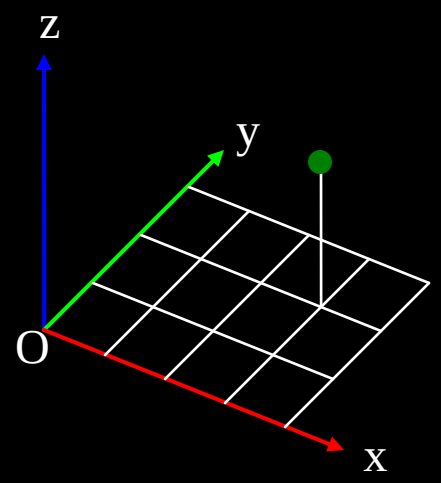
Visual SLAM

Prof Tom Drummond

Representing position:

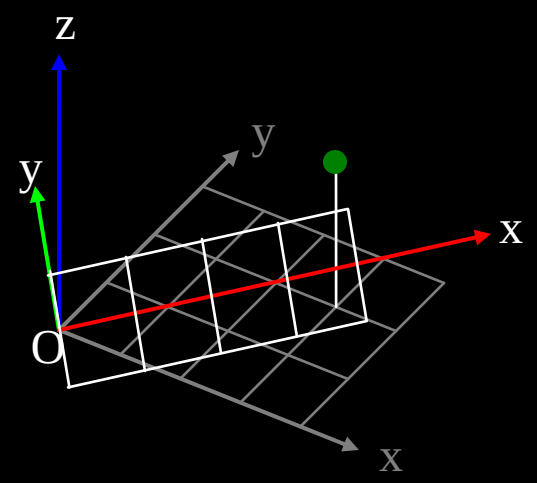
Position in space is represented using a vector for the x, y and z coordinates of a point:



For this to work, we have to agree where the origin is and the direction of each axis.

Rotated Axes

If the axes point in different directions, then the point has different coordinates:



This rotation can be represented by a 3x3 matrix:

Rotation Matrices

Rotations in 3D are represented by 3x3 matrices

But not just any 3x3 matrix – what are the rules?

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Lengths must be preserved:

Angles must be preserved:

Preserving lengths

Preserving angles

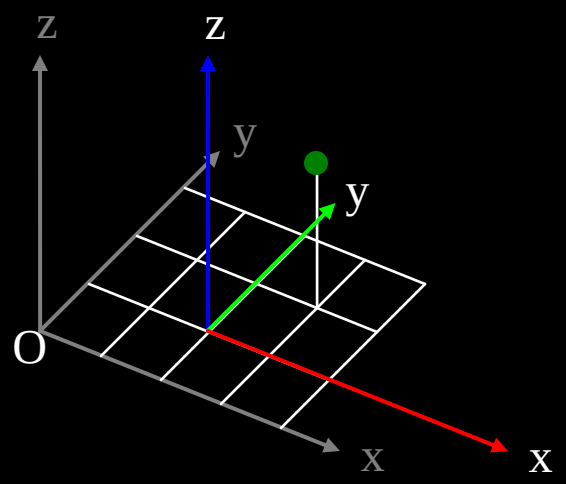
What is $R^T R$

The matrices for rotations about x, y and z axes are very simple:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

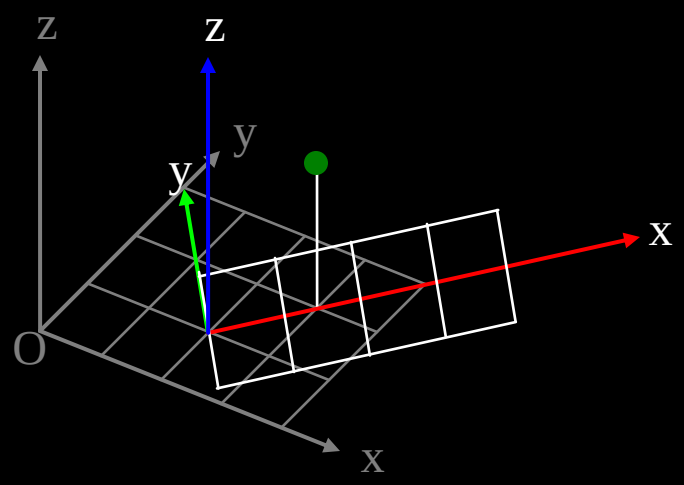
$$R_x = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If we want to put the origin in a different place:



This shift (translation) can be represented by adding a vector:

We can move the origin and rotate the frame at the same time



This can be represented using a rotation matrix and a vector:

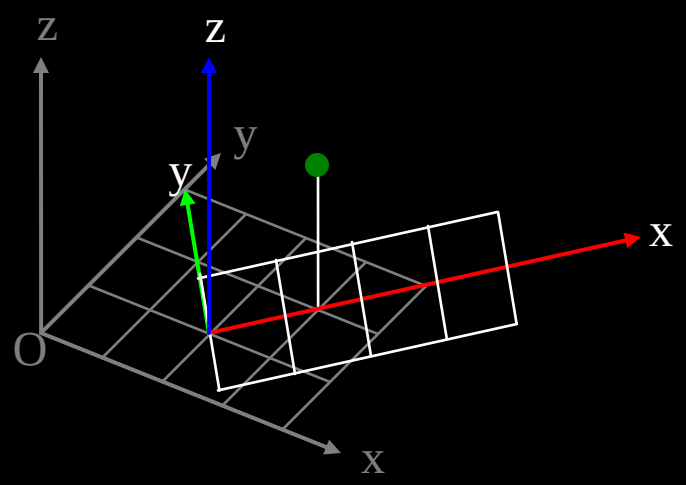
Homogeneous coordinates

Position vectors can be extended to 4 dimensions by adding a 1 at the end. This representation of position is called *homogeneous coordinates*.

This lets us use matrix multiplication to apply a translation:

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} =$$

Homogeneous coordinates



$$\begin{bmatrix} 0.707 & 0.707 & 0 & -2.121 \\ -0.707 & 0.707 & 0 & 0.707 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} 3 \\ 2 \\ 2 \\ 1 \end{pmatrix} =$$

Structure of coordinate transformation matrices is:

$$M = \begin{bmatrix} \begin{bmatrix} & R & \end{bmatrix} & \begin{vmatrix} \\ t \\ \\ \end{vmatrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \qquad M^{-1} =$$

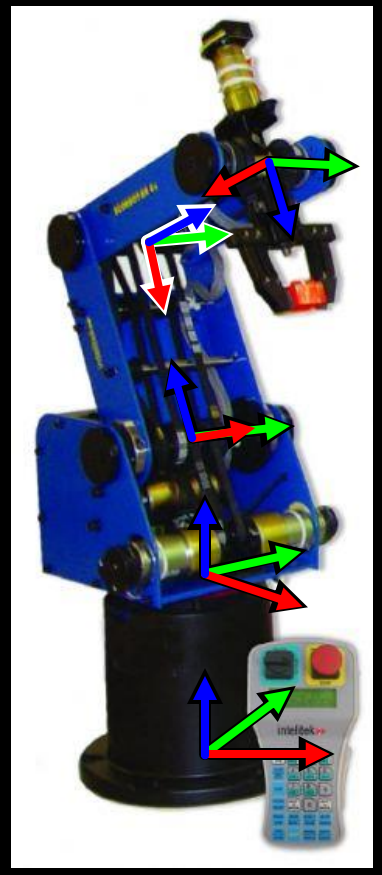
Why does this matter?

The rotation matrix and the translation vector describe the change in coordinate frame.

They tell us how to convert the vector that turns the location of a point in one coordinate frame into the vector that gives the location of *the same point* in another coordinate frame

In robotics we often have *lots* of coordinate frames

Robots have lots of coordinate frames



wrist

elbow

shoulder

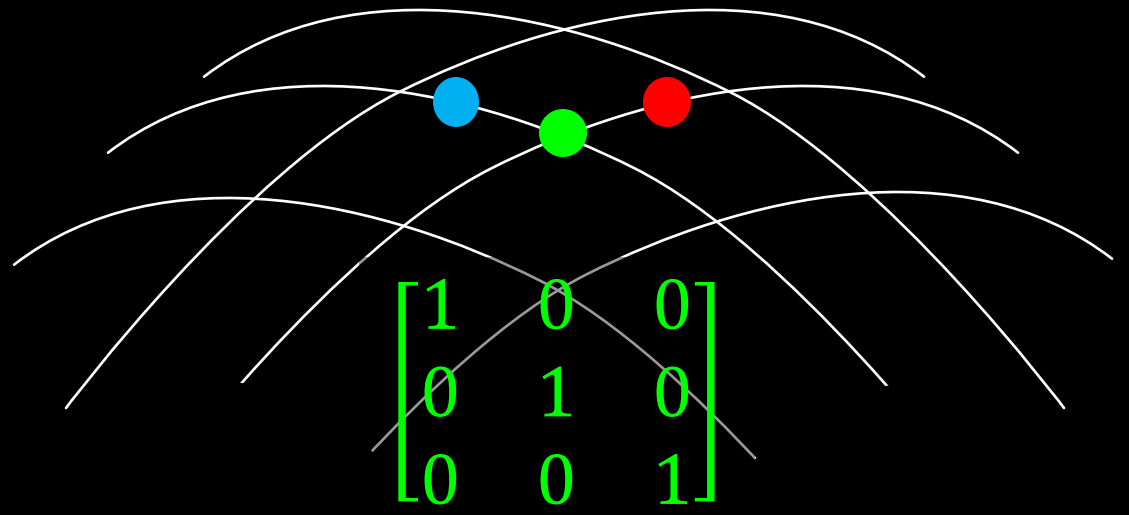
waist

world

Rotation matrices make a 3D “surface” in 9D space

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.98 & -0.17 \\ 0 & 0.17 & 0.98 \end{bmatrix}$$

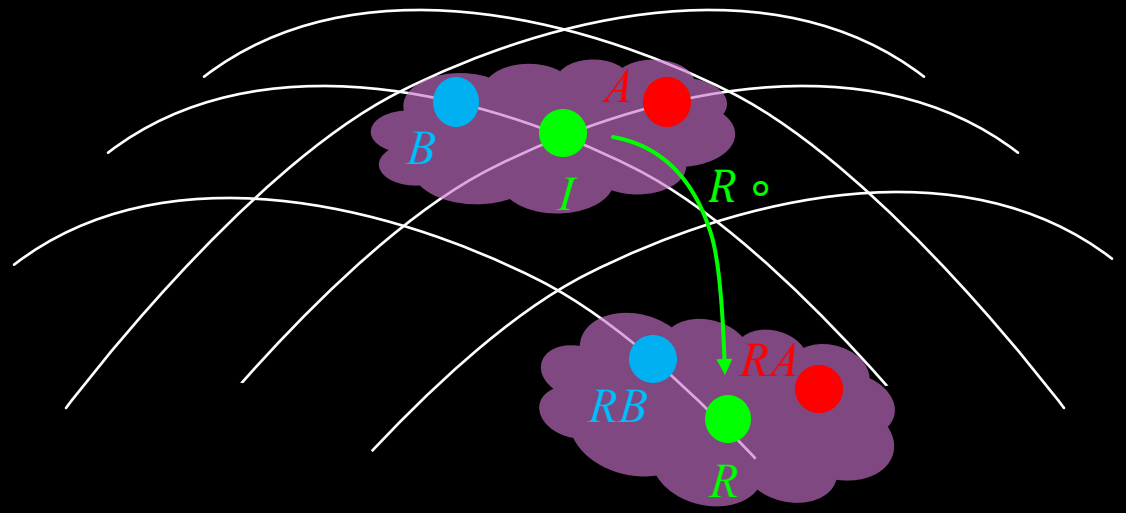
$$\begin{bmatrix} 0.98 & -0.17 & 0 \\ 0.17 & 0.98 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

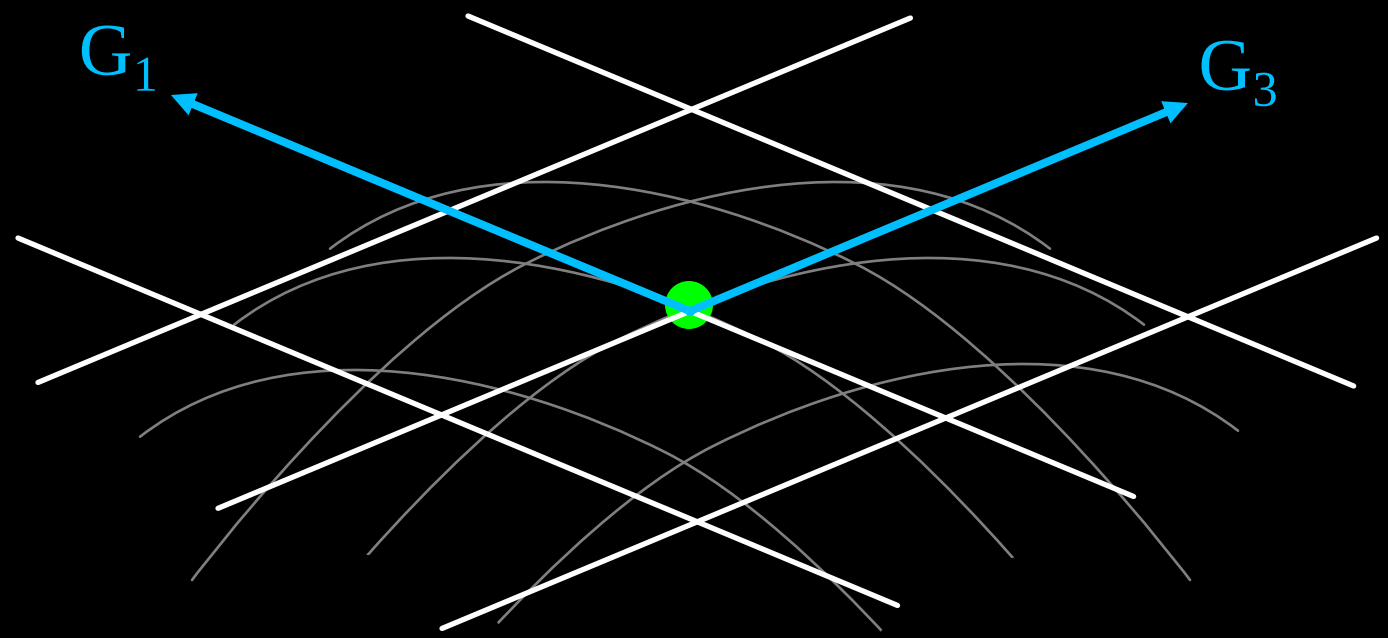
We can learn everything we need to know about the whole manifold by looking at the neighbourhood of the Identity matrix

Why do we care so much about the IDENTITY?



Because a neighbourhood of the Identity can be mapped to a neighbourhood of R by left multiplying by R

Can fit a tangent to the surface at the Identity



The tangent space of rotations is a 3D vector space
The basis axes are called generator matrices (G_1, G_2 , G_3)
Can add (infinitesimally) small amounts of these on to Identity and still have a rotation matrix

Rotation matrices near Identity

Add small values to each element of the Identity matrix

$$R = \begin{bmatrix} 1+a & b & c \\ d & 1+e & f \\ g & h & 1+i \end{bmatrix} \qquad a, b, \ldots, i \quad \text{are small}$$

But $RR^T = I$

$$RR^T = \begin{bmatrix} 1+2a+a^2+b^2+c^2 & d+ad+b+be+cf & g+ga+bh+c+ci \\ d+ad+b+be+cf & d^2+1+2e+e^2+f^2 & dg+h+eh+f+fi \\ g+ga+bh+c+ci & dg+h+eh+f+fi & g^2+h^2+1+2i+i^2 \end{bmatrix} = I$$

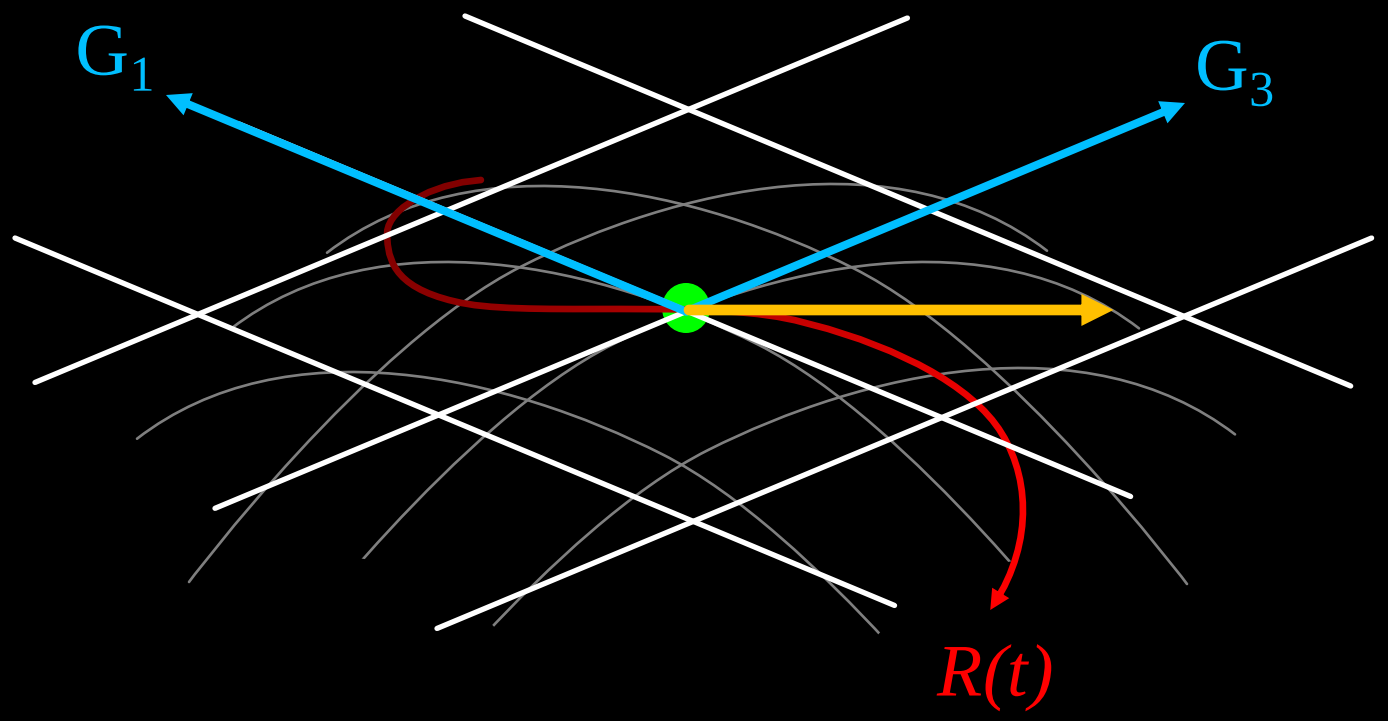
Generators are a basis for the tangent space

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Tangent space is also the space of derivatives

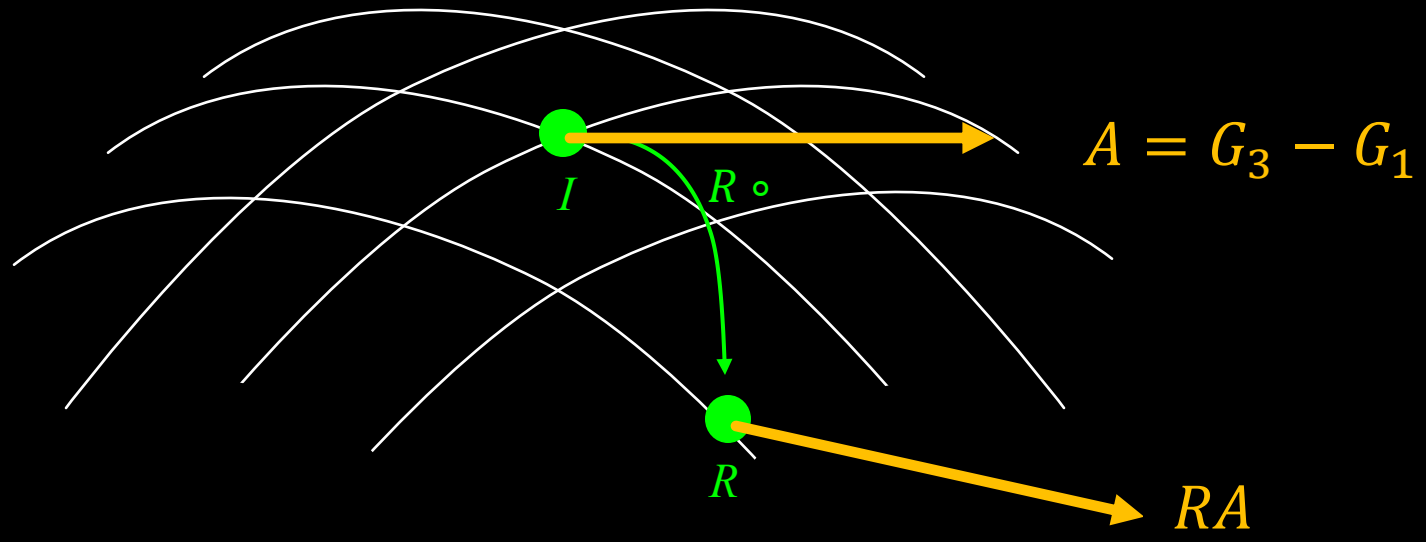


$$\frac{dR}{dt} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$= G_3 - G_1$$

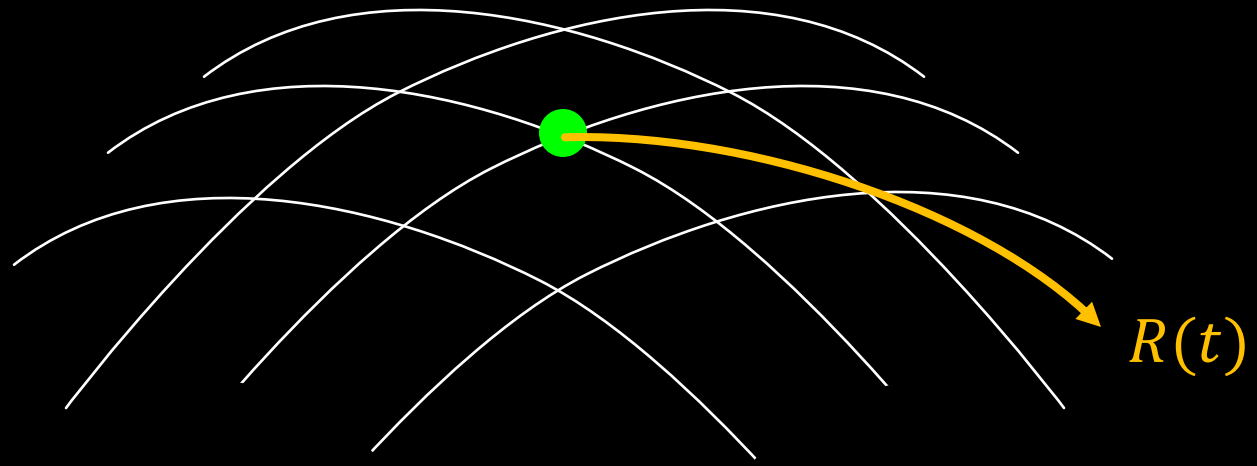
Derivatives are linear combinations of Generators
(any anti-symmetric matrix)

Group elements can move derivatives too



Walking in a straight line (Geodesic)

Start at the Identity ($R(0) = I$)
Choose a direction (derivative = A)
Move the derivative with us as we walk (derivative = RA)



$$\frac{dR}{dt} = RA$$

Simple example:

$$e^{\begin{bmatrix} 0 & -\theta \\ \theta & 0 \end{bmatrix}} =$$

Exponentials of matrices

This can be interpreted using a different expansion for exp

$$e^{A} = \lim_{n \to \infty} \left(I + \frac{A}{n} \right)^{n}$$

As n becomes large M/n becomes infinitesimal

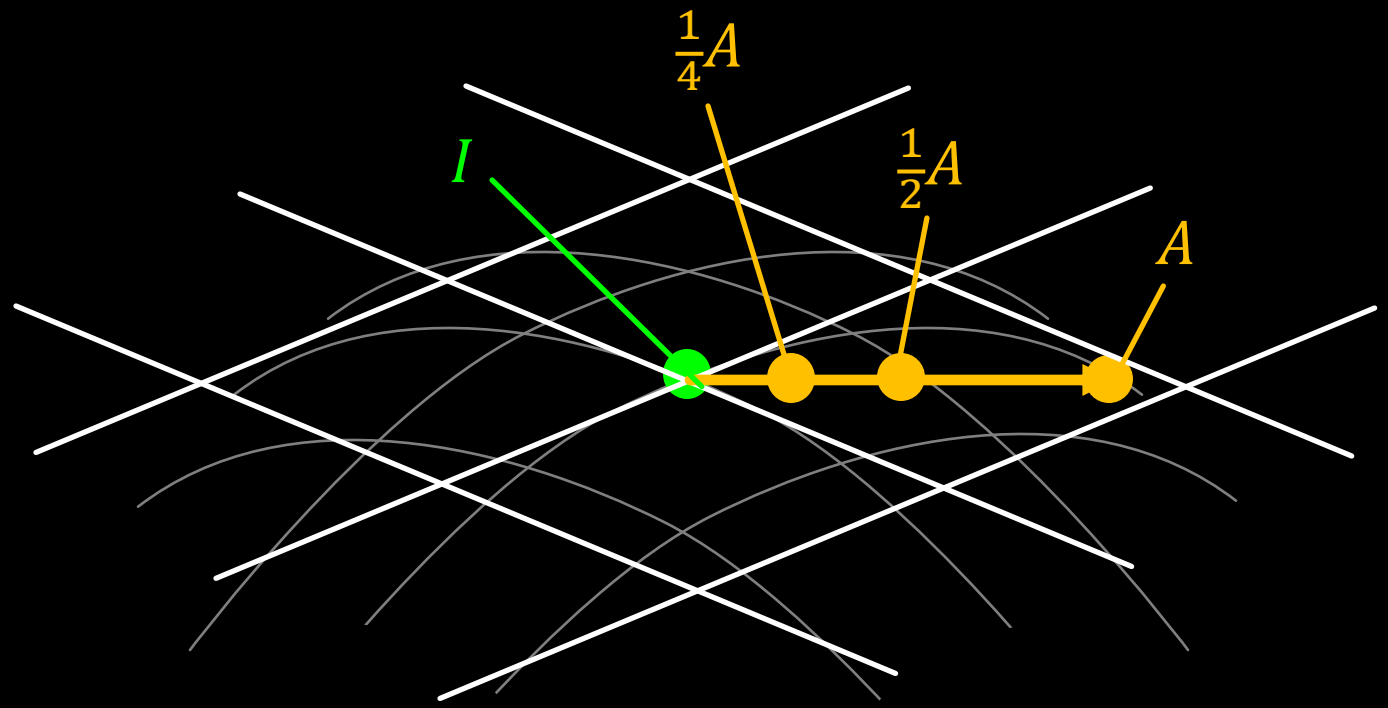
So I+M/n becomes closer to a rotation matrix

(nearer to Identity in tangent plane)

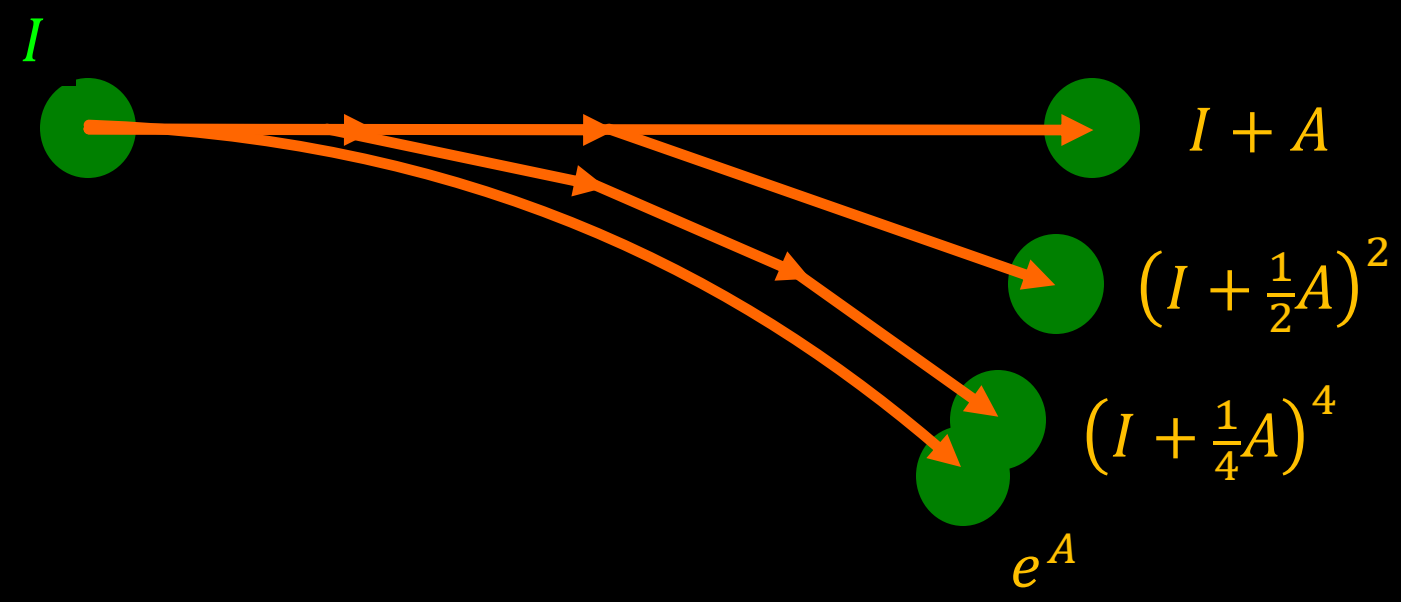
Raising to power n is just multiplying lots of rotations together

Exponentials of matrices

As smaller fractions of M are taken, we get closer to the manifold of rotations

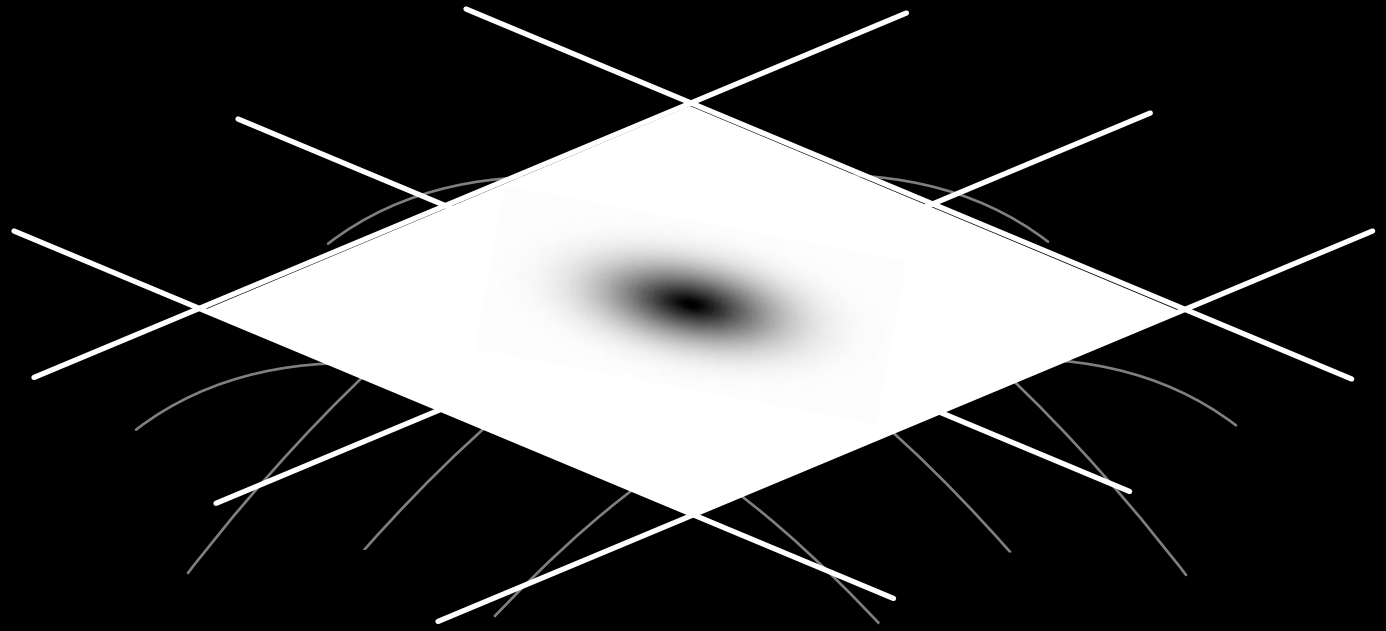


Exponentials of matrices

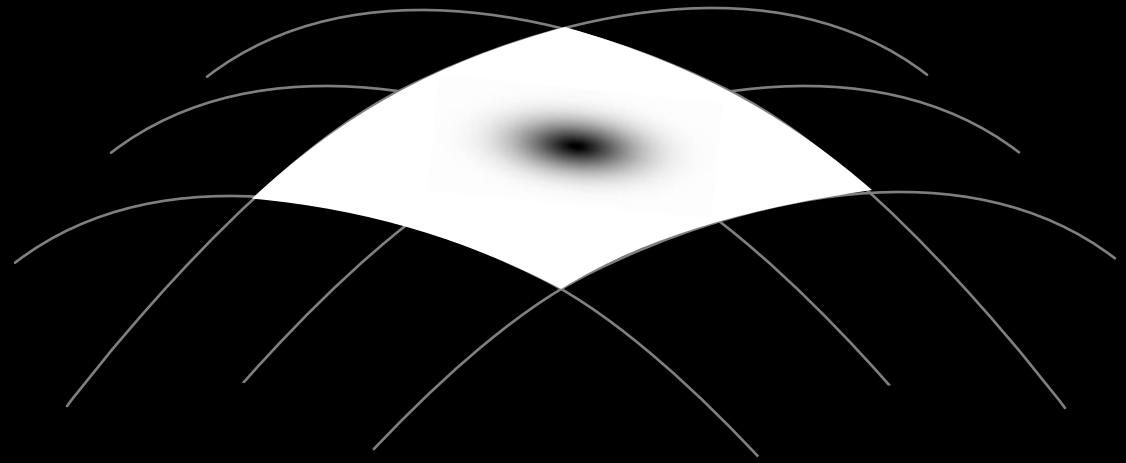


What about uncertainties?

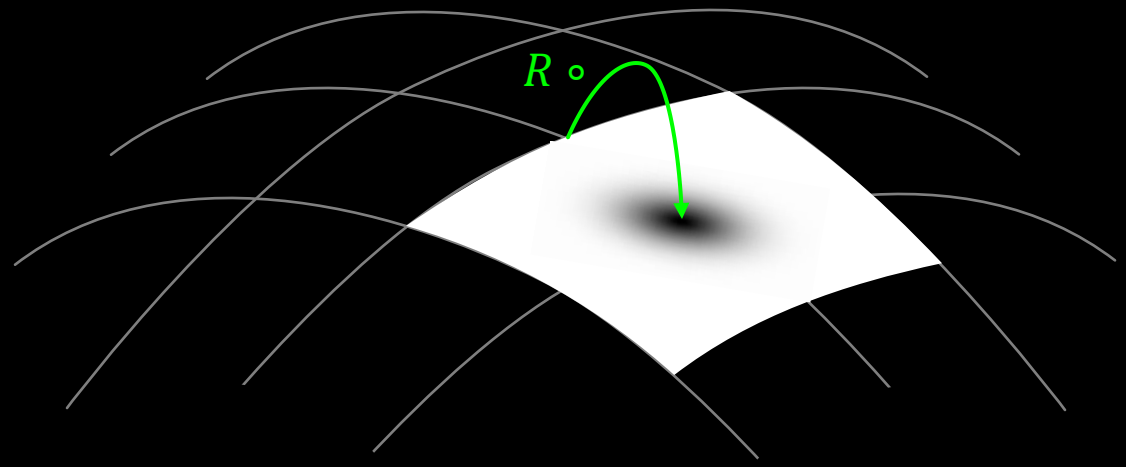
Can place a Gaussian PDF on the tangent space…



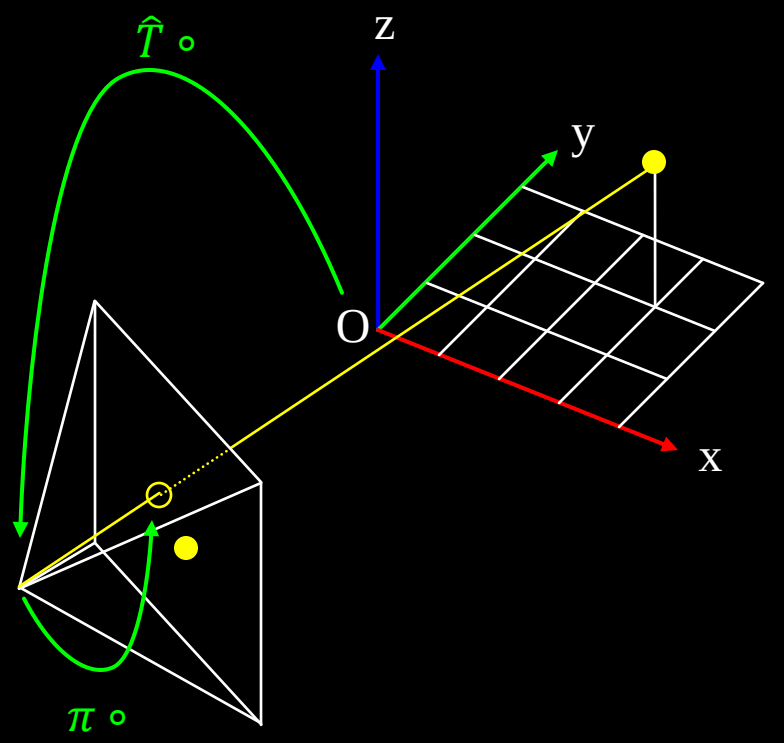
… and project it onto the group using the exponential map



and move the centre of the PDF by left (or right) multiplying by some group element



How do we use this?



Building an edge-based tracker

Start with:

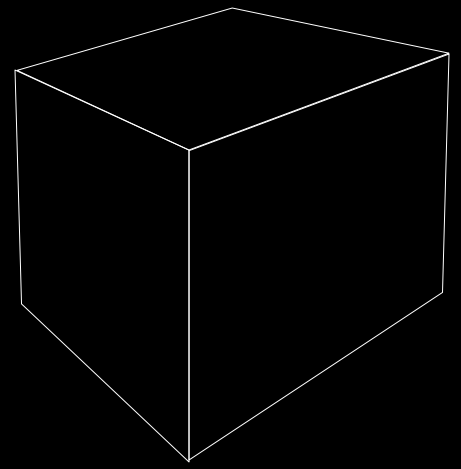
1) a CAD edge model of the object we want to track
2) an estimate of E (the pose)

Want:

- a refined estimate of E

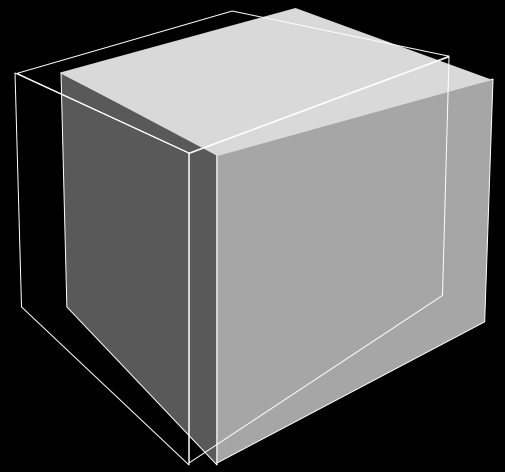
Algorithm

1) Render the CAD model using the estimate of E.



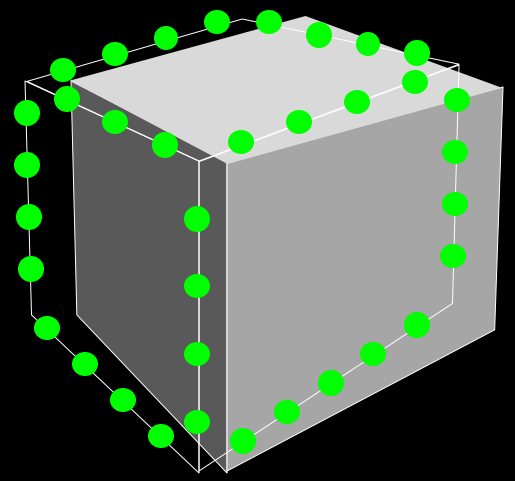
Algorithm

1) Render the CAD model using the estimate of E.
2) Get a frame of video (this will not line up exactly)



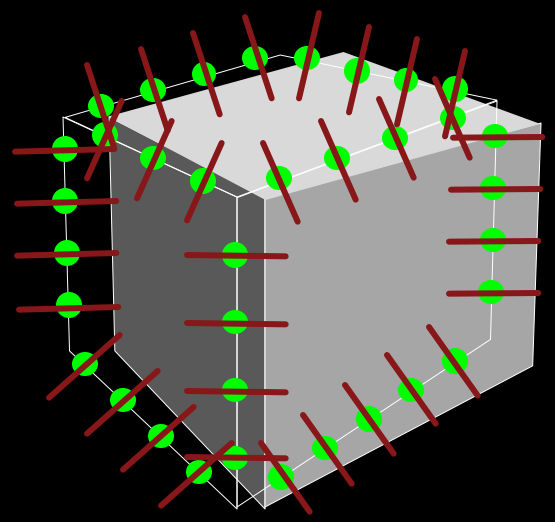
Algorithm

1) Render the CAD model using the estimate of E.
2) Get a frame of video (this will not line up exactly)
3) Sample the rendered edges



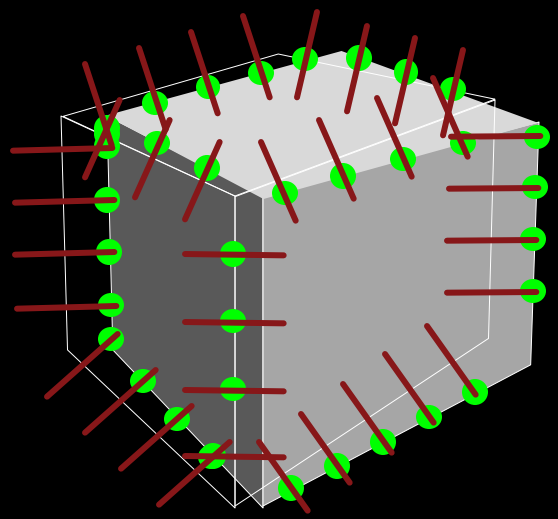
Algorithm

1) Render the CAD model using the estimate of E.
2) Get a frame of video (this will not line up exactly)
3) Sample the rendered edges
4) Walk perpendicular to the rendered edge from each sample



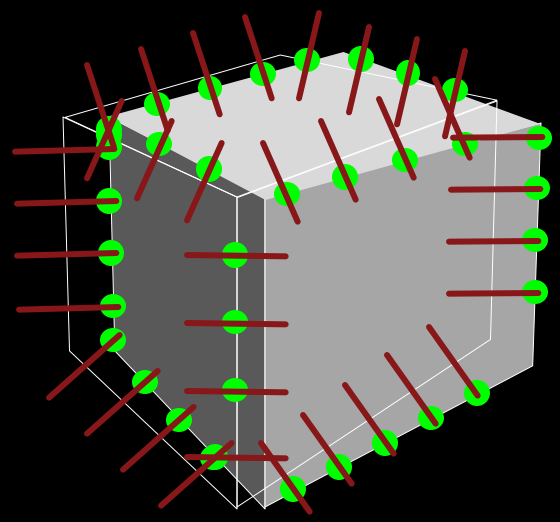
Algorithm

1) Render the CAD model using the estimate of E.
2) Get a frame of video (this will not line up exactly)
3) Sample the rendered edges
4) Walk perpendicular to the rendered edge from each sample
5) and find a sudden change in brightness in the live image



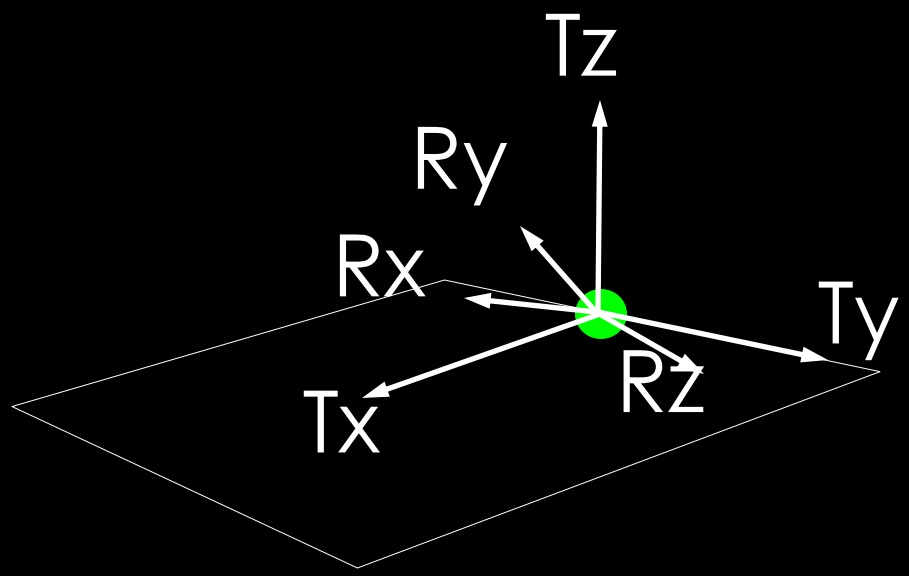
Algorithm

6) Use all these measurements to update E



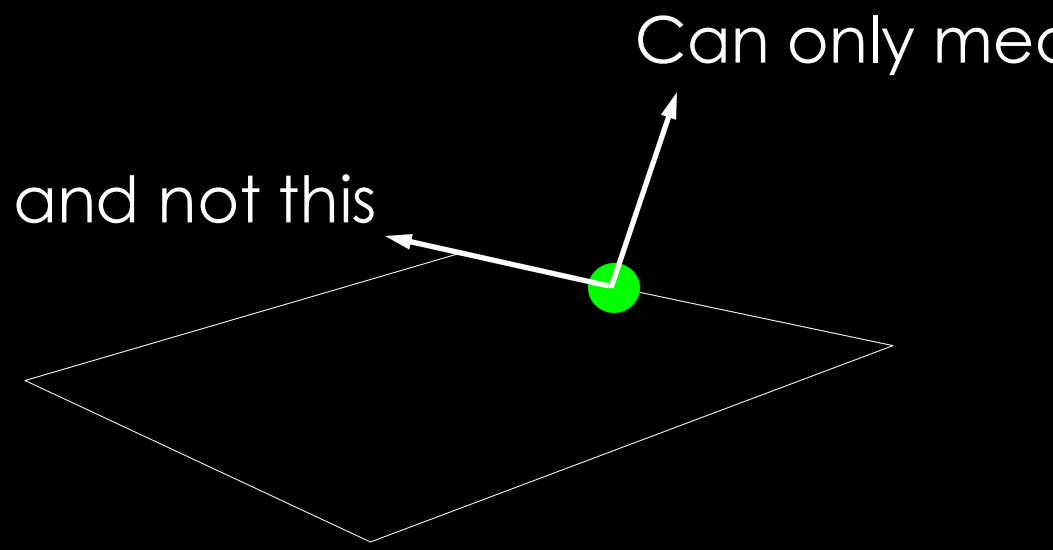
Motion from changing parameters

For each sample point, we can work out how it would move in the image if we were to change each of the six parameters controlling E (Translation in the x, y and z directions and Rotation around the x, y and z axes).



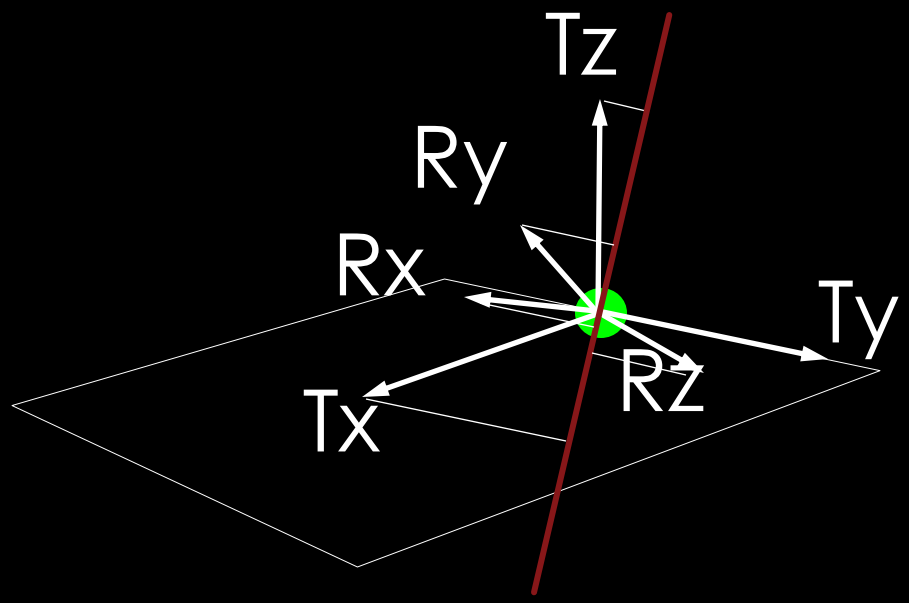
What can be measured?

But we can only measure how much the sample point has moved perpendicular to the edge. The component of motion parallel to the edge can't be seen.



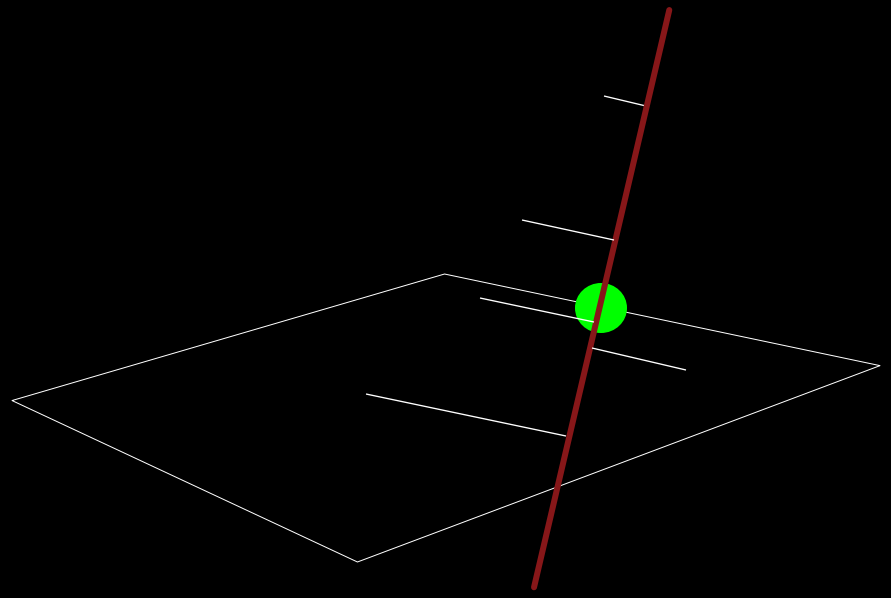
Projecting the parameter motions

So we project each of the 6 motions onto the line perpendicular to the edge. This is how far we should have to look along the red line to find the edge if 1 unit of that motion were applied to the CAD model.



Compare to observed motion

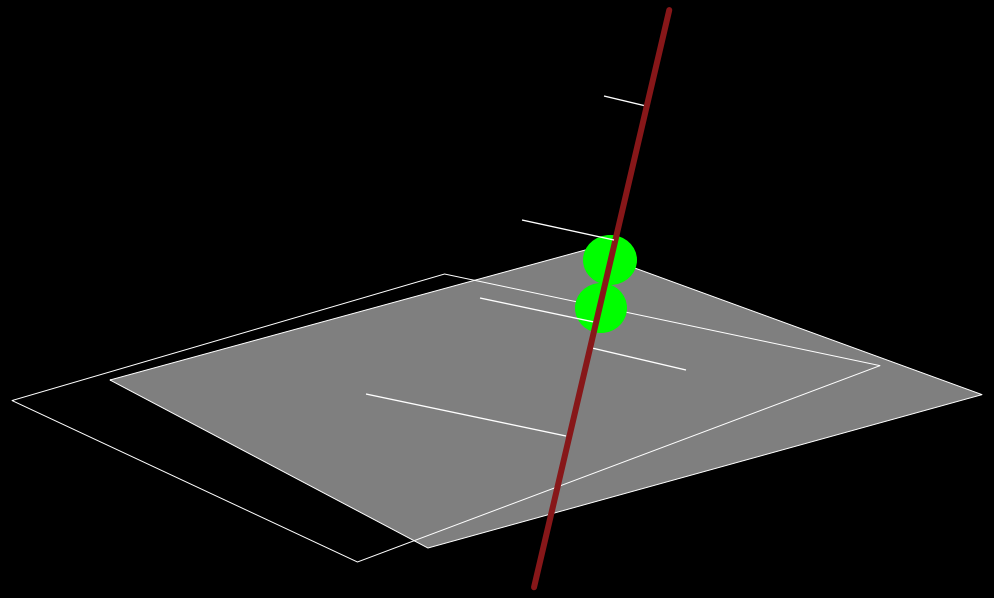
So we project each of the 6 motions onto the line perpendicular to the edge. This is how far we should have to look along the red line to find the edge if 1 unit of that motion were applied to the CAD model.



Observed motion is a combination of parameter motions

And we can measure the amount the edge has actually moved

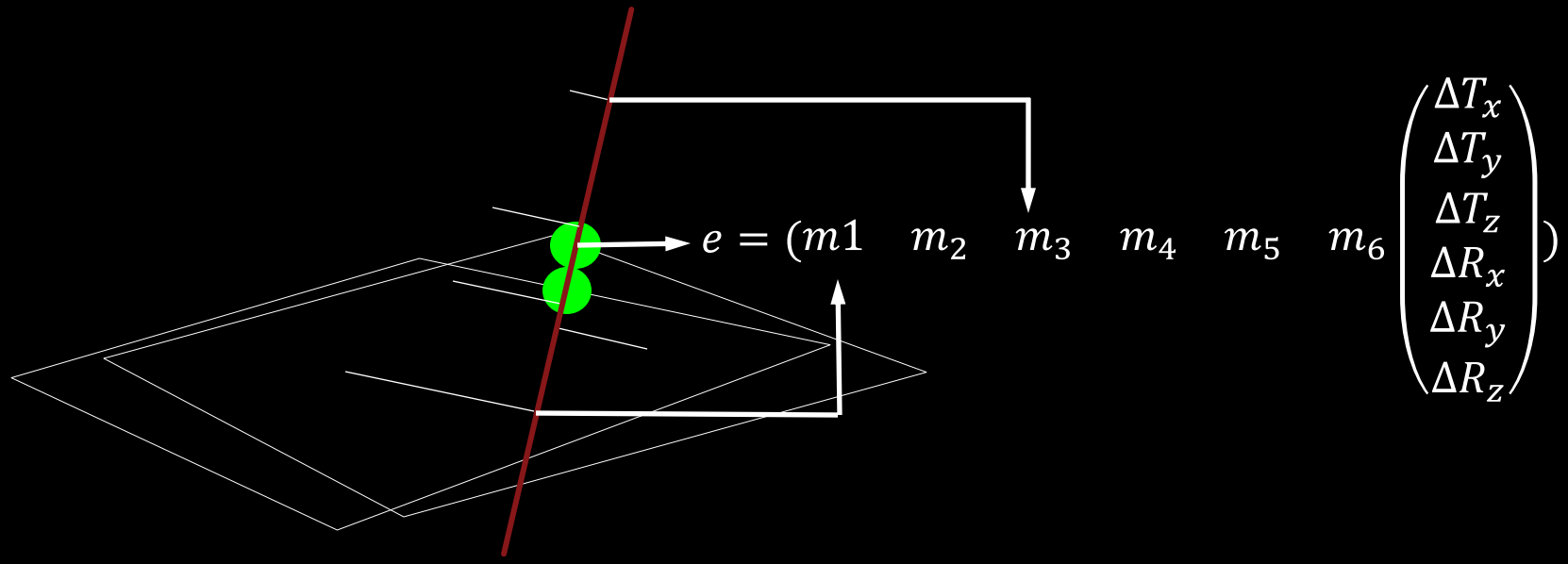
So we have to explain the observed motion as a combination of the six calculated motions (e.g. a positive amount of Tz or a negative amount of Tx or some combination)



Observed motion is a combination of parameter motions

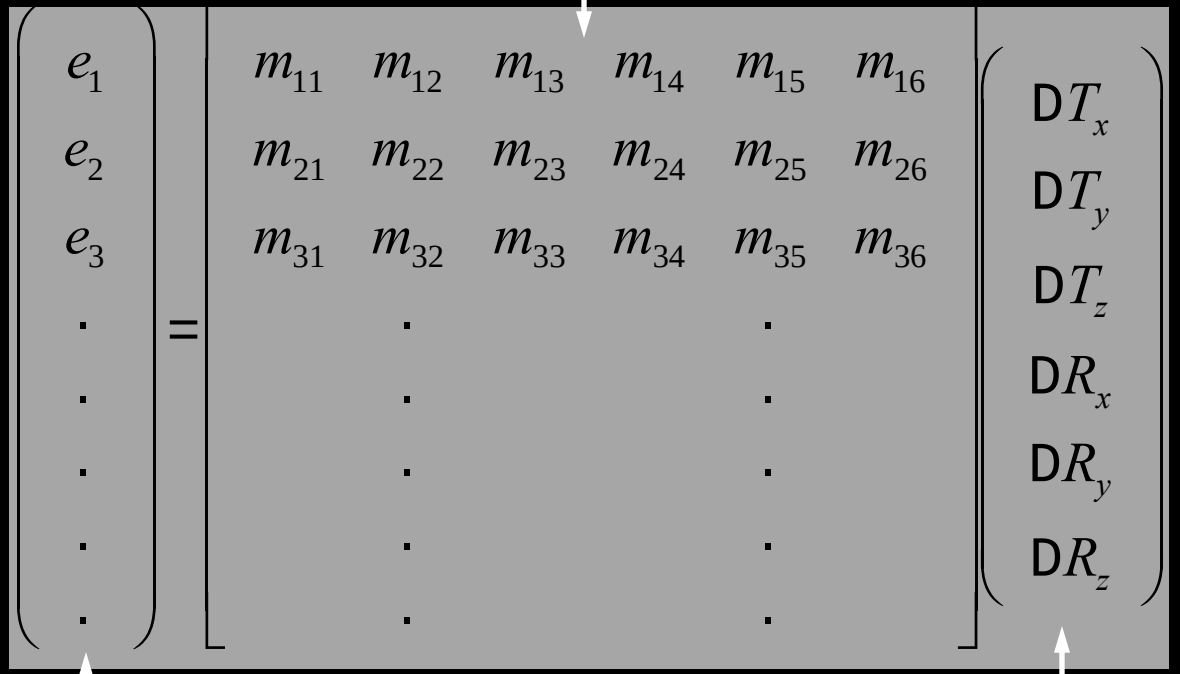
And we can measure the amount the edge has actually moved

So we have to explain the observed motion as a combination of the six calculated motions (e.g. a positive amount of Tz or a negative amount of Tx or some combination)



As a matrix equation

We can calculated one row of 6 motions for each sample point



We can measure one error for each sample point

We want a single set of 6 parameter changes

As a matrix equation

We can write this using vector and matrix notation

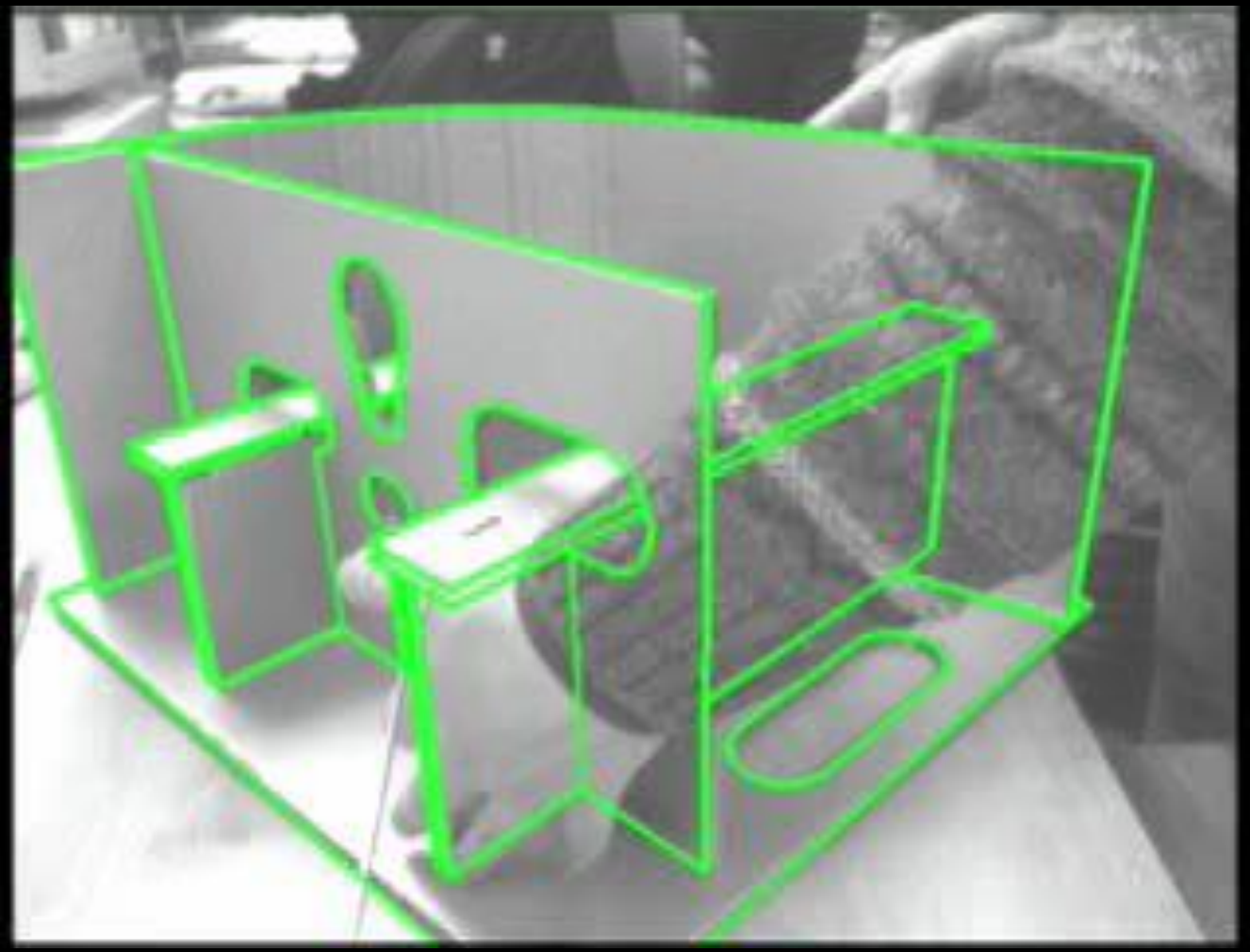
$$\bar{e} = M\bar{\Delta}$$

This has the least squares solution:

$$\bar{\Delta} = M^{\dagger}\bar{e}$$

$$= [M^T M]^{-1} M^T \bar{e}$$

What about bad data?

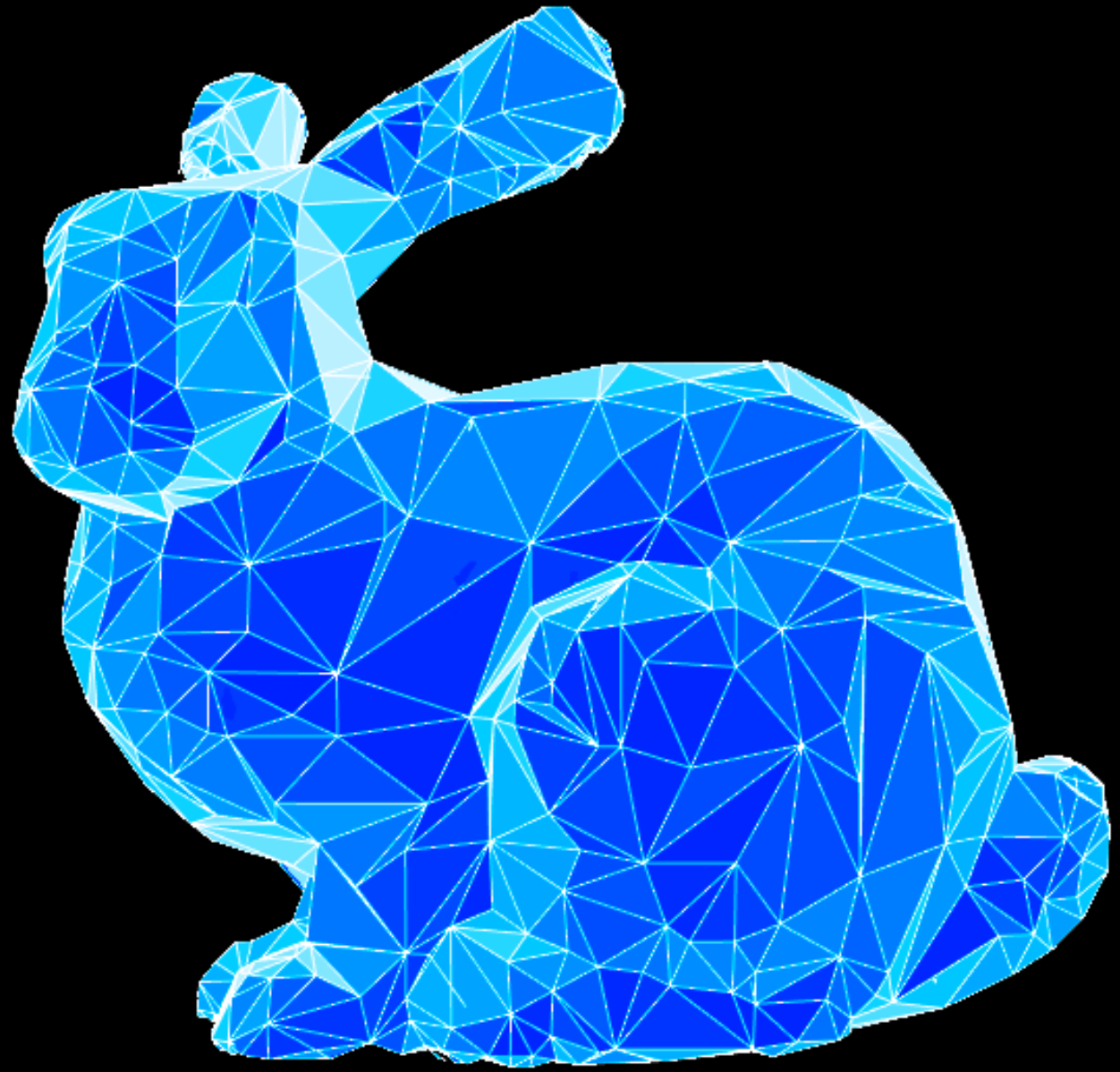


Scene representations

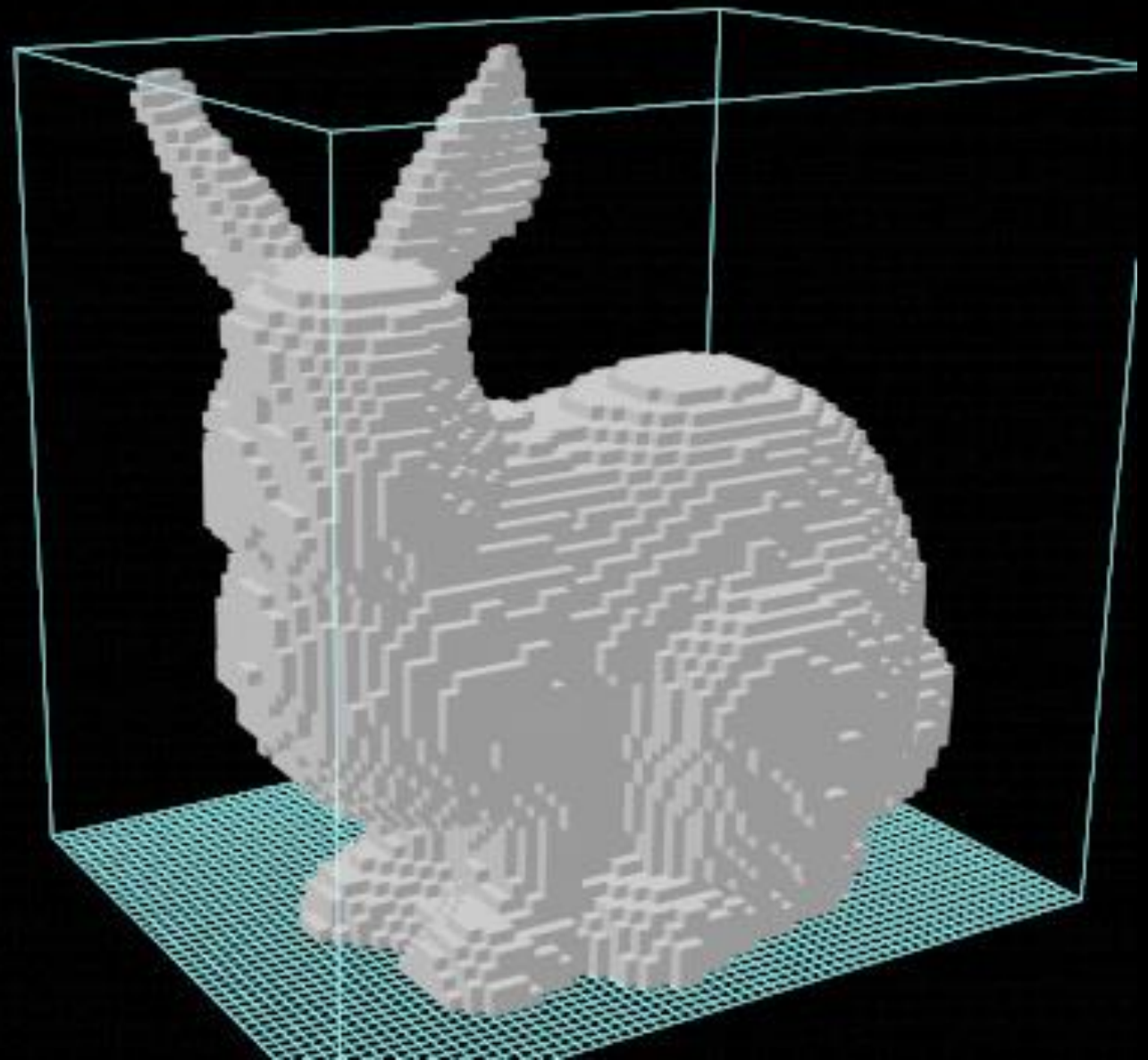
- Point Clouds
  - Scene is represented by a finite set of points each with (x,y,z) coordinate (and possibly a colour).
- Surface Mesh
  - Join the points up with triangles to form a surface. Usually have a texture for each triangle so we can render textured objects (e.g. in a computer game).
- Occupancy grids (voxels, octrees)
  - Block worlds like Minecraft. The scene is represented as a set of occupied blocks (possibly with colour or texture). Can save storage by recursively grouping voxels together into groups of 8 that are all filled or all empty to get Octrees
- Signed distance functions
  - Block world, but with a more accurate representation of the surface. Each voxel has a floating point number that represents the distance from the centre of the voxel to the surface with a sign convention (e.g. –ve means inside the object and +ve means outside). Often only care about voxels that are close to the surface so we ‘truncate’ (ie clamp) the SDF to some range

Point Cloud

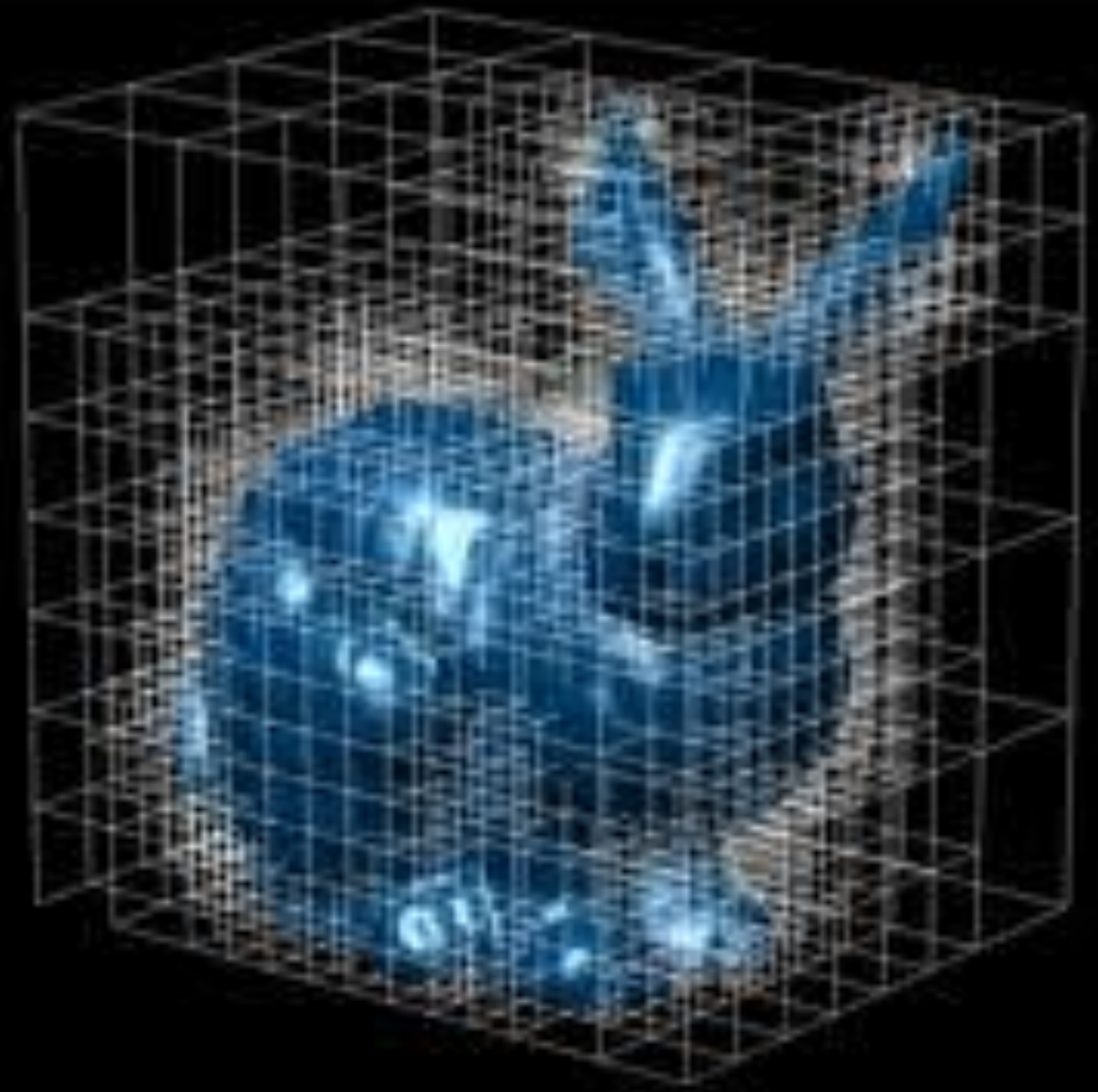
Surface Mesh



Voxels



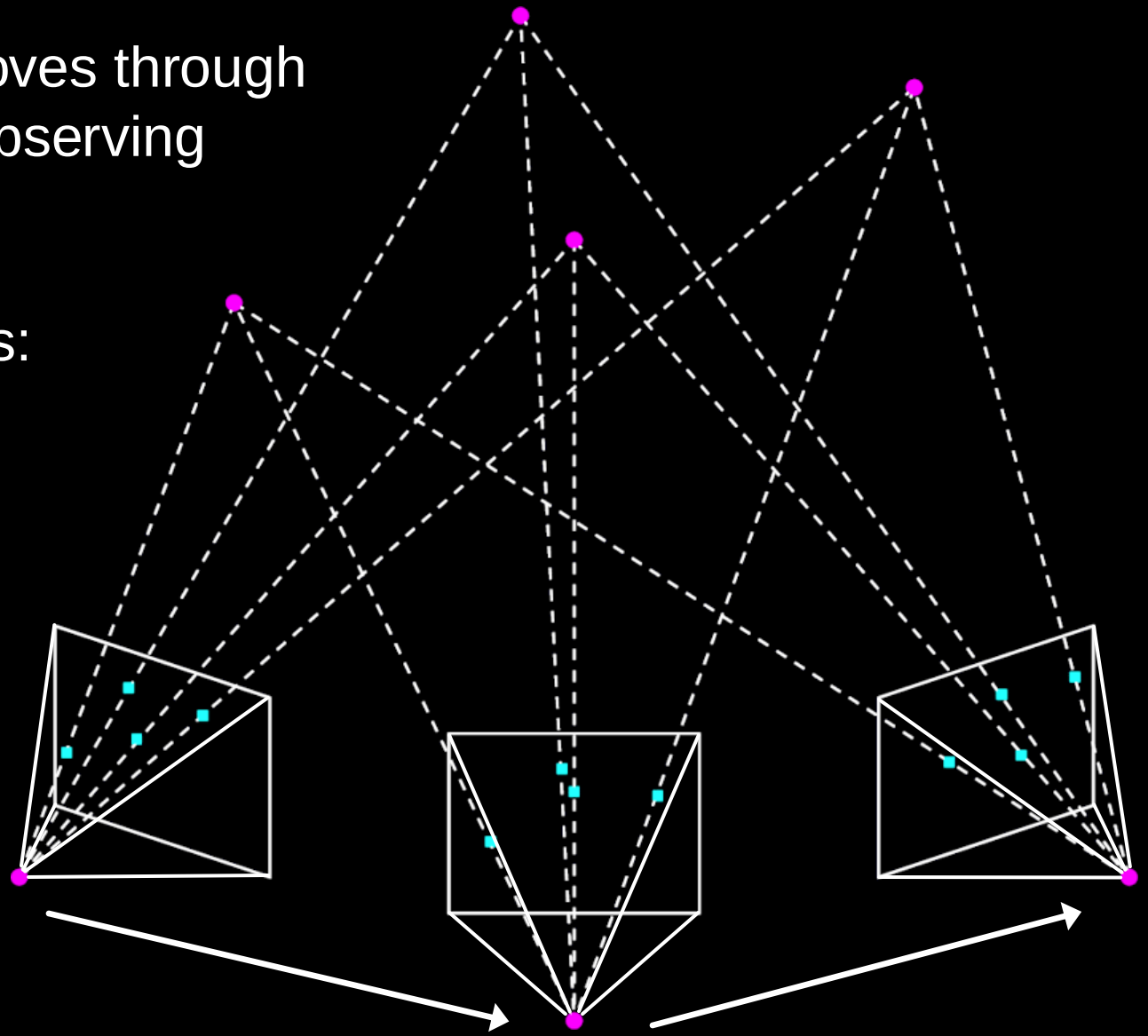
Octree



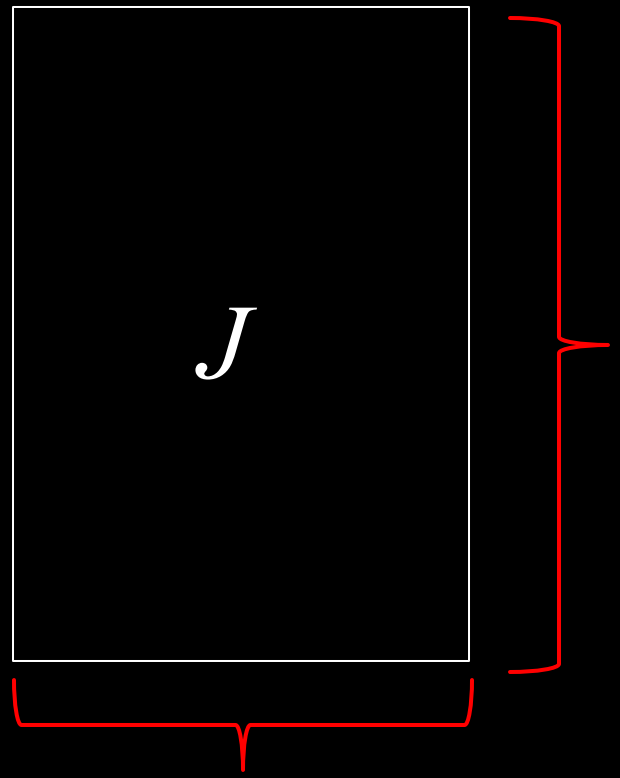
Simultaneous Localisation and Mapping (SLAM)

Camera moves through the world observing landmarks

2 Questions:



Jacobian of partial derivatives



One row per measurement.

We get two (or three) of these every time a camera sees a landmark (x, y [,z]) or (u, v [,q]).

There are 1000s of these per camera per frame – millions or billions in total

One column per model parameter: 3 (or more) per landmark, 6 per camera per frame, + calibration parameters – thousands or millions in total.

Typically solve $Jx=b$

$x = J^{\dagger}b = [J^{T}J]^{-1} J^{T}b$

Jacobian of partial derivatives

Fortunately, J is sparse

Unfortunately, J is ill conditioned (often 10^{20} or worse)

Fortunately there are some good solvers: g2o, Ceres, GTSAM, SLAM++