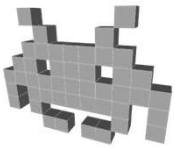
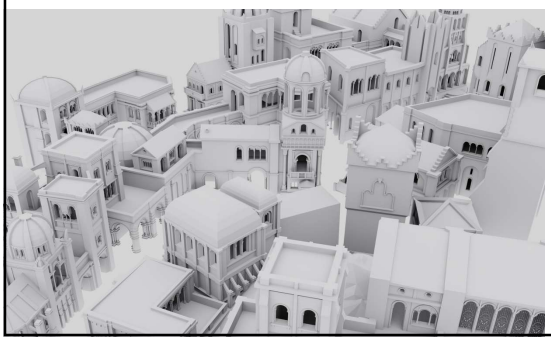
3D video games 2021/2022

the Scene Graph

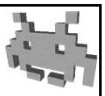


Marco Tarini



2

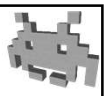
Course Plan



lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** + **Particle Systems**
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**

3

Recap: 3D Spatial Trasforms



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

Thus, can be applied to any 3D thing (apply them to all positions directions etc …)

- Three components:
  - Scaling
  - Rotation
  - Translation

… modelling the State / Act of:

- Size / Scale up (if > 1), down (if <1)
- Orientation / Rotate
- Position / Displace

can be "uniform" ("isotropic")
or not ("anisotropic", different factors in X,Y,Z)

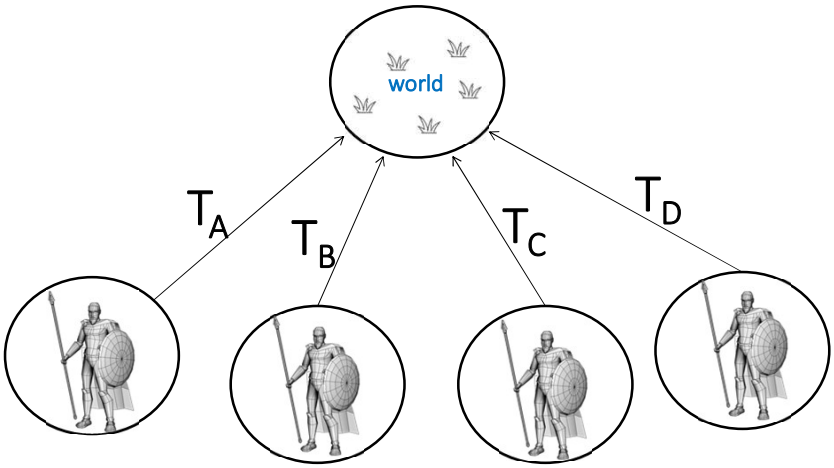
6

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

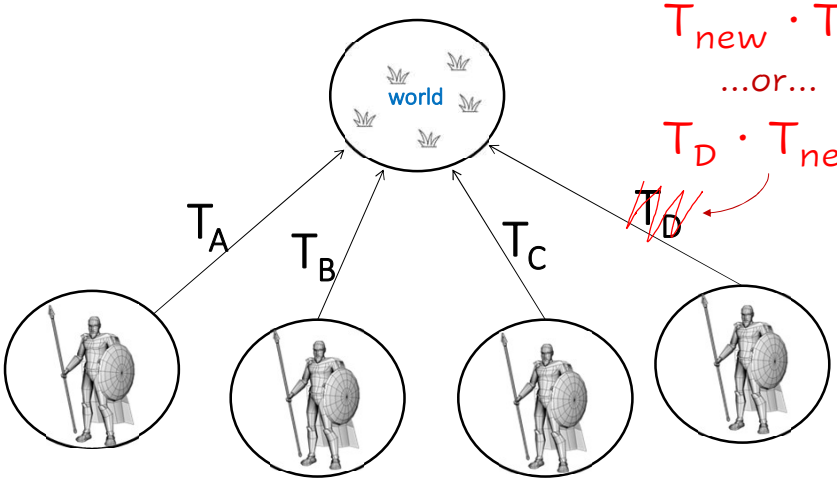
8

Transforms associated to each object in game



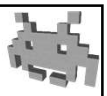
9

Moving objects around = updating their transforms



10

Moving objects: 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., "*West-ward*")

We call this "applying the new transformation in local space" or "in global space respectively"
E.g., in unity: see parameter "relativeTo" of method Transform.Translate

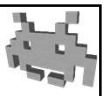
11

Moving objects: 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* ×2" (note: volume gets ×8 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)

12

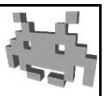
Moving object: two ways of updating per-object Transforms



- 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. goes supine to prone if was laying down)
  - $T_D \leftarrow T_{new} \cdot T_D$ = object D rotates in world's up axis

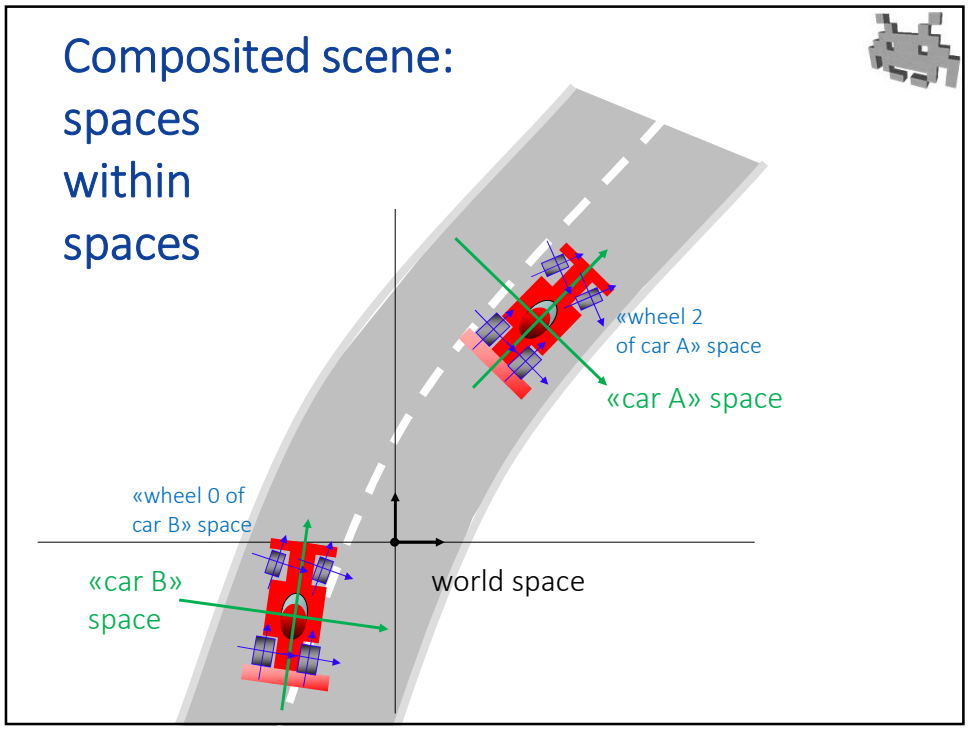
13

Composite scenes: hierarchical transformations

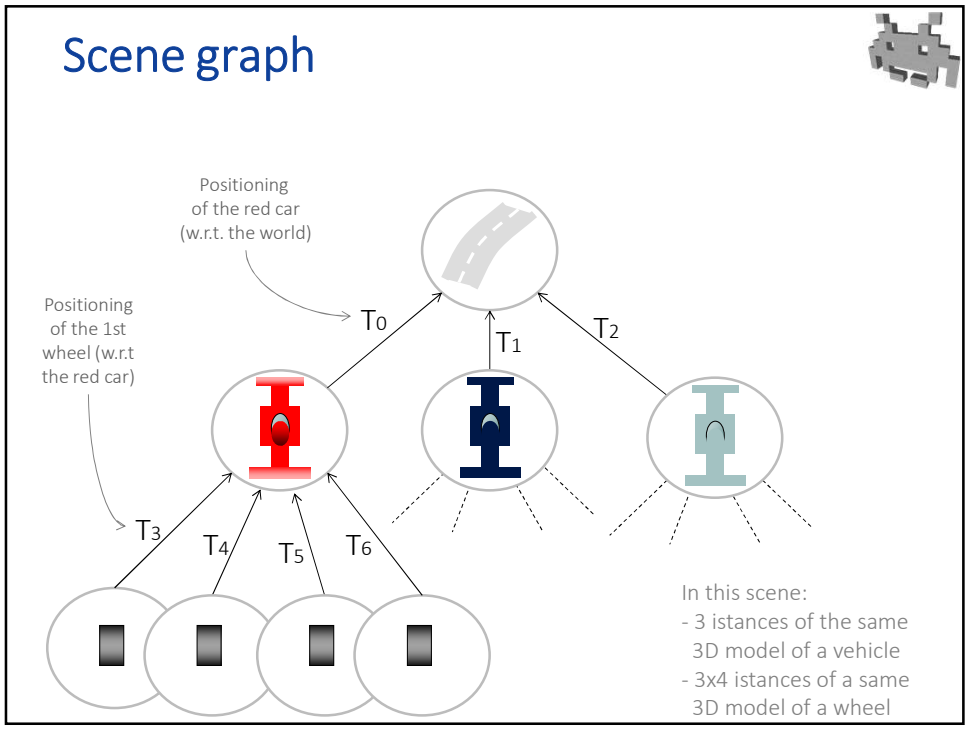


- So far, we assumed that the transform of each object goes from local to global in one step
- In reality, scenes can be defined hierarchically
- That is, objects have sub-objects in them
  - a «city» is made of «houses» made of «walls» made of «bricks»
  - a «hat» sits on an «head» which is part of a «character» who sits in a «spaceship» moving across the «galaxy»
  - a car is a «hull» plus four «wheels»

14

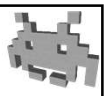

Composited scene:
spaces
within
spaces
«wheel 2
of car A» space
«car A» space
«wheel 0 of
car B» space
«car B»
space
world space

16


Scene graph
Positioning
of the red car
(w.r.t. the world)
Positioning
of the 1st
wheel (w.r.t
the red car)
T0
T1
T2
T3
T4
T5
T6
In this scene:
- 3 istances of the same
3D model of a vehicle
- 3x4 istances of a same
3D model of a wheel

17

Composite scenes: multi-instancing



- Each node contains a reference / pointer / index to one 3D object (e.g. a 3D mesh, etc)model
  - E.g. all wheels of all cars are the same “wheel” model
- 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
  - Advantage:
    only one 3D model in RAM,
    but many identical 3D models on the screen
  - Each model is associated to a different transform, plus other data, e.g. different “materials”

18

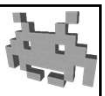
Scene graph

A tree (i.e. a hierarchical structure)

- Each nodes has its own 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
- To each arch: we associate the “local” transform
  - the transform going from the local space of the child node to the local space of the parent node

19

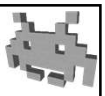
Local VS Global Transform[-ation]s

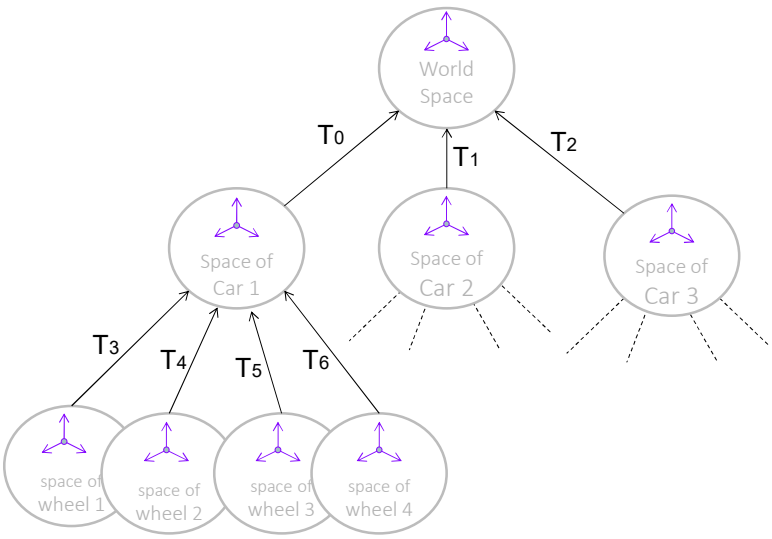


- Local transform (a.k.a. «relative» transform)
  - from: the local space of a node
    to: the local space of its parent
  - Stored per object!
- Global transform (a.k.a. «absolute» transform)
  - from the local space of a node
    to the world space
    (which the "local" space of the root)
  - Procedurally obtained/defined by:
    cumulating all local transforms from node to root
- benefit: changing the transform associated to a node affects its entire subtree

20

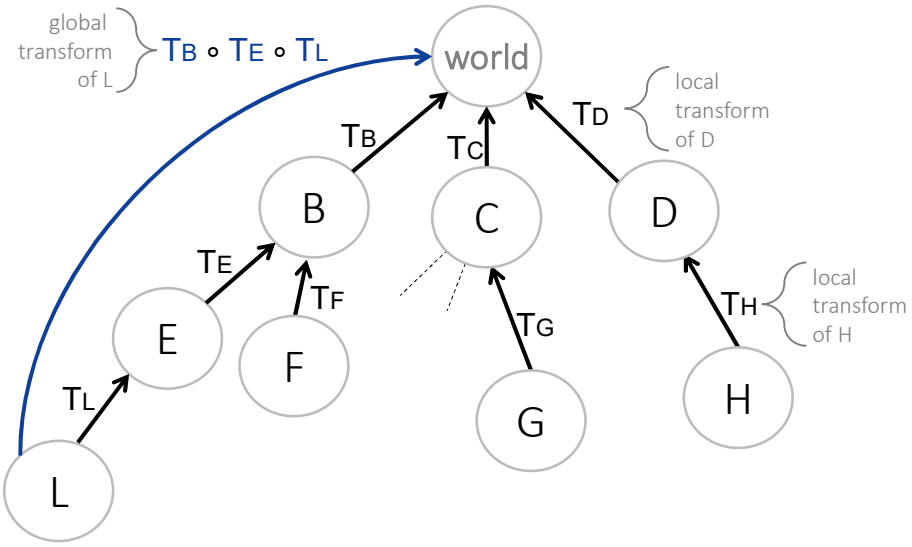
Scene graph





21

Local VS Global Transforms



22

Changing the hierarchy… *without* changing the position

- Event:
  - In the above example, node **L** is detached from its parent (**E**), and becomes a child of the node world
  - (this means that, from now on, it's positioning won't be affected by movement of **E** or of **B** : **L** it is no longer a part of the same compound object, it detached)
  - When that happens, we don't want (the content of the) node **L** to change its world positioning (pos, orientation…).
  - That is, Global transformation of **L** must stay constant
- Question:
  - How to achieve this result by reassign its associated local transform T_L (which is the only thing we store for **L**)?

23