



## 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 **Textures** ●●
- lec. 9: Game **Materials** ●
- lec. 8: Game **3D Animations** ●●● (with a red arrow pointing to the yellow dot)
- lec. 10: **Networking** for 3D Games ●
- lec. 11: **3D Audio** for 3D Games ●
- lec. 12: **Rendering Techniques** for 3D Games ●
- lec. 13: **Artificial Intelligence** for 3D Games ●

93

## $N_{max}$ = How many bone links for each vertex



- It's a call of the Game engine!
- typical used value:
  - 1 (non-blended skinning) (bonus: no need to store weights)
  - 2 (cheap, e.g., for mobile games)
  - 4 (top quality – standard)
  - more: never in games (currently)
- Can one lower  $N_{max}$  ?
  - yes, in preprocessing
  - e.g., task for a game tool
  - e.g.: Unity does this during skinned mesh import (if asked to)

98

## (but why put a hard-wired bound on the number of bone links?)



- Reduces performance cost
  - $N_{max}$  transforms need be interpolated in GPU
    - in vertex shader
  - GPU = no good at control:
    - always uses exactly  $N_{max}$  trasform
    - unused bones: weight = 0
- Reduces VRAM footprint
  - reduces storage
  - fixed length arrays: good for GPU
    - $N_{max}$  (index,weight) pairs
    - even where fewer are locally needed (e.g., if 1 bone, weight is automatically 1)

example:

Bone Index	Weight
9 (Head)	1.0
--	0.0
--	0.0
--	0.0

99

## 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

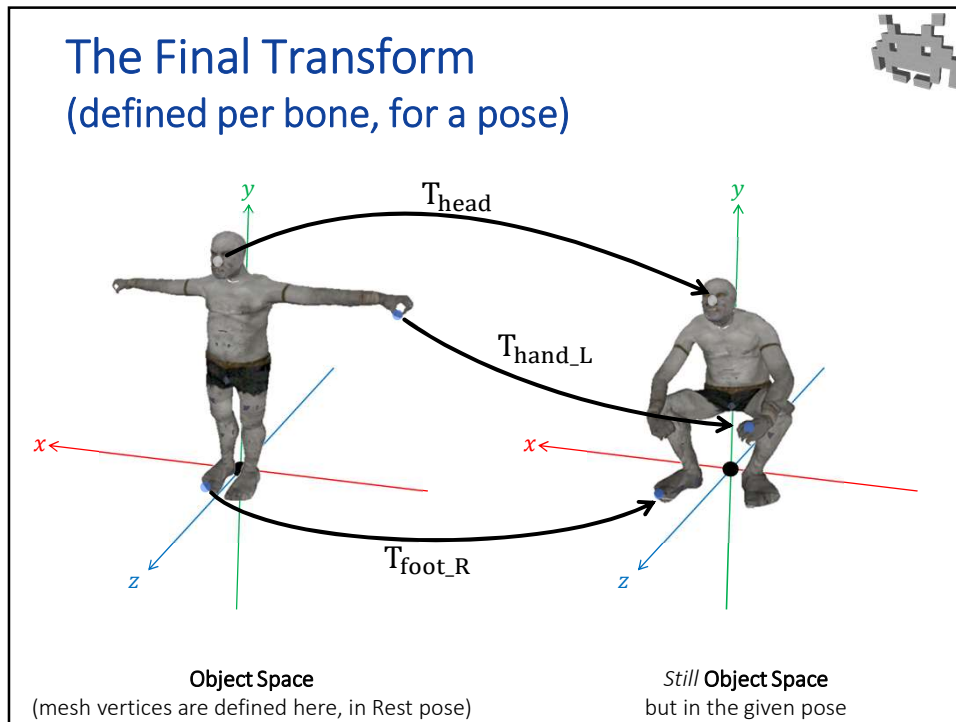
includes **skinning (bone weights)**

THE MESH ASSET

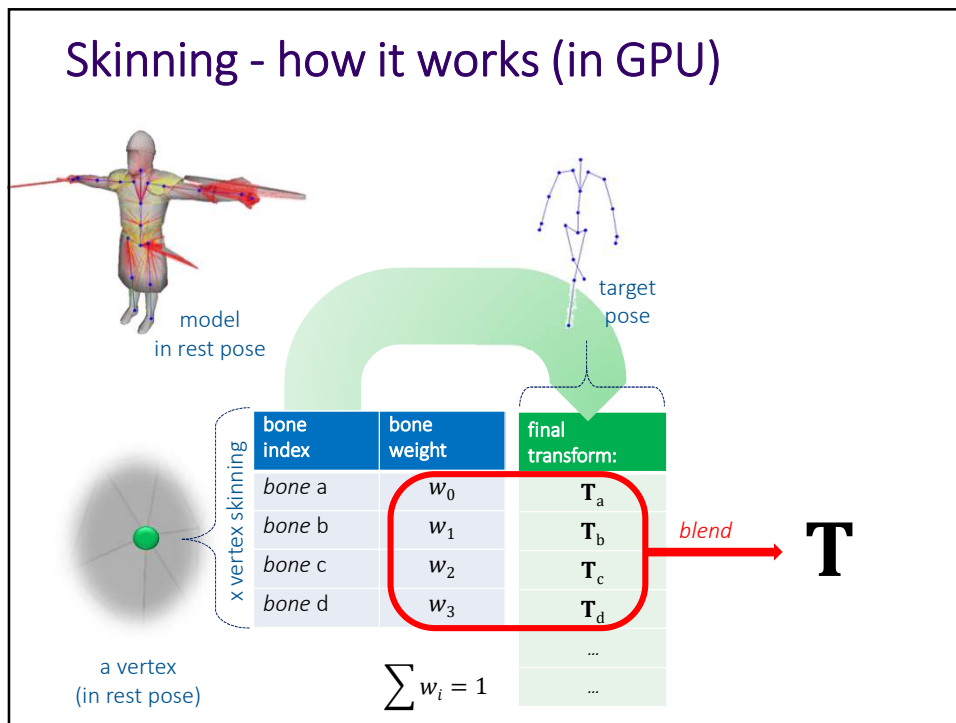
THE ANIMATION (current frame)

THE MATERIAL ASSET

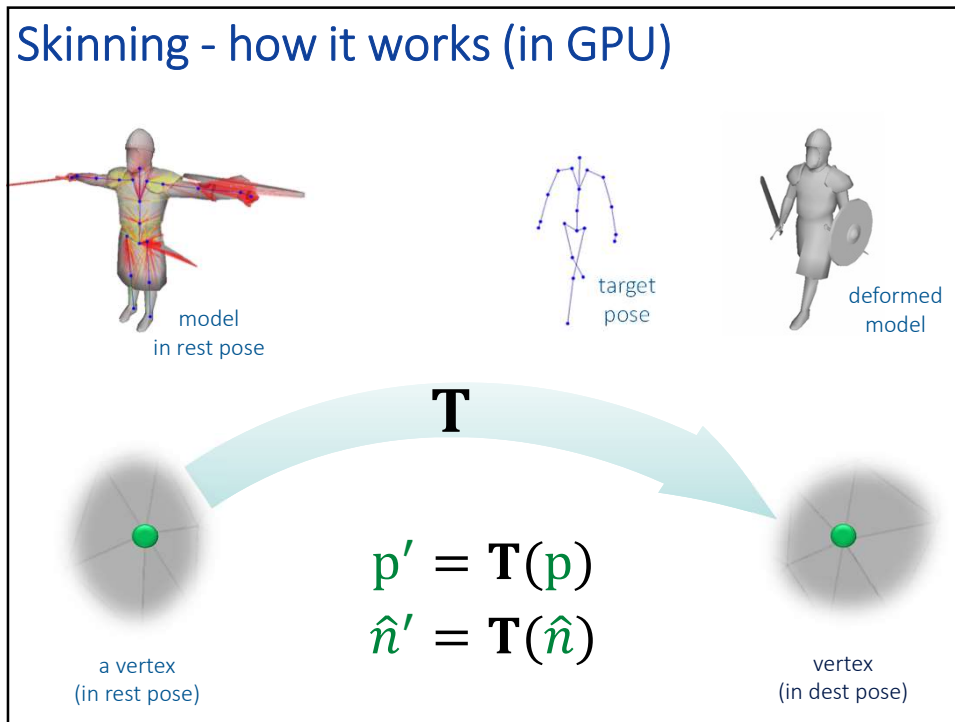
100



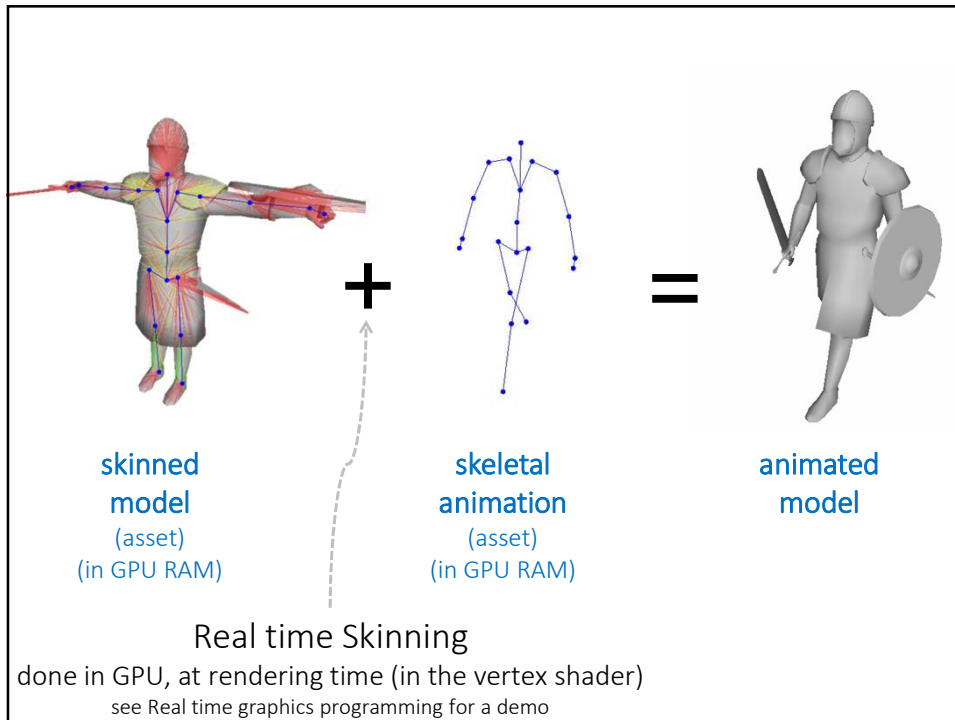
101



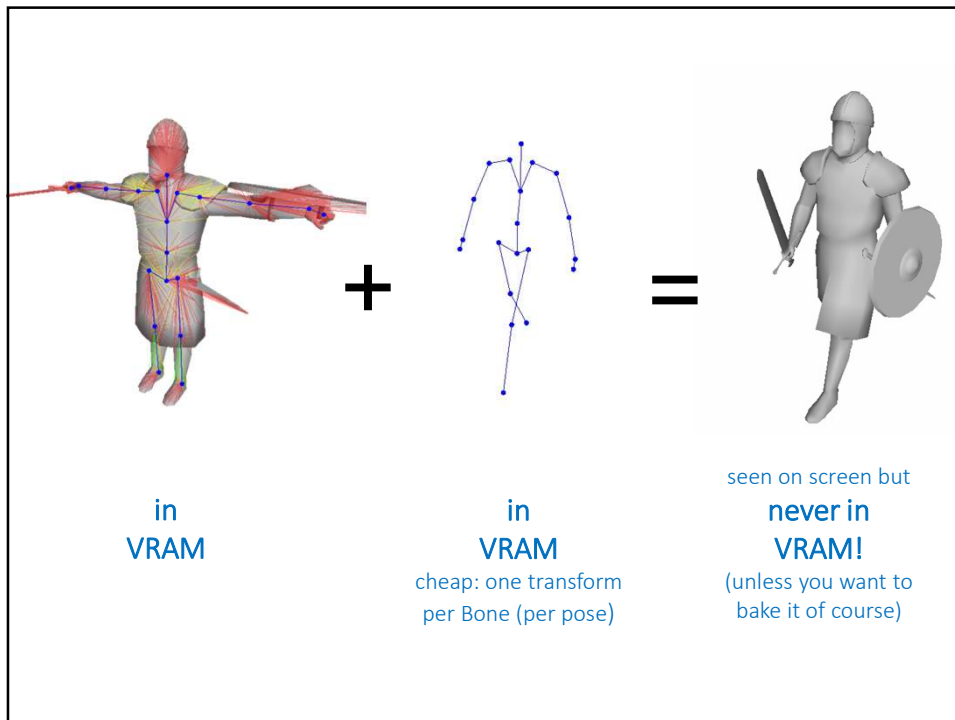
102



103



104



105

### GPU real time Skinning – variants

(a choice of the rendering engine)

Bone	Weight	Final Transform
bone a	$w_0$	$T_a$
bone b	$w_1$	$T_b$
bone c	$w_2$	$T_c$
bone d	$w_3$	$T_d$

$\xrightarrow{\text{blend}}$  **T**

how are they stored?

how is this done?

<p><i>Answer 1:</i> «Linear Blend Skinning»</p>	<p>as a 4x4 matrix transform</p>	<p>with linear matrix interpolation</p>
<p><i>Answer 2:</i> «Dual Quaternion Skinning»</p>	<p>as a dual quaternion</p>	<p>with dual quaternion interpolation</p>

*nothing else works!*

106

### Linear Blend Skinning (LBS)

more in general, Nmax-1

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

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

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

interpolation of per-bone transformed points

rest position of the vertex  $\mathbf{p}_R$

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  $\mathbf{p}_P$

107

### Dual Quaternion Skinning (DQS)

Interpolated DUAL QUATERNION (still a rigid transform)

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

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  $\mathbf{p}_R$

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  $\mathbf{p}_P$

108


## Representations for roto-translations (recap)

← see lecture on transform representation

- 3×3 Matrix
- Euler Angles
- Angle + Axis
- Quaternion

+ Translation  
(displacement vector)

- 4×4 Matrix (or 3×4) ← Solution 1 (LBS)
- Dual Quaternion ← Solution 2 (DQS)




109

## Dual Quat: intuition on why they work

(but storing translations and rotation separately doesn't)

- Problem with storing & interpolating  
Rot. and Transl. of FINAL transforms separately:
  - a final trans 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
  - 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”



110

## Affine matrix: intuition on why they work (but storing translations and rotation separately doesn't)



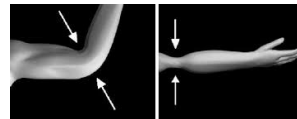
- If final transformation are expressed (and interpolated) as Affine matrices, then “everything is linear”:
- Interpolating the matrices is equivalent to interpolate the transformed points (see formulas)
  - Nothing can go wrong
- Problem: interpolation of rotation matrices does not produce a rotation (as we know)
  - A unwanted shear and scale-down is introduced
  - Therefore, LBS can “shrink” the object a bit while deforming it from rest pose to current pose

111

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



- LBS ...
  - Is a bit cheaper to compute  
Can shrink surfaces a little  
(look for: candy wrapper effect)
  - Can also express (uniform) scaling in per-bone transformations  
(local ones, therefore final ones)
  - Is older, more established
- DQS ...
  - costs some 50% more FLOP operations per vertex  
(but depends on implementation details)
  - Works better, avoid candy wrapper effects
  - May have the opposite defect of *enlarging the volumes*.
  - Can only express rigid motions



Both are used.

They use the exact same set of ASSETS! (skinned mesh, skeletons, anims)

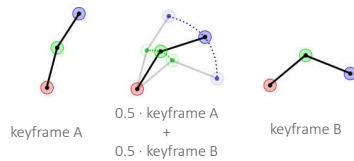
112



## Keyframes and in-betweens in a skeletal animation



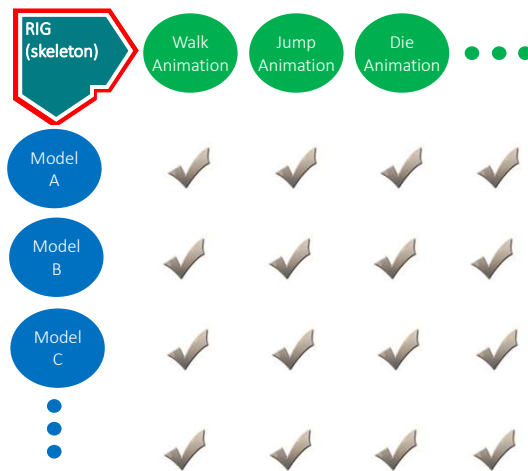
- Poses in a skeletal animations can be easily mixed (always, by blending **local** per-bone transforms)
  - This interpolation is very **expressive**:
    - 2 (or more) very different key-frames can be blended with good results



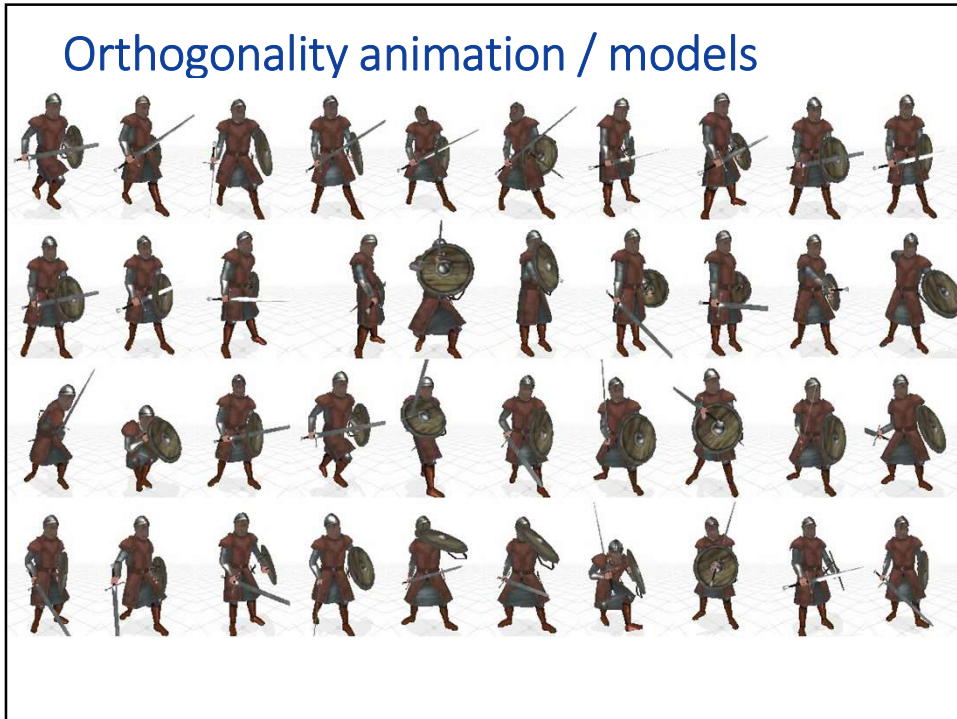
- much superior interpolation power than, for example, blend-shapes!
- Keyframes can be very far apart
- e.g.: decent walk-cycles with just 4 key-frames! (2 per step, mirrored)
- e.g.: decent attack animations with just 2 key-frames! (charge, discharge)
- (better results can always be obtained inserting new key-frames, as usual)

116

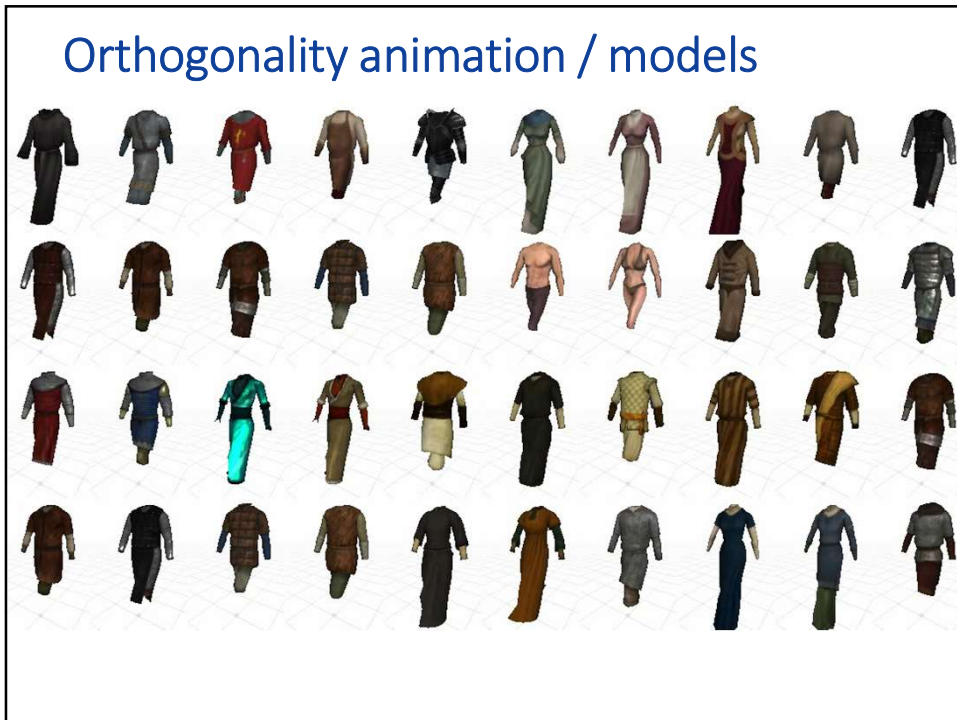
## Orthogonality animation / models



117



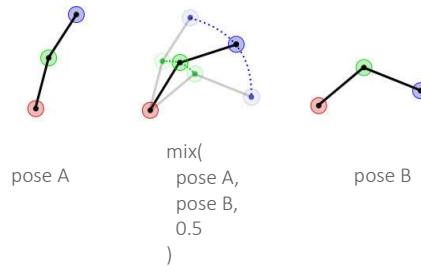
118



119

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

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

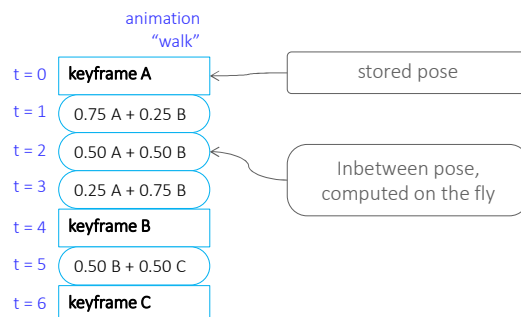


- as long as they are defined on the **same rig**
- mix is defined by interpolating the per-bone **local** transform
- this requires re-computation of **final** transforms (after interpolation)

120

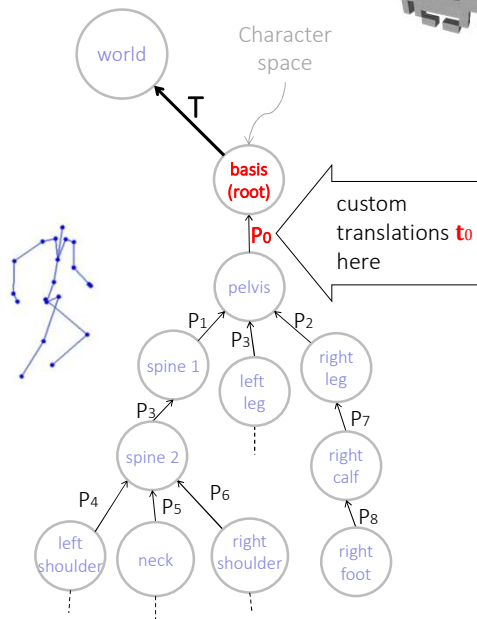

## Pose = keyframe

- Compress animations



121

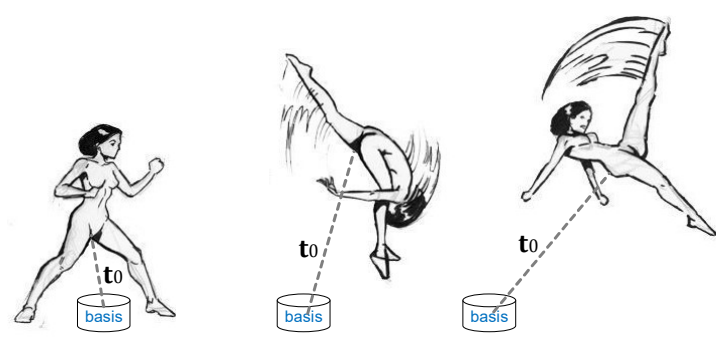
### The root is usually an abstract "basis" bone



The diagram illustrates a skeletal hierarchy. At the top is a 'world' node. Below it is the 'basis (root)' node, which is highlighted in red. A translation vector  $T$  connects 'world' to 'basis (root)'. Below 'basis (root)' is the 'pelvis' node, with a translation vector  $P_0$  pointing to it. A callout box points to  $P_0$  with the text 'custom translations to here'. Below 'pelvis' are 'spine 1' and 'right leg'. Below 'spine 1' is 'spine 2'. Below 'spine 2' are 'left shoulder' and 'neck'. Below 'right leg' is 'right calf'. Below 'right calf' is 'right foot'. Translation vectors  $P_1$  through  $P_8$  connect the parent nodes to their children.

122

### Basis bone



keyframe 1      keyframe 2      keyframe 3

so that each animation asset can include a global displacement  $t_0$  in each keyframe

the basis bone is (normally) the only one redefining the *translation* in the rest pose!

123

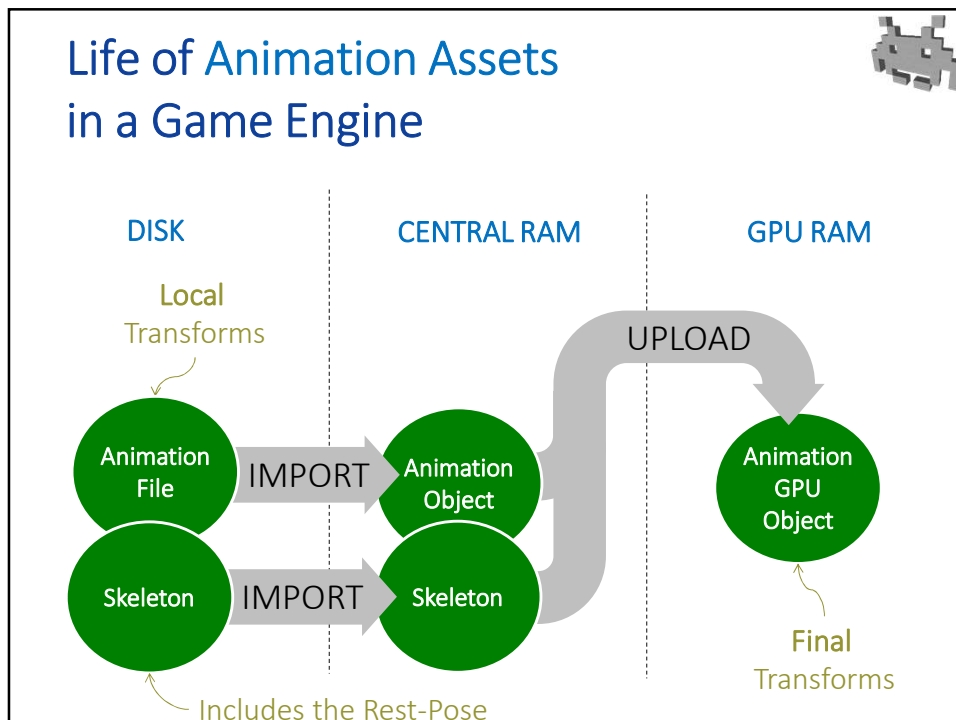
(recap) **Skeletal animations:**  
**3 types of Assets (data structures)**

- **Skeleton** (or “rig”)
  - Tree of **bones**
  - $\forall$  **bone** => reference frame (in rest pose)
    - reference frame root bone = object space
- **Skinned 3D Models**
  - Mesh with links: **vertices** => **bones**
  - $\forall$  vertex: attributes: [ **bone index** , **weights** ] x  $N_{max}$
- **Skeletal animations**
  - Sequence of keyframe **poses**
  - $\forall$  pose,  $\forall$  **bone** = a **local** transform

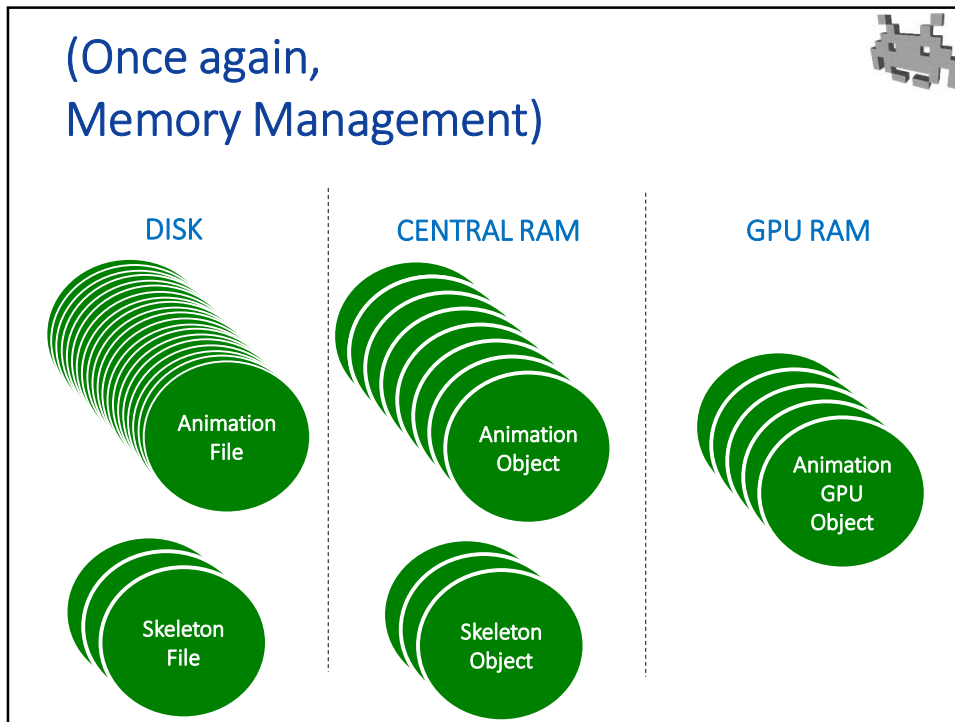
examples of interchange formats (for all three):

- **.SMD** (Valve), **.FBX** (Autodesk), **.BVH** (“behaviour” Biovision)

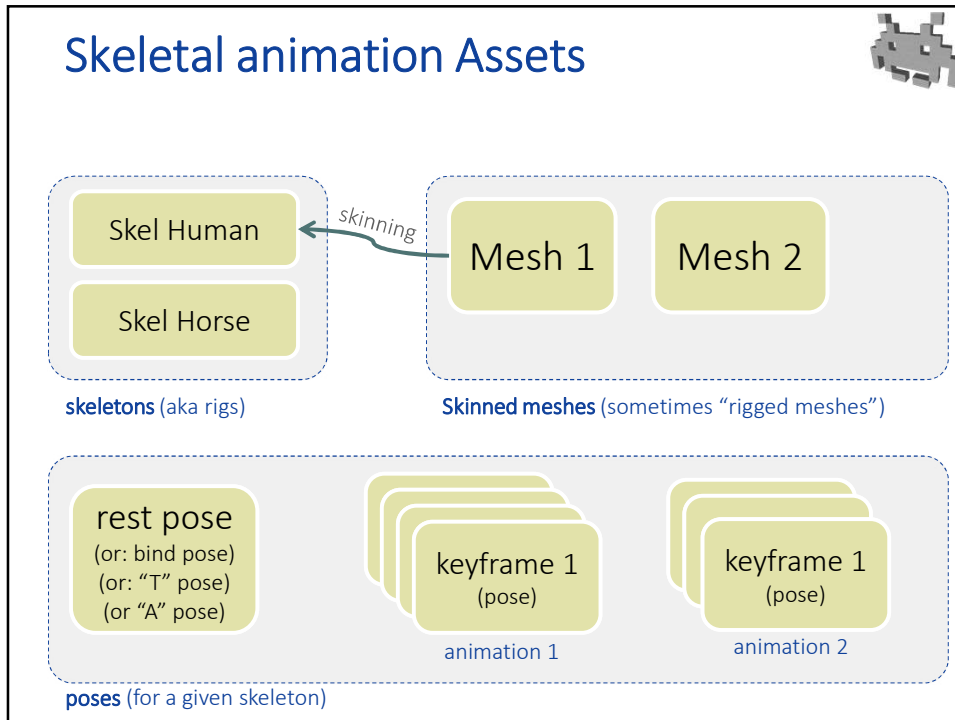
124



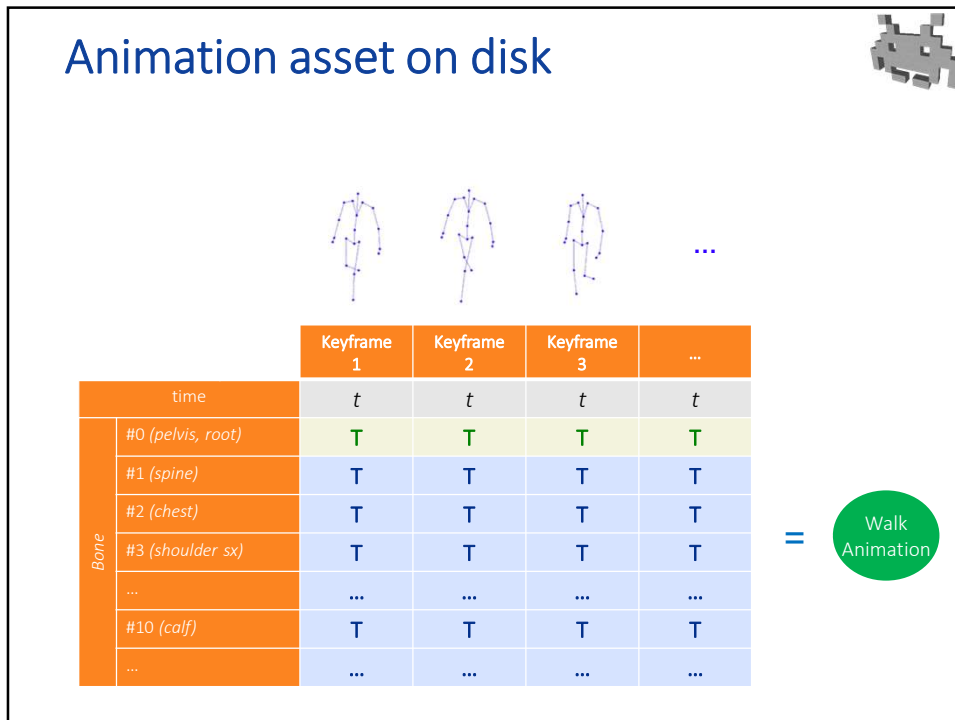
125



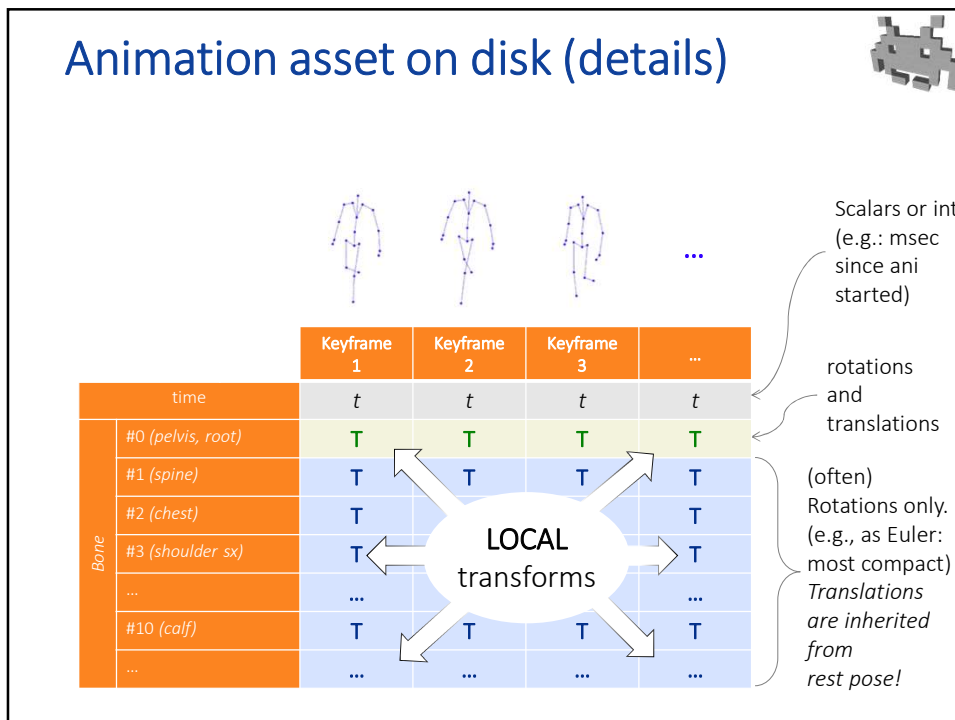
126



127



128



129

## Animation asset on disk (details)

T : rotation + translation  
T : rotation only  
✗ : not stored (interpolated)

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 store every transformation is stored: (the other ones are not stored are interpolated between prev and next)

130

## Animation asset on disk (details)

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

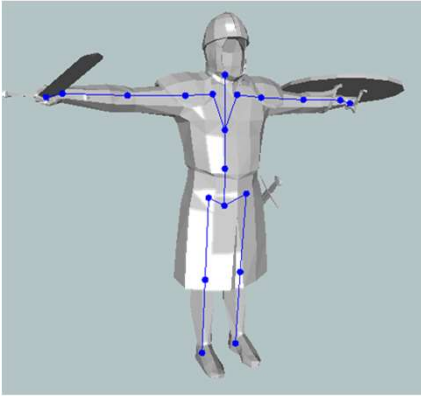
rest pose      walk animation (cycled)      run animation (cycled)      idle animation (cycled)

		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		$t$	$t$	$t$	$t$	$t$	$t$	$t$	$t$	$t$	$t$	$t$	$t$
Bone	#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	✗
...	...	...	...	...	...	...	...	...	...	...	...	...	

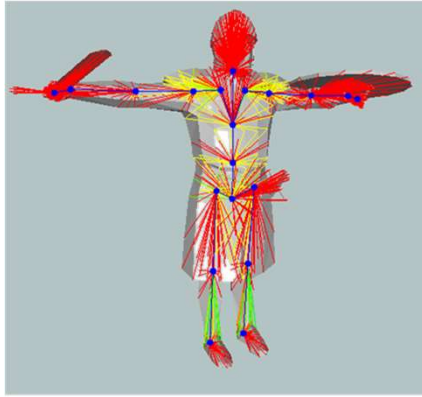
131



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



**Rigging – authoring of a rig**  
 defining the skeleton  
 (often: also of the controls  
 to define poses for it)



**Skinning – authoring of the skinning**  
 “paint” of weighted links  
 between vertices and bones


132

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


by Digital modeller  
(helped / replaced  
by automatic algorithms)

by Digital  
animator


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



*rigger*



*skinner*



*animator*

133

## Animation assets: a metaphor

4 } the animation  
2 } the rig  
3 } the skinning of the mesh  
1 } the mesh

134

## Pose = keyframe

- Compress animations

animation "walk"

t = 0	keyframe A	← stored pose
t = 1	0.75 A + 0.25 B	
t = 2	0.50 A + 0.50 B	← Inbetween pose, computed on the fly
t = 3	0.25 A + 0.75 B	
t = 4	keyframe B	
t = 5	0.50 B + 0.50 C	
t = 6	keyframe C	

135

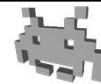
## Dynamically combining different animations



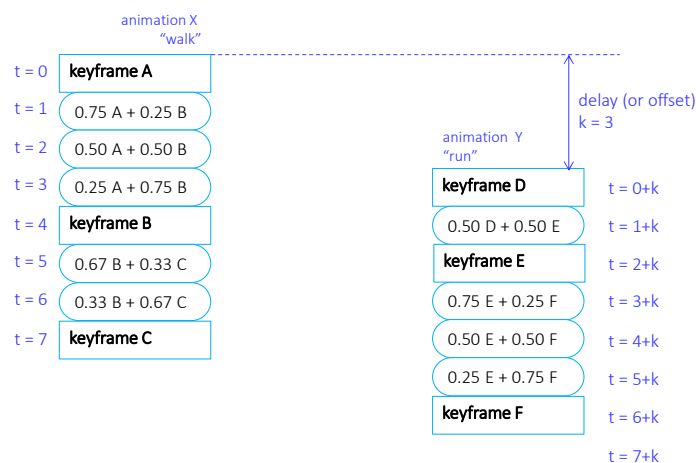
- Poses in a skeletal animations can be easily blended...
  - As long as they share the same skeleton
- ... so, entire animations can be combined into new animations!
  - In real time, during game execution
  - Remember results can always be *baked*, and edited, during asset production
- Two ways to do combine animations:
  - **Transitions** between two animations (or more)
  - **Compositing (layering)** two animations (or more)

136

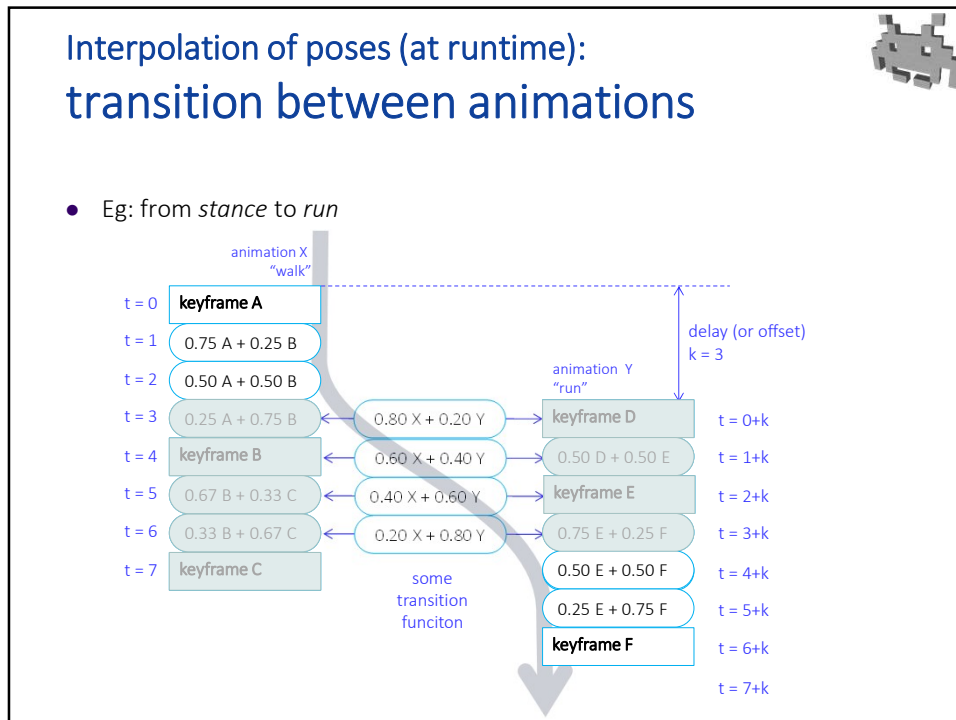
## Interpolation of poses (at runtime): transition between animations



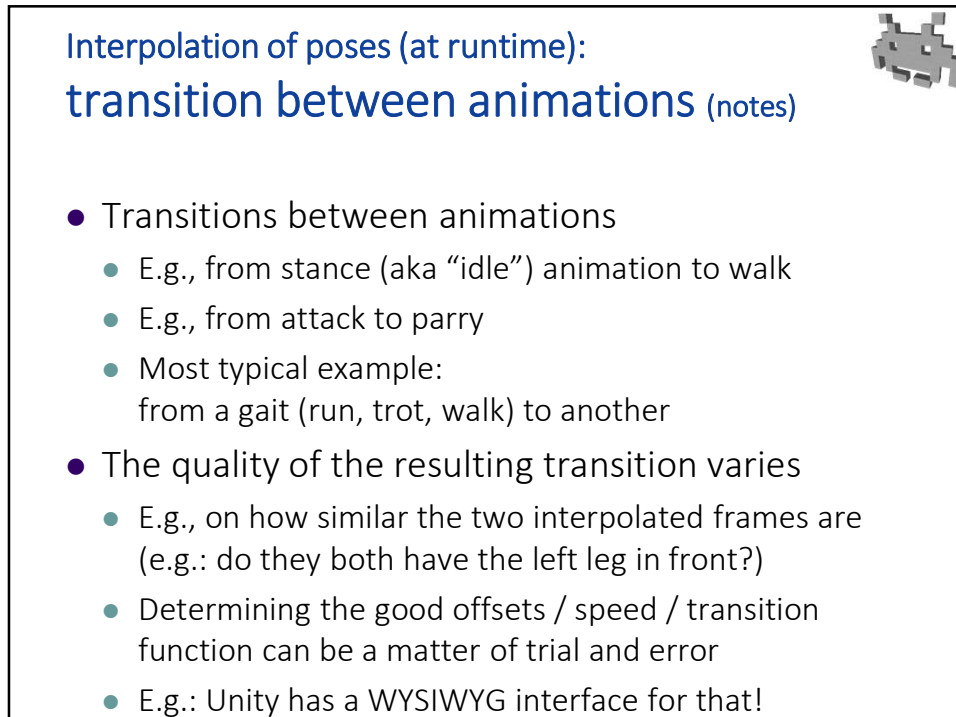
- Eg: from *stance* to *run*



137



138



139

### Compositing (layering) poses (→ and animations)

The diagram illustrates the process of compositing two poses. Pose A (left) is circled in blue and labeled 'lower joints'. Pose B (middle) is circled in green and labeled 'upper joints'. The resulting 'New Pose' (right) is a combination of the two. Below each pose is a skeletal hierarchy diagram with joints labeled P1 through P12. The 'New Pose' diagram shows the joints from both poses combined.


140

### Compositing (layering) poses (→ and animations)

The diagram illustrates the process of compositing two poses. Pose A (left) is circled in blue and labeled 'lower joints'. Pose B (middle) is circled in green and labeled 'upper joints'. The resulting 'New Pose' (right) is a combination of the two. Below each pose is a skeletal hierarchy diagram with joints labeled P1 through P12. An equation is shown:  $P_1 = 0.45 \cdot P_1 + 0.55 \cdot P_1$ .

141

## 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:
    - is the local animation of this bones “overwritten” by this animation?

142

## Inverse Kinematics (notes)

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



	← Designed / scripted <small>(ASSETS)</small>	→ Procedural <small>(PHYSIC ENGINE / ETC)</small>
Rigid 	Kinematic animations	Rigid body dynamics
Articulated 	Skeletal Animations	Ragdolling    Inverse kinematics
Free form 	Blend-Shapes	<del>(general) soft body simulation</del> <i>usually too expensive</i>

IK

143

## Kinematics in skeleton animations

- **Forward** kinematics:
  - “given **local** transforms  $P_1, P_2 \dots P_N$ , where does the foot go?”
  - **one solution** ← you already know how to find it!
- **Inverse** kinematics
  - “if I need the foot to be in pos  $p$ , how should I set local transforms  $P_1, P_2 \dots P_N, ?$ ”
  - (under these constraints)
  - **0, 1, or  $\infty$  solutions, and not trivial**

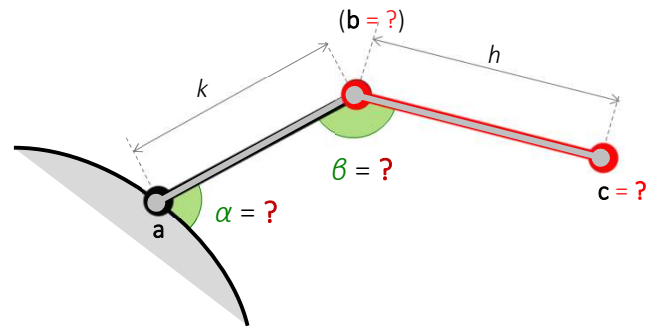
145

## Forward Kinematic

find  $c$  (and  $b$ ), given  $a, \alpha, \beta, k, h$

146

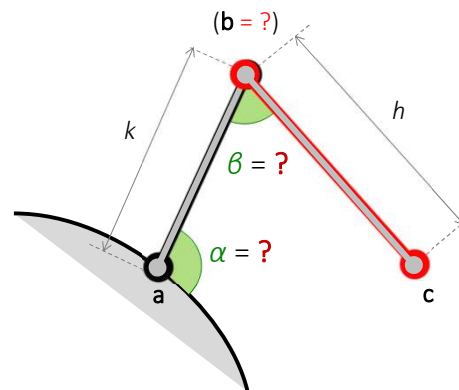
## Forward Kinematic



find  $c$  (and  $b$ ), given  $a, \alpha, \beta, k, h$

148

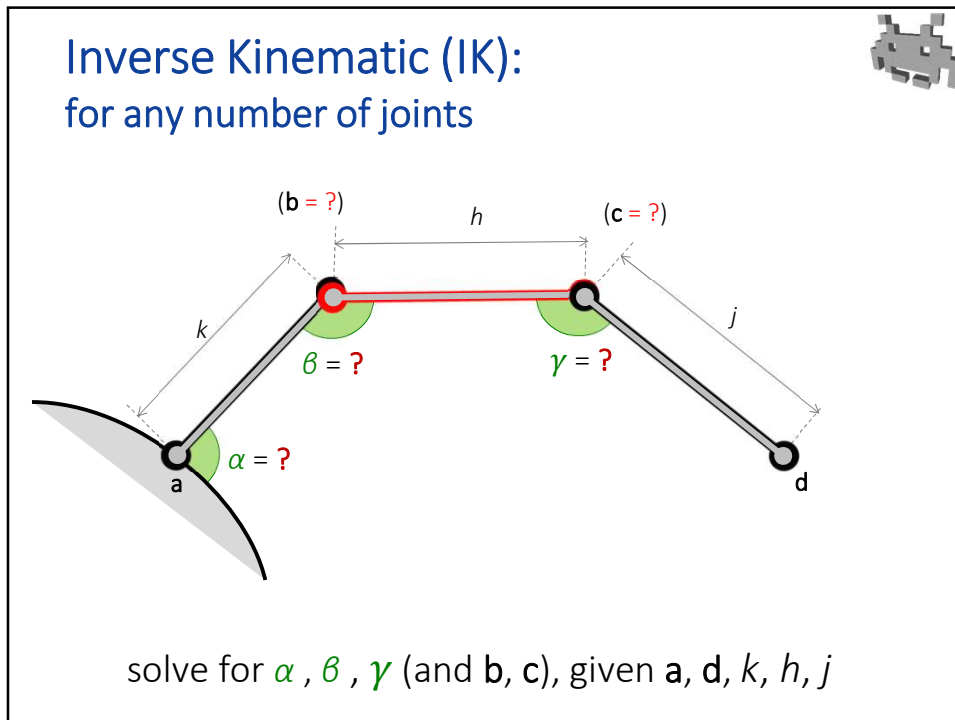
## Inverse Kinematic (IK)



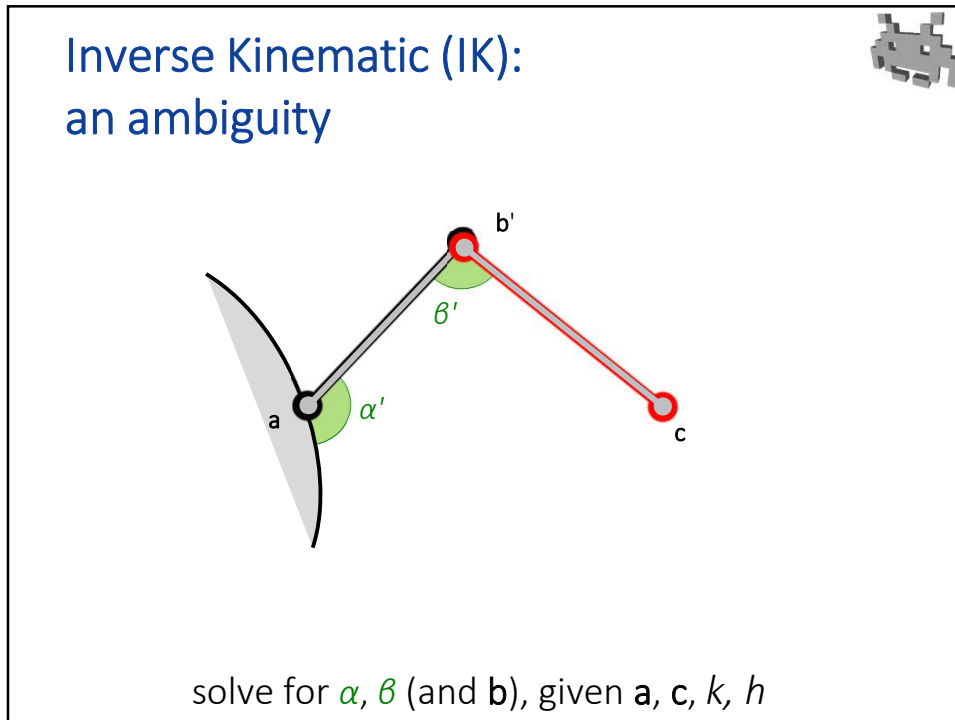
find  $\alpha, \beta$  (and  $b$ ), given  $a, c, k, h$

149



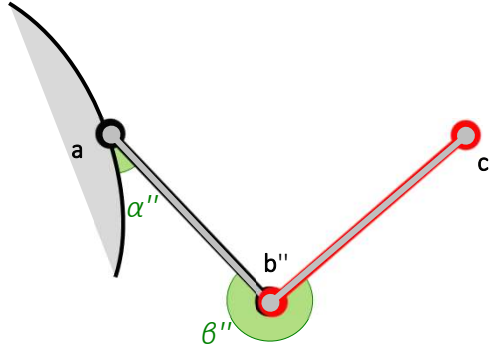


152



153

### Inverse Kinematic (IK): an ambiguity

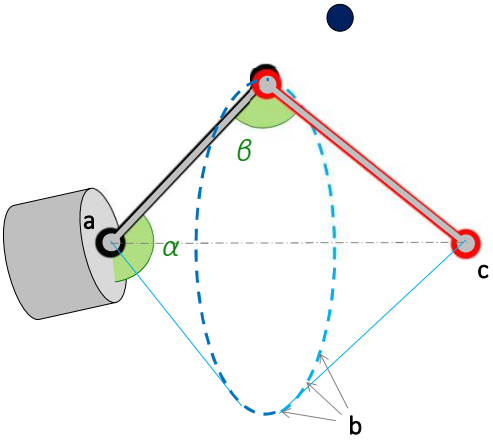


The diagram shows a 2D kinematic chain. A grey slider joint is attached to a vertical plane at point  $a$ . A grey link of length  $k$  connects point  $a$  to point  $b''$ . The angle between the link and the normal to the plane is  $\alpha''$ . A red link of length  $h$  connects point  $b''$  to point  $c$ . The angle between the red link and the normal to the plane is  $\beta''$ . A green circle is centered at  $b''$ .

solve for  $\alpha, \beta$  (and  $b$ ), given  $a, c, k, h$

154

### Inverse Kinematic (IK) in 3D: more ambiguities



The diagram shows a 3D kinematic chain. A grey revolute joint is attached to a cylinder at point  $a$ . A grey link of length  $k$  connects point  $a$  to point  $b$ . The angle between the link and the axis of the cylinder is  $\alpha$ . A red link of length  $h$  connects point  $b$  to point  $c$ . The angle between the red link and the axis of the cylinder is  $\beta$ . A blue dashed ellipse represents the possible positions of point  $b$ . A blue dot is shown above the chain.

solve for  $\alpha, \beta$  (and  $b$ ), given  $a, c, k, h$

155

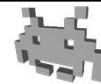
## Inverse Kinematic (IK): useful in editing, and at run-time



- Direct kinematics
  - Single solution exists
  - An example is going from Local Transform to Final Transform
- Inverse Kinematics
  - it's more difficult to solve (as it's often the case with inverse problems!)
  - Often, trivial solutions are all that we need in Games: e.g., just two bones (for articulated legs, or arms)
  - Multiple solutions exists: which one to pick?
  - Disambiguate with additional constraint, such as: minimize the distance from the intermediate joint to a given "attractor" (a position)

156

## Inverse Kinematic (IK): useful in editing, and at run-time




- Many 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 feet on ground
  - Exact positioning of hand to object to be grabbed
  - The two hands need to be joined
    - 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
  - Helps attack animation "connect" with target



157

## Authoring / producing skeletal animations (1/3)




- Using **physics simulation** 
  - Dynamically, by the physics engine
  - The results can always be baked
  - How to: link skeleton links with constraints (e.g. with Verlet)
    - Equidistance constraints (between connected joint)
    - Additional constraint on angles (e.g. “keens don’t bend backward”)
- Done in real time (dynamically) for (typically defunct / unconscious) characters is referred to as **Rag-dolling**
  - Clearly, results can always be baked (and re-edited)

158

## Authoring / producing skeletal animations (1/3)



- **physics simulation** 
  - Can be in control of transforms associated to “secondary” bones (“secondary animations”)
  - For example: “hair bone” (controlling character’s wig)

159

## Authoring / producing skeletal animations (2/3)


- Manual **keyframe editing** 



img by Blizzard Entertainment


160

## Manual Keyframe editing

- Task performed by a digital animator 
- As usual: using a **timebar**
  - inserting / removing keyframes,
  - editing transition functions between them,
  - etc.
- Using a “**rig**” to pose individual keyframes
- In this context, rig = not only the “skeleton” itself, but also:
  - a set of constraint
  - a complete GUI made ready to ease the task by the animator, including controls for...
  - IK (GUI for positioning hands/limbs, attractor nodes for knees/elbows)
  - gaze direction (control eyeball bones)
  - blend-shapes (they can be used in conjunction with skeletal animations! see later)

161


## Authoring / producing skeletal animations (3/3)

- Motion capture (“mocap”) 



162

## Motion capture

- Requires heavy setup (maybe not in the future?)
  - 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 (for cutscenes)
- Requires postprocessing
  - cleanups
  - extraction of keyframes (removal of in-betweens)

163

## Some of the benefit of skinning (recap)

- Animations and models being animated are reciprocally orthogonal
  - Can be produced / stored independently
  - 100 animations for 100 models: 100+100 assets, *NOT* 100x100 assets
  - Possible to retarget animations over different models
- Models appear deformed by pose on screen, but only models and animations need be stored in VRAM
- Tremendous interpolation power
  - Thanks to Forward Kinematics
  - Keyframes can be very far and spare and still produce good in-betweens
- Flexible: possible to dynamically combine different animations into new ones
  - By animations interpolation, or by layering
- Efficacious ways to capture, edit, compute, simulate anims
  - Also, IK possible and easy to adjust them

Important:  
local transformation  
are manipulated  
Always need to  
combine them into  
final transform

164

## Some of limitation of skinning (recap)

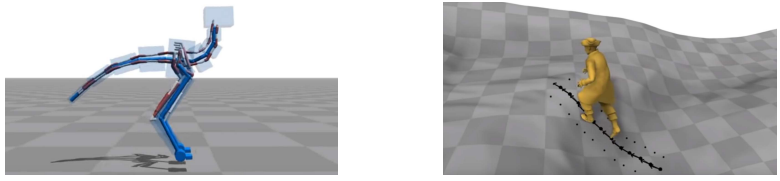
- **Deformations** induced on the rest mesh by the pose are simple and may be unrealistic
  - No: dynamic effects, collisions, volume preservation...
  - But can be helped by:  
good skinning (skinner artist must be good!)  
various technique, such as additional bones?
  - DQS is arguably a bit better than LBS, but still
- Interpolated / layered / IK-ed / rag-dolled **animations** can be unrealistic / simplistic / unaware of the broader context

In summary, (blend) skinning is an extremely cost-effective technique, providing a wealth of complex behaviors / effects for a small cost (in terms of computation, memory, construction times...), but the quality of animations however is intrinsically bounded and may become a bottleneck in 3D games visual quality.

165

## Research topic: apply ML to generate skeletal animations

- A very active area of research...

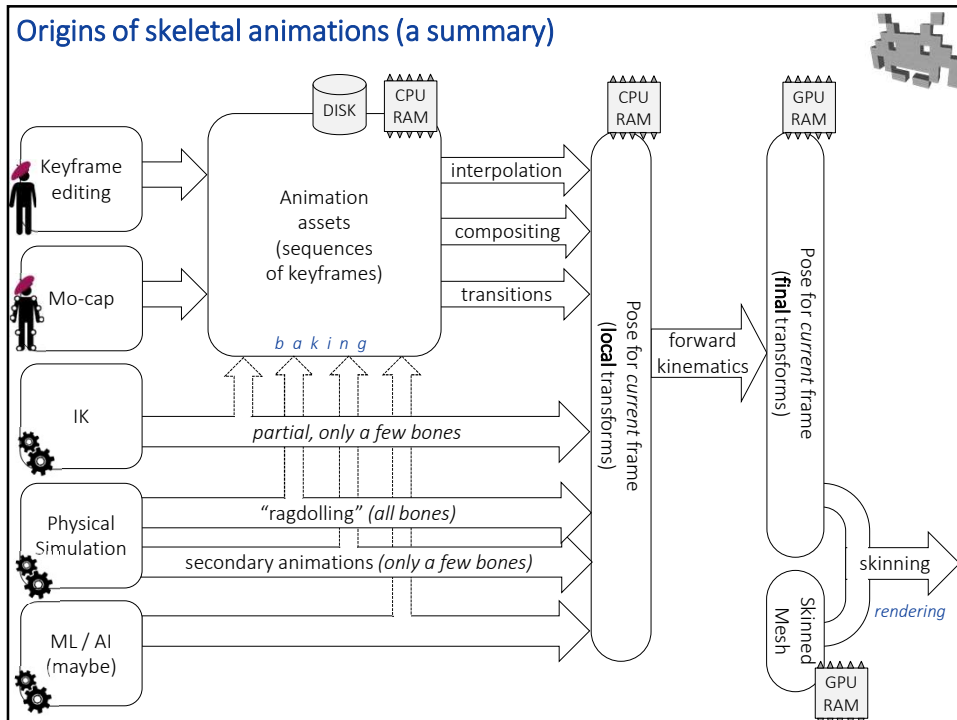


*Flexible Muscle-Based Locomotion for Bipedal Creatures*  
Thomas Geijtenbeek, Michiel van de Panne, A. Frank van der Stappen  
SIGGRAPH 2013

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

(among MANY others) Next lecture!

166



167