COMP30019 Graphics and Interaction Kinematics

Adrian Pearce

Department of Computing and Information Systems
University of Melbourne

The University of Melbourne

Lecture outline

Kinematics

'I am robot (am I?)'

- Forward kinematics for me is:
  - *Apply a behaviour to my robot arm*—e.g. rotate one of my arms by pre-determined angles for each joint (degree of freedom)
  - *Apply a steering behaviour to my robot body*—e.g. translate myself by a pre-determined vector on the 2D surface (using my wheels)
- Inverse kinematics for me is:
  - *How do I configure my robot arm to reach it*?—determine all of the previously unknown angles for my joints.
  - *Can my robot arm reach that object?*
  - *Is the object reachable from where I am?*—do I need a forward kinematics translation by moving adjacent to it?

Forward kinematics

Forward kinematics can be posed as applying homogeneous transformations relative to the coordinate system.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In forward kinematics we know the aggregated rotation and scale factors, r_{ij}, and translation factors, t_i, of the transformation matrix.
Therefore, given point x, y, z we can computer point x', y', z'.

Inverse kinematics

The inverse kinematic problem is one of the most difficult to solve, as a set of simultaneous equations must be solved.

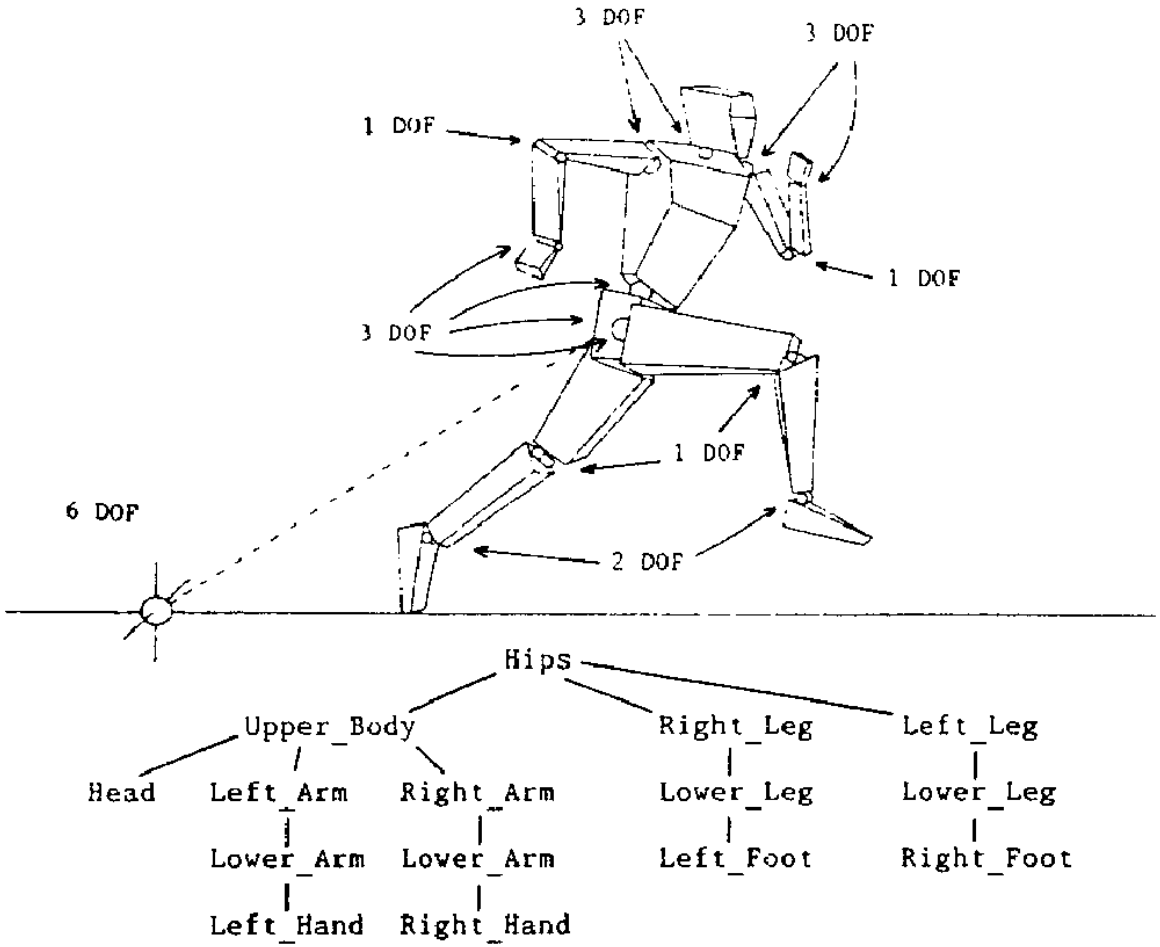
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In inverse kinematics we know points x, y, z and x', y', z'. We must therefore induce the aggregated rotation and scale factors, r_{ij}, and translation factors, t_i, for the required transformation matrix.

Degrees of freedom in human animation

In the human skeleton, each joint angle is typically specified by three angles and therefore have three degrees of freedom (DOF).

- With increased degrees of freedom (more joints) get more possible solutions (or redundancy of movement).
- After including, legs, hands, feet, head and a spine will have well over *100 degrees of freedom*, possibly without even including fingers and toes!



For the human figure shown in the figure there are 38 degrees of freedom—*Controlling dynamic simulation with kinematic constraints, behaviour functions and inverse dynamics*, by Paul M. Isaccs and Micael F. Cohen, Cornell, SIGRAPH 1987

Solving Inverse Kinematics

Problems can arise in solving simultaneous equations for inverse kinematics, including

- the existence of multiple solutions (or joint configurations),
- the possible non-existence of any solution (unreachable), or
- singularities of matrix equations (we will see later).

Solution techniques

We can solve simultaneous equations using *iterative* or *algebraic* algorithms.

Algebraic algorithms

Algebraic techniques for solving linear simultaneous equations are known from the field of *numerical methods*.

Iterative algorithms

- Iterative algorithms work by calculating an *approximate solution* which converges to the exact solution,
- typically be used for solving non-linear simultaneous equations or when numerical stability if paramount important.

Algebraic approach

If $\mathbf{Ax} = \mathbf{b}$ and we want to solve for $\mathbf{x}$, then $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.
$\mathbf{A}^{-1}$ is termed the inverse matrix of $\mathbf{A}$.
$\mathbf{A}^{-1}$ can be constructed whenever $\mathbf{A}$ is *non-singular*.
The following conditions are true and equivalent if a matrix has an inverse, or at least one or more *non-singularities* (solutions)
e.g. if lines or planes intersect in geometrical terms

1. no one equation in the system can be expressed as a linear combination of the others
2. the determinant of $\mathbf{A}$ is non-zero ie. $\mid \mathbf{A} \mid \neq 0$
3. the columns (rows) of the coefficient matrix are *linearly independent*

Therefore, to solve $\mathbf{Ax} = \mathbf{b}$ directly, we must compute

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

where

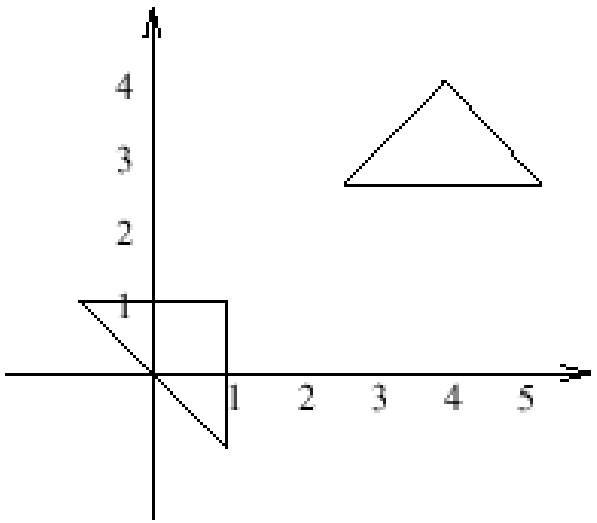
$$A^{-1} = \frac{1}{det(A)} adj(A)$$

and before doing so we must also check if $|\ \mathbf{A}\ | \neq 0$ to make sure inverse exists, otherwise $\mathbf{A}$ is singular (and therefore doesn't have a solution).

- Note that checking if the determinant is zero requires the concept of "zero" in floating point, which requires the use of a small non-zero constant that is just larger than the maximum expected error as a result of numerical precision.
- This will depend on the machine you implement on.

Algebraic example

Initially, one of the simpler inverse problems is explored which can be solved algebraically — it takes three points to define an affine transformation in two dimensions (2D). For example, an affine transformation matrix can be derived in homogeneous form, that defines the transformation where point $(1, 1)$ goes to $(4, 4)$, point $(1, -1)$ goes to $(4 + \sqrt{2}, 4 - \sqrt{2})$ and point $(-1, 1)$ goes to $(4 - \sqrt{2}, 4 - \sqrt{2})$.



Looking at this visually suggests (i) a rotation 45 degrees anticlockwise about origin followed by (ii) a translation by 4 in x direction and $4 - \sqrt{2}$ in the y direction.

- However we want to automate this!

The (unknown) transformation matrix must therefore satisfy each of the following transformations for each point (since it is an affine transformation)

$$\begin{bmatrix} a_{xx} & a_{xy} & b_x \\ a_{yx} & a_{yy} & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a_{xx} & a_{xy} & b_x \\ a_{yx} & a_{yy} & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4+\sqrt{2} \\ 4-\sqrt{2} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a_{xx} & a_{xy} & b_x \\ a_{yx} & a_{yy} & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4-\sqrt{2} \\ 4-\sqrt{2} \\ 1 \end{bmatrix}$$

Which gives rise to the following set of six simultaneous equations (for each unknown coefficient of the unknown transformation)

$$
\begin{array}{rrrrrrrl}
a_{xx} & +a_{xy} & & & +b_x & & = & 4 \\
 & & +a_{yx} & +a_{yy} & & +b_y & = & 4 \\
a_{xx} & -a_{xy} & & & +b_x & & = & 4+\sqrt{2} \\
 & & +a_{yx} & -a_{yy} & & +b_y & = & 4-\sqrt{2} \\
-a_{xx} & +a_{xy} & & & & +b_y & = & 4-\sqrt{2} \\
 & & -a_{yx} & +a_{yy} & & +b_y & = & 4-\sqrt{2}
\end{array}
$$

So to get the required transformation matrix, we must solve equation $\mathbf{Ax} = \mathbf{b}$ to derive coefficients in terms of solution vector $\mathbf{x}$ where

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 4 \\ 4 \\ 4+\sqrt{2} \\ 4-\sqrt{2} \\ 4-\sqrt{2} \\ 4-\sqrt{2} \end{bmatrix} \text{ and}$$

$$\mathbf{x} = \begin{bmatrix} a_{xx} \\ a_{xy} \\ a_{yx} \\ a_{yy} \\ b_x \\ b_y \end{bmatrix}$$

Notice that the problem of deriving a suitable transformation appeared to rely on two factors

(i) The calculation of determinants of matrices (necessary for direct calculation of inverse matrices), and

(ii) Consideration of numerical precision in the notion of *zero* when checking to see if a solution exists (that determinant is not zero).

Question: what is the computational complexity of calculating determinants? What does it mean in practice for computer graphics?

The complexity of matrix algebra

If $\mathbf{A} = \left[a_{ik}\right]$ $(m \times n)$ and $\mathbf{B} = \left[b_{kj}\right]$ $(n \times s)$ then under multiplication

$$\mathbf{C} = \mathbf{AB}$$

$$= \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{i1} & \cdots & a_{in} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{ij} & \cdots & a_{1j} \\ \vdots & & \vdots & & \vdots \\ b_{n1} & \cdots & b_{nj} & \cdots & a_{mn} \end{bmatrix} = \left[c_{ij}\right]$$

Each element c_{ij} must be found by taking the dot product of the ith row vector of **A** and the jth column vector of **B**.

Matrix multiplication therefore involves a lot of arithmetic operations, particularly for large matrices.

ie. $c_{ij} = \begin{bmatrix} a_{i1} & \cdots & a_{in} \end{bmatrix} \begin{bmatrix} b_{1j} \\ \vdots \\ b_{nj} \end{bmatrix} = \Sigma_{k=1}^{n} a_{ik} b_{kj}$

For a $n \times n$ matrix, there are

$$n \times n \times (n \times \textit{multiplications} + n \times \textit{additions}) = n^3 \times (\textit{multiplcations} + \textit{additions})$$

therefore worst case complexity is $\mathcal{O}(n^3)$.
The complexity of matrix multiplication stems from the complexity of computing dot products of vectors.

Recall that the determinant of a two-dimensional matrix is denoted as $| \mathbf{A} |$, or $det(\mathbf{A})$.

For example, the determinant of matrix $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ is

$$| \mathbf{A} | = \begin{vmatrix} a_{11} a_{12} \\ a_{21} a_{22} \end{vmatrix} = a_{11} a_{22} - a_{12} a_{21}$$

More generally, a determinant for a matrix **A** is determined by

$$|\mathbf{A}| = \sum_{i=1}^{k} a_{ij} C_{ij}$$

where C_{ij} is the cofactor of a_{ij} defined by

$$C_{ij} \equiv (-1)^{i+j} M_{ij}$$

and M_{ij} is the minor of matrix **A** formed by eliminating row i and column j from **A**.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1k} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & a_{k3} & \dots & a_{kk} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} & \dots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k2} & a_{k3} & \dots & a_{kk} \end{vmatrix}$$

$$-a_{12} \begin{vmatrix} a_{21} & a_{23} & \dots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k3} & \dots & a_{kk} \end{vmatrix} + \dots \pm a_{1k} \begin{vmatrix} a_{21} & a_{22} & \dots & a_{2(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \dots & a_{k(k-1)} \end{vmatrix}$$

Calculating determinants directly (eg. using cofactors) therefore has complexity $\mathcal{O}(n!)$, growing factorially with the size of the matrix!

This won't work for 100 Degrees of Freedom Human model.

Calculating a determinant directly has very bad worst-case computational complexity, $\mathcal{O}(N!)$.

- This is way beyond the complexity of algorithms we like to invoke even less often than at the screen refresh rates experienced in graphics. For example hashing is $O(n)$ and retrieval from balanced trees is typically $\mathcal{O}(nlogn)$.

This means that direct methods of calculating determinants are not computationally tractable in large matrices, except for small n.

The challenge of Inverse Kinematics

Since solutions to simultaneous equations is central to geometric transformations and inverse kinematics

- can't efficiently calculate inverse matrices directly for solving simultaneous equations fast enough for real time, and
- can't efficiently check for singularities necessary to guarantee numerical stability or meaningful solutions.

In practice iterative algorithms are used that reduce the complexity of the arithmetic.

Cyclic Coordinate Descent Algorithm

https://sites.google.com/site/auraliusproject/ccd-algorithm

Final IK - CCD script from Unity Assest Store:
https://www.youtube.com/watch?v=-z_l7Jdz8Bo

Unity IK resources

Unity Documentation: `https://docs.unity3d.com/Manual/InverseKinematics.html` Unity 5 IK Tutorial

(YouTube):
`https://www.youtube.com/watch?v=EggUxC5_lGE`

Kinematics Summary

- Forward kinematics equations are relatively easy to solve (provided required transformations are known ahead of time) since solution simply involves matrix multiplication.
- Solution to inverse kinematics problems involves the solution of simultaneous equations.
- If sufficiently many points are defined,
  - then inverse kinematics problems can be posed in terms of solutions to sets *linear* simultaneous equations,
  - otherwise iterative (or incremental) inverse kinematic solution techniques must be used (covered later).