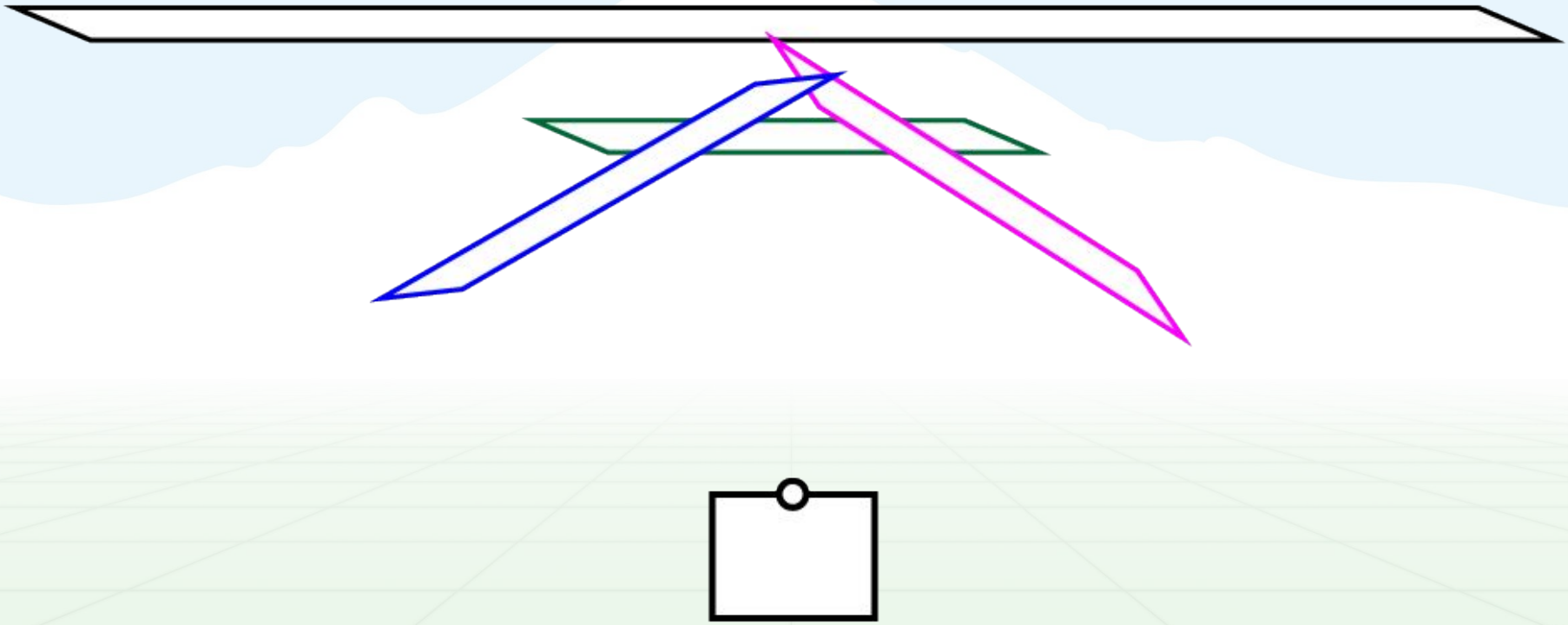
What’s happening?



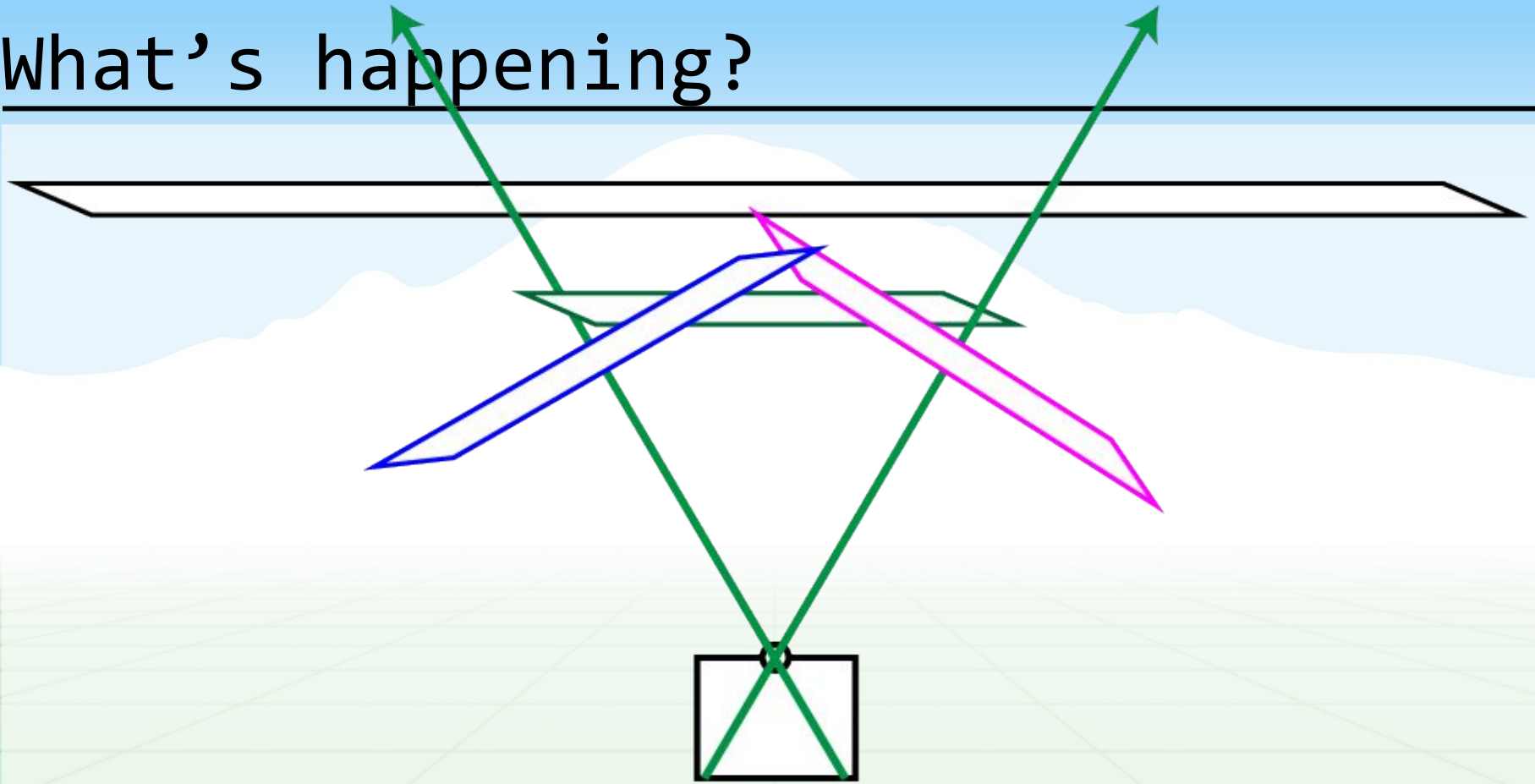
What’s happening?



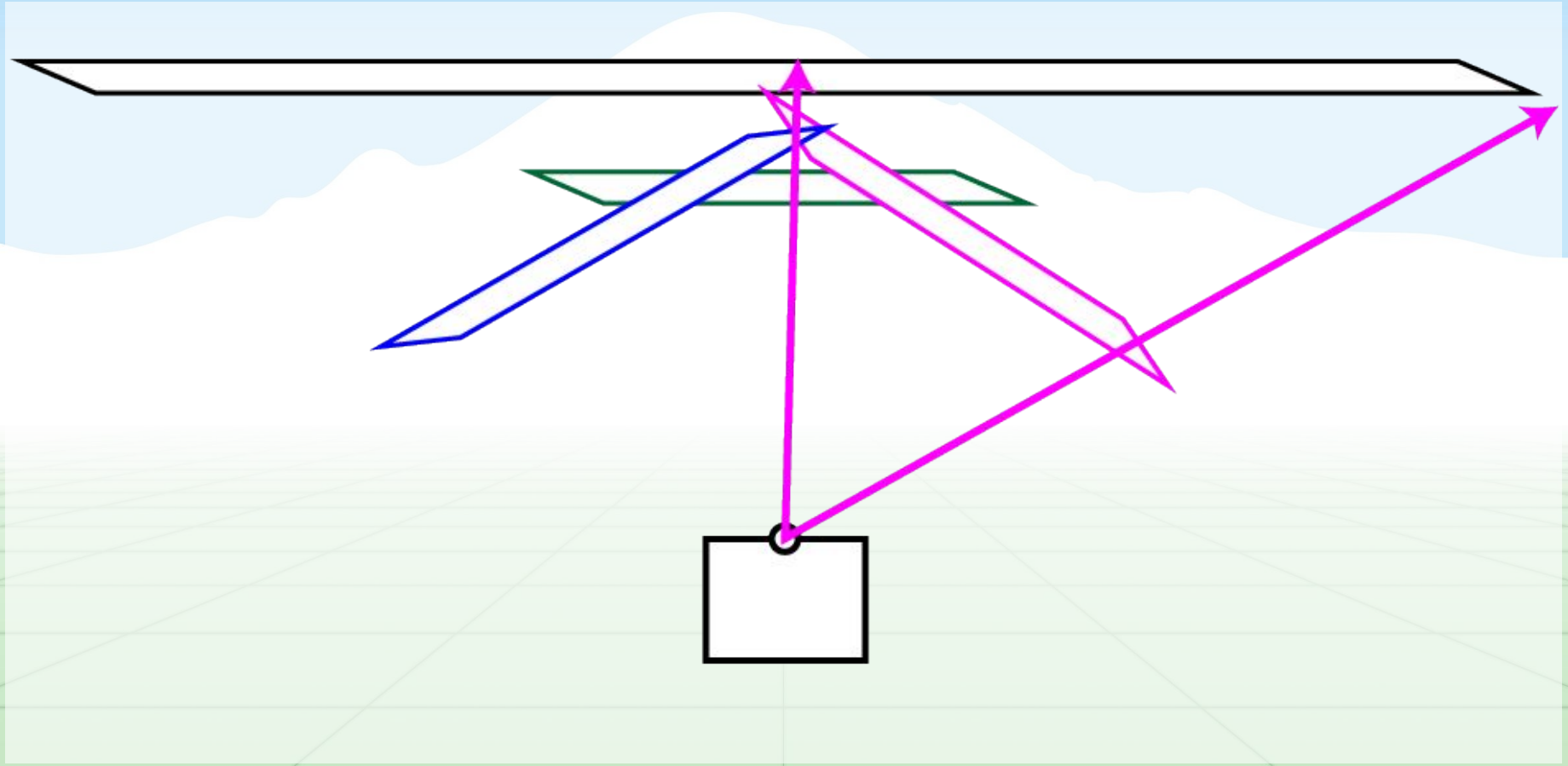
What’s happening?



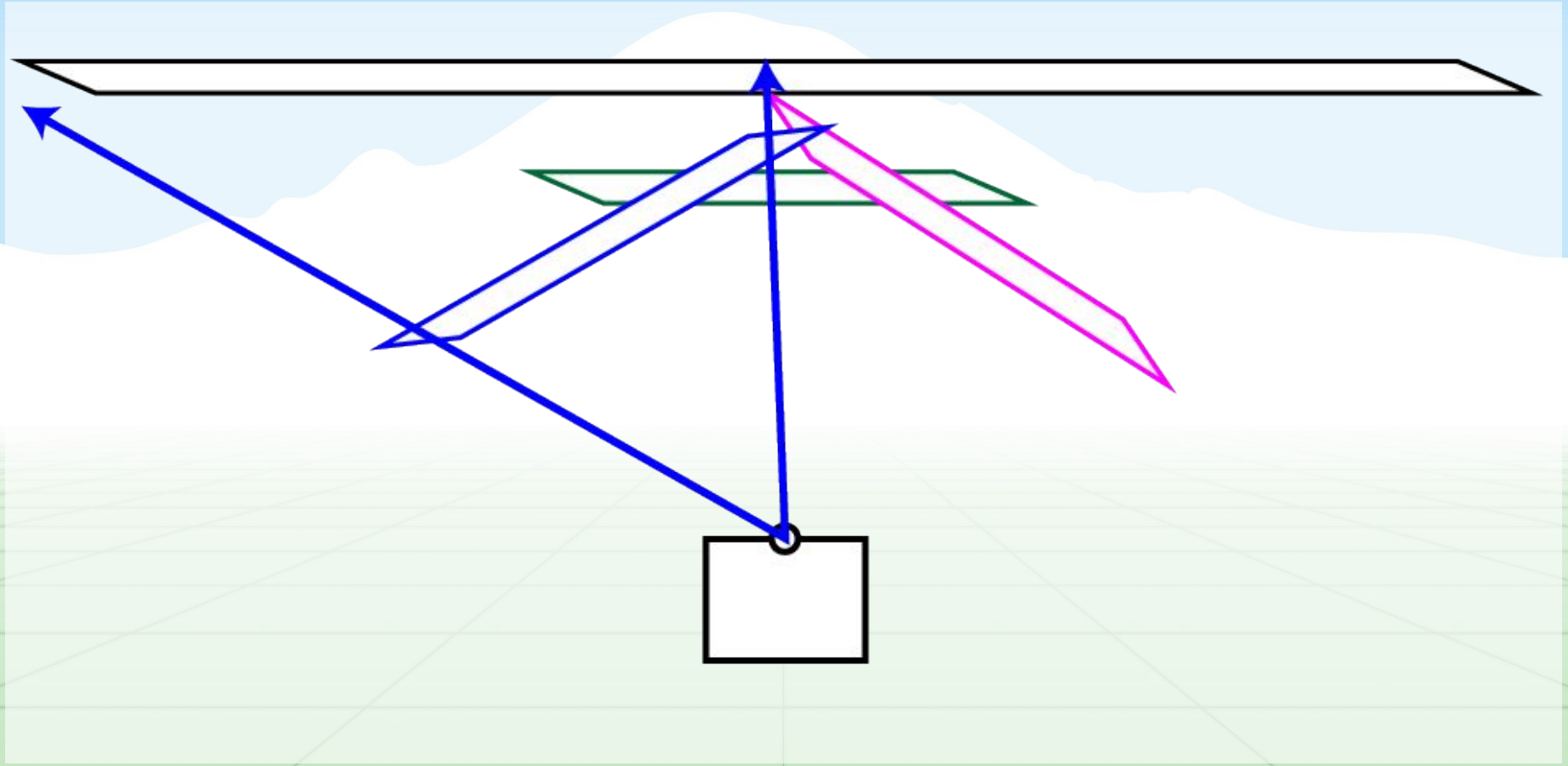
What’s happening?



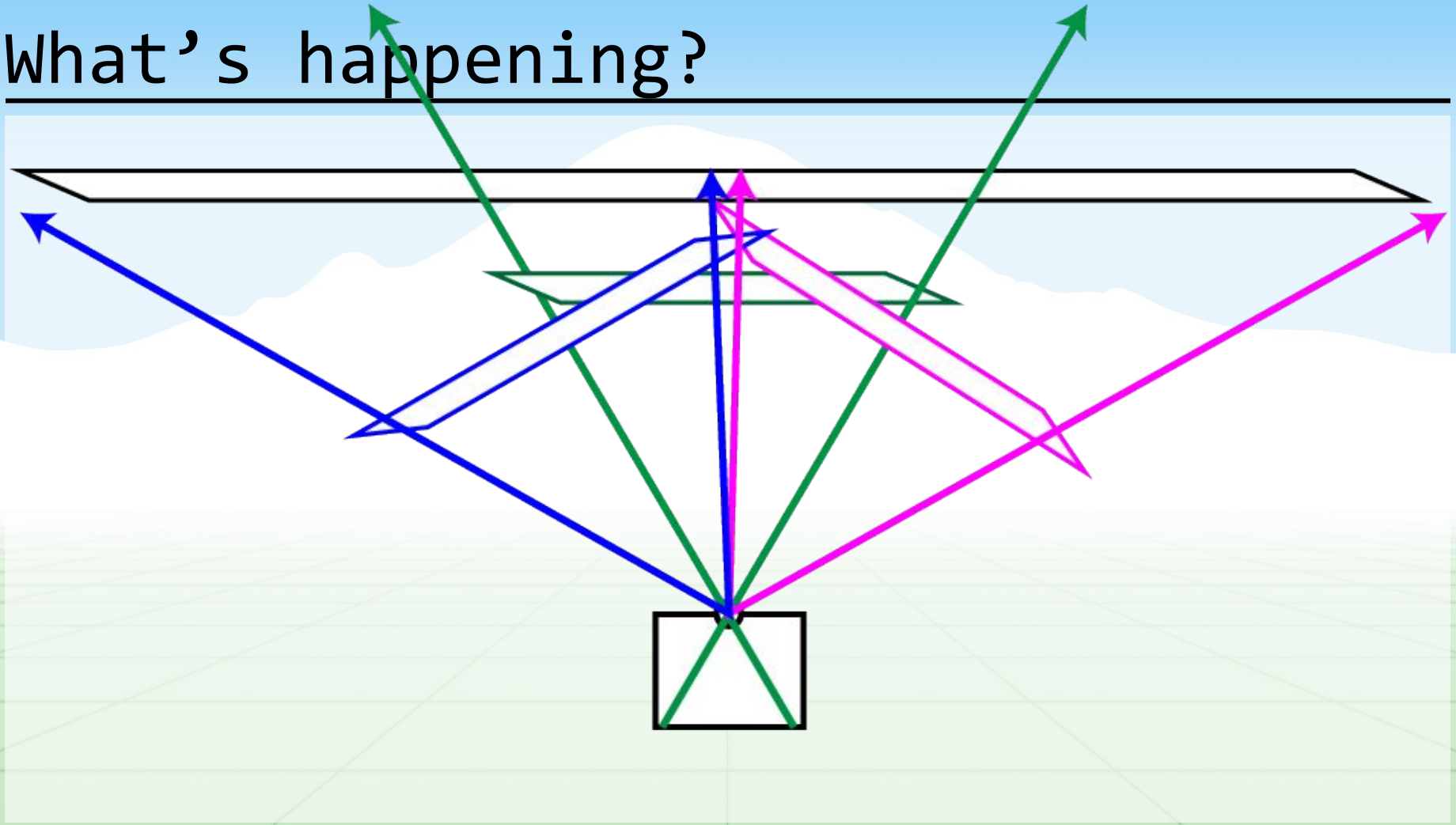
What’s happening?



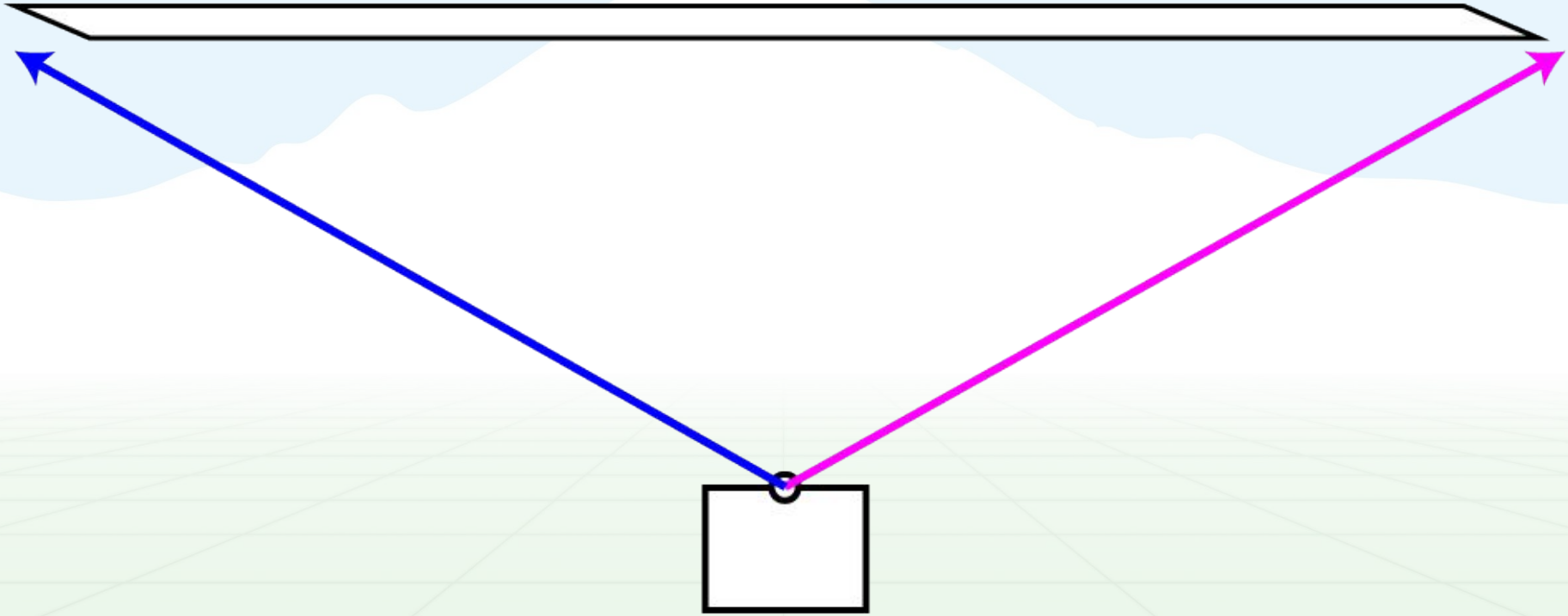
What’s happening?



What’s happening?

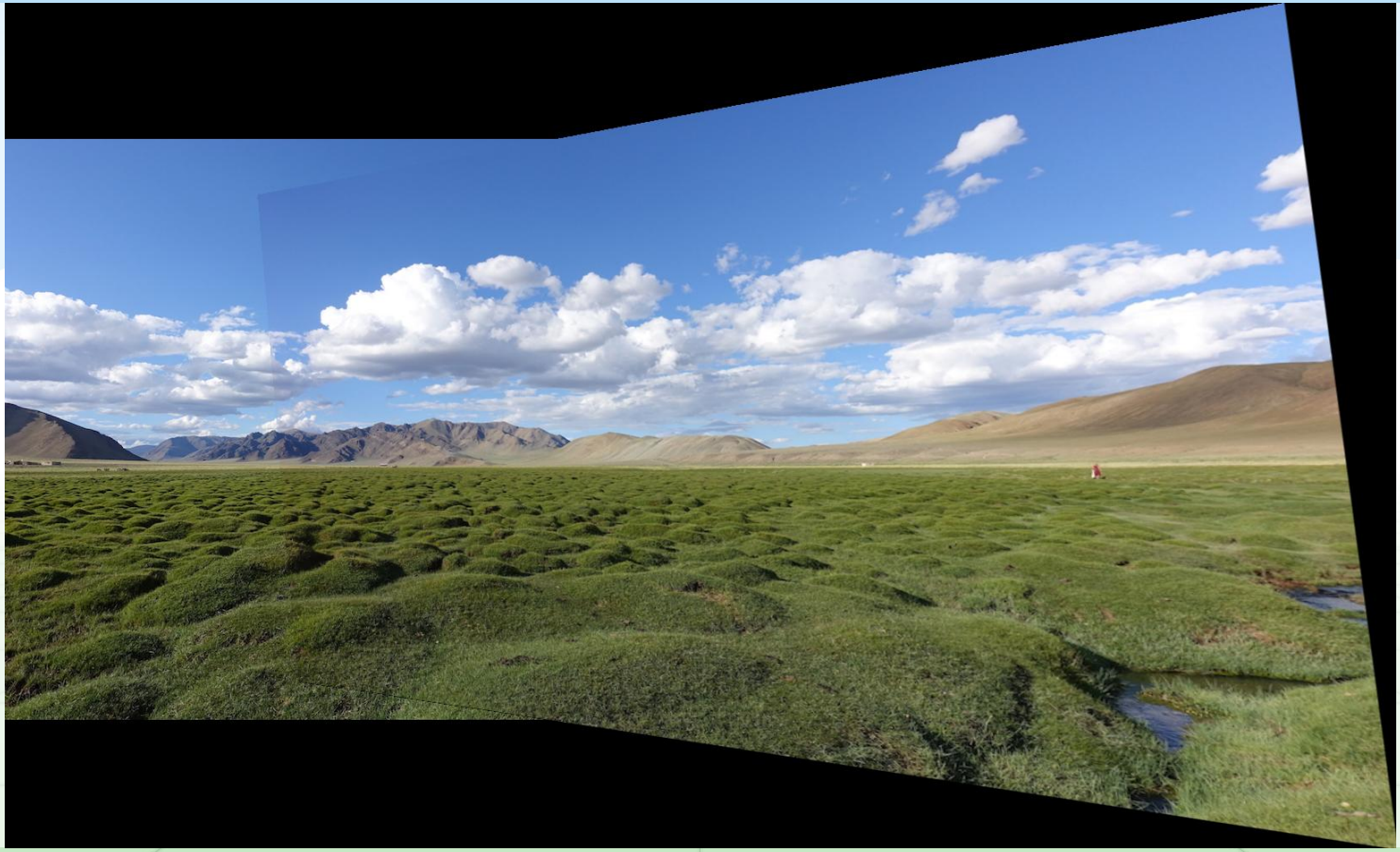


What’s happening?

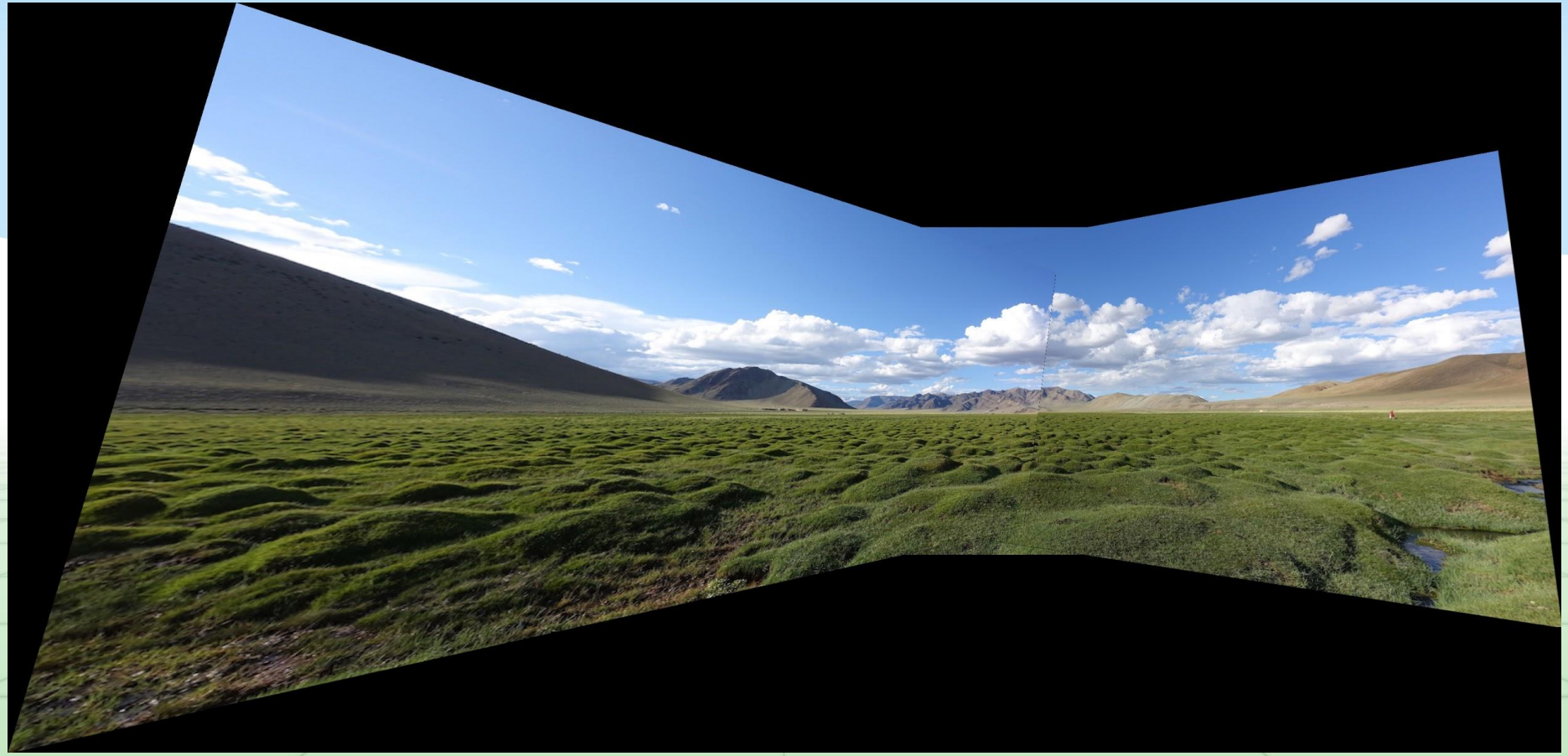


Very bad for big panoramas!

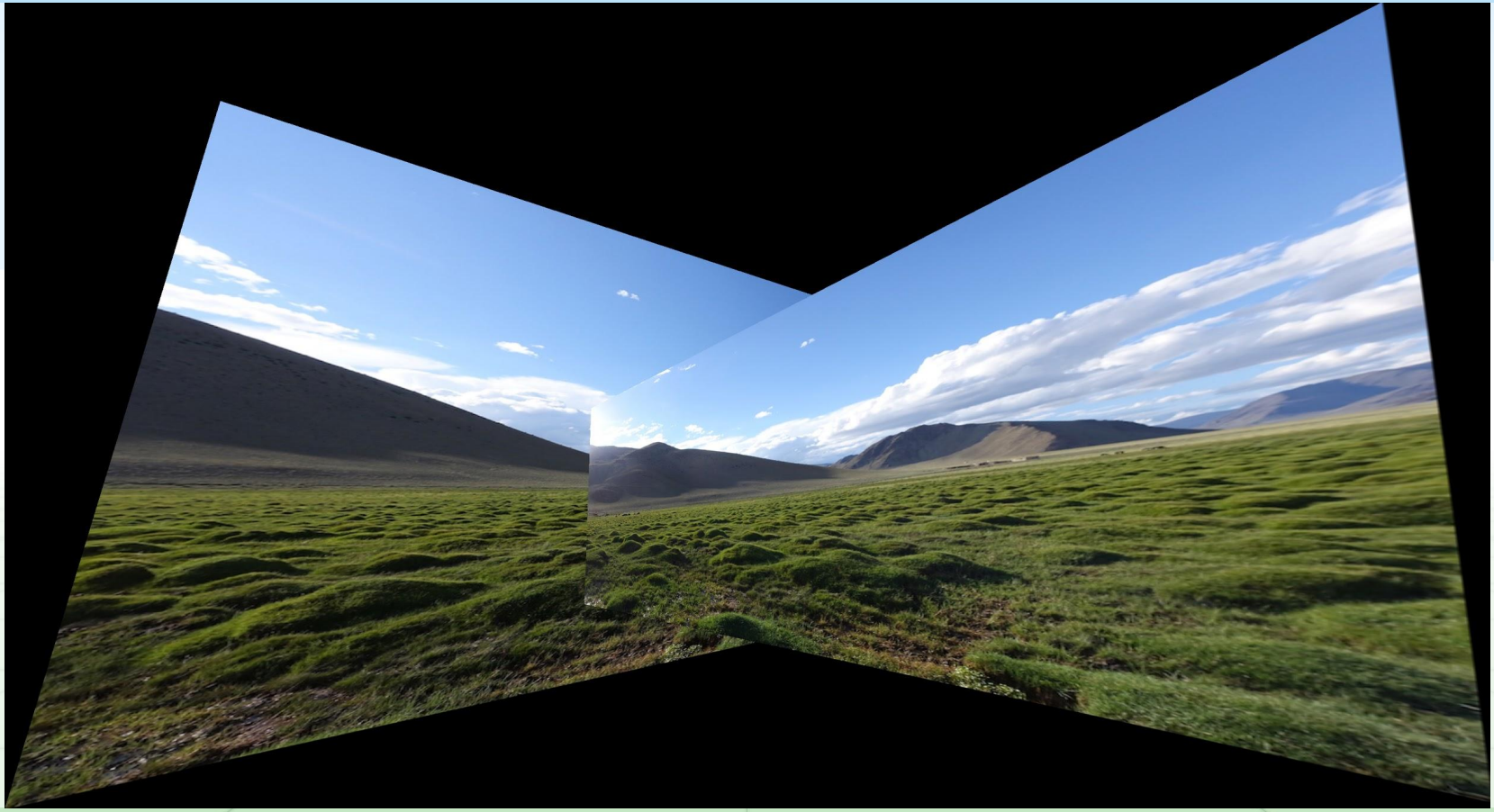
Very bad for big panoramas!



Very bad for big panoramas!

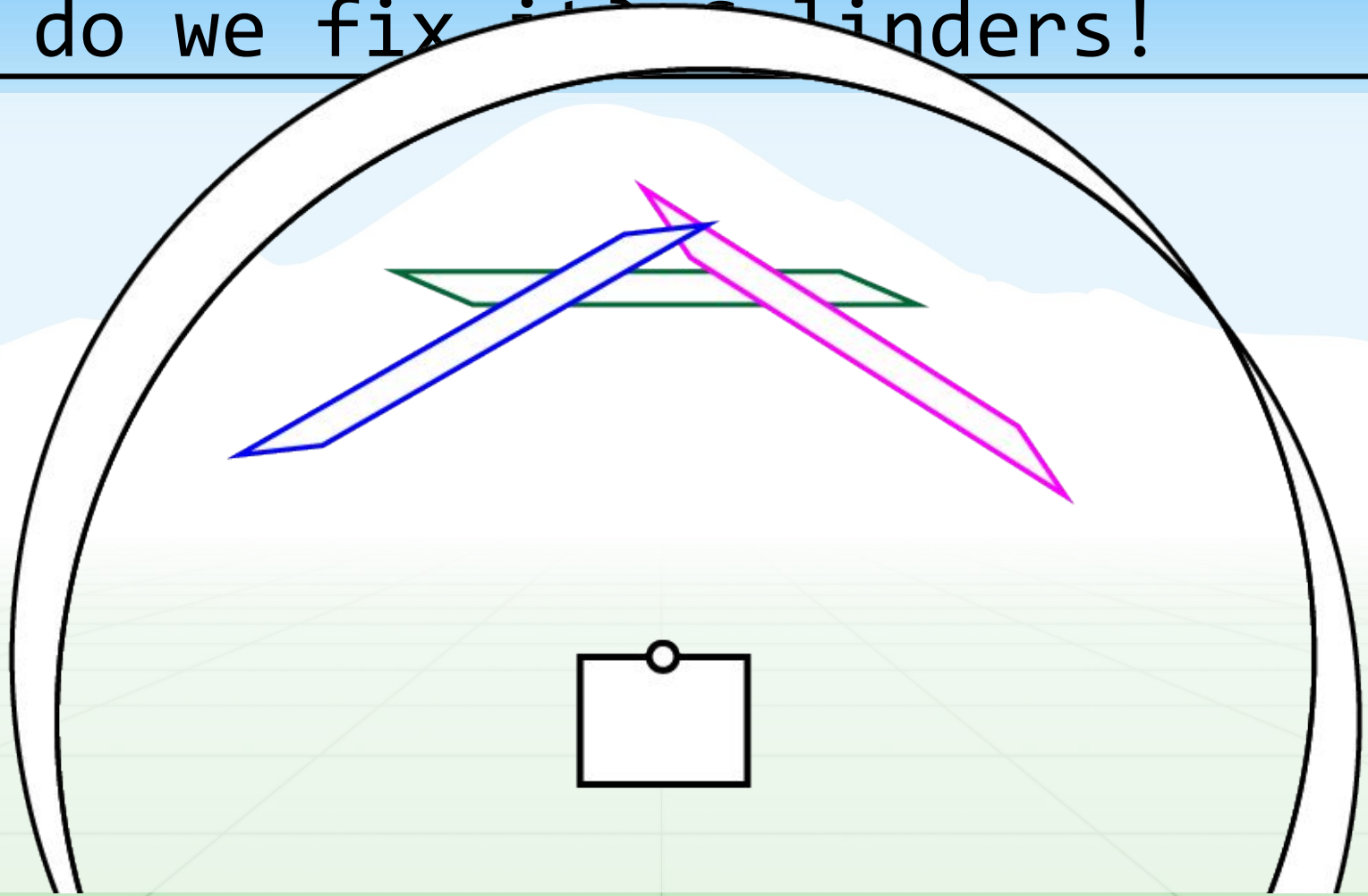


Fails :-(

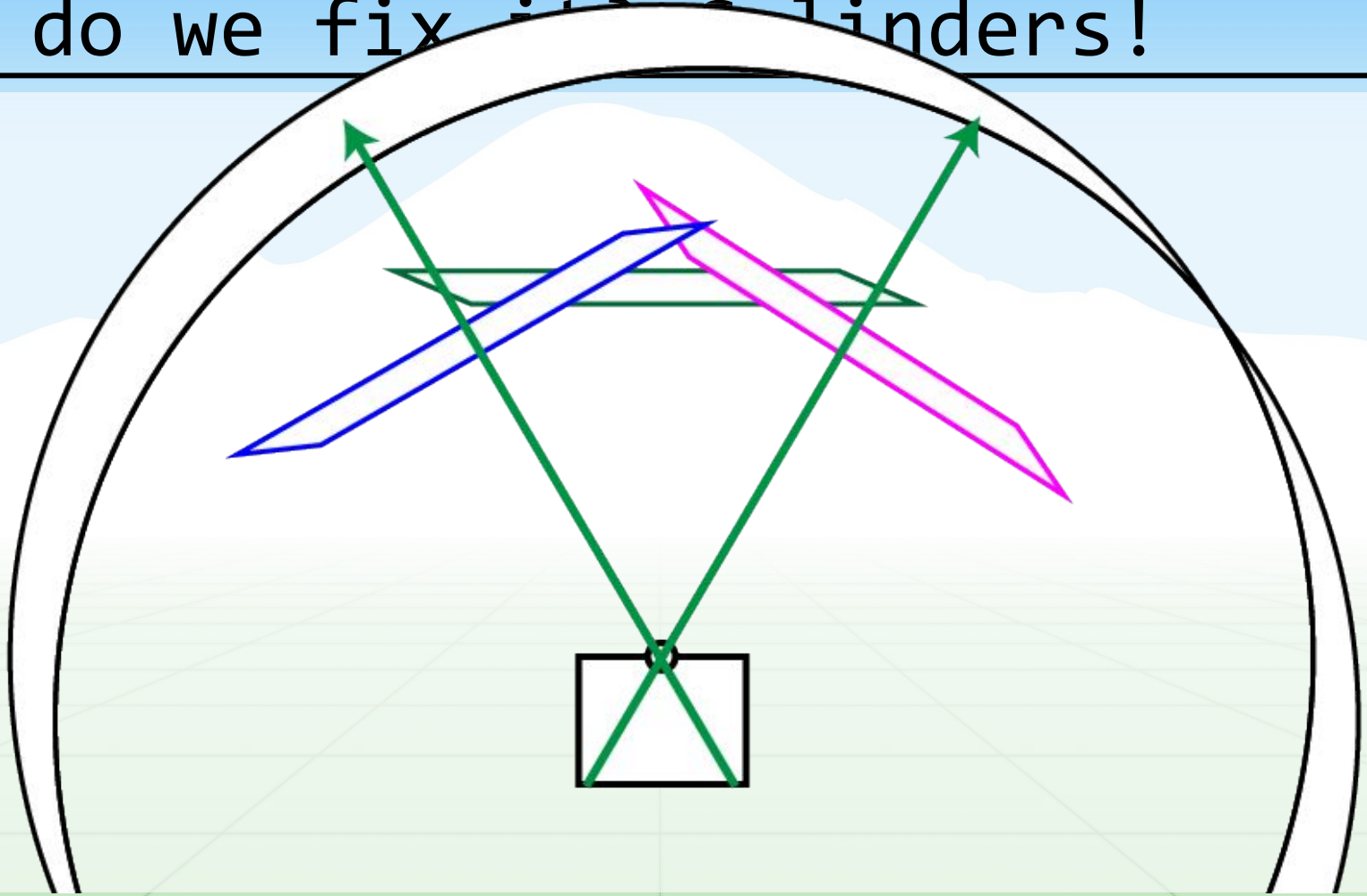


How do we fix it? Cylinders!

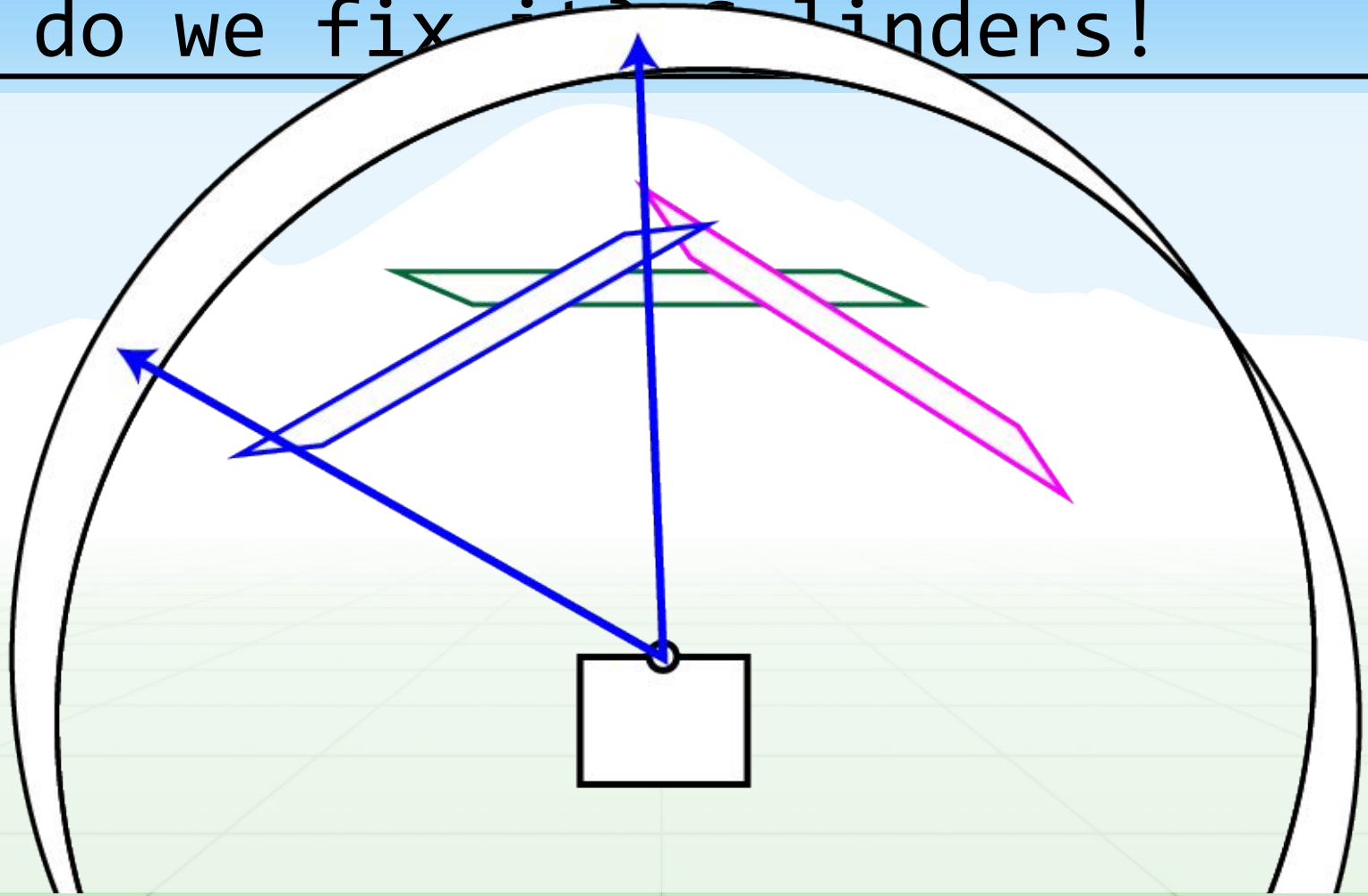
How do we fix it? Cylinders!



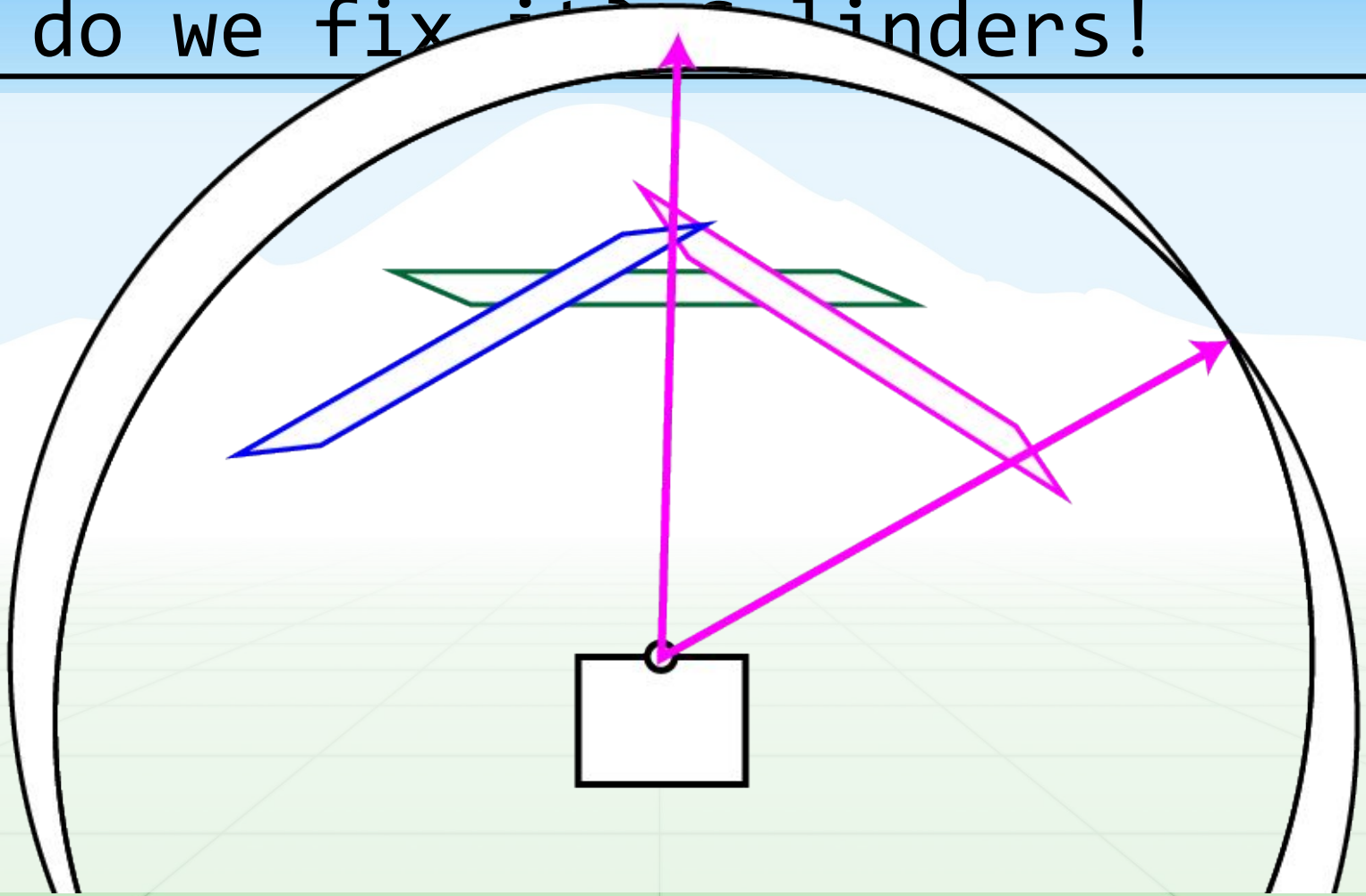
How do we fix it? Cylinders!



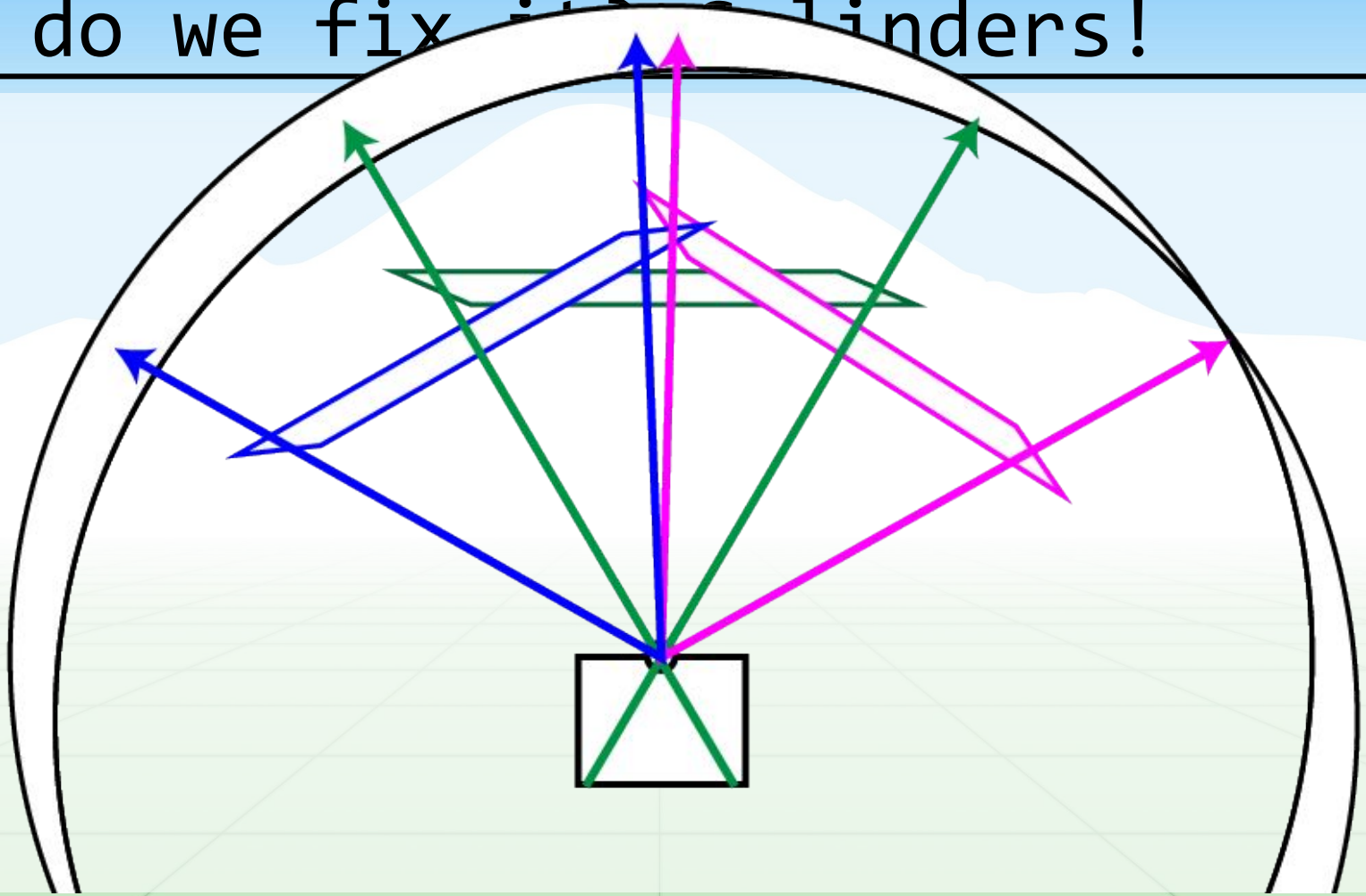
How do we fix it? Cylinders!



How do we fix it? Cylinders!



How do we fix it? Cylinders!



How do we fix it? Cylinders!

Calculate angle and height:

$\theta = (x - xc) / f$

$h = (y - yc) / f$

Find unit cylindrical coords:

$X' = \sin(\theta)$

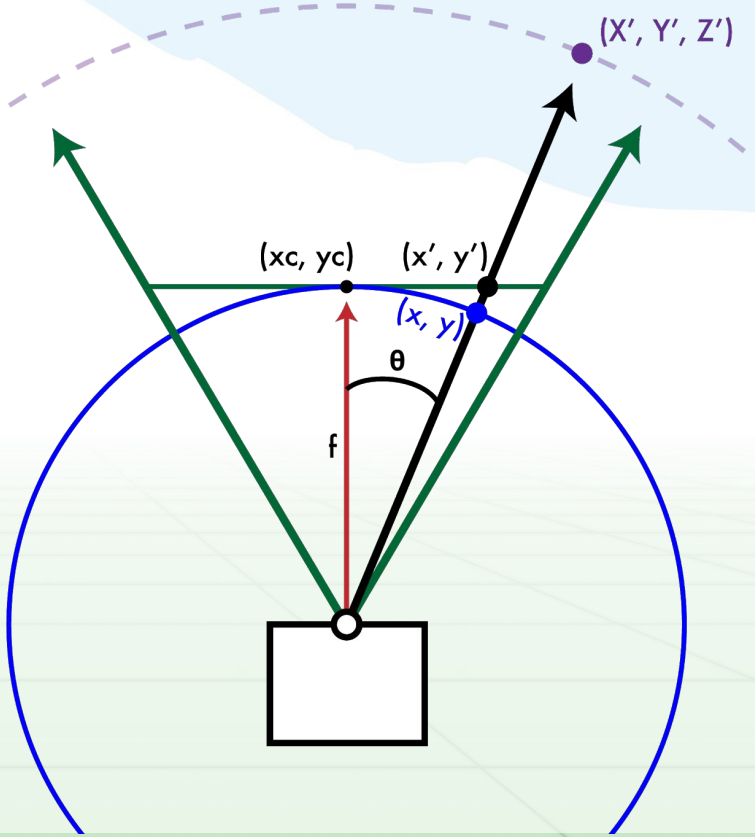
$Y' = h$

$Z' = \cos(\theta)$

Project to image plane:

$x' = f\, X'/Z' + xc$

$y' = f\, Y'/Z' + yc$



Dependant on focal length!

f = 300

f = 500

f = 1000

f = 1400

f = 10,000

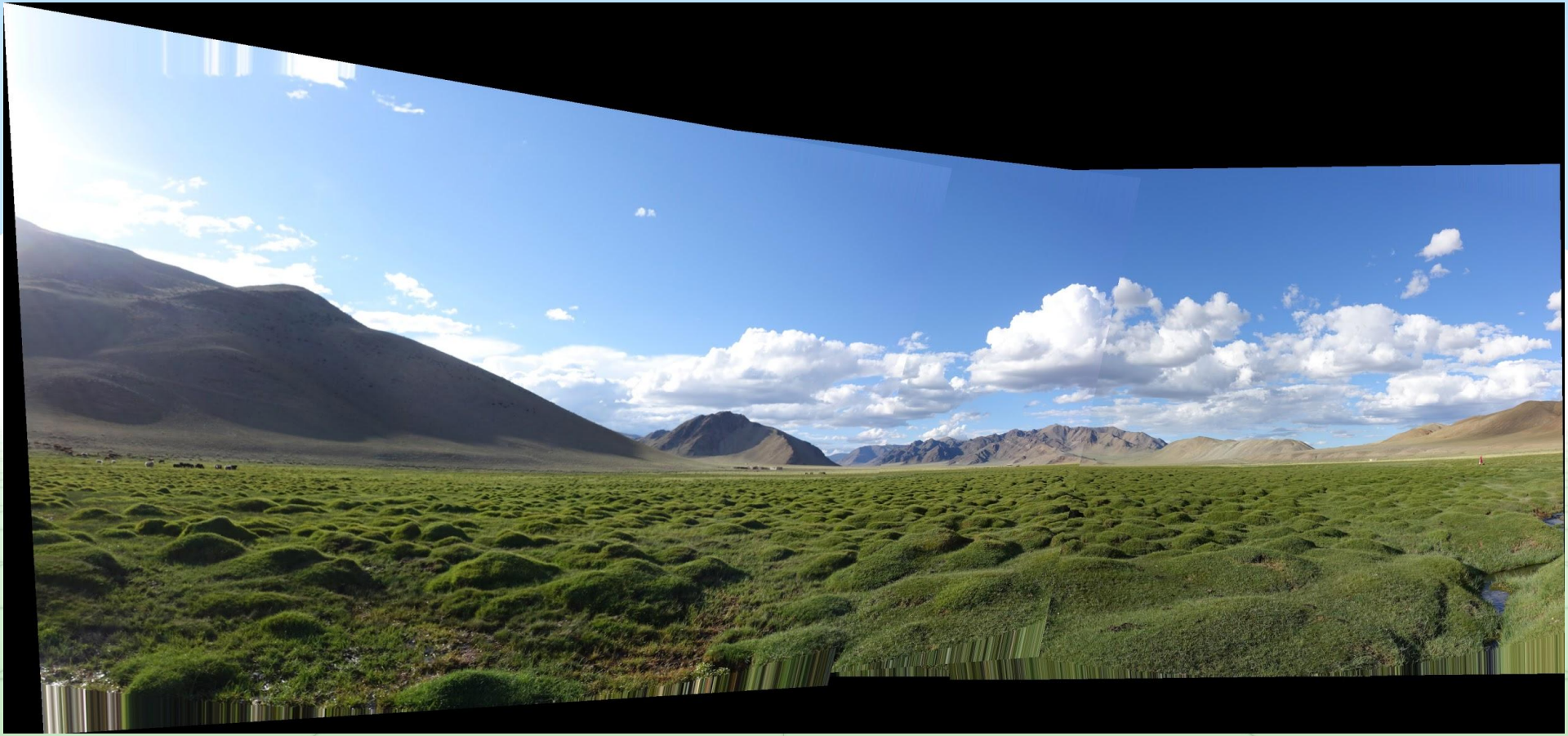
f = 10,000

Does it work?

Does it work?

Does it work?

Does it work?

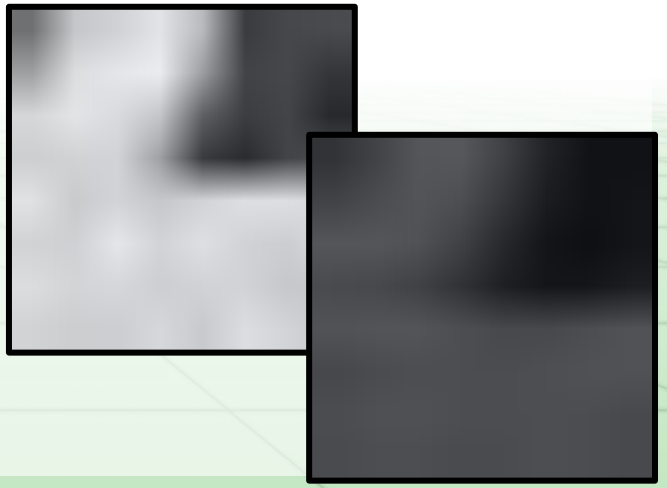


Does it work?

Does it work? Yay!

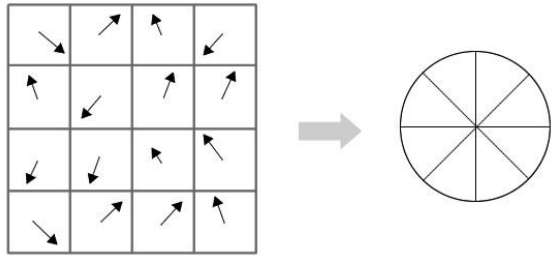
THIS IS AS GOOD AS IT GETS!

- Not so fast…
- Our descriptor has some issues:
  - Not invariant to lighting changes!
- Want features invariant to lighting



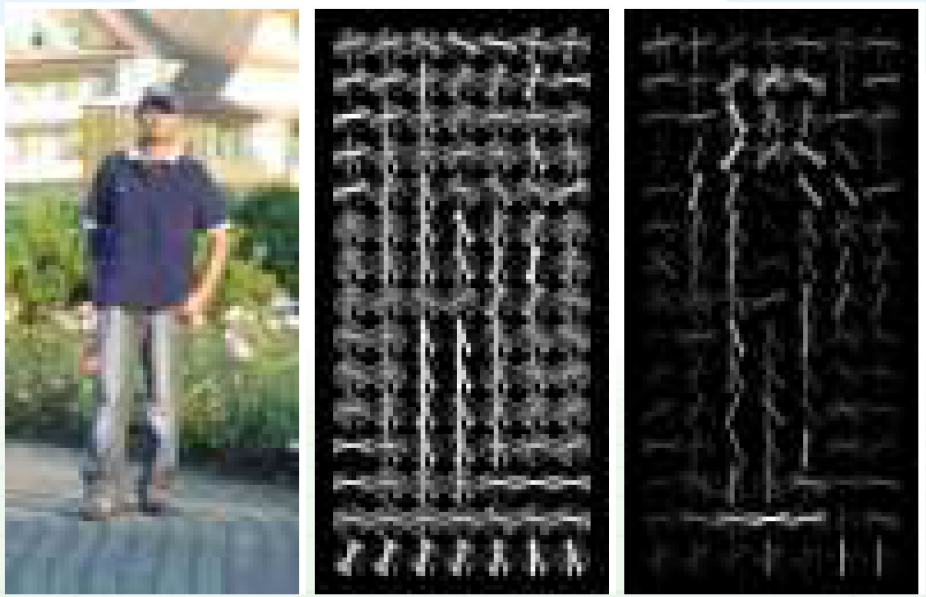
Histogram of Oriented Gradients (HOG)

- Dalal and Triggs 2005
- Better image descriptor
  - Compute gradients
  - Bin gradients
  - Aggregate blocks (4x4, 16x16 cells)
  - Normalize gradient magnitudes
- Not reliant on magnitude, just direction
  - Invariant to some lighting changes
- Train SVM to recognize people



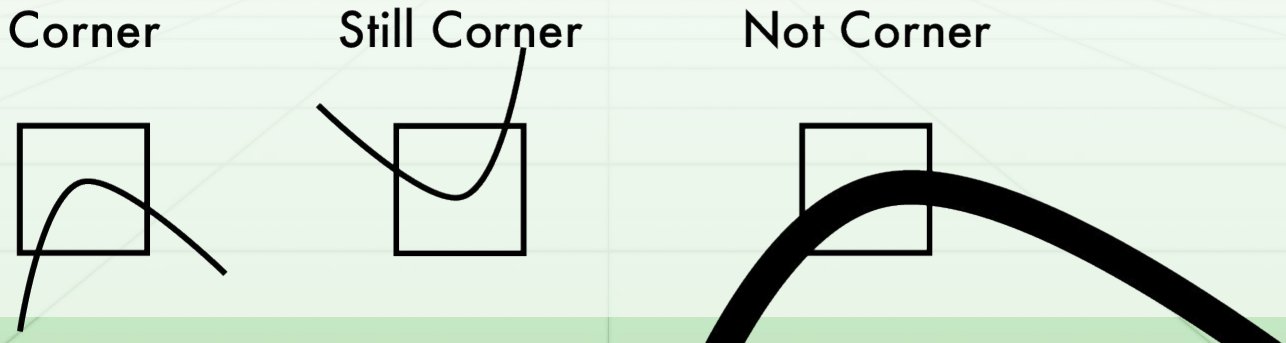
Binning from http://aishack.in/tutorials/sift-scale-invariant-feature-transform-features/

Histogram of Oriented Gradients (HOG)



THIS IS AS GOOD AS IT GETS!

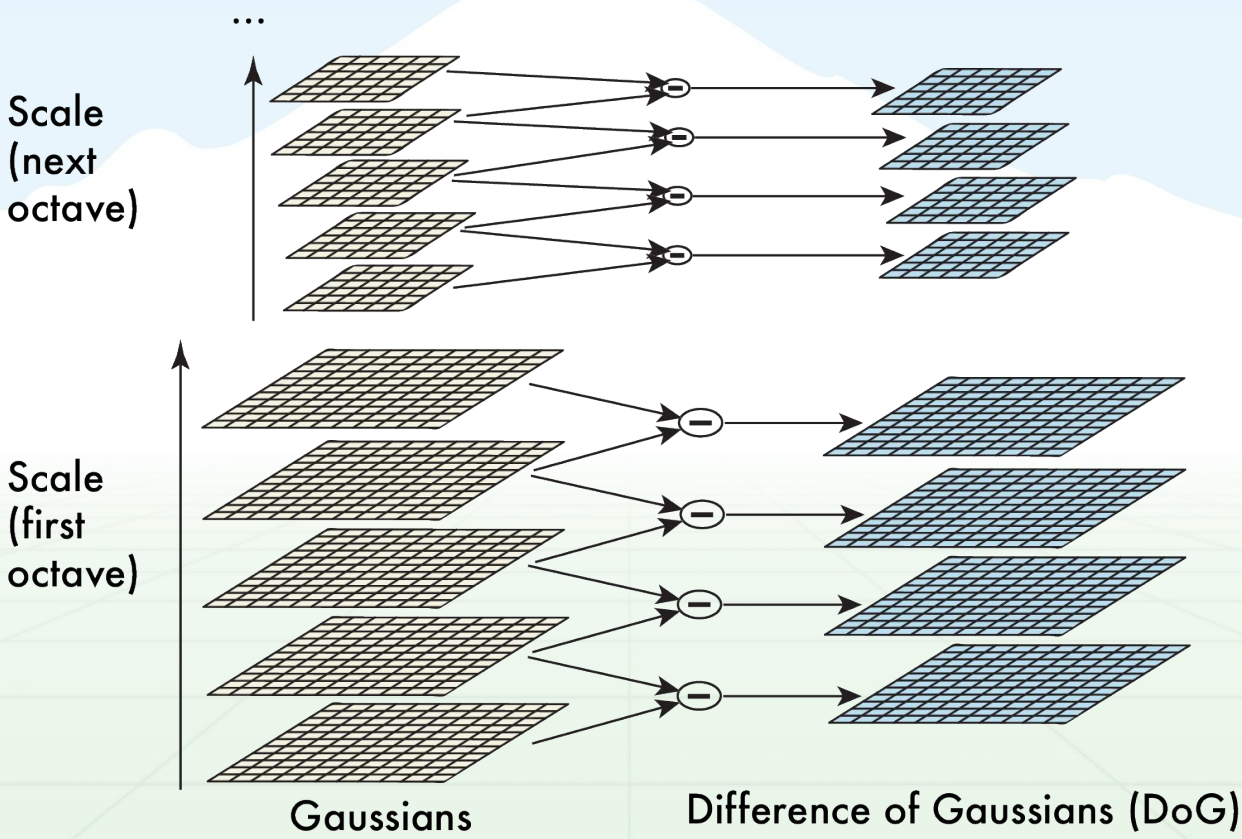
- Not so fast…
- Harris and has some issues:
  - Corners detection is rotation invariant
  - Harris not invariant to scale
- Descriptors are also hard
  - Just looking at pixels is not rotation invariant!
  - HOG also not rotation invariant



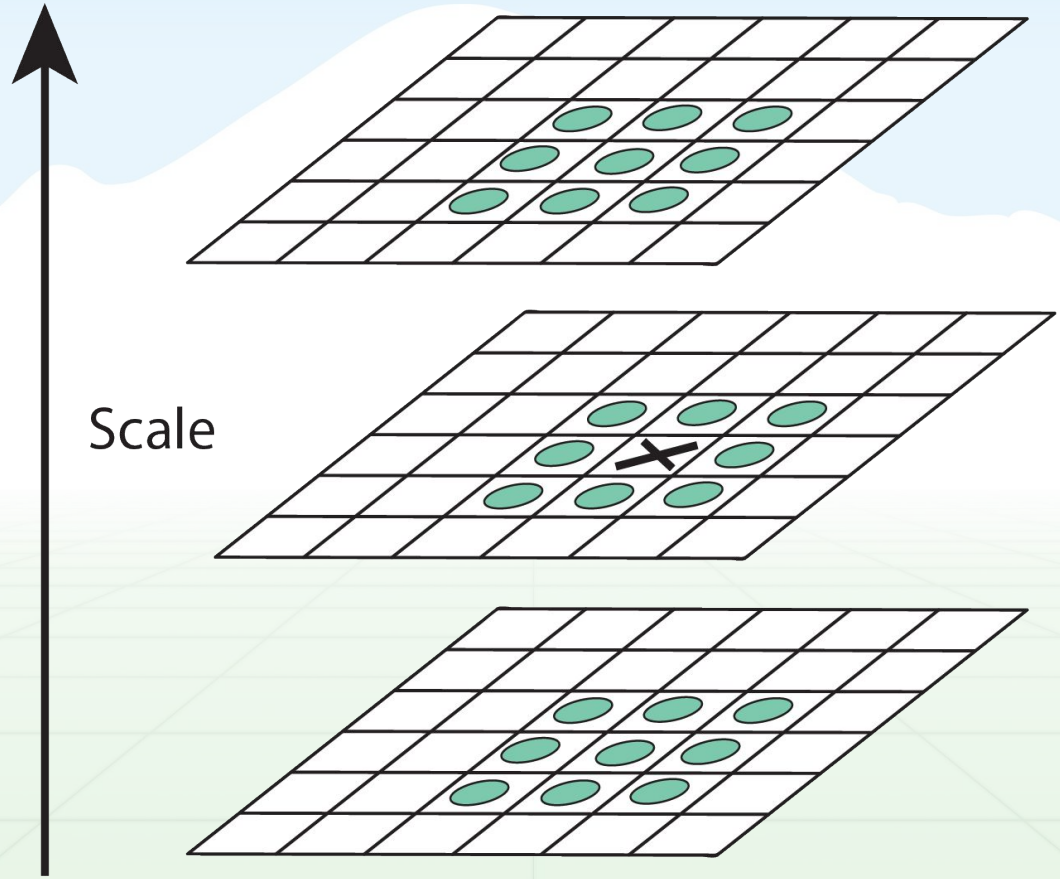
Want features invariant to scaling, rotation, etc.

- Scale Invariant Feature Transform (SIFT)
  - Lowe et al. 2004, many images from that paper
- Get scale-invariant response map
- Find keypoints
- Extract rotation-invariant descriptors
  - Normalize based on orientation
  - Normalize based on lighting

Extract DoG features at multiple scales



Find local-maxima in location and scale



Throw out weak responses and edges

- Estimate gradients
  - Similar to before, look at nearby responses
  - Not whole image, only a few points! Faster!
  - Throw out weak responses
- Find cornery things
  - Same deal, structure matrix, use det and trace information

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

Find main orientation of patches

- Look at weighted histogram of nearby gradients
  - Any gradient within 80% of peak gets its own descriptor
    - Multiple keypoints per pixel
  - Descriptors are normalized based on main orientation

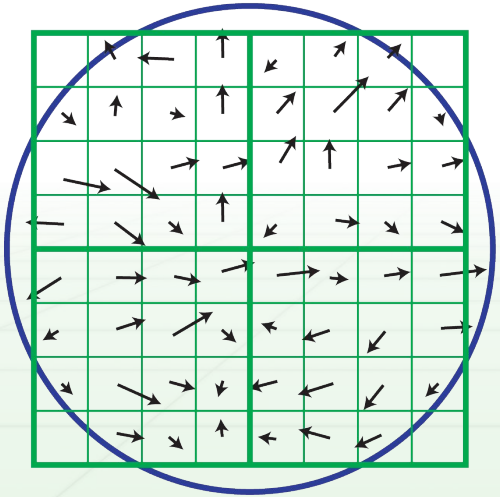
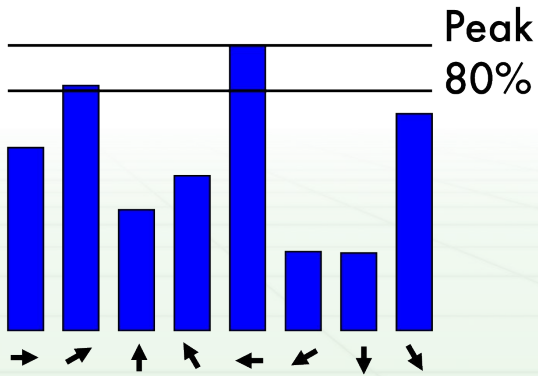
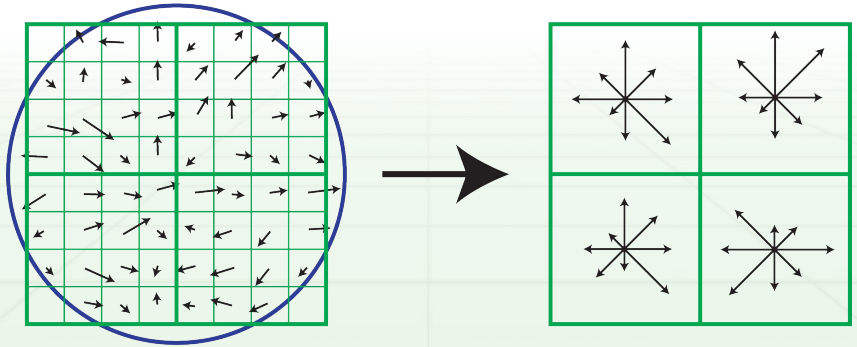


Image gradients



Keypoints are normalized gradient histograms

- Divide into subwindows (2x2, 4x4)
- Bin gradients within subwindow, get histogram
  - Normalize to unit length
  - Clamp at maximum .2
  - Normalize again
  - Helps with lighting changes!



SIFT is great!

- Find good keypoints, describe them
- Finding objects, recognition, panoramas, etc.

SIFT is great!