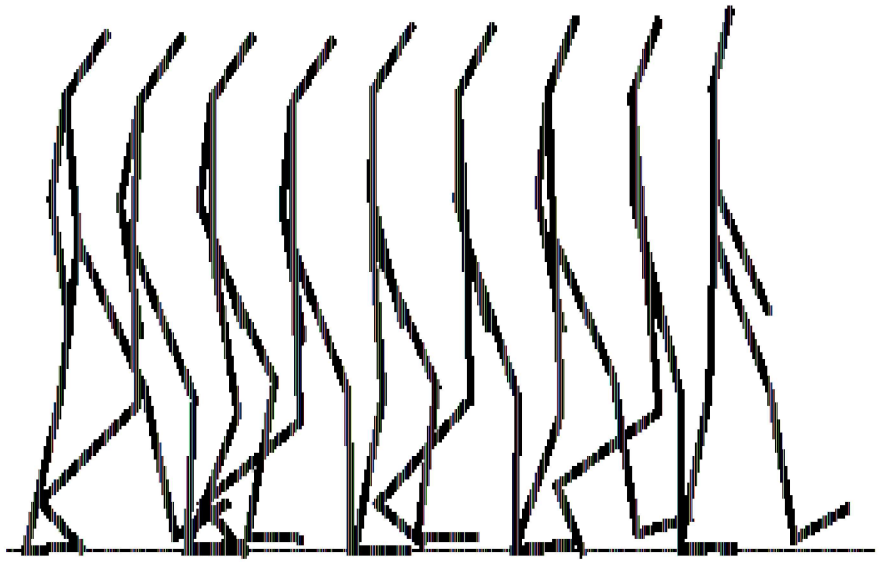
Character Modeling & Animation in Computer Game

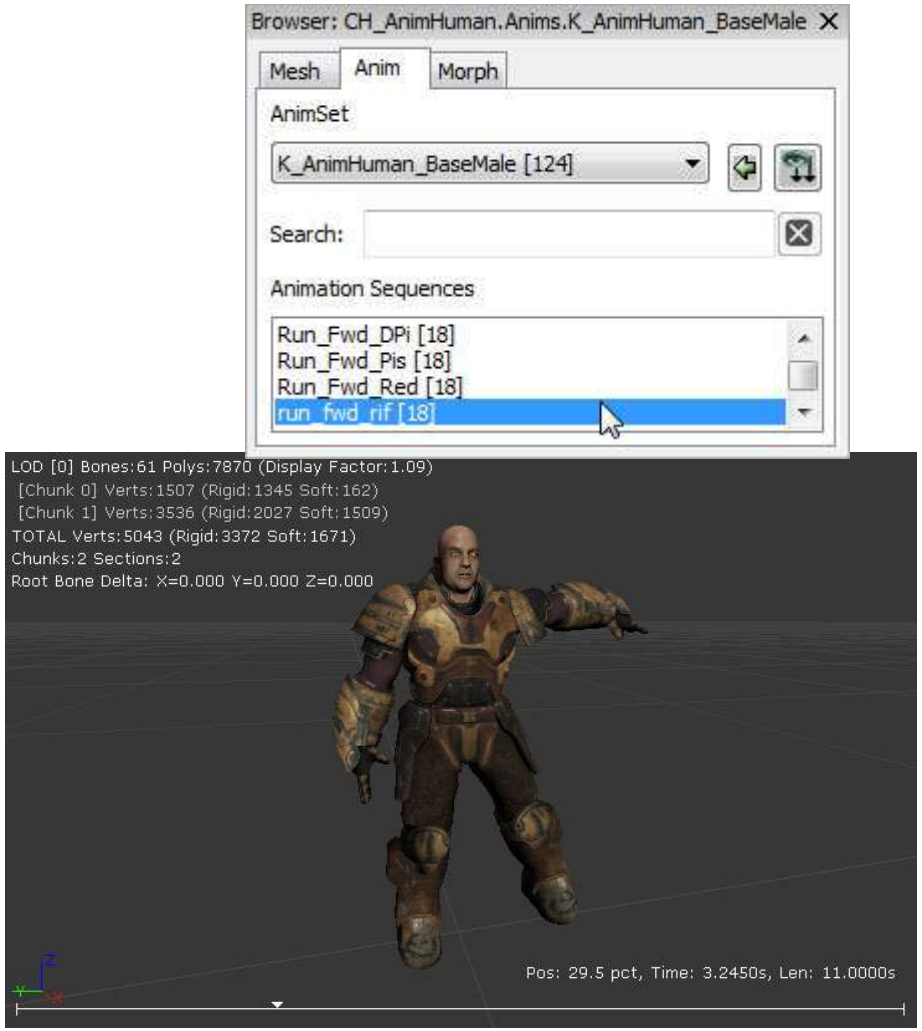
Animation of character

- Important to convince people the character is life-like
- Integral part of game production pipeline:
  1. design and model the character
  2. Produce different animate sequences
  3. Iterates according to designer requirement



In-game Character

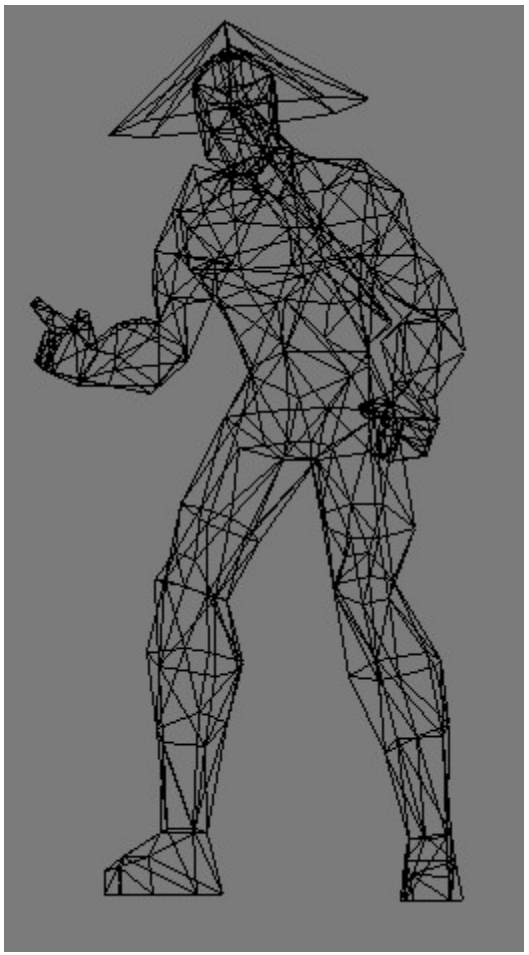
- Consists of 3D model descriptions & various animations
- Game program/scripts choose suitable animations (motions) according to the situation



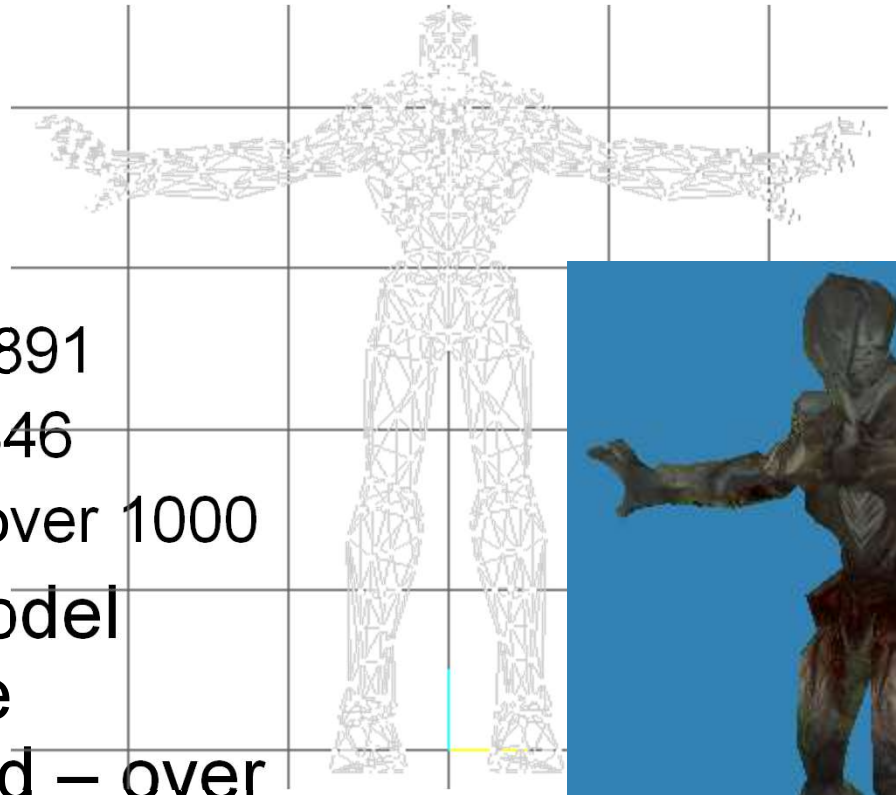
Using animation sequences in UE4 to control skeletal mesh

Character Modeling

- Quake 2 (1996)
  - Vertices: 320
  - Faces: 648
  - Frames: 198
- Hand manipulated both for vertices & polygons



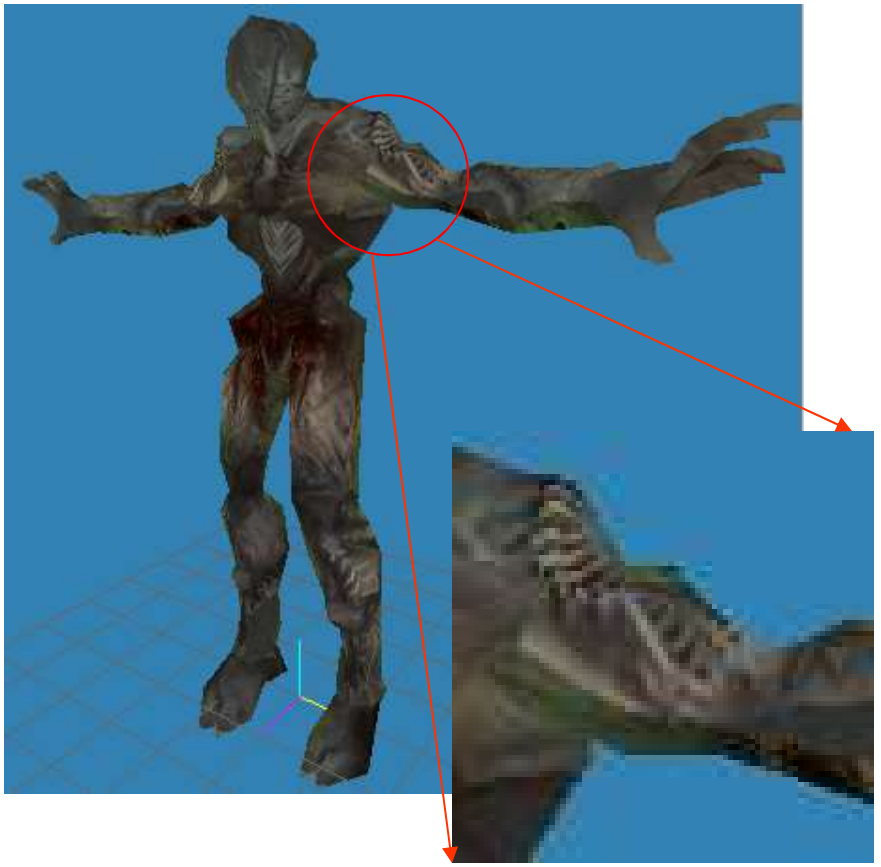
Doom3



- Imp
  - Vertices: 891
  - Faces: 1346
  - Frames: over 1000
- Original model much more complicated – over 100,000 triangles
- Cannot be hand tuned per triangle

Char. Modeling

- Modern game engines can only render limited no. of triangles per character e.g. ~5000
- Even with good artists, the resulting model looks blocky

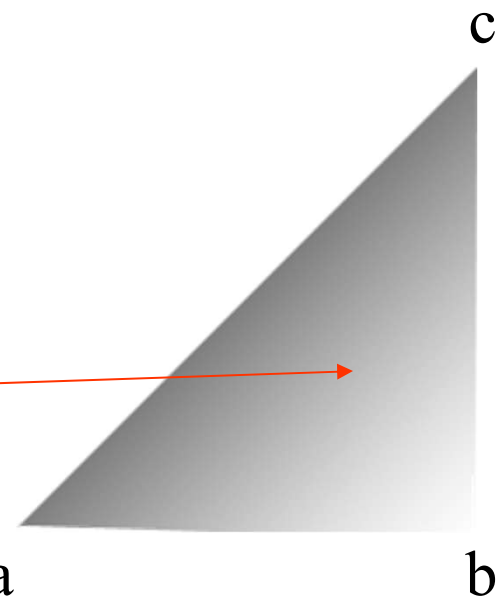


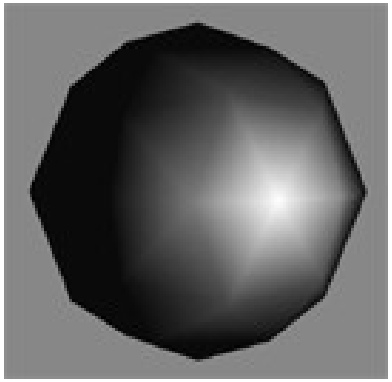
Char. Modeling

- We want a 1.5 Million triangles “look & feel”
- Achieve through bump/normal mapping
- Idea:
  - save the high resolution details on as a texture
  - Map that texture to the model in game

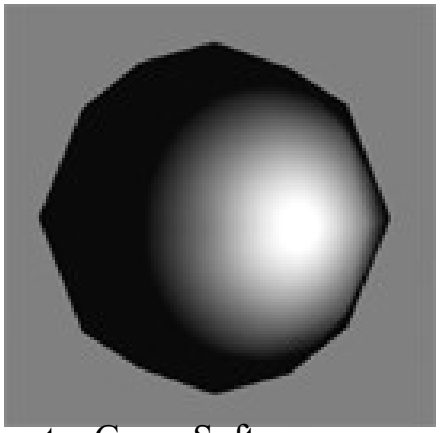
Per-vertex/per-pixel Lighting

- To calculate the lighting inside the triangle, we can only interpolate across the vertices(a,b,c) of triangle
- Will produce incorrect result





per-vertex lighting

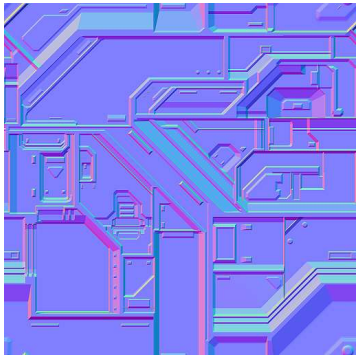


Per-pixel lighting (correct)

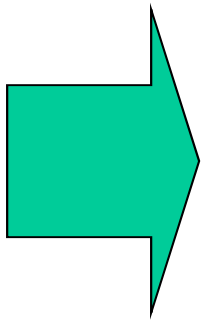
Normal Map

- Use an additional *texture* to model the surface roughness
- Details in form of:
  1. Displacement from surface(**bump map**)
  2. Change of reflection direction of light (**normal map**)

Object colors

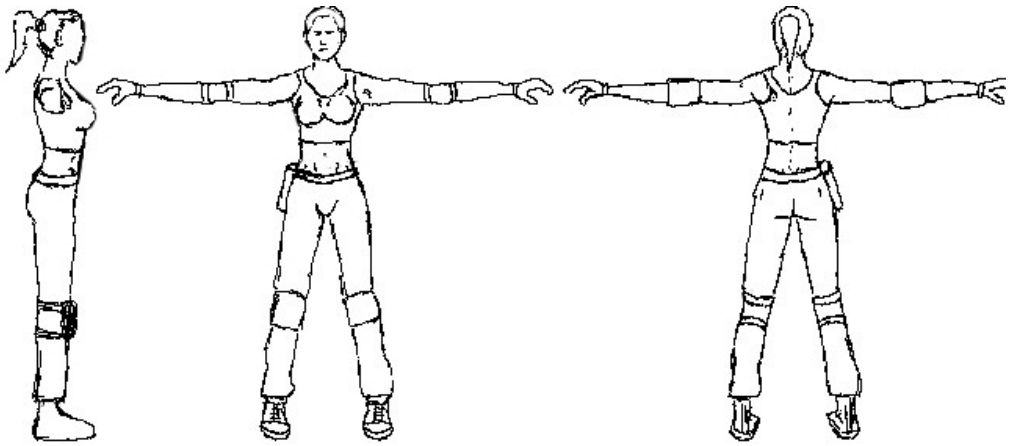


Object details(normal)



Character Modeling

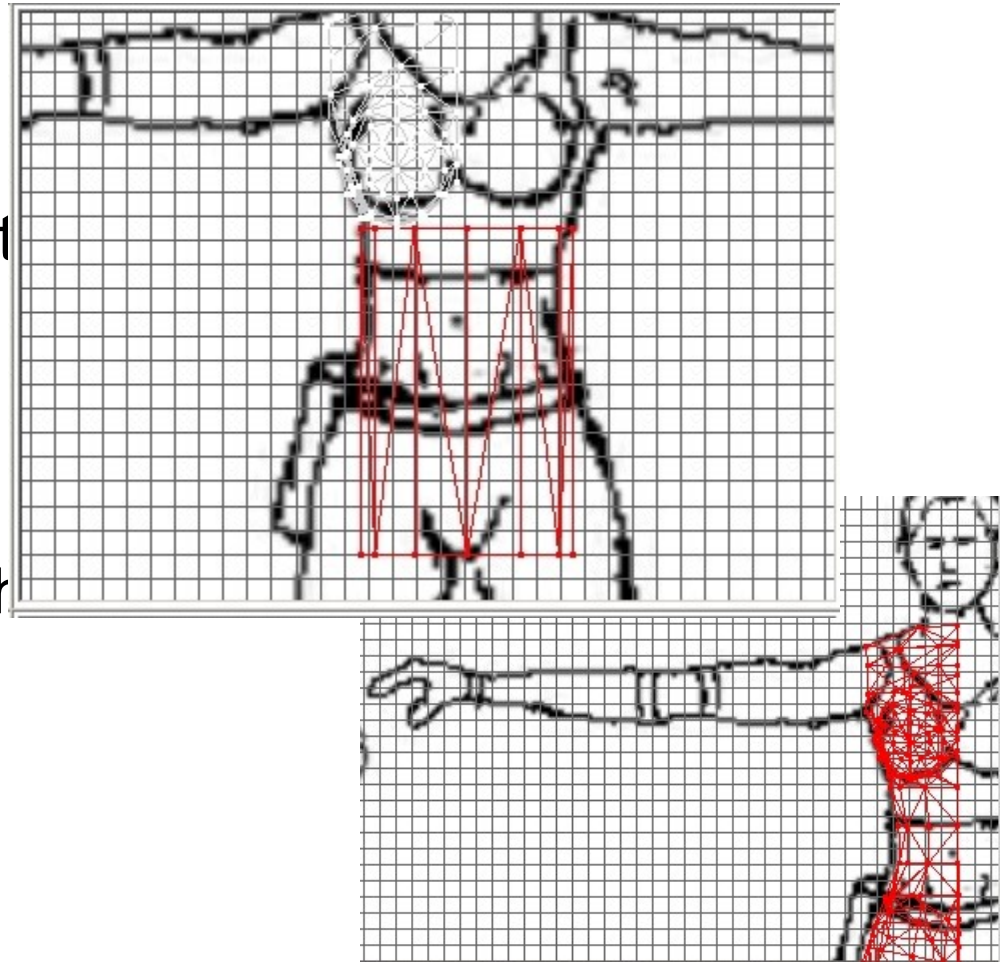
- Procedures
  1. Set up reference pose & arts
  2. Model with triangles
  3. Attach with skeleton
  4. Unwrapping the mesh
  5. Texture painting



A good reference: csgirl tutorial on Blackboard System

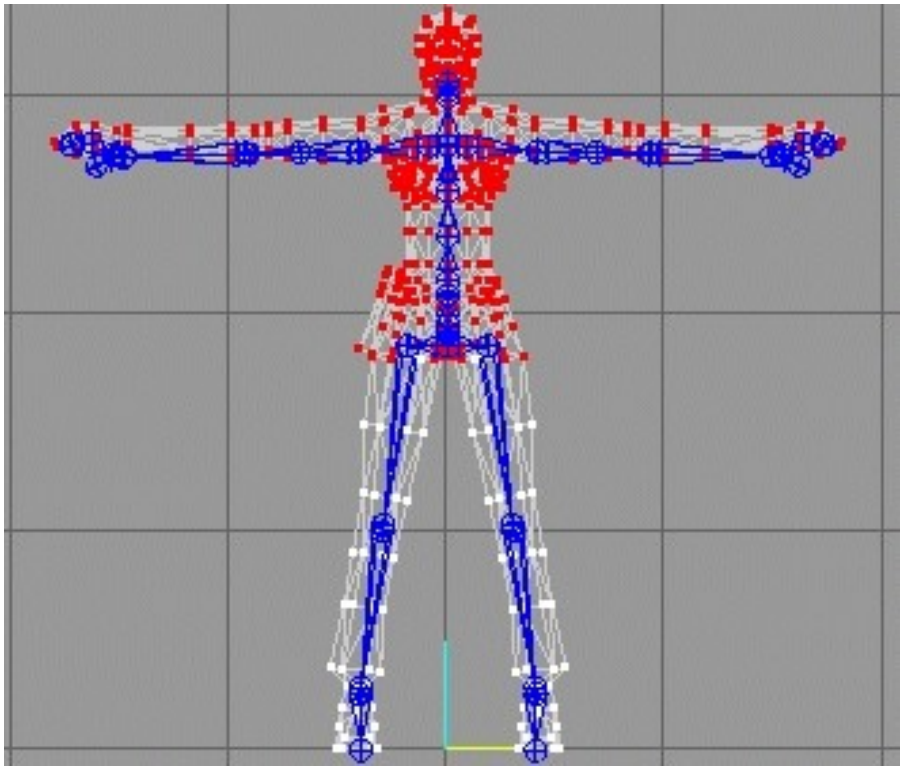
Character Modeling

- Procedures
  1. Set up reference post & arts
  2. Model with triangles
  3. Attach with skeleton
  4. Unwrapping the mesh
  5. Texture painting



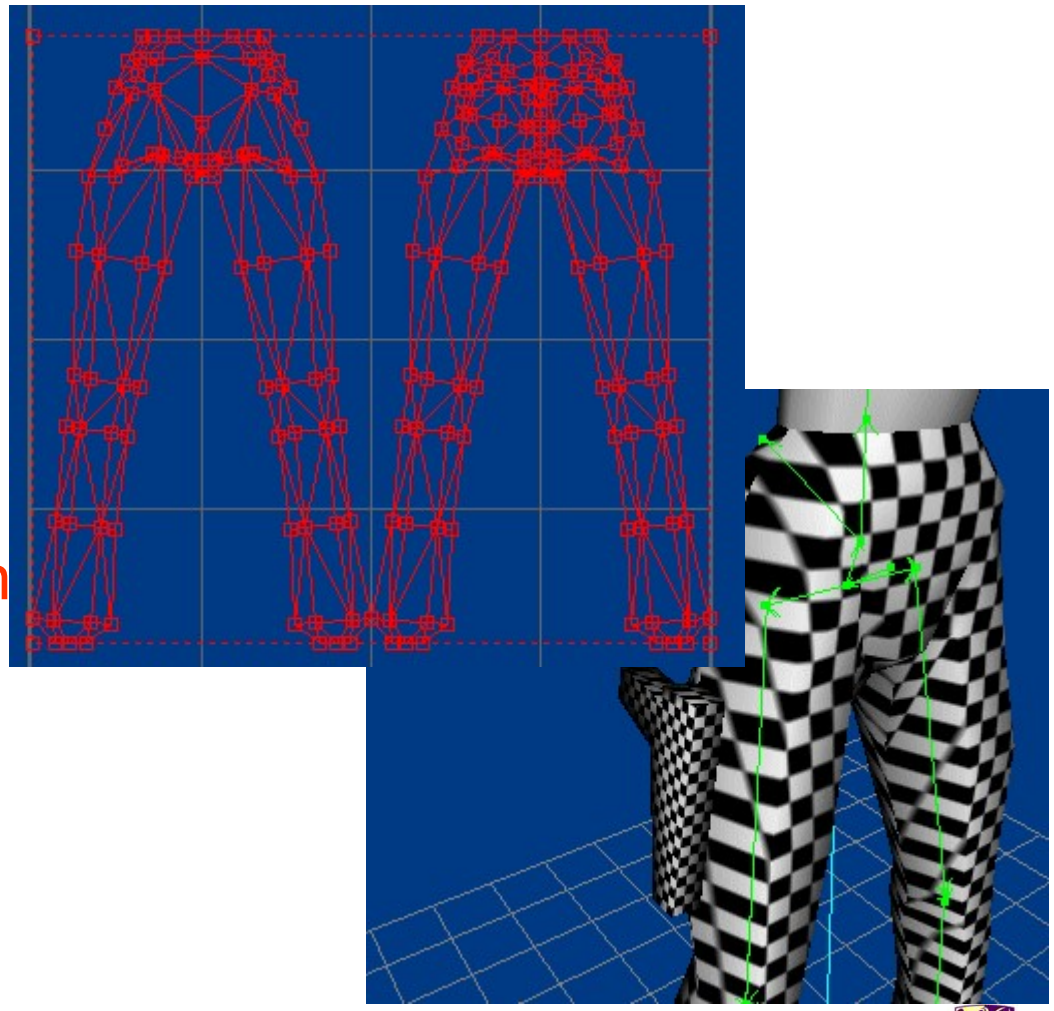
Character Modeling

- Procedures
  1. Set up reference post & arts
  2. Model with triangles
  3. Attach with skeleton(Rigging & Skinning)
  4. Unwrapping the mesh
  5. Texture painting



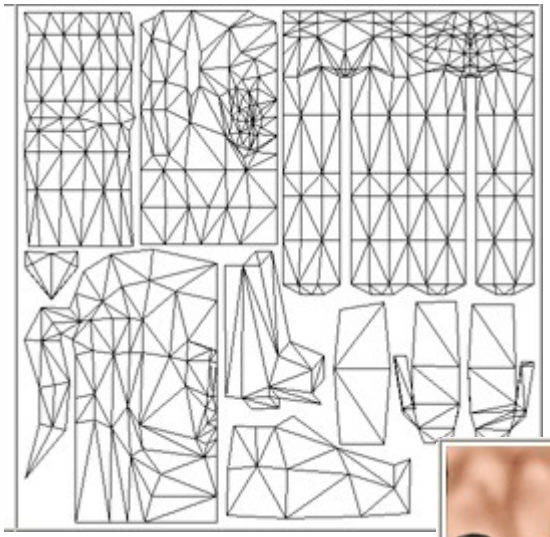
Character Modeling

- Procedures
  1. Set up reference post & arts
  2. Model with triangles
  3. Attach with skeleton
  4. Unwrapping the mesh
  5. Texture painting



Character Modeling

- Procedures
  1. Set up reference post & arts
  2. Model with triangles
  3. Attach with skeleton
  4. Unwrapping the mesh
  5. Texture painting



Character Modeling

Character Animation

- Animate the motion of a character with high realism
- Difficult to perform it manually as degree of freedom is huge e.g. 700 DOFs for character in Toy Story
- use motion capture to pre-record the sequence and playback in game
- Drawback is only contain pre-recorded sequences
- No single solution in all circumstances

Explicit & Implicit solutions

- Explicit (Key frame) methods
  - Store & play all the frames data for each movement

- Advantages

1. Simple implementation
2. Memory intensive
3. Suitable for large number of characters

Explicit & Implicit solutions

- Implicit(skeletal/bone) methods

1. Maintain high level description of motion
2. Compute actual movement data during playback
3. Computation intensive
4. Can adapted to the scenario e.g. walking up a sloped terrain

Explicit

- Key frame animation – storing only the key frames data and interpolate to obtain the full sequence

Interpolated frames

Key frame 1

Key frame 2

Explicit

- Linear interpolation is used typically

In the render loop include

`elapsed time = elapsed time + dt`

where **`dt`** = time used in last frame

for each model vertex at time frame t (V_i^t)

$$V_i^t = (V_i^{finish} - V_i^{start}) \frac{elapsed\_time - t_0}{length\_of\_animation}$$

Vertex position at current & previous key frame

- may produce unrealistic result, may have to use curve interpolator for better result

Explicit

- Consider a rigid transformation

$$M(t) = \begin{bmatrix} a_{11}(t) & a_{12}(t) & a_{13}(t) & t_x(t) \\ a_{21}(t) & a_{22}(t) & a_{23}(t) & t_y(t) \\ a_{31}(t) & a_{32}(t) & a_{33}(t) & t_z(t) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Previous formula will be rewritten into
- $V_i^t = M(t)(V_i^{finish} - {}_i^{start})$ (q1)
- With all a_{ii} set to 0 & $t_i(t)$ as $\frac{elapsed\_time - t_0}{length\_of\_animation}$

- Note : Linear interpolation cannot be applied directly to rotational motion

Explicit

- To incorporate rotation, using (q1) directly to interpolate a_{ii} (not zero) is not valid
  - Rotation matrix (a_{ii} i=1..3) is orthonormal
  - Directly linear interpolate a_{ii} will deform the animated object without achieving angle interpolation (column vector must be unit)

- Need representation which is more friendly to linear interpolation of rotation (cover later)

Explicit method

- Typical file format

Frame1 ← vertex & texture coordinates

frame 2

:

Animation

Stand

{frame1, frame2, ..}

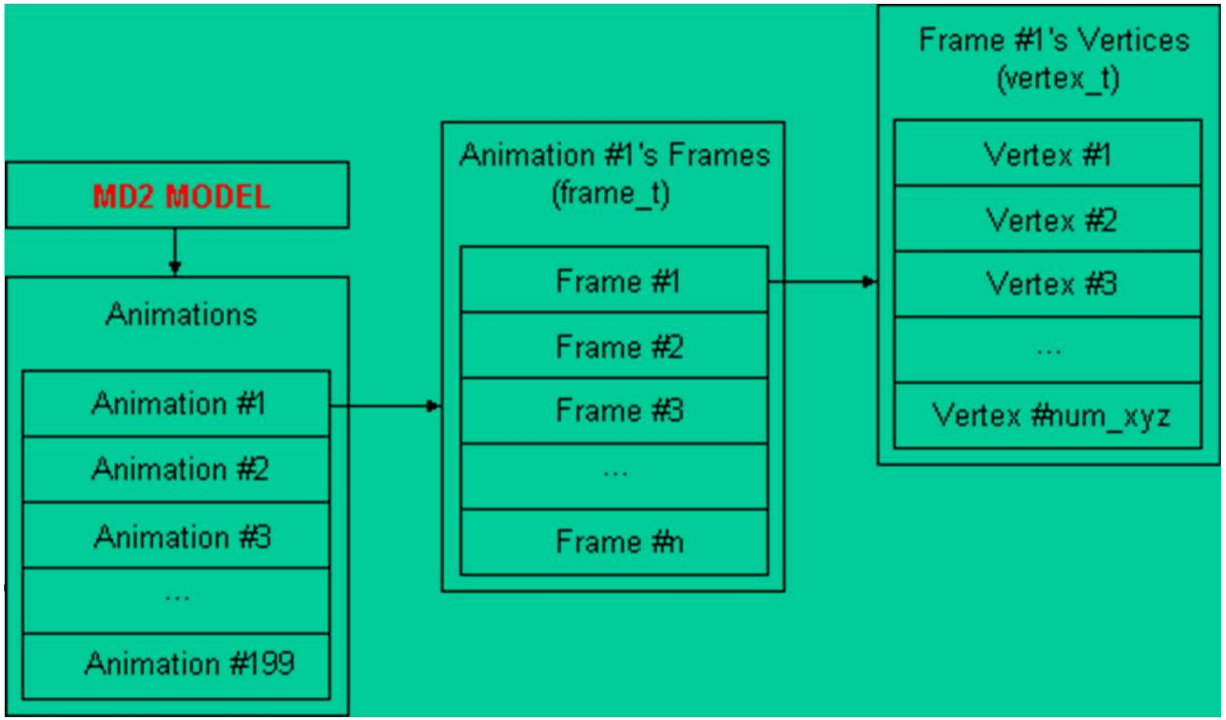
Run

{frame20, frame21, ..

Mesh data

{v1,v2,v3}

.



Tagged Interpolation

- Consider creating a character for an action game, typical 10 actions are: *stand still, walk forward, walk backward, run forward, run backward, jump, crouch, shoot weapon 1, shoot weapon 2, shoot weapon 3*.

- Have problem if we want shoot while moving

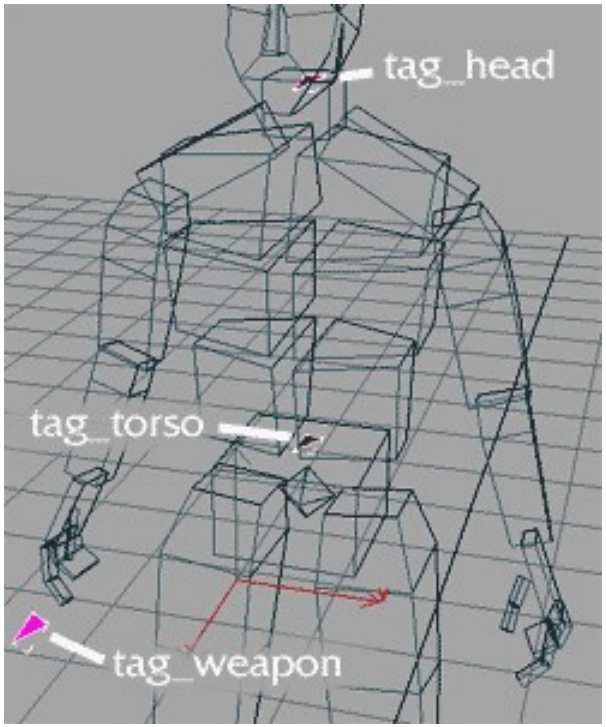
- One solution is to play shoot animation with legs kept still – “player skating” in many old games

Quake 3 Approach

- Divide each character into several body parts e.g. head, torso (neck to belt), leg blocks
- Each block has its own animation e.g.
- *Legs* – stand still, walk forward/backward, run forward/backward, jump, crounch
- *Torso* – stand still, shoot weapon 1/2/3
- *Head* – still, rotate left/right

Tagged Animation

- To animate the character, designer manually specify a pivot point in each body part
- Pivot point called *tags(sockets in UE4)*, are used as a reference
  - Tag_floor – indicate ground level
  - Tag_legs – specify joint between legs & torso
  - Tag_head – binding between torso & head
  - Tag_weapon – can hold interchangeable weapons



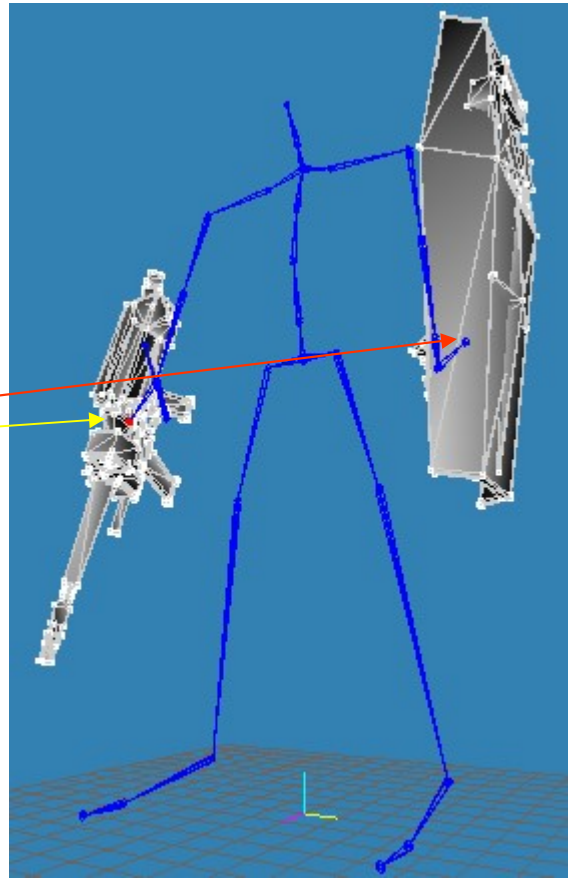
Tagged Animation

- Flexible & easily extend to binding between character & other entities for special effect
- Reduced memory footprint & roughly same CPU cycles
- Cannot provide environment adaptation
- Coordination effort between different body will increase fast with number of body parts

Prop Handling

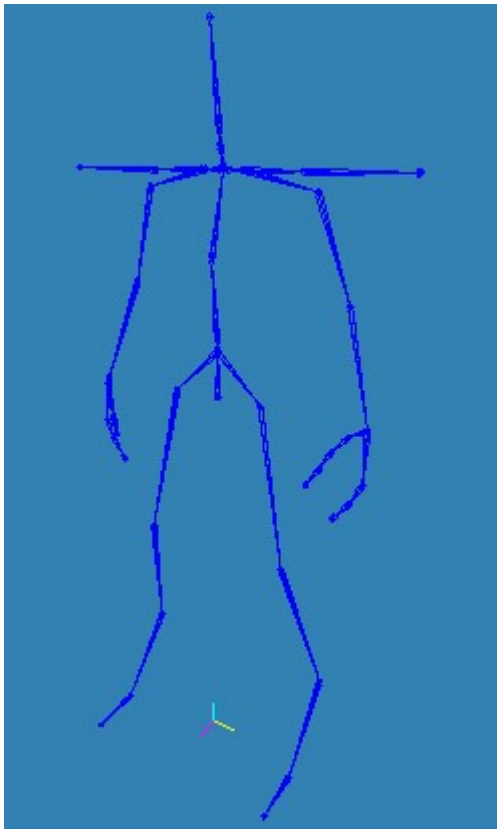
- *Props* – weapons, clothes, vehicles etc. add richness to the game

- Add tags/sockets which store orientation of the prop throughout animation to the character



Implicit Animation

- Here we refer to articulated structure i.e. *human body*
- Stores description of motion and compute actual pose in real time
- Skeletal structure (bones) used



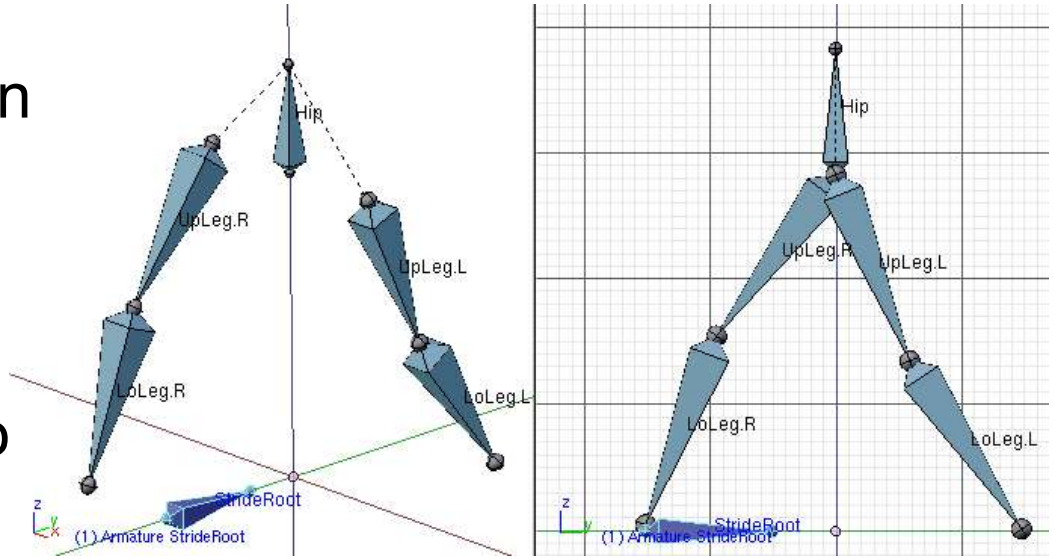
Bones

Implicit Animation

- Connections between bones(joint) must be defined in order for motion propagation

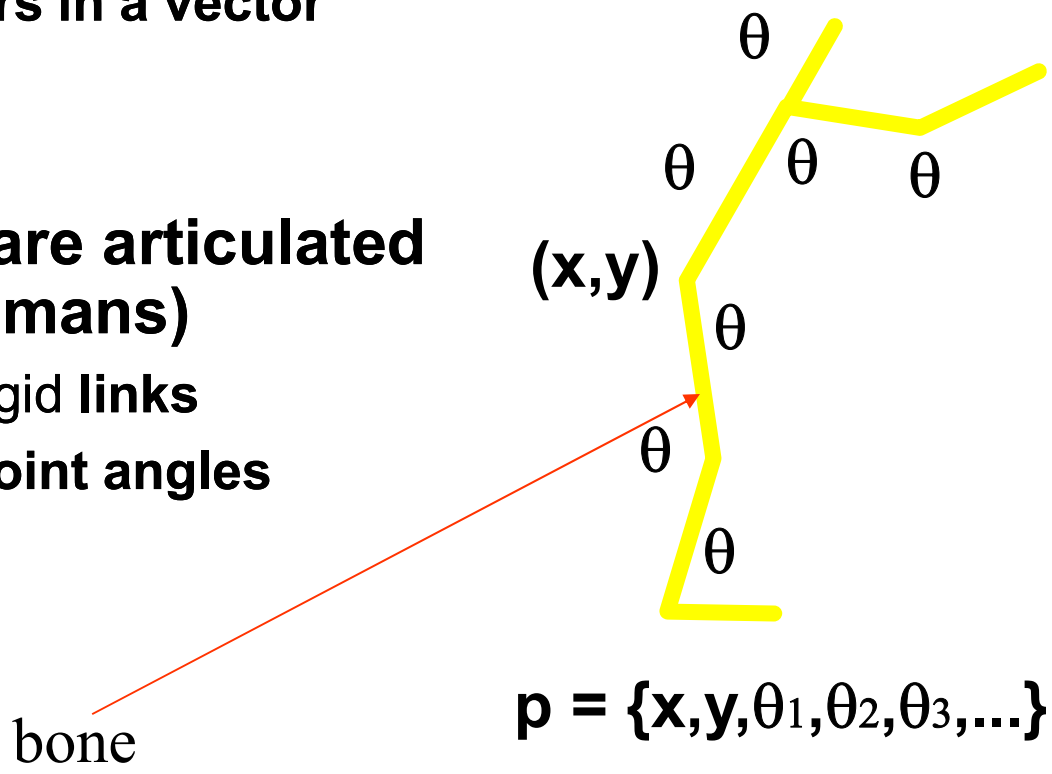
- Motion info. (rotation of joint, translation etc.) also recorded

- Forward/inverse kinematics FK/IK used to do animation



Motion of a character

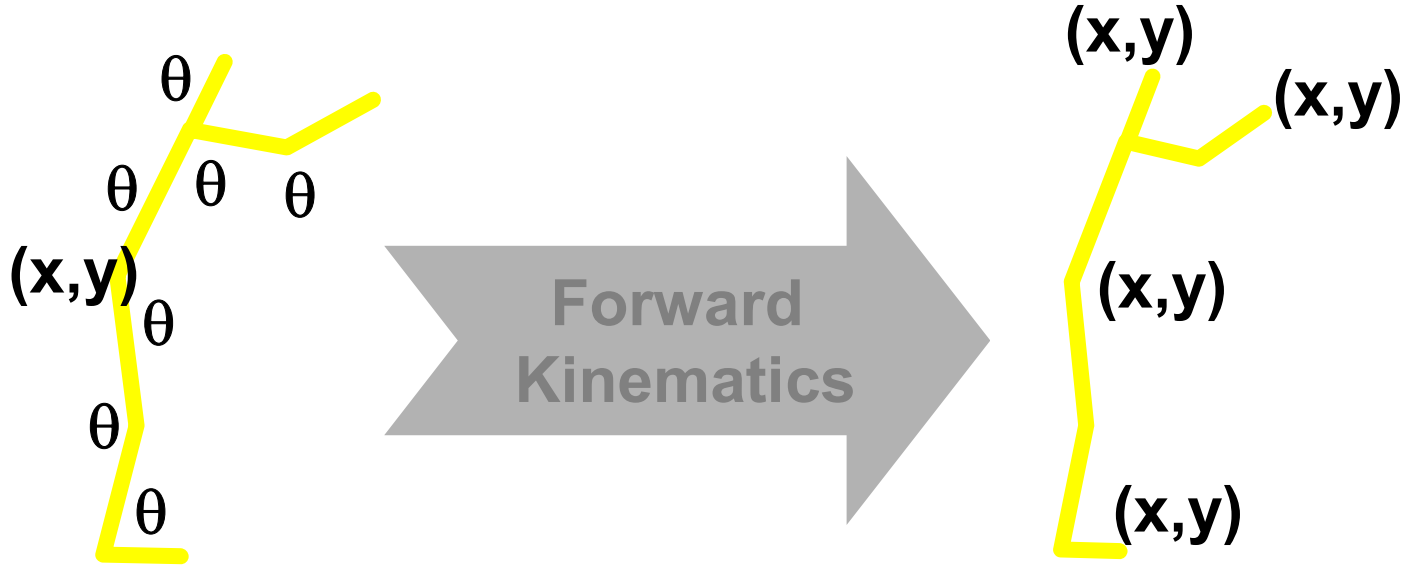
- **Pose or Configuration**
  - **parameters in a vector**
  - $\mathbf{p} \in \Re^n$
- **Examples are articulated figures (humans)**
  - **trees of** rigid **links**
  - **center + joint angles**



p = {x,y,θ1,θ2,θ3,...}

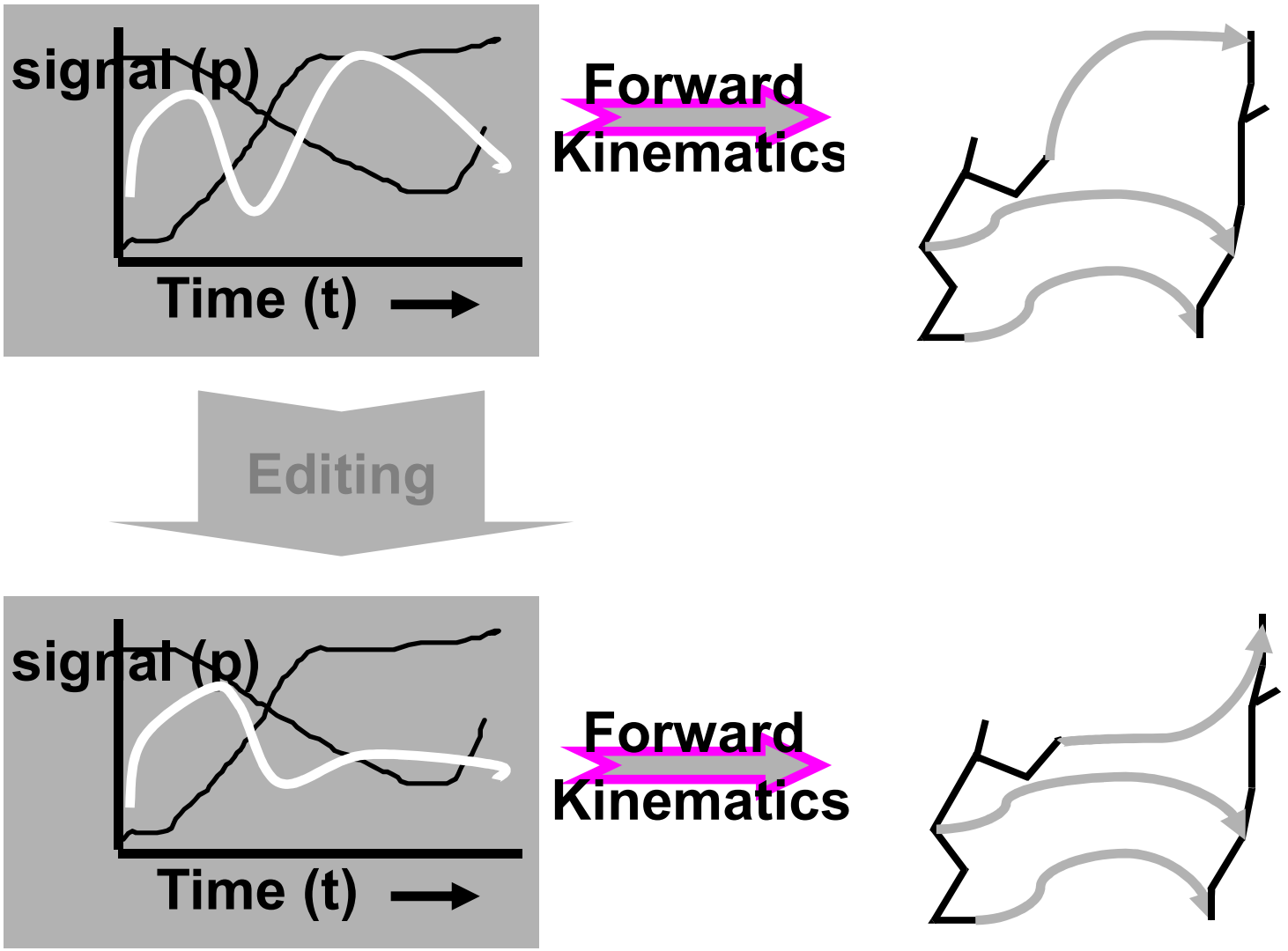
Motions in Parameter Space

- **Motions are values of parameters**



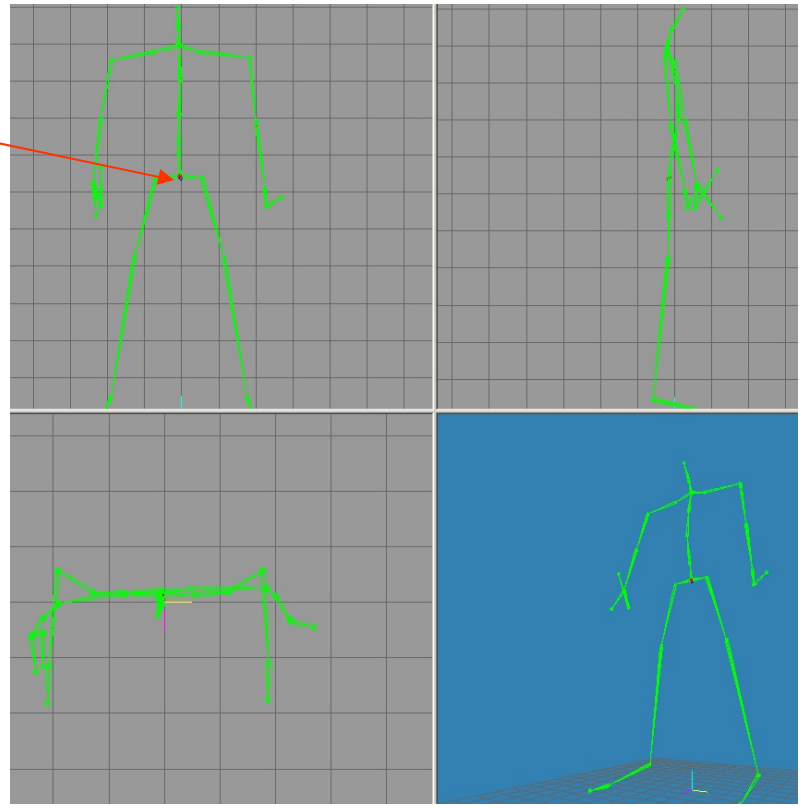
m(t): bones,orientation

x(t), y(t): vertex coordinates


signal (p)
Time (t)
Forward
Kinematics
Editing
signal (p)
Time (t)
Forward
Kinematics

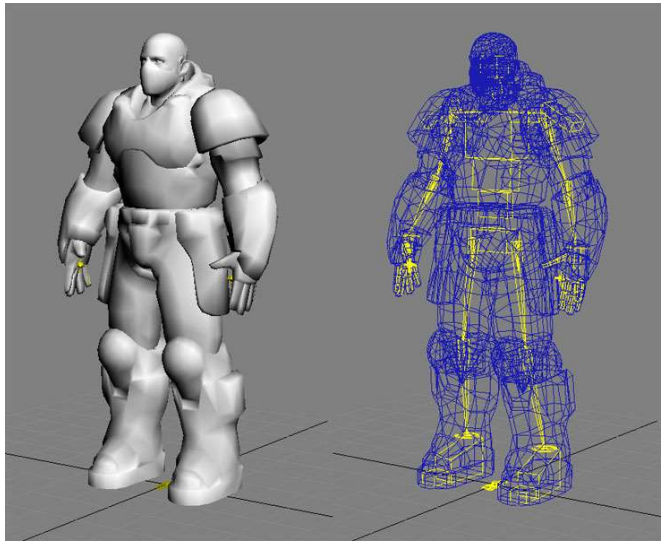
Forward Kinematics

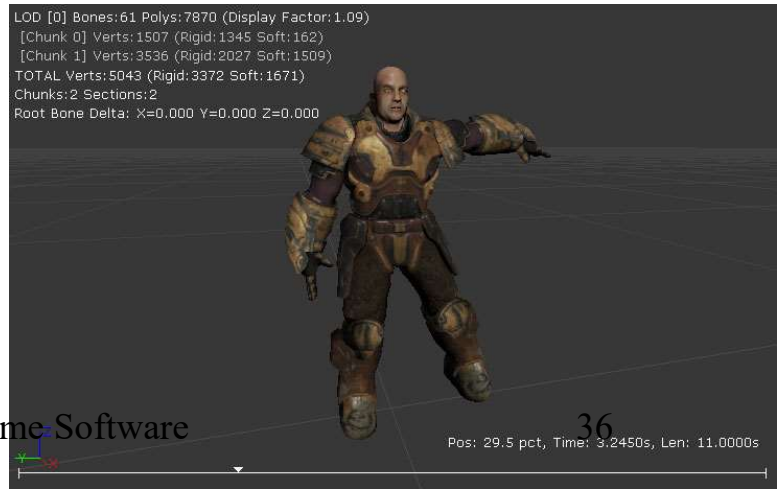
- Start from root node (pelvis) and propagate down and inheriting motions along
- Correspond to stacking transformation matrices as we enter body part
- Used together with motion capture due to too many degree of freedom in animating



Forward Kinematics

- Three components: *skeleton*, *base mesh*, *animation cycles*
- Skeleton define the hierarchy of body parts
- Contains bones such as hand, lower arm, etc.
- Mesh is affected by the underlying bones





Forward Kinematics

- Skeleton & mesh are stored in a reference pose, thus assume no joint rotations

- Animation cycles simply store joint rotation

Joint type	**example**
– Mono axial joint :	elbow
– Biaxial :	shoulder
– 3rd degree of freedom:	neck

Run time FK

- Rotations must be interpolated – a problem
- *Quaternion* & matrix used in general
- Reconstruct matrices for each body part
  - Begin at base node(*pelvis*) and follow skeletal hierarchy, stacking each joint's transform
  - The condensed matrix is applied to vertices of base mesh

Quaternions

- A rotation matrix needs 9 parameters (3x3), but the actual degree of freedom is 3 only i.e. rotation about the three axis.
- Quaternions uses 4 only
  $q = s + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$
  $= (s, \boldsymbol{v})$

e.g. X axis unit vector
$(0, \mathbf{i})$

- **i,j,k** are unit quaternions i.e. unit vector, with $\mathbf{i^2=j^2=k^2=ijk=-1}$, **ij=k, ji=-k, ik=-j …**

Quaternions

Basic operation

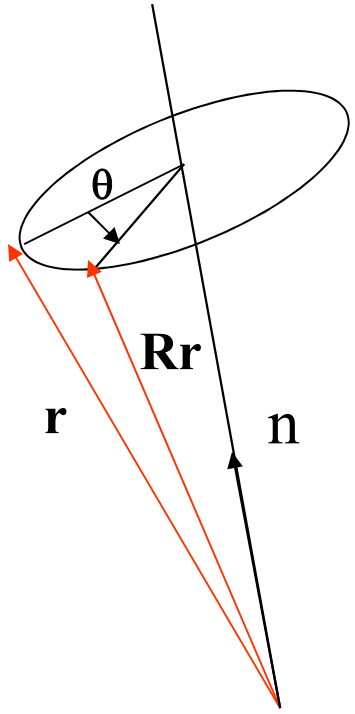
- Addition
  q + q' = (s+s', ***v*** + ***v***')
- Multiplication
  qq' = (ss'-***v***.***v***', ***v***x***v***'+s***v***'+s'***v***)
- Conjugate

$$\overline{q} = (s, -v)$$

- Magnitude $q\overline{q} = s^2 + \left|v^2\right| = \left|q\right|^2$

Quaternions

- if $|q| = 1$, then q is called a unit quaternion.
- Used in specifying general rotations, e.g.
- Consider rotating *r* about axis *n* by **q,**
  in quaternion space *r* is represented as p=(0,***r***)

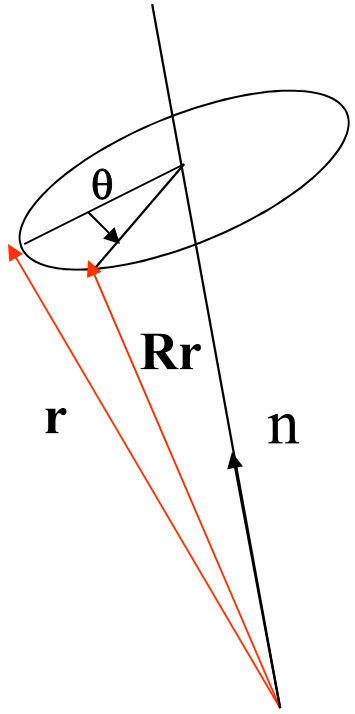


Angular displacement (θ,n) of r

Quaternions

- in three dimensional space, rotated vector R***r***
- In quaternion space, rotation θ about ***n*** (matrix R) is

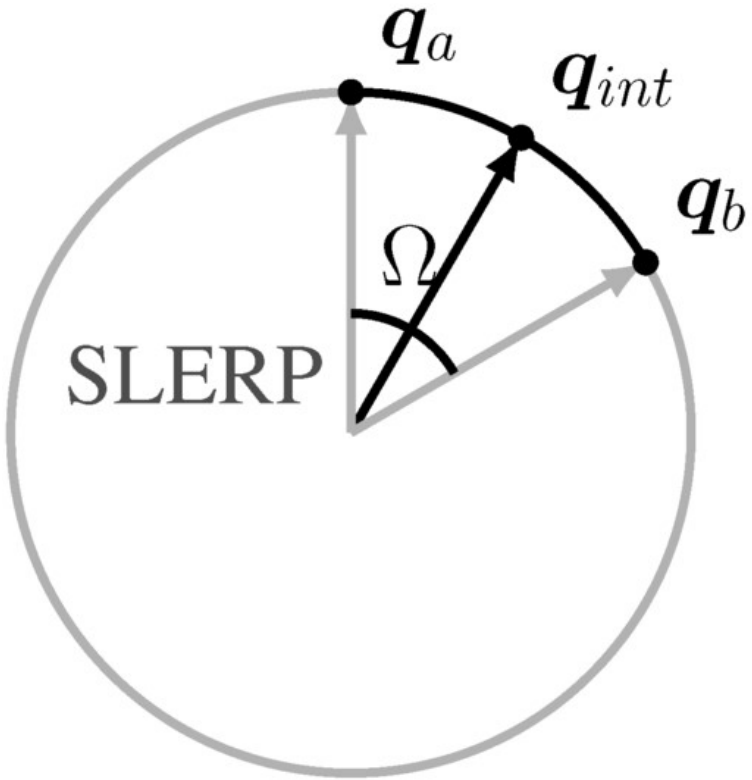
  q=(cos (θ**/2**), sin (θ**/2**) ***n***)
- The rotated vector R***r*** is given by
  $R_q(p) = \boldsymbol{qpq}^{-1}$



Angular displacement (θ,n) of r

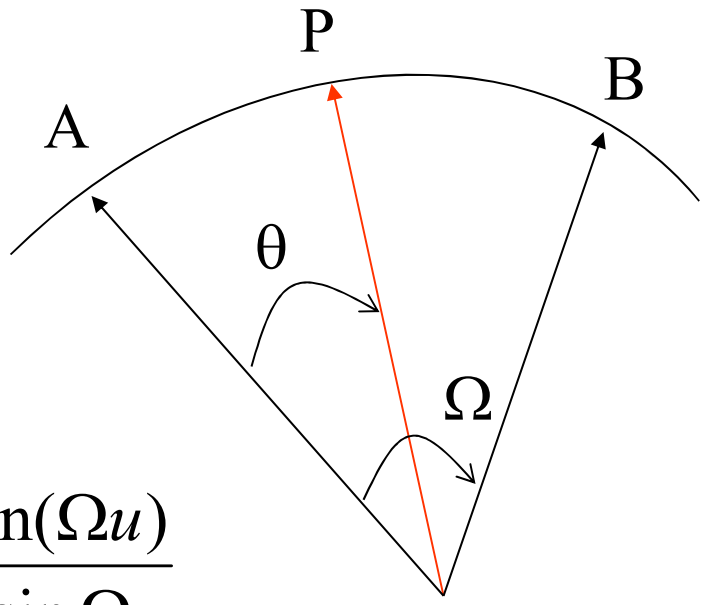
Interpolating Quaternions

- A rotation is a unit quaternion => all rotations can be mapped onto a surface of 4D unit hypersphere
- interpolating between any two rotations is equivalent to an arc on the surface joining them (spherical linear interpolation ***slerp***)



Slerp in quaternions

- Suppose quaternion of A and B is q_1, q_2 & $q_1.q_2=\cos\Omega$



- To interpolate between q_1 & q_2

$$slerp(q_1,q_2,u)=q_1\frac{\sin((1-u)\Omega)}{\sin\Omega}+q_2\frac{\sin(\Omega u)}{\sin\Omega}$$

where $\boldsymbol{u}\ \varepsilon\ [0,1]$

Strength & Weakness of Quaternion

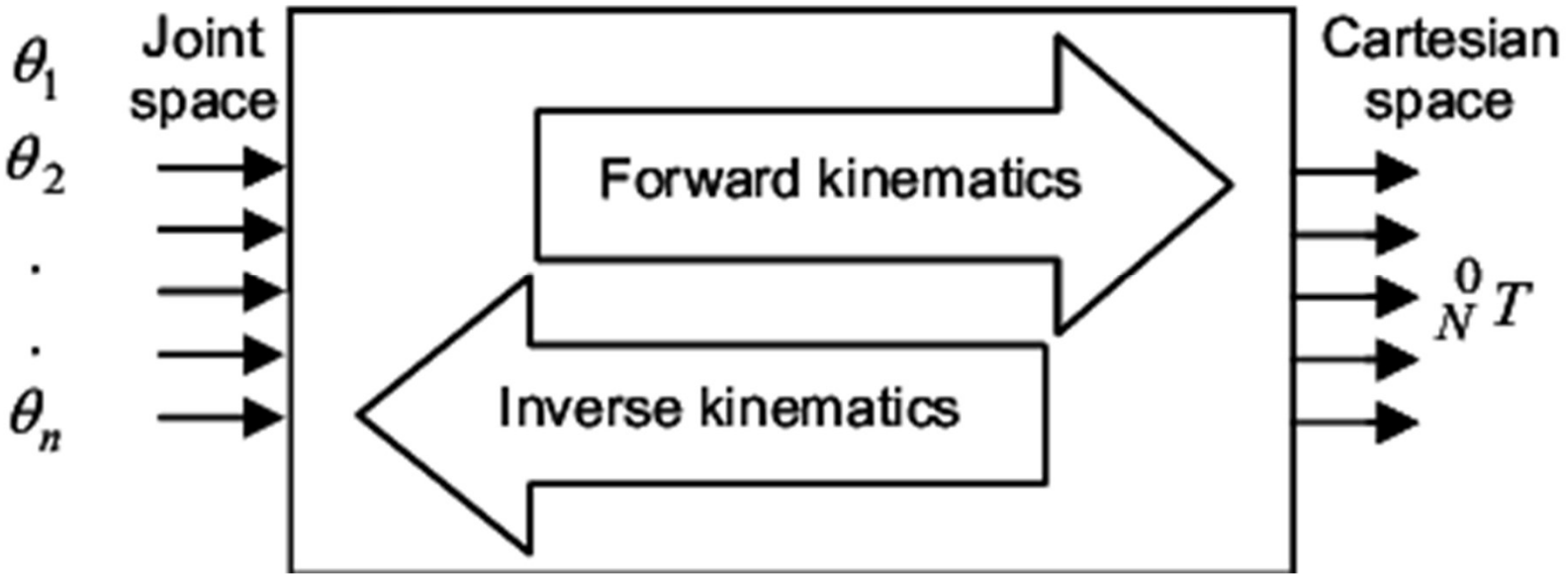
- ***Parametrize*** rotations more clearly than Euler angles (roll,pitch,yaw) i.e. by simple multiplication => useful in bone rotation calculation in character
- Only 4 parameters, less than 9 elements in matrix
- Clean formulation of interpolation (slerp) => easier to handle complicated skeletal structure

Strength & Weakness of Quaternion

- Not easy to comprehend
- Cannot ***control*** which path slerp undergo
  Quaternion q and –q is the same as
  $(-q)()(-q)^{-1} \Leftrightarrow q()q^{-1}$
  this results in for every pair of quaternion, there exists 2 possible interpolation paths – one having longer (not preferred) trajectory

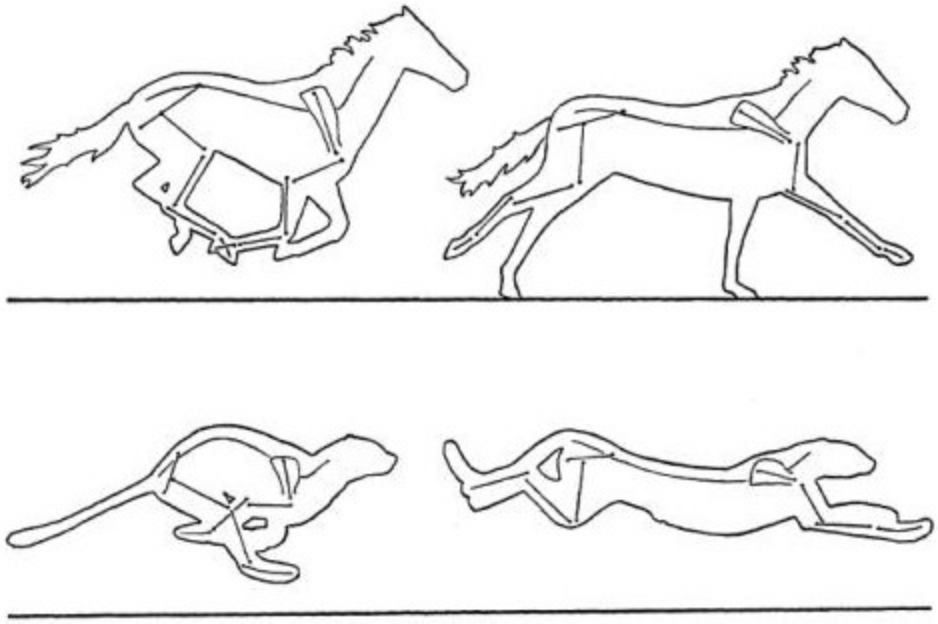
Forward/Inverse Kinematics

- Forward and inverse kinematics are two different ways to control a complex articulated system to produce animation
- The idea is how to make the system to reach a target point with adjustment of its parameters



Forward Kinematics

- A vertex can be influenced by one or more bones
- Gaps would develop if a vertex is being affected by only one bone
- Animal structures are not rigid and the links between joint deformed when in motion
- FK is thus limiting the creativity in making animations

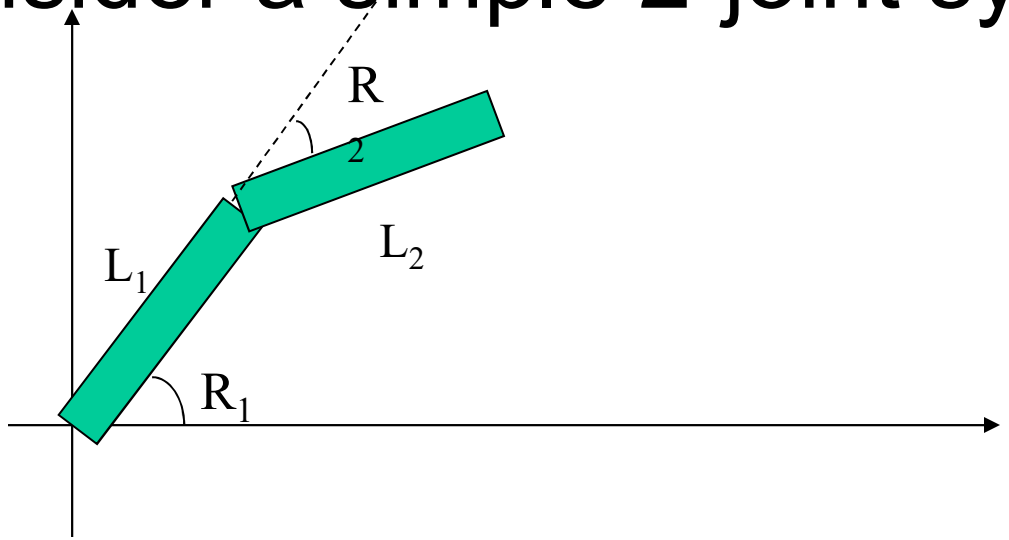


Inverse Kinematics

- Start with terminal body part & compute joints as it moves higher in skeleton
- Suitable for situation where an element is located in space & calculating the needed configuration of ancestor joints
- Useful for adaptive animation
- Not realistic enough

Analytic IK

- Analytic solver to find solution for sets of equations describing the system
- Suitable for small number of joints
- Consider a simple 2-joint system



Analytic IK

- End point P_1 of first bone

$$P_x = L_1 \cos(R_1)$$
$$P_y = L_1 \sin(R_1)$$

- End point P_2 of second bone

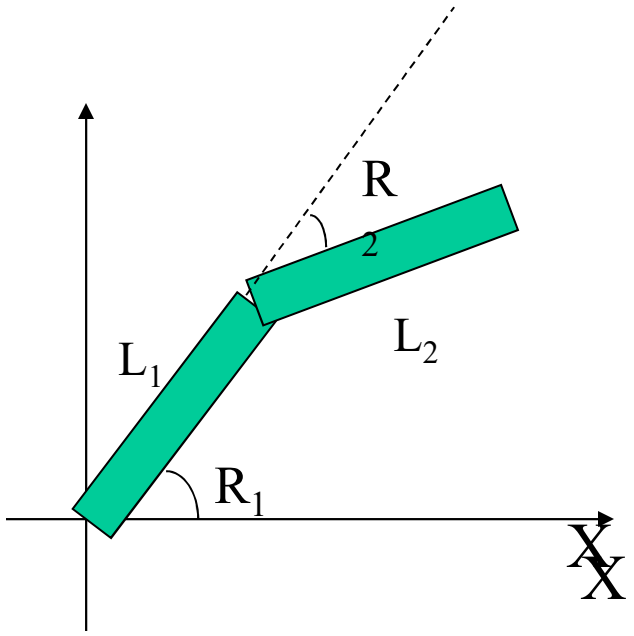
$$P_x = L_1 \cos(R_1) + L_2 \cos(R_1 + R_2)$$
$$P_y = L_1 \sin(R_1) + L_2 \sin(R_1 + R_2)$$

- Solving give

$$P_x^2 + P_y^2 = L_1^2 + L_2^2 + 2L_1L_2 \cos(R_2)$$

thus

$$R_2 = \cos^{-1} \frac{P_x^2 + P_y^2 - (L_1^2 + L_2^2)}{2L_1L_2}$$

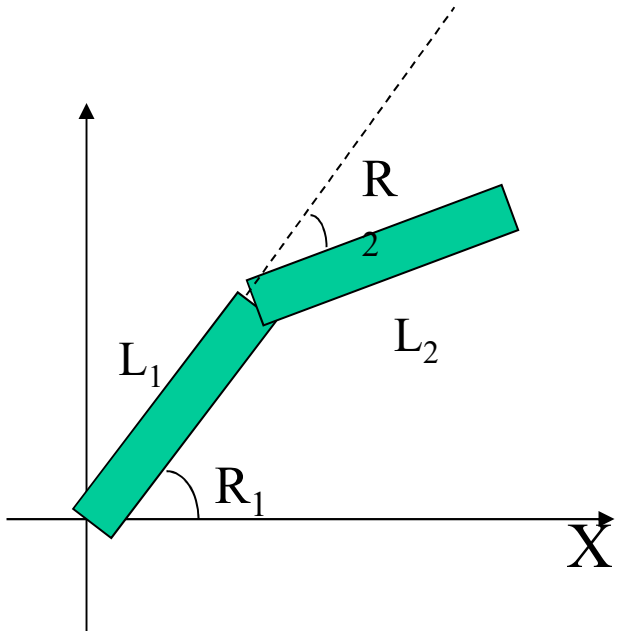


Analytic IK

- Substituting and give

$$R_1 = \tan^{-1} \frac{-(L_2 \sin(R_2))P_x + (L_1 + L_2 \cos(R_2))P_y}{-(L_2 \sin(R_2))P_y + (L_1 + L_2 \cos(R_2))P_x}$$

$$R_2 = \cos^{-1} \frac{P_x^2 + P_y^2 - (L_1^2 + L_2^2)}{2L_1L_2}$$



Analytic IK

- In solution for R_2, if the arc cosine value is not within range –1 to 1, it means P_x, P_y is too far away (not reachable)
- For physically possible (e.g. human arm), constraints have to be applied

$$0 < R_2 < \pi$$

$$0 < R_1 < \pi$$

Analytic IK

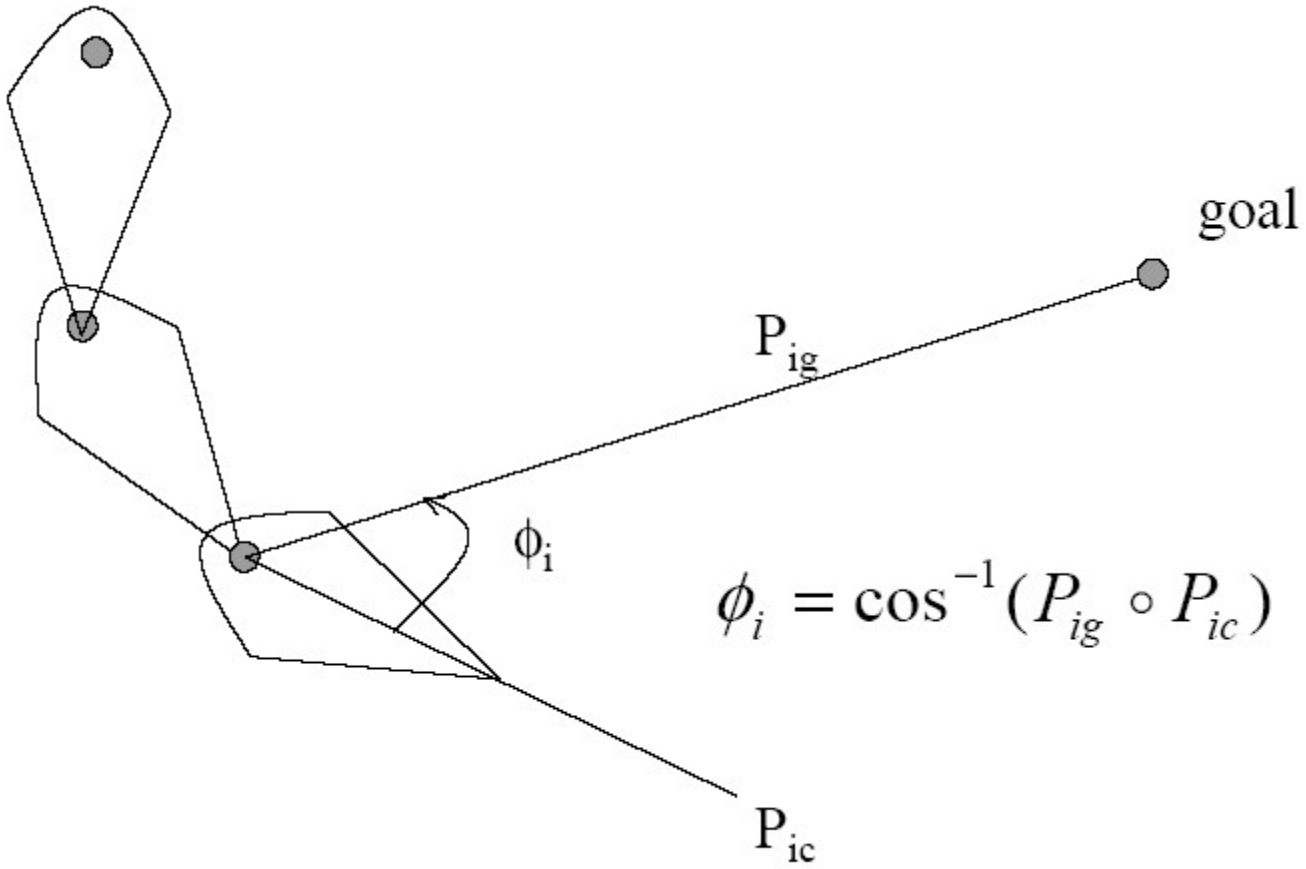
- Advantage: fast
- For 3D & more articulated solution is harder to find
- Iterative solution needed in more complicated situations

Cyclic Coordinate Descent(CCD)

- Heuristic, fast algorithm that minimize error by operating one joint angle at a time

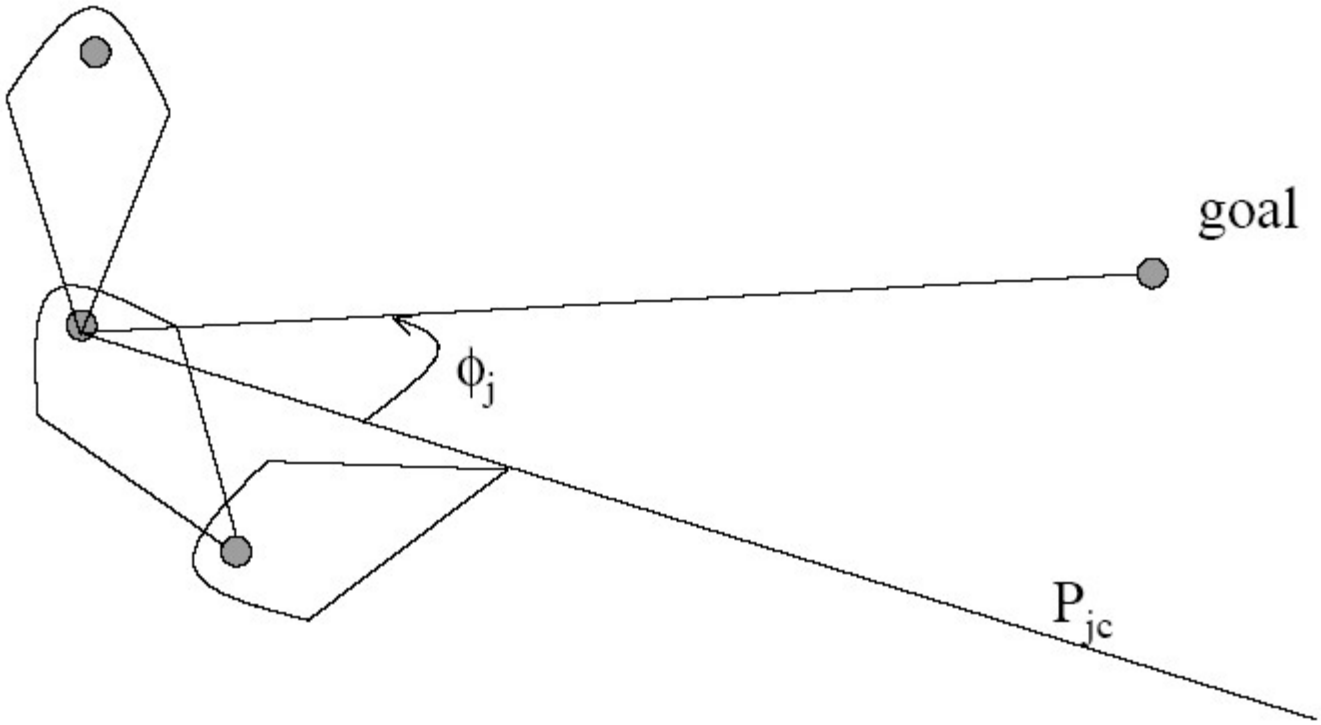
- At each iteration, pass from end effector to the root, minimizing the position and orientation of end effector for each joint angle

CCD(initial)

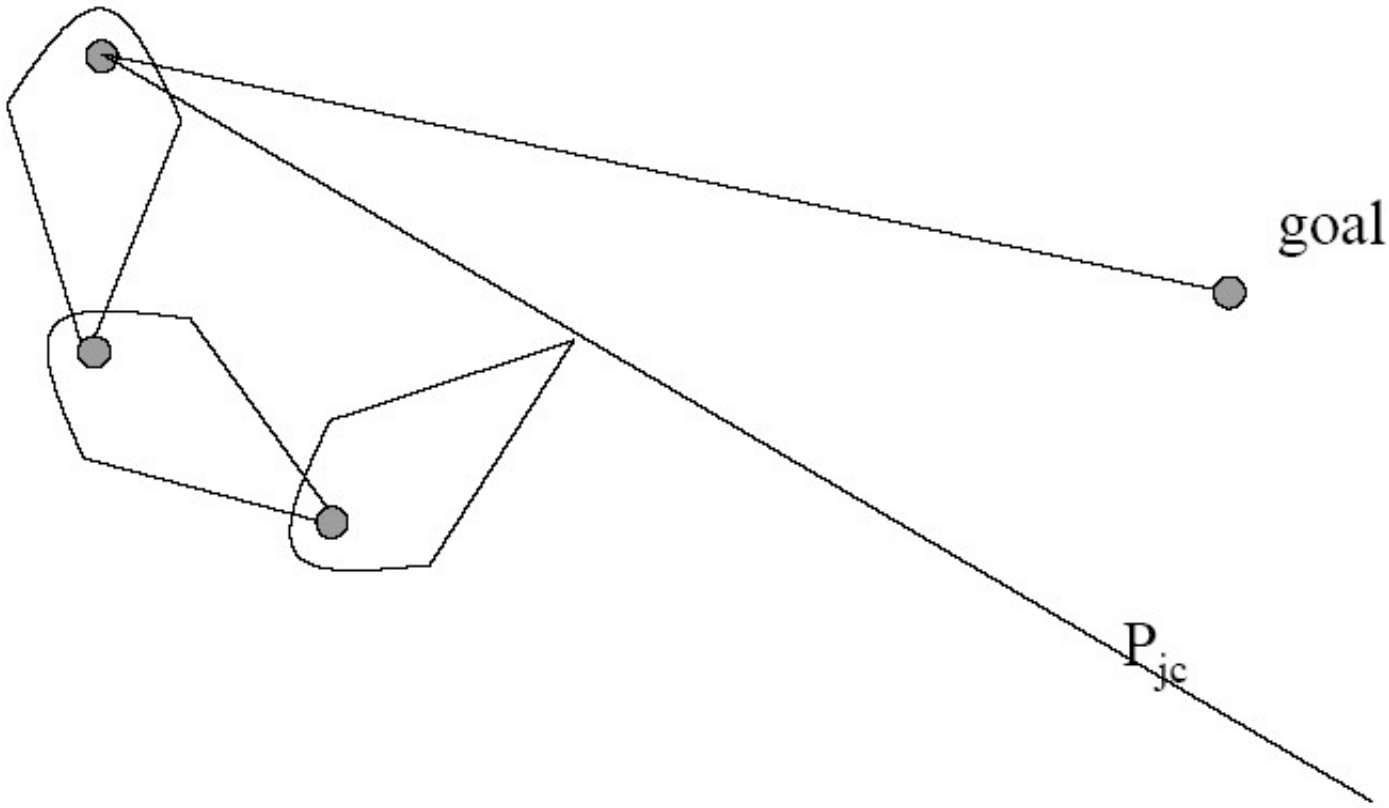


$$\phi_i = \cos^{-1}(P_{ig} \circ P_{ic})$$

CCD(1st iteration)



CCD(2nd iteration)



CCD

- Joint limit constraint easily incorporate as each joint handles individually
- Drawback: links near to end effector are favored, producing unnatural looking results
- Angle results are used to drive a quaternion or matrix interpolator top create the animation

Blending forward & inverse kinematics

- FK when coupling with motion capture gives high quality animation
- IK can produce miscellaneous movement such as reaching out, which FK can't
- IK produce unnatural animation
- Mixing FK & IK on bone-by-bone basis

Blending forward & inverse kinematics

- Bones closer to root more FK-oriented
- Bones closer to terminals more IK-based

- Use CCD on top of an FK animation
- Making joints more stiff as reaching higher in hierarchy

- FK provide overall movement & IK ensure good interactions with environment

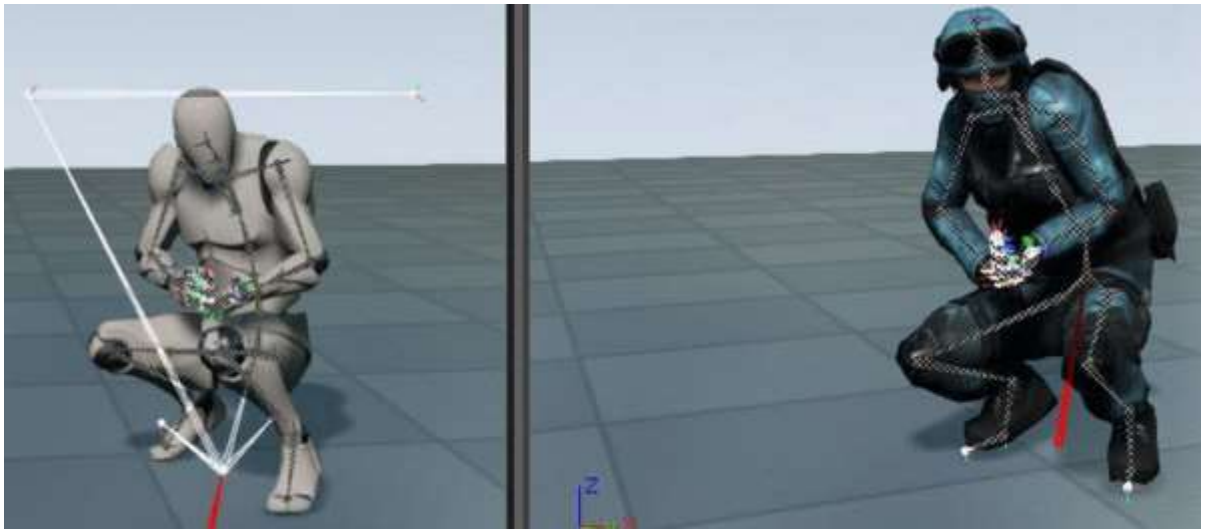
- Use in movie production such as “Lord of the ring”

Motion Retargeting

- Most motion capture are based on some standard human model
- The animations thus would only fit corresponding metric models

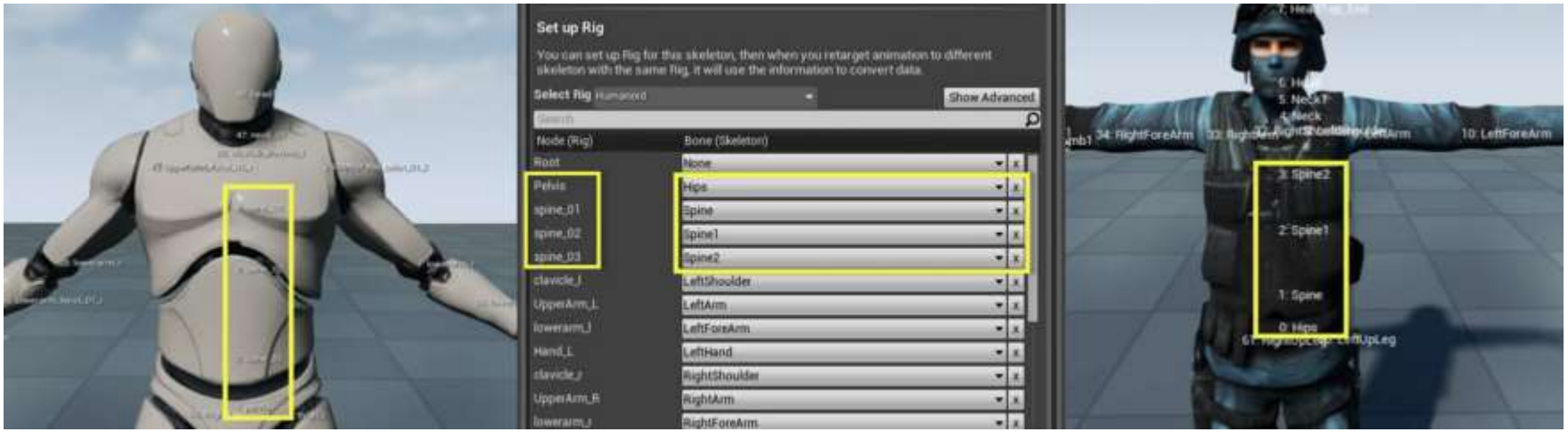
Motion Retargeting

- Allows animations to be reused between characters that don't share same skeletons eg. Taller, shorter etc.



Motion Retargeting

- Usually need some generic rig for bone matching between different skeletons
- Once bone matching done, animations can be copied to new skeleton



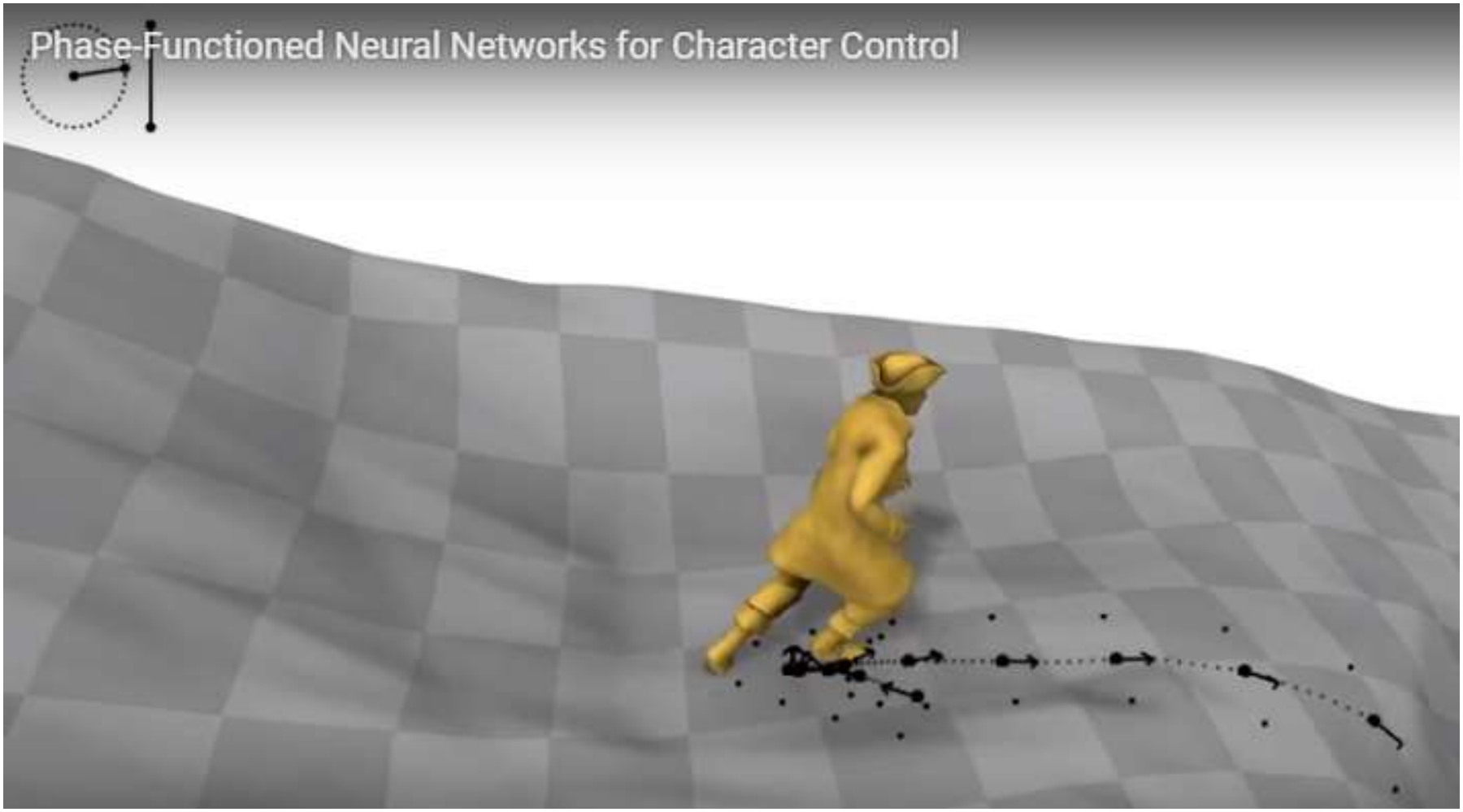
Motion Retargeting

- Retargeting adjustment usually need be done as bone matching usually can't be perfect

Improper retargeted animation

animation after Adjusting ref pose

Recent Advances



- Using AI in character animation

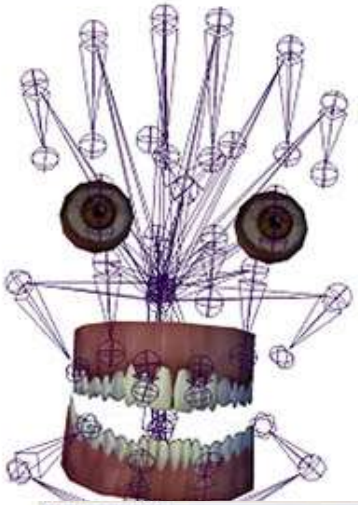
Facial Animation

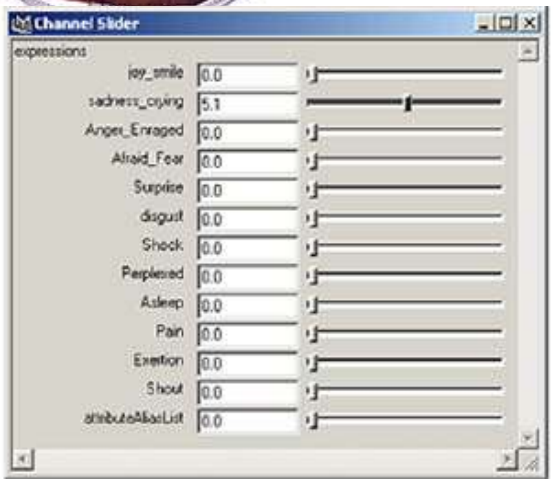
- To produce realistic facial expression for in-game character
- Also involve lip-synchronization for speech in game

Halflife 2

Facial Animation

- Produce expressions by distorting the facial mesh
  1. Attaching muscle unit to affect selected vertices e.g. eyebrows, eyeballs, teeth, mouth etc.
  2. Animators further edit the expressions by adjusting degree of muscle units

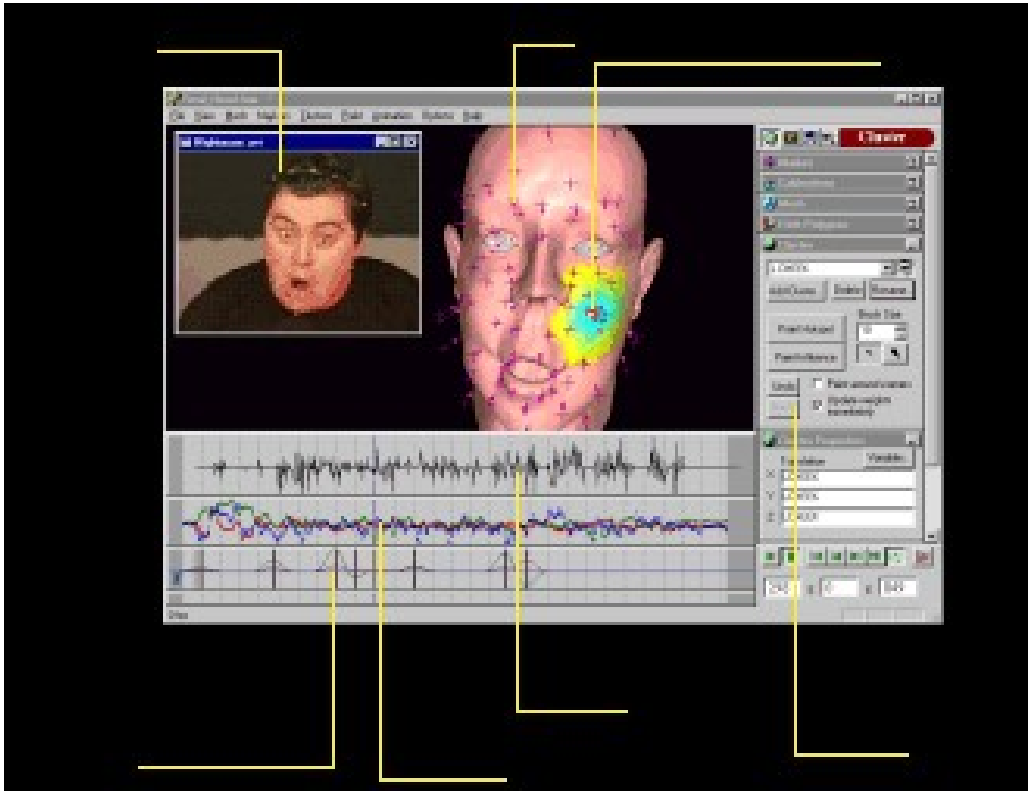




The Getaway

Facial Animation

- Produce expressions by distorting the facial mesh
  - Using motion capture device to drive a reference face mesh



Facial Animation

- Lip synchronization
- Produce correct lip motion with corresponding speech
  1. extracting phonemes from a .wav file
  2. forming the corresponding phoneme blend shape for the mouth

Valve's proprietary Faceposer

Reference

Motion capture in making of “Death Stranding”(2019)
https://www.youtube.com/watch?v=faatWOs2c5c

Summary

- Only touch on the topic
- Physically based character that keep self balance and implement correct joint rotation restriction begin to deploy for games
- realistic facial expressions is becoming more popular
- Motion retargeting is now also available as a tool