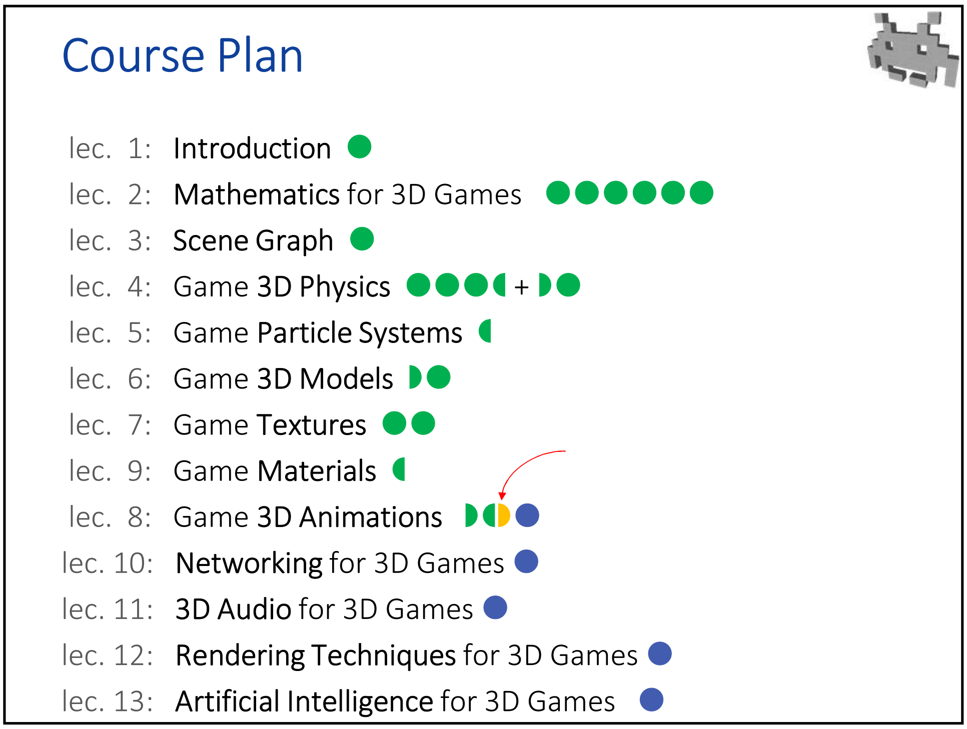
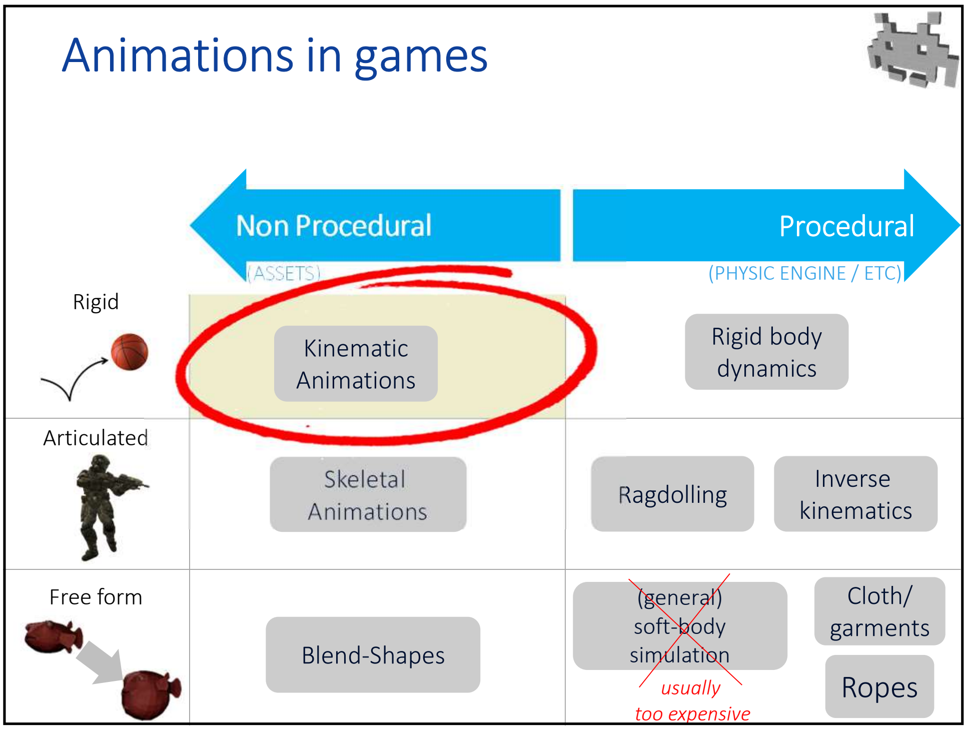


Course Plan

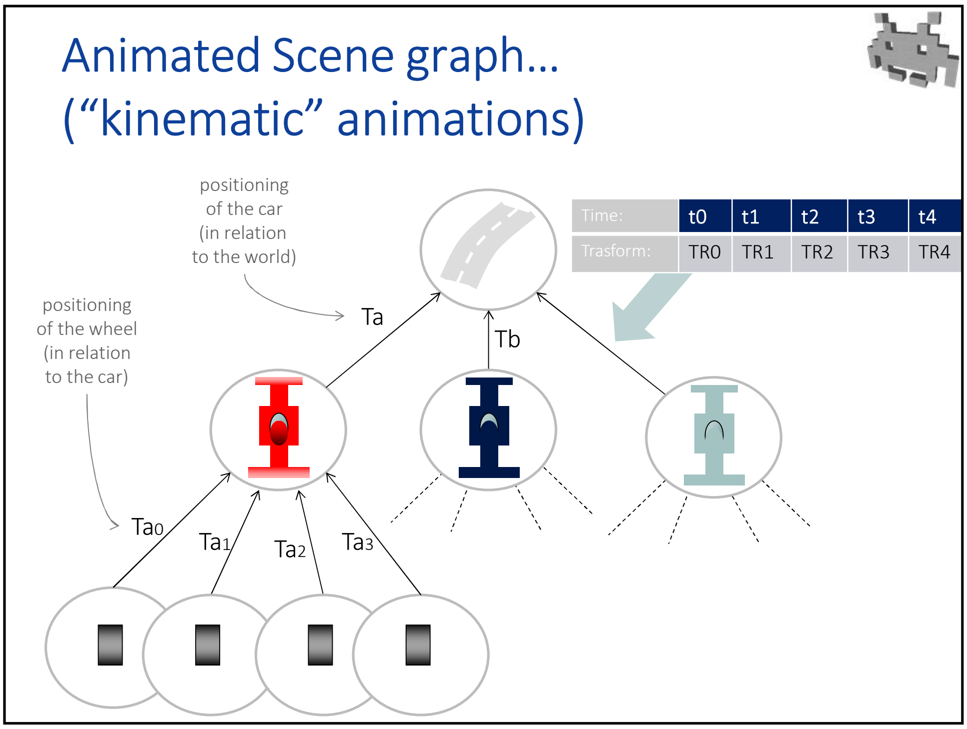
lec. 1: Introduction ●
lec. 2: Mathematics for 3D Games ●●●●●●
lec. 3: Scene Graph ●
lec. 4: Game 3D Physics ●●●◖ + ◗●
lec. 5: Game Particle Systems ◖
lec. 6: Game 3D Models ◗●
lec. 7: Game Textures ●●
lec. 9: Game Materials ◖
lec. 8: Game 3D Animations ◗◖◗●
lec. 10: Networking for 3D Games ●
lec. 11: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

58



Animations in games

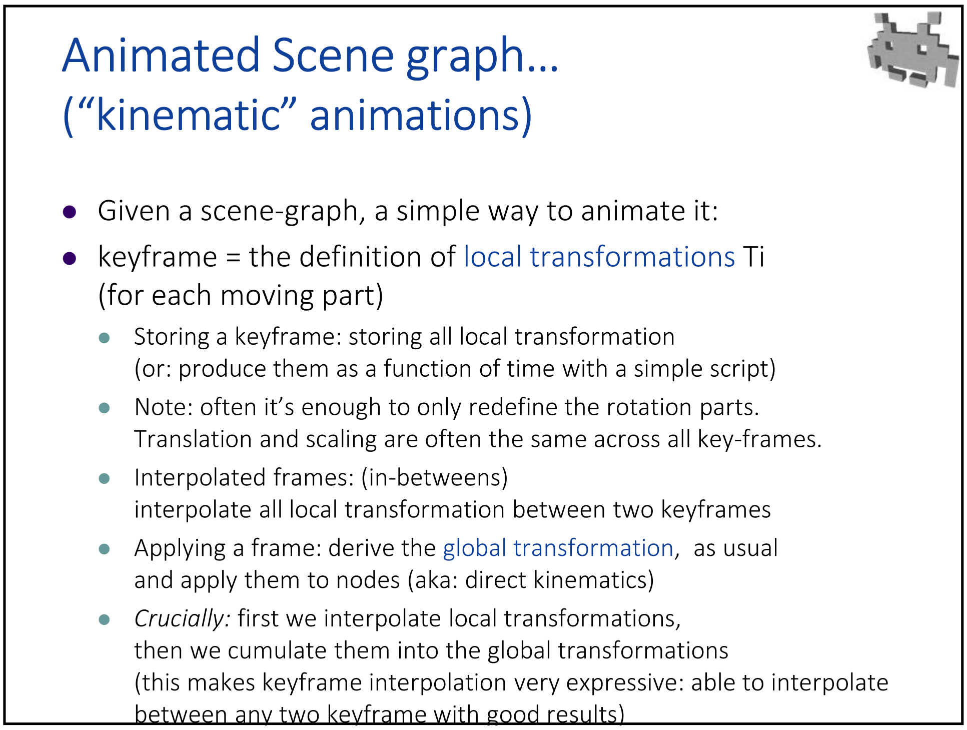
59



Animated Scene graph… (“kinematic” animations)

Time:	t0	t1	t2	t3	t4
Trasform:	TR0	TR1	TR2	TR3	TR4

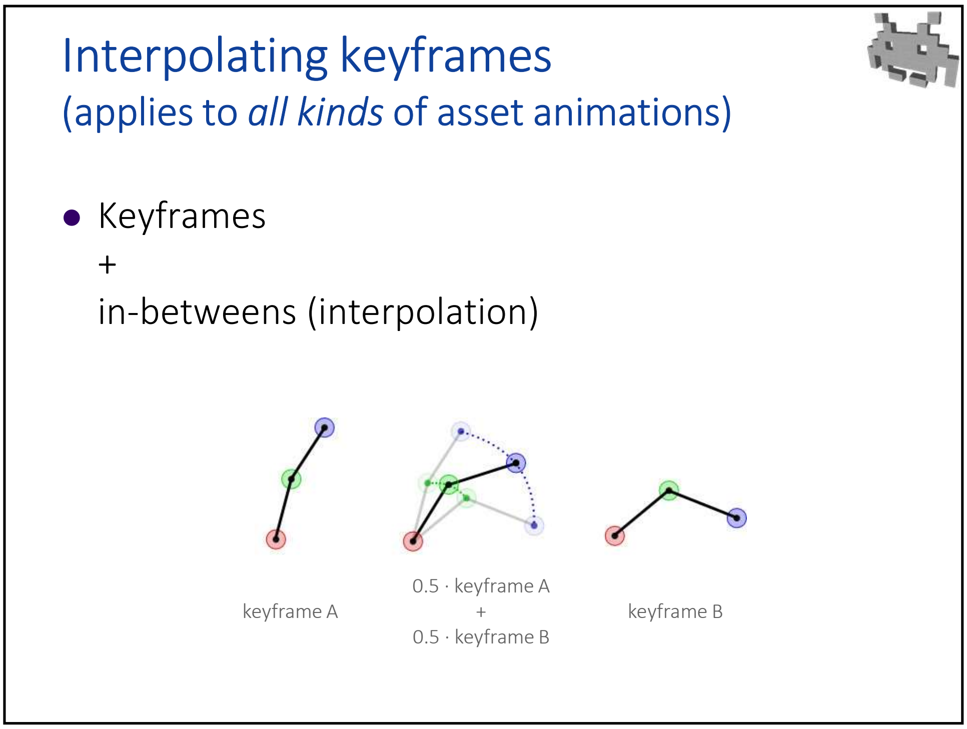
61



Animated Scene graph… (“kinematic” animations)

- Given a scene-graph, a simple way to animate it:
- keyframe = the definition of local transformations Ti (for each moving part)
  - Storing a keyframe: storing all local transformation (or: produce them as a function of time with a simple script)
  - Note: often it’s enough to only redefine the rotation parts. Translation and scaling are often the same across all key-frames.
  - Interpolated frames: (in-betweens) interpolate all local transformation between two keyframes
  - Applying a frame: derive the global transformation, as usual and apply them to nodes (aka: direct kinematics)
  - *Crucially:* first we interpolate local transformations, then we cumulate them into the global transformations (this makes keyframe interpolation very expressive: able to interpolate between any two keyframe with good results)

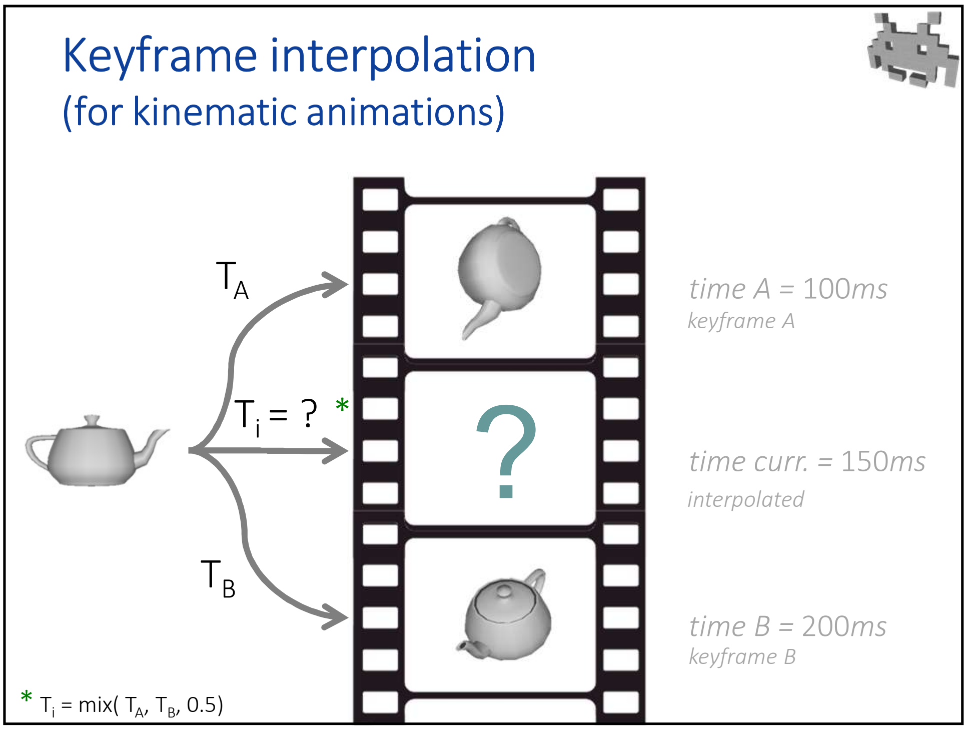
62



Interpolating keyframes
(applies to *all kinds* of asset animations)

- Keyframes
  +
  in-betweens (interpolation)

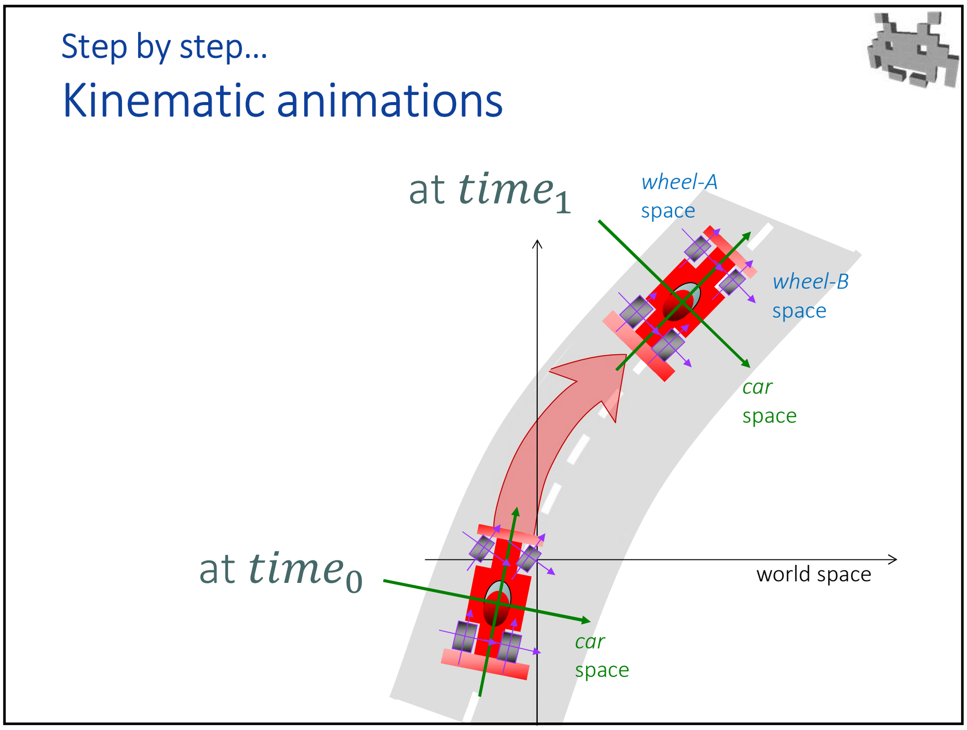
65



Keyframe interpolation
(for kinematic animations)

* $T_i = \text{mix}(T_A, T_B, 0.5)$

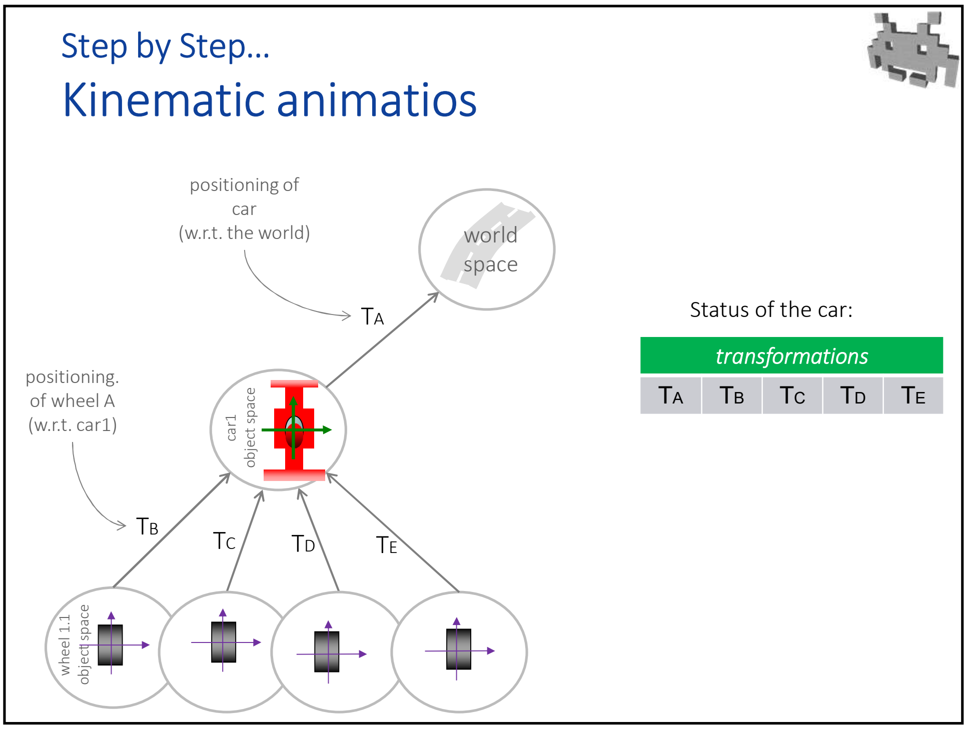
66



Step by step...

Kinematic animations

67



Step by Step...

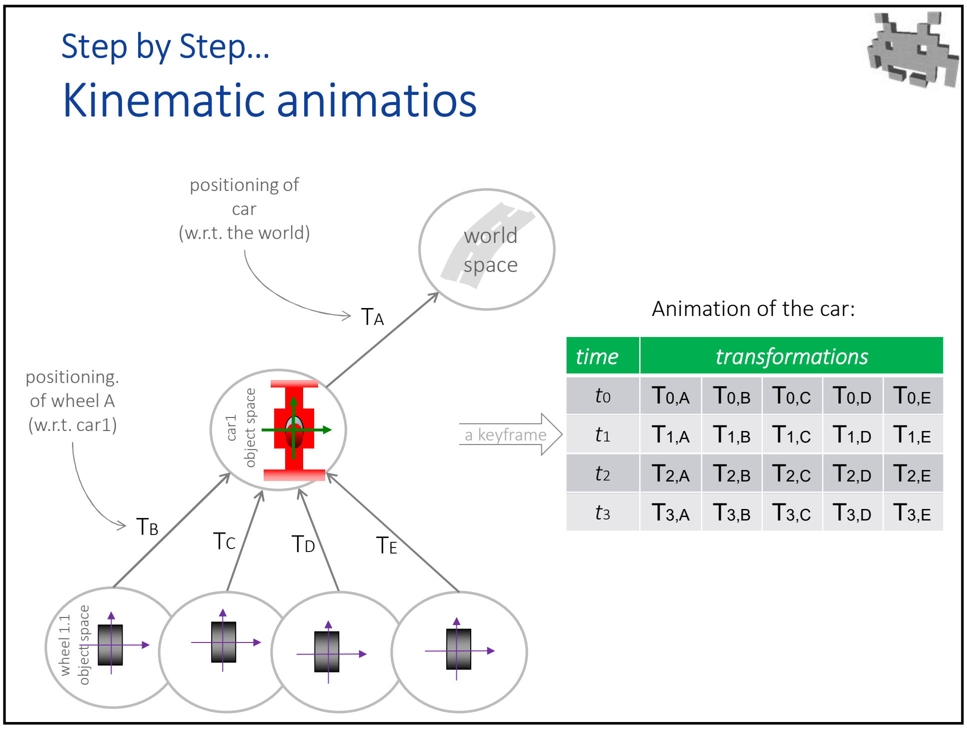
Kinematic animatios

positioning of car (w.r.t. the world)
TA
world space
positioning. of wheel A (w.r.t. car1)
car1 object space
TB
TC
TD
TE
wheel 1.1 object space

Status of the car:

transformations				
TA	TB	TC	TD	TE

68



Step by Step...

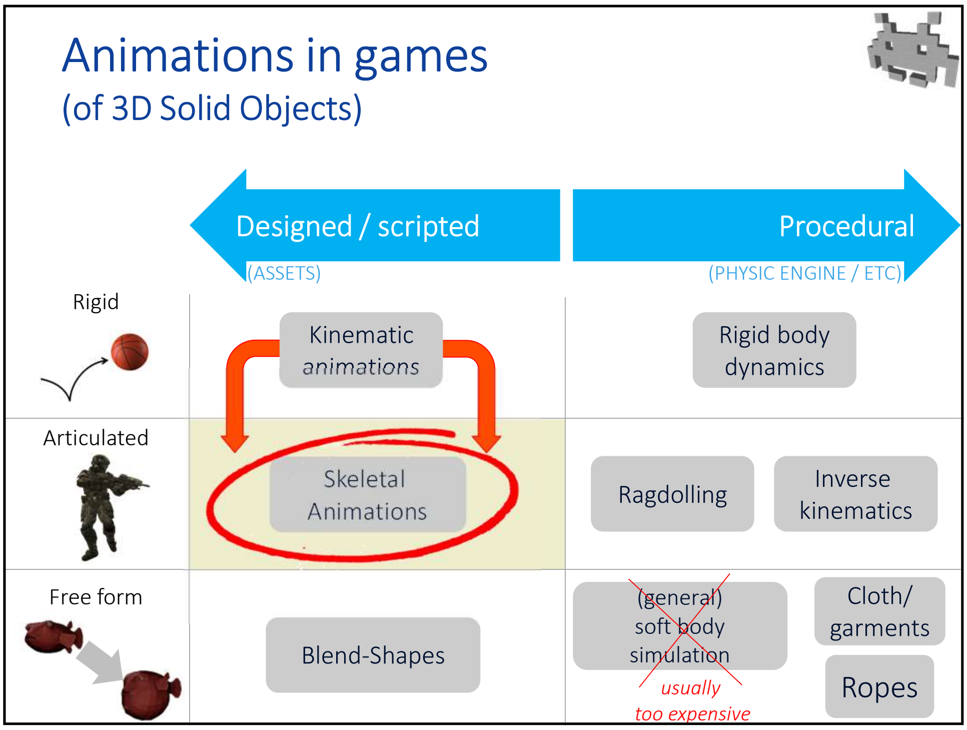
Kinematic animatios

positioning of car (w.r.t. the world)
world space
TA
positioning. of wheel A (w.r.t. car1)
car1 object space
TB
TC
TD
TE
wheel 1.1 object space
a keyframe

Animation of the car:

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	$T_{3,A}$	$T_{3,B}$	$T_{3,C}$	$T_{3,D}$	$T_{3,E}$

69



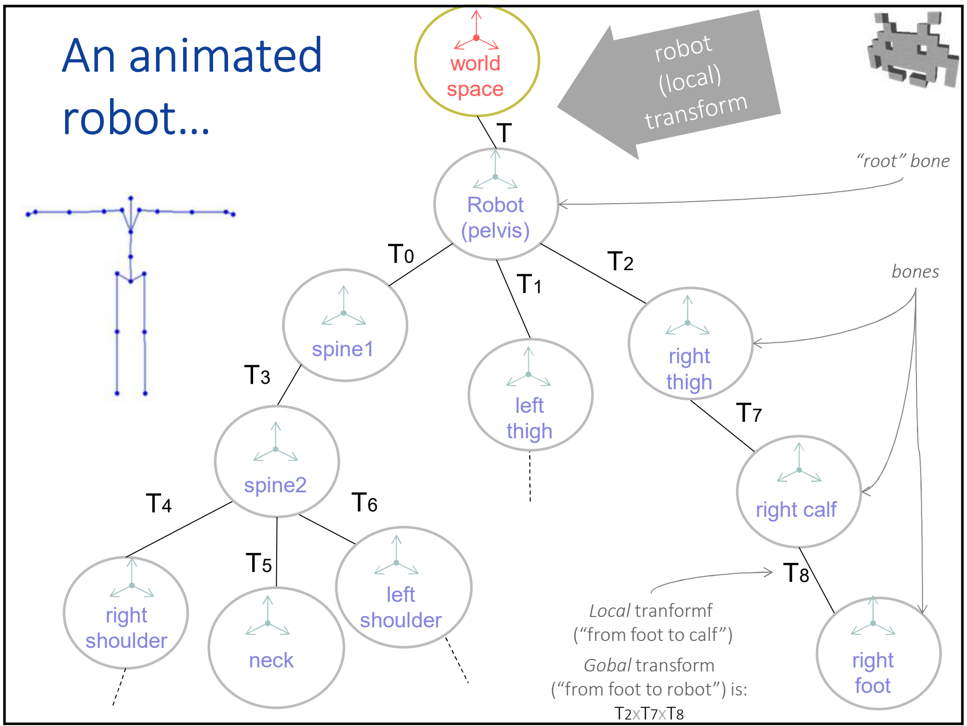
Animations in games

(of 3D Solid Objects)

	Designed / scripted (ASSETS)	Procedural (PHYSIC ENGINE / ETC)
Rigid	Kinematic *animations*	Rigid body dynamics
Articulated	Skeletal Animations	Ragdolling Inverse kinematics
Free form	Blend-Shapes	~~(general) soft body simulation~~ *usually too expensive* Cloth/ garments Ropes

70

An animated robot…



world space

robot (local) transform

T

"root" bone

Robot (pelvis)

T_0

T_1

T_2

bones

spine1

left thigh

right thigh

T_3

T_7

spine2

right calf

T_4

T_6

T_5

right shoulder

neck

left shoulder

T_8

Local tranformf ("from foot to calf")

Gobal transform ("from foot to robot") is: $T_2 \times T_7 \times T_8$

right foot

71

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!)

72

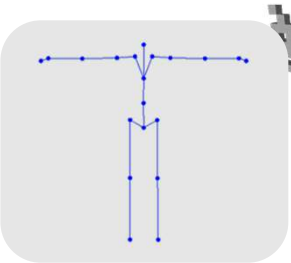
… to articulated models…

- 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!
- 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 skeleton)
  - →500 models + 500 animations = 1000 things in GPU RAM
    - not: 500x500 combinations
- The tasks required from digital artists:
  - "rigging": define the skeleton (the rig) inside the mesh (by riggers)
  - "skinning": define vertex-to-bone links, i.e. the skinning (by skinners)
  - "animation": define the actual animations (by animators)

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

73

Skeleton (or rig): data structure 1/2



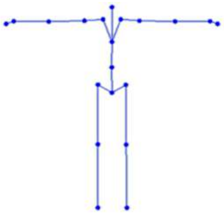
- A tree of bones
- bone:
  - Vectorial frame (space) used to express pieces of the animated model
  - eg, for a humanoid: *forearm, calf, pelvis,* …
  - (animation bones != biological bones)
- Space of the *root* bone =(def)= object space (of the entire character)
- How many bones in a skeleton of a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

74

Skeleton (or rig): 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



75

Pose: data structure

One transformation for each bone *i*

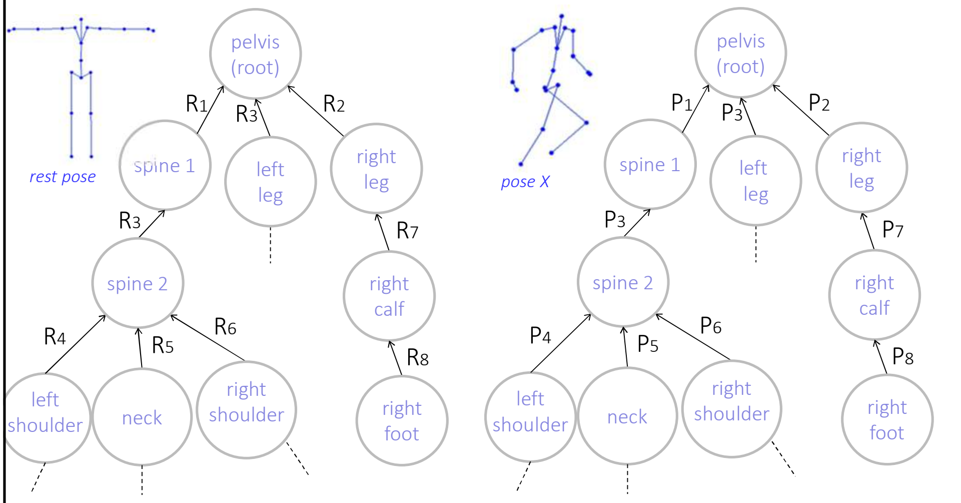
- Local transform: (of bone *i*)
  - from: space of one *i*
  - to: space of bone father of *i*

often, only the rotation component

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

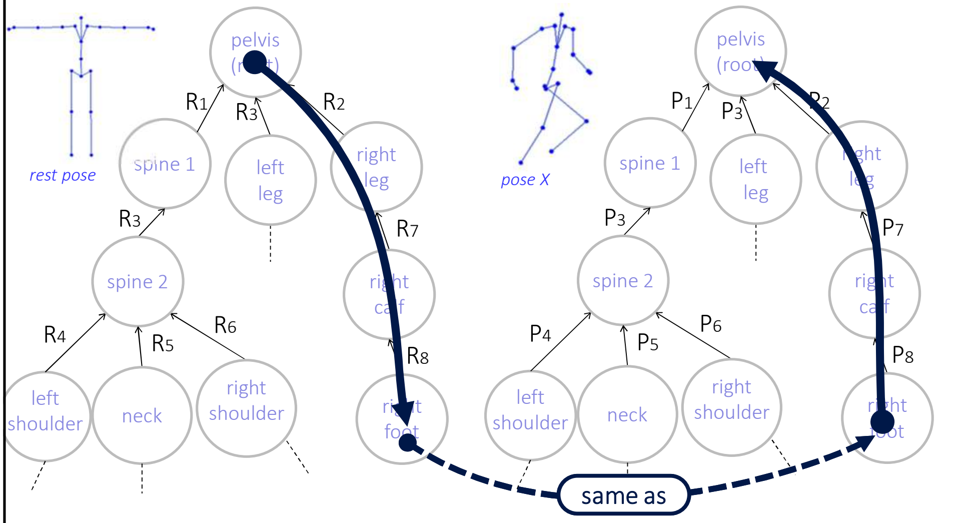
76

From Rest Pose to a given pose



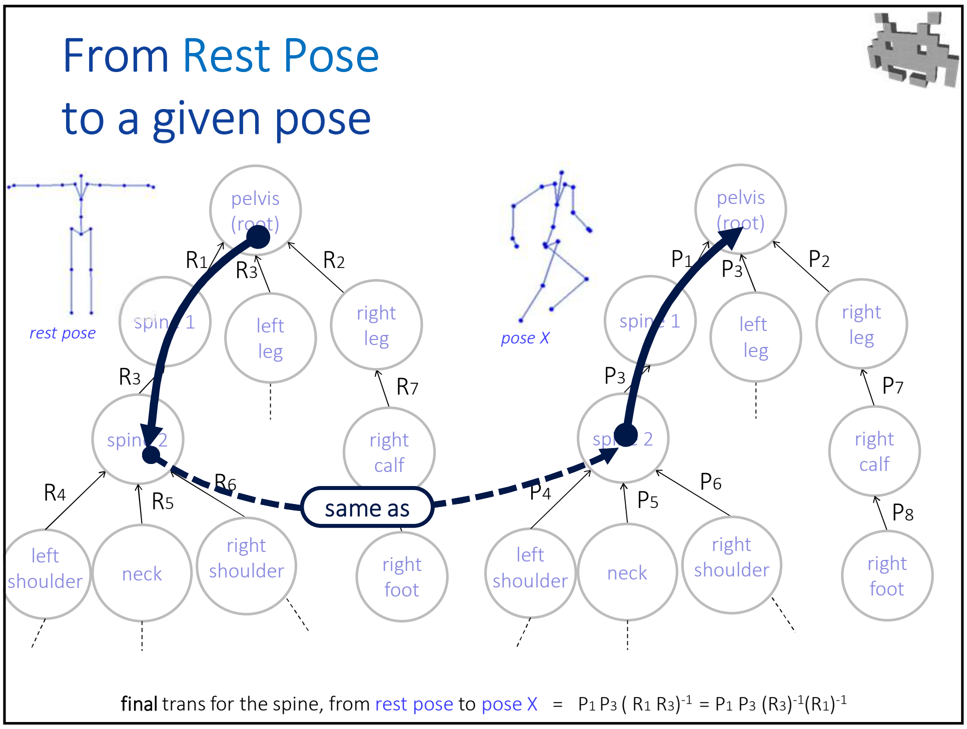
77

From Rest Pose to a given pose



final trans for foot, from rest pose to pose X $= P_2\,P_7\,P_8\,(R_2\,R_7\,R_8)^{-1} = P_2\,P_7\,P_8\,(R_8)^{-1}(R_7)^{-1}\,(R_2)^{-1}$

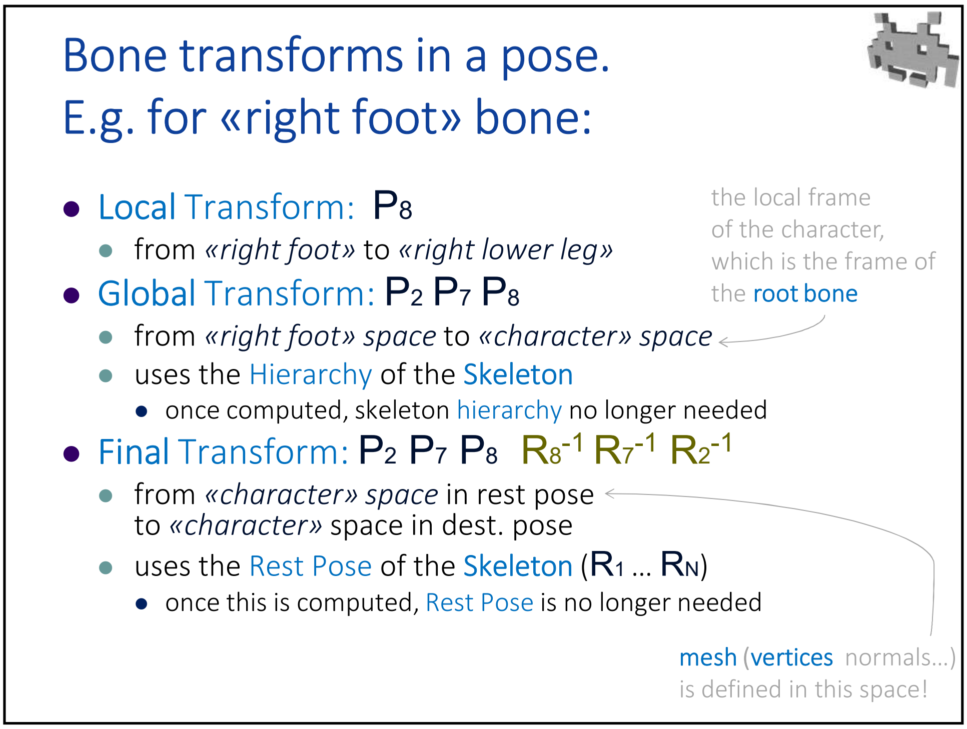
78



From Rest Pose to a given pose

final trans for the spine, from rest pose to pose X = $P_1 P_3 (R_1 R_3)^{-1} = P_1 P_3 (R_3)^{-1}(R_1)^{-1}$

79



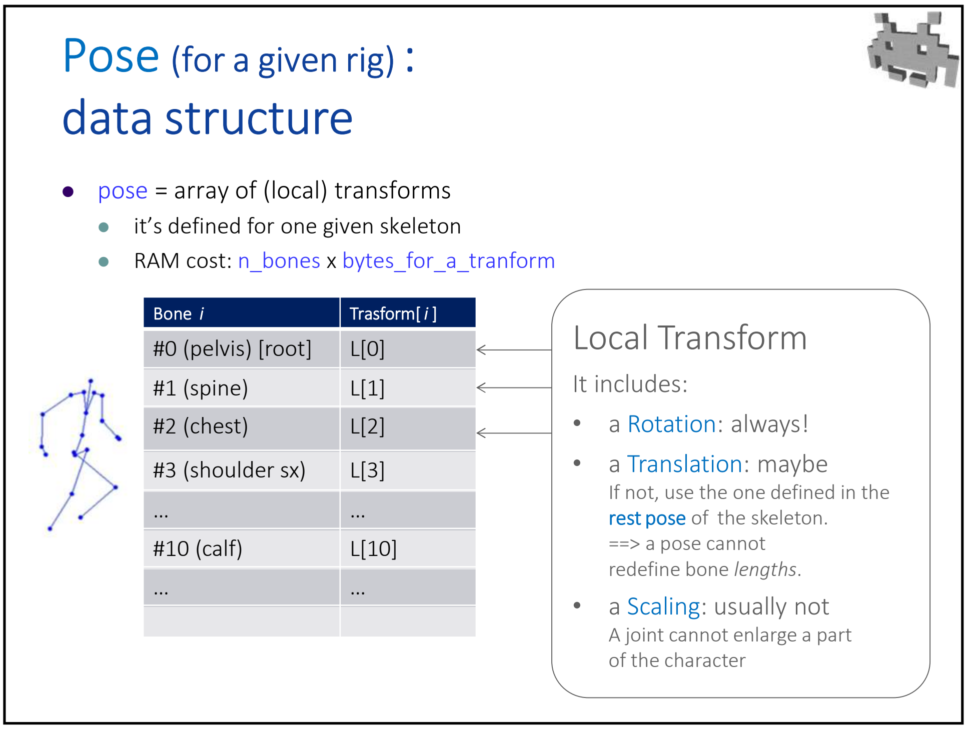
Bone transforms in a pose. E.g. for «right foot» bone:

- 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 ($R_1 \dots R_N$)
    - 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!

80



Pose (for a given rig) : data structure

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

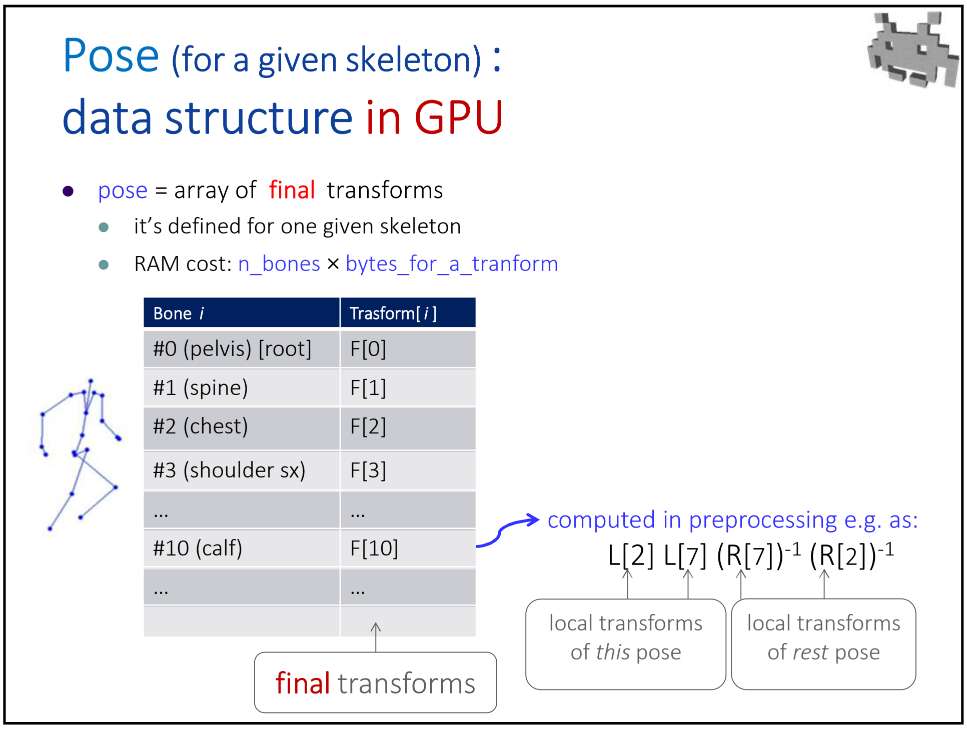
Bone *i*	Trasform[*i*]
#0 (pelvis) [root]	L[0]
#1 (spine)	L[1]
#2 (chest)	L[2]
#3 (shoulder sx)	L[3]
…	…
#10 (calf)	L[10]
…	…

Local Transform

It includes:

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

81



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

- pose = array of final transforms
  - it's defined for one given skeleton
  - 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

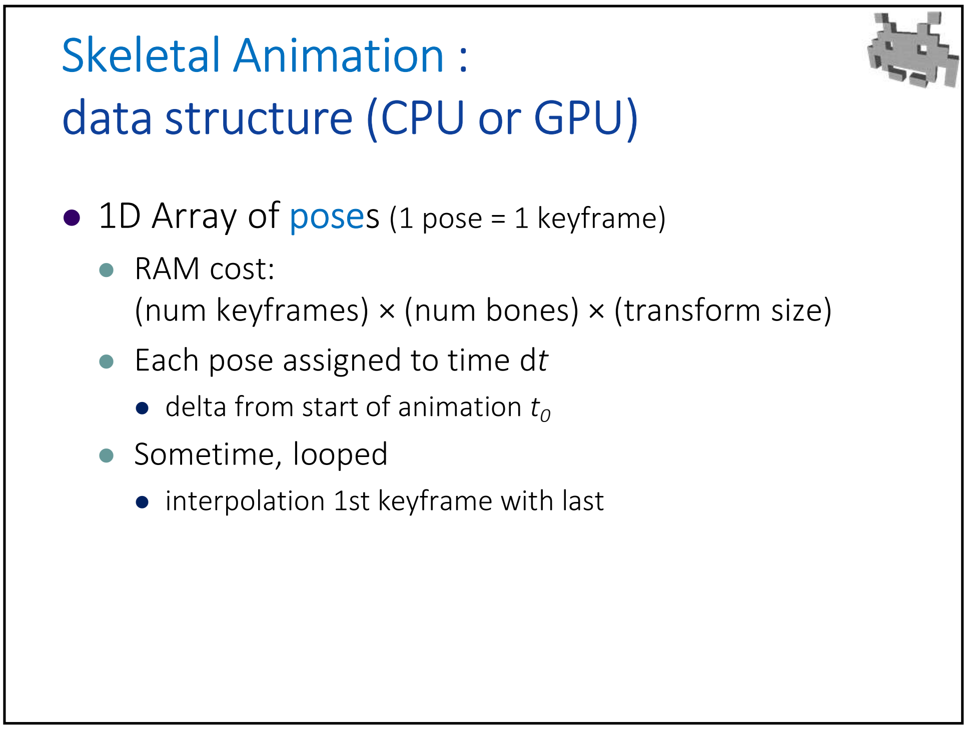
computed in preprocessing e.g. as:

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

local transforms of *this* pose

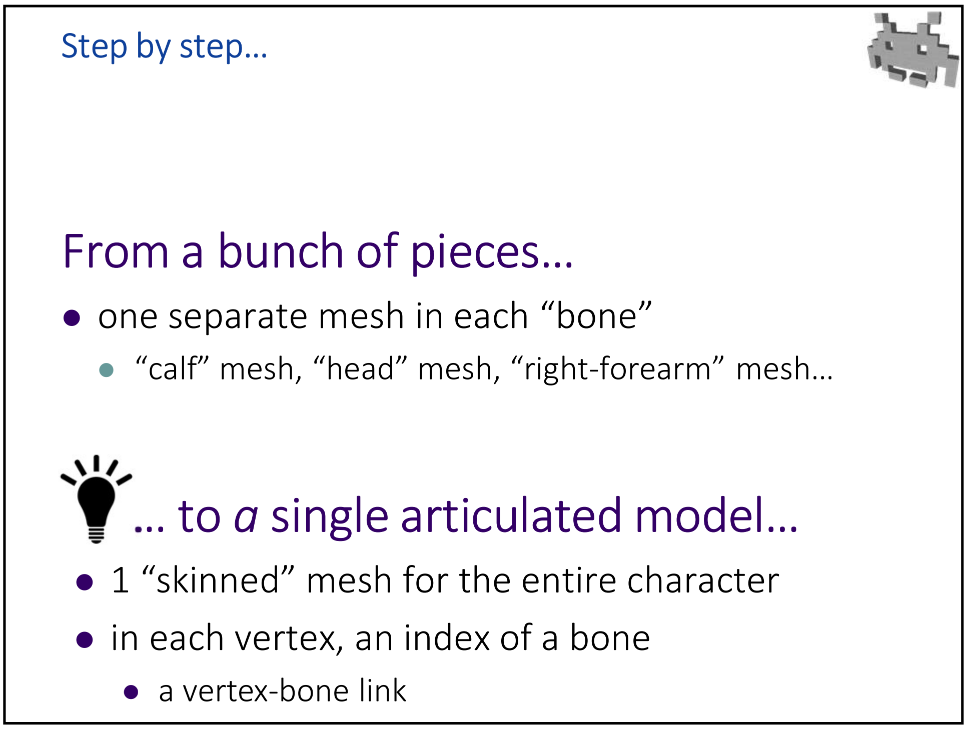
local transforms of *rest* pose

82



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



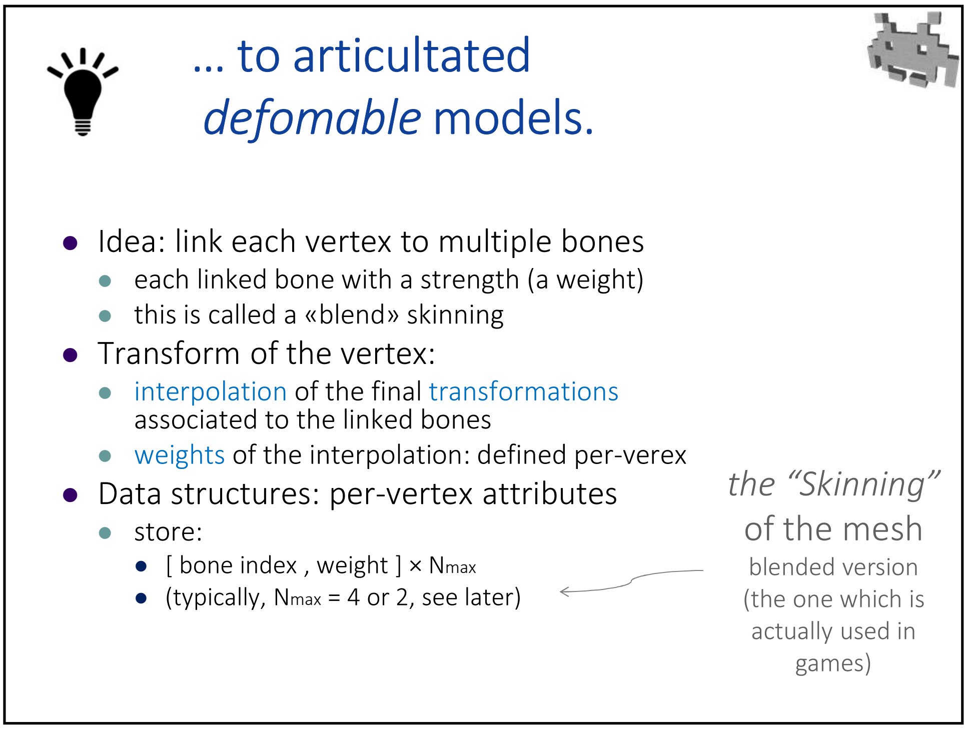
Step by step…

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

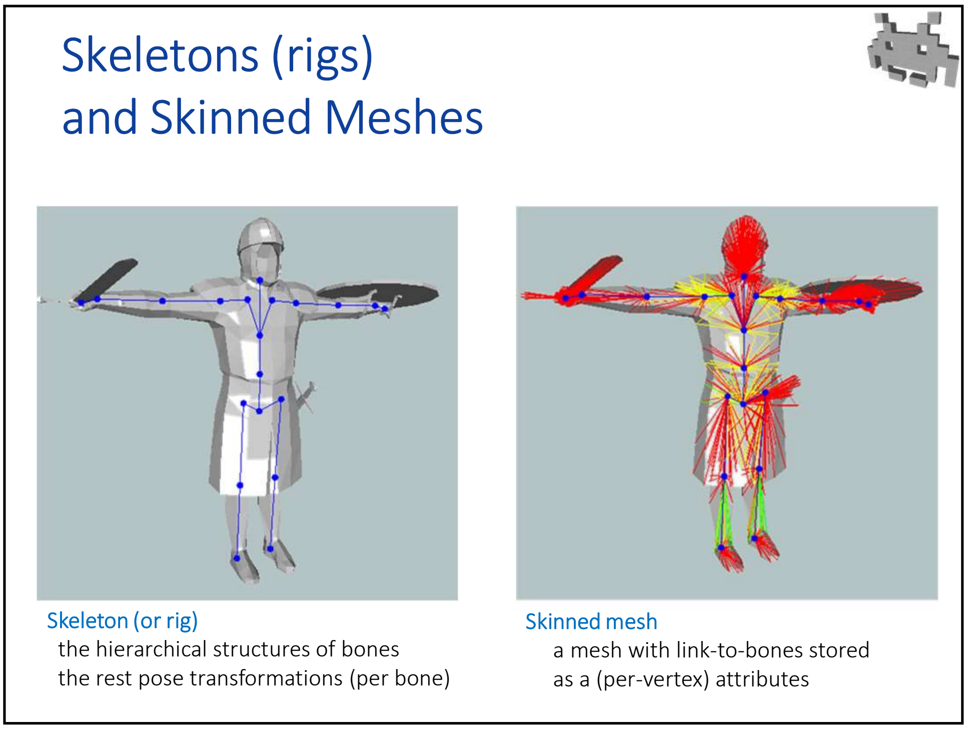


… to articultated *defomable* models.

- Idea: link each vertex to multiple bones
  - each linked bone with a strength (a weight)
  - this is called a «blend» skinning
- 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)

85



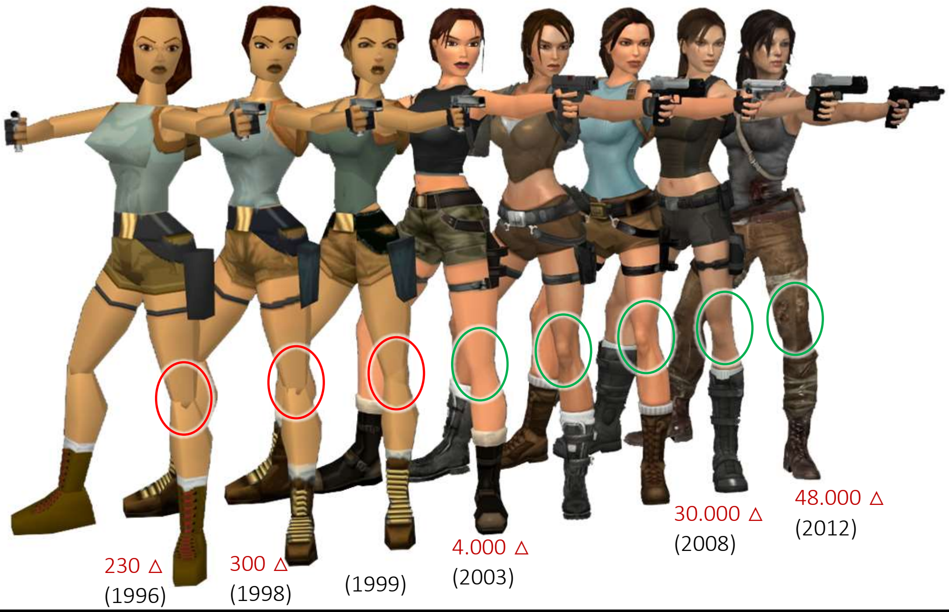
Skeletons (rigs) and Skinned Meshes

Skeleton (or rig)
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

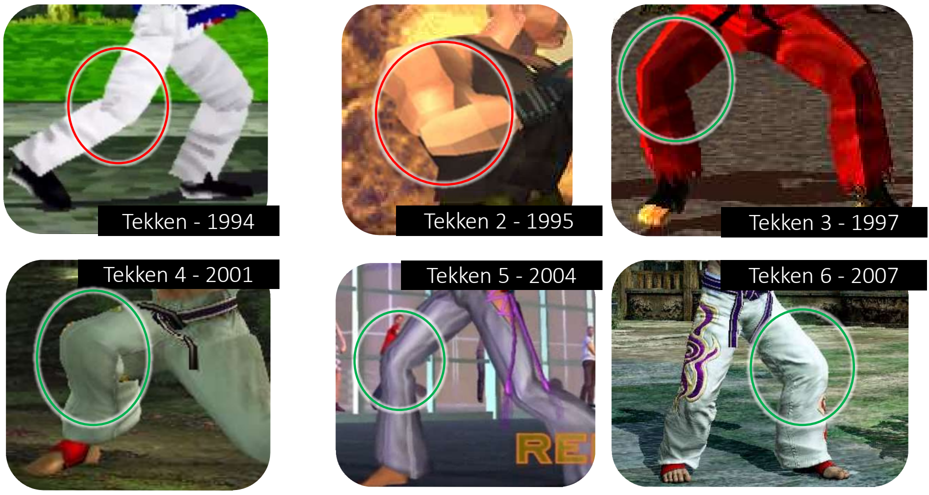
86

(this story actually happened)



87

(this story actually happened)



89

Skinned Mesh: data structure

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

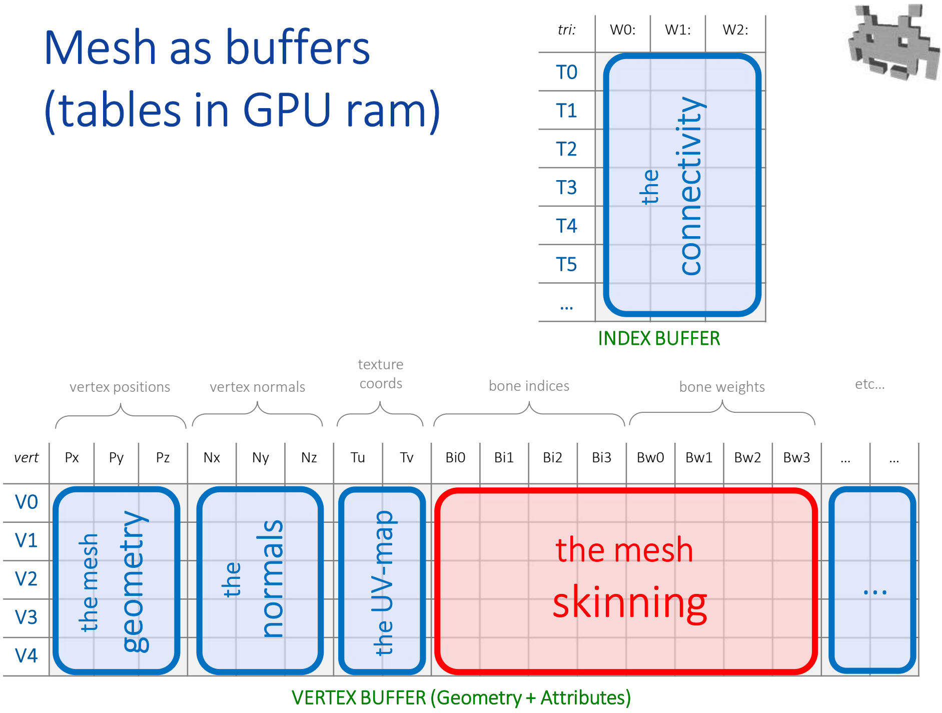
bone links

Vertex 134 →

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

91

Mesh as buffers (tables in GPU ram)



92