Representations for ~~rotations~~ **roto-translations**

- 3×3 Matrices
- Euler Angles
- Angle + Axis
- Quaternions

} + Traslation (displacement vect.)

- 4×4 Matrices (or 3×4)
- Dual Quaternions

2

Representations for roto-rotations (notes)

- So far, we assumed that the rotation and translation component of a transformations are stored separatedly
  - We have seen reasons why this is convenient
- There are a few representations which store rotation and translation (roto-translations, aka “rigid” transformations) jointly:
  - 4x4 matrices (we have seen the problem with them)
  - Dual quaternsions

3

Dual Quaternions: their math in a nutshell

- New "fantasy" assumption: there is a ε such that $\varepsilon \neq 0$, $\varepsilon^2 = 0$ (for the rest, ε behaves like any real)
- A dual quaternion: $p + \varepsilon q$, with $p,q \in \mathbb{H}$ ← Quaternion set
- That is, eight scalars
  - weights for: i, j, k, 1, εi, εj, εk, ε
- A dual quaternion represents:
  - a point / vector in 3D, when p = 0 and Real(q) = 0
  - a roto-translation, when ||p|| =1 and p·q = 0
- To roto-translate a point a, with roto-trans b just conjugate their representations $a \leftarrow b \cdot a \cdot \bar{b}$ (dual-quat conjugate)

4

Course Plan

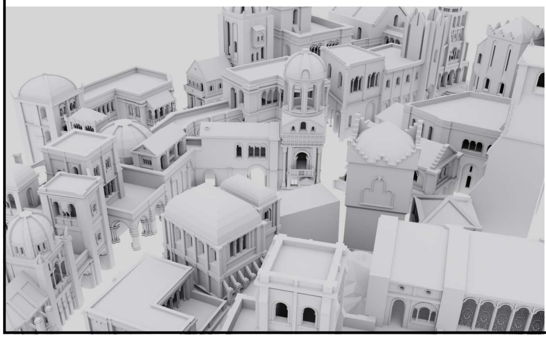
lec. 1: Introduction ●
lec. 2: Mathematics for 3D Games ●●●●●
lec. 3: Scene Graph ● ←
lec. 4: Game 3D Physics ●●● + ●●
lec. 5: Game Particle Systems ◖
lec. 6: Game 3D Models ●◖
lec. 7: Game Textures ●●
lec. 8: Game 3D Animations ●●●
lec. 9: Game 3D Audio ●
lec. 10: Networking for 3D Games ●
lec. 11: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

5

3D video games 2018/2019

the Scene Graph

Marco Tarini



6

Recap:
3D Spatial Trasforms

- Math functions
  - input: point / vector / versor
  - output: point / vector / versor

  Thus, can be applied to e.g. 3D models (apply it to every vertex position and normal...)
- Typically:
  - Scaling + rotation + translation
- They capture:
  - Size / scaling up or down
    - With deformations (anisotropic) or not (isotropic , uniform)
  - Orientation in space / rotation
  - Position / movement (translation)

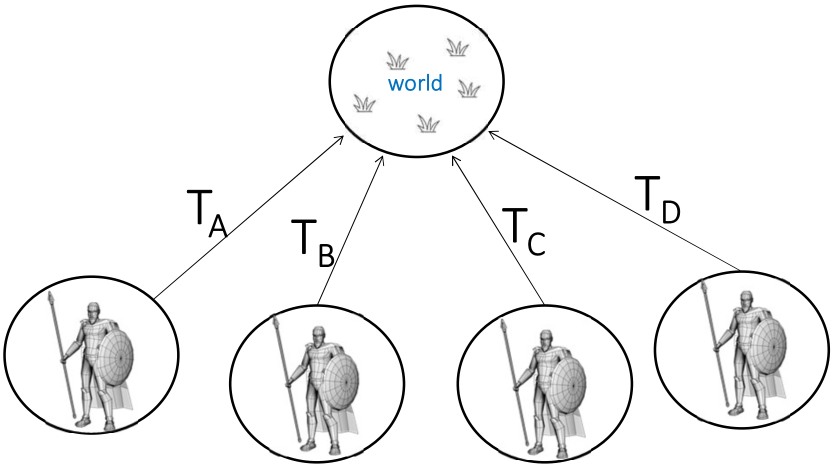
7

Recap: transformation associated to an object in the scene

- Any object associated to a spatial location in the game is given its transformation, which goes
- From:
  - local space *a.k.a.*
  - object space *a.k.a.*
  - pre-transform space
  - *a.k.a.* «*castle*» space / «*hero*» space / «*camera*» space / «*chainsaw*» space / «*bazooka*» space / etc
- To:
  - global space *a.k.a.*
  - world space *a.k.a.*
  - post-transform space

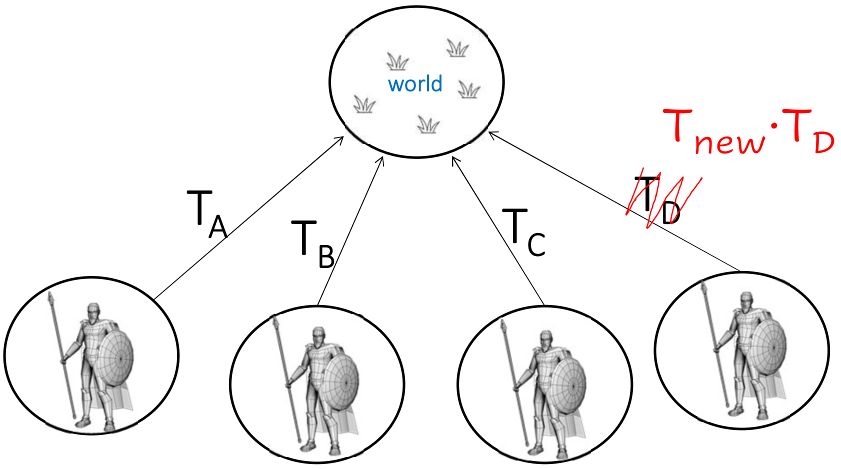
9

Transforms associated to each object in game



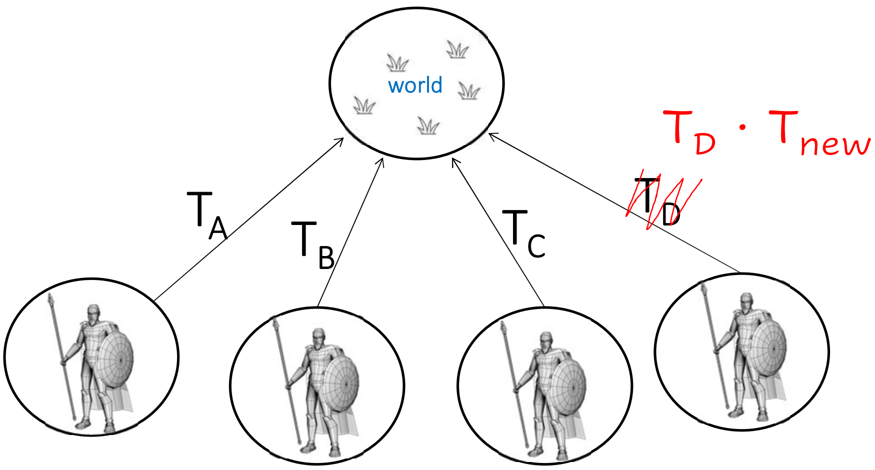
10

Moving Object: updating per-object Transforms



11

Moving Object: updating per-object Transforms



12

Moving Object: two ways of updating per-object Transforms

- Let T_{new} be a new transformation to be applied to move object D (w.r.t. its current placement)
  - Say: rotation = ide scaling = 1 translation = (-2,0,0)
  - T_{new} = "*move two units to the left*" (assuming X = right)
- How to update transformation T_D ? Two ways:
  - $T_D \leftarrow T_D \cdot T_{new}$ = object D moves 2 units on its left
  - $T_D \leftarrow T_{new} \cdot T_D$ = object D moves 2 units on world's left (meaning, i.e., "*Westward*")

Info: Unity calls this applying the new transformation in local space or in global space respectively both in interface, and in scripts (see parameter relativeTo of Transform.Translate)

13

Moving Object: two ways of updating per-object Transforms

- Let T_{new} be a new transformation to be applied to change object D (w.r.t. its current placement)
  - Say: rotation = ide scaling = 2 translation = (0,0,0)
  - T_{new} = "*double by x2*" (volume gets x8 bigger)
- How to update transformation T_D ? Two ways:
  - $T_D \leftarrow T_D \cdot T_{new}$ = object D enlarges from its center
  - $T_D \leftarrow T_{new} \cdot T_D$ = object D enlarges from world's center (i.e. moves away from it too)

14

Moving Object: two ways of updating per-object Transforms

- Let T_{new} be a new transformation to be applied to change object D (w.r.t. its current placement)
  - Say: rotation = *j* scaling = 1 translation = (0,0,0)
  - T_{new} = "*flip by* 180° *around Up axis*" (assuming Y = up)
- How to update transformation T_D ? Two ways:
  - $T_D \leftarrow T_D \cdot T_{new}$ = object D rotates around its up axis (e.g. going supine to prone if laying down)
  - $T_D \leftarrow T_{new} \cdot T_D$ = object D rotates in world's up axis

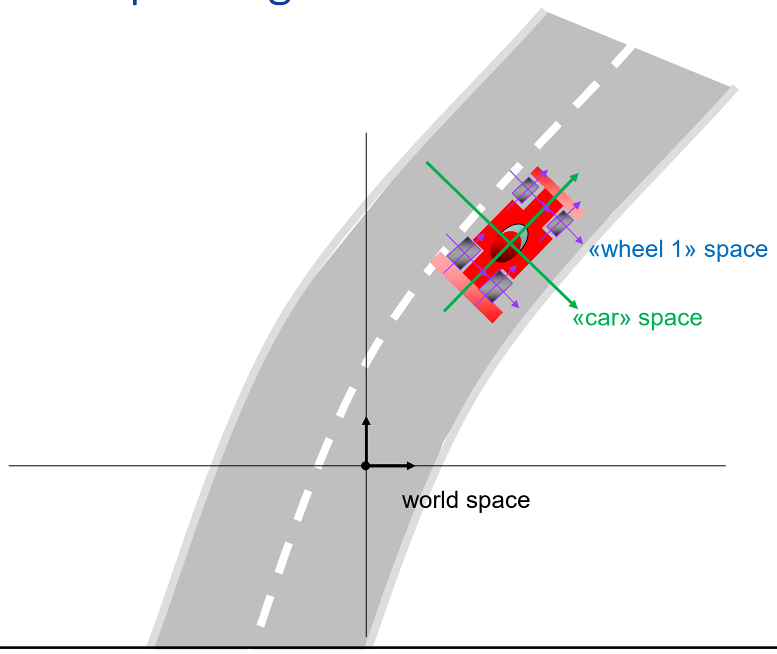
15

Composite scenes: hierarchical transformations

- So far, we assumed that the transform of each object goes from local to global in one step
- In reality, the scene is defined hierarchically
- Objects have sub-objects in them
  - a city is made of houses made of walls made of bricks
  - a «hat» sits on an «head» which sits on a «character» which sits in a «spaceship» moving across the «scene»
- Also: different instances of the same object can appear in multiple locations of the scene
  - E.g. all wheels of all cars are the same "wheel" model

16

Compositing scenes



18

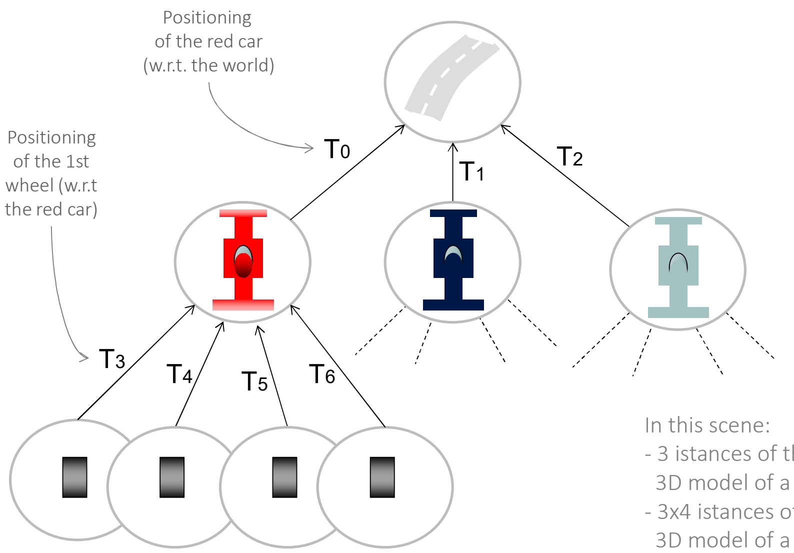
Scene graph

A tree (i.e. hierarchical structrure)

- Each nodes: a space (a reference frame)
  - The Local Space of that node
- To each node we associate:
  - Instances to… stuff:
    anything at all that has a place in the virtual scene:
  - 3D models, lights, cameras, virtual microphones
    spawn points, explosions, etc
- Root node: world space
  - Global Space = local space of the root
- On the arches: we associate the transform
  - the "local" trasform

19

Scene graph



In this scene:
- 3 istances of the same 3D model of a vehicle
- 3x4 istances of a same 3D model of a wheel

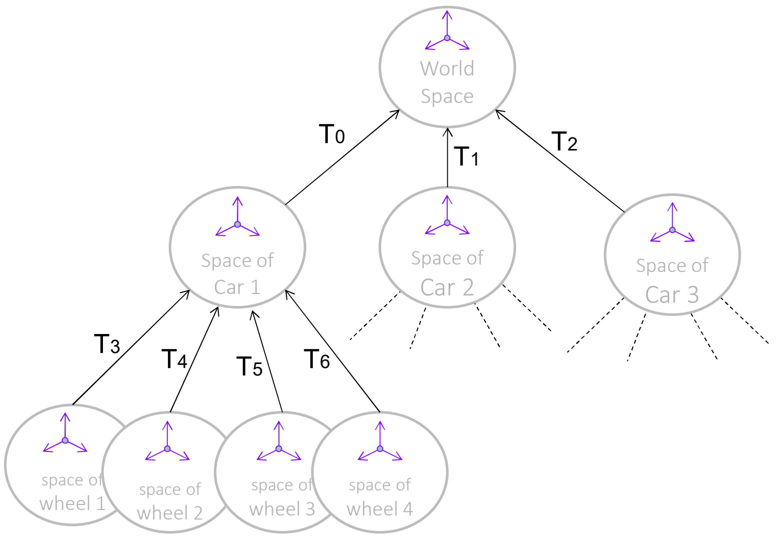
20

Local VS global Transform

- Local transform (a.k.a. «relative» transform)
  - from the local space of a node
    to the local space of its parent space
- Global transform (a.k.a. «absolute» transform)
  - from the local space of a node
    to the world space (the “local” space of the root)
  - obtained by: *cumulating* local transforms to the root!
- benefit: changing the transform associate to a node affects its entire subtree!

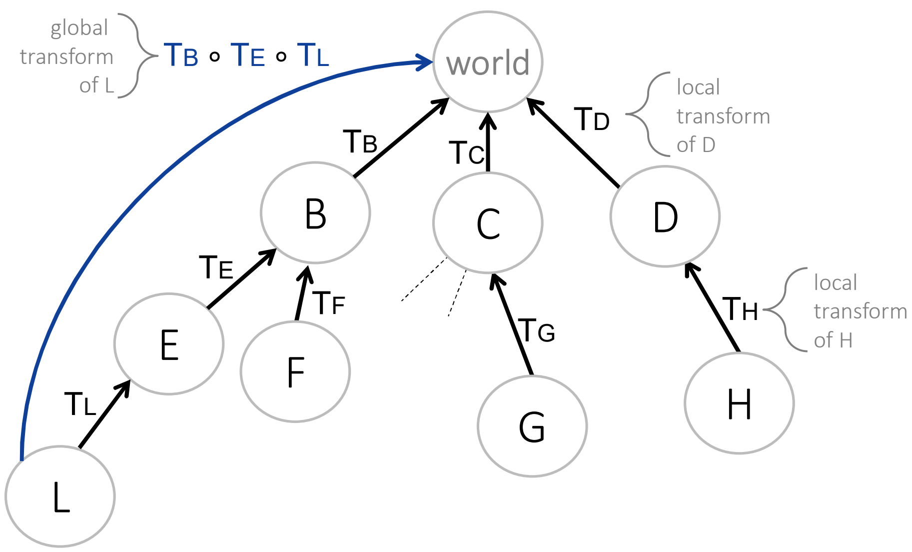
21

Scene graph



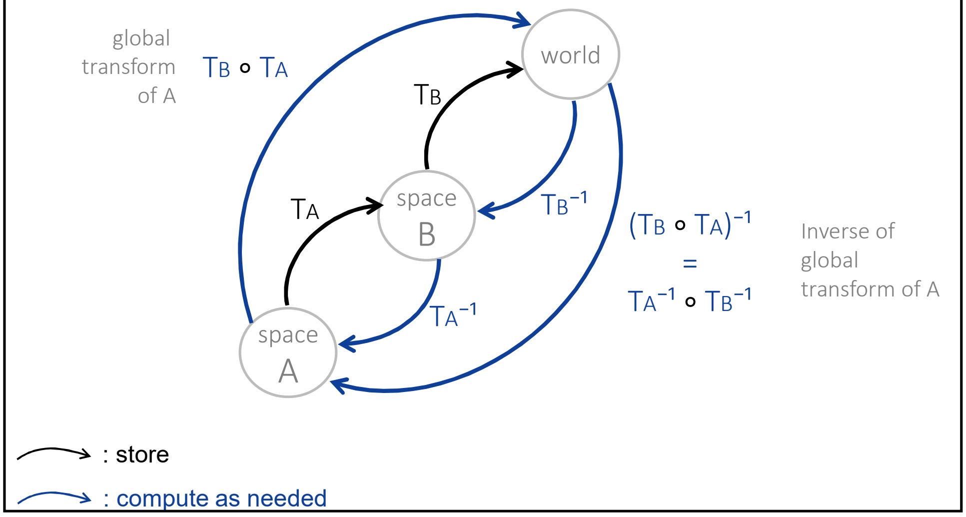
22

Local VS Global Transforms



23

Reminder: inverse of a composite transform (or, in general, function)



24

Reminder: inverse of a composite transform (or, in general, function)

$$(T_B \circ T_A)^{-1} = T_A^{-1} \circ T_B^{-1}$$

- The inverse of "first Ta then Tb" is "the inverse of Tb" followed by "the inverse of Ta"
- As it's natural: if you...
  - "take a step *forward,* then turn by 90° on the *left*"

  ...then, to go back to the starting pos you need to...
  - "turn by 90° on the *right,* then take a step *backward*"

25

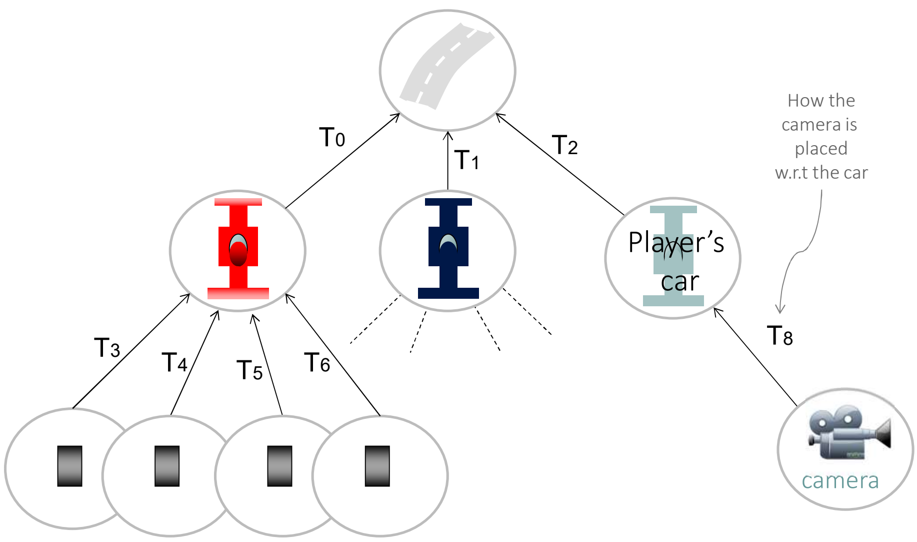
The camera in the scene graph

- Camera:
  - Like any other object in the scene, the camera sits in a node the scene-graph
  - for the scene to be rendered, there must be a camera somewhere in the graph!
  - View Space = Local Space of the camera
  - (Screen Space is a similar, and sometimes equivalent, concept)
- the View Space is crucial for the rendering engine
  - In view space, coordinates describe where things are in front of the camera!
  - For example: $z > 0 \Rightarrow$ in front of the camera, $z<0 \Rightarrow$ behind the camera (don't render)
- Camera animations = move camera
  - by anything that changes its global transformation
  - e.g. a script changing its local transform, or the one of it's parent

26

The camera is in the scene graph

E.g.: to make the camera follow the car…



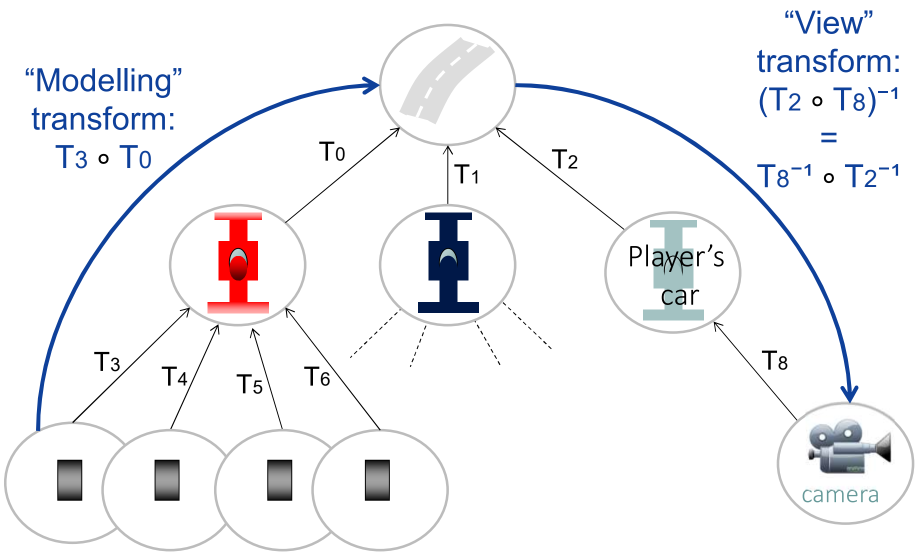
27

Transforms for the Graphics engine (link to Computer Graphics courses)

- The rendering engine uses a few standard transformations, when rendering an object,
- They are named:
  - "Model" matrix: from object space to world space
    - Captures how the scene is **modelled** (by a scener)
  - "View" matrix: from world space to view space
    - Captures how the scene is **viewed** (by the camera)
  - "Model-View" matrix: from object space to view space
  - ("matrix" because trasnforms are usually modelled as 4x4 matrices by Rendering engines & APIs)
- Computing them from the scene graph is easy

28

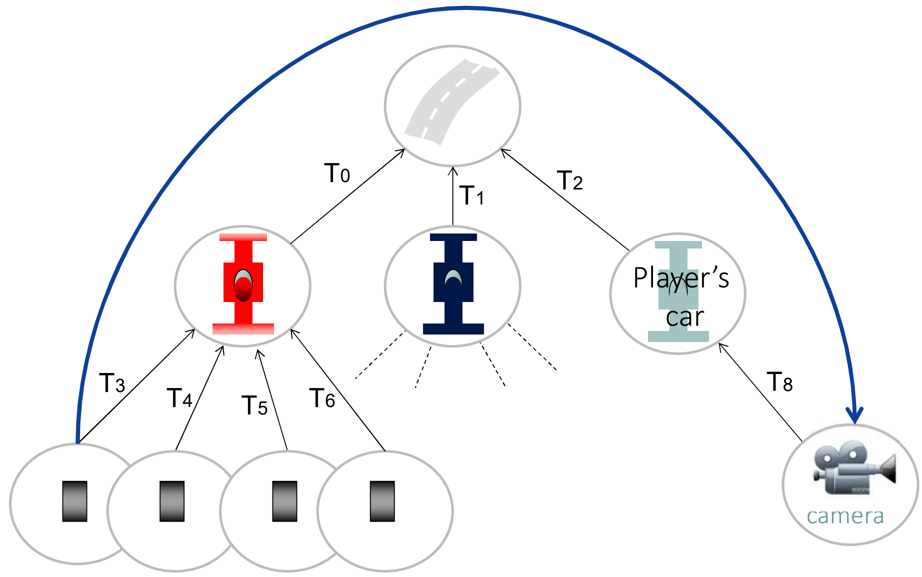
Transforms for the Graphics engine



29

Transforms for the Graphics engine

"Model-View" transform: $T_8^{-1} \circ T_2^{-1} \circ T_0 \circ T_3$



30

Authoring a 3D scene in a game

- E.g. as a part of the Level Design
- Two different parts, by different artists:
  - 3D modellers make «scene props»
    - artists
    - the 3D models to be assembled
    - (including their texutres etc)
  - sceners compose the scene
    - they assemble the props into a Scene Graph

= asset

31

Authoring a 3D scene

- Examples of other assets associated to a scene
  - a Collision Mesh (a Geometry Proxy)
    - one for each "solid" scene-prop
    - can this be made automatic? Possible, not easy
    - assigned to nodes (for dynamic objects), or (for static objects) possibly all merged into one
    - needed for: physics, visibility computation, AI, plus all sorts of gameplay reasons…
  - a Navigation Mesh (aka AI mesh)
    - usually, one for the entire scene (stored in the root node)
    - needed by: AI (routing – see later)
    - can this be made automatic? Possible, not tribial

32

Authoring a 3D scene

- Examples of other assets associated to a scene:
  - Scripts
    - by the level designer
  - Sky box
  - Outer terrain mesh…
  - Ambient sounds
  - Other data such as spawn points, and more

33

Scene Graph as a data structure

- Each engine / library adopts its own solution
- No standards
  - but file formats exists which can include a scene graph: e.g. COLLADA

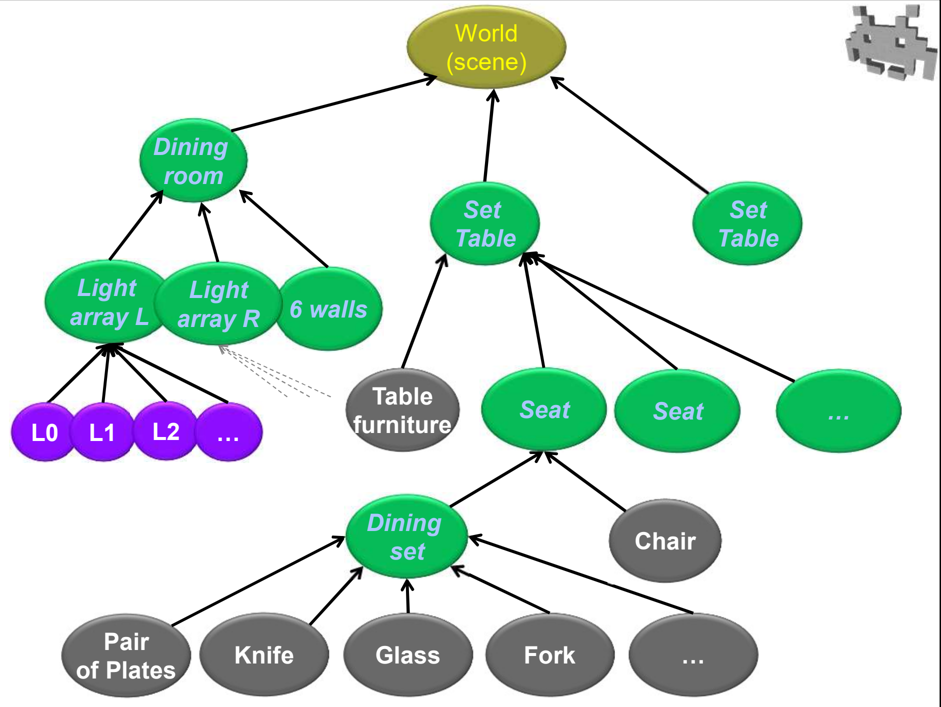
Typical concepts:

- each Node class stores
  - the local transform
  - link to parent
  - maybe, and/or to children, sibilings…)
  - links to instances / assets
- global transforms / inverse are computed on demand
- some mechanism is used for repeated sub-trees

34

Example: a dining table

35



36

Example: a dining table

37

Nodes of a scene-graph in unity
GameObjects & Transforms

A node = a GameObject with

- a transform field, containing
  - its local transform
  - links to Parent, Children (and siblings) – which are transforms
- any number of associated "components", which represent anything residing in that node, like
  - Meshes (to display at this nodes)
  - Cameras: active one(s) produces the rendering(s)
  - "RigidBodies": objects controlled by the physics
  - "Colliders": geom proxies used for collisions
  - "Particle systems" : (i.e. the "emitters" of particles)
  - Sound producers / receivers
  - Scripts …
  - basically any asset!

38

Nodes of a scene-graph in unity
GameObjects & Transforms

- The Transformation actually stores the local transf:
  - `localPosition`, `localRotation`, `localScale`
  - goes from a node to its parent
- the Global transformation can be accessed via the properties: ← feels like assigning / reading a field, actually means invoking setters/getters (C# trick)
  - `position`, `rotation`, `scale` ("global" is left implicit)
  - what does getting / setting them really do? exercise!
  - this it doesn't always work for "scale"! why? (A: it's because anisotropy)

40

Digression on unity : properties and components

- Properties (C# mechanism)
  it feels like a field (you can read or assign it)
  but it's actually a getter and setter method
  - `obj.xx = 3` *...means...* `obj.set_xx( 3 )`
  - `foo = obj.xx` *...means...* `foo = obj.get_xx()`
- Components (Unity library mechanism)
  - A generic something attached to a GameObject ← base class for everything
  - `GameObject g;`
    `g.getComponent< type >()`
    returns component of required type
    (if it exists)

41

Nodes of a scene-graph in UNREAL ENGINE UScenComponent

A node within a graph with:

- link to parent / children:
  - getParentComponents
  - getChildComponent(index)
- associated stuff to it:
  UPrimitiveComponent (subclass)
  - for models, physical bodies, etc
- Local Transform: (fields)
  - RelativeLocation , RelativeRotation, RelativeScale
- Global Transform: (methods)
  - GetComponentTransform() /* return transformation */

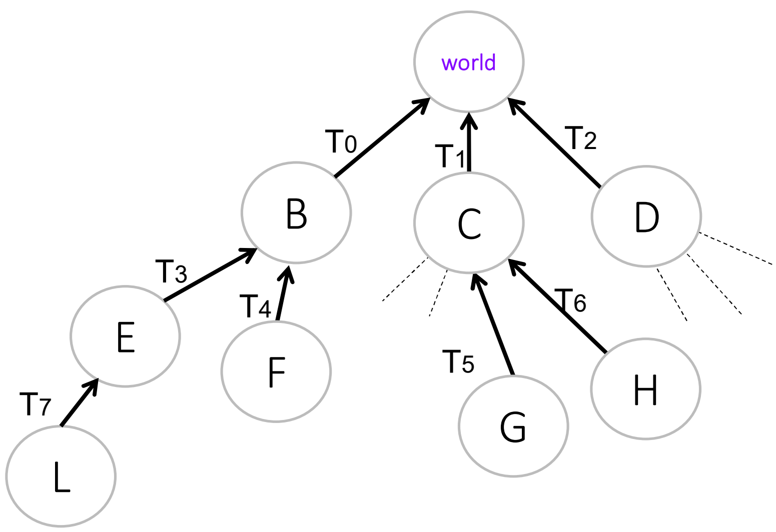
42

Mechanisms for shared subtrees

- In Unity: see "Prefabs"
- In Unreal: see "BluePrints"

43

Drawing for the exercises



44

Exercises 1/2

What is the new transform T'_7 which should subtitute T_7 if...

- ...node **L** il reattached as a child of **D**, leaving its position in *world space* unaffected (e.g. by a scener, or a script)
- ...node **D** is attached under node **L**, without affecting its world space position.
- ...the object in node **L** must be moved 1 unit on the right in view space (camera is in node **C**)
- ...the object in node **L** must be moved by 1 unit ON ITS RIGHT
  ...the object in node L must be displaced by a new transform T applied in post-transform space.

Note: these kinds of problems are silently solved by Unity all the times (in the scripts & when user manipulates the the GUI)

46

Exercises 2/2

- Report the *global transform* of node **L**
- I place a camera in node H:
  report the View Transform for this scene
- What does it mean to apply a translation (0,2,0) to L ...
  1. in L Space (the local space of L)?
  2. in World space?
  3. in View Space?
- Say T7 is the identity, and the camera is in H:
  how to modify T7 to get the case 1,2 or 3?
- Find the origin of space E in space H, and viceversa
- A microfone is in (the origin of) node E, and a speaker is in (the origin of) node H. Find the distance from the mic to the speaker

47