Computer Animation and Visualisation – Assignment 1

Transformations, Forward and Inverse Kinematics

Sebastian Starke
Ph.D. Student in Character Animation & Artificial Intelligence
Sebastian.Starke@ed.ac.uk
Graphics & Animation is fun!

- Powerful state-of-the-art engine for graphics and game development
- Extremely user friendly / easy-to-use
- Available for Win, Mac, Linux :)

Transformations, FK and IK

- Fundamental maths / tools in graphics, animation and robotics
- Different representations, algorithms and solving techniques
- Although IK is a very traditional field, is still an active research problem for complex articulated geometries
Component-based programming paradigm with support for C# / JS
Variables can be serialised during edit-time, and everything is processed in the update-loop during runtime
Highly active community, lots of tutorials, AssetStore full of helpful stuff
Game-update loop (most important callbacks only)

The functions below apply to every object which derives from Mono Behaviour

- void Awake() Called once at the very beginning
  (Set up your references here for example)
- void Start() Called once in the beginning after „Awake“
  (Initialise your variables here for example)
- void Update() Called every frame at the specified frame rate
- void LateUpdate() Called every frame after Update()
- void FixedUpdate() Called every frame rate at a parallel constant frame rate for physics updates
<Transform> Class
https://docs.unity3d.com/ScriptReference/Transform.html

→ Minimal component for any game object in the scene

→ To access its variables and attributes, simply type „transform.<variable>“

Components in general

→ Components are available for editing in the inspector in edit-mode, but can also be modified during runtime via code. Components are beautiful, because they are modular.

→ To add any component on your game object, simply type:
  myComponent = gameObject.AddComponent<Type>();

→ To retrieve any component on your game object, simply type:
  myComponent = gameObject.GetComponent<Type>();
Homogeneous transformations are a uniform representation for performing translation (1x3) and rotation (3x3) in Euclidean space.

\[ T = T_1 \cdot T_2 \cdot \ldots \cdot T_n \]

Matrix multiplication is NOT commutative! Left-hand side multiplication refers to a relative-to-world transformation, while right-hand side multiplication refers to a local relative-to-parent transformation.

Rotations can be represented using Euler-angle matrices or Quaternions.

Euler-angles require a predefined rotation order (like ZXY, XYZ, ...)

Common problem is the „gimbal lock“, which results in a loss of a degree of freedom.

Quaternions are 4D vectors that can avoid such issues by modeling a rotation uniquely by an arbitrary vector \((x,y,z)\) and an angle \(\alpha\).

\[ q_\alpha(\alpha) = (q_x, q_y, q_z, q_w) = (a_x S^{\alpha/2}, a_y S^{\alpha/2}, a_z S^{\alpha/2}, C^{\alpha/2}) \]
FK and IK describe the mapping between joint and Cartesian space.

**Forward Kinematics:** $f(\theta) = X_{1,\ldots,k}$  
**Inverse Kinematics:** $\theta = f^{-1}(X_{1,\ldots,k})$

→ Computing FK is straightforward and always yields the same solution.

→ Computing IK is not as easy, and can have zero up to infinite solutions.

→ Typically, IK is solved by means of numerical optimisation by minimising an objective function. One of the most popular methods is based on the Jacobian matrix.
The Jacobian

→ The Jacobian matrix denotes a first-order derivative of the system, which in our case maps from joint positions to joint velocities.

→ Traditionally used as a gradient-descent technique for solving IK.

\[ J(\theta)_{ij} = \left( \frac{\delta X_i}{\delta \theta_j} \right) \]

→ To compute the Jacobian, slightly modify each joint variable independently and store the resulting end effector vector either row- or column-wise for each variable. This gives an approximation of the end effector velocity. (Can generally be used for differentiable optimisation.)

→ Use this matrix in combination with the Jacobian Transpose or Pseudoinverse method for solving IK.

**Transpose:** \[ \Delta \theta = \alpha J^T \hat{e} \]

**Pseudoinverse:** \[ \Delta \theta = J^T (J J^T)^{-1} \hat{e} \]

**Reading:** [http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/iksurvey.pdf](http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/iksurvey.pdf)
You may use this as well:

Matrix Utility Class:
www.starke-consult.de/UoE/Matrix.cs

Transformation Utility Class:
www.starke-consult.de/UoE/Transformation.cs
Questions? :)

Questions? :)