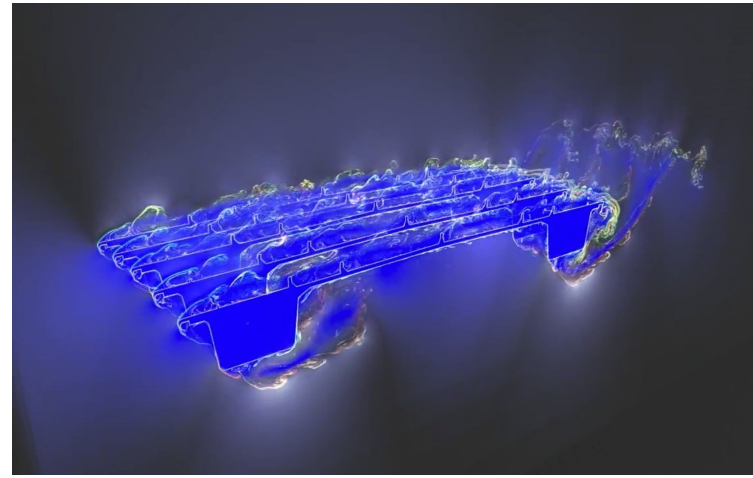
Computer Graphics: 14-Computer Animation

Prof. Dr. Charles A. Wüthrich,
Fakultät Medien, Medieninformatik
Bauhaus-Universität Weimar
caw AT medien.uni-weimar.de

Introduction

- Computer Animation has been a fascinating branch of Computer Graphics
- Plenty of complex themes:
  - physically-based animation (forward/ inverse kinematic, spring-mass systems, particle systems, rigid body simulation, etc.)
  - physics simulation
  - motion capture from real entities, like humans (face, body, movements, etc.)
  - animation of fluids, like liquids and gasses (fluid dynamics, etc.)



Introduction

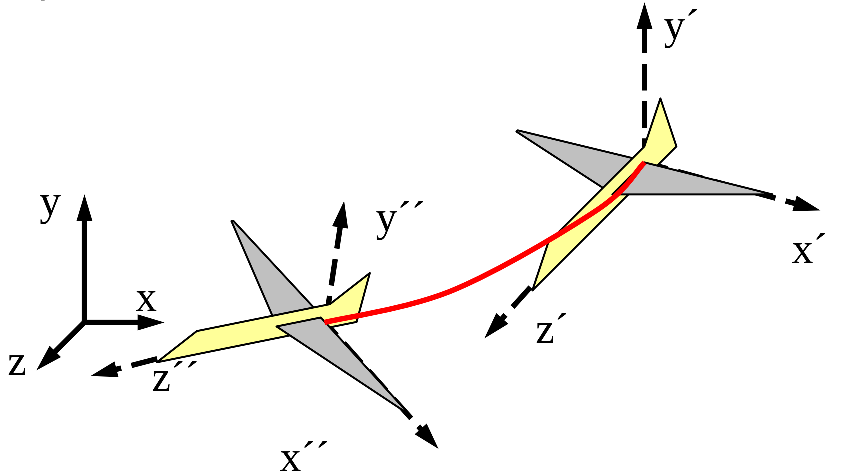
- modeling and animating human figures (reaching, grasping, walking, dressing, etc.)
- motion capturing
- Facial animation (muscle models, skin, lip synchronization, etc.)
- Particle Systems, Herds. Schools, Crowd simulations

Animation

- Object definition for animation
- Movement paths, camera paths
- Articulated figures, Forward and Inverse Kinematics
- Motion capturing

Representing object orientation

- Suppose that I defined two key positions of a rigid body, and that I want to compute the equal steps between the two positions to compute the animation (each key position been defined by a Rotation-translation pair)
- For the translation part, it seems to be easy to interpolate between the positions.... but the rotation?



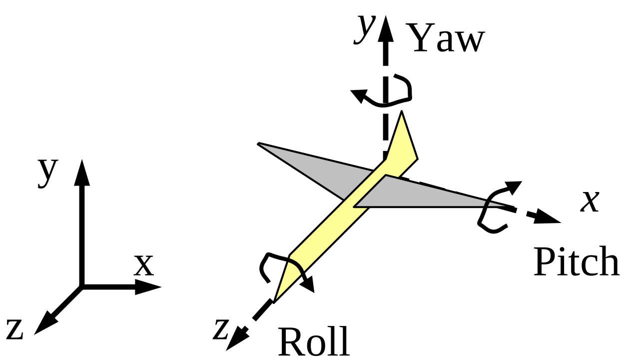
- Direct interpolation does not work, because the resulting interpolation matrices will not be normalized....
- But there ARE alternative methods to do this:
  - Fixed angle
  - Euler angle
  - Axis angle
  - Quaternions

Fixed angle representation

- Angles used to rotate around fixed axes
- One can rotate first around one main axis, then the second and then the third
- As long as one keeps always the same order, one should be fine
- But, if you apply consequently those, the second rotation will influence back the first rotation
- This effect is called *gimbal lock*
- The same problem makes interpolation between key positions a problem sometimes
- The resulting rotations will make the object swing out of the desired rotating plane

Euler angle representation

- Here the axes of rotation are on the local coordinate system of the object



- Also here, the order of the rotations is indifferent
- In fact, this method is very similar to fixed axes, and has same advantages and disadvantages
- Euler‘s rotation theorem: any orientation can be derived from another by ONE rotation around a particular axis

Quaternions

- This is the better approach to do interpolation of intermediate orientations when the object has 3 DOF
- A *quaternion* is a 4-tuple of real numbers $[a,b,c,d]$.
- Equivalently, it is a pair $[s,\underline{v}]$ of a scalar s and a 3D vector $\underline{v}$.
- More, it can be defined as $w + xi + yj + zk$ (where $i^2 = j^2 = k^2 = -1$ and $ij = k = -ji$ with real w, x, y, z)

- On quaternions one defines two operations:
  - Addition:
    $[s_1,\underline{v}_1]+ [s_2,\underline{v}_2]=$
    $[s_1+s_2,\underline{v}_1+\underline{v}_2]$
  - Multiplication:
    $[s_1,\underline{v}_1] \cdot [s_2,\underline{v}_2]=$
    $[s_1 \cdot s_2-\underline{v}_1\cdot \underline{v}_2,$
    $s_1 \cdot \underline{v}_2+s_2 \cdot \underline{v}_1 +\underline{v}_1 \times \underline{v}_2]$
  - Note that multiplication is associative, but NOT commutative $\Rightarrow q_1q_2{}^1q_2q_1$

Quaternions: definitions

- Units:
  - Additive: $[0,\underline{0}]$
  - Multiplicative: $[1,\underline{0}]=[1,0,0,0]$
- Let $\underline{v}=[x,y,z]$.
  Inverse:
  - $q^{-1}=[s,\underline{v}]^{-1}=(1/\|q\|)^2 \cdot [s,-\underline{v}]$, *where* $\|q\|=(s^2+\|\underline{v}\|)^{1/2}$
- Obviously, $qq^{-1}=[1,0,0,0]$
- A point in 3D space can be also represented as the quaternion $[0.\underline{v}]$.
  - or, alternatively, a vector from the origin
- Property:
  $[0,\underline{v}_1] \cdot [0,\underline{v}_2]=$
  $[0,\underline{v}_1 \times \underline{v}_2]$ iff $\underline{v}_1 \times \underline{v}_2=0$
- *Def:* Unit-length quaternion is a quaternion q such that $\|q\|=1.$
- Obviously $\forall q$, $q/\|q\|$ is a unit length quaternion

Rotating vectors through quaternions

- Consider a vector $[0,\underline{v}]$, and consider a quaternion q:
  - The rotated vector v' of v through the quaternion q is the vector
    $$v' = Rot_q(v) = q \cdot v \cdot q^{-1}$$
  - A sequence of rotations can be chained:
    $$Rot_p(Rot_q(v)) = q(p \cdot v \cdot p^{-1}) \cdot q^{-1}$$
    $$= (q \cdot p) \cdot v \cdot (p^{-1} \cdot q)^{-1} = Rot_{pq}(v)$$
  - Note that:
    $$Rot^{-1}(Rot(v)) = v$$

Camera paths

- Like in real movies, in Computer Animation cameras are allowed to move
- This create a number of problems and issues which have been addressed with time
- How do I define movement of a camera?

Following a path

- Animating an object to move along a path is quite natural and common
- Not only following the path is needed: also moving the orientation
- Typically, one would have a local coordinate system associated with the object
- Let the coordinates be (u,v,w), and suppose they are right handed

- Suppose the origin of the coordinate system follows the curve P(s), and that the movement of P(s) is specified
- Call POS the current position
- One can view the u,v,w coordinates as a view vector, an up vector and a vector perpendicular to u and v
- This is similar to camera definition in Computer Graphics

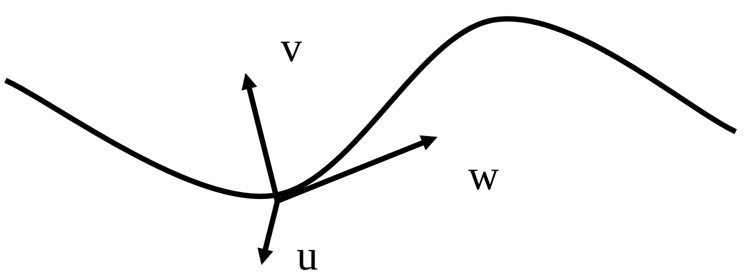
Following a path: Frenet Frame

- The orientation of the camera system can be made dependent from the properties of the curve P(s)
- A Frenet frame is given by the following axes definitions
  - w follows the tangent of the curve (its first derivative P´(s))
  - v is orthogonal to w and in the direction of the second order derivative (P´´(s))
  - u is the cross product of w and v
- In symbols:

  $w=P'(s)$

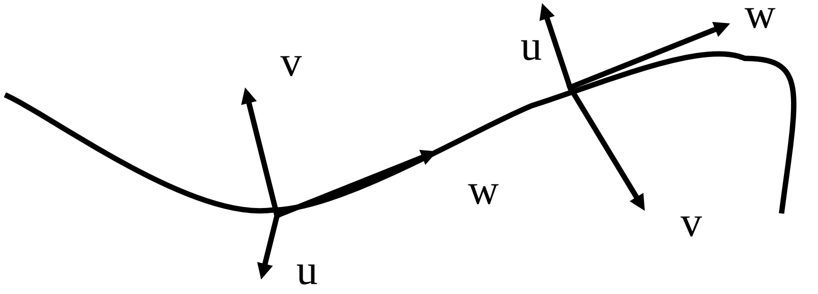
  $u=(P'(s) \times P''(s)$

  $v=w \times u$



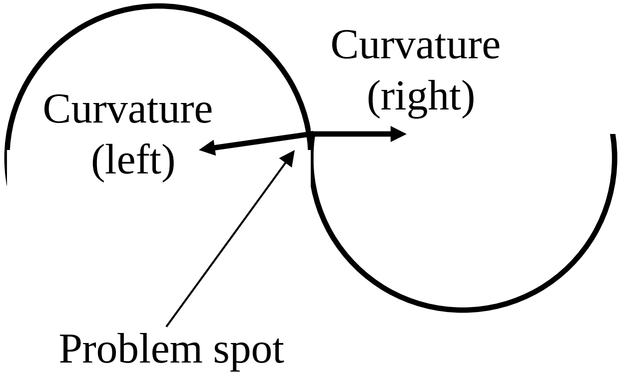
Following a path: Frenet Frame

- Frenet frames are quite nice, but bear some flaws
- When the curve has no curvature, its second order derivative is zero. Here the Frenet frame is undefined
  - This problem can be solved by interpolating the Frenet frames at the start and end of the rectilineal trait
  - Since the tangent vector must be the same at the extremities, it is only a rotation that has to be interpolated



Following a path: Frenet Frame

- A more complicated problem occurs at discontinuities in the curvature vector
- For example, when the path follows first a circle, and then a second circle
- At the problem point, the curvature will switch to pointing from one circle center to the other one
- Here, the Frenet frame is defined everywhere but is discontinuous
- Here, the object will rotate wildly along the path with „instant switches“



Following a path: Frenet Frame

- The worst problem is that the path following is not so natural:
  - when we view at something, we we do not look along the tangent
  - When we move, we anticipate curves
- Similar effect to your car light not following the road
- Also, one might want to make the object bend towards the interior to „anticipate the force"
- .... or, opposite, to let it bend out to give the effect of a force acting on the object

Camera path following: Center of Interest

- A more natural way of specifying the orientation of a camera is to use the center of interest (COI)
  - One can view towards a fixed point
  - Or alternatively the center of an object
- Good method for a camera circling some arena of action
- The center of interest is specified, and so the view vector w=COI-POS
- This leaves one degree of freedom in camera specification
- One simple way is to set the view vector v as viewing „up“, i.e. perpendicular to w and lying in the *wy* plane
  w=COI-POS
  $u = w \times y$
  $v = u \times w$
- This works quite well for a camera moving along a path and focussing to a single object.
- When it gets very close to the object, this results in drastic changes (fly-near effect)
- This is not always bad!!!

Camera path following: Center of Interest

- There are variations to specifying a fixed point
- One can for example specify various points on the camera path itself
- The up vector
  - is usually specified as lying in the *wy* plane
- But one can also allow the user to input
  - Either a tilting value with respect to the default up vector
  - Or the up vector on a whole

- Following a points on the path is relatively easy:
  - If P(s) describes the position on the curve, then P(s+δs), with δs >0, specifies its position in the future
  - It is advisable to choose points at equidistances on the curve, so as to make changes not that noticeable
  - Alternatively, one can take the baricenter of some future points to avoid too much hopping
- The real flaw of this method is the fact that camera views look jerky

Camera path following: Center of Interest

- A better method is to use instead of some function of the position path, a different function altogether for the POI
- Let P(s) be the curve of the camera path, and C(s) the curve of the COI (obviously the animator specifies this)
- Similarly, and up vector path must be specified U(s), so that the general up direction is U(s)-P(s)
- The resulting coordinates for the camera will then become

  $w=C(s)-P(s)$
  $u=w \times (U(s)-P(s))$
  $v=u \times w$

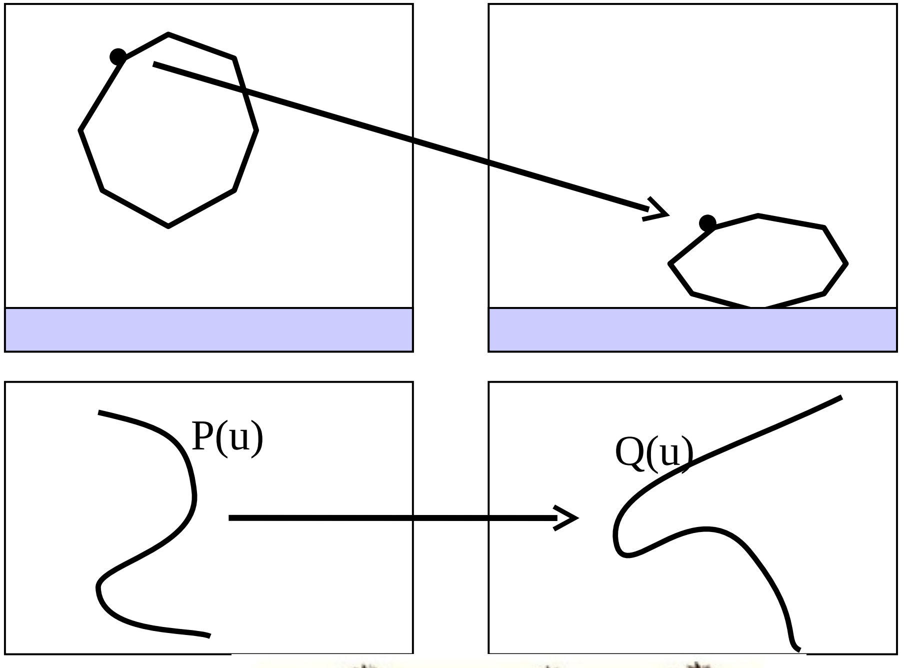
- This gives maximum control, but is also difficult to control.
- An easy way of specifying C(s) is to use fixed positions, with ease-in/ease-out moves between the different fixed points

Path along a surface

- If an object needs to follow a surface when it moves, then a path on the surface itself has to be found
- If we know start and endpoints, then this is simple:
  - trace a plane „perpendicular“ to the surface
  - Compute the intersection plane-surface
- Alternatively, other methods can be used, for example if one wants to follow the „valleys“ on the surface
- Here „greedy“ methods can be used, or methods that compute the normal to the surface and follow it

Keyframe Interpolation

- Objects and topic events are usually set by the animators: these are called *keyframes*
- The computer interpolates between the keyframes to compute the whole movement along time
- Interpolation is done onto any parameter, like:
  - object positions,
  - Control points of curves
  - Colour
  - Normals
- Intepolation is either linear or higher order
- Interpolation is easy if the defining parameters are the same number



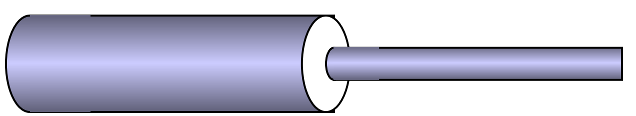
Hierarchical models: articulated figures

- Hierarchical modeling is placing constraints on objects organized in a tree like structure
- Examples can be:
  - A planet system
  - A robot arm
- The latter is quite common in graphics: it is constituted by objects connected end to end to form a multibody jointed chain
- These are called *articulated figures*
- They stem from robotics

- Robotics literature speaks with a different terminology:
  - Manipulator: the sequence of objects connected by joints
  - Links: the rigid objects making the chain
  - Effector: the free end of the chain
  - Frame: local coordinate system associated to each link

Hierarchical Modeling

- In graphics, most of the links are revolute joints: here one link rotates around a fixed point of the other link
- The other interesting joint for graphics is the prismatic joint, where one link translates relative to the other



- Joints restrain the degree of freedom (DOF) of the links
- Joints with more than one degree of freedom are called *complex*
- Typically, when a joint has n>1 DOF it is modeled as a set of n one degree of freedom joints

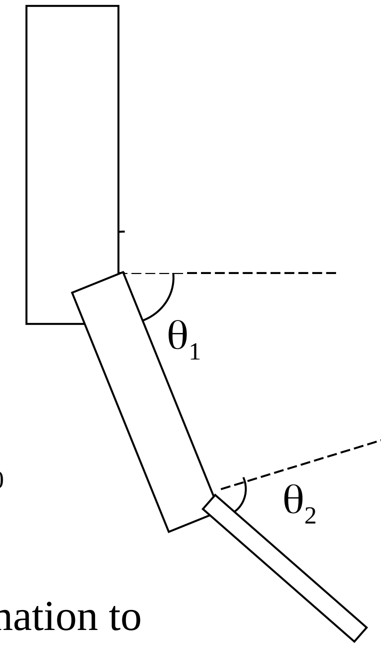
Hierarchical Modeling

- Humans and animals can be modeled as hierarchical linkages
- These are represented as a tree structure of nodes connected by arcs
- The highest node of this structure is called the root node, and is the node that has position WRT the global coordinate system
- All other nodes have their position only as relative to the root node
- A node that has no child is called a leaf node
- Each node contains the info necessary to define the position of the corresponding part
- Two types of transformations are associated with an arc leading to a node:
  - Rotation and translation of the object to its position of attachment to the father link
  - Information responsible for the joint articulation

Hierarchical Modeling

- How does this work?
- The idea is simple, store at each node
  - Info on the node geometry
  - The transformation (its rotation) with respect to the father node in the tree
- To obtain the position of the i-th node in the chain, one has to simply multiply the transformations to obtain the position of the current arc to be displayed
- The root node of course contains info of its absolute position and orientation in the global coord. system

T_0: transformation to rotate K_0 in WCS



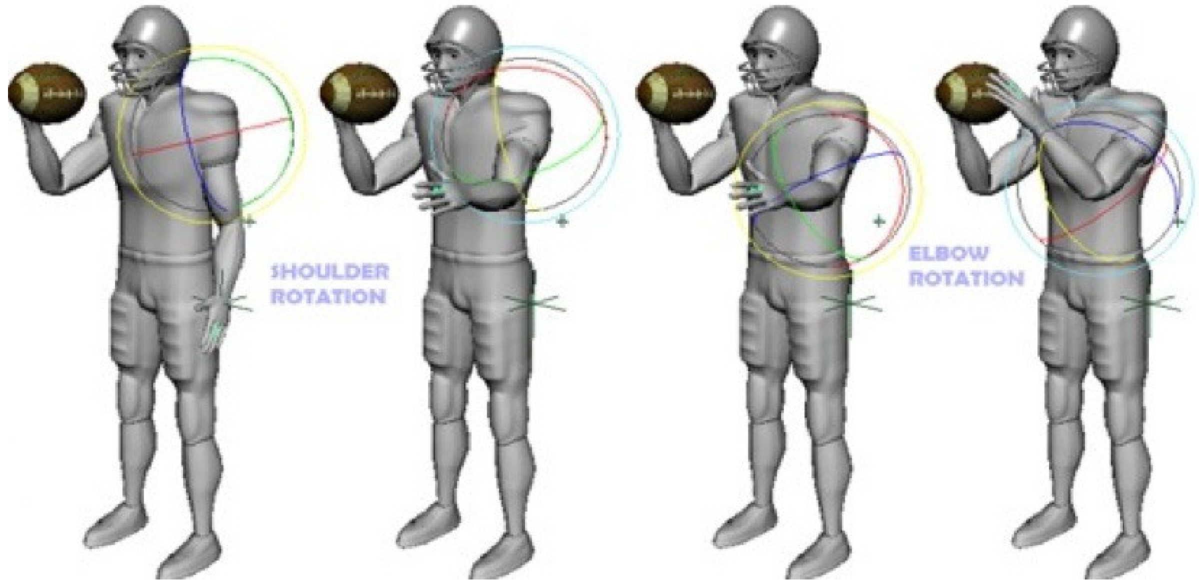
T_1: transformation to rotate K_1 WRT K_0 = rotation by θ_1

T_2: transformation to rotate K_2 WRT K_1 = rotation by θ_2

- To obtain the position of K_2 in WCS, one will then have to multiply $T_0T_1T_2$

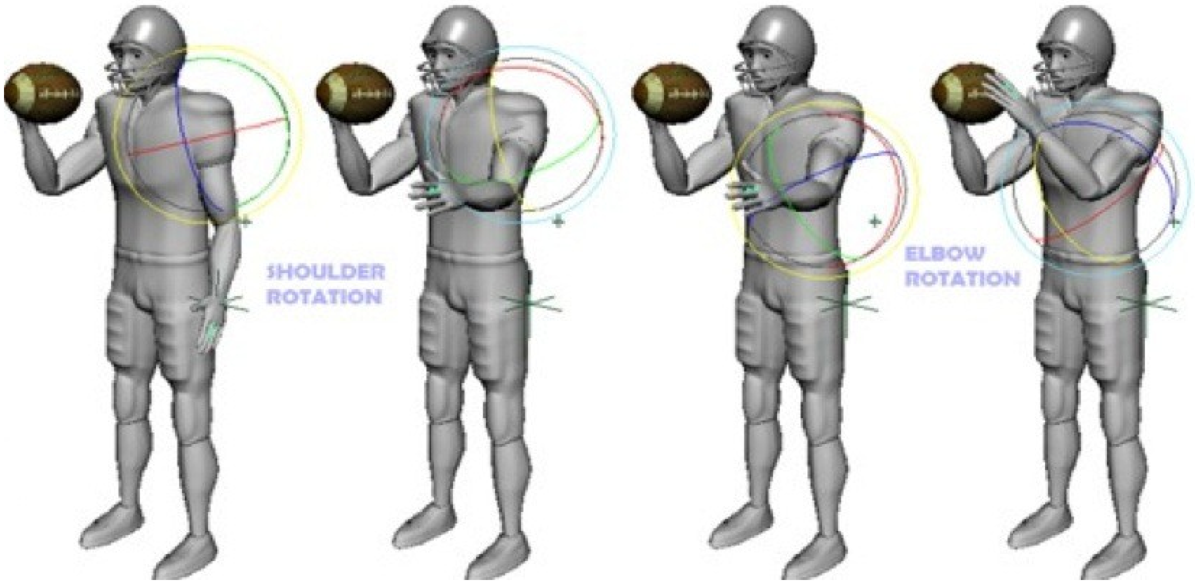
Forward Kinematics

- Traversing the tree of the nodes produces the correct picture of the object
- Traversal is done depth first until a leaf is met
- Once the corresponding arc is evaluated, the tree is backtracked up until the first unexplored node is met
- This is repeated until there are no nodes left inexplored
- A stack of transforms is kept
- When tree is traversed down-wards, the corresponding trans-formation is added to the stack
- Moving up pops the transformation from the stack
- Current node position is generated through multiplying the current stack transforms



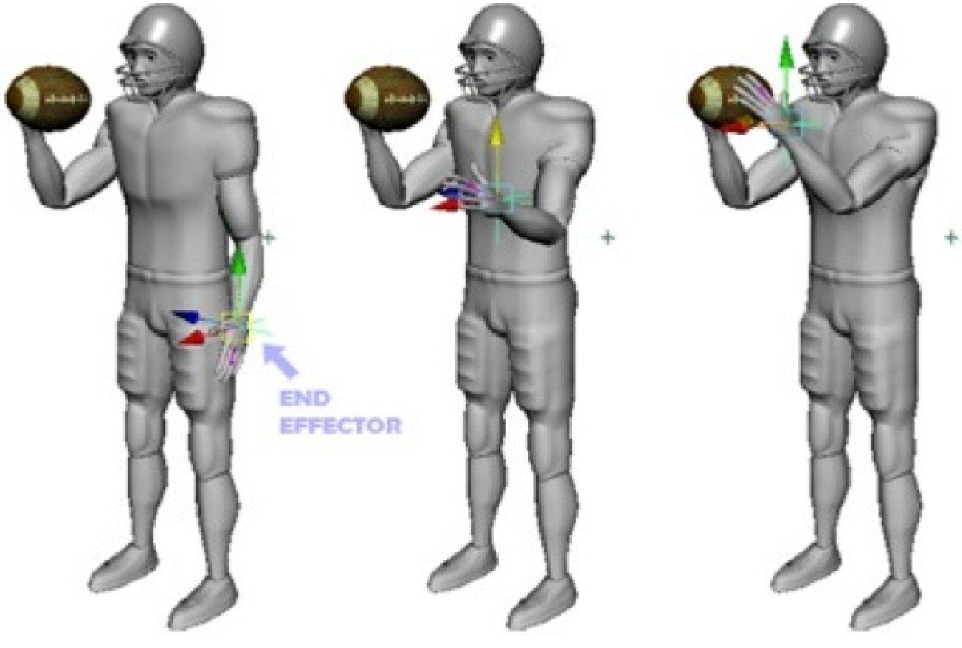
Forward Kinematics

- To animate the whole, the rotation parameters are manipulated and the corresponding transforms are actualized
- A complete set of rotations on the whole arcs is called a *pose*
- A pose is obviously a vector of rotations
- Moving an object by positioning all its single arcs manually is called forward kinematics
- This is not so user-friendly



Inverse Kinematics

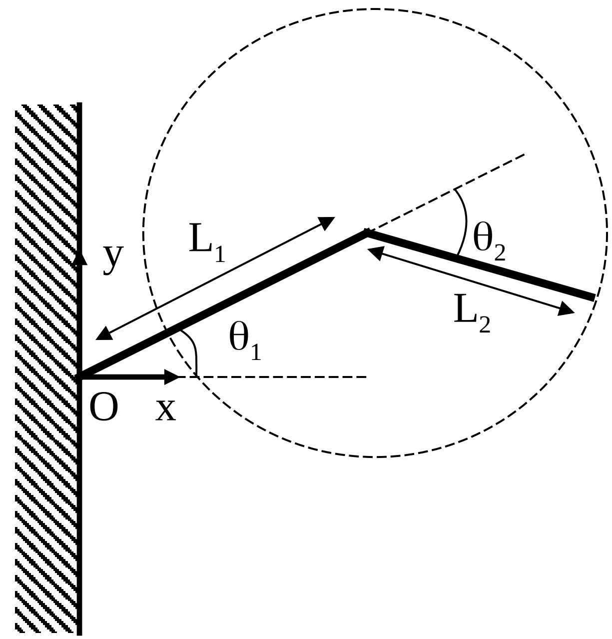
- Instead of specifying the whole links, the animator might want to specify the end position of the effector (inverse kinematics)
- The computer computes then the position of the other links and their mutual angles



- One can have zero, one or multiple solutions
  - No solution: overconstrained problem
  - Multiple solutions: underconstrained problem
  - Reachable workspace: volume that end effector can reach
  - Dextrous workspace: volume that end effector can reach in any orientation
- Computing the solution to the problem can at times be tricky

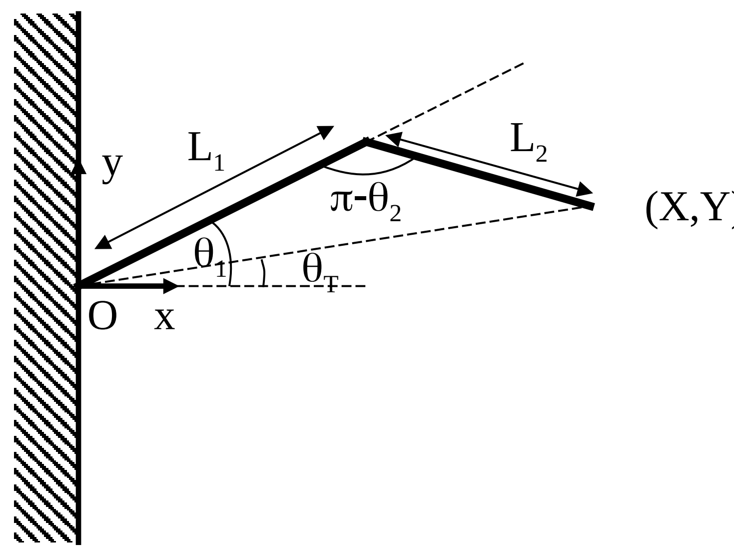
Inverse Kinematics

- If the mechanism is simple enough, then the solution can be computed analytically
- Given an initial and a final pose vector, the solution can be computed by interpolating the values of the pose vector
- Consider the figure: the 2nd arm rotates aroound the end of the 1st arm.
- It is clear that all positions between $|L_1-L_2|$ and $|L_1+L_2|$ can be reached by the arm.
- Set the origin like in the drawing
- In inverse kinematics, the user gives the (X,Y) position of the end effector
- Obviously there are only solutions if
  $|L_1-L_2| \leq \sqrt{X^2+Y^2} \leq |L_1+L_2|$



Inverse Kinematics

- $\cos\theta_T = X/(X^2+Y^2)^{½}$
  $\Rightarrow \theta_T = \mathrm{acos}(X/(X^2+Y^2)^{½})$
- Because of the cosine rule we have also that
  $\cos(\theta_1-\theta_T) = (L_1^2+X^2+Y^2-L_2^2)/2L_1\sqrt{X^2+Y^2}$
  and
  $\cos(\pi-\theta_2) = (L_1^2+L_2^2-(X^2+Y^2)^{½})/2L_1L_2$
  from which we have
  $\theta_1 = \mathrm{acos}((L_1^2+X^2+Y^2-L_2^2)/2L_1(X^2+Y^2)^{½} + \theta_T$
  and
  $\theta_2 = \mathrm{acos}((L_1^2+L_2^2-(X^2+Y^2))/2L_1L_2)$
- Note that two solutions are possible, simmetric with respect to the line joining the origin and (X,Y)



Inverse Kinematics

- In general, for the quite simple armatures used in robotics it is possible to implement such analytic solutions
- Unfortunately this works only for simple cases
- For more complicated armatures, the number of possible solutions there may be infinite solutions for a given effector location, and computations become so difficult to do that iterative numeric solution must be used

Jacobians

- Suppose you have
  - six independent variables and
  - six unknowns that are functions of these variables

$y_1=f_1(x_1,x_2,x_3,x_4,x_5,x_6)$
$y_2=f_2(x_1,x_2,x_3,x_4,x_5,x_6)$
$y_3=f_3(x_1,x_2,x_3,x_4,x_5,x_6)$
$y_4=f_4(x_1,x_2,x_3,x_4,x_5,x_6)$
$y_5=f_5(x_1,x_2,x_3,x_4,x_5,x_6)$
$y_6=f_6(x_1,x_2,x_3,x_4,x_5,x_6)$

$\underline{Y}=F(\underline{X})$

- When the solution is not analytically computable, incremental methods converging to the solution are used
- To do this, the matrix of the partial derivatives has to be computed
- This is called the *Jacobian*

$$J=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

- The Jacobian can be seen as a mapping of the velocities of $\underline{X}$ to velocities of $\underline{Y}$.
- In other words, how changes of the X variables map into effector changes.

Using Jacobians

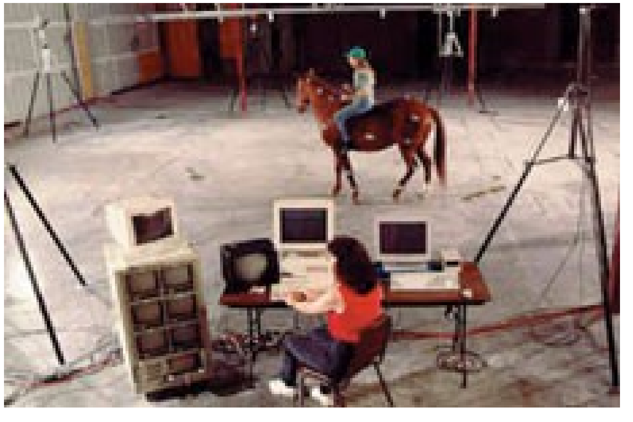
- The Jacobian matrix is a linear function of the x_i variables
- When time moves on to the next instant, X has changed and so has the Jacobian

- The desired change will be based on the difference between the current position/orientation to the desired goal configuration
- If one can invert this equation, we can compute from the Y positions the necessary X positions

- Of course the math is not easy
- Finding the real solution will involve writing the Taylor series of the original equations, which is beyond the scope of this course.

Motion tracking

- Making synthetic movement of „real characters“ is complicated.
- Recently, devices appeared that are capable of capturing real movement and applying it to virtual characters.
- This is called motion capture:
  - The idea is to use either sensor positioning, or capture images and identify the marker positions
  - Real humans/animals are therefore equipped with sensors (or markers) applied to the different body parts
  - The xyz positions of these markers in time are recorded while the „actor“ is performing movement
  - More recent equipment (e.g. Kinect) do this without markers (using IR+SW)

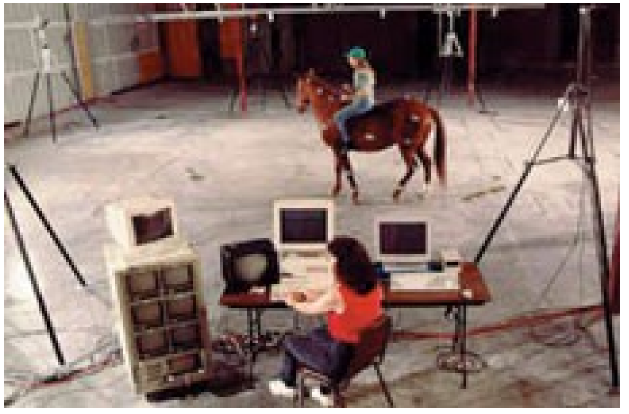


Motion tracking

- There are basically two ways of doing motion tracking:
- Electromagnetic sensors:
  - uses sensors positioned at the joints that transmit their position and orientation
  - Transmission is done either by cable (= limit freedom of movement) or by wireless
  - De facto real time
  - Main problem: room must be free of field distortions
  - Limited range, accuracy problems
  - High purchase cost

Motion tracking: optical

- Optical tracking:
  - Uses video cams to record motion of the subject
  - Easier to wear (reflective markers are applied to subject)
  - Wider range
  - No cables
  - Real time difficult
  - Data is noisy and error prone
  - Because orientation is not directly generated, more markers are required than with magnetic trackers
  - Cameras may vary in quality and principle:
    - Infrared
    - Very high resolution
    - But also available for consumer videocams => cheap!
  - In the next, we will take a look at how optical tracking works



Motion tracking: optical

- Objective is to reconstruct the three-dimensional model of a motion and apply it to a synthetic model
- Work can be subdivided in 3 tasks:
  - Image processing: Images need to be processed so as to be able to locate, identify and correlate the markers
  - Camera calibration: 3D locations of markers have to be extracted from the 2D images
  - Constraint satisfaction: The 3D marker locations have to be constrained to the physical model whose motion is being captured

Optical tracking: Image Processing

- Optical markers can be of different shapes: pingpong balls, other markers...
- Stuck to the joints with velcro/tape
- One of the problem is that they stick out of the body, so there is a difference between where they are and where the real joints are
- Moreover, they can moveWRT the real joint too

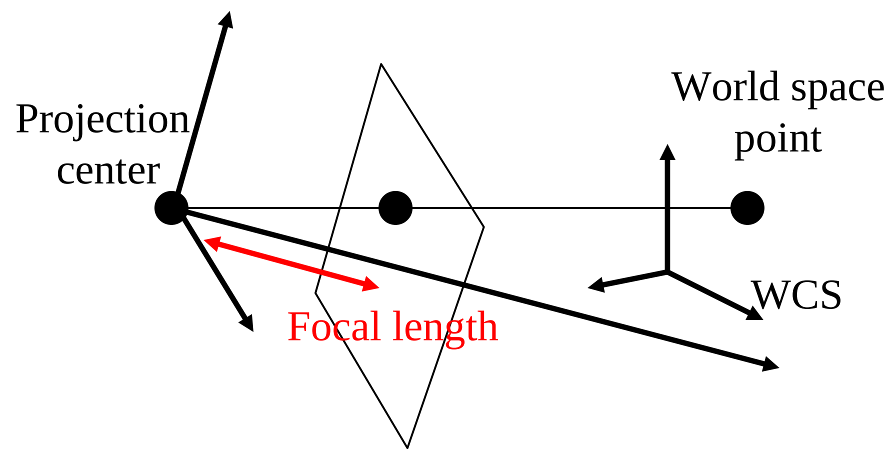
- Once video digitized, it can be analyzed
- If background static, it can be subtracted
- Once this is done, the marker gets searched for
  - Of course, with more markers it is more complicated, because they may get occluded
  - Therefore one has to track the markers across the frames

Optical tracking: Image Processing

- Tracking trackers across the frames is also difficult
- One can use frame coherence, which works as long as the subject moves slowly enough
- One can also use logical coherence, i.e. when walking feet are always at the floor
- One can use also prediction methods: if I know how fast the subject is, I can try to „guess“ the whereabouts of the marker in the next frame
- Occlusion is a further problem: if more markers disappear, it is difficult to know which is which when they reappear
- Also, when markers pass near each other, they might be swapped next frame
- This might generate markers swapping positions
- Sometimes, this can be solved by taking a 3D image (with 2 cameras).
- Other times, human intervention is necessary

Optical tracking: Camera Calibration

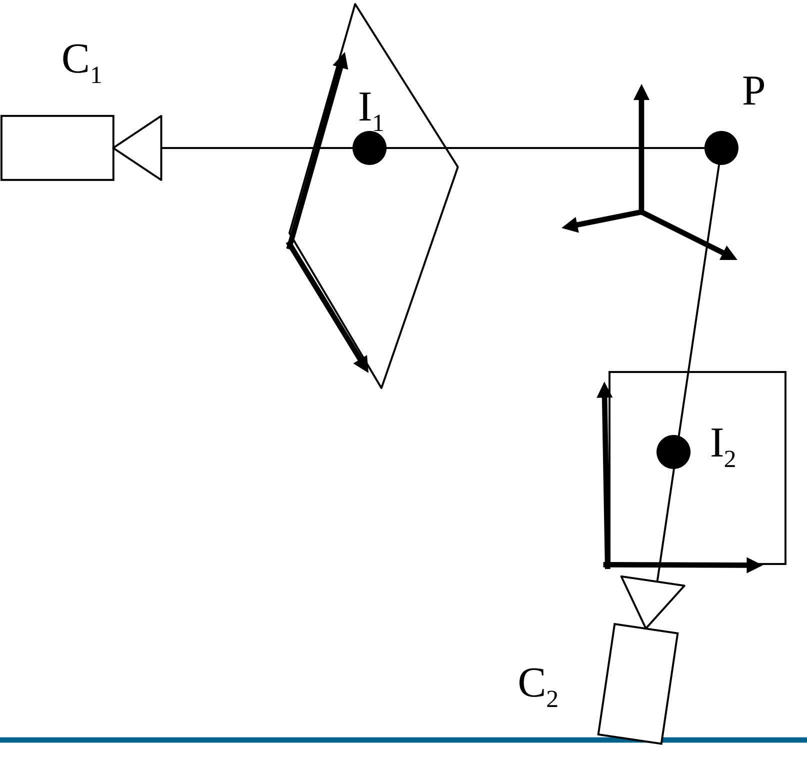
- Before the 3D position of a marker can be reconstructed, one needs to know
  - location and orientation of the cameras in world coords
  - Focal length, image center and aspect ratio have to be known
- The camera system is modelled like in Computer Graphics
- The image of a point is done by projecting a ray from the point to the center of projection
- Calibration is done by recording a number of known points in space



Optical tracking: Position reconstruction

- At least two views are needed to reconstruct 3D
- Since we know I1 and I2, we deduce
  P=C1+k1(I1-C1)
  P=C2+k2(I2-C2),
  thus
  C1+k1(I1-C1)= C2+k2(I2-C2)
  which are 3 equations in 2 variables, and this solvable
- Unfortunately, noise complicates it, because the two straight lines do not necessarily touch

- This can be solved by finding P_1 and P_2 $\perp$ to the lines through the other cameras, and computing the midpoint of the segment P_1P_2



Optical tracking: Position reconstruction

- As few as 14 markers can provide some simple tracking of a human figure
- Complete marking sets include 31 markers, including elbow, kneews, chest, hands, toes, ankles, and spine, as well as scapulae and more...

- The more markers one has, the more it is necessary to have more than 2 cameras, so as not to have marker occlusion
- Each marker at each frame needs to be seen by at least two cameras
- A typical system would have 8 cams
- Multiple cams requre some more effort in synchronizing them

Optical tracking: fitting to skeleton

- The next step is to attach the markers to the skeleton
- One could do it directly, but unfortunately it does not work well, because in general, marker distances are not preserved
- Markers are not exactly on the joints, but on the skin
- One can compensate for that by setting markers at their right positions, but it is still imprecise because the body is elastic
- Another solution is to put two markers on the sides of the joint
- This works well (but doubles complexity), but not for joints which are inaccessible
- Simple geometric calculations lead to deduce the correct joint-marker mutual positions
- Once this is known, the movement can be applied to the skeleton
- Watch out for imprecisions of the data obtained, that can lead to visible artifacts (avoid floor penetration)

Conclusion

- There are loads of other research themes connected with Computer Animation
- This set of slides was simply an appetizer, like these

- In the Masters course, I give a complete Computer Animation lesson in the Summer Term

End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++