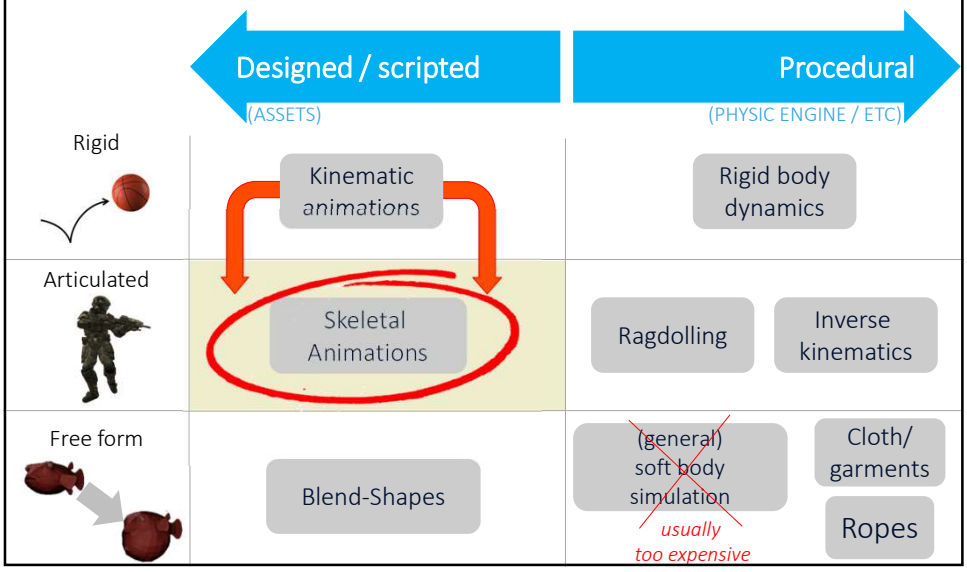
Course Plan

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. 8: Game 3D Animations
lec. 9: Game 3D Audio
lec. 10: Networking for 3D Games
lec. 11: Artificial Intelligence for 3D Games
lec. 12: Game 3D Rendering Techniques

63

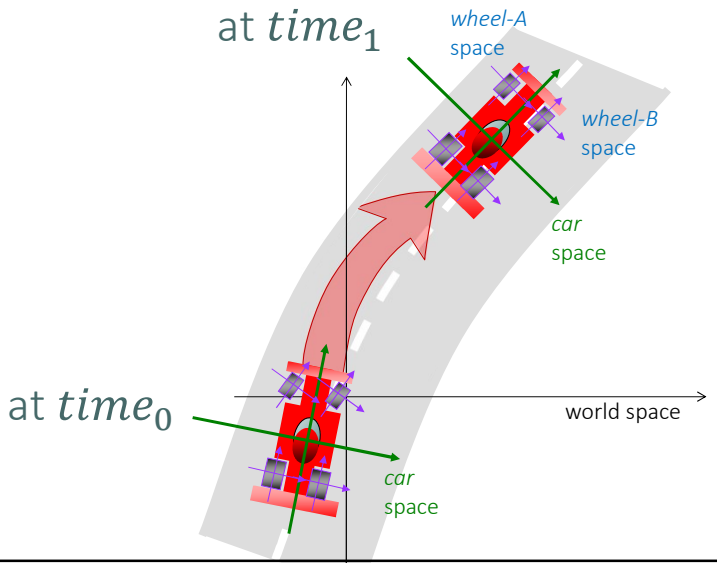
Animations in games
(of 3D Solid Objects)



64

Step by step...

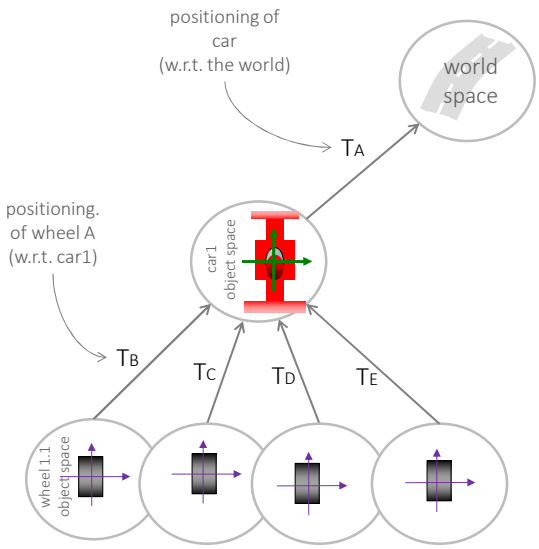
Kinematic animations



65

Step by Step...

Kinematic animatios

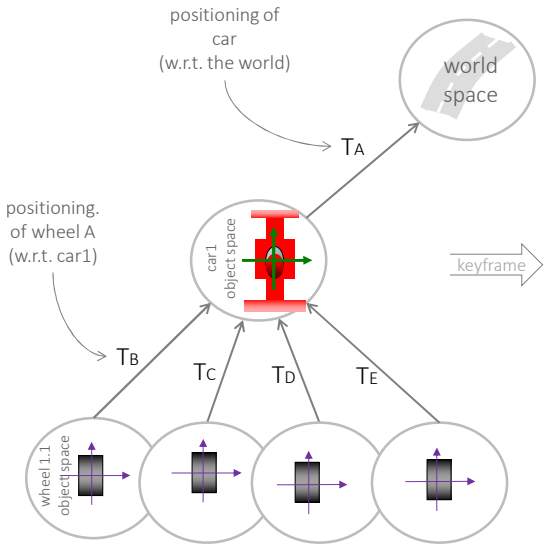


Car position & "pose" :

transformations				
TA	TB	TC	TD	TE

66

Step by Step...
Kinematic animatios



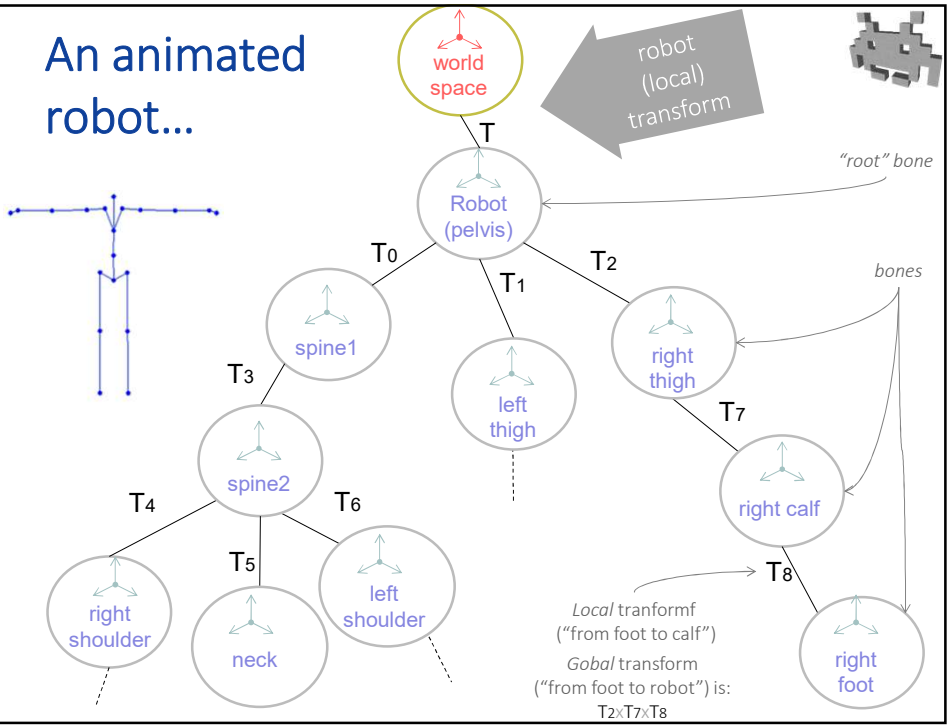
Car animation:

time	*transformations*				
t_0	$T_{0,A}$	$T_{0,B}$	$T_{0,C}$	$T_{0,D}$	$T_{0,E}$
t_1	$T_{1,A}$	$T_{1,B}$	$T_{1,C}$	$T_{1,D}$	$T_{1,E}$
t_2	$T_{2,A}$	$T_{2,B}$	$T_{2,C}$	$T_{2,D}$	$T_{2,E}$
t_3	$T_{3,A}$	$T_{3,B}$	$T_{3,C}$	$T_{3,D}$	$T_{3,E}$

keyframe

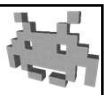
67

An animated robot...



68

Step by step…

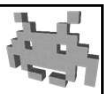


From a bunch of pieces…

- So far: one mesh in each "bone"
  - (e.g., car-cockpit, car-wheel)
- Ok, for simple structure
  - (like a car, a windmill…)
- What about a humanoid "robot" with 25-60 "bones"?
  - Individual meshes for arms, forearms, legs… three meshes for each finger?
  - Possible, but…
    - inefficient to render (lots of "draw calls")
    - uneasy to manage (lots of files?)
    - a nightmare to design / author ("sculpt me a nice looking calf")
    - and… looks right only for robots (each object rigid!)

69

… to articulated models…



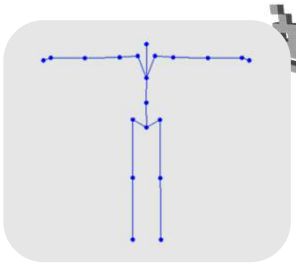
- Idea: one mesh, but *skinned*
  - 1 mesh per the entire character
  - a new attribute per vertex: *index of bone*
  - the 3D model can now be animated!

"Skinning" of the mesh (1st version).

- Orthogonality models / animations!
  - that is:
    - one skinned mesh: runs with any animations
    - one skeletal animation: can be appliecable to any model
  - (as long as they use the same RIG – set of bones)
  - →500 models + 500 animations = 1000 things in GPU RAM
    - not: 500x500 combinations
- The tasks required from digital artists:
  - "rigging": define the skeleton inside the mesh (riggers)
  - "skinning": define vertex-to-bone links, i.e. the skinning (skinners)
  - "animation": define the actual animations(animators)

70

Rig (or skeleton): data structure 1/2



- A tree of bones
- bone:
  - Vectorial frame (space) used to express pieces of the animated model
  - eg, for a humanoid: *forearm, calf, pelvis, …*
  - (rigging bone != biological bones)
- Space of the *root* bone =(def)= object space (of the entire character)
- How many bones in a skeleton of a humanoid:
  at least: 22-24 (typically)
  reasonable: ~40 bones.
  very high: few 100s

71

Pose: data structure

One transformation for each bone *i*

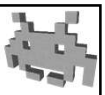
- Local transform: (of bone *i*)
  - from: space of one *i*
  - to: space of bone father of *i*

often, only the rotation component

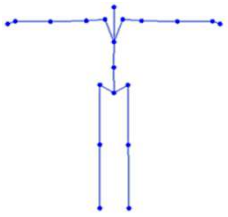
("fixed length bones": translations defined once and for all by the skeleton)

72

Rig (or skeleton): data structure 2/2

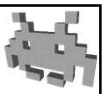


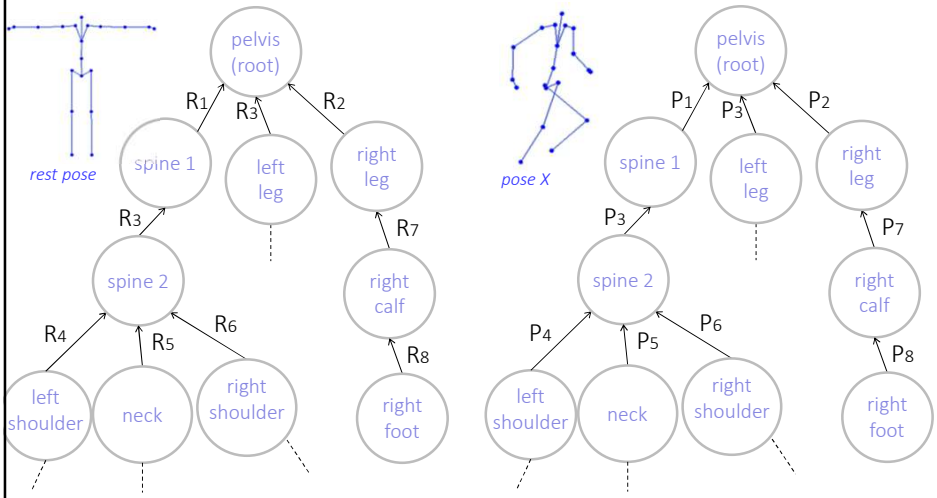
1. Hierarchy (tree) of bones
   - a root bone on top
2. A special pose «rest pose»
   - 3D models are to be modelled in this pose
   - also: «T-pose», «T-stance»,
     - same reason why T-shirts are called T-shirts ;)
   - also: «A-pose», when arms are bent down



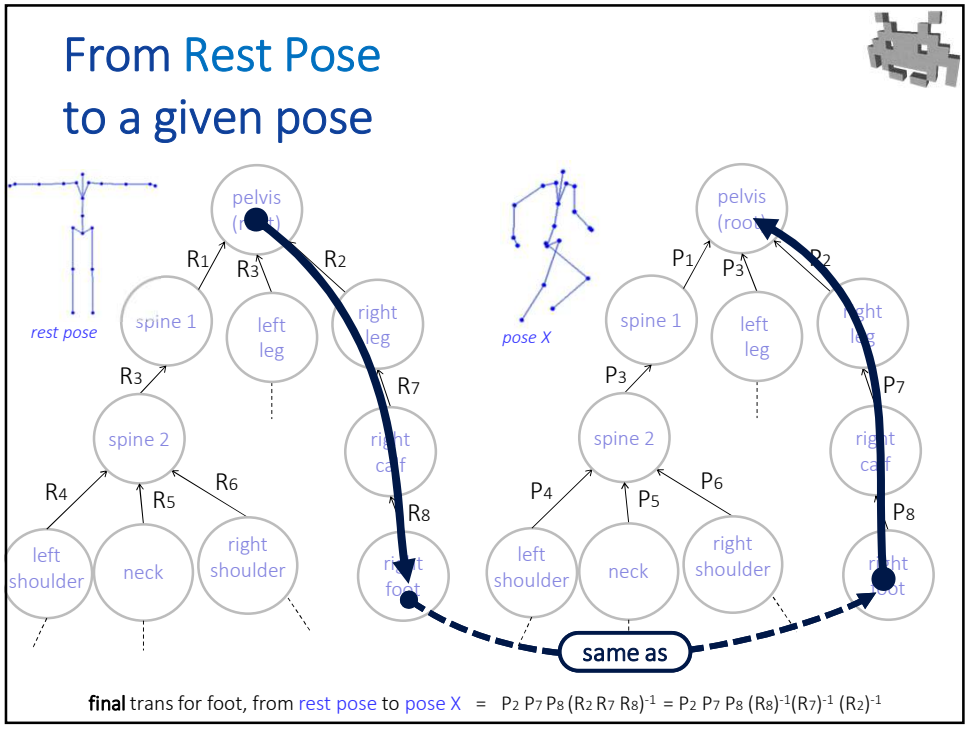
73

From Rest Pose to a given pose

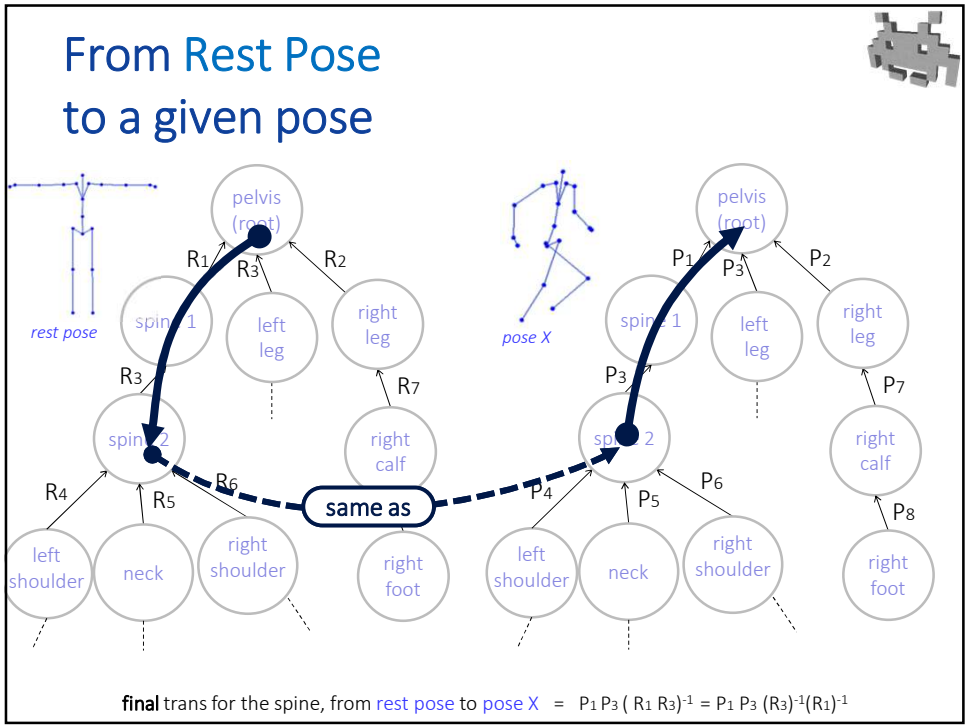




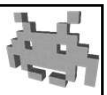
74


From Rest Pose
to a given pose
rest pose
pose X
pelvis (root)
spine 1
left leg
right leg
spine 2
right calf
left shoulder
neck
right shoulder
right foot
R1
R2
R3
R4
R5
R6
R7
R8
P1
P2
P3
P4
P5
P6
P7
P8
same as
final trans for foot, from rest pose to pose X = P2 P7 P8 (R2 R7 R8)-1 = P2 P7 P8 (R8)-1(R7)-1 (R2)-1

75


From Rest Pose
to a given pose
rest pose
pose X
pelvis (root)
left leg
right leg
right calf
left shoulder
neck
right shoulder
right foot
R1
R2
R3
R4
R5
R6
R7
P1
P2
P3
P4
P5
P6
P7
P8
same as
final trans for the spine, from rest pose to pose X = P1 P3 (R1 R3)-1 = P1 P3 (R3)-1(R1)-1

76



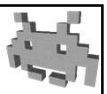
Bone transforms in a pose. E.g. for «right foot» bone:

- Local Transform: P_8
  - from *«right foot»* to *«right lower leg»*
- Global Transform: $P_2\ P_7\ P_8$
  - from *«right foot» space* to *«character» space*
  - uses the Hierarchy of the Skeleton
    - once computed, skeleton hierarchy no longer needed
- Final Transform: $P_2\ P_7\ P_8\ R_8^{-1}\ R_7^{-1}\ R_2^{-1}$
  - from *«character» space* in rest pose to *«character»* space in dest. pose
  - uses the Rest Pose of the Skeleton (R_1 … R_N)
    - once this is computed, Rest Pose is no longer needed

the local frame of the character, which is the frame of the root bone

mesh (vertices normals…) is defined in this space!

78



Pose (for a given rig) : data structure

- pose = array of (local) transforms
  - it's defined for one given rig
  - RAM cost: n_bones x bytes_for_a_tranform

Bone *i*	Trasform[*i*]
#0 (pelvis) [root]	L[0]
#1 (spine)	L[1]
#2 (chest)	L[2]
#3 (shoulder sx)	L[3]
…	…
#10 (calf)	L[10]
…	…

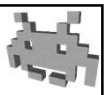
Local Transform

It includes:

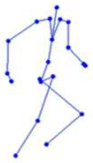
- a Rotation: always!
- a Translation: maybe
  If not, use the one defined in the rest pose of the rig.
  ==> a pose cannot redefine bone *lengths*.
- a Scaling: usually not
  A joint cannot enlarge a part of the character

79

Pose (for a given rig) : data structure in GPU



- pose = array of final transforms
  - it's defined for one given rig
  - RAM cost: n_bones × bytes_for_a_tranform



Bone *i*	Trasform[*i*]
#0 (pelvis) [root]	F[0]
#1 (spine)	F[1]
#2 (chest)	F[2]
#3 (shoulder sx)	F[3]
…	…
#10 (calf)	F[10]
…	…

final transforms

computed in preprocessing e.g. as:

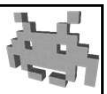
$L[2]\ L[7]\ (R[7])^{-1}\ (R[2])^{-1}$

local transforms of *this* pose

local transforms of *rest* pose

80

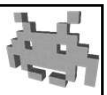
Skeletal Animation : data structure (CPU or GPU)



- 1D Array of poses (1 pose = 1 keyframe)
  - RAM cost:
    (num keyframes) × (num bones) × (transform size)
  - Each pose assigned to time d*t*
    - delta from start of animation t_0
  - Sometime, looped
    - interpolation 1st keyframe with last

81

Step by step…



From a bunch of pieces…

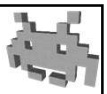
- one separate mesh in each “bone”
  - “calf” mesh, “head” mesh, “right-forearm” mesh…

… to *a* single articulated model…

- 1 “skinned” mesh for the entire character
- in each vertex, an index of a bone
  - a vertex-bone link

82

… to articultated *defomable* models.

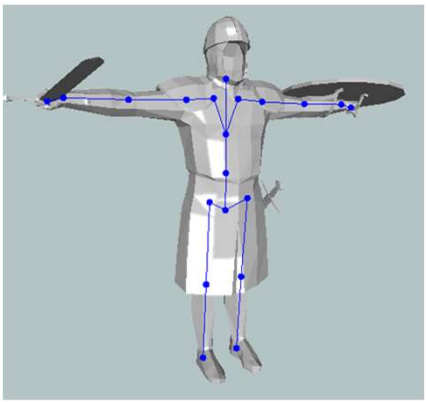


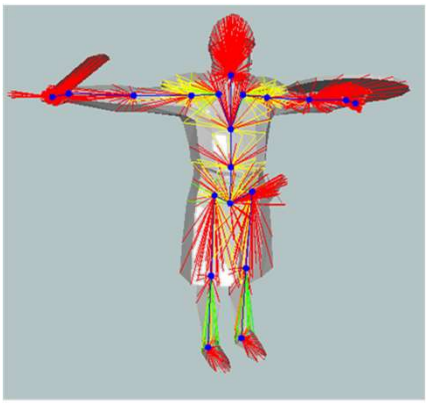
- Idea: link each vertex to multiple bones
  - «blended» skinning
  - each linked bone with a strength (a weight)
- Transform of the vertex:
  - interpolation of the final transformations associated to the linked bones
  - weights of the interpolation: defined per-verex
- Data structures: per-vertex attributes
  - store:
    - [bone index , weight] × N_{max}
    - (typically, N_{max} = 4 or 2, see later)

the “Skinning” of the mesh
blended version (the one which is actually used in games)

83

Rigs (skeletons) and Skinned Meshes



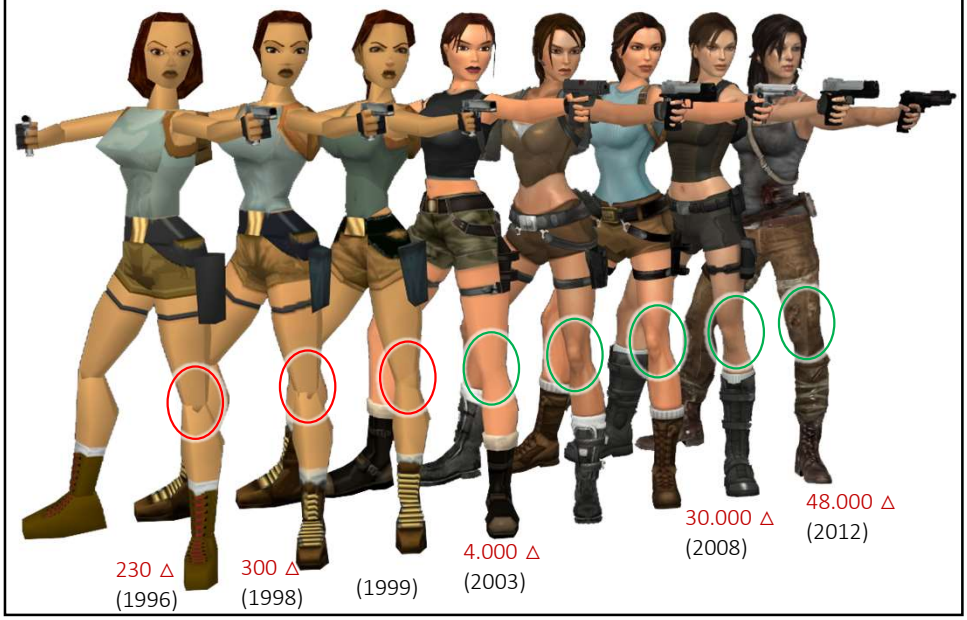


Rig (or skeleton)
the hierarchical structures of bones
the rest pose transformations (per bone)

Skinned mesh
a mesh with link-to-bones stored
as a (per-vertex) attributes

84

(this story actually happened)



85

(this story actually happened)

87

Skinned Mesh: data structure

- A Mesh with a skinning
  - A per vertex attribute
  - Stored per vertex:
    - [bone index , weight] x N_{max}
    - example:

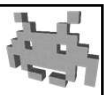
bone links

Vertex 134 →

Bone Index	Weight
9 *(Spine B)*	0.4
13 *(Chest)*	0.1
15 *(Shoulder Right)*	0.4
16 *(Forearm Right)*	0.1

89

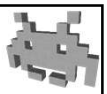
N_max = How many bone links for each vertex



- 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)

90

(but why put a hard-wired buond on bone links?)



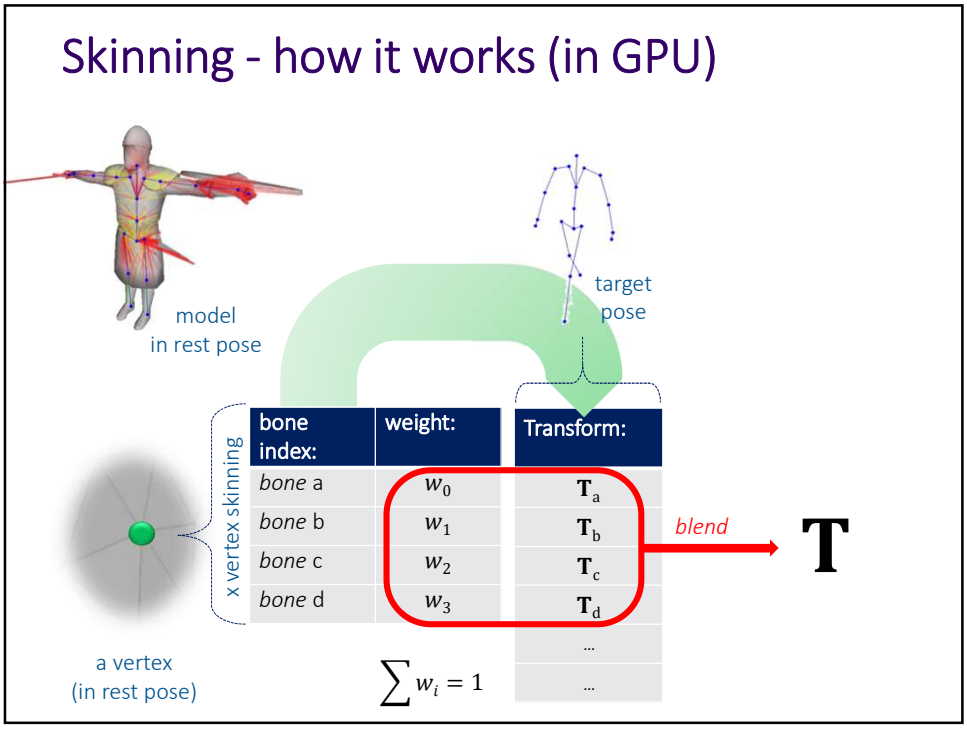
- Reduces performance cost
  - N_{max} tranforms need be interpolated in GPU
    - in vertex shader
  - GPU = no good at control:
    - always uses exactly N_{max} trasf
    - unused bones: weight = 0
- Reduces GPU RAM cost
  - reduces storage
  - fixed length arrays: good for GPU
    - N_{max} (index,weight) 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

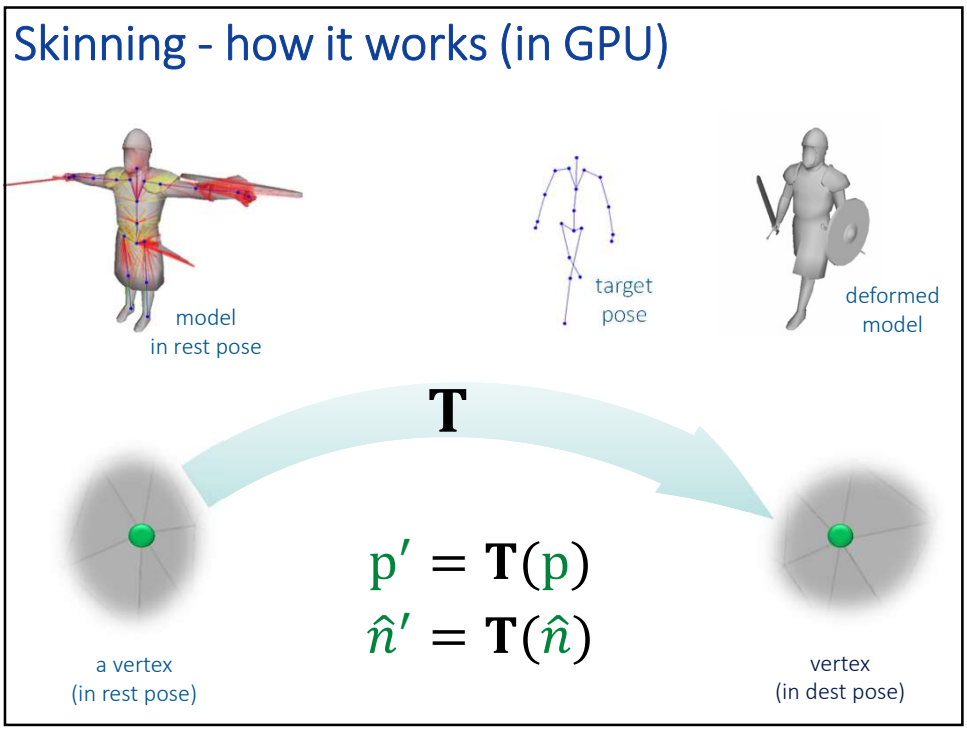
91

Skinning - how it works (in GPU)

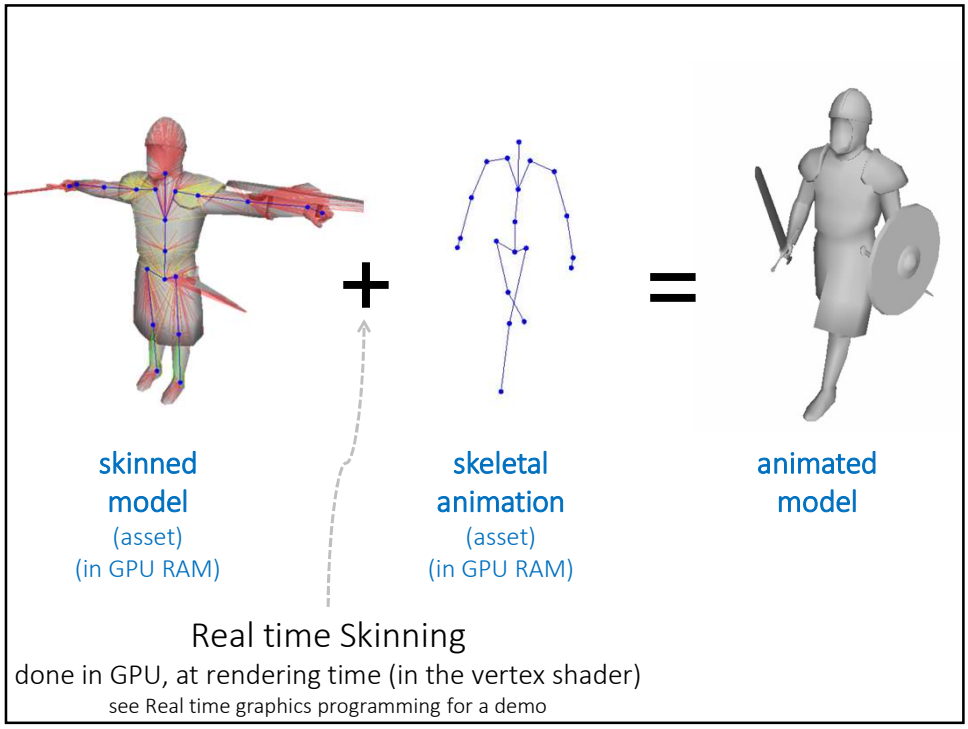


92

Skinning - how it works (in GPU)

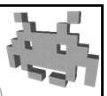


93



94

GPU real time Skinning – variants



(a choice of the rendering engine)

Bone:	Weight:	Transform:
bone a	$w0$	$T0$
bone b	$w1$	$T1$
bone c	$w2$	$T2$
bone d	$w3$	$T3$

blend → **T**

how are they stored? | how is this done?

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

nothing else works!

95

Linear Blend Skinning (LBS)

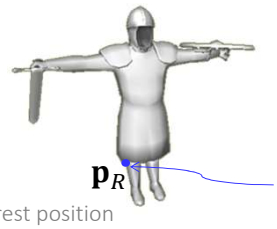
interpolation of the per-bone matrices

more in general, Nmax-1

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

interpolation of per-bone transformed points

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

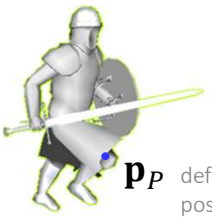


$\mathbf{p}_R$

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

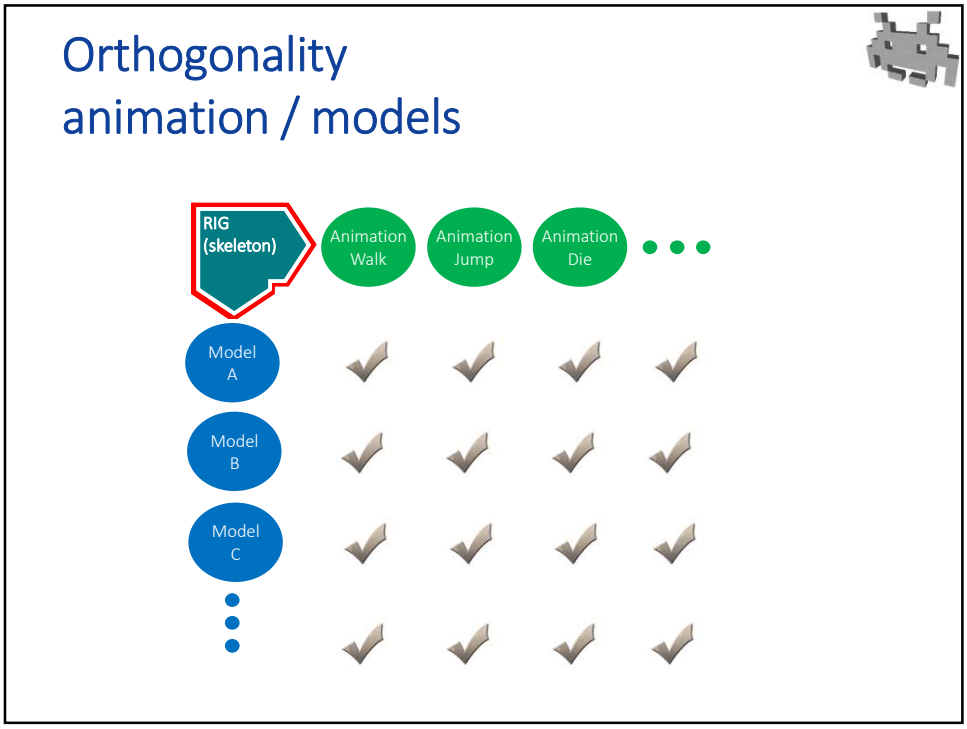
96

Dual Quaternion Skinning (DQS)

- Per-bone final transform stored as a dual quaternion ← See lecture on transform representation
  - better quality
  - better interpolation
  - > GPU cost
    - (necessary ops: around +50%)
- LBS or DQS?
  - a call of the game engine

97

Orthogonality animation / models



103

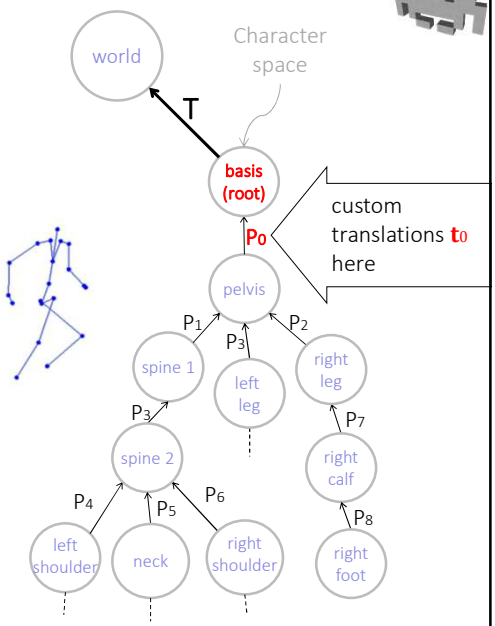
Intro: skinning

104

Intro: skinning

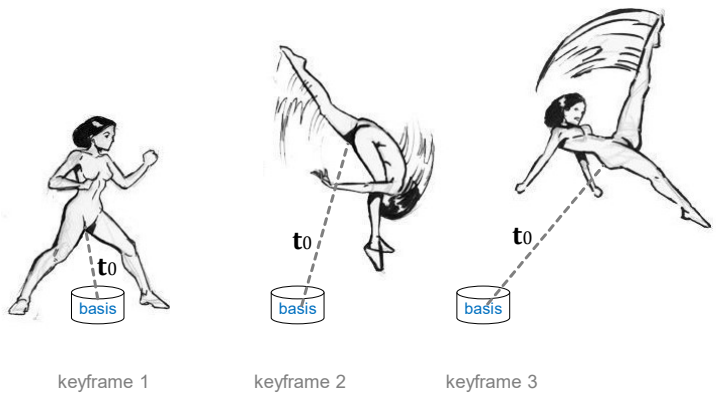
105

The root is usually an abstract "basis" bone



106

Basis bone



so that each animation asset can include
a global displacement $\mathbf{t}_0$ in each keyframe

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

107

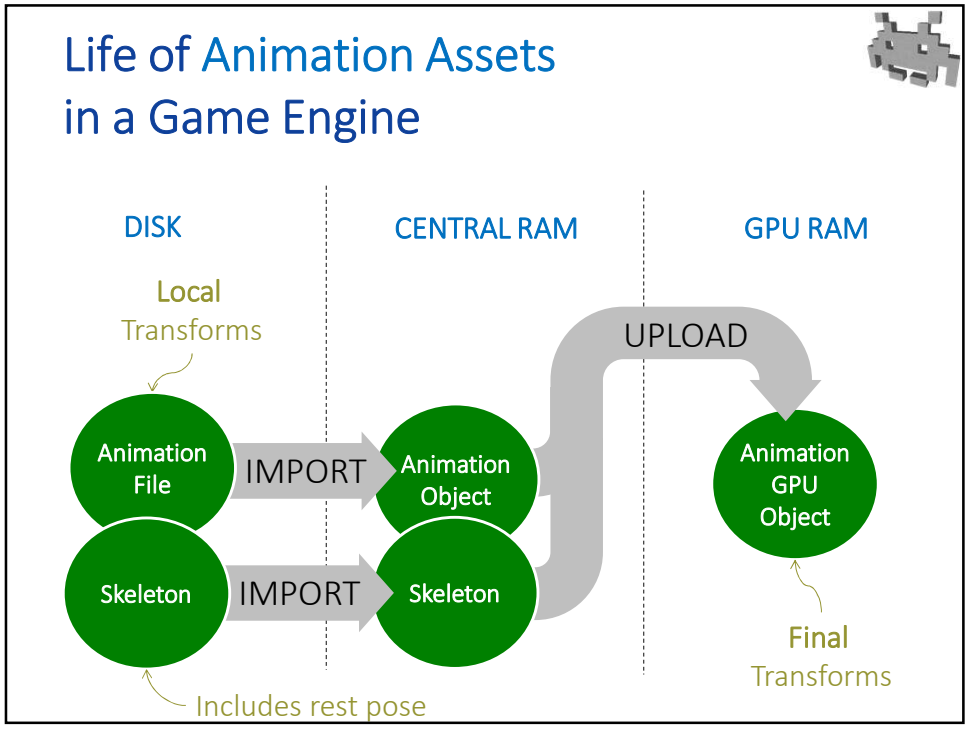
(recap) Skeletal animations: 3 Assets (data strcutures)

- Rig (or "skeleton")
  - Tree of bones
  - ∀ bone => reference frame (in rest pose)
    - reference frame root bone = object space
- Skinned 3D Models
  - Mesh with links: vertices => bones
  - ∀ vertex: attributes: [bone index , weights] x N_{max}
- Skeletal animations
  - Sequence of keyframe poses
  - ∀ pose, ∀ bone = a local transform

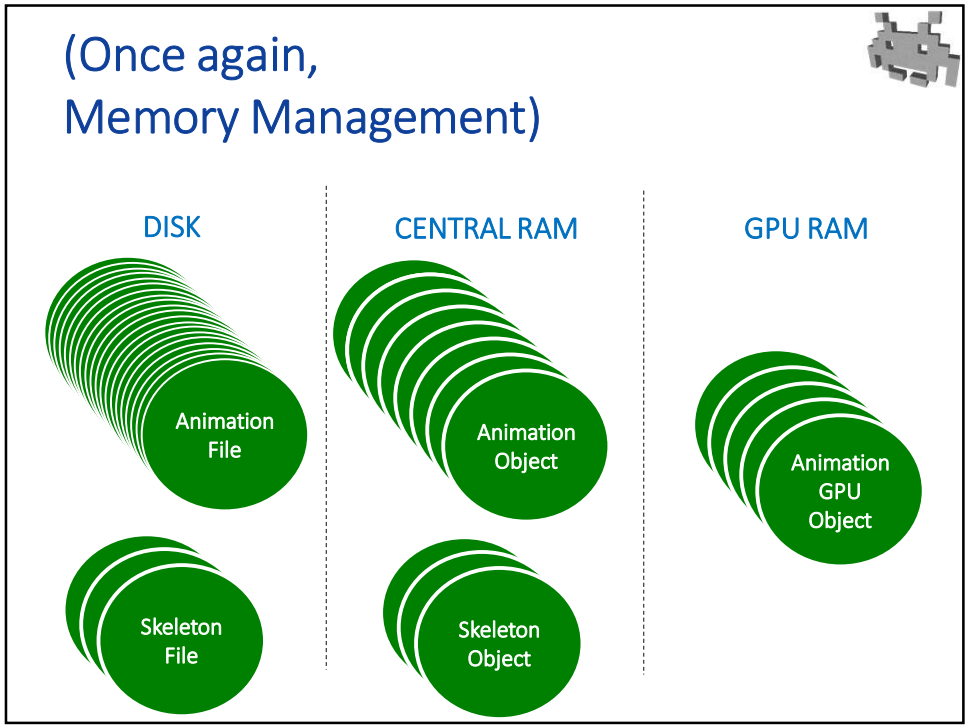
examples of interchange formats (for all three):

- .SMD (Valve), .FBX (Autodesk), .BVH ("behaviour" Biovision)

108

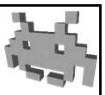

Life of Animation Assets in a Game Engine
DISK
CENTRAL RAM
GPU RAM
Local Transforms
UPLOAD
Animation File
IMPORT
Animation Object
Animation GPU Object
Skeleton
IMPORT
Skeleton
Final Transforms
Includes rest pose

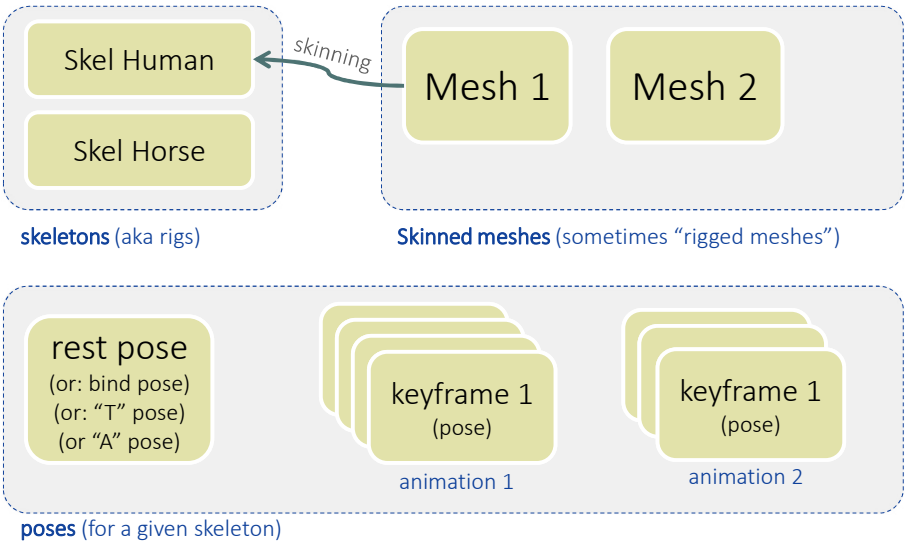
109


(Once again, Memory Management)
DISK
CENTRAL RAM
GPU RAM
Animation File
Animation Object
Animation GPU Object
Skeleton File
Skeleton Object

110

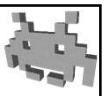
Skeletal animation Assets

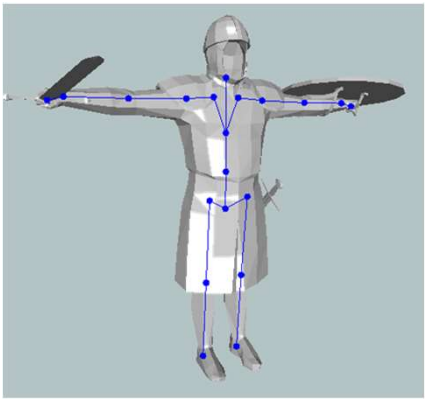




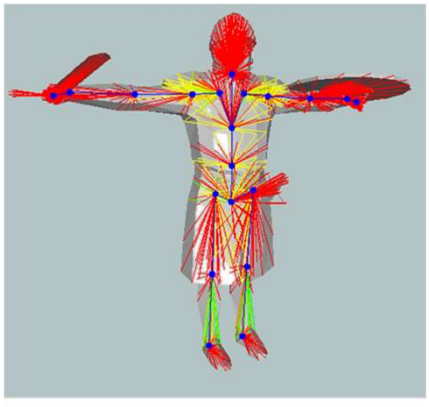
111

Task in an asset production pipeline: Rigging & Skinning (of a 3D mesh)





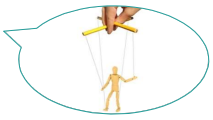
Rigging – authoring of a rig
defining the skeleton
(often: also of the controls
to define poses for it)



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

112

Content creation Tasks: 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* rig)
  - also: define controls for animator
- **Skinning** (of a mesh):
  - painting link vertex-bones

rigger

skinner

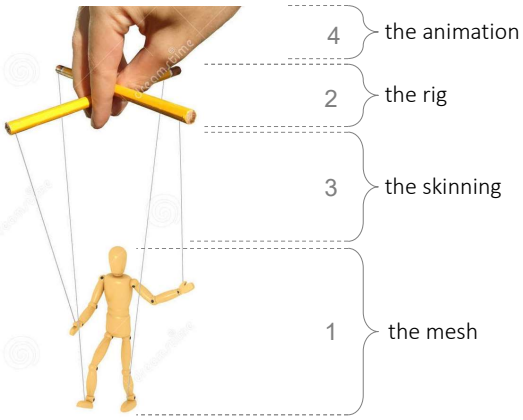
by **Ditigal animator**

- **Animation** (of a rig)
  - authoring of (skeletal) animations
  - (More about this later)

animator

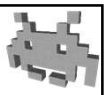
113

Asset production



114

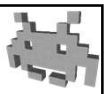
Skeletal animations: authoring / obtaining them



- Manual editing
  - digital animators
  - help from:
    IK (in animation interfaces),
    physical simulations (for “secondary” animations)
- From physics simulation
  - just use the right set of constraints!
    (easy, in Verlet)
  - in preprocessing (bake them) or
    on the fly: “Ragdolling”
- Or…

115

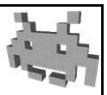
Skeletal animations: authoring / obtaining them



- Motion capture
  (“mocap”)

116

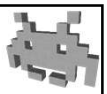
Motion capture



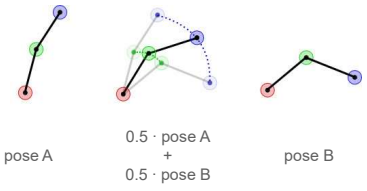
- Requires heavy setup (maybe not in the future?)
  - Markers / suits
  - Controlled cameras
  - Studio
  - Action must take space in a working space
- Requires skilled actors / performers / athletes
- Can be used to capture
  - single animations (a football stunt, walking)
  - joint performances (cutscenes)
- Requires much postprocessing
  - (cleanup, extraction of keyframes)

117

Interpolation poses



- any two poses can be interplated!



  - just interpolate the per bone *local* transform
  - attention: this requires re-computation of *final* transforms after interpolation

118