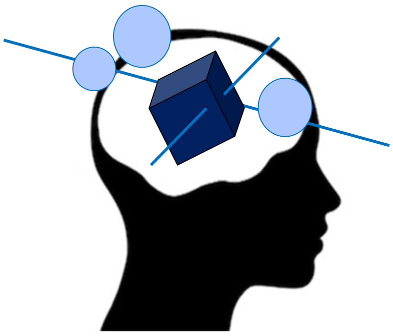
3D videogames

Spatial transforms for 3D games

Marco Tarini



2

Course Plan

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

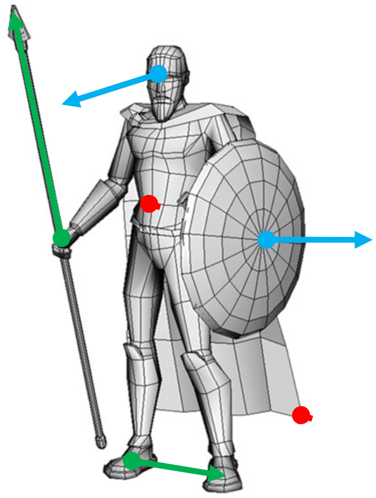
4

Recap:
things in games
are made of

points

vectors

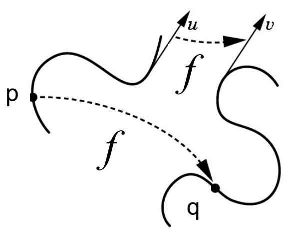
versors



5

A Spatial Transformations is a function

- input:
  - a point, or
  - a vector, or
  - a versor
- output:
  the same type
  as the input



q = f(p)
v = f(u)

6

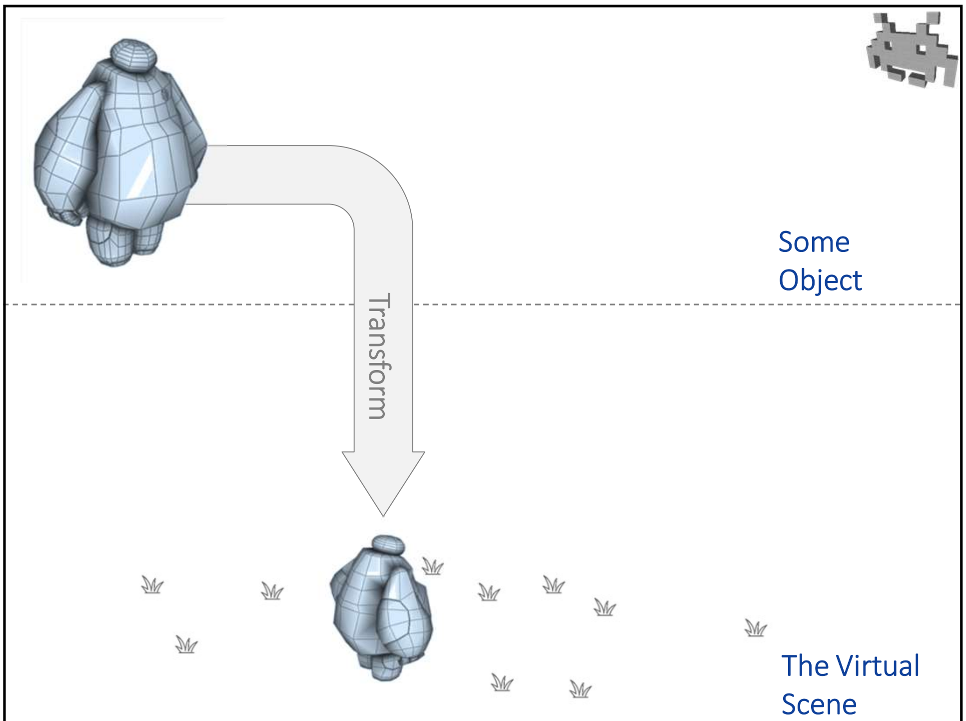
A Spatial Transformations is a function

point $\xrightarrow{f}$ point

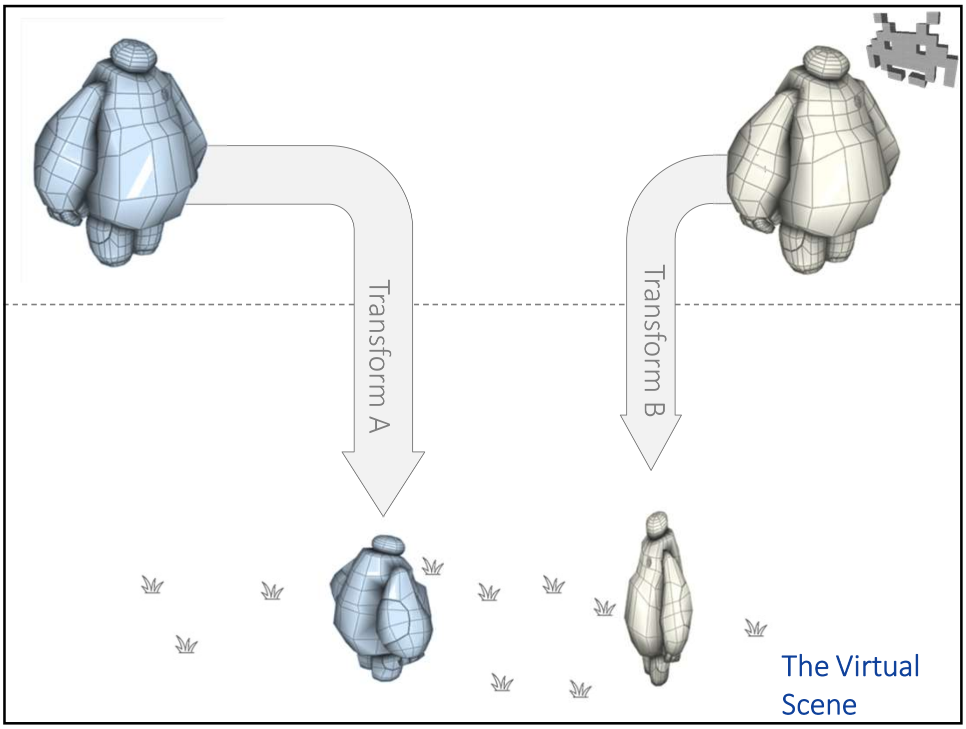
vector $\xrightarrow{f}$ vector

versor $\xrightarrow{f}$ versor

7

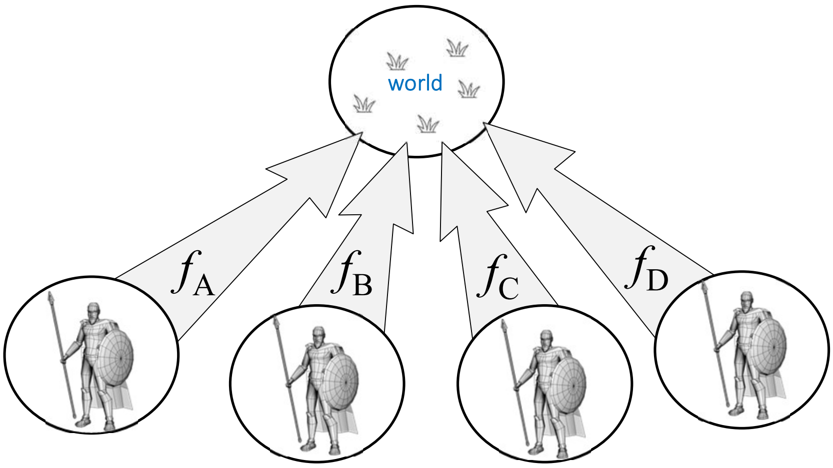


9



10

Basic concept: associate (and store)
a Transofrm to each object in the game



11

Transforms in 3D games

- Each object of the game is placed in the scene
  - the virtual world
  - *shared* by all the current objects

a character, a spaceship,
a bullet, a house, a camera,
a light source, an explosion,
a sound emitter, a spawn pos,
...*anything at all*!

- This is done by transforming that object
  - That is, by applying a transform to all points, vectors, versors of its representation
  - in all the corresponding assets
  - (for meshes: this is done on-the-fly, during rendering, by the rendering engine)
- A transform is associated and stored to each object
  - in CG, it would be called its « modelling transform »

12

Each object in the game: we store it’s transform

- The affine transformation $\mathbf{T}$ associated to an object in the game goes…
  - *from:* its own «object space»
    (or «local space» , or «pre-transform space»)
  - *to:* the common «world space»
    (or «global space» , or «post-transform space»)
- in CG, $\mathbf{T}$ would be called its « modelling transform »

13

How do we internally model and store a spatial transform?

- Many answers are possible and valid!
- In Computer Graphics and other fields, a particular useful class of transformations is used: the Affine transformations
- They can conveniently be stored as a 4×4 matrices
- *SPOILER*: for 3D Video-Games, this is not the ideal solution. Instead, we use a subset or another of that class
  - A better class is the one termed, in math, a "similarity"
- Because the transforms used in games are still affine, we will first discuss how Affine Transformation work

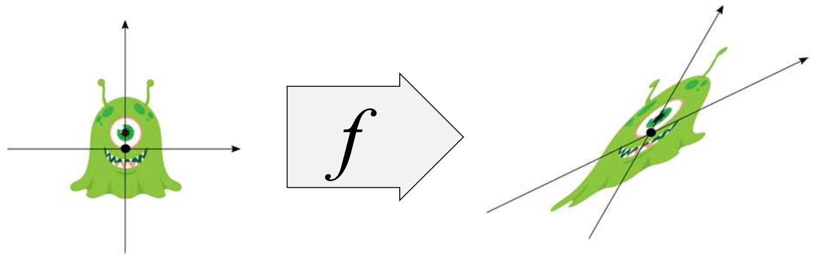
14

Affine transformations in a nutshell

- An affine transformation is just an arbitrary redefinition of the reference frame (orgin+axis)
  - The object will be transformed by re
- To define affine transformation, just *freely* a new reference frame (or space):
  - a new origin (a point)
  - a new set of 3 axis (3 vectors)
- Objects (vectors & points) will be transformed by reinterpreting their coordinates in the new reference frame

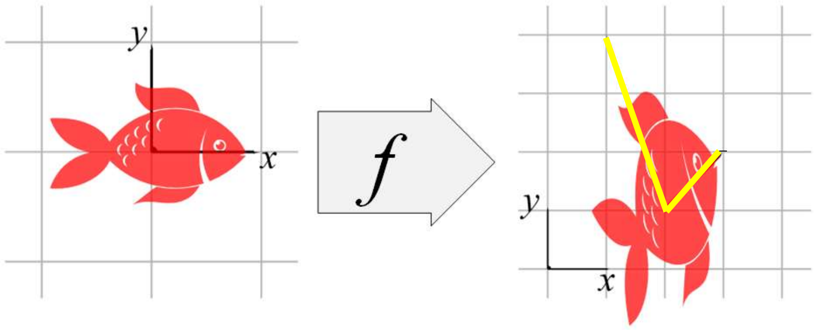
15

Affine transformations in a nutshell

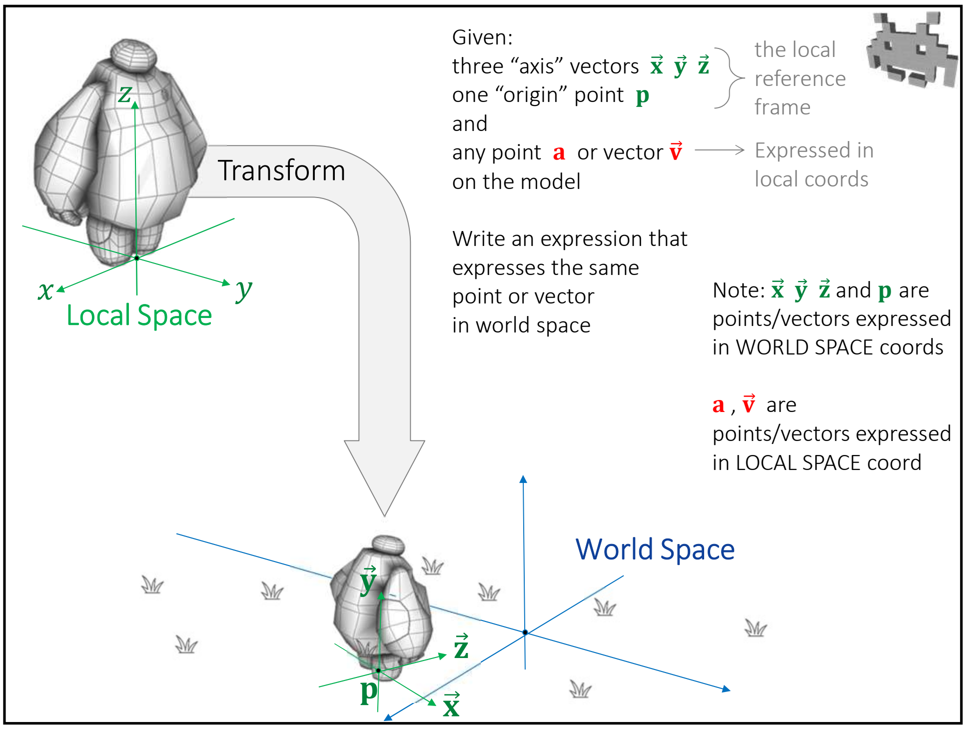


16

Affine transformations in a nutshell

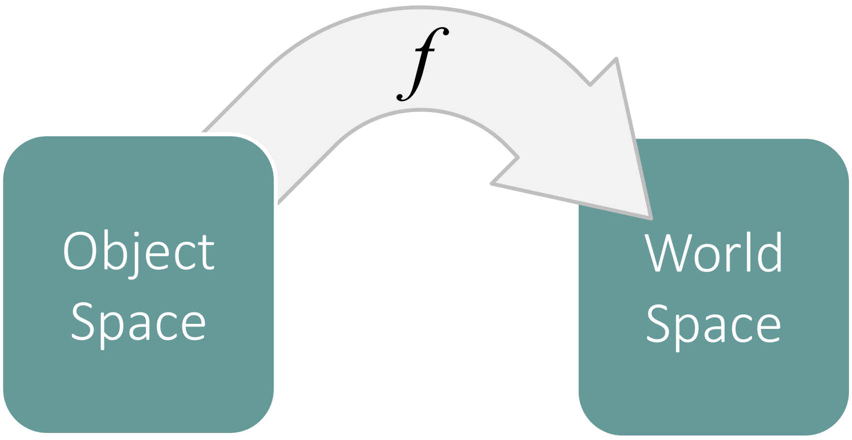


17



18

«Modelling» trasform as a change of reference frame



19

Math-problem: switching reference frame

Note: $\vec{\mathbf{x}}$ $\vec{\mathbf{y}}$ $\vec{\mathbf{z}}$ and $\mathbf{p}$ are points/vectors expressed in WORLD SPACE coords

$\mathbf{a}$, $\vec{\mathbf{v}}$ are points/vectors expressed in LOCAL SPACE coord

- Given
  - the local reference frame
    - three "axis" vectors $\vec{\mathbf{x}}$ $\vec{\mathbf{y}}$ $\vec{\mathbf{z}}$
    - one "origin" point $\mathbf{p}$

  and

  expressed in local coords
  - a point $\mathbf{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$ or vector $\vec{\mathbf{v}} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ on the model
- Write an expression to find
  - the corresponding point $\mathbf{a}'$ or vector $\vec{\mathbf{v}}'$ but expressed in world space

20

Math-problem: switching reference frame

$$\mathbf{a}' = \vec{\mathbf{p}} + a_x\,\vec{\mathbf{x}} + a_y\,\vec{\mathbf{y}} + a_z\,\vec{\mathbf{z}}$$
$$\vec{\mathbf{v}}' = v_x\,\vec{\mathbf{x}} + v_y\,\vec{\mathbf{y}} + v_z\,\vec{\mathbf{z}}$$

these equations can be written concisely using *matrix notation*…

21

Affine Transf: how to apply them (in one slide)

points: $\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$ vectors: $\begin{bmatrix} X \\ Y \\ Z \\ 0 \end{bmatrix}$ versors: $\begin{bmatrix} X \\ Y \\ Z \\ 0 \end{bmatrix}$ transforms: $\begin{bmatrix} M & t \\ 0\ 0\ 0 & 1 \end{bmatrix}$

23

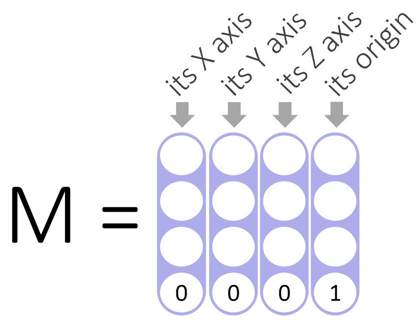
Affine Transf: how to apply them (in one slide) – [notes]

- Take the (*x*,*y*,*z*) cartesian coords of the point / vector / versor to be transformed
- Append a 4th “affine” coordinate *w* as
  - 1 , for points
  - 0 , for vector (or versors - sadly, we can’t discriminate)
  - Terminology: the resulting 4D vector is the “homogeneous coordinates” of the point/vector
- Multiply the transform matrix M by this (column) 4D vector to get the transformed point / vector
  - Note: as we wanted, points always become points, vectors (and versors) become vectors

24

Why it works: the Matrix is…

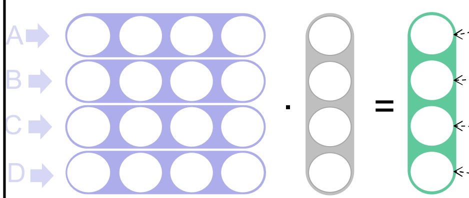
- …a direct description of the “starting” reference frame

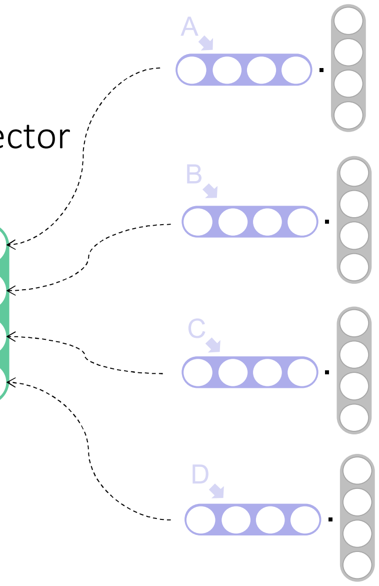


25

The Matrix-Vector product is…

- *n* dot products of its rows with the vector



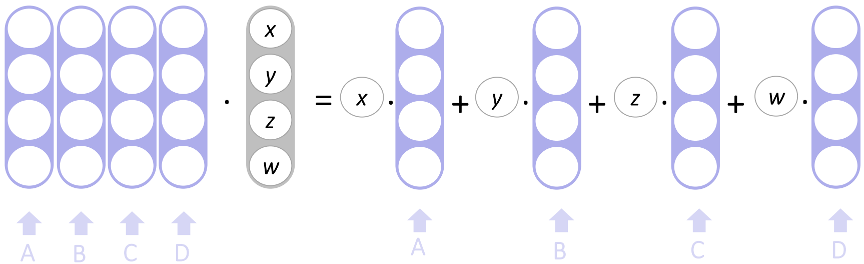


- *but also…*

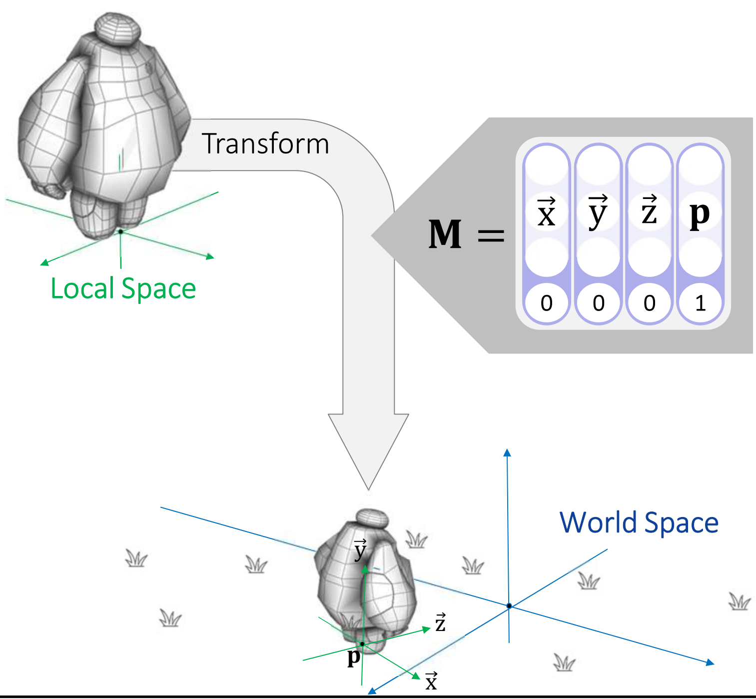
26

The Matrix-Vector product is…

- …a linear combination of its columns



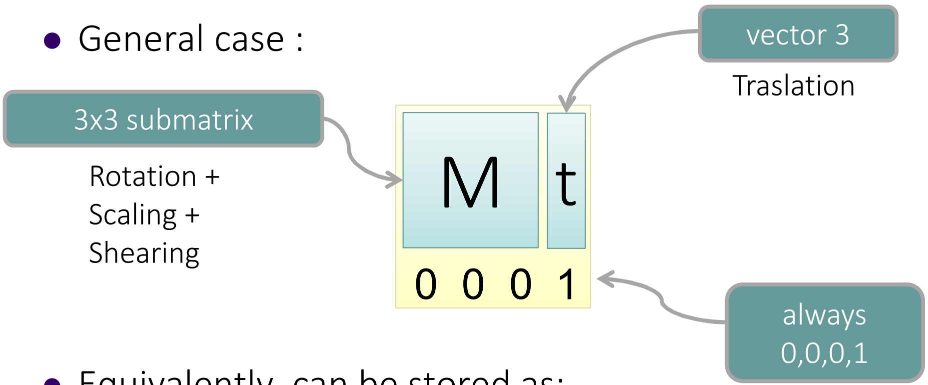
27



28

An affine transformation (in 3D) is simply a 4×4 matrix

- General case :



- Equivalently, can be stored as:
  Mat3x3 M   and   Vec3 t

30

Affine transformations: equivalent definitions

- a linear function:
  $$f(\mathrm{p} + k\vec{\mathrm{v}}) = f(\mathrm{p}) + k f(\vec{\mathrm{v}})$$
  $$f(h\vec{\mathrm{v}} + k\vec{\mathrm{w}}) = h\, f(\vec{\mathrm{v}}) + k\, f(\vec{\mathrm{w}})$$

- a transform which can be expressed as pre-multiplication of the transformed point/vector in affine coords by a 4×4 matrix M having as last row: 0,0,0,1

- a change of reference frame
  from a given *source* frame
  to a given *destination* frame

  origin
  + set of 3 axes

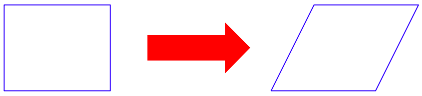
  not degenerate

33

Affine Transforms: what do they do in practice

EXAUSTIVE LIST!

- Rotations
- Translations
  - (of points – directions are unaffected)
- Scaling
  - uniform or not uniform
- Shearing (aka skewing)



- ... and their combinations

they include all "isometries" aka "isometric transform" aka "rigid transforms"

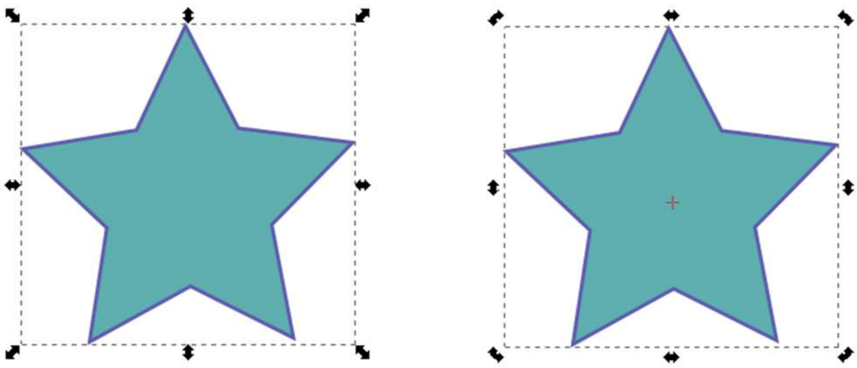
They include all "similitudes" or "conformal transform" (they don't change, the angles i.e. the shape)

closed w.r.t. compisition (we just multiply the matrices)

34

GUI tools to let an artist choose an affine transform in 2D

[DEMO]

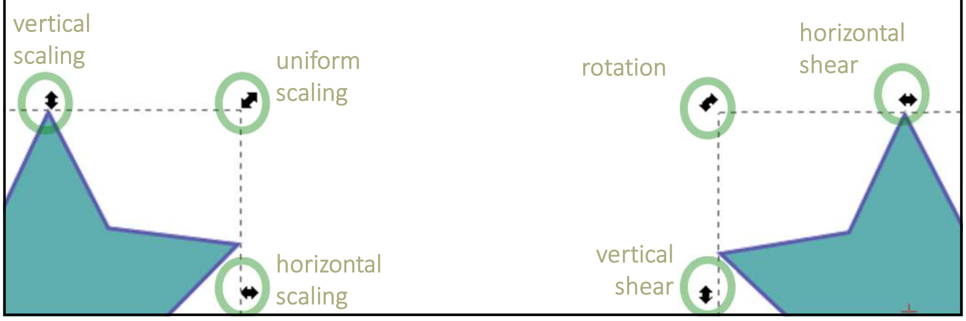


these familiar controls (plus drag-and-drop to translate) can be used to specify *any* affine transformation in 2D

35

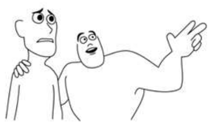
GUI tools to determine an affine transform in 2D

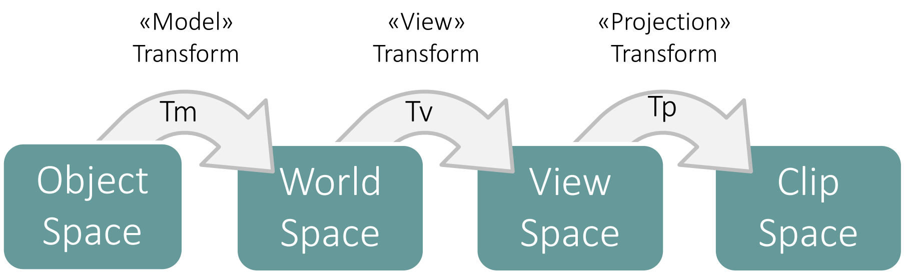
- 2D gizmos to specify an affine transformations



36

Affine transforms everywhere (in CG)

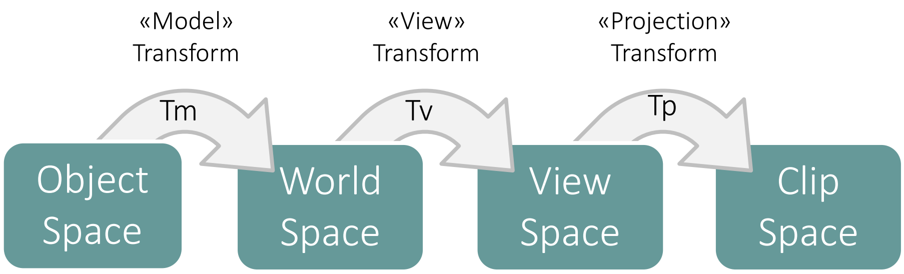




Transformation pipeline in Rendering (see CG course)

37

In CG, Transforms are used for many other purposes too (see CG course)



38

CG students please take note: 3D transformations are *not* necessarily 4×4 matrices

- a 4×4 Matrix is certainly *one way* to represent *one class* of 3D transformation
  - specifically: affine transformations
- sure, it's a useful class, and it's a good representation
  - elegant, sound, convenient…
  - in CG, this is so established that "matrix" is basically used a synonym of "transformation". E.g.: the "view matrix"
  - to learn more, see a Computer Graphics course
- In games, this method is not ideal
  - Q: What is the ideal way to represent something?
  - A: It depends on what you need to do with it!
  - What games need to do with transformations?

46

What do 3D *games* need to do with a transformations?

- store them
- apply them
- composite them
- invert them
- interpolate them

- and, design them

47

We want transformations to be…

- compact to store
  - what’s the memory footprint for one transform?
- fast to apply
  - how quick is it to apply it to one (or 99999) points / vectors / versors?
- fast/accurate to composite
  - given 2 transforms, is it easy to find their *composition* ?
  - (note: transform composition is not commutative!)
- fast to invert
  - how easy or fast is to find or apply the inverse transformation?
- easy to interpolate
  - given 2 transforms, is it possible/easy to *interpolate them*?
  - and, how «good» is the result?
- Intuitive to author / edit
  - how easy is it for modellers / sceners / animators / etc to define one?

48

Why we need fast compositions:

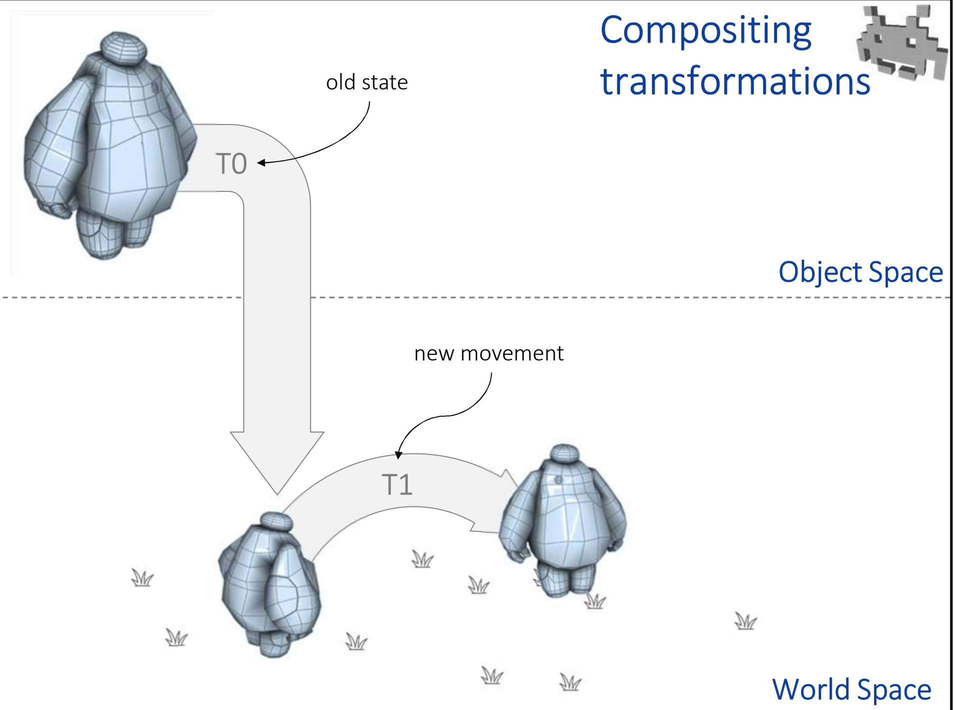
Moving objects in a 3D Game

- We move the objects in the scene by *changing the associated transform*
- Which is done by:
    - the scener / level designer ← at design time
    - the game physics
    - the AI scripts
    - the control scripts (press left arrow: move left)
    - …

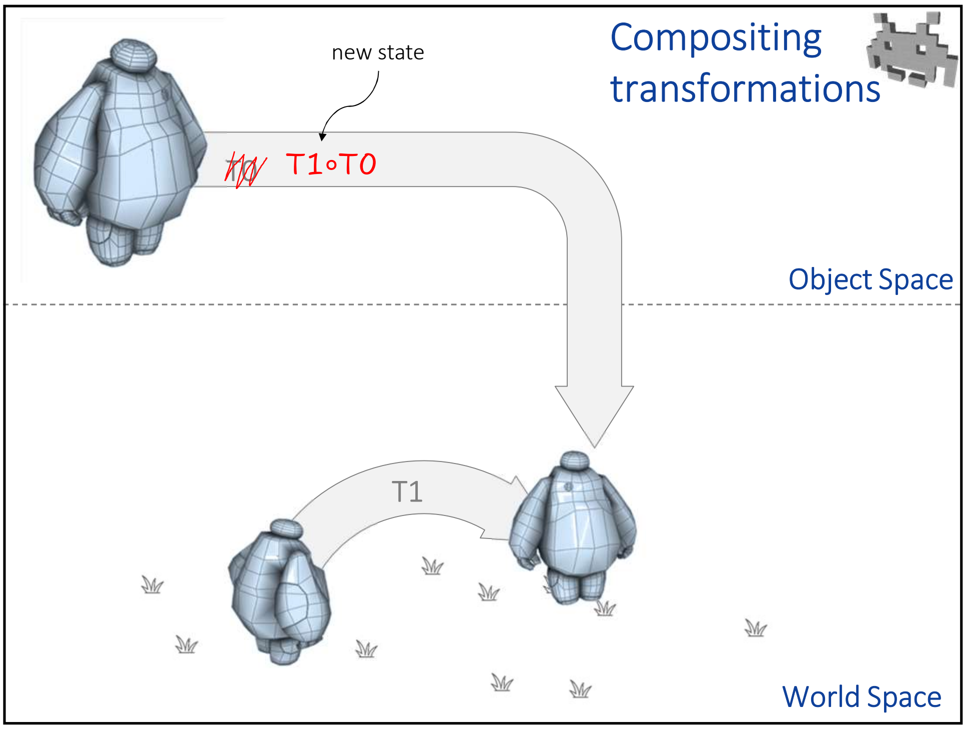
  at game execution time
- To apply transform T_{new} to an object, we substitute its transfrom T_{old} with $\underbrace{\mathrm{T}_{new} \circ \mathrm{T}_{old}}_{\text{composition}}$

49

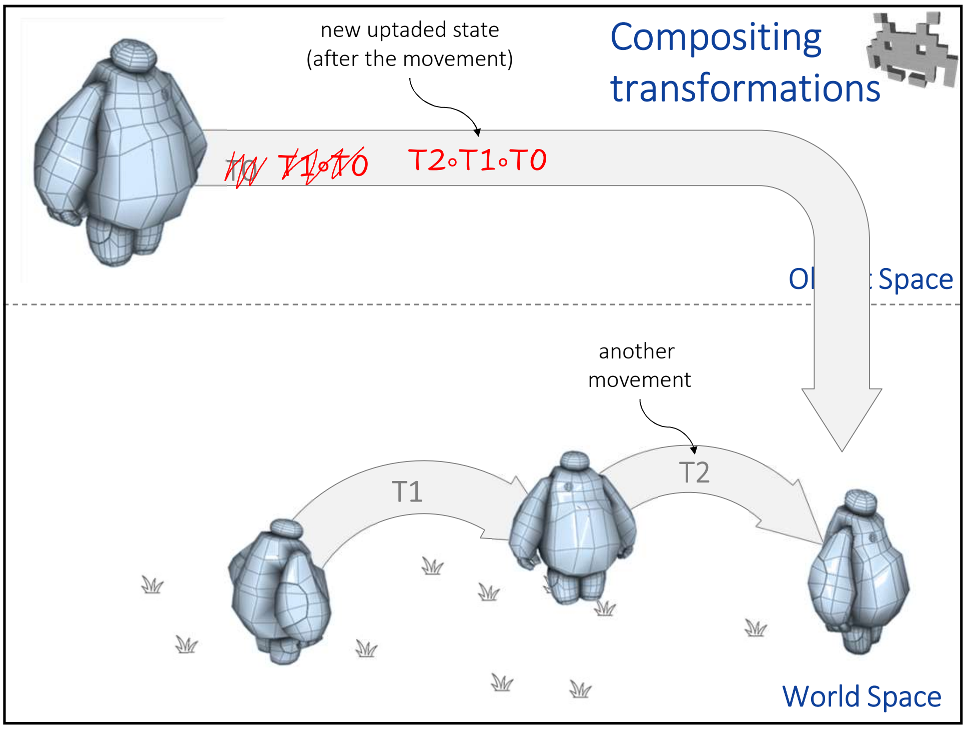
Compositing transformations



53


Compositing transformations
new state
~~T0~~ T1∘T0
Object Space
T1
World Space

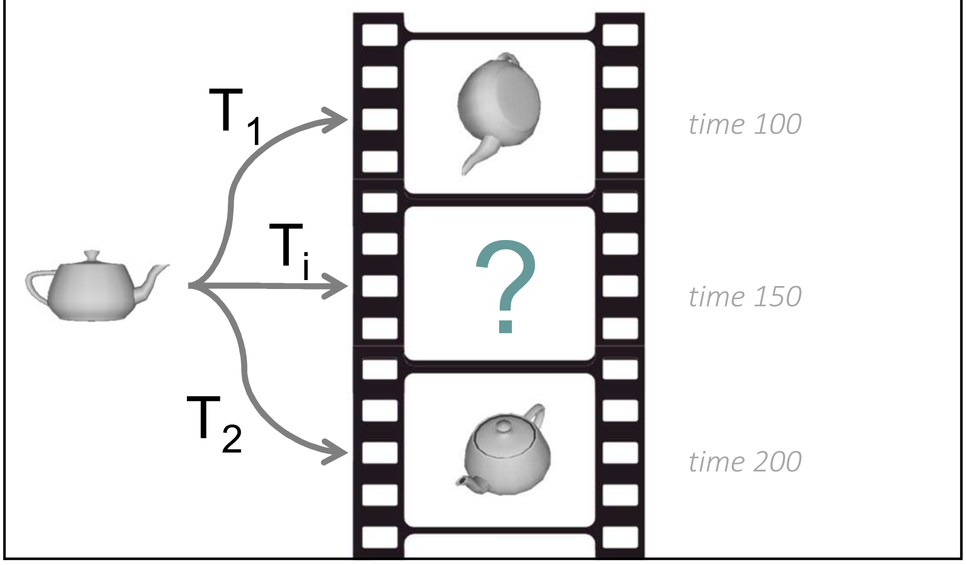
54


Compositing transformations
new uptaded state
(after the movement)
~~T0~~ ~~T1∘T0~~ T2∘T1∘T0
another
movement
T1
T2
World Space

55

Why we need transform interpolation:

Animations



58

Why we need transform inversions : ex1

«Where has Bob been hit?»

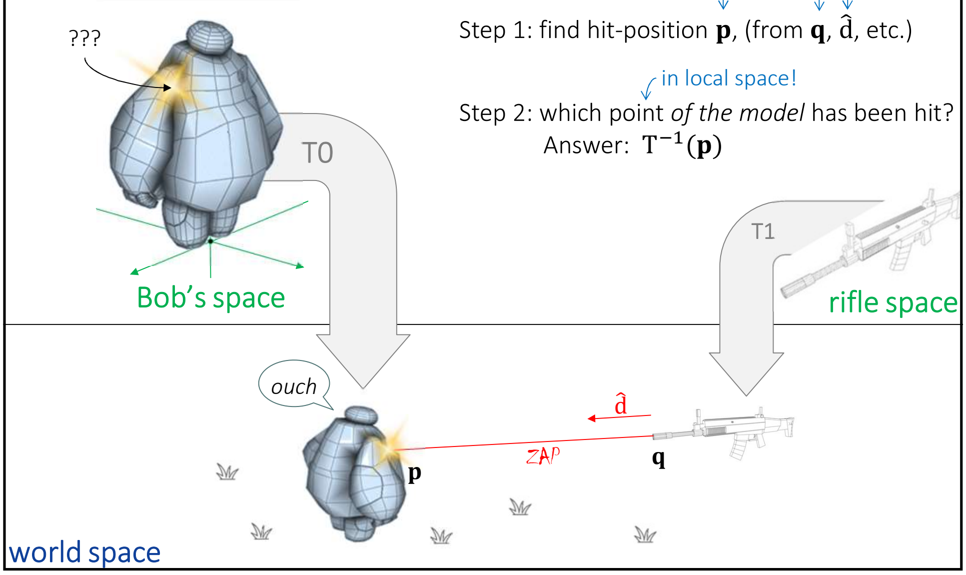
in global space!

Step 1: find hit-position $\mathbf{p}$, (from $\mathbf{q}$, $\hat{\mathrm{d}}$, etc.)

in local space!

Step 2: which point *of the model* has been hit?

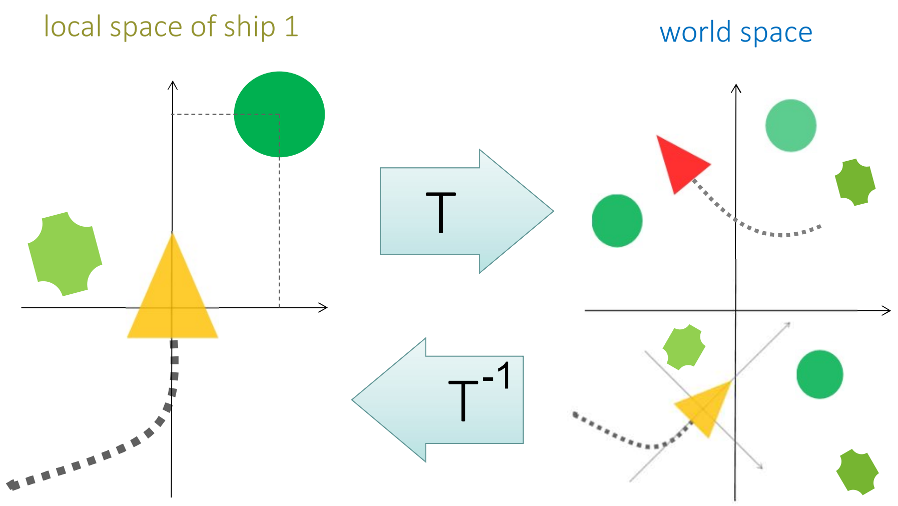
Answer: $\mathrm{T}^{-1}(\mathbf{p})$



59

Why do we need to invert transformations: example 2

the AI point of view



60

Recap: what do 3D games need to do with a transformations?

- store
- apply
- composite
- invert
- interpolate

- (and, design/author)

61

Recap:
we want transformations that are …

- compact to store
  - With a 4x4 Matrix: 16 numbers ☹
- convenient to apply (matrix: 16 numbers ☹)
  - With a 4x4 Matrix: matrix-vector product (not too bad)
  - But: versors become vectors ☹
- good to composite
  - With a 4x4 Matrix: matrix-matrix products (~128 scalar operations!)
  - Plus: they become distorted after many compositions
- fast to invert
  - With a 4x4 Matrix: matrix inversion. Not the quickest!
- easy to interpolate
  - With a 4x4 Matrix: we can interpolate easily each of 16 numbers, but results aren't the expected one: distortions
  - i.e. the interpolation between of 2 rigid transformation is not rigid
- intuitive to author / define
  - With a 4x4 Matrix: not always. Need to specify all vectors axes

62

keep the components *separated*

a Transformation = {
- a Rotation
- + a Scaling — uniform ~~or not~~ no need!
- + a Translation
- ~~+ Shearing~~ no need!

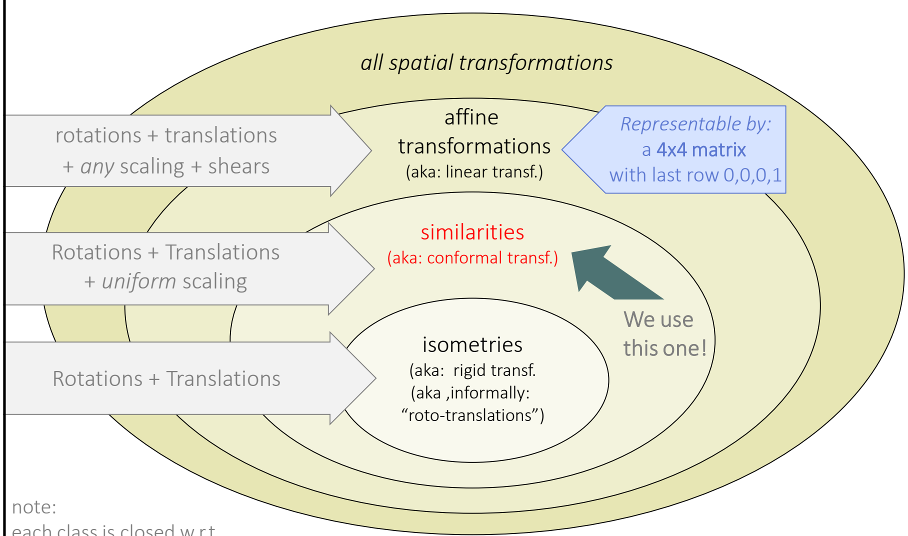
63

Which component do we need supported in a 3D game?

- Translation : necessary
  - and trivial
- Rotation : necessary.
  - and not that trivial (in 3D)
  - *will cover this in the next lecture (for now, rotation = **black-box function**)*
- Uniform scaling : may be useful
  - potentially useful, but…
  - alternative: scale 3D models once after import – maybe that's all you need
- Non uniform scaling : may be useful too
  - but problematic – see later
  - alterative: same as above
- Shear : least useful
  - and inconvenient: let's do ourselves a favor and NOT support it

65

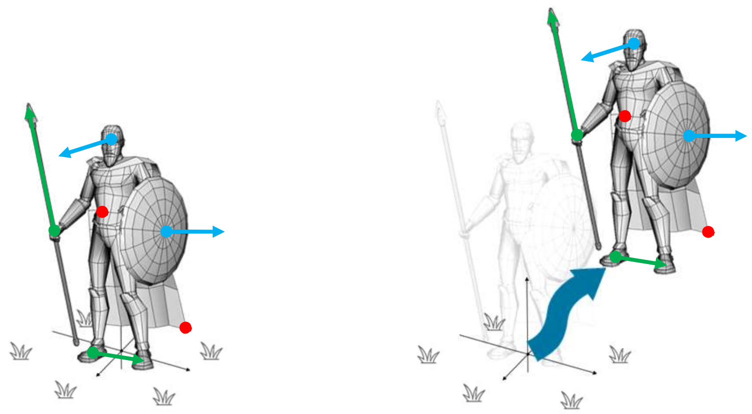
A math classification of spatial transformations



note:
each class is closed w.r.t.
cumulation, and (when they are invertible) inversion

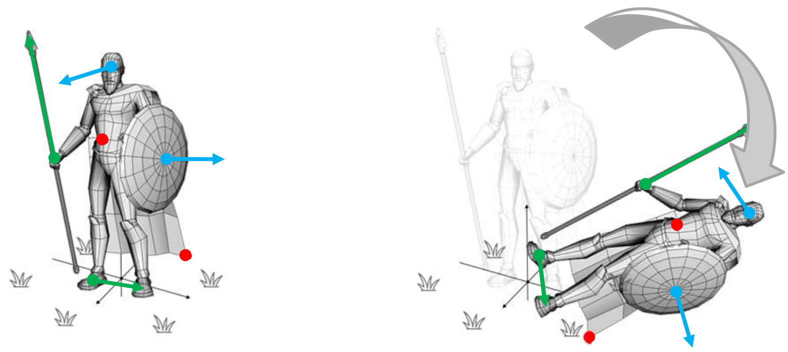
67

Transforms = Translations



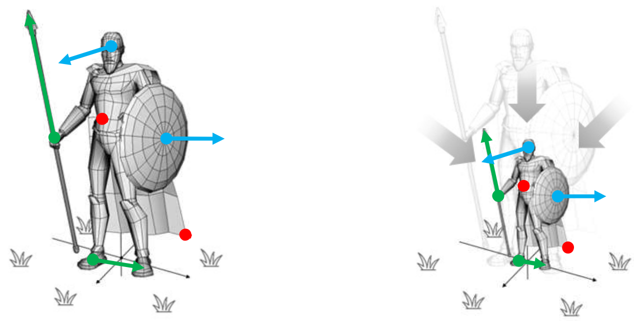
70

Transforms = Translations + Rotations



71

Transforms =
Translations + Rotations + <u>Scalings</u>



72

Effect of a transform on different things

	rotate:	scale:	translate:
points:	✓	✓	✓
vectors:	✓	✓	✗
versors:	✓	✗	✗

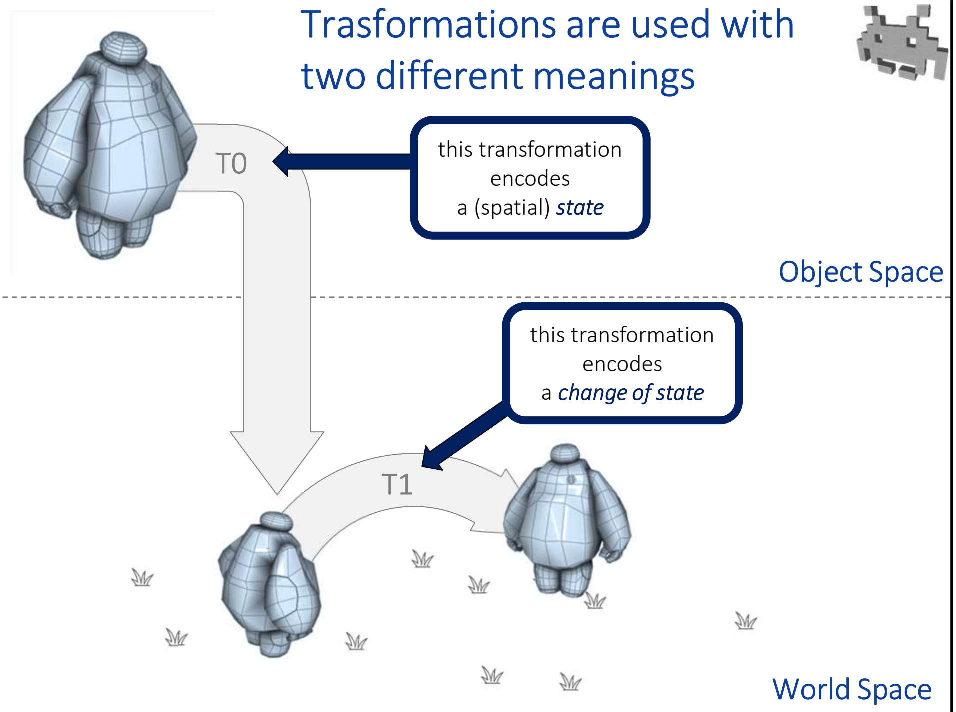
73

Effect of a transform on different things

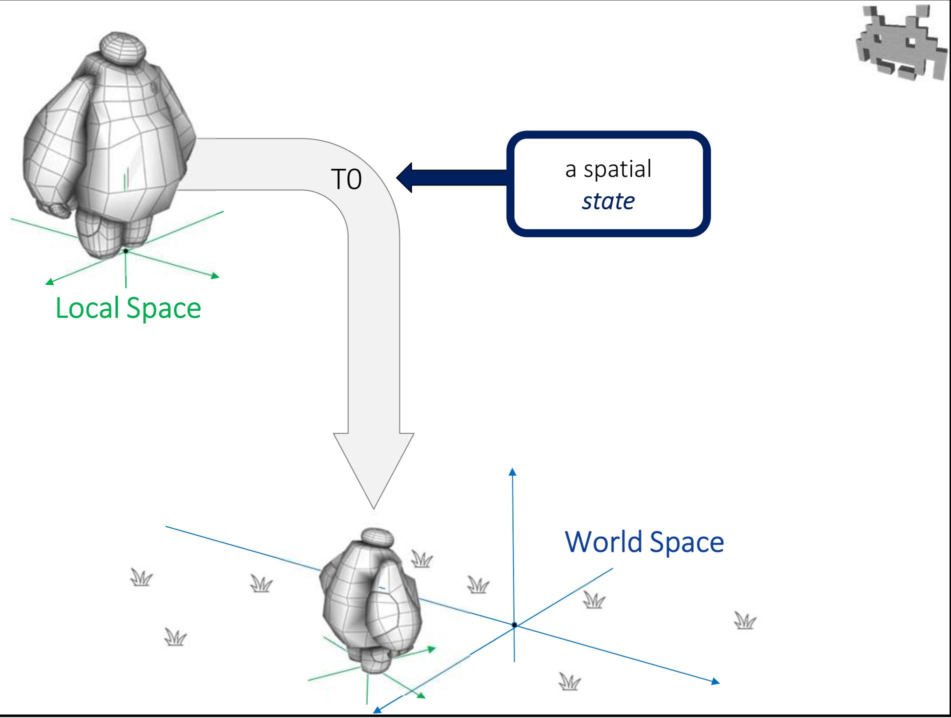
- Rotation:
  - Applies to Points, Vectors, Versors (just the same)
- Uniform Scaling:
  - Applies to Points, Vectors (just the same)
  - Leaves Versors unaffected!
- Translation:
  - Applies to Points only.
  - Leaves Vectors, Versors unaffected!

75

Trasformations are used with two different meanings

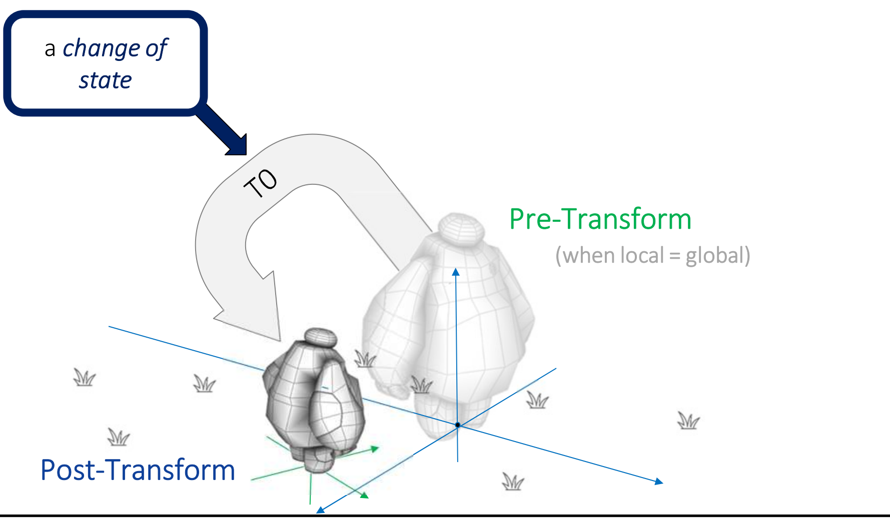


77



78

Or, equivalently…



79

two ways to see a transformation:

a change of state		a state
Translation *the act of displacing (moving) an object*	OR	Position *where the object currently is*
Rotation *the act of spinning an object, reorienting it*	OR	Orientation *how object is currently oriented, its facing*
Scaling *the act of enlarging or shrinking an object*	OR	Size *how big the object currently is (1 = original size)*

80

UNREAL ENGINE

Translation *the vector of which the object is displaced*	⬅	Position *where the object currently is*
Rotation *how much the object is spun, re-orienting it*	⬅	Orientation *where the object is currently facing toward*
Scale *by how much the object is enlarged or shrunk*	⬅	Size *how big the object currently is (1 = original size)*

81

unity

Translation *the vector of which the object is displaced*	→	**Position** *where the object currently is*
Rotation *how much the object is spun, re-orienting it*	←	Orientation *where the object is currently facing toward*
Scale *by how much the object is enlarged or shrunk*	←	Size *how big the object currently is (1 = original size)*

82

A transformation class (example) 1/4 Fields

```
class Transform {
  // fields:
  float s;     // scaling/size
  Rotation r; // rotation/orientation
  Vector3 t;   // translation/position

  ...
}
```

used as a black-box for now

See next lecture!

83