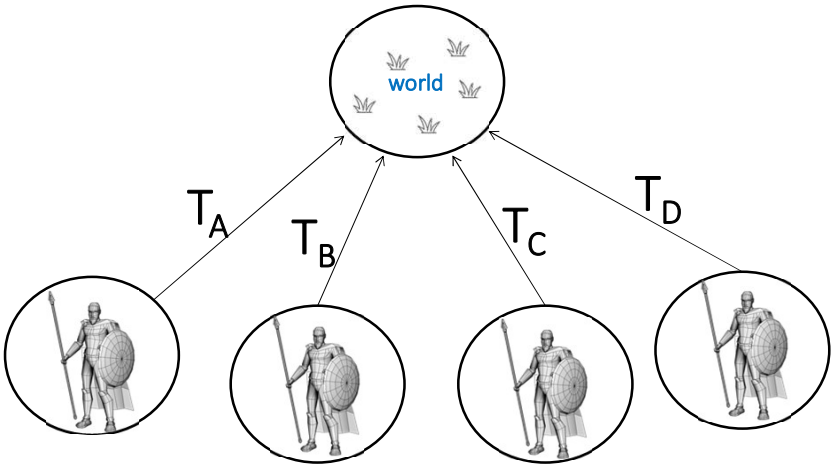
Course Plan

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 Materials
lec. 8: Game Textures
lec. 9: Game 3D Animations
lec. 10: Audio for 3D Games
lec. 11: Networking for 3D Games
lec. 12: Interactive Agents for 3D Games
lec. 13: Rendering Techniques for 3D Games

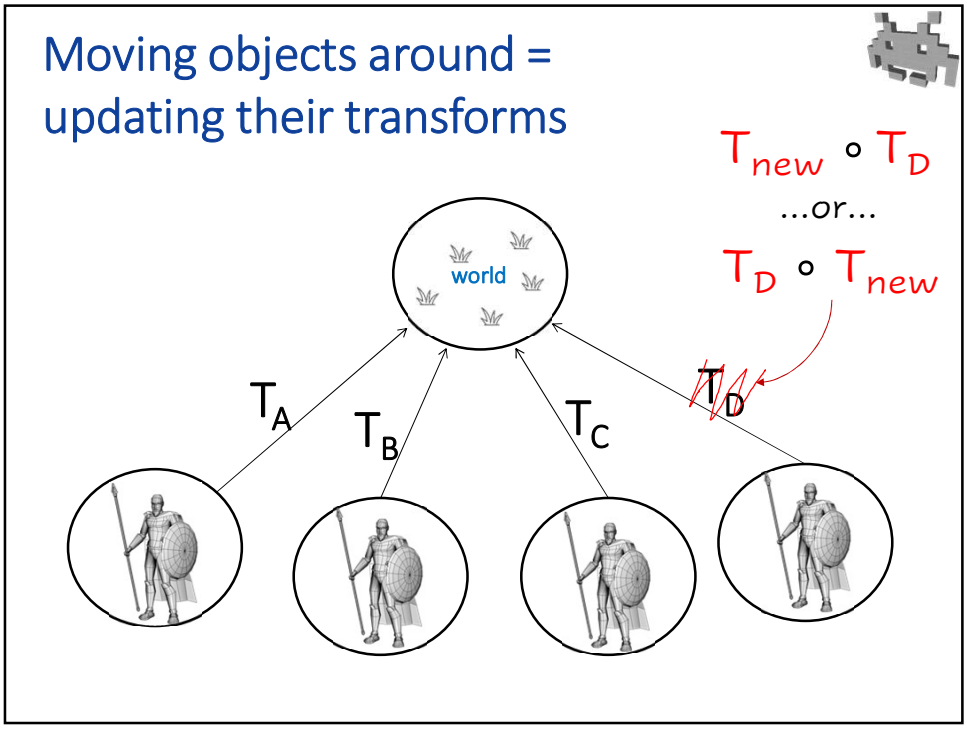
21

Transforms associated to each object in game



22

Moving objects around = updating their transforms



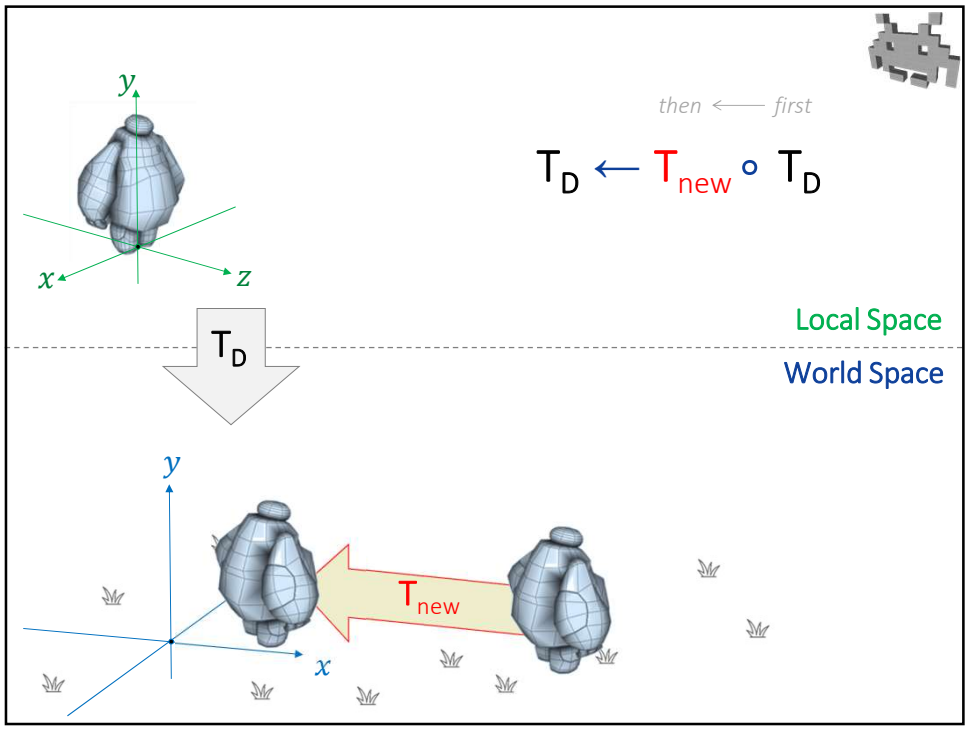
23

Moving objects: two ways of updating per-object Transforms

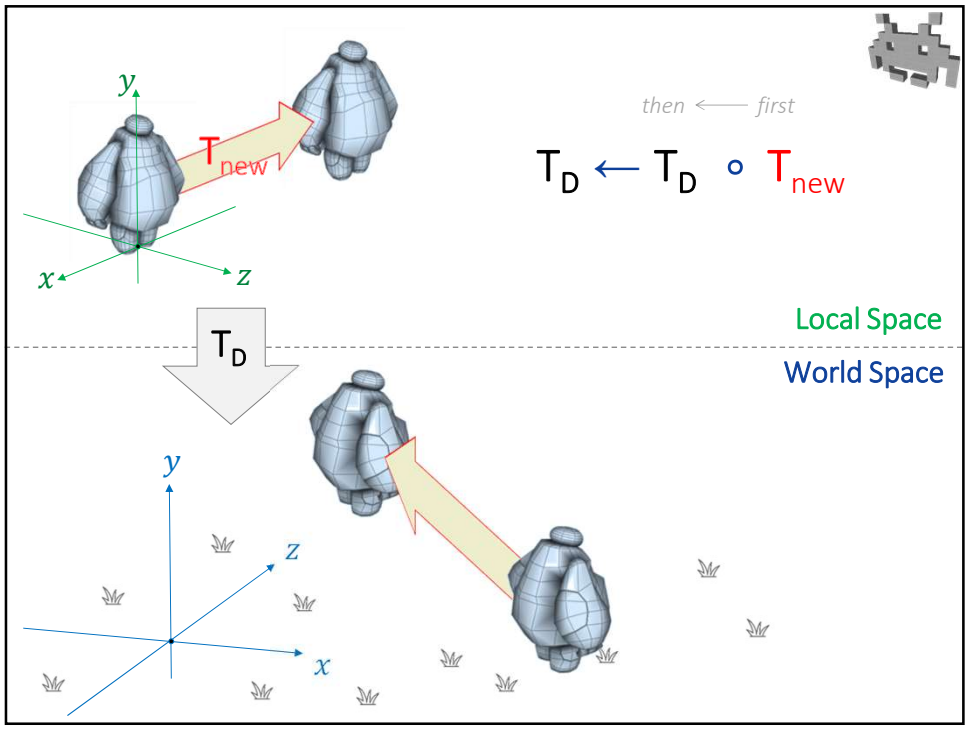
- Let T_{new} be a new transformation to be applied to object D (w.r.t. its current placement)
  - Say: **rotation** = ide **scaling** = 1 **translation** = (-2,0,0)
  - T_{new} = "*move two units to the left*" (assuming X = right)
- How to update transformation T_D ? Two ways:
  - $T_D \leftarrow T_D \circ T_{new}$ = object D moves 2 units on its left
  - $T_D \leftarrow T_{new} \circ T_D$ = 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

24


y
x
z
then ⟵ first
$T_D \leftarrow T_{new} \circ T_D$
Local Space
T_D
World Space
y
T_{new}
x

25


y
T_{new}
x
z
then ⟵ first
$T_D \leftarrow T_D \circ T_{new}$
Local Space
T_D
World Space
y
z
x

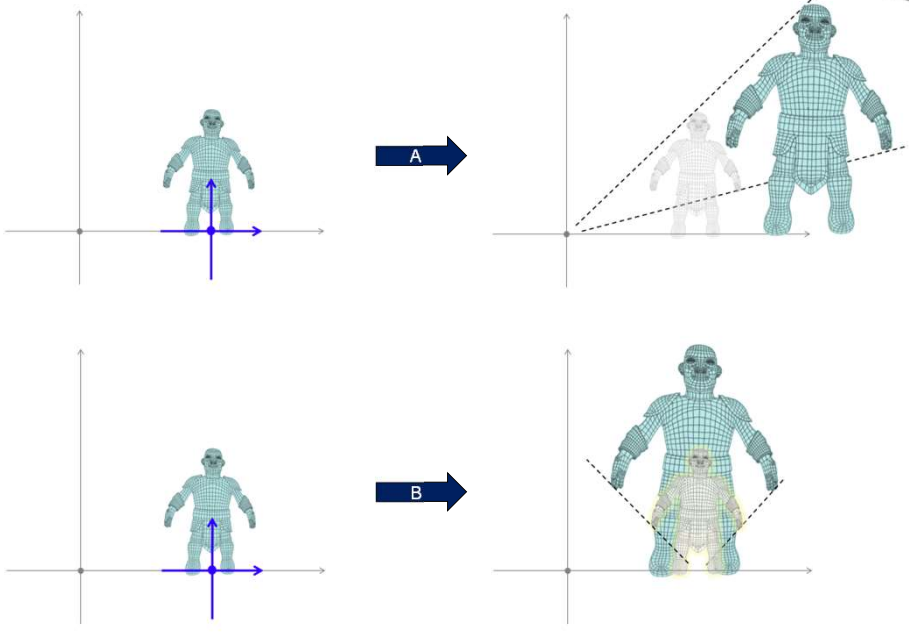
26

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} = "*scale it up by* ×2" (note: volume gets ×8 bigger)
- How to update transformation T_D ? Two ways:
  - $T_D \leftarrow T_D \circ T_{new}$ = object D enlarges from its center
  - $T_D \leftarrow T_{new} \circ T_D$ = object D enlarges from world's center (i.e. moves away from it)

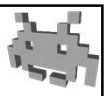
27

Compare:



32

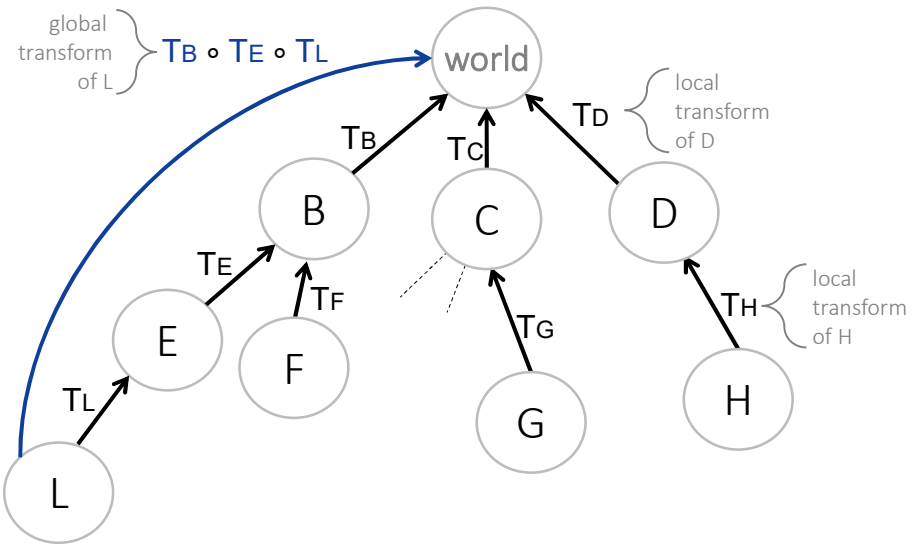
Moving objects: two ways of updating per-object Transforms



- Let T_{new} be a new transformation to be applied to change object D (w.r.t. its current placement)
  - Say: **rotation** = *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:
  - $T_D \leftarrow T_D \circ T_{new}$ = object D rotates around its up axis (e.g., goes supine-to-prone, if was lying down, e.g. on a bed)
  - $T_D \leftarrow T_{new} \circ T_D$ = object D rotates in world's up axis

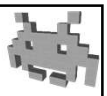
33

The same, in a scene graph…



44

Changing a node positioning…
in local space (refer the schema in prev slide)



- Say T is the transform consisting of moving an object 2 units on the X
  - T = { Scale = 1, Rotation = ide, Translation = (0,0,2) }

transform expressing an **action** on L

- Task:
  - we want node L to undergo transform T in local space.
  - Meaning: we want L to be moved 2 units (its *own* units) in the direction of its right (assuming Unity axis conventions)
  - How do we do it?

$$T_L \leftarrow T_L \circ T$$

(make sure you understand why!)

45

Changing a node positioning…
in global space (refer the schema in the prev page)

- Say T is the transform consisting of moving an object 2 units on the X
  - T = { Scale = 1, Rotation = ide, Translation = (0,0,2) }

transform expressing an **action** on L

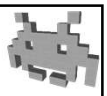
- Task:
  - We want node L to undergo transform T in global space.
  - Meaning: we want L to be moved 2 units (*world* units) in the East direction (if that's how global ref. frame works)
  - Note: we can only change its local transformation (because we only want to affect node L)

$$T_L \leftarrow T'_L$$

  - How to the new value T'_L ?

46

Changing a node positioning… *in global space* - solution



- *Global transform* of L before the change:
$$T_B \circ T_E \circ T_L$$
- The *Global transform* of L which we want (after the change):
$$T \circ T_B \circ T_E \circ T_L$$
- The *Global transform* of L which we have (after the chamge):
$$T_B \circ T_E \circ T'_L$$
- Matching them:
$$T \circ T_B \circ T_E \circ T_L = T_B \circ T_E \circ T'_L$$
- Doing the math…
$$T'_L = \underbrace{T_E^{-1} \circ T_B^{-1}}_{\text{therefore, this is the transformation applied to the local transform of node L to make T happen in global space to node L}} \circ T \circ T_B \circ T_E \circ T_L$$

47

Changing a node positioning… *in global space* - solution

$$T'_L = \underbrace{T_E^{-1}}_{\text{Inverse of the global transform of E (the parent of L)}} \circ \underbrace{T_B^{-1} \circ T \circ T_B \circ T_E}_{\text{the global transform of E (the parent of L)}} \circ T_L$$

48

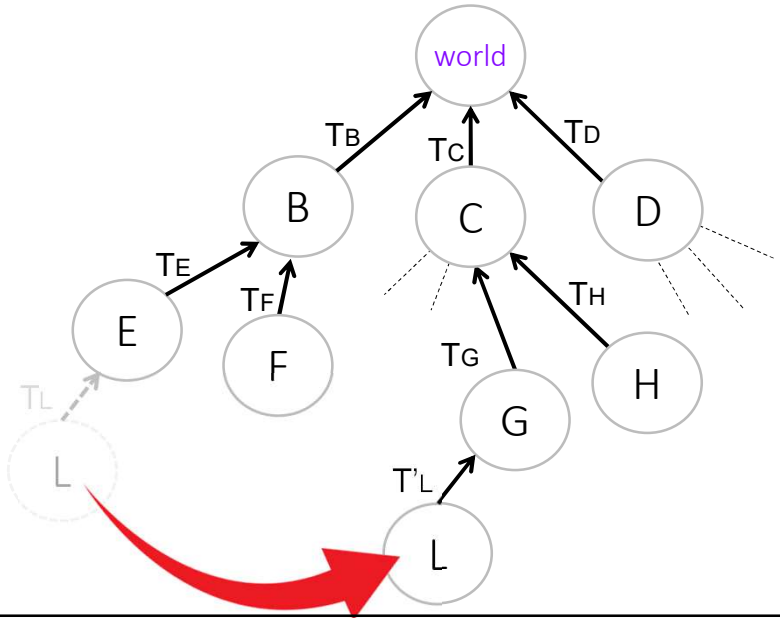
Assigning a new positioning…
in global space (refer the prev schema in slide 44)

- Say T is a transform describing a new global positioning we want for object L in world space
  - T = { scale: (global) sizing of L,
    rotation: (global) orientation of L,
    translation: (global) position of L }

  transform expressing the **state** of L
- How to replace its local transformation T_L , so that its global transformation becomes T?

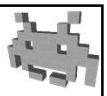
50

Changing the hierarchy



52

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 **G**
  - (this means that, from now on, its 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 was detached)
  - In that moment, we don't want (the content of the) node **L** to change its world positioning (position, orientation…).
  - That is, the 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**)?

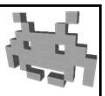
53

Changing the hierarchy… *without* changing the position

- The local tranform T_L stored for L is substituted by some new local transformation T'_L :
  $$T_L \leftarrow T'_L$$
  the problem is then to find this T'_L
- *Global transform* of node **L** *before* the change:
  $$T_B \circ T_E \circ T_L$$
- *Global transform* of node **L** *after* the change:
  $$T_C \circ T_G \circ T'_L$$
- They must be the same, so (doing the math!)…
  $$T'_L = T_G^{-1} \circ T_C^{-1} \circ T_B \circ T_E \circ T_L$$

54

Changing the hierarchy… *without* changing the position



- The math:

$$T_B \circ T_E \circ T_L = T_C \circ T_G \circ T'_L$$

composite both sides with T_C^{-1} on the left…

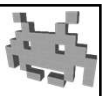
$$T_C^{-1} \circ T_B \circ T_E \circ T_L = \cancel{T_C^{-1}} \circ \cancel{T_C} \circ T_G \circ T'_L$$

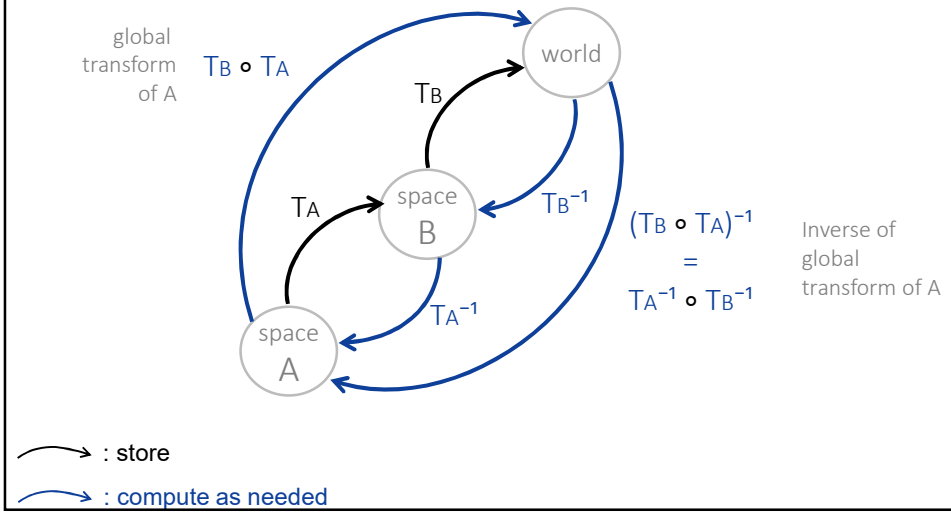
composite both sides with T_G^{-1} on the left…

$$T_G^{-1} \circ T_C^{-1} \circ T_B \circ T_E \circ T_L = \cancel{T_G^{-1}} \circ \cancel{T_G} \circ T'_L$$

55

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





56

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° *clockwise*"

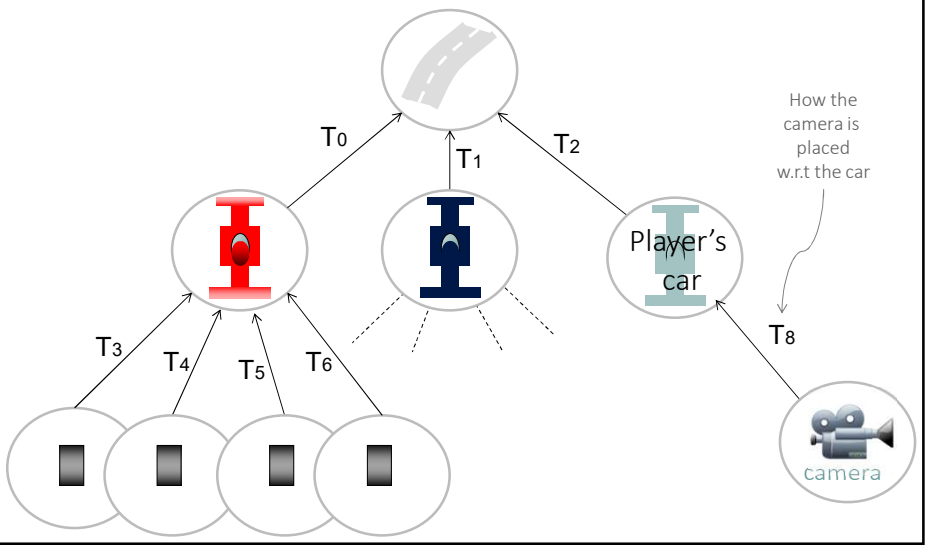
…then, to go back to the starting pos, you need to…

  - "turn by 90° *counter-clockwise*, *then*, take a step *backward*"

57

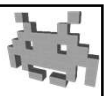
The camera is in the scene graph

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



59

The camera in the scene



- The camera node is particularly important for the rendering (of course)
- The inverse of its associated global transform goes from Camera space (or View Space)…
  - (where the camera is in the origin, looks toward Z (or minus Z in some systems) etc.)
  - (its a space where the rendering is convenient to do)

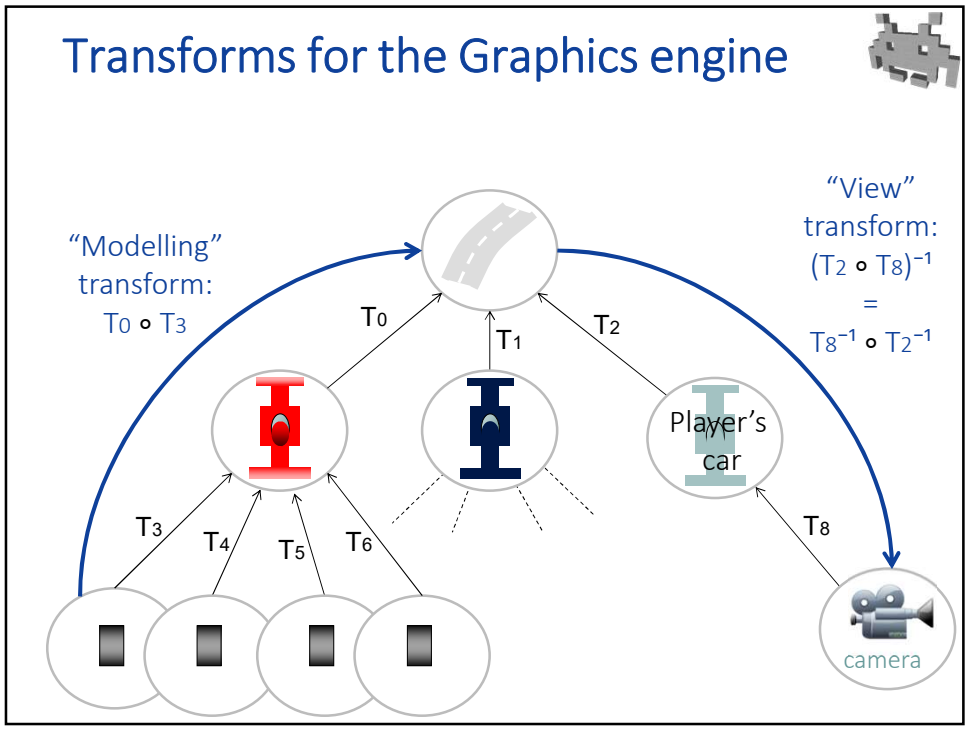
  …to World Space
- In Computer Graphics, the *inverse* of global transform of the camera is called the View Transforms

60

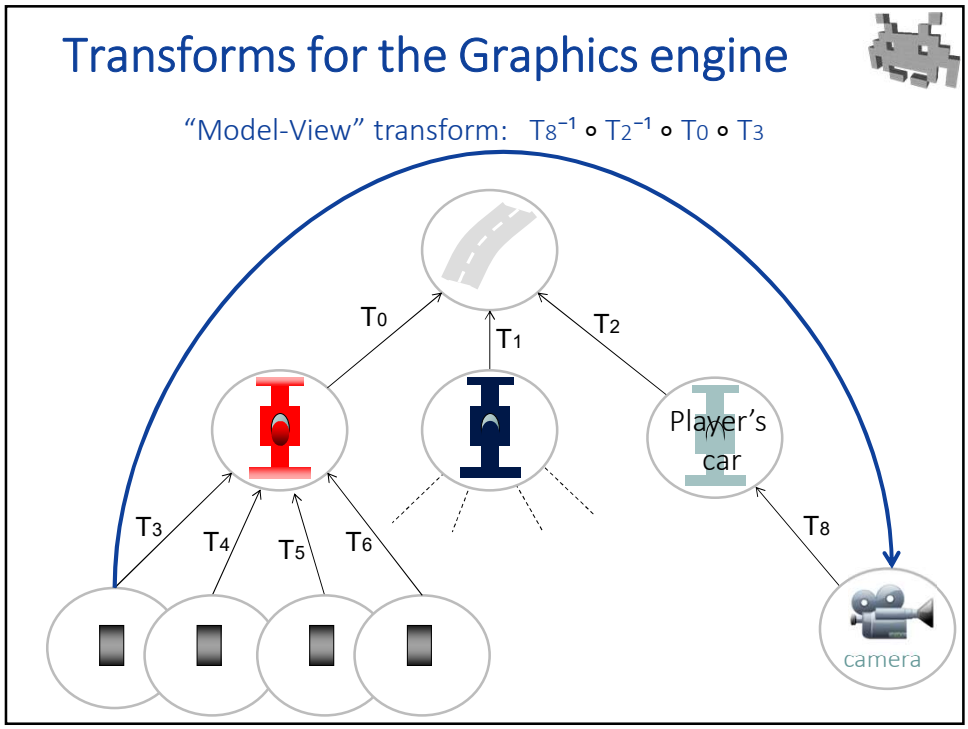
Transforms for the Graphics engine (link to Computer Graphics course)

- In a rendering engine, there are a few „traditional” transformations useful when rendering an object
- They are named:
  - "Model" matrix: from object space to world space
    - Captures how the scene is modelled (by a scener)
    - It's what we call "global" transformation
    - "matrix" only because trasnforms are usually encoded as 4x4 matrices by Rendering engines & graphics APIs
  - "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
- Computing them from the scene graph is direct!

61

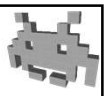

Transforms for the Graphics engine
"Modelling" transform: T0 ∘ T3
"View" transform: (T2 ∘ T8)−1 = T8−1 ∘ T2−1
T0
T1
T2
T3
T4
T5
T6
T8
Player's car
camera

62


Transforms for the Graphics engine
"Model-View" transform: T8−1 ∘ T2−1 ∘ T0 ∘ T3
T0
T1
T2
T3
T4
T5
T6
T8
Player's car
camera

63

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 convenient to perform the rendering
  - Because, in view space, coordinates describe where things are w.r.t. the camera!
  - For example: $z > 0 \Rightarrow$ object in front of the camera, $z<0 \Rightarrow$ object behind the camera (don't render)
- Camera animations = move camera
  - by doing anything that changes its global transformation
  - e.g., a script changing its local transform… or the one of its parent!

64

Changing a node positioning…

in view-space (refer the schema in the prev page)

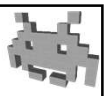
- Say $\mathbf{T}$ is (again) the transform consisting of moving an object 2 units on the X
- Assume the camera is in node H
- Event:
  - We want node L to undergo transform $\mathbf{T}$ in view space.
  - Meaning: we want L to be moved 2 units (camera space units) on the right of the screen
  - This is useful e.g. from a GUI point of view. Move an object as dragged by a mouse
  - Note: we still can only change its local transforamtion:

$$\mathbf{T}_L \leftarrow \mathbf{T}'_L$$

  - Task: find $\mathbf{T}'_L$

65

Changing a node positioning… *in view-space* : solution



- *View-space positioning* of L before the event:

$$T_H^{-1} \circ T_C^{-1} \circ T_B \circ T_E \circ T_L = V_L \circ T_L$$

V_L Model-View transform of L

- *After the event,* we want it to be:

$$T \circ V_L \circ T_L$$

- *After the event,* it will be:

$$V_L \circ T'_L$$

- Matching them:

$$T'_L = V_L^{-1} \circ T \circ V_L \circ T_L$$

66

Summary

Thanks to the ability to efficiently compute compositions and inverses of transforms…

- …we can store only the **local transform** of every node (= from its local space to its parent space), and dynamically get
  - the global transform (from its local space to world space),
  - the model-view transform (from its local space to camera space)
  - or actually any transform from the space of any node A to the space of any other node B in the graph

  (these transforms then encode the positioning of B w.r.t A)
- …we can quickly update local transforms to reflect any reconfiguration of the tree without affecting the spatial pos of objects

transforms representing **states**

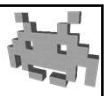
- …we can apply
  - any new transform T (rescaling, reposition, re-orientation)
  - to update spatial positioning of any node X (a node in the graph)
  - in the reference frame of *any other node* Y (another node in the graph) (e.g., apply transl, rot, scaling in world space, in local space, in view space, or in the space of any chosen node)

  by acting solely on the **local transformation** stored for X

transforms representing **actions**

67

Exercises
(refer the the schema in slide 44)



- Report the *global transform* of node **L**
- I place a camera in node **H**:
  report the View Transform for this scene
- Let T be a transformation that translates by (0,2,0)
- How do you apply that translation to node L …
  1. in L Space (the local space of L)?
  2. in World space?
  3. in View Space?

  (that is, which of the stored transformations changes, and how)
- Find the origin of space E in space H, and viceversa
- A microphone 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

71

Authoring a 3D scene in a game

- E.g. as a part of the Level Design
- Two different parts, by different artists:

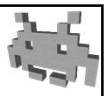
3D modellers make «scene props»
  - the 3D models to be assembled
  - (including their texutres etc)

sceners compose the scene
  - they assemble the props into a Scene Graph

= asset

72

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

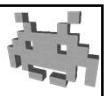
74

Scene Graph as a data structure: Mechanisms for shared subtrees

- The scene-graph will often contain multiple copies of shared subtrees
  - Existing implementation implement shared subtrees in different ways
  - In Unity: see “Prefabs”
  - In Unreal: see “BluePrints”

76

Rendering composite scenes: multi-instancing



- Each node contains a reference (e.g. pointer, or index) to one 3D object (e.g. a 3D mesh, etc) 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"

77

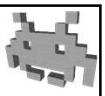
Nodes of a scene-graph in unity GameObjects & Transforms

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 (see physics)
  - "Colliders": geomtry proxies used for collisions (see physics)
  - "Particle systems" : (i.e. the "emitters" of particles)
  - Sound producers / receivers
  - Scripts …

81

Nodes of a scene-graph in unity
GameObjects & Transforms

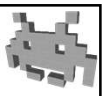


- The Transformation actually stores the local transf:
  - **`localPosition`**, **`localRotation`**, **`localScale`**
  - goes from a node to its parent
- the Global transformation can be accessed via the *properties*:
  - **`position`**, **`rotation`**, **`scale`** ("global" is left implicit)
  - what does getting / setting them really do? (exercise)
  - this it doesn't always work for "scale":
    ~~**`scale`**~~ `lossyScale` (read only)
    Why? (A: it's because anisotropic scaling)

it feels like assigning / reading a field, it actually means invoking setters/getters (C# trick)

83

Nodes of a scene-graph in UNREAL ENGINE
USceneComponent



A node within a graph with:

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

85