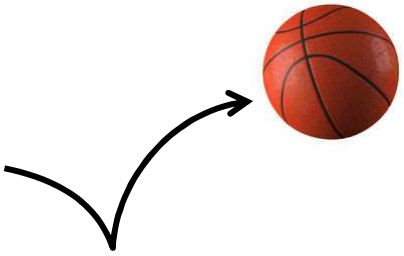
Types of animations in games

1. of rigid objects
   - animate modelling transformations



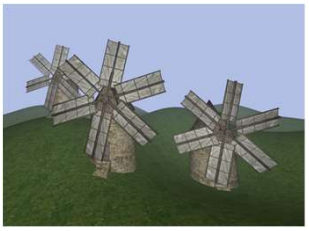
(6 DoF per object)

3

Types of animations in games

1. of rigid objects
   - or objects made of rigid sub-parts

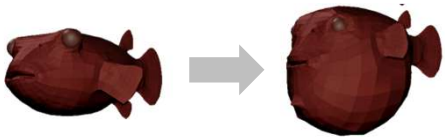




4

Types of animations in games

2. Free-Form deformations
   - generic transformations of the object



5

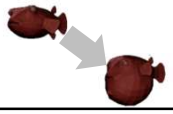
Types of animations in games

3. of articulated models
   - internal skeleton
   - most virtual characters!
   - "skinning"

6

Types of animation and DoF (per keyframe)

DoF = Degrees of Freedom

Rigid 	6 DoF per object (or, e.g., 9, with anisotropic scaling)
Articulated 	~50-100 DoF per object (e.g. 3 DoF per joint x 25 joints)
Free form 	300-10.000 DoF per object (e.g. 3 per-vertex)

7

Animations in games

Non procedural	Procedural
Assets!	**Physic engine**
Control: easy. full control by artists (e.g. for dramatic fx)	Control: hard
Realism: hard it's up to the artist skill	Realism: easy built-in physical laws
Flexibility: little Doesn't adapt to env.	Flexibility: great Adapts to env / contexts
(consumes RAM)	(consumes GPU)

8

Animations in games

- Or… a mix
  - ex1: *primary* animations: scripted
    secondary animations: computed
  - ex2: *alive* characters: scripted
    dead characters: computed (ragdolls)
  - *and more*

- a frontier in CG and Interactive techniques!

9

Summary: Types of scripted animations

- of objects made of rigid subparts
  - including joints: robots, cars…
  - → "forward kinematics animations"
    (dynamic changes of modelling transform)

- of deformable articulated objects
  - with some internal skeleton
  - e.g: most virtual characters:
    humans / animals / monsters / anything in between
  - → "skinning" / "rigging"

- of generic deformable objects
  - e.g. faces, an umbrella, stuff with membrane…
  - "per-vertex animations" / "blend shapes" / "morph targets"

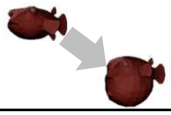
10

A digression on terminology

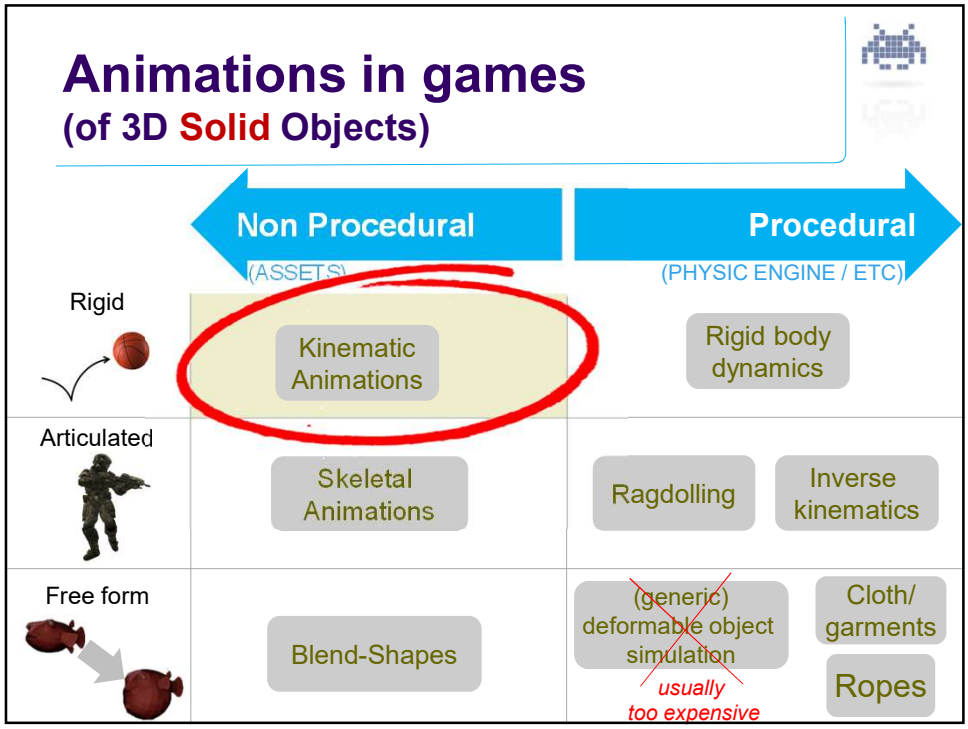
- The opposite of "**procedural**", depending on the context, can be…
  - **baked** : 'pre-cooked', 'frozen' into an asset, (when it was produced procedurally)
  - **asset** : stored as an asset (irrespective of origin), that is, read from the disk (or streamed from web)
  - **scripted** : stored as a (*simple!*) script
    (but, the procedure to create something can well be a script!
    e.g. "this *script procedurally generates* a level")
  - **(manually) designed / edited** : made by an artist (as opposed to: by a program)
  - **(fully) simulated** : the output of a (complex!) simulation

11

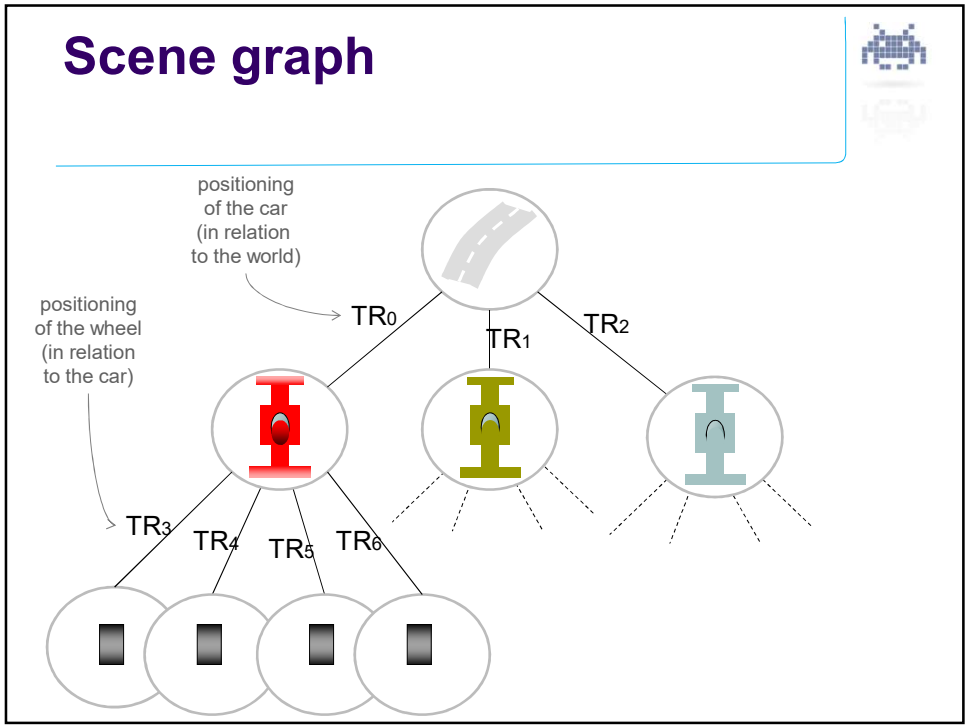
Animations in games
(of 3D Solid Objects)

	Non Procedural (ASSETS)	Procedural (PHYSIC ENGINE / ETC)	
Rigid 	Kinematic Animations	Rigid body dynamics	
Articulated 	Skeletal Animations	Ragdolling	Inverse kinematics
Free form 	Blend-Shapes	~~(generic) deformable object simulation~~ *usually too expensive*	Cloth/ garments Ropes

12

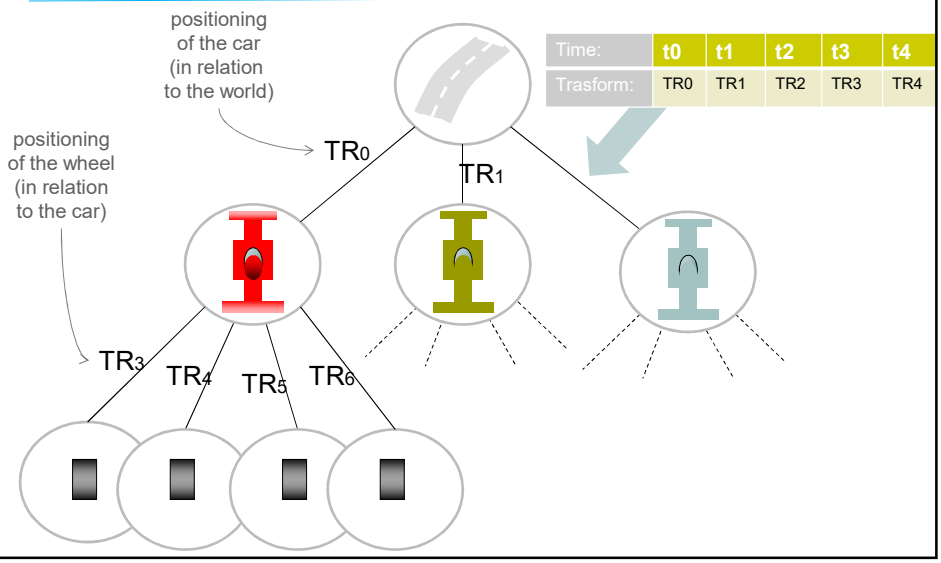

Animations in games
(of 3D Solid Objects)
Non Procedural
(ASSETS)
Procedural
(PHYSIC ENGINE / ETC)
Rigid
Kinematic Animations
Rigid body dynamics
Articulated
Skeletal Animations
Ragdolling
Inverse kinematics
Free form
Blend-Shapes
(generic) deformable object simulation
usually too expensive
Cloth/ garments
Ropes

13


Scene graph
positioning of the car (in relation to the world)
positioning of the wheel (in relation to the car)
TR0
TR1
TR2
TR3
TR4
TR5
TR6

14

Animated Scene graph…
(“kinematic” animations)



15

Kinematic animations: how?

- way 1:
  - just scripting
- way 2:
  - editing in a animation software
    - cinema 4D, blender, 3D max, …
    - (including use of I.K. as part of the interface)
  - export animation
    - as a sequence of keyframes
    - File formats: collada, fbx, …

asset: the script

DEMO !

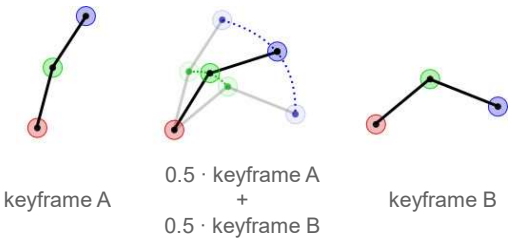
Time:	t0	t1	t2	t3	t4
A==>B:	TR0	TR1	TR2	TR3	TR4
B==>C:	TR0	TR1	TR2	TR3	TR4

asset: the animation

16

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

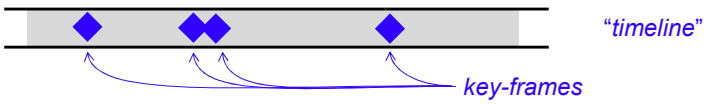
- Keyframes
  +
  interpolation



17

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

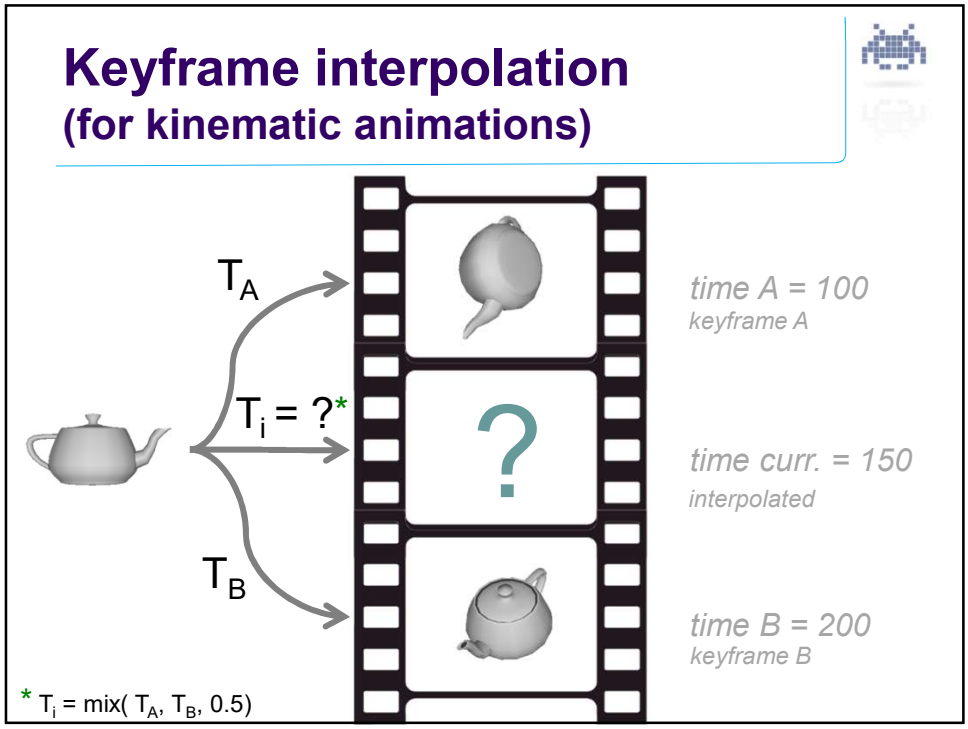
- The animator authors:
  - a set of *keyframes*
  - each with an associated time



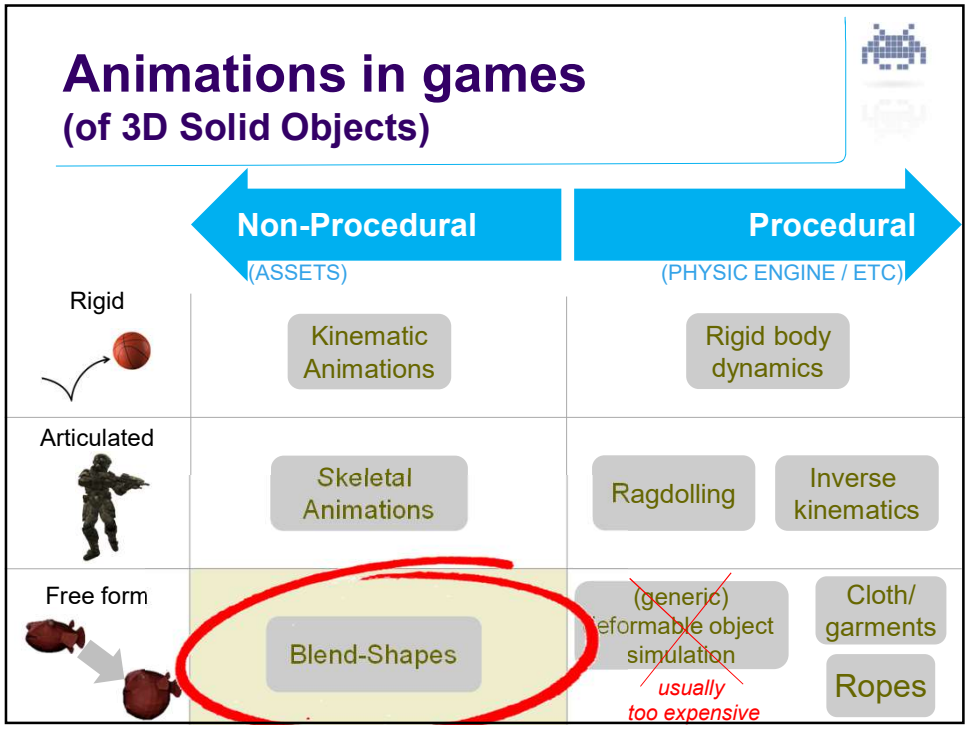
- Status of the ani at any other **frame**:
  interpolation between **keyframes**
  - saves artist work
  - saves storage

- note: keyframes distribution can be *adaptive*
  - concentrate keyframes where needed / not linear

18


Keyframe interpolation
(for kinematic animations)
T_A
T_i = ?*
T_B
?
time A = 100
keyframe A
time curr. = 150
interpolated
time B = 200
keyframe B
* T_i = mix(T_A, T_B, 0.5)

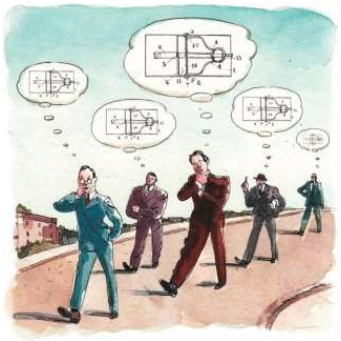
19


Animations in games
(of 3D Solid Objects)
Non-Procedural
(ASSETS)
Procedural
(PHYSIC ENGINE / ETC)
Rigid
Kinematic Animations
Rigid body dynamics
Articulated
Skeletal Animations
Ragdolling
Inverse kinematics
Free form
Blend-Shapes
(generic) deformable object simulation
usually too expensive
Cloth/ garments
Ropes

20

Asset for free-form animations: Blend shapes

- A.K.A:
  - Blend shapes
  - Per-vertex animations
  - Vertex animations
  - Face morphs
  - Shape keys
  - Morph targets
  - …



BARRY BLITT (THE NEW YORKER)

21

Blend shapes: concept

Walk cycle
(Monkey Island
LucasArt 1991)

- Animation in 2D (old school):
  a sequence of sprites
- Animation in 3D:
  a sequence of meshes?

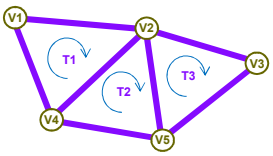
22

Reminder: representation of a mesh

- **Indexed** mode :
  - Geometry: array of vertices position
  - Attributes:
    - stored at vertices
  - Connectivity:
    - Array of triangles (or polygons)
    - Each triangle:
      - a triplet of **indexes** to vertice

23

Blend shapes
(as a data structures)



*connectivity (**indexed**)*

Tri:	Wedge 1:	Wedge 2:	Wedge 3:
T1	V4	V1	V2
T2	V4	V2	V5
T3	V5	V2	V3

geometry:

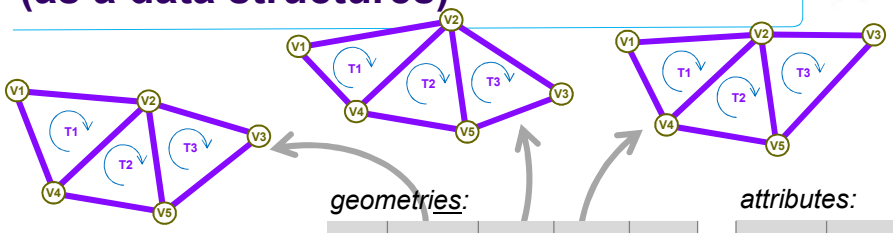
Vert:	Pos
V1	(x,y,z)
V2	(x,y,z)
V3	(x,y,z)
V4	(x,y,z)
V5	(x,y,z)

attributes:

UV	Col
(u,v)	(r,g,b)
(u,v)	(r,g,b)
(u,v)	(r,g,b)
(u,v)	(r,g,b)
(u,v)	(r,g,b)

24

Blend shapes
(as a data structures)



geometries:

Vert:	Base Shape	Shape 1	Shape 2	…
V1	(x,y,z)	(x,y,z)	(x,y,z)	…
V2	(x,y,z)	(x,y,z)	(x,y,z)	…
V3	(x,y,z)	(x,y,z)	(x,y,z)	…
V4	(x,y,z)	(x,y,z)	(x,y,z)	…
V5	(x,y,z)	(x,y,z)	(x,y,z)	…

attributes:

UV	Col
(u,v)	(r,g,b)
(u,v)	(r,g,b)
(u,v)	(r,g,b)
(u,v)	(r,g,b)
(u,v)	(r,g,b)

*connectivity (**indexed**)*

Tri:	Wedge 1:	Wedge 2:	Wedge 3:
T1	V4	V1	V2
T2	V4	V2	V5
T3	V5	V2	V3

25

Blend shapes

- A mesh with several associated geometries
- I.e. a sequence of meshes ('shapes') with
  - shared connectivity
  - shared attributies
    - (except maybe normals / tangents)
  - different geometries
  - (and shared textures as well)

or 'morph'
or (key)-'frame'
or 'shape-key'

- Variants (they are equivalent):
  - Relative mode:
    - *base shape*: explicitly stored as points
    - any other *shape*: as vectors: difference with *base shape*
  - Absolute mode:
    - each *shape* independently stored as points

26

Blend shapes
(as a data structure, e.g. C++)

- **Indexed** mesh :

```
class Vertex {
  vec3 pos;
  rgb color;
  vec3 normal;
};

class Face{
   int vertexIndex[3];
};

class Mesh{
  vector<Vertex> vert; /* geom + attr */
  vector<Face> tris;   /* connectivity */
};
```

27

Blend shapes
(as a data structure, e.g. C++)

- **Indexed** mesh :

```
class Vertex {
  vec3 pos [ N_SHAPES ] ;
  rgb color;
  vec3 normal [ N_SHAPES ] ;
};

class Face{
   int vertexIndex[3];
};

class Mesh{
  vector<Vertex> vert; /* geom + attr */
  vector<Face> tris;   /* connectivity */
};
```

28

Blend-shapes: most common file formats

- Simple:
  - **.MD5** (“quake”, valve)
  - or, just a sequence of meshes (es **.OBJ**)
    - making sure connectivity is coherent! (vertex ordering = the same)
- Complex:
  - **.DAE** (Collada)
  - **.FBX** (Autodesk)

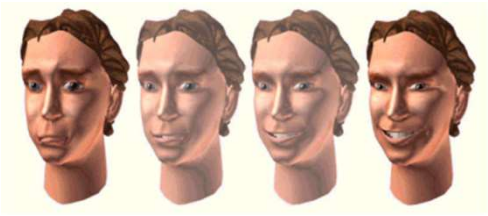
29

Interpolation between shapes

- Interpolating just the geometry:
  - with assolute BlendShapes : $shape_A \cdot w_A + shape_B \cdot w_B$
  - with relative BlendShapes: $shape_{base} + delta\_shape_A \cdot w_A + delta\_shape_B \cdot w_B$
- with: $0 \le w_A \le 0$
  $0 \le w_B \le 1$
  $w_A + w_B = 1$

30

Uses of Blend-Shapes



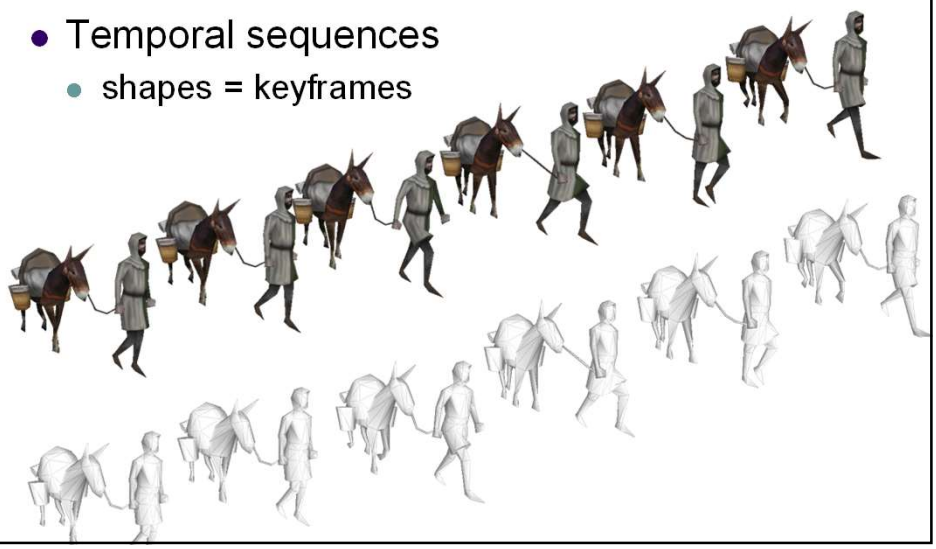
shape A shape B

(shapes = facial expressions)

31

Uses of Blend shapes

- Temporal sequences
  - shapes = keyframes



32

Use of Blend shapes

- Temporal sequences
  - shapes = keyframes

33

Use of Blend shapes as temporal sequences

- Shapes == keyframes of the animation
  - $shape_A$ with time t_A
  - $shape_B$ with time t_B
  - $shape_C$ with time t_C
  - $shape_D$ with time t_D
- current time: t with $t_B < t < t_C$
- interpolate betw. shapes: $shape_B$, $shape_C$
- weights : $w_B = \frac{t - t_C}{t_B - t_C}$ $w_C = (1 - w_B) = \frac{t - t_B}{t_C - t_B}$

34

Use of Blend shapes as temporal sequences

- Shapes == keyframes of the animation
  - $shape_A$  with time t_A
  - $shape_B$  with time t_B
  - $shape_C$  with time t_C
  - $shape_D$  with time t_D
- current time: t with $t_B < t < t_C$
- interpolate betw. shapes: $shape_B$, $shape_C$
- weights : $w_B = f\left(\frac{t-t_C}{t_B-t_C}\right)$ $w_C = f\left(\frac{t-t_B}{t_C-t_B}\right)$

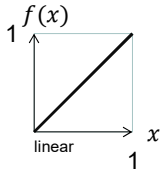
transition function

35

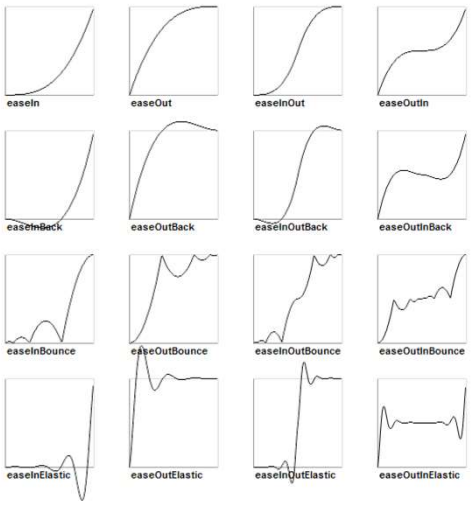
(general concept,
applies to all animation types)

Transition functions

- Not necessarily the Linear one



$f(x) = x$

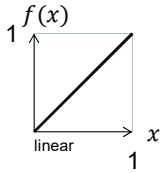


36

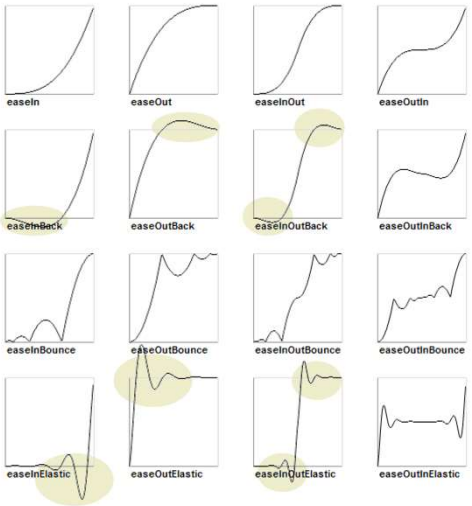
(general concept,
applies to all animation types)

Transition functions

- Not necessarily the Linear one



$f(x) = x$

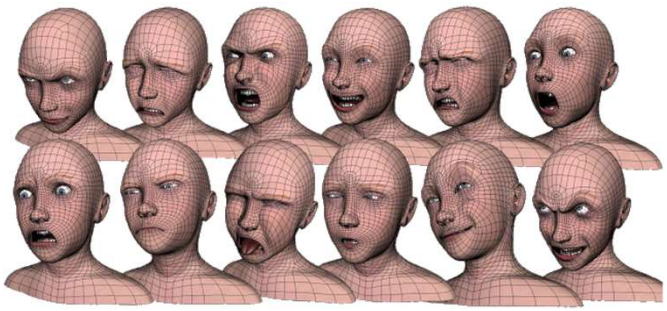


NB: → extrapolation !
(→ exaggeration)

37

Uses of Blend shapes

- Facial animations
  - (one of the commonest uses)

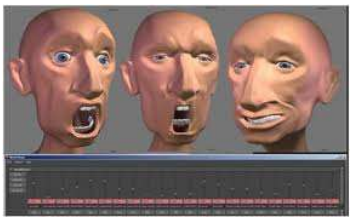


Here together with skeletal animations (see later)
(mandible, neck, eyeballs)

38

Usi delle Blend shapes

- Facial animations
  - (one of the commonest uses)



Here together with skeletal animations (see later)
(mandible, neck, eyeballs)

39

Interpolation between more than two shapes

- Blending the shapes:
  - Absolute:
    $\text{shape}_A \cdot w_A + \text{shape}_B \cdot w_B + \text{shape}_C \cdot w_C + \ldots$

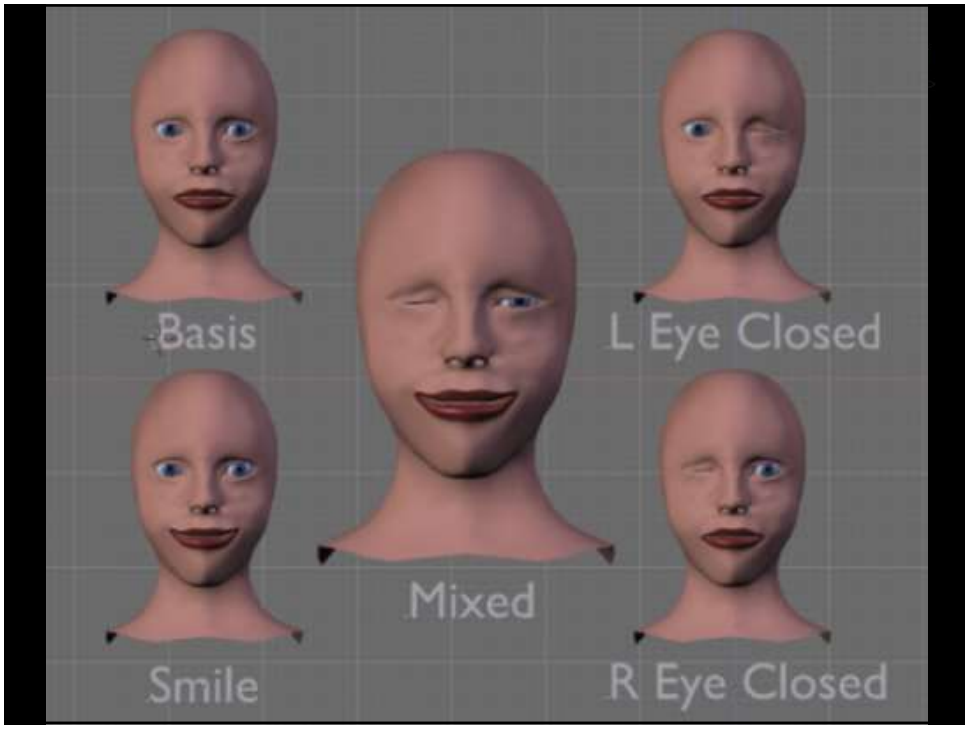
    with:
    $0 \le w_{A,B,C,\ldots} \le 1$

    or maybe not (extrapolation). Useful for…

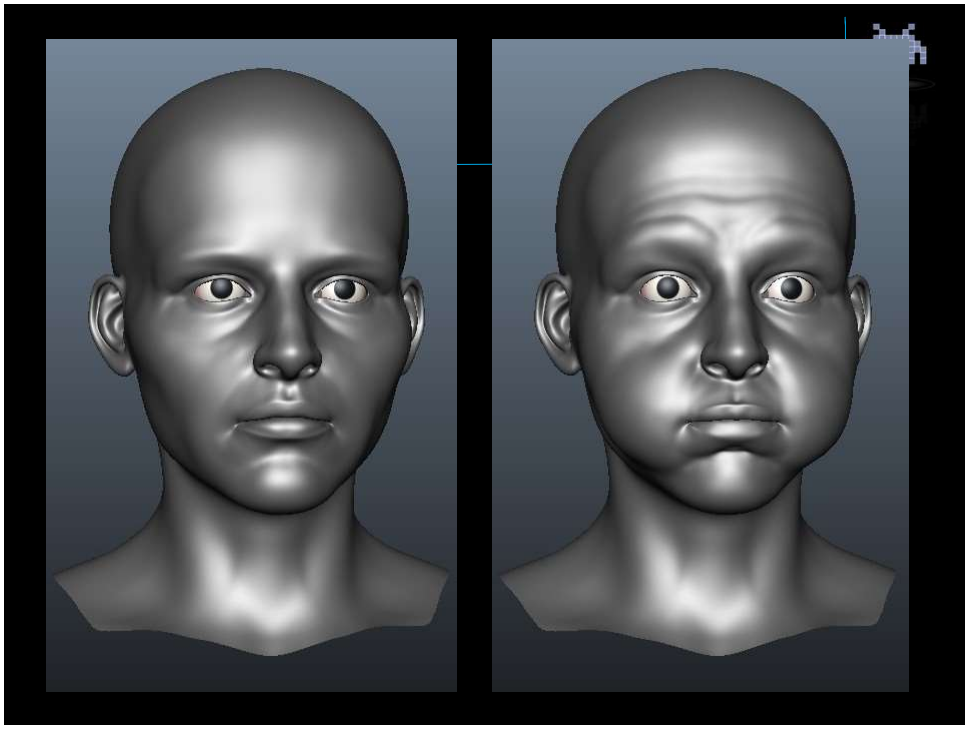
    $w_A + w_B + w_C + \ldots = 1$

  - Relative:
    $\text{shape}_{base} + \text{shape}_A \cdot w_A + \text{shape}_B \cdot w_B + \text{shape}_C \cdot w_C + \ldots$

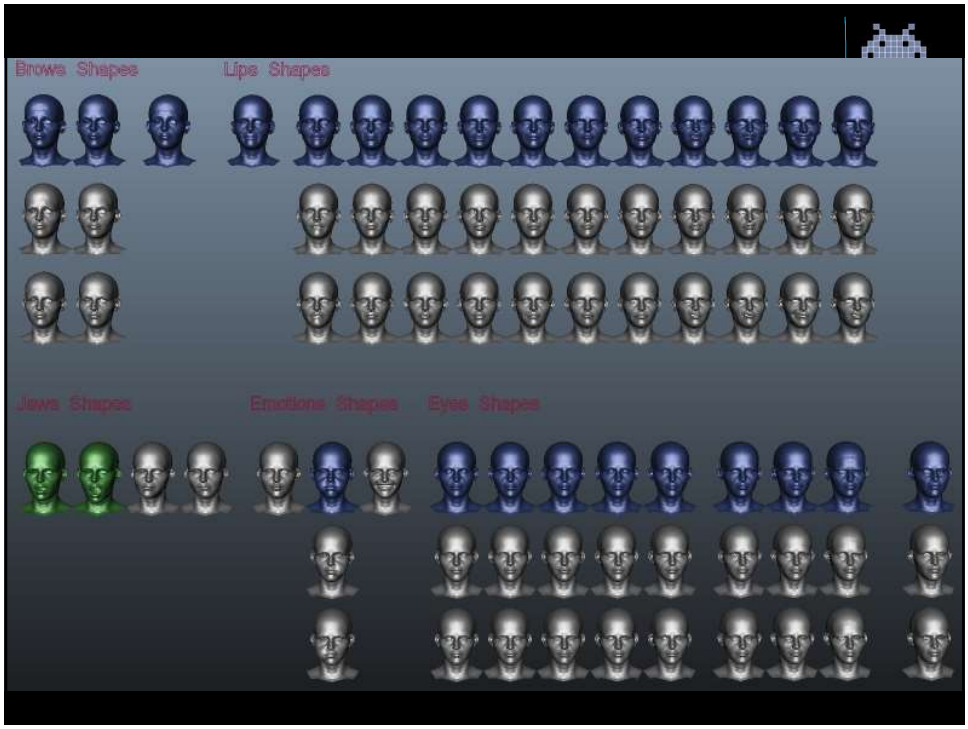
40



41



42



43

Using facial animations as Blend shapes: pipeline

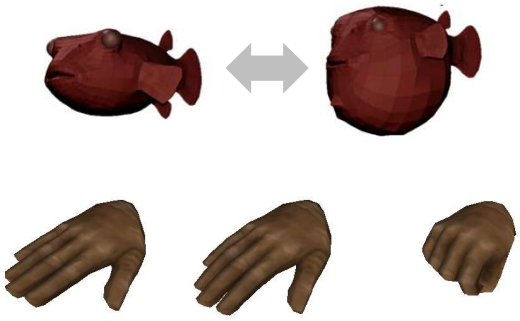
- 3D Modeller authors:
  the set of blend-shapes
- Animator (of expressions) picks:
  weights
  - eg.: with sliders
  - assisted / substituted by automatisms
    - lip sync
    - dynamically determined expressions
- Keyshape Blending: by rendering engine

[VIDEO]

44

Uses of Blend-Shapes

- Set of useful/typical physical configurations
- Baked poses



45

Uses of Blend-Shapes

- Variants of one give objects
  - (mixable!)

masculine outfit

feminine outfit

46

Uses of Blend-Shapes

- Variants of one give objects
  - (mixable!)

huamn *orc* *goblin* *dwarf*

47

Uses of Blend-Shapes

- Defines shapes of a class of objects
  - get a shape in the class = just choose the weights
    - 3D modelling at an high-level of abstraction
  - the weights "span" one **shape space**
    - one given shape = one point in the space
    - weights = coords
  - the space is the more useful the more:
    - *all and only* the reasonable shapes are represented in the space
- Typical Example: face morphologies
  - "face-space"
  - note: face morphology ≠ facial expression

48

Uses of Blend shapes

- A blend shape modelling a face space ("face-morphs")

[DEMO]

49

What a blend shape *cannot* do

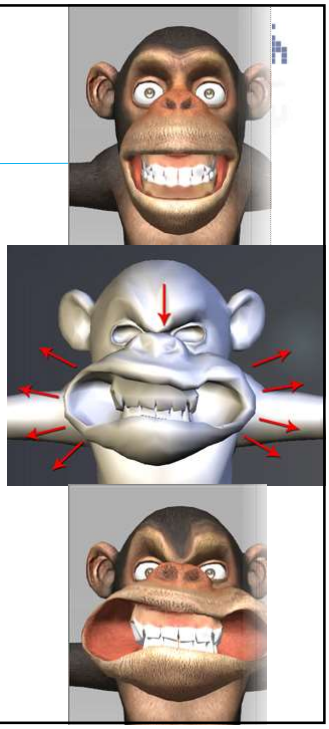
- Change connectivity
  - eg. change res, remeshing
- Change topology
  - breaking apart, fusing parts
- Change attributes
  - (eg color…)
- Change textures
  - Use a texture animation instead, maybe?

50

Blend shapes: authoring

Manual authoring:

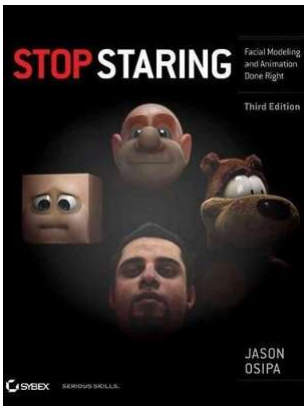
1. Editing base shape
   - including:
     uv-mapping, texturing, etc.
2. Re-edit it
   for each shape-key!
   …while preserving:
   connectivity,
   textures, etc
   - low poly editing
   - or with subdivision surfaces…
   - or with parametric surfaces…
   - or with scupting.



51

Blend shapes: authoring

- Handbook for blend-shape based face animation:
  - "Stop Staring" (3d edition) Jason Osipa
  - Covers: style, expression…
  - Non technical (high level)
  - Not about specific tools e.g. Blender, Maya



52

Blend shapes: hot to obtain them

- Capture:
  - 3D acquisition of base shape B0
    - (including: simplification, remeshing, uv-mapping, etc)
  - capture subsequent shapes B1, B2…
    - e.g. real-time (kinect), o 3D scanning for each shape
  - compute a morph B0 => B1
    - “non rigid mesh alignment”

[VIDEO]

53

Blend shapes: pro and con

- ☺ Flexible, expressive, huge number of DOF… too much?
  - ☹ RAM cost
  - ☹ Work intensive to construct

(but, not as bad as old sprites,

because (1) sharing of connectivity, textures, attributes
(2) keyframes / interpolation!

- ☺ Easy to use / efficient, once they are built
  - just define global weights

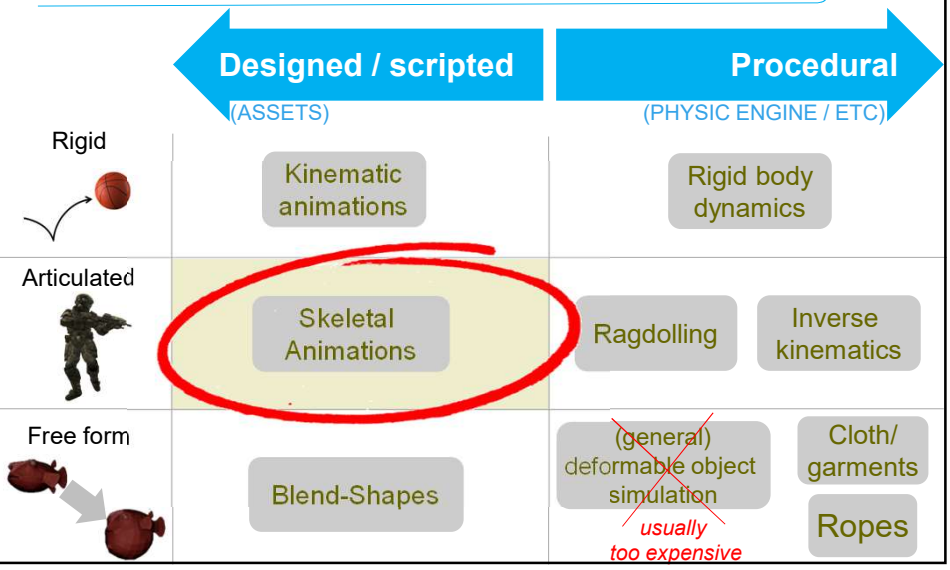
54

Blend shapes: open challanges

- Capturing from a stream of meshes
- Compression
  - eg: prediction + store corrections + Huffman
- Streaming
- LOD-ding

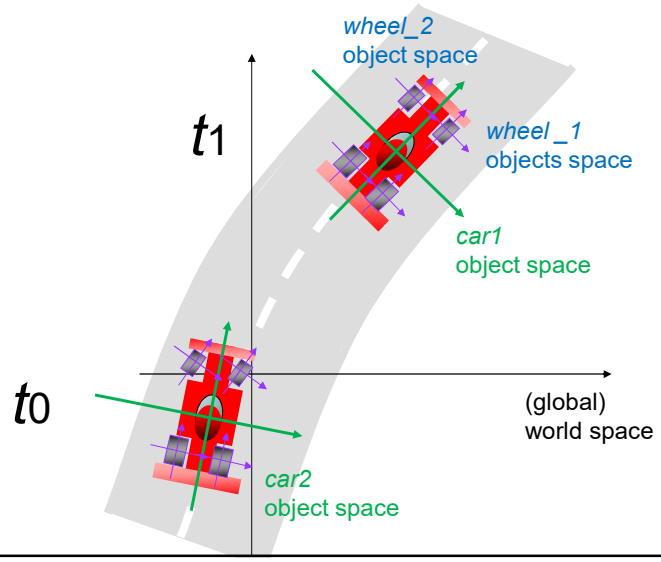
55

Animations in games (of 3D Solid Objects)



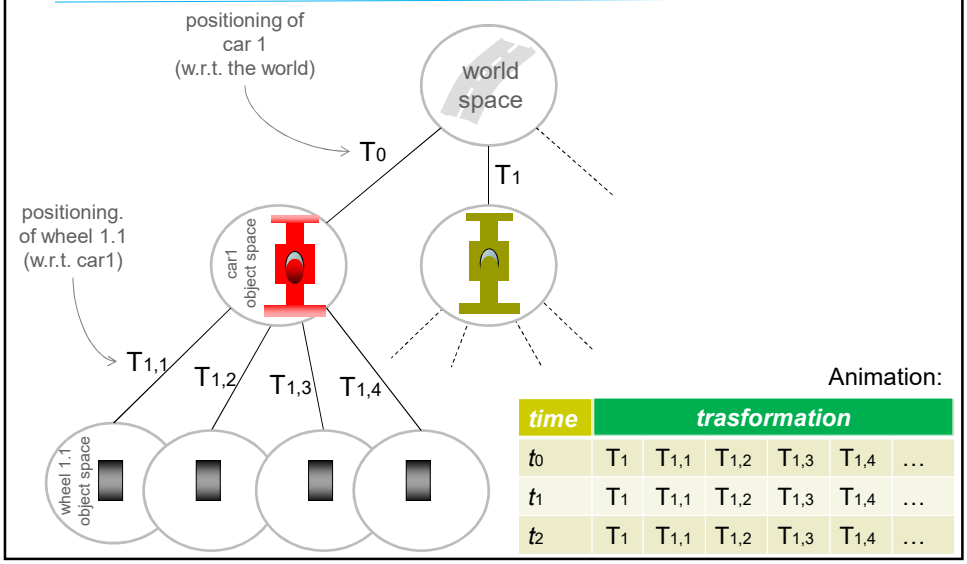
56

Step by step…
Kinematic animations



57

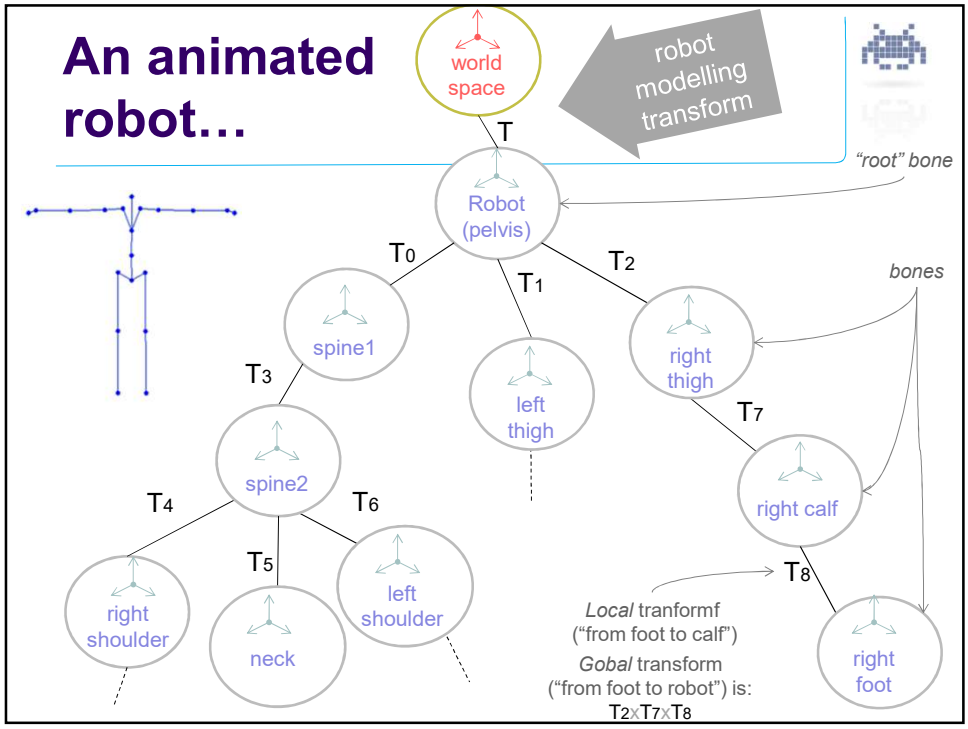
Step by Step…
Kinematic animatios



Animation:

time	trasformation					
t_0	T_1	$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,4}$	…
t_1	T_1	$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,4}$	…
t_2	T_1	$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,4}$	…

58



59

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 (many "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!)

60

… to articulated models…

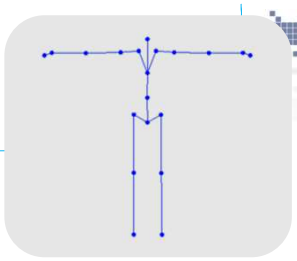
- Idea: one mesh, but *skinned*
  - 1 mesh per the entire character
  - a new attribute per vertex: *index of bone*
  - the 3D model can now be animated!

"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)

61

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

62

Pose: data structure

One trasformation for each bone *i*

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

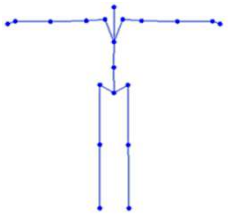
often, only the rotation component

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

63

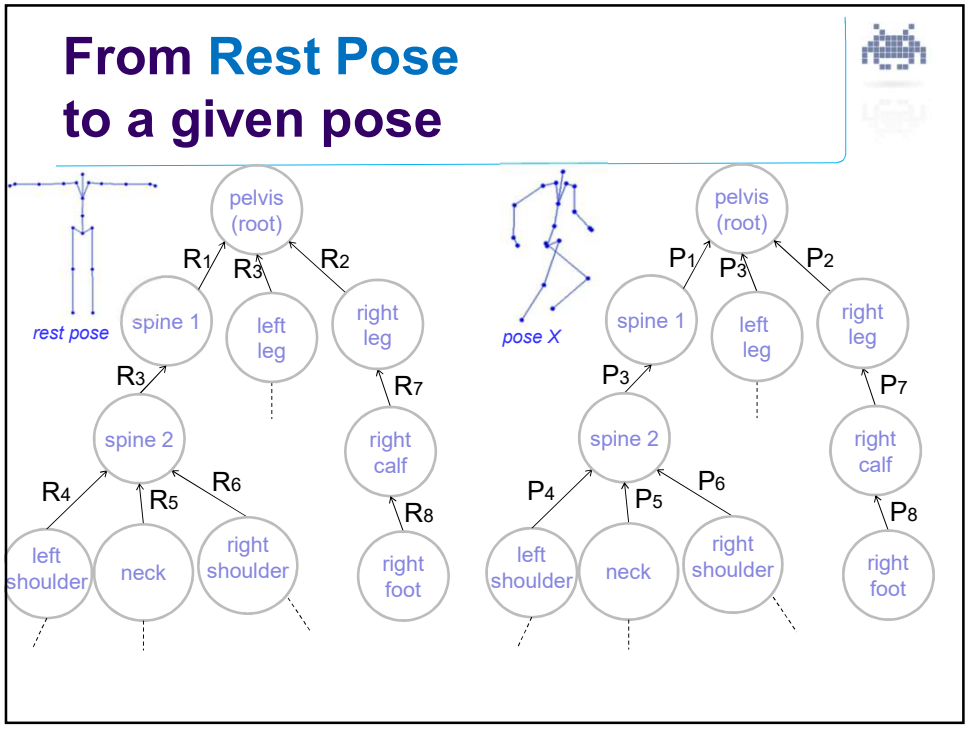
Rig (or skeleton): data structure 2/2

1. **Hierarchy** (tree) of **bones**
   - a **root bone** on top

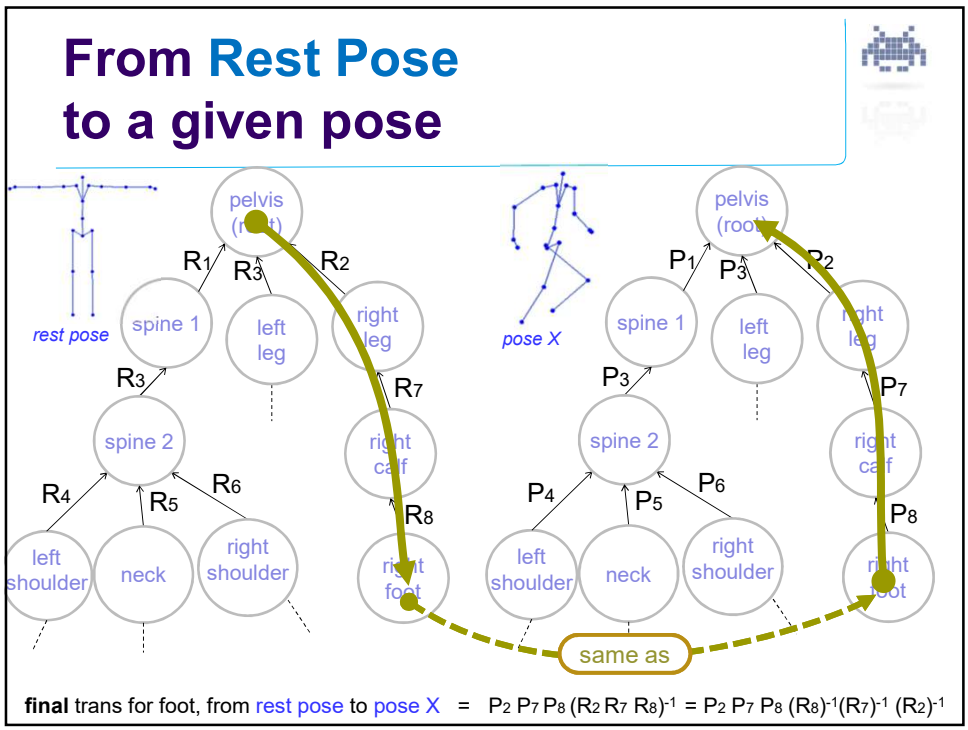


2. A special **pose** «**rest pose**»
   - 3D models are to be modelled in this pose
   - also: «T-pose», «T-stance»,
     - same resason why T-shirts are called T-shirts ;)

64


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

65


From Rest Pose
to a given pose
rest pose
pose X
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

66

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»* to *«character»*
  - uses the Hierarchy of the **Skeleton**
    - once global t are computed, Hierarchy's no longer needed!
- **Final** Transform: $P_2\ P_7\ P_8\ R_8^{-1}\ R_7^{-1}\ R_2^{-1}$
  - from *«character»* in rest pose to *«character»* in dest. pose
  - uses the Rest Pose of the **Skeleton** (R_1 … R_N)
    - once it's computed, Rest Pose no longer needed either!

the object frame of the character, i.e. the frame of the **root bone**

the space where **mesh vertices** are defined!

68

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

Osso *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

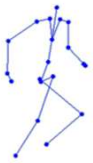
It includes:

- 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: maybe
  If not ==> a pose cannot redef the *size* of the body part.

69

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 x bytes_for_a_tranform



Osso *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]
…	…

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

final transforms

70

Skeletal Animation : data structure (CPU or GPU)

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

71

Step by step…

1. From a bunch of pieces…

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

… to articulated models…

- 1 mesh for the entire character
- in each vertex, a link to a bone

72

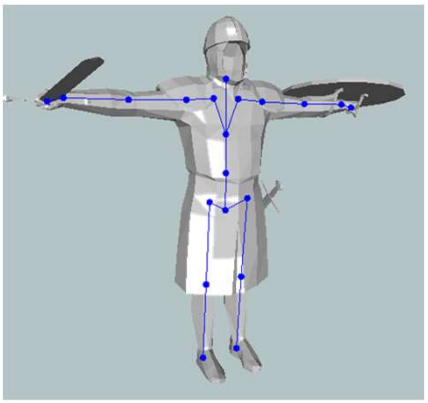
… to articultated *defomable* models.

- Idea: link each vertex to >1 bones
- Transform of the vertex:
  - interpolation of the trasformations associated to the linked bones
  - weights of the interpolation: defined per-verex
- Data structures: per-vertex attributes
  - store:
    - [bone index , weight] x N_{max}
    - (typically, N_{max} = 4 or 2, see later)

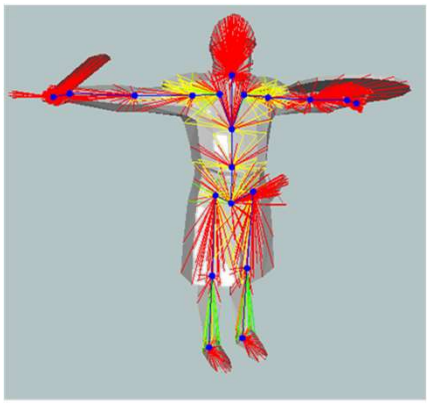
the “Skinning” of the mesh
ver 2.0
(the real one)

73

Content creation Tasks: Rigging & Skinning (of a 3D mesh)



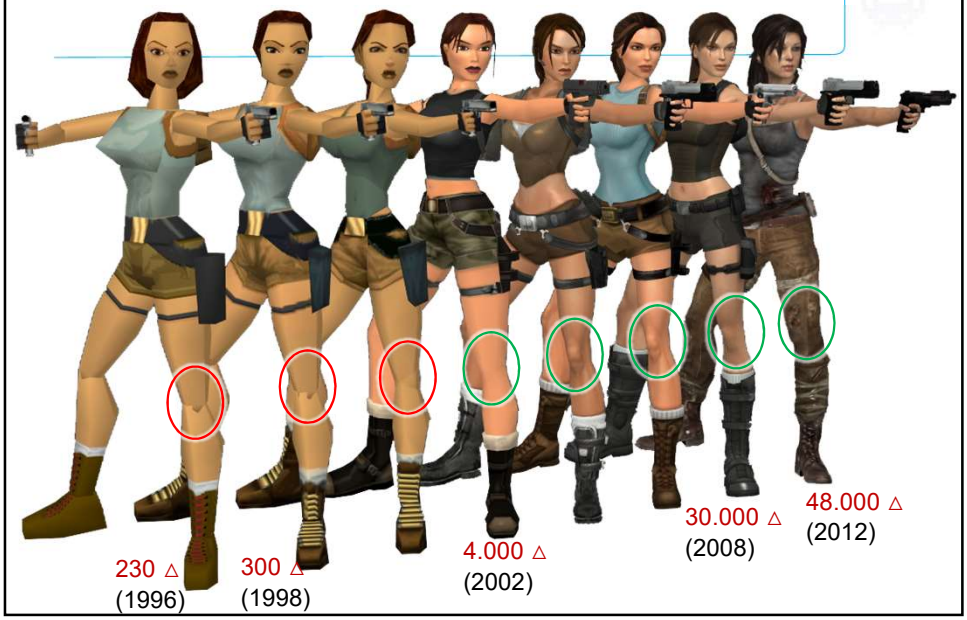
Rigging – authoring of a rig
defining the skeleton
(often: also of the controls
to define poses for it)



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

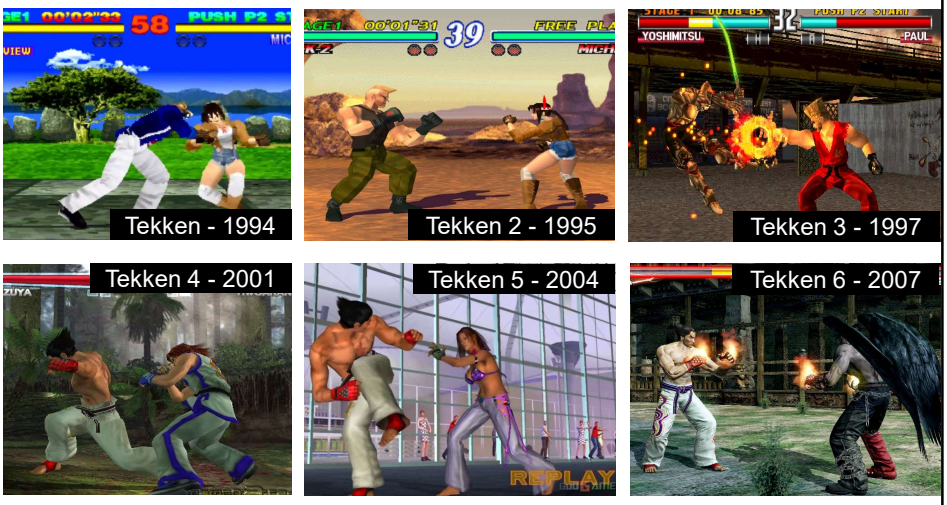
74

(this story actually happened)



75

(this story actually happened)



76

(this story actually happened)

77

Skinned Mesh: data structure

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

Vertex **120** →

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

79

N_max = How many bone links for each vertex

- It's a call of the Game engine!
- typical used value:
  - **1** (rigid pieces) (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 un game tool)
  - e.g.: Unity does this during skinned mesh import
    (if asked)

80

(but why put an hard-bound on bone links?)

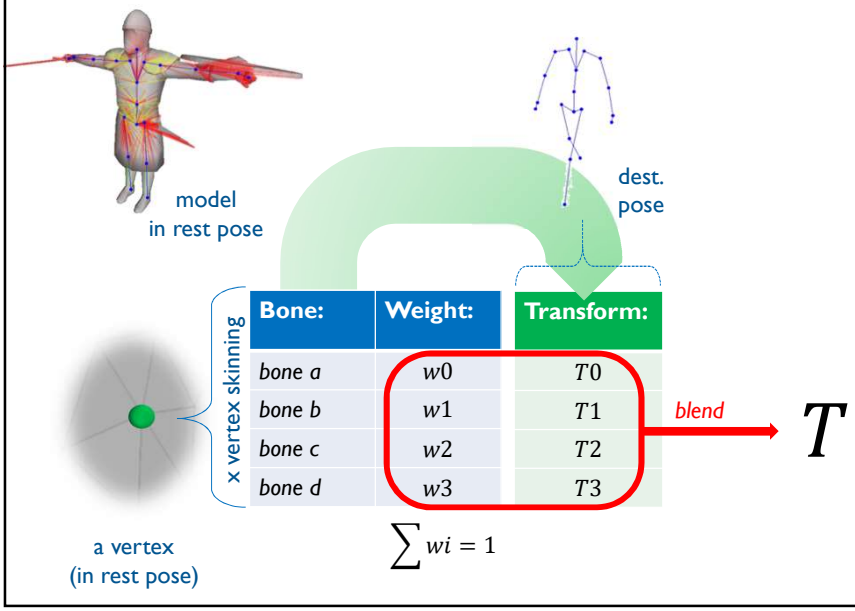
- Reduces performance cost
  - N_{max} tranforms need be interpolated in GPU
    - in vertex shader
  - GPU = no good at control:
    - always uses exactly N_{max} trasf
    - unused bones: weight = 0
- Reduces GPU RAM cost
  - not only fewer data to store, but:
  - fixed leght arrays: the only way in GPU
    - N_{max} (index,weight) pairs
    - even where fewer are locally needed (e.g. 1 bone, weight automatically 1)

es:

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

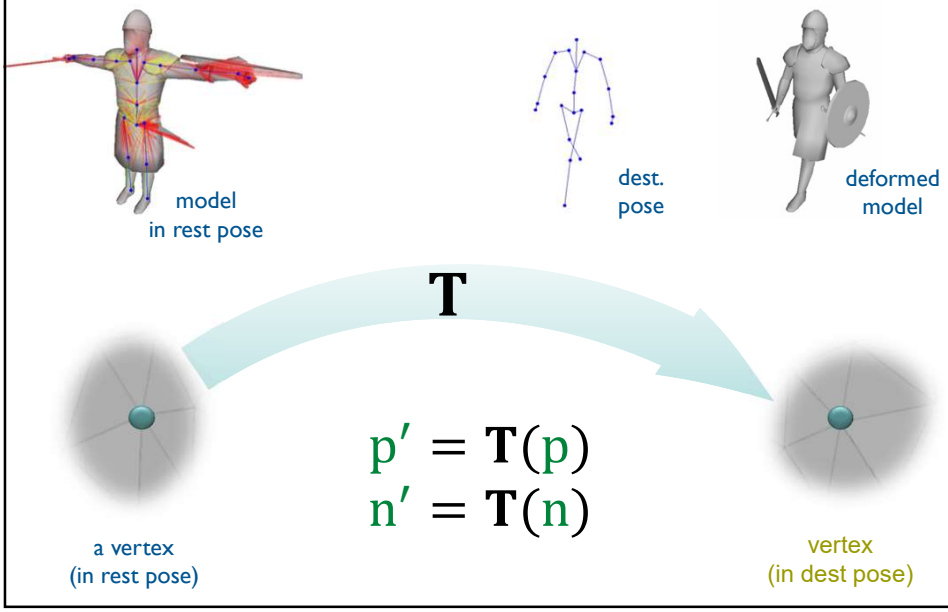
81

Skinning - how it works (in GPU)



82

Skinning - how it works (in GPU)



83

GPU real time Skinning – two variants

(a choice of the rendering engine)

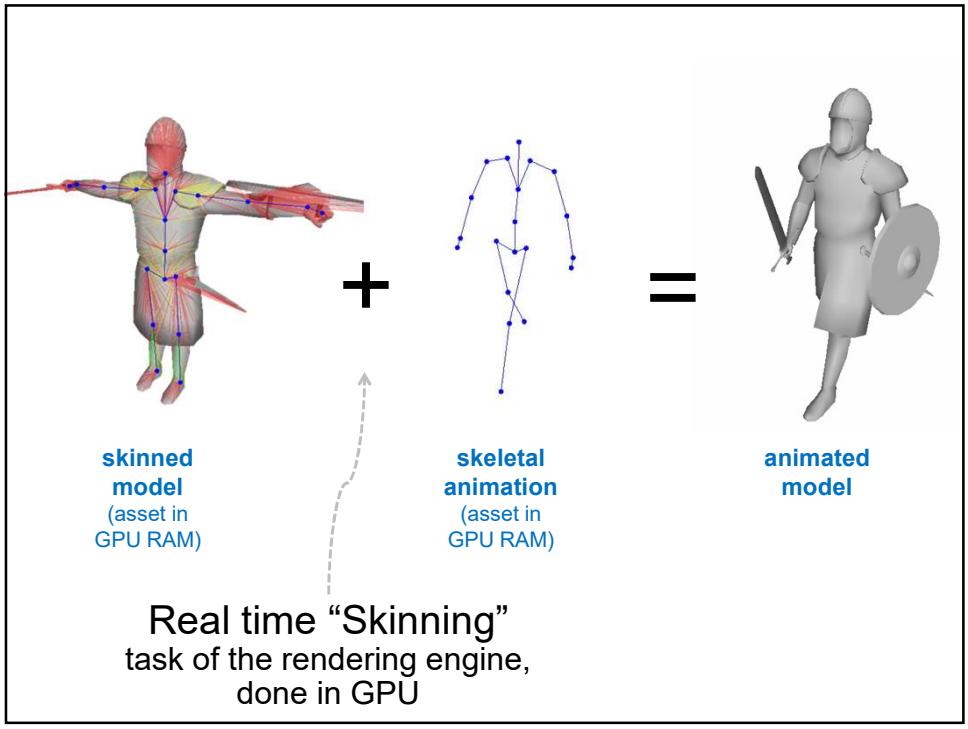
Bone:	Weight:	Transform:
bone a	$w0$	$T0$
bone b	$w1$	$T1$
bone c	$w2$	$T2$
bone d	$w3$	$T3$

blend → T

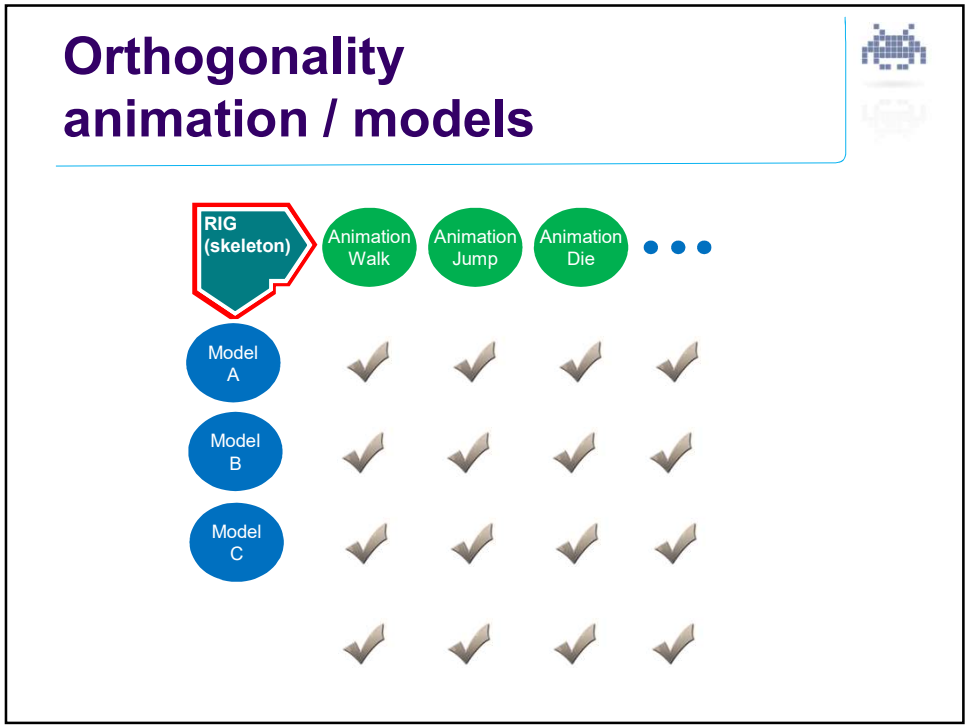
How this is done?

- linear interpolation of *matrices*:
  "Linear Blend Skinning"
  (old school, still very much used)
- *quaternion* based interpolation:
  "Dual Quaternion Skinning"
  (more complex, fewer artifacts)

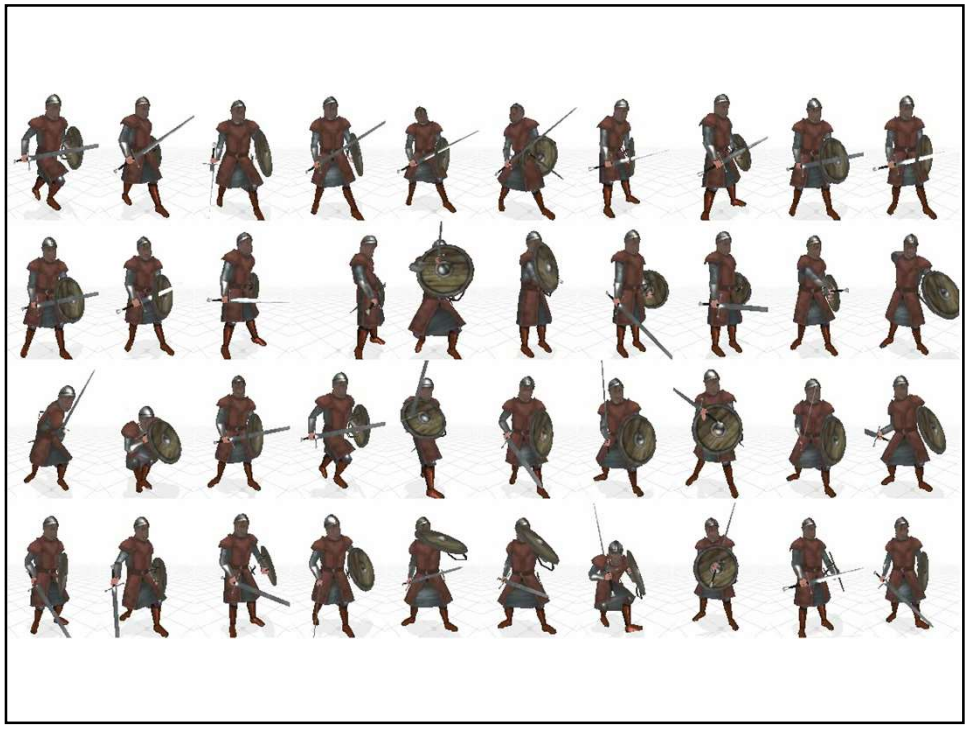
84


+
=
skinned model (asset in GPU RAM)
skeletal animation (asset in GPU RAM)
animated model
Real time “Skinning”
task of the rendering engine, done in GPU

86


Orthogonality animation / models
RIG (skeleton)
Animation Walk
Animation Jump
Animation Die
Model A
Model B
Model C

87



88

89

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

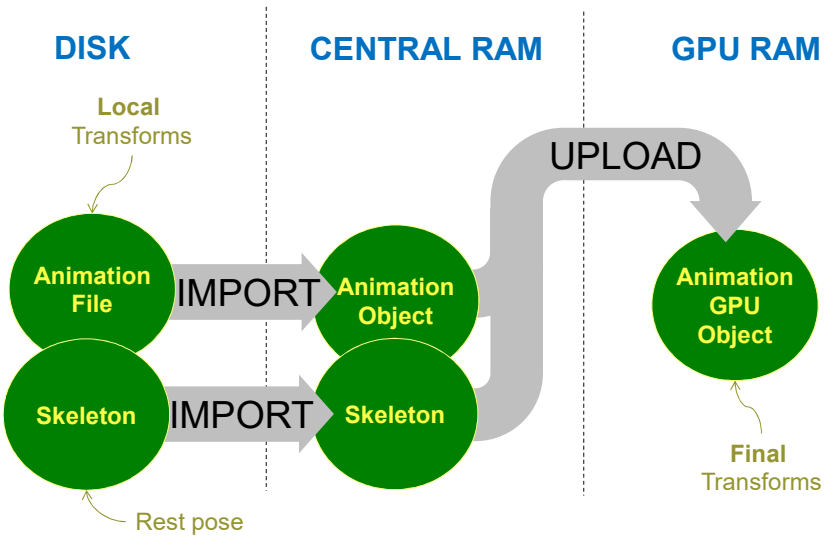
- **Rig** (or skeleton)
  - Tree di bones (ossa)
  - ∀ bone => reference frame (in rest pose)
    - (reference frame root bone = objects space)
- **Skinned** 3D Model
  - Mesh with links: vertices => bones
  - ∀ vertex: attributes: [bone index , weights] x N_{max}
- **Skeletal animations**
  - Sequence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

example of file formats (for all three):

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

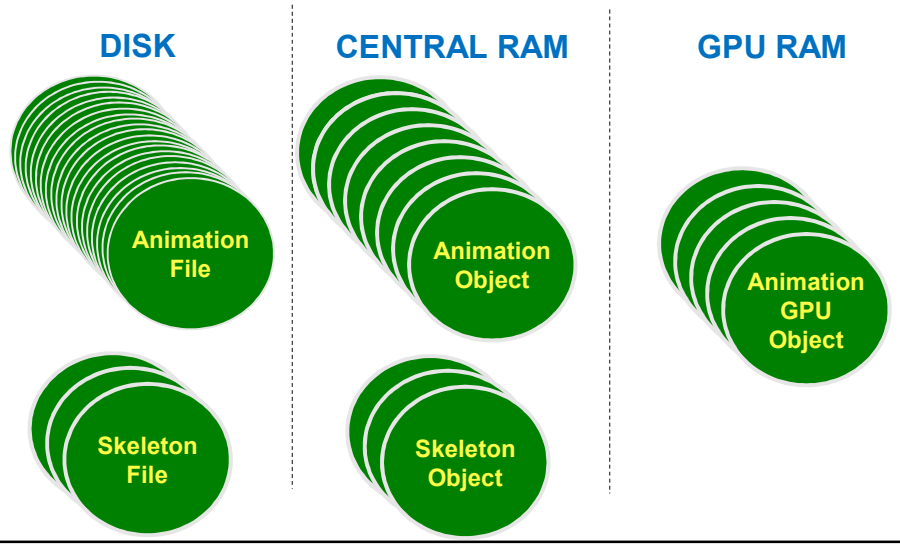
90

Life of Animation Assets in a Game Engine



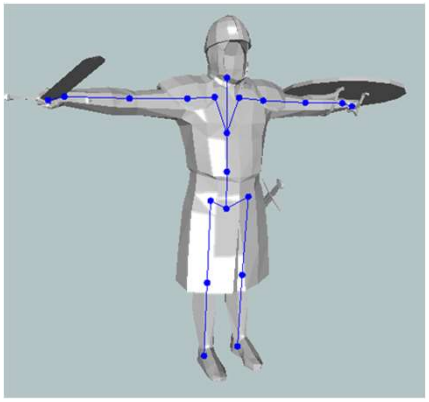
91

(Once again, Memory Management)

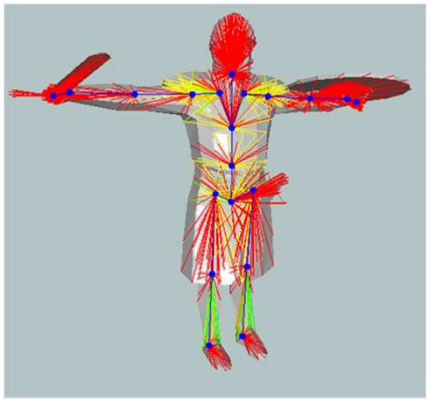


92

Content creation Tasks: Rigging & Skinning (of a 3D mesh)



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



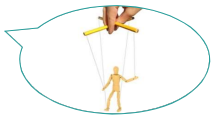
Skinning – authoring of the skinning
"paint" of (weighted) links
between vertices and bones

93

Content creation Tasks: Rigging & Skinning (of a 3D mesh)

Digital modeller task! (helped / replaced by automatic algorithms)

- Rigging :
  - define a skeleton (with a rest pose)
  - inside one mesh, (or a set of meshes: a *shared* rig)



- Skinning (of a mesh):
  - painting link vertex-bones

Ditigal animator task

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

94

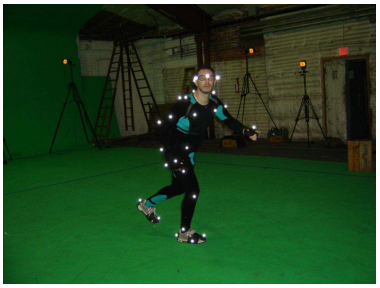
Skeletal animations: authoring / obtaining them

- 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…

95

Skeletal animations: authoring / obtaining them

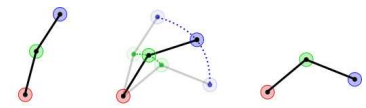
- Motion capture:



96

Interpolation poses

- any two poses can be interplated!

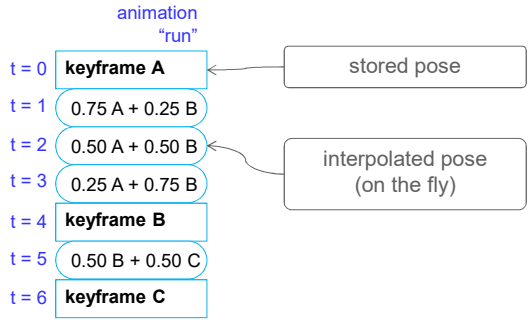


- just interplolate the per bone transform
- (note: requires recomputation of final transf from local ones)

97

Pose = keyframe

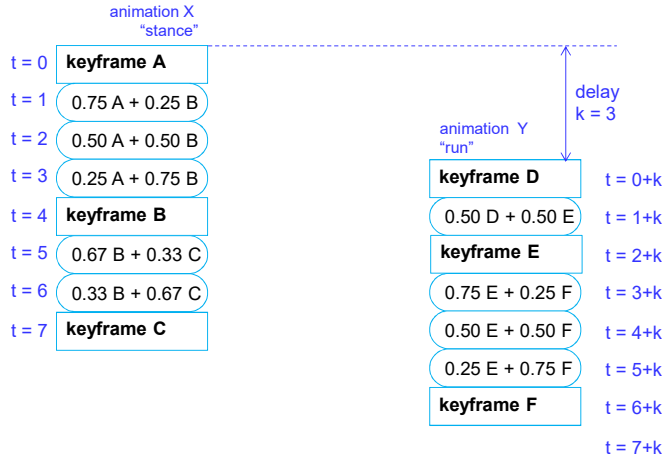
- Compress animations



98

Interpolation of poses (at runtime): transition between animations

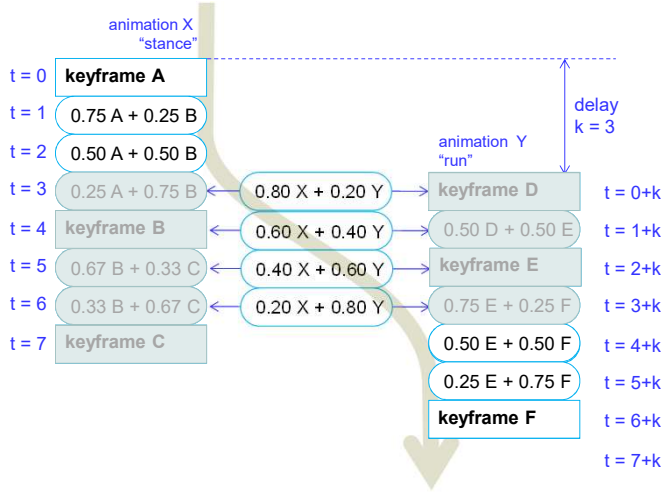
- Eg: from *stance* to *run*



99

Interpolation of poses (at runtime):
transition between animations

- Eg: from *stance* to *run*



100

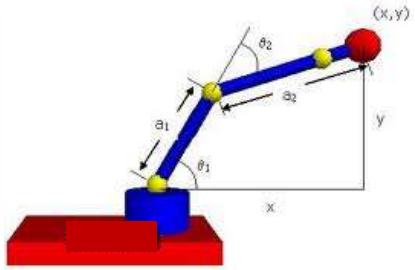
Applying poses:
Dual Quaternion Skinning

- A variant:
  per-bone transforms are stored as
  dual quaternion
  - isometries only
  - (arguably) better quality
    - (better interpolation)
  - > GPU cost
    - (necessary ops: ~ +50%)
- LBS (Linear Blend Skinning) or DQS?
  - a call of the game engine

101

Forward kinematic VS inverse kinematic

- (in robotics)



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

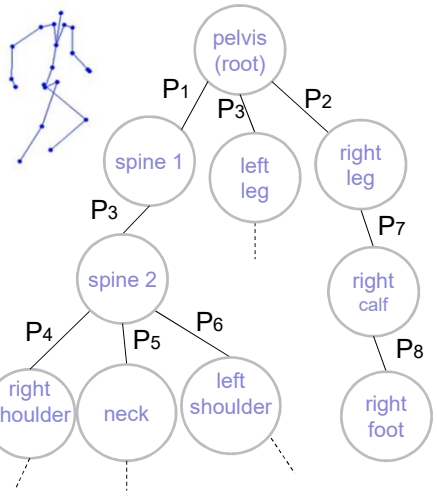
VS

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

103

Inverse Kinematic (IK): an useful tool

- Forward kinematics:
  - "given local transfer $P_1, P_2 \dots P_N$, (as angles) where does the foot go?"
  - (one solution, trivial)
- Inverse kinematics
  - "if I need the foot to be in pos p, how to set $P_1, P_2 \dots P_N$, ?"
    - (under constraints, e.g. DoF at joints)
  - (no one solution, not trivial)



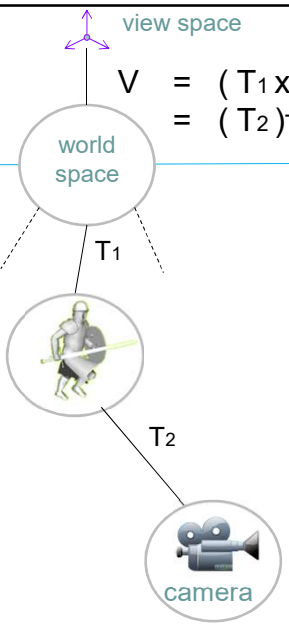
104

Inverse Kinematic (IK): an useful tool

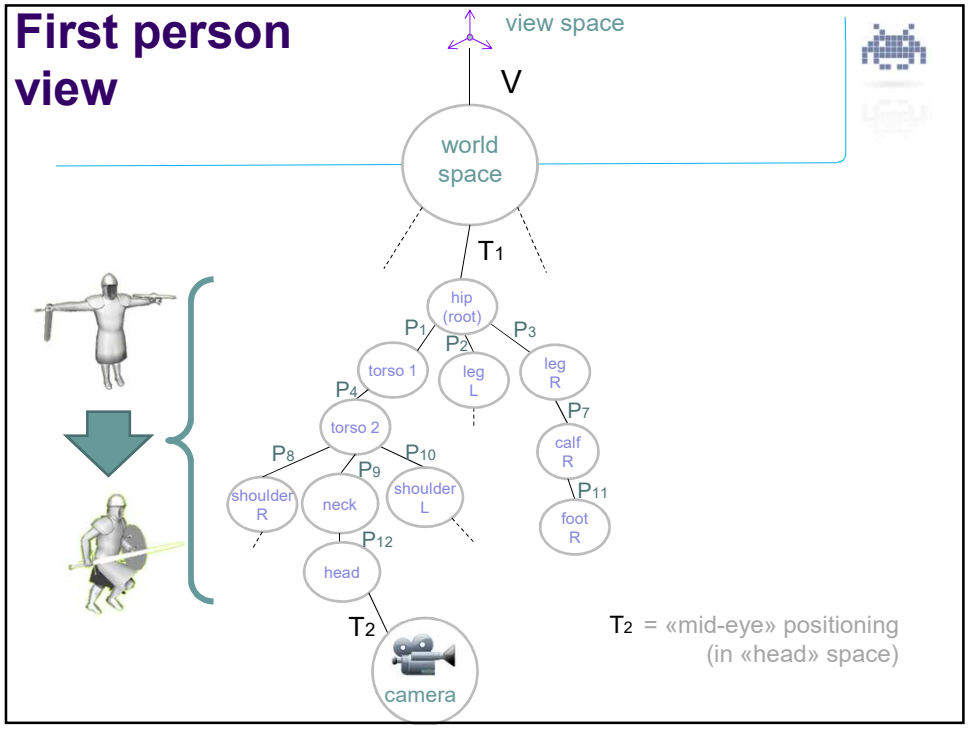
- Two uses:
  - in preprocessing (helping the task of the animator)
  - in real time (done by the game engine)
    (e.g. in unity: API in scripts)
- Examples of real-time uses:
  - Exact positioning of feet on ground
  - Exact positioning of hand to object to be grabbed
  - Hands need to be joined
    (e.g. 2-handed weapon wielding)
    - (e.g. making the system correct for small changes in bone lengths)
    - (e.g. during interpolated keyframes)
  - etc.

105

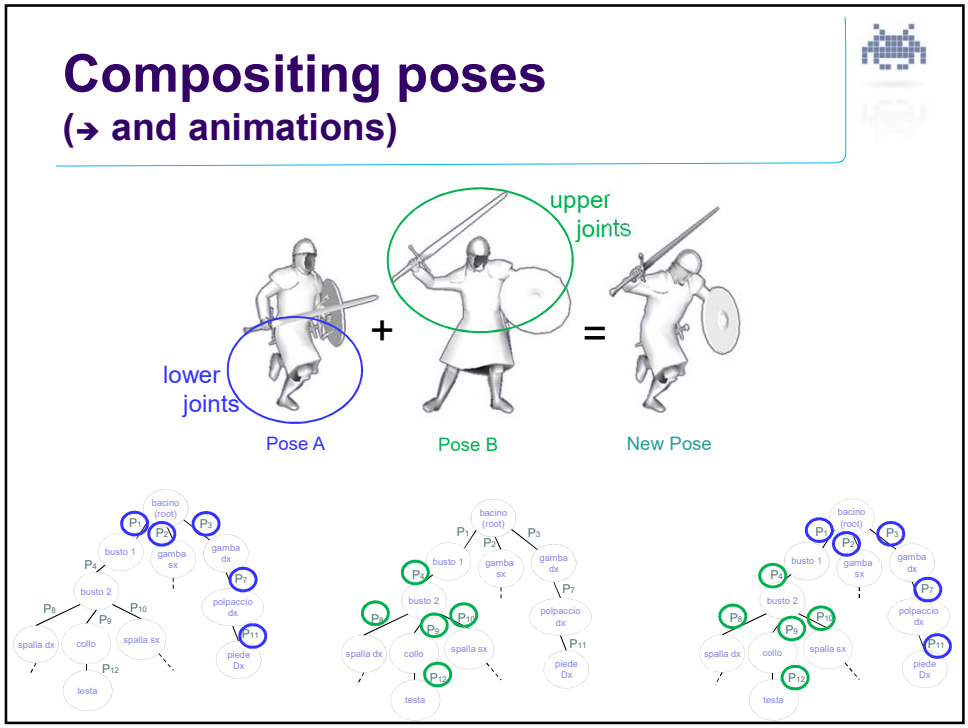
Third person view



106

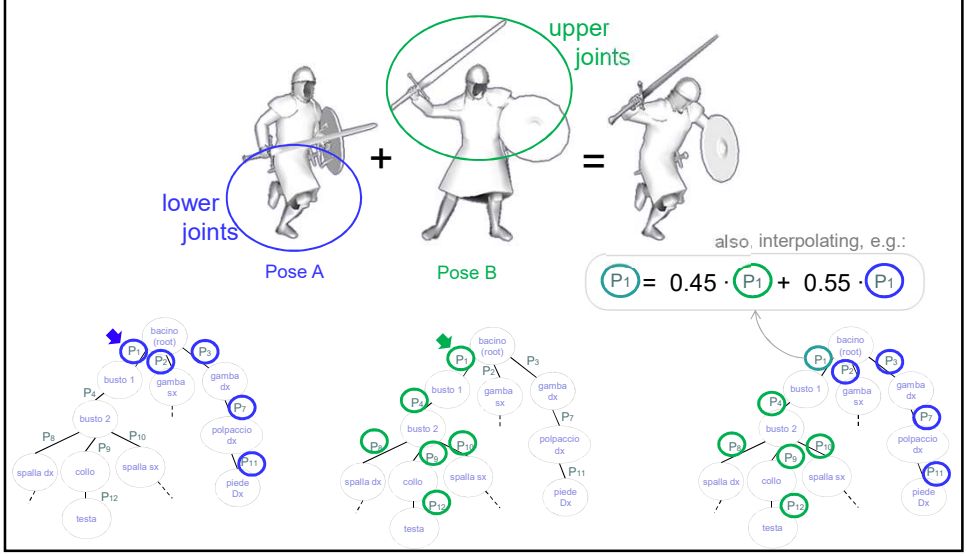

First person view
view space
V
world space
T1
hip (root)
P1
P2
P3
torso 1
leg L
leg R
P4
torso 2
P7
calf R
P8
P10
P9
shoulder R
neck
shoulder L
P11
foot R
P12
head
T2
camera
T2 = «mid-eye» positioning
(in «head» space)

107


Compositing poses
(➔ and animations)
upper joints
lower joints
+
=
Pose A
Pose B
New Pose

108

Compositing poses
(➔ and animations)



109

Compositing poses
(➔ and animations)

- Useful in different contexts:
  - e.g. different character parts following different ani (e.g. lower body: run. Upper body: aims/shoots/reload)
- Note:
  requires updating the final transformations
  - (after changing the local ones)
- Implementation note (Unity):
  - Unity does this with “Layers” in Animation Controller
    - Layer = a mask:
      which bones are driven by this animation?

110

A few (pre)processing tasks for skeletal animations

- Compression
  - input: ani with N keyframes
  - output: ani with M<N keyframes
- Retargeting
  - input: Rig1 + (Skel animat for Rig1) + Rig2
  - output: (Skel animat for Rig 2)
- Building from a blend-shape animation
  - input: Blend-shape
  - output: Rig + Skinned Mesh + Anim
  - note: the opposite is a trivial («baking»)

111

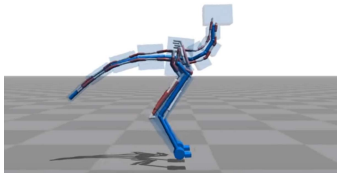
Compression of skeletal animations

- Objective: remove keyframes
  - the “redundant” ones
  - preprocessing task (e.g. as a game tool)
- Basic algorithm concept:
  - for each keyframe P_x
    - 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

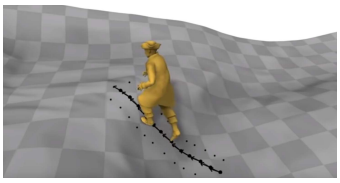
112

Research topic: apply ML to 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

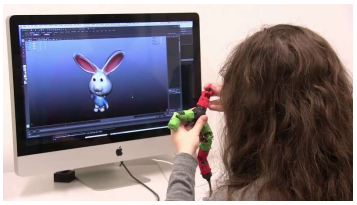


Phase-Functioned Neural Networks for Character Control
Daniel Holden, Taku Komara, Jun Saito
SIGGRAPH 2017

(Among MANY others)

113

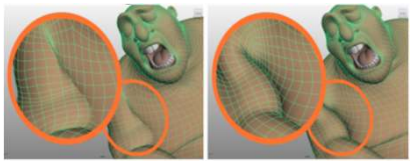
Research topic: better interfaces to author animations



Tangible and Modular Input Device for Character Articulation
Alec Jacobson, Daniele Panozzo, Oliver Glauser, Cedric Pradalier, Otmar Hilliges, Olga Sorkine-Hornung
SIGGRAPH 2014

114

Research topic: Deformation beyond standard skinning



Efficient Elasticity for Character Skinning with Contact and Collisions
Aleka McAdams et al (Disney animation)
SIGGRAPH 11

Note: usually way more complex than direct methods (LBS / DQS).
More offline animation oriented than videogames

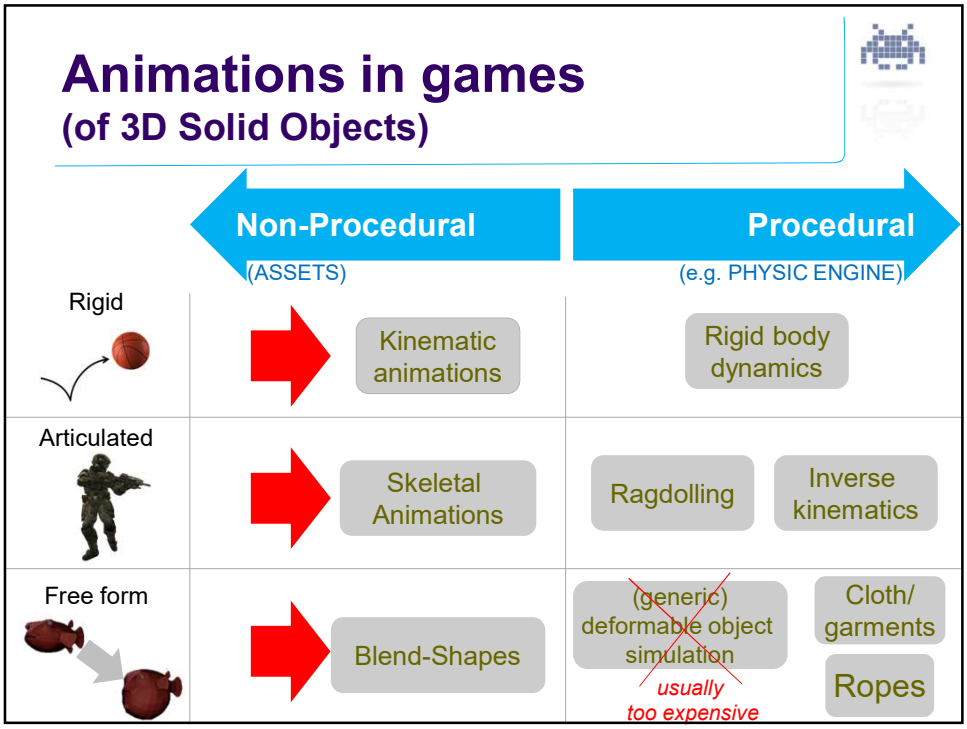
115

Per-vertex animations VS Skeletal-Animations

- Per Vertex animations
  - can interpolate keyframes (but linear trajectories)
  - heavy in RAM
    - replications of normals / positions
  - light to render / compute
- Skeletal animations
  - can interpolate keyframes *better* (curved trajectories)
  - light in RAM
    - animations / models orthogonality
  - minor overheads
    - transform interpolation (x vert!)
    - updates final transoform before (unless can be baked)

116

Animations in games
(of 3D Solid Objects)

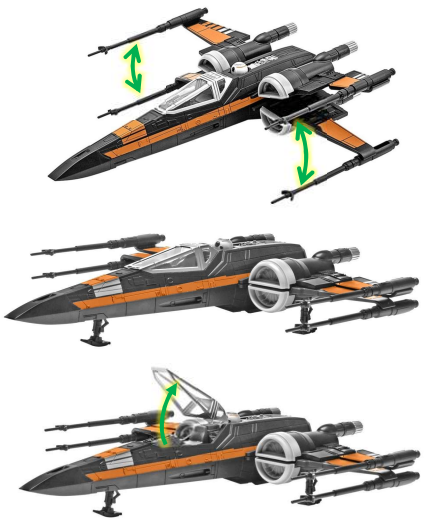


117

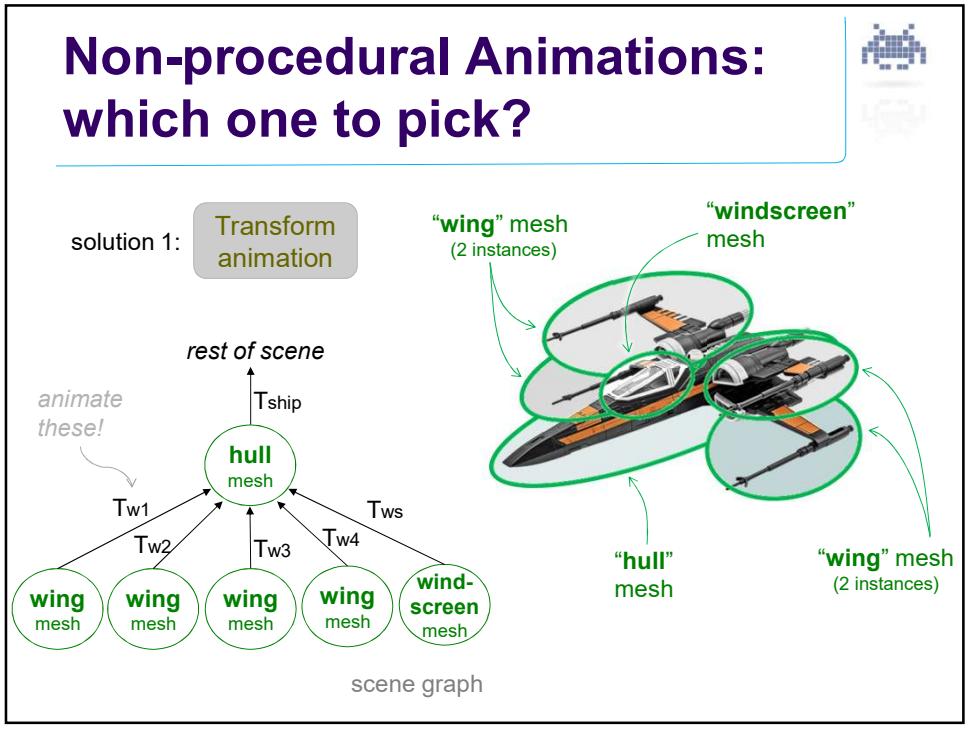
Non-procedural Animations: which one to pick?

- Which format to pick?

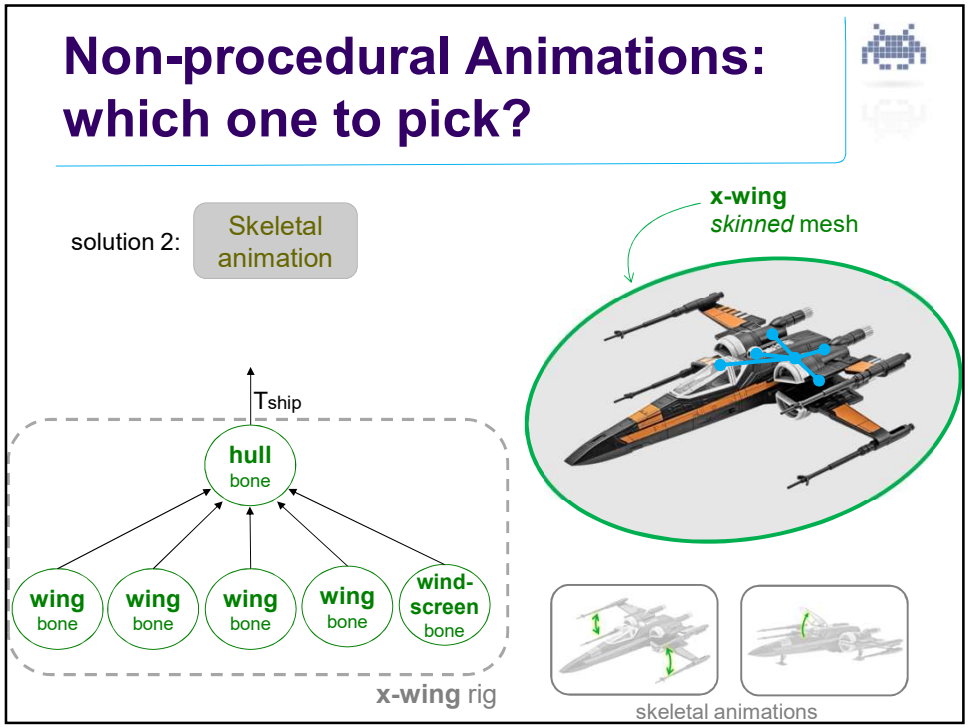
EXAMPLE:
say we want
a model capable of
doing this:



118


Non-procedural Animations: which one to pick?
solution 1:
Transform animation
"wing" mesh
(2 instances)
"windscreen" mesh
rest of scene
animate these!
Tship
hull mesh
Tw1
Tw2
Tw3
Tw4
Tws
wing mesh
wing mesh
wing mesh
wing mesh
wind-screen mesh
"hull" mesh
"wing" mesh
(2 instances)
scene graph

119


Non-procedural Animations: which one to pick?
solution 2:
Skeletal animation
x-wing
skinned mesh
Tship
hull bone
wing bone
wing bone
wing bone
wing bone
wind-screen bone
x-wing rig
skeletal animations

120

Non-procedural Animations: which one to pick?

solution 3: Blend-shape

base shape | morph 1 | morph 2

"x-wing" blend-shape

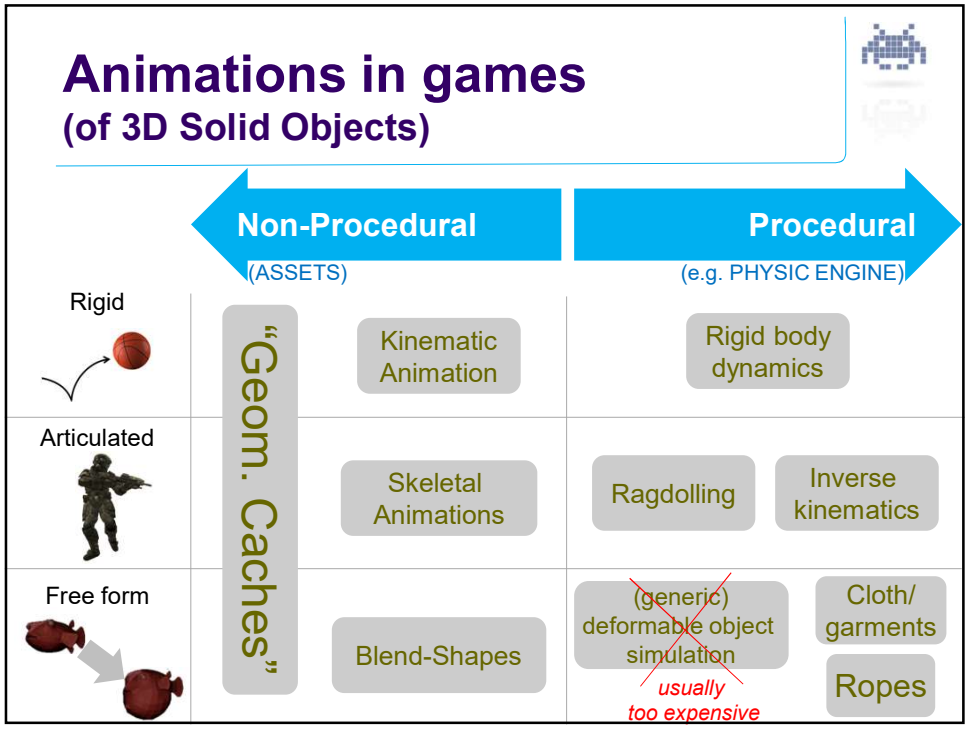
121

Non-procedural Animations: which one to pick?

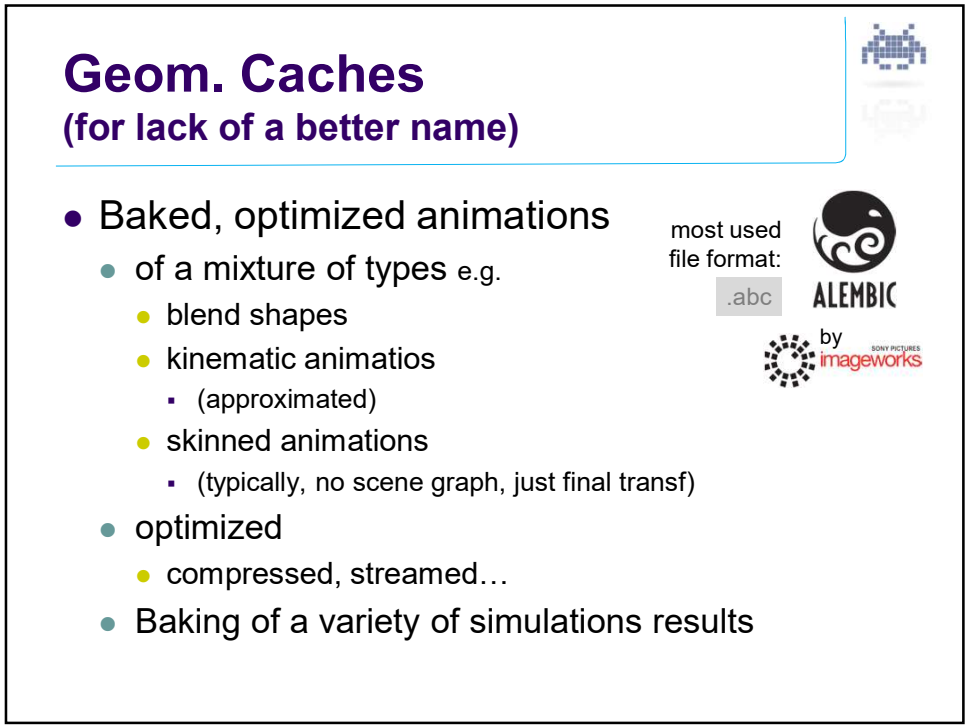
In this example:

- **Animation of transforms (of the scenegraph)**:
  - *how:* 3 (rigid) meshes, 5 instances, animate scenegraph transforms
  - can reuse geometry for all wings: most compact on RAM ☺
  - simpler rendering
  - 5 separated draw calls! ☹
- **Skeletal animation**:
  - *how:* one rig + one skinned mesh + few skeletal animations
    - mesh skinning: single bone enough in this case
  - if very low poly mesh (few polys): a waste?
  - more taxing rendering (a bit) ☹
    - real time skinning on vertex any
  - single draw call! ☺
- **Blend shapes**:
  - *how:* blend shape with one base shape + 2 morphs
  - minimal impact
  - worst quality interpolation: linear → straight (non curved) paths
    - vertices on straight paths (unless, more shapes added)
  - heaviest on RAM ☹
    - (a waste of DoF!)
    - not important, if very low res
  - single draw call! ☺
    - but to different buffers each frame / or to a larger buffer

122


Animations in games
(of 3D Solid Objects)
Non-Procedural
(ASSETS)
Procedural
(e.g. PHYSIC ENGINE)
Rigid
Articulated
Free form
"Geom. Caches"
Kinematic Animation
Rigid body dynamics
Skeletal Animations
Ragdolling
Inverse kinematics
Blend-Shapes
(generic) deformable object simulation
usually too expensive
Cloth/ garments
Ropes

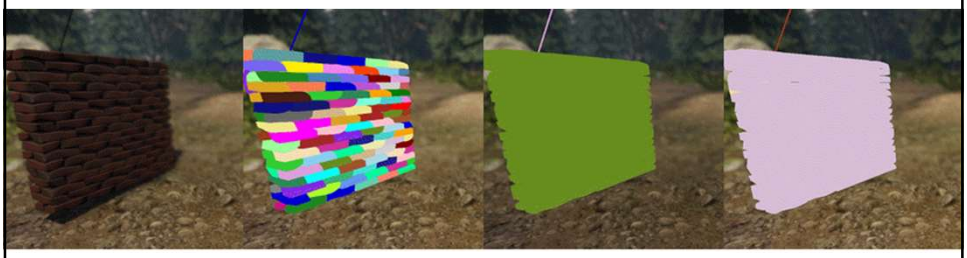
123


Geom. Caches
(for lack of a better name)
Baked, optimized animations
of a mixture of types e.g.
blend shapes
kinematic animatios
(approximated)
skinned animations
(typically, no scene graph, just final transf)
optimized
compressed, streamed…
Baking of a variety of simulations results
most used file format:
.abc
ALEMBIC
by imageworks

124

Geom. Caches
(for lack of a better name)

- Baked, optimized animations
  - of the appropriate types including mixtures



Input:
170 Meshes
88400 Verts

as Pre-made Tansforms:
Meshes: 170
Data rate: 0.13 MB/s
Draw calls: 170
(same ones each frame)

as a Blend Shape:
Meshes: 1, with N shapes
Data rate: 4.3 MB/s
Draw calls: 1
(different one each frame)

as a Skeletal Animation
Meshes: 1, w skinning (*)
Data rate: 0.13 MB/s
Draw calls: 1
(same one each frame)
(*) just 1 bone per vertex

Geometry Caches
(a subset of Alembic)

by CRYENGINE

125

Animations
in Unity (+Mecanim) (notes)

- Assets (models, animation, skeletons) imported as formats:
  - fbx, collada
- Animation compression
  - available during import / builds
  - auto reduction of: num of links per vertex, num of keyframes … :
- «Animator Controller» module → deals with:
  - blending between animations: «transitions»
  - compositing animations: «layers»
    - e.g.: a layer overwrites upper body bones
  - and is nicely WYSIWYG (graph visualization)
- Inverse Kinematic: with scripts (`Avatar.SetIKPoistion`)
- Skeletons:
  - way 1: custom (imported as assets)
  - way 2: built-in standard humanoid skeleton provided
    - (~21 ossa)
    - simplified: rigging (predefined constrains), layers (predef. labelling)

126