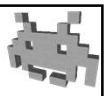
Course Plan



lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** +
lec. 5: Game **Particle Systems**
lec. 6: Game **3D Models**
lec. 7: Game **Materials**
lec. 8: Game **Textures**
lec. 9: Game **3D Animations**
lec. 10: **3D Audio** for 3D Games
lec. 11: **Networking** for 3D Games
lec. 12: **Interactive Agents** for 3D Games
lec. 13: **Rendering Techniques** for 3D Games

★ ★
computer animation

107

Skinning: it's part of the rendering (in GPU)

To render a mesh…

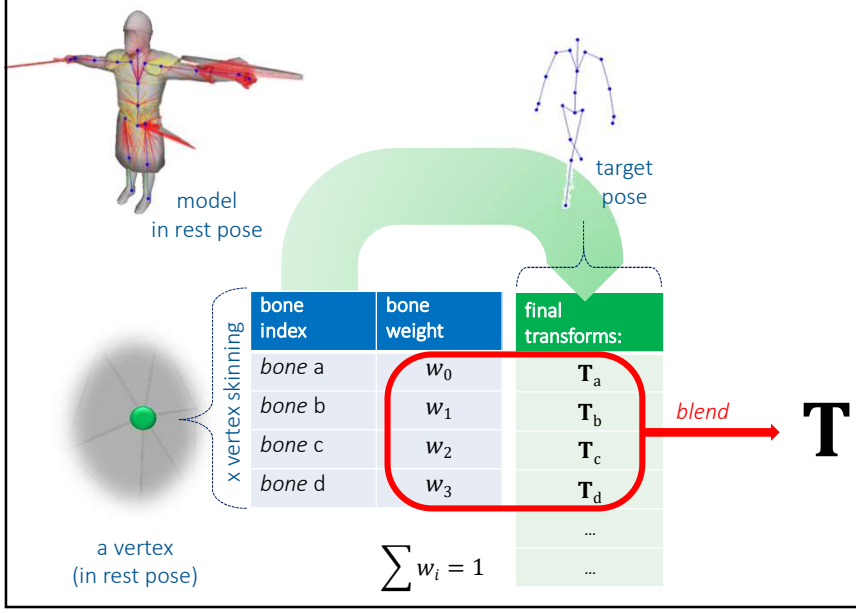
- Load…
  - make sure all data is ready in **GPU RAM**
    - Geometry + Attributes
    - Connectivity
    - Pose: *final* transforms per bone
    - Textures
    - Shaders
    - Material Parameters…
- …and Fire!
  - issue the Draw Call

includes **skinning (bone weights)**

THE MESH ASSET
THE ANIMATION (current pose)
THE MATERIAL ASSET

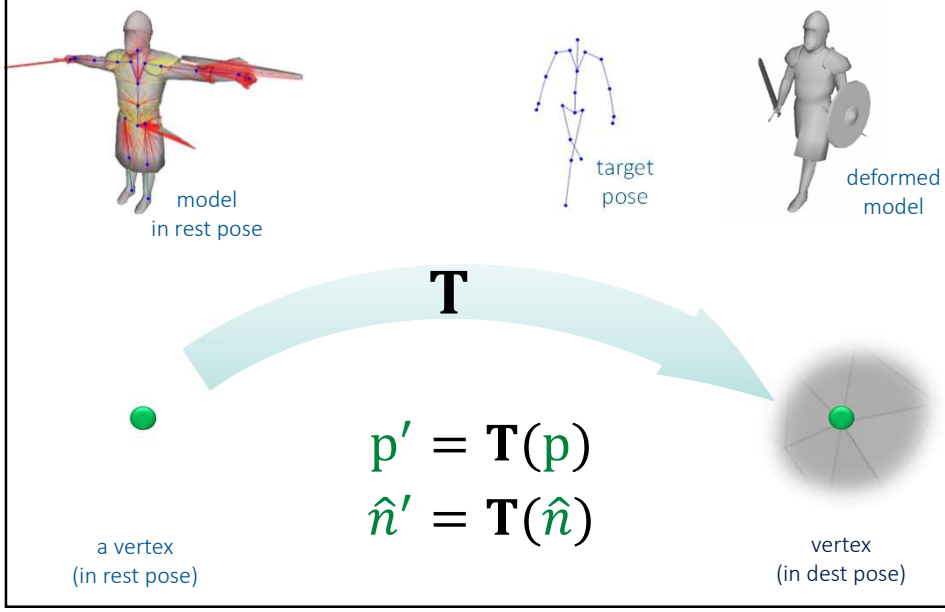
108

Skinning - how it works (in GPU)

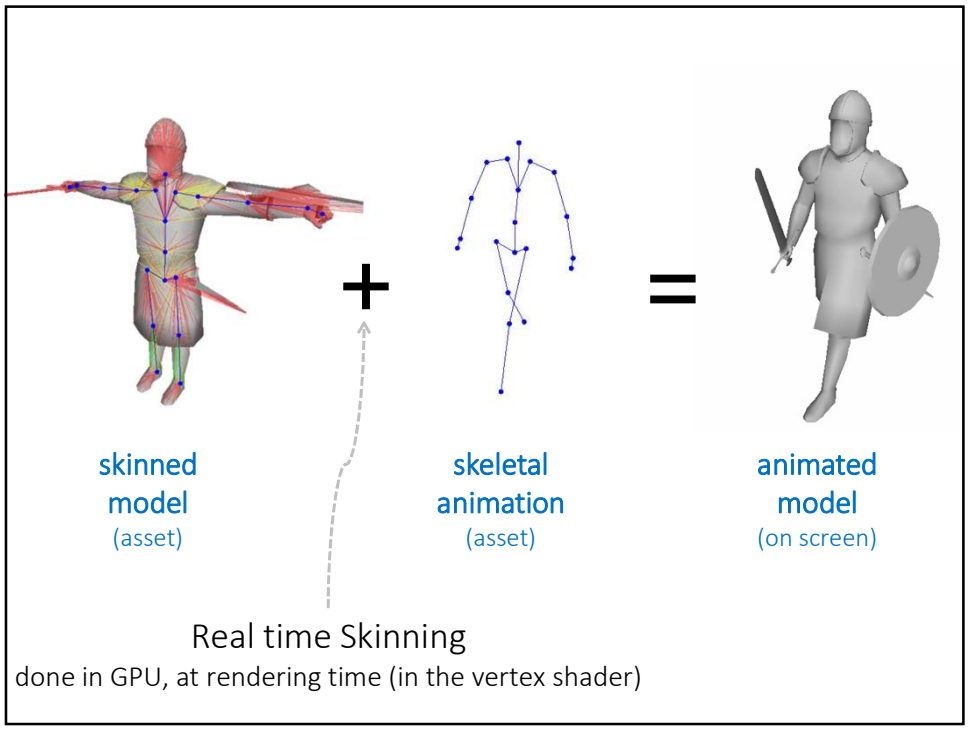


109

Skinning - how it works (in GPU)

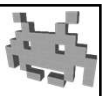


110



111

Skinning - how it works (in GPU)



To render a mesh…

- Load…
  - make sure all data is ready in **GPU RAM**
    - Geometry + Attributes
    - Connectivity
    - **Pose**: *final* transforms per bone
    - Textures
    - Shaders
    - Material Parameters…
- …and Fire!
  - issue the Draw Call

includes **skinning (bone weights)**

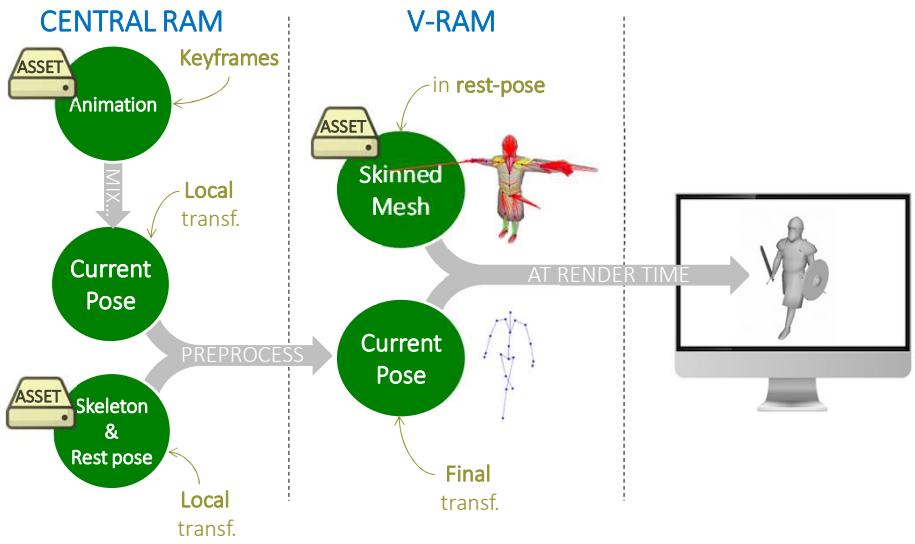
THE MESH ASSET

THE ANIMATION (current pose)

THE MATERIAL ASSET

113

The deformed mesh is computed on-the-fly during rendering (not stored!)



114

GPU real time Skinning – variants

(a choice of the rendering engine)

Bone	Weight	Final Transform
bone a	w_0	$\mathbf{T}_a$
bone b	w_1	$\mathbf{T}_b$
bone c	w_2	$\mathbf{T}_c$
bone d	w_3	$\mathbf{T}_d$

blend → $\mathbf{T}$

how are they stored? | how is this done?

solution 1: «Linear Blend Skinning»	as a 4x4 matrix transformatiton	with linear matrix interpolation
solution 2: «Dual Quaternion Skinning»	as a dual quaternion	with dual-quaternion interpolation

nothing else works! (that we know)

115

Representations for roto-translations (recap)

see lecture on transform representation!

- Euler Angles
- Angle + Axis
- Quaternion

} + Transl. (vector)

won't work for blend skinning

- 4×4 Matrix (or 3×3 Matrix + Translat.) — solution 1 (LBS)
- Dual Quaternion — solution 2 (DQS)

116

Linear Blend Skinning (LBS)

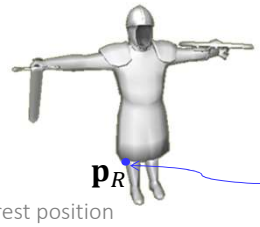
more in general, Nmax-1

linear interpolation of per-bone matrices (no longer rigid)

$$\mathbf{p}_P = \left(\sum_{i=0}^{3} w_i \, \mathrm{T}[b_i] \right) (\mathbf{p}_R)$$

interpolation of per-bone transformed points

$$= \sum_{i=0}^{3} w_i \left(\mathrm{T}[b_i](\mathbf{p}_R) \right)$$

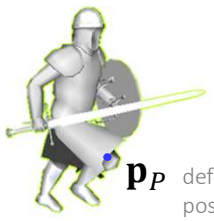


$\mathbf{p}_R$

rest position of the vertex

skinning (per vert attribute):

(b_0, w_0)
(b_1, w_1)
(b_2, w_2)
(b_3, w_3)

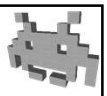


$\mathbf{p}_P$

deformed position of the vertex

117

Dual Quaternion Skinning (DQS)

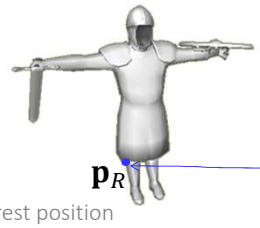


Interpolated DUAL QUATERNION
(still a *rigid* transform)

$$\mathbf{p}_P = (\text{mix}(w_0\ , \text{T}[b_0], \dots))(\mathbf{p}_R)$$

weighted interpolation of DUAL QUATERNIONS (see lecture on quat and dual-quat)

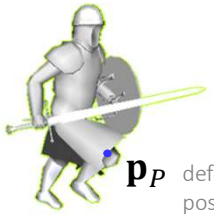
Final transforms (a rigid transform) expressed as a Dual Quaternion



rest position of the vertex

skinning (per vert attribute):

(b_0, w_0)
(b_1, w_1)
(b_2, w_2)
(b_3, w_3)



deformed position of the vertex

118

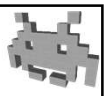
LBS: intuition on why it works (robustly!)

$$\left(\sum_{i=0}^{3} w_i\ \text{T}[b_i]\right)(\mathbf{p}_R) = \sum_{i=0}^{3} w_i\big(\ \text{T}[b_i](\mathbf{p}_R)\big)$$

- If final transformations are expressed (and interpolated) as affine matrices, then "everything is linear":
  - Interpolating the matrices (left of =) **is equivalent to** interpolating the transformed points (right of =)
  - Therefore: nothing can go really wrong!
- **Problem:** interpolation of rotation matrices does not produce a rotation (as we know)
  - Unwanted shears and down-scalings are introduced
  - Conclusion: LBS can "shrink" the object (a bit) (or a lot, when rotations are very different)

119

DQS: intuition on why it works
(but separately interpolating transl & rot would not)

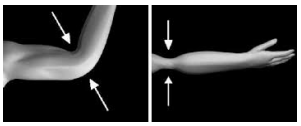


- Problem with storing (and interpolating) Rot. and Transl. of FINAL transforms separately:
  - a **final transf** is a long sequence of "rotate-then-translate"
  - it all boils down to a single roto-translation (as we know)
  - to represent it, we must **choose**: "which one goes first" (R then T, or, T then R)?
  - Both choices are equivalent, in that they let us express *any* roto-translation (in one way only)
  - But different choices → very different interpolation results
  - Usually, neither produces good results
- Dual quaternions bypass the problem
  - The representation picks neither step to "go first"

121

Dual Quaternion Skinning (DQS) VS *Linear Blend Skinning* (LBS)

- LBS …
  - Is a bit cheaper to compute
  - Can shrink surfaces a little
    Example: "candy wrapper" effect ==>

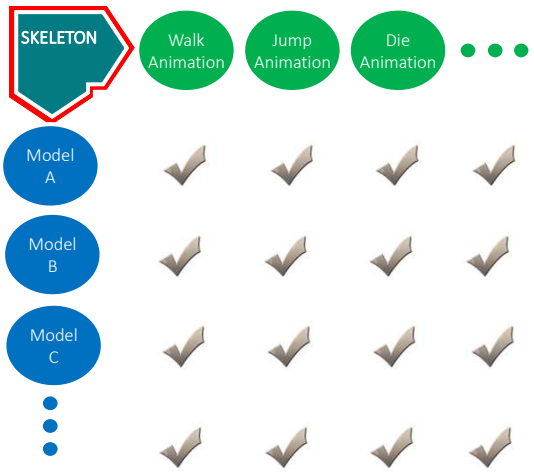


  - Ability to express (uniform) scaling in per-bone transformations (local ones, therefore final ones)
  - Is an older technique, more established
- DQS …
  - costs some ~20-50% more FLOP operations per vertex (depending on implementation details)
  - Works better, avoids candy wrapper effects
  - May have the opposite defect of *enlarging the volumes*.
  - Can only express rigid motions (no per-bone scaling)

Both are used in practice.
They use the exact same set of ASSETS! (skinned mesh, skeletons, anims)

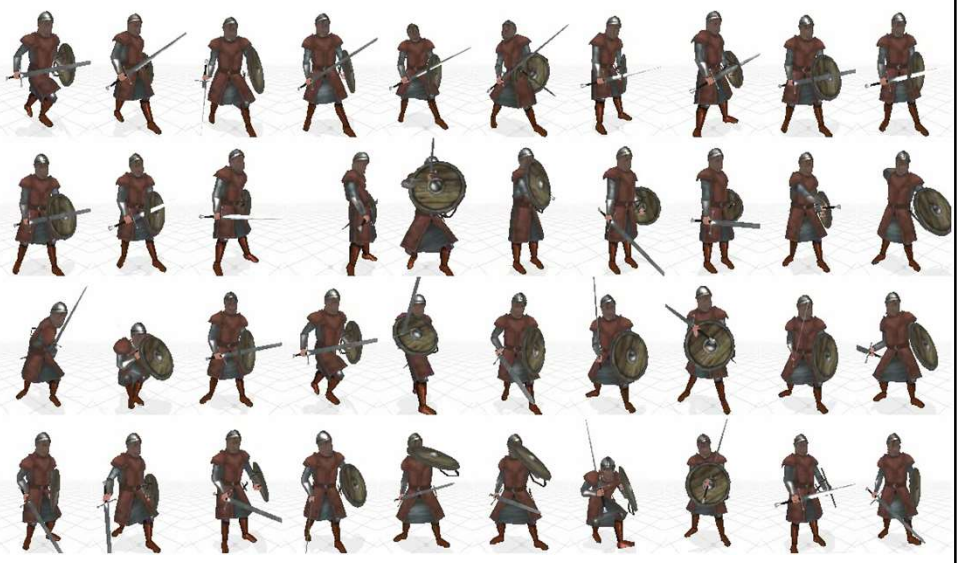
122

Orthogonality animation / models



127

Orthogonality animation / models



same model, different anims

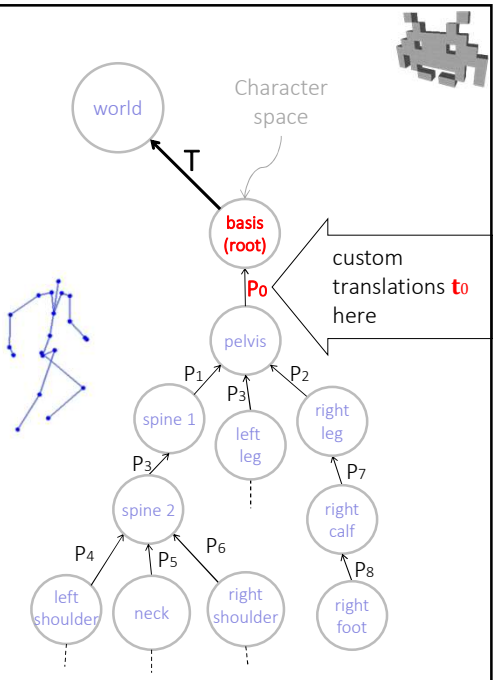
128

Orthogonality animation / models

same anim, different models

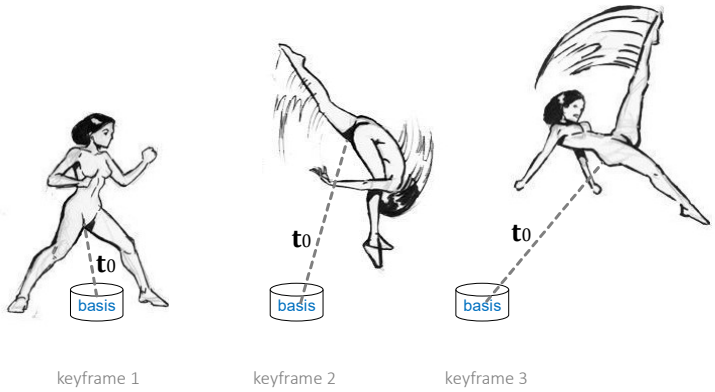
129

The root is usually an abstract "basis" bone



130

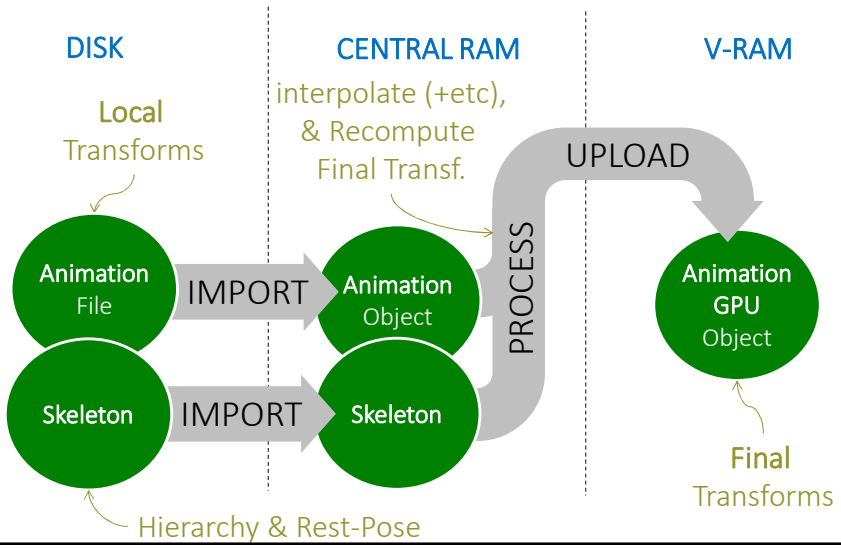
Basis bone



Each animation can include
a global translation $\mathbf{t}_0$ for each keyframe ,
as the translation of the top-most bone (hip bone → basis)
Normally, no other bone redefinines the *translation* defined in the rest pose!

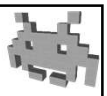
131

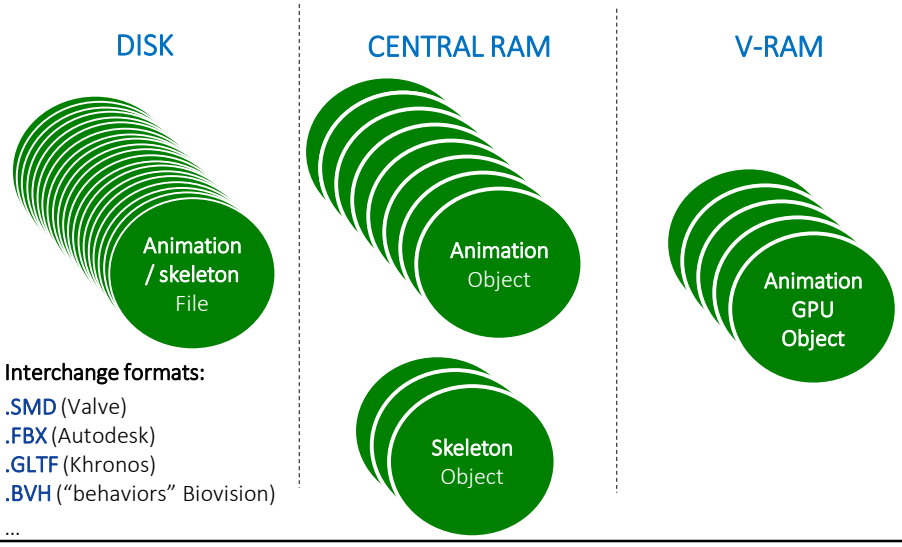
Life of Animation Assets in a Game Engine



132

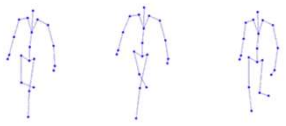
(and once again, Memory Management)





133

Animation asset on disk / RAM



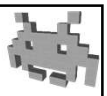
...

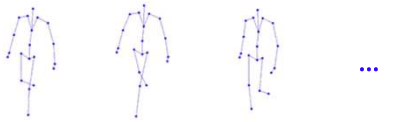
		Keyframe 1	Keyframe 2	Keyframe 3	...
time		*t*	*t*	*t*	*t*
Bone	#0 *(pelvis, root)*	T	T	T	T
	#1 *(spine)*	T	T	T	T
	#2 *(chest)*	T	T	T	T
	#3 *(shoulder sx)*	T	T	T	T
	...	...	...	...	...
	#10 *(calf)*	T	T	T	T
	...	...	...	...	...

= Walk Animation

135

Animation asset on disk (details)





		Keyframe 1	Keyframe 2	Keyframe 3	…
time		*t*	*t*	*t*	*t*
Bone	#0 *(pelvis, root)*	T	T	T	T
	#1 *(spine)*	T	T	T	T
	#2 *(chest)*	T	T	T	T
	#3 *(shoulder sx)*	T	T	T	T
	…	…	…	…	…
	#10 *(calf)*	T	T	T	T
	…	…	…	…	…

LOCAL transforms

Scalars or int (e.g.: msec since ani started)

rotations and translations

(often) Rotations only. (e.g., as Euler: most compact) *Translations are inherited from rest pose!*

136

Animation asset on disk / RAM (details)

T : rotation + translation

T : rotation only

✕ : not stored (interpolated on the fly)

Sometimes times are implicit (if equally spaced)

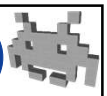
		Keyframe 1	Keyframe 2	Keyframe 3	…
time		*t*	*t*	*t*	*t*
Bone	#0 *(pelvis, root)*	T	✕	✕	T
	#1 *(spine)*	T	T	T	✕
	#2 *(chest)*	T	T	T	T
	#3 *(shoulder sx)*	✕	T	✕	T
	…	…	…	…	…
	#10 *(calf)*	T	✕	✕	T
	…	…	✕	…	…

Sparse Representation! (to save space)

Not every transformation is stored: (the ones which aren't will be interpolated between prev and next transf)

137

Animation asset on disk / RAM (details)



Many animations can be stored together (e.g., in a .BVH file)

T : rot + trans
T : rot only
✗ : not stored

rest pose | walk animation (cycled) | run animation (cycled) | idle animation (cycled)

Bone	Keyf 0	Keyf 1	Keyf 2	Keyf 3	Keyf 4	Keyf 5	Keyf 6	Keyf 7	Keyf 8	Keyf 9	Keyf 10	Keyf 11
time	-	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*	*t*
#0 *(pelvis, root)*	T	T	T	✗	T	T	T	T	T	✗	✗	T
#1 *(spine)*	T	✗	T	T	T	T	✗	T	T	✗	T	T
#2 *(chest)*	T	✗	T	T	T	✗	✗	T	T	T	T	T
#3 *(shoulder sx)*	T	T	T	T	T	✗	T	T	✗	T	T	T
...	...	...	...	...	...	...	...	...	...	...	...	...
#10 *(calf)*	T	T	T	✗	✗	T	T	T	✗	T	✗	T
...	...	...	...	...	...	...	...	...	...	...	...	...

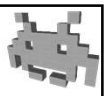
138

Sparsification of keyframes (reduce number of keyframes)

- Objective: removal of redundant keyframes
  - "Redundant" = can be approximated by in-betweens
  - A preprocessing task
- Basic algorithm concept:
  - for each keyframe Px
    - tentatively remove P_x
    - compute interpolated version P_i from remaining keyframes
      - the prev. and next ones
    - if *distance*(P_i , P_x) > MAX_ERR then reinsert keyframe P_x

139

Ways to dynamically combine different animations

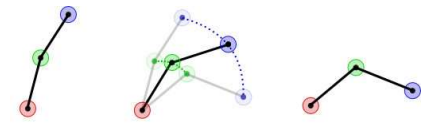


- Poses in a skeletal animations can be easily blended…
  - As long as they share the same skeleton
- … so, entire animations can be combined into new ones!
  - In real time, during game execution
  - Remember results can always be *baked*, and edited, during asset production
- Let's see two ways to combine animations:
  - Transitions between two animations (or more)
  - Compositing (layering) two animations (or more)
- As usual, we act on the local transformations
  - Before turning them into final ones

140

Interpolation of poses
(that is, of keyframes of a skeletal ani)

- any two (or more) poses can be interpolated!

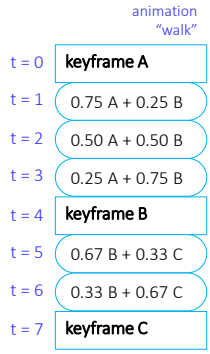


  - as long as they are defined on the same rig
  - mix is defined by interpolating the per-bone local transform
  - this requires re-computation of final transforms (after interpolation)

141

Interpolation of poses (at runtime):
transition between keyframes of an anim

- Eg:



Legend:

a stored pose

an interpolated pose

Remember poses are always interpolated as local transformations

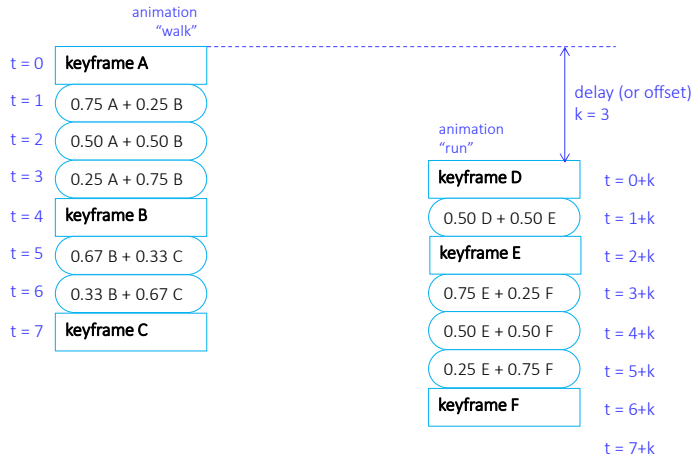
Final per bone tranformations must be recomputed after interpolation.

(of course, they can be backed and cached for efficiency)

142

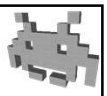
Interpolation of poses (at runtime):
transition between animations

- Eg: from *walk* to *run*

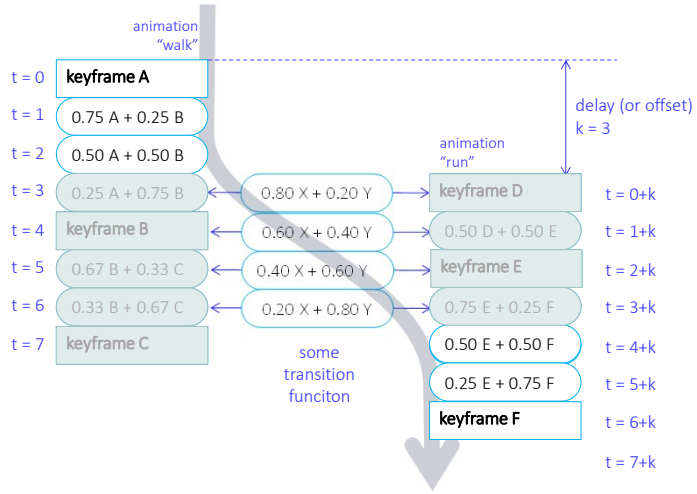


143

Interpolation of poses (at runtime):
transition between animations



- Eg: from *walk* to *run*



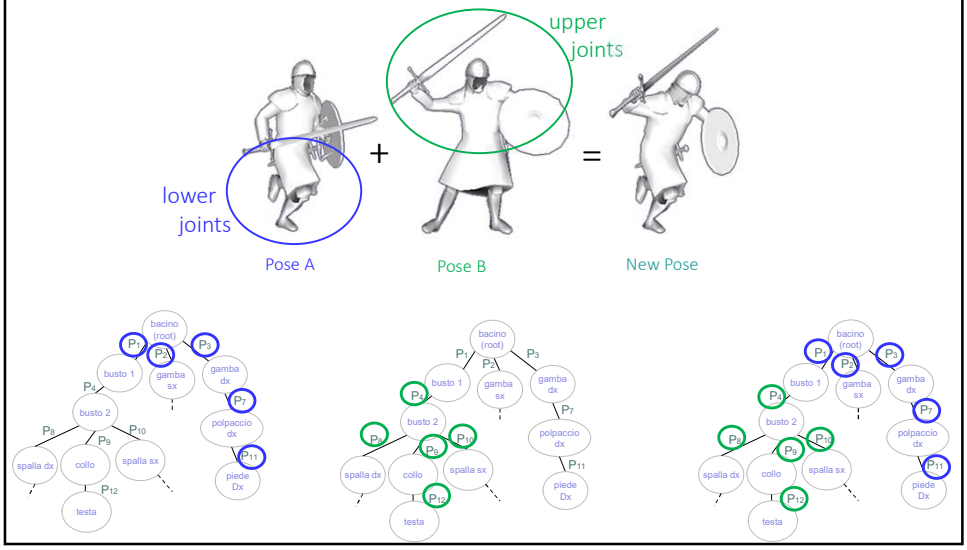
144

Interpolation of poses (at runtime):
transition between animations (notes)

- Transitions between animations
  - E.g., from stance (aka "idle") animation to walk
  - E.g., from attack to parry
  - A typical example:
    from a *gait* (run, trot, walk…) to another
    according to current character *speed*
- The quality of the resulting transition varies
  - E.g., on how similar the two interpolated frames are
    (e.g.: do they both have the left leg in front?)
  - Determining the good offsets / speed / transition function
    can be a matter of trial and error
  - E.g.: Unity has a WYSIWYG interface for that!

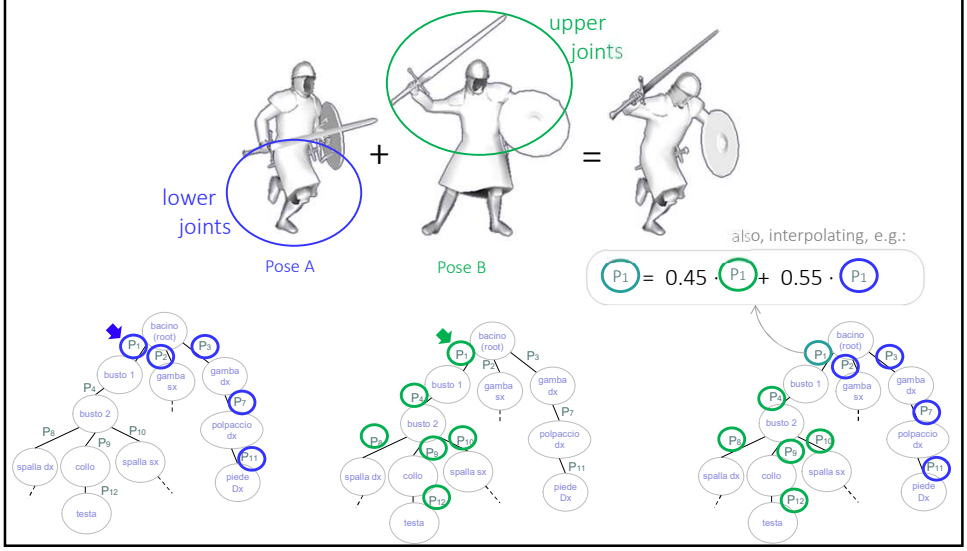
145

Compositing (layering) poses
(and thus animations)



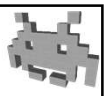
146

Compositing (layering) poses
(and thus animations)



147

Compositing (layering) poses
(notes)



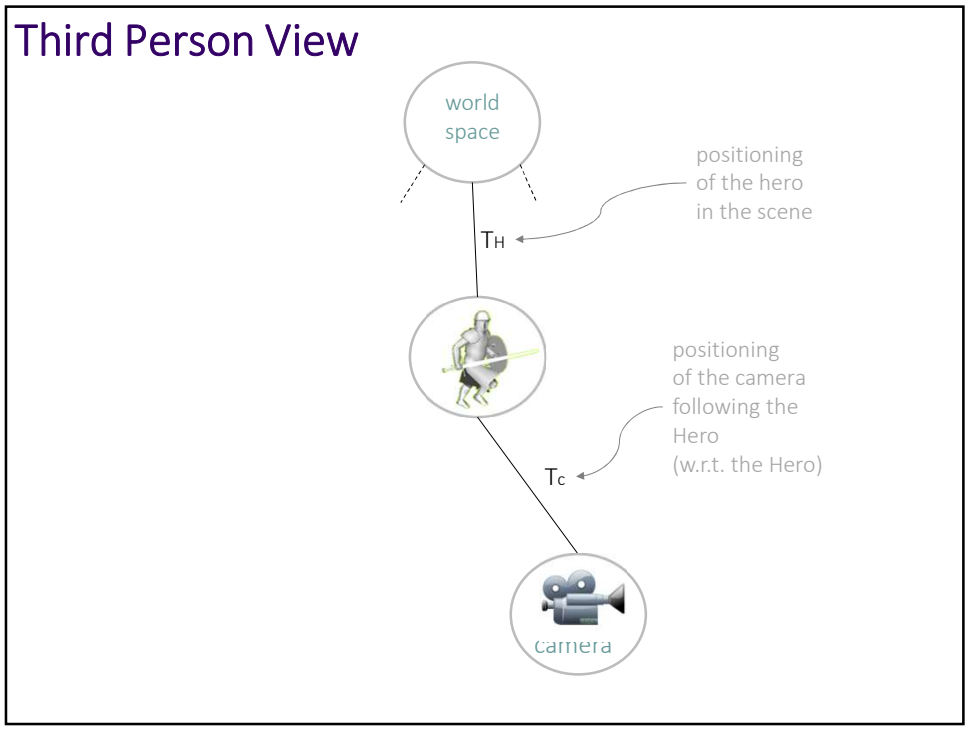
- Useful in different contexts:
  - e.g., different character parts following different anims
  - e.g., lower body: run. Upper body: aims/shoots/reload
- Note:
  local transformations are mixed (as usual).
  Final transformations need be updated after mixing
  - (after changing the local ones)
- Unity has an interface for this:
  - Per-bone Layer = a mask of per-bone Booleans:
    to local animations of which bones are "overwritten"
    by this animation?

148

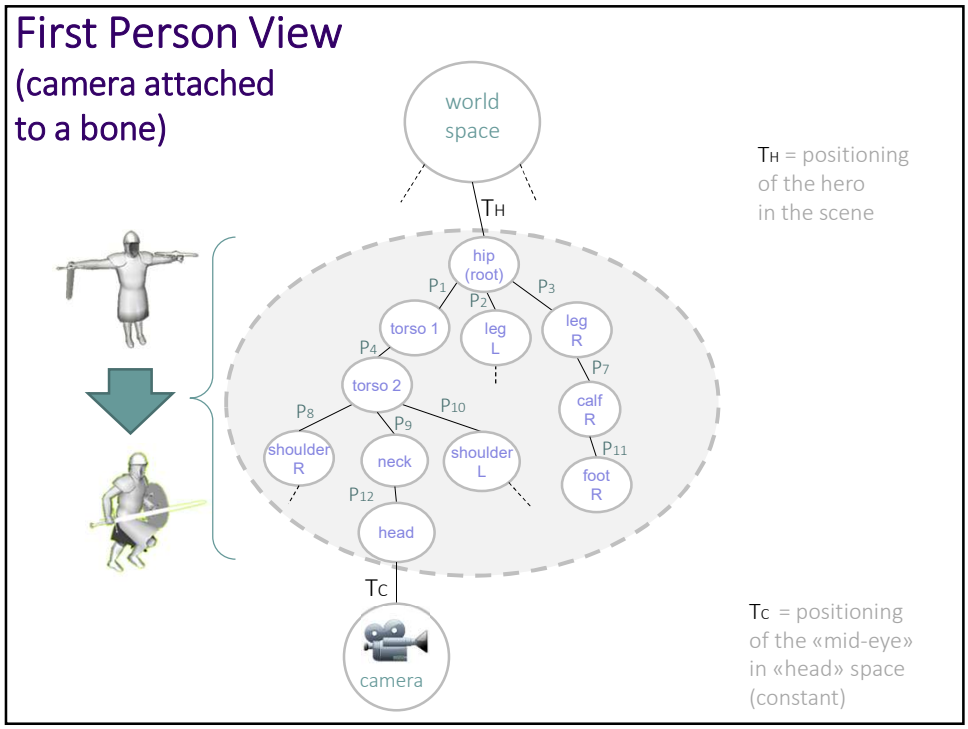
Skeleton = subtree in the scene-graph

- A skeleton can always be considered a subtree in the scene-graph
  - the local transforms in it are defined by the current frame,
    in the current skeletal animation asset
    (so they are decided by the artist / the assets)
- Examples of reasons to do so:
  - Placement of the camera in a bone (e.g. the head bone):
    the skeletal animation doubles as a camera animation!
  - Defining geometry proxies (hit-boxes) for collision detection in bones
    Note:
    in a game, collision proxies can be per-character
    (e.g. a given capsule for the entire character)
    or per-bone.
    (or both, for different things)
    Can you tell which is it, in a 3D video game that you are playing?

149

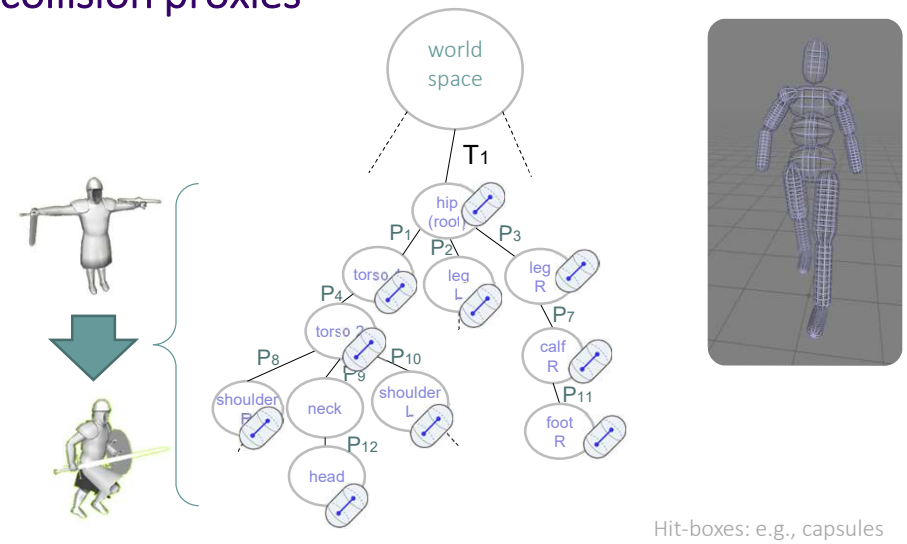

Third Person View
world space
positioning of the hero in the scene
TH
positioning of the camera following the Hero (w.r.t. the Hero)
TC
camera

150


First Person View
(camera attached to a bone)
world space
TH
TH = positioning of the hero in the scene
hip (root)
P1
P2
P3
torso 1
leg L
leg R
P4
P7
torso 2
calf R
P8
P10
P9
P11
shoulder R
neck
shoulder L
foot R
P12
head
TC
camera
TC = positioning of the «mid-eye» in «head» space (constant)

151

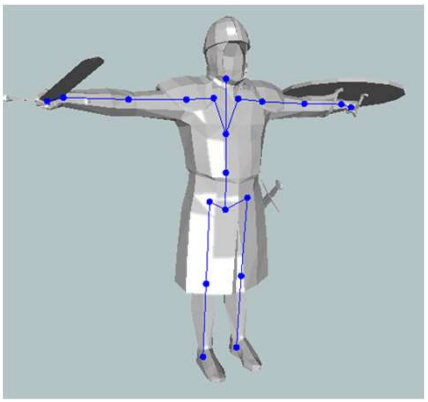
Per-bone collision proxies

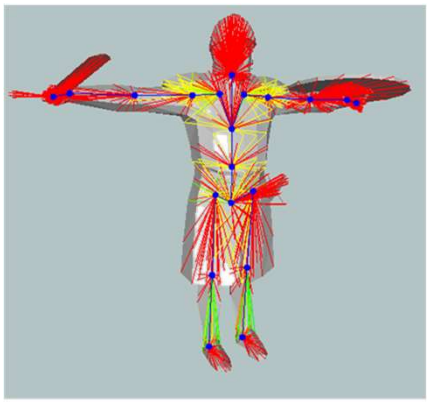


Hit-boxes: e.g., capsules
(not necessarily in *every* bone)

152

Steps in the asset-creation pipeline: Rigging & Skinning (of a 3D mesh)



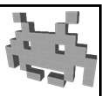


Rigging – authoring of a rig
the "control rig" (or "rig")
includes the skeleton
(and GUI controls usable to pose it)

Skinning – authoring of the skinning
"paint" of weighted links
between vertices and bones

153

Steps in the asset-creation pipeline: Rigging & Skinning (of a 3D mesh)



by **Digital modeller** (helped / replaced by automatic algorithms)

- Rigging :
  - define a skeleton (with a rest pose)
  - inside one mesh, (or a set of meshes: a *shared* skeleton)
  - also: define controls for animator
- Skinning (of a mesh):
  - painting link vertex-bones

rigger

skinner

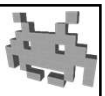
by **Ditigal animator**

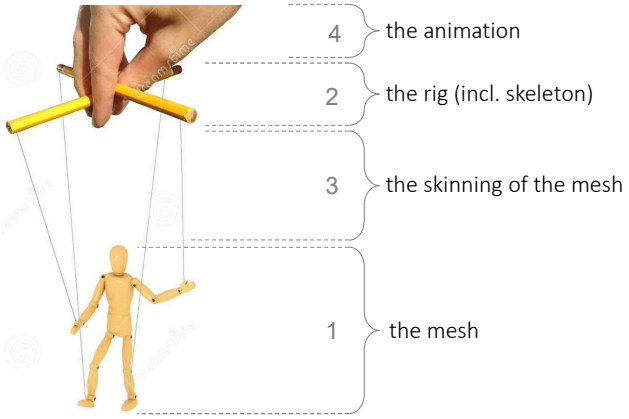
- Animation (of a control “rig”)
  - authoring of (skeletal) animations
  - see later

animator

154

Animation assets: a metaphor

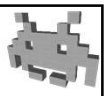


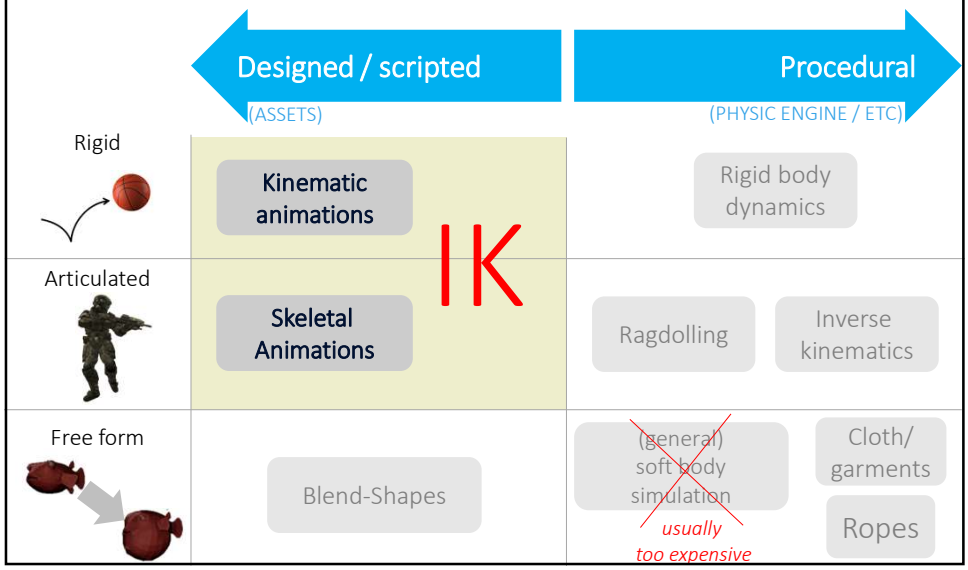


155

Inverse Kinematics (notes)

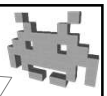
to learn more on IK, see course on → Virtual Reality





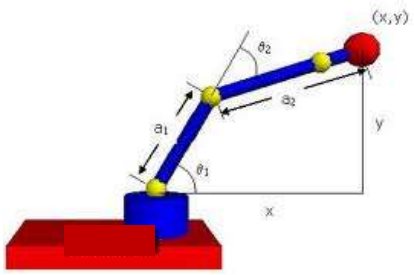
156

Inverse kinematic (IK)



to learn more on IK, see course on → Virtual Reality

- Intensively used in robotics



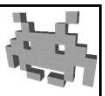
Forward kinematics:
«from angles (and dists) to (x,y)»

VS

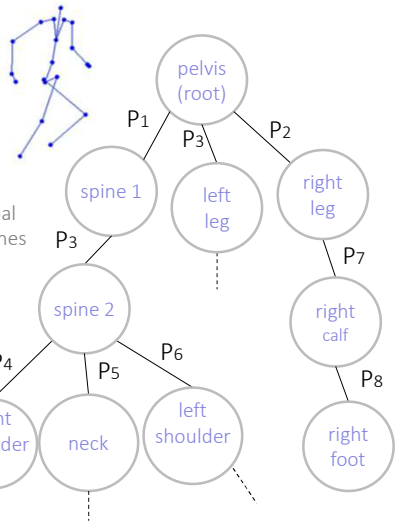
Inverse kinematics:
«from (x,y) (and dists) to angles»

157

Kinematics in skeleton animations

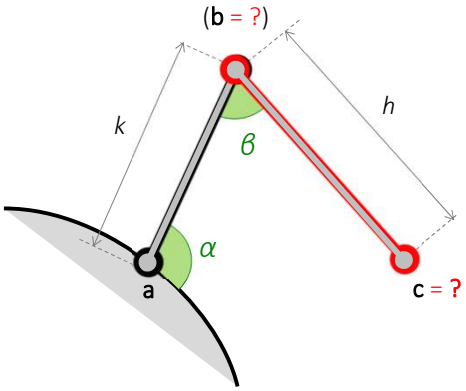


- Forward kinematics:
  - "given **local** transforms P1, P2… PN, (including rotations) where does the foot go?"
  - one solution
    - we already know how to find it! It's computation of global transforms from local ones
- Inverse kinematics
  - "if I need the right foot origin to be in positon **p**, how should I set the local transforms P1, P2… PN, ?"
  - under these constraints: …
  - 0, 1, or ∞ solutions, and not necessarily trivial



158

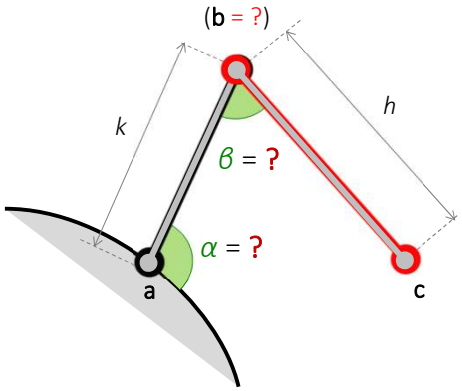
Forward Kinematic



find **c** (and **b**), given **a**, α, β, k, h

159

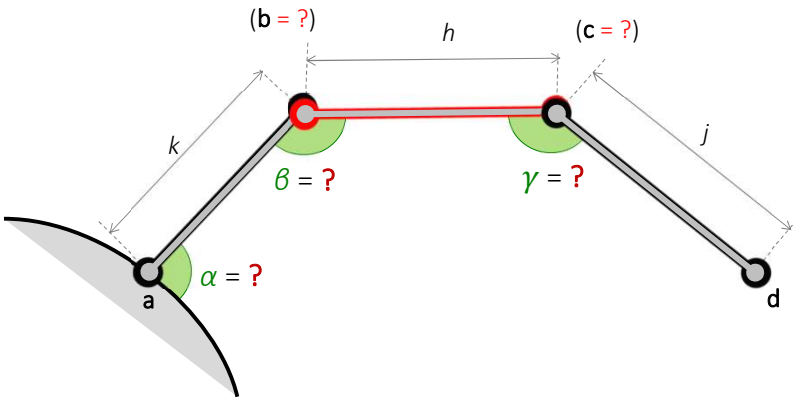
Inverse Kinematic (IK)



find α, β (and **b**), given **a**, **c**, k, h

162

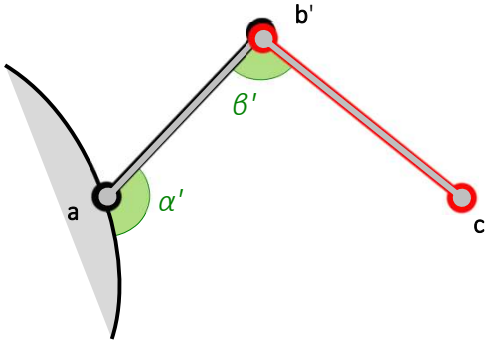
Inverse Kinematic (IK):
for any number of joints



solve for α , β , γ (and **b**, **c**), given **a**, **d**, k, h, j

165

Inverse Kinematic (IK): an ambiguity (in 2D)

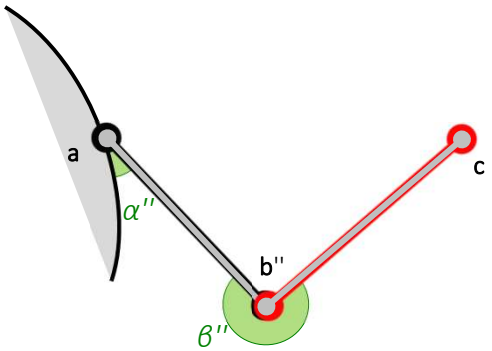


solution 1

solve for α, β (and **b'**), given **a**, **c**, *k*, *h*

166

Inverse Kinematic (IK): an ambiguity (in 2D)

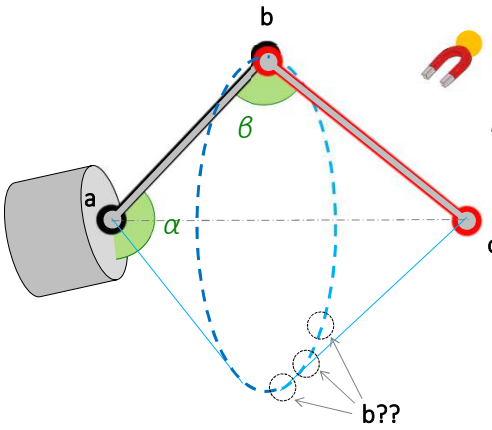


solution 2

solve for α, β (and **b**), given **a**, **c**, *k*, *h*

167

Inverse Kinematic (IK) in 3D: more ambiguities



e (extra input)
attractor for **b**
"please pick **b** *as close as possible to here"*
✓ disambiguates!

solve for α, β (and **b**), given **a**, **c**, k, h

168

Uses of IK (Inverse Kinematic) : during animation editing – or at run-time

- Direct kinematics
  - single solution exists
  - to find it: go from Local Transform to Global Transform, take its translation / position
- Inverse Kinematics
  - in general, it's more difficult to solve (as it's often the case with inverse problems)
  - but in practice, trivial problem instances are often all we need in Games
  - typical case: just two bones, one intermediate joint
    Articulated leg:
    pelvis joint → upper leg (thigh) → knee joint → lower leg (calf) → foot
    Articulated arm:
    shoulder joint → upper arm (brachium) → elbow joint → lower arm (forearm) → hand
  - still, infinite solutions exist for the pos of the intermediate joint: which one to pick?
  - disambiguate with an additional requirement:
    minimize distance of intermediate joint from its given "attractor" position
    (chosen by the human animator or by scripts or by heuristics)

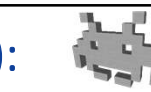
169

Uses of IK (Inverse Kinematic) :
during animation editing – or at run-time

- Examples of uses:
  - in preprocessing (helping the task of the animator)
  - in real time (performed by the game engine)
- Examples of real-time uses:
  - Exact positioning of character's feet on ground
  - Exact positioning of hand over object to be grabbed
  - Put the two hands together
    - e.g., if character wields a weapon with 2-hands
    - e.g., making the system auto-correct for small changes in bone lengths – helps animation retargeting
    - e.g., auto-correct interpolated frames
  - Make attack animation "connect" with target

170

Producing skeletal animations (1/4): procedurally



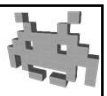
Regardless of its origins,
a skeletal animation (for a skeleton S)
controls a mesh (skinned over S)
in the same way

A central example:
using physics simulation

- Accounting for:
  - gravity, external forces,
    collisions with other objects,
    self-collisions (i.e., collision between different proxies associated to each bone)
- Requires:
  - Per-bone proxies (maybe for just a subset of the bones)
  - Constraints on, e.g., rotations (e.g., "knees don't bend backward or sideways", angular limits)
  - default constraint for the rigidity of pieces attached to bones
    (e.g. equidistance between connected bones to replicate the skeleton structure)
  - They can all be expressed as positional constraint in a Position Based Dynamics simulation
  - Physical data (per bone), e.g. mass, drag, moment of inertia…
- Computed on the fly
  - as part of the physics engine
  - as usual, can be baked into skeletal anims (e.g. to be retouched by animators?)
  - note: global transforms

171

Ragdolling (notes)



- Idea: let a physical simulation determine the evolution of the skeleton (and attached geom. proxies)
  - Includes: gravity, external forces, collisions with other objects, self-collisions
    (i.e., collision between proxies associated to the bones)
- Ingredients:
  - Per-bone proxies (in at least a subset of the bones)
  - Constraints, such as…
    attachments of bones, constraint on rotations
    (e.g., "knees don't bend backward or sideways")
  - The latter can be expressed as positional constraint in a Position Based Dynamics simulation
- Result: procedural skeletal animation!

172

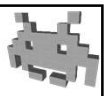
Producing skeletal animations (1/4): procedurally

Using physics simulation to control…

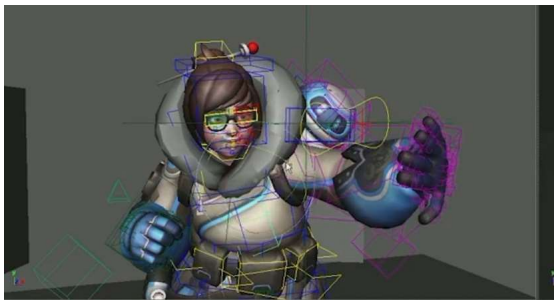
- the entire skeleton? ("rag-dolling")
  - Result: (suddenly) defunct / unconscious characters falling like a… rag-doll (or a sack of potatoes)
  - Problem: consciousness it's very yes/no;
    Intermediate states not trivial (but attempted in several ways)
- only a few peripheric bones? ("secondary animations")
  - The rest of the bones are still controlled by an animation assets
  - Examples, used to control…
    "Hair bone(s)" (the ones linked character's wig)
    "Cloth bones" (the ones linked to pieces of garment)
  - Result: whatever the character does, the hair braid / cape / etc follows

173

Producing skeletal animations (2/4): manually



- Authored by animators (digital artists) using manual keyframe editing



img by Blizzard Entratainement

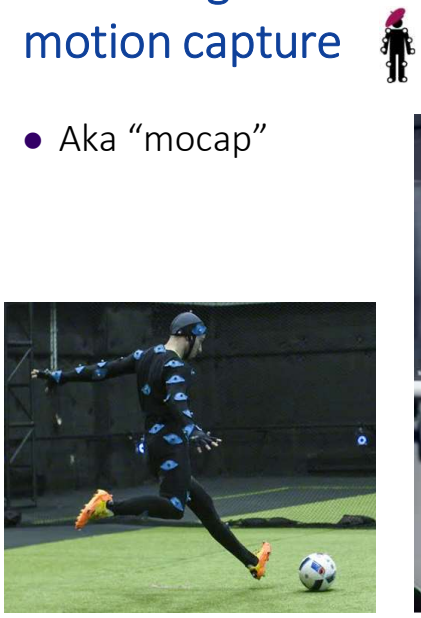
174

Manual Keyframe editing

- Task performed by a digital animator
  - The artist poses the character in every keyframe
- As usual for animations: using a timebar
  - artist inserts / removes keyframes (where in-betweens look bad),
  - author can edit the transition functions between them
  - etc.
- A control "rig" = set of GUI control to ease the task to pose individual keyframes
  - Task to construct the rig = rigging the model
- Rig is made of …
  - the skeleton, rest pose included
  - a set of constraint (e.g. knee don't bend backward)
  - the specification of a GUI
  - IK mechanism (eg: artists positions the hands, positions the attractor for the elbow. The IK produces the shoulder / wrist / elbow rotations)
  - controls such as gaze direction (control eyeball bones)
  - Sliders to control blend-shapes
    (they can be used in conjunction with skeletal animations! see later)

175

Producing skeletal animations (3/4): motion capture



- Aka “mocap”

176

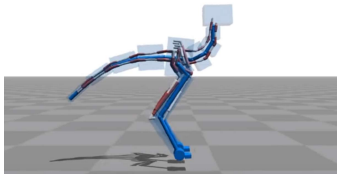
Motion capture

- Requires heavy setup (but cheaper/easier every year)
  - Markers / suits
  - Controlled cameras
  - Studio
  - Action must take space in a working space
- Requires skilled actors / performers / athletes
- Can be used to capture
  - single animations (a football stunt, walking, running)
  - joint performances by a group of actors (e.g. for cutscenes)
- Requires postprocessing (automatic or not)
  - cleanups (often, substantial)
  - Manual re-touces (e.g. adding animations for smaller bones, e.g. fingers)
  - extraction of keyframes (removal of in-betweens)

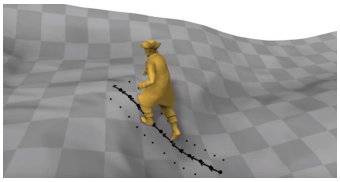
177

Producing skeletal animations (4/4): With Machine Learning

- A very active area of research…



Flexible Muscle-Based Locomotion for Bipedal Creatures
Thomas Geijtenbeek, Michiel van de Panne, A. Frank van der Stappen
SIGGRAPH 2013

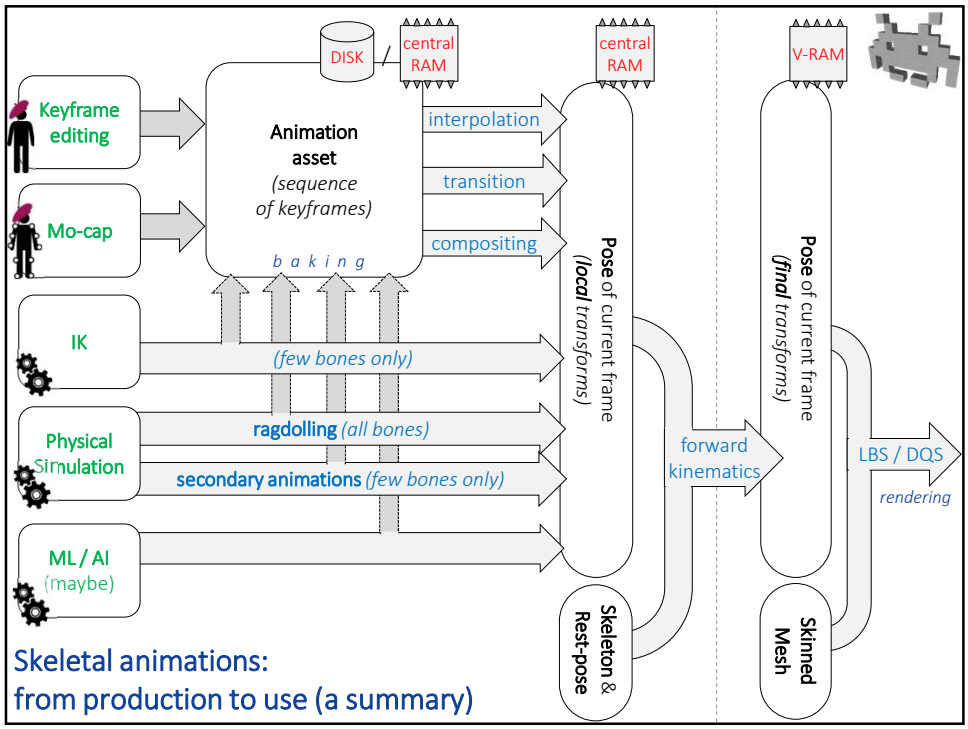


Phase-Functioned Neural Networks for Character Control
Daniel Holden, Taku Komara, Jun Saito
SIGGRAPH 2017

For example, see a demo:
https://happydagger.itch.io/human-sim-demo
https://ai.meta.com/research/publications/zero-shot-whole-body-humanoid-control-via-behavioral-foundation-models/

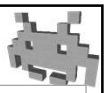
(among MANY others)

178



180

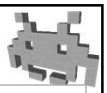
Transforms per bones (for a pose) [summary]



LOCAL	"FINAL" (not a fully established term)
Def: from space of child bone *to* space of parent bone	*Def: from* character space in rest pose *to* character space in target pose (character space = space of the skeleton root)
Stored in the keyframes of the animation assets	*Stored* in V-RAM when a skinned mesh is rendered
Manipulated during: animation transitioning, animation layering, by scripts (e.g., to make the head look toward s.t.)…	Computed(*) from the local transfs. of current pose and the (inverses of) local transfs. of rest pose, according to the hierarchy of bones in the skeleton. (*) can be baked of course (see next slide).
Can be only interpreted with a skeleton (hierarchal tree of bones)	Once computed, independent from skeleton hierarchy! (or its rest pose)
Interpolated (for the same bone) during: EASY keyframe interpolation, animation transitioning, animation layering…	*Interpolated* (among different bones) TRICKY during blend skinning (for vertices associated to multiple bones).
Representable in any way; typically, as we know: a scaling (if needed, usually not) *followed by* a rotation (of the joint!) *followed by* a translation: a vector (in the space of the parent!) positioning the joint w.r.t the parent bone	*Representable* as: a 4x4 matrix (LBS) or a <= candy wrapper effect dual-quaternion (DQS). <= no support for rescaling (other formats don't support a sufficiently good interpolation)
Reflects the status of a single joint ("how bent is the knee?"	*Reflects* the combined effects of this joint all joints preceding this one in the skeleton (to the root)

181

Transforms per bones (for a pose) [summary]



LOCAL	"FINAL" (not a fully established term)
The (uniform) scaling component usually not needed, unless animations enlarges / shrinks / elongate bones! (not supported by DQS)	What if store them instead in RAM too? That is, animations = sequence of poses described as final transforms per bone? (e.g. compute and bake them in preproc) [sometimes done, e.g. Geo Caches later] The good: * ready to use – just upload to VRAM * avoids need to composite local transforms into final ones at each frame * no need for a skeleton anymore! (no hierarchy) The bad: we lose the ability to… * interpolate poses (no keyframes and in-betweens, no animation transition) * composite anims, etc
The rotation component be stored as… * Euler angles, in file exchange formats (for maximal compactness!) * Quaternions, in RAM (for good iterpolability)	
The translation component is * always the same for all poses (for rigid, i.e. not elongating bones). * thus, typically not stored in anims, inherited by the skeleton rest-pose * except for the first "real" bone, (e.g. "pelvis") the child of the "base" bone	

182