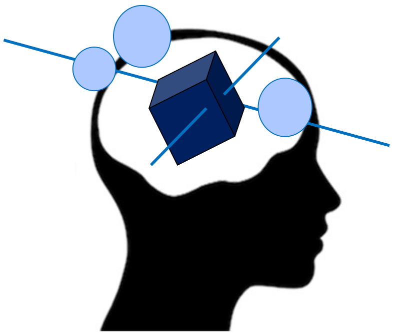
3D videogames

Spatial transforms for 3D games

Marco Tarini



2

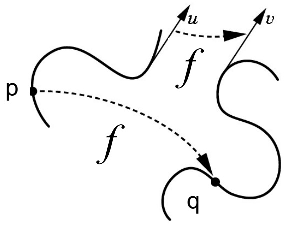
Course Plan

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: 3D Audio for 3D Games
lec. 11: Networking for 3D Games
lec. 12: Artificial Intelligence for 3D Games
lec. 13: Rendering Techniques for 3D Games

5

A Spatial Transformations is a function

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



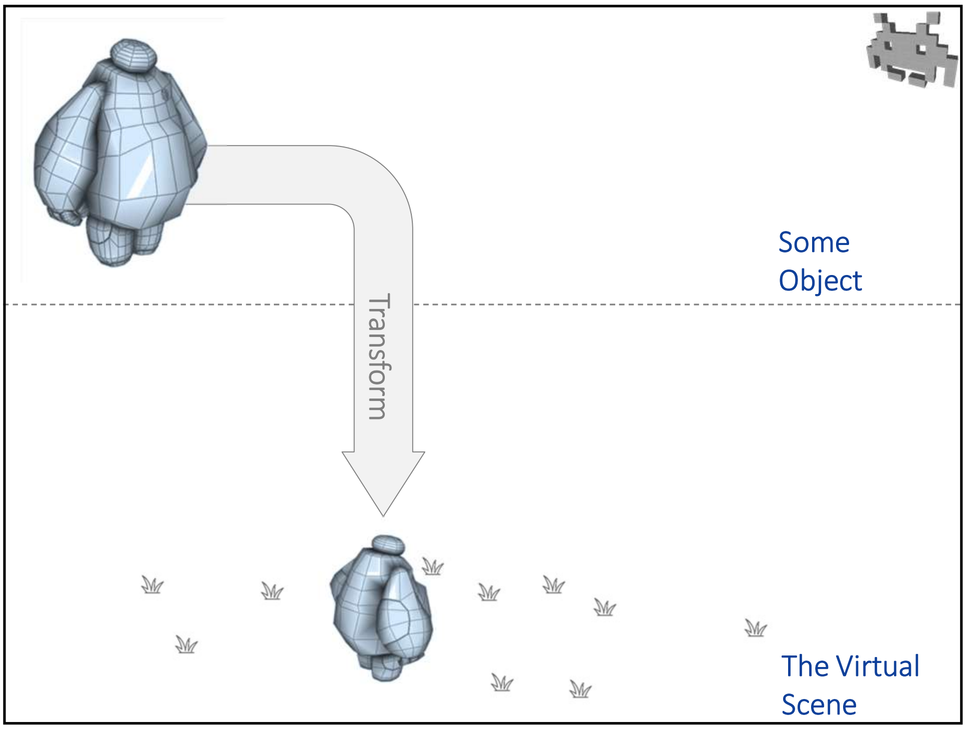
q = f(p)
v = f(u)

7

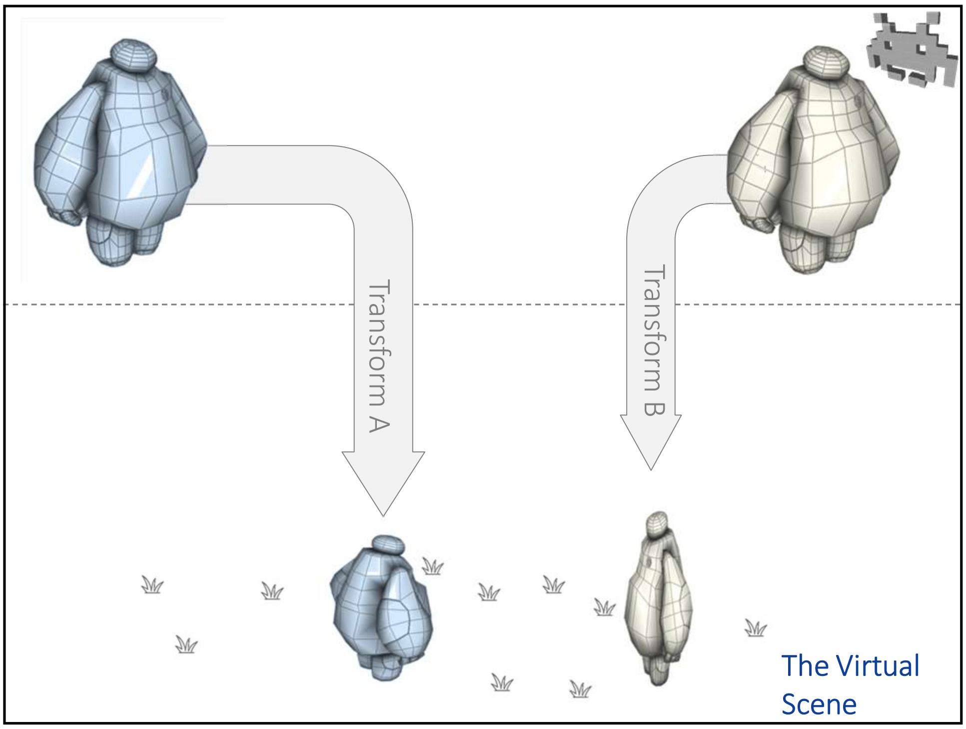
A Spatial Transformations is a function

point	f	point
vector	f	vector
versor	f	versor

8

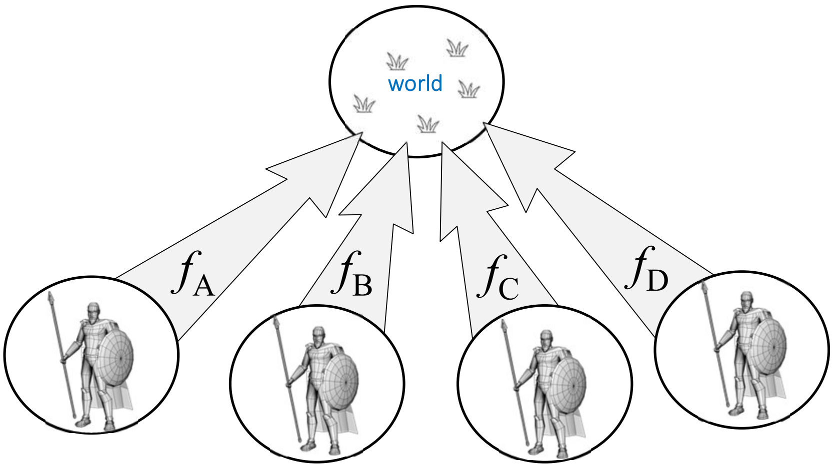

Transform
Some
Object
The Virtual
Scene

10


Transform A
Transform B
The Virtual
Scene

11

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



12

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 »

13

Each object in the game: we store its transform

- The transformation T associated to a 3D object in the game is a function that 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 Computer Graphics, T would stored as a matrix and would be called the « modelling transform » associated to that 3D object

14

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

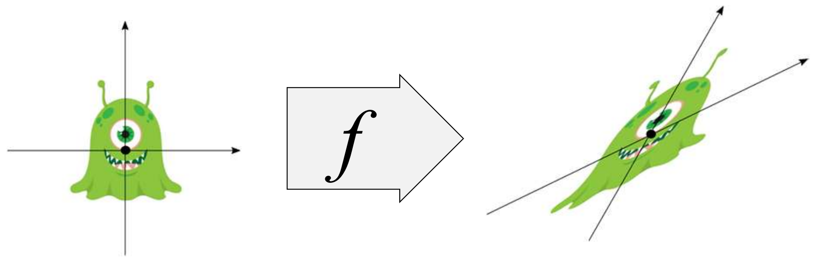
15

Affine transformations in a nutshell

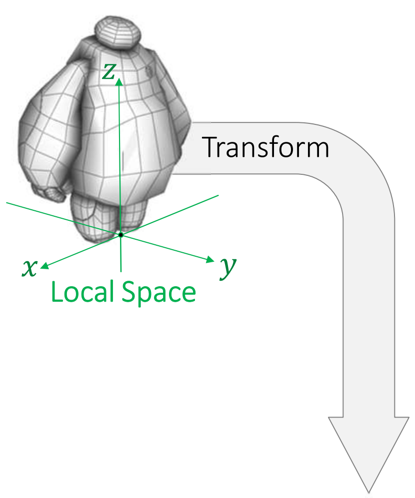
- An affine transformation can be seen as an arbitrary redefinition of the reference frame (orgin+axis)
- To define affine transformation, just *freely* a new reference frame (or space):
  - a new origin (a point)
  - a new set of 3 axis (3 vectors)
- Objects (vectors & points) will be transformed simply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

16

Affine transformations in a nutshell



17



Given:
three "axis" vectors $\vec{\mathbf{x}}$ $\vec{\mathbf{y}}$ $\vec{\mathbf{z}}$
one "origin" point $\mathbf{p}$
and
any point $\mathbf{a}$ or vector $\vec{\mathbf{v}}$
on the model

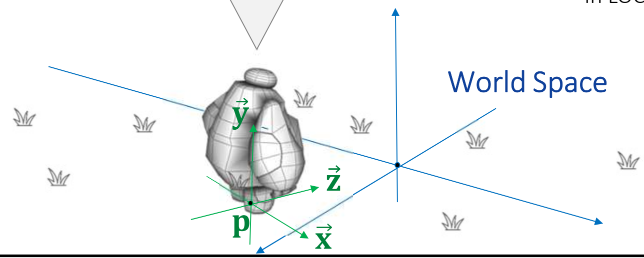
the local reference frame

Expressed in local coords

Write an expression that expresses the same point or vector in world space

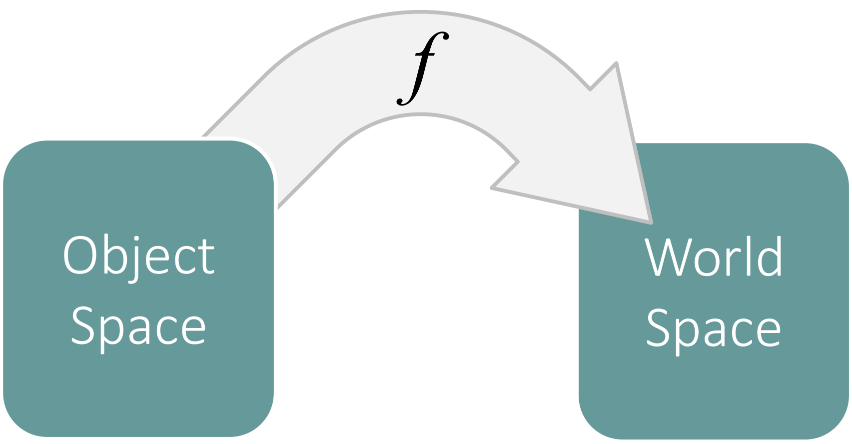
Note: $\vec{\mathbf{x}}$ $\vec{\mathbf{y}}$ $\vec{\mathbf{z}}$ and $\mathbf{p}$ are points/vectors expressed in WORLD SPACE coords

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



19

«Modelling» trasform as a change of reference frame



20

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

21

Math-problem: switching reference frame (solution)

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

$$\vec{\mathbf{v}}' = v_x\,\vec{\mathbf{x}} + v_y\,\vec{\mathbf{y}} + v_z\,\vec{\mathbf{z}}$$

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

22

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

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

23

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

- Take the (*x*,*y*,*z*) cartesian coordinates of the point, vector or versor to be transformed
- Append a 4th "affine" coordinate *w* as
  - *w* = 1 , for points
  - *w* = 0 , for vector (or versors - sadly, we can't discriminate)
  - Terminology: the resulting 4D vector is called 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

In code

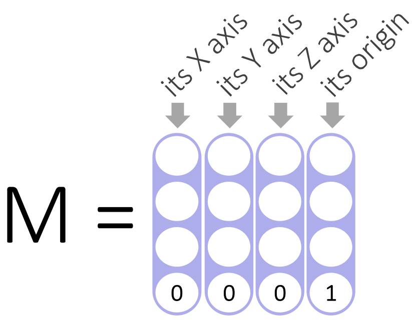
- Transforms as a 4x4 matrix

```
class Transform {
  // fields:
  Mat4x4 m;

  // methods:
  Vec3 applyToPoint( Vec3 p ){
     return toVec3( m * Vec4( p.x, p.y, p.z, 1 ) );
  }

  Vec3 applyToVector( Vec3 v ){
     return toVec3( m * Vec4( v.x, v.y, v.z, 0 ) );
  }
}
```

27

Why it works: the Matrix is…

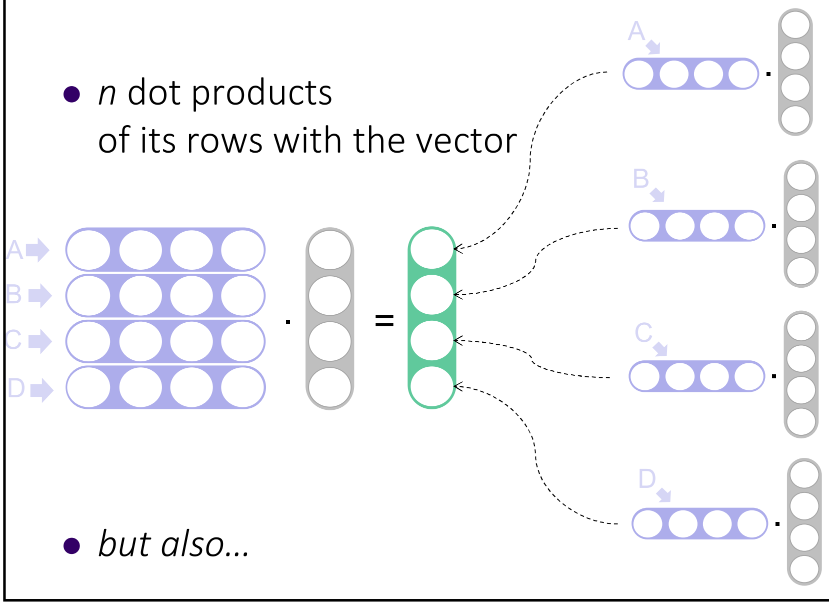
- …a direct description of the "starting" reference frame



29

The Matrix-Vector product is…

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

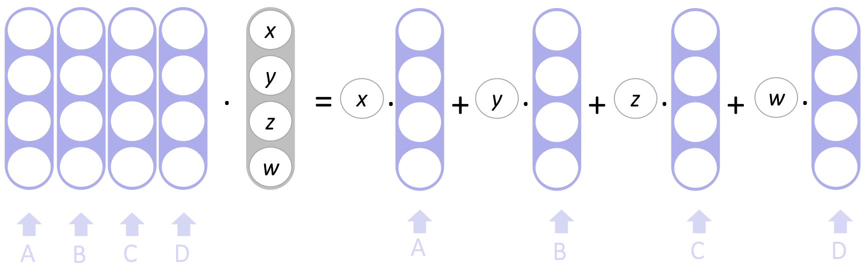


- *but also…*

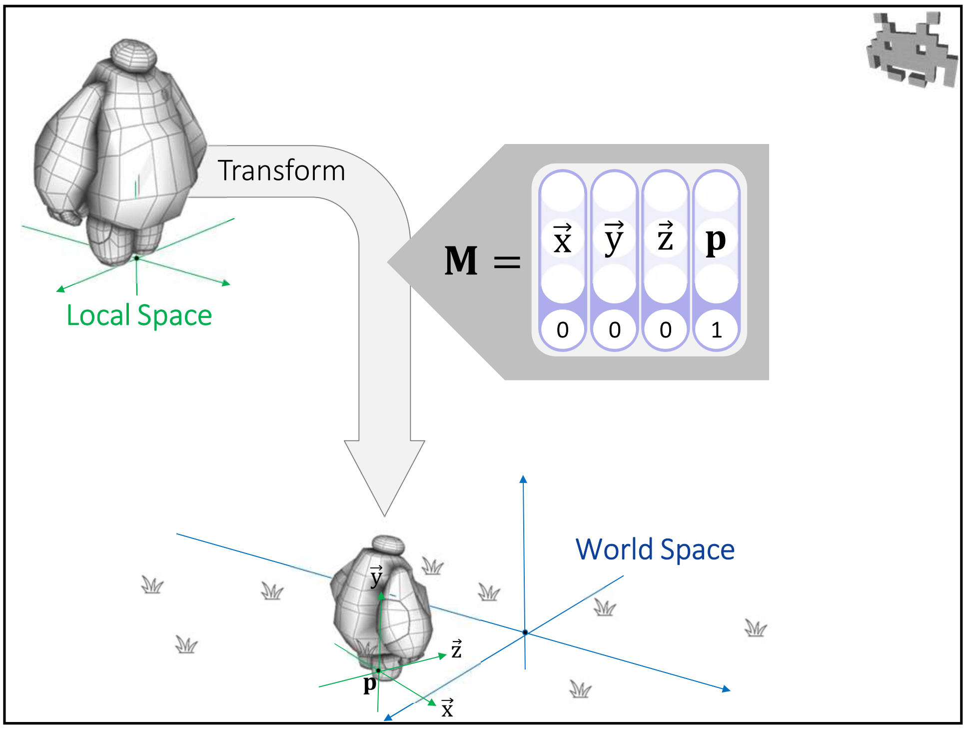
30

The Matrix-Vector product is…

- …a linear combination of its columns



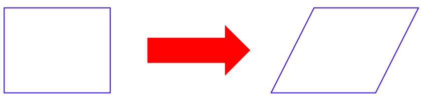
31



32

Affine Transforms: what do they do in practice

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



- … and their combinations

they include all “isometries” aka “isometric transform” aka “rigid transforms”

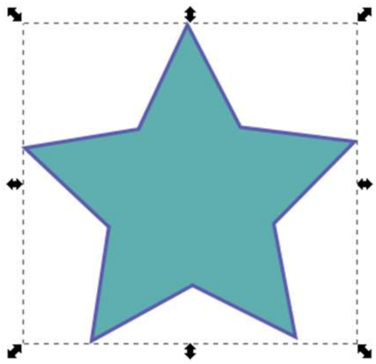
They include all “similitudes” or “conformal transform” (they don’t change, the angles i.e. the shape)

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

35

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

[DEMO]

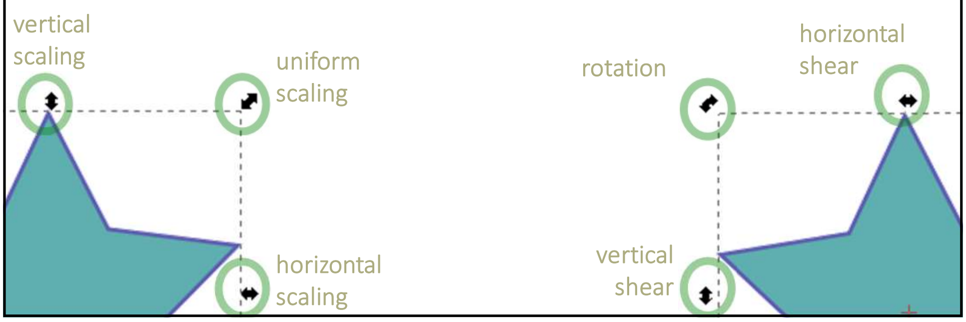


these familiar controls (plus drag-and-drop to translate)
can be used to specify *any* affine transformation in 2D

36

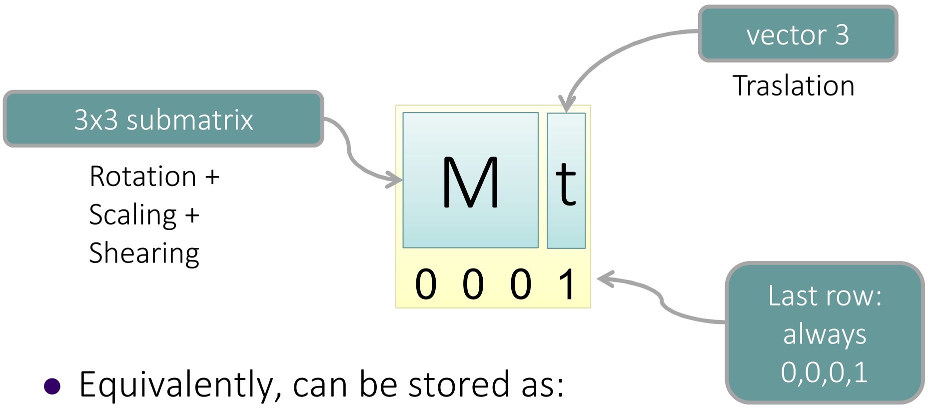
GUI tools to determine an affine transform (in 2D)

- 2D gizmos to specify an affine transformations



37

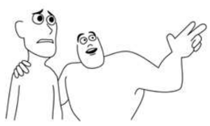
A 4×4 matrix representing an affine transformation

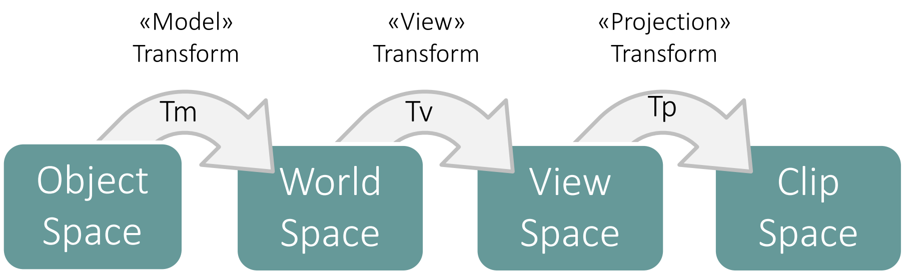


- Equivalently, can be stored as:
  Mat3x3 M and Vector3 t

38

Affine transforms everywhere (in CG)





Transformation pipeline in Rendering (see CG course)

39

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: all the affine transformations
- sure, it's a good representation
  - Elegant & sound – used by most graphics API (OpenGL, DX…)
  - in CG, this is so established that "matrix" is basically a synonym of "transformation". E.g.: the "view-matrix" = "view transform"
  - to learn more, see a Computer Graphics course
- In video games, this method is not ideal
  - Q: What is the ideal way to represent something? (in general)
  - A: It depends on what we need to do with it!
  - What games need to do with transformations?

45

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

- store them
- apply them
- composite them
- invert them
- interpolate them
- and, author them

46

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 it 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 / design
  - how easy is it for modellers / sceners / animators / etc to define one?

47

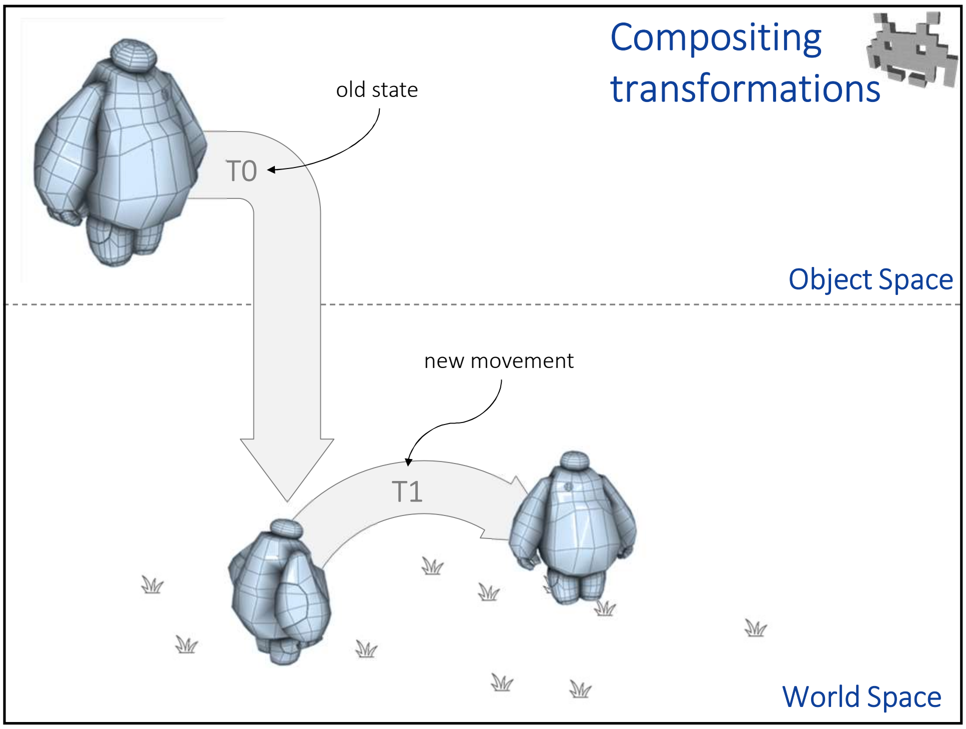
Why we need fast compositions:

Moving objects in a 3D Game

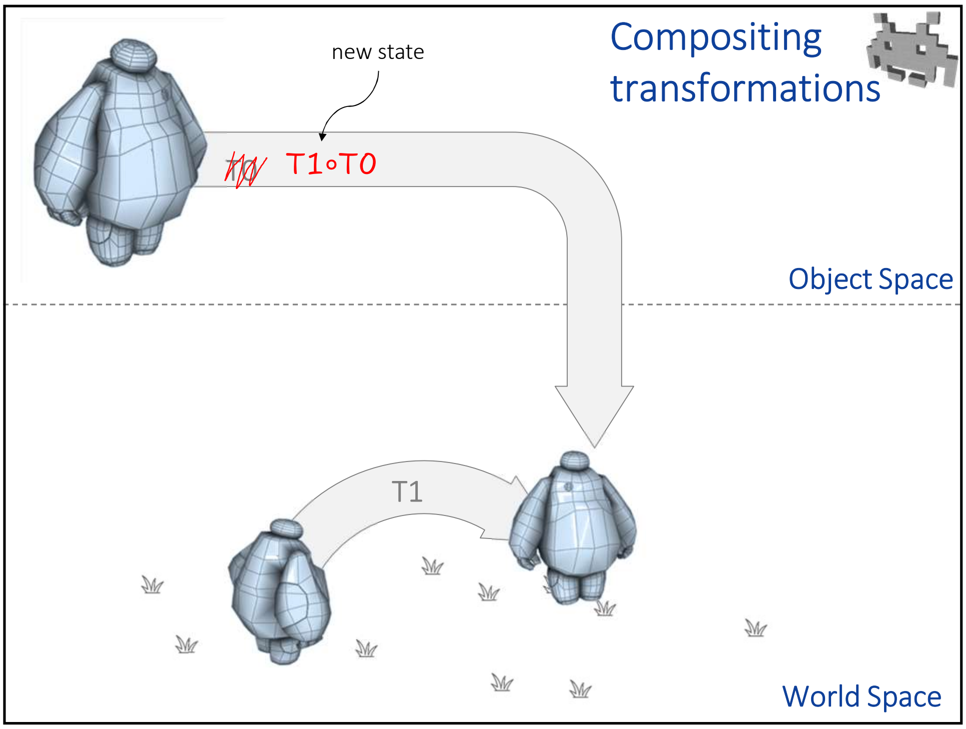
- We move the objects in the scene by *changing the associated transform*
- Which is done by:
  - the scener / level designer ← at design time
  - the game physics
  - the AI scripts
  - the control scripts (press left arrow: move left)

  (the game physics, the AI scripts, the control scripts: at game execution time)
  - …
- To apply transform T_{new} to an object, we substitute its transfrom T_{old} with $\mathrm{T}_{new} \circ \mathrm{T}_{old}$ (composition)

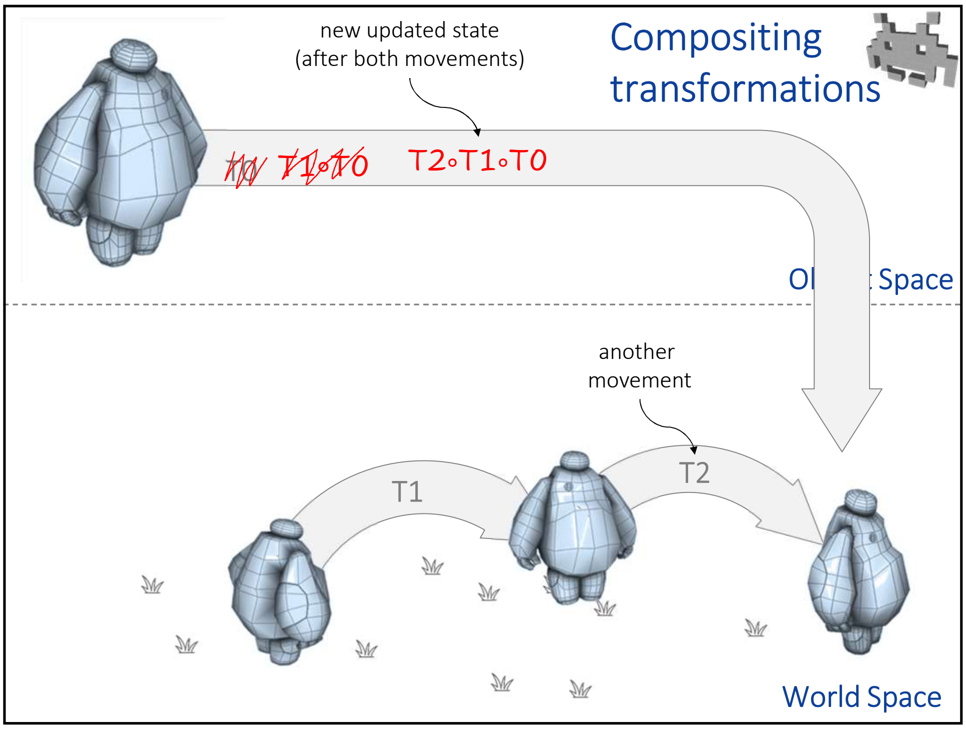
48


Compositing transformations
old state
T0
Object Space
new movement
T1
World Space

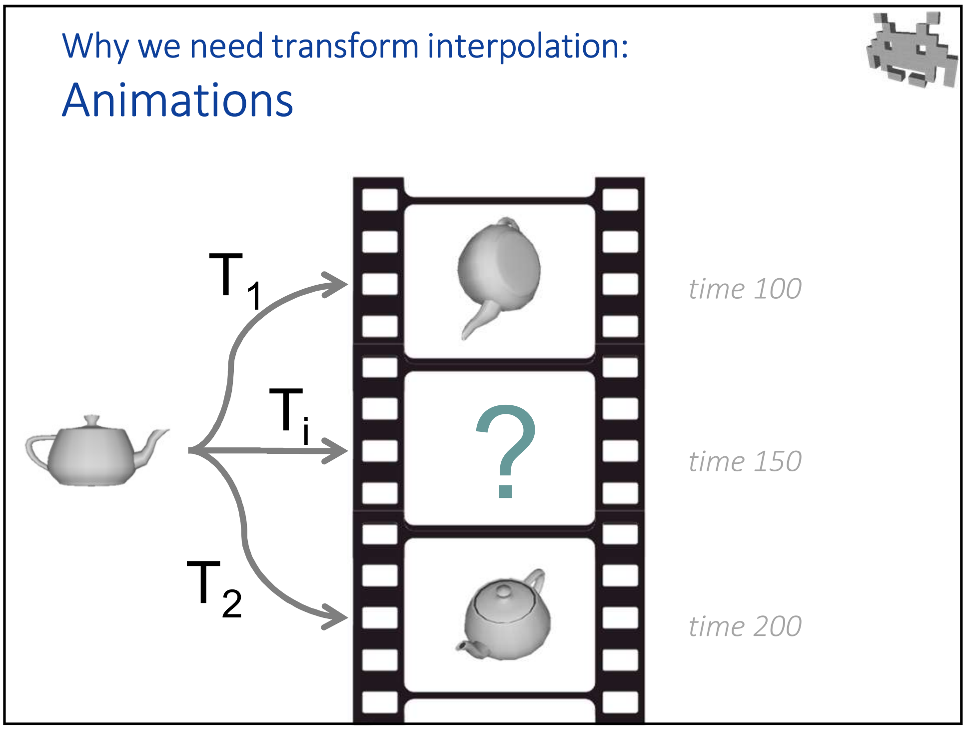
52


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

53


new updated state
(after both movements)
Compositing
transformations
T0 T1∘T0
T2∘T1∘T0
Space
another
movement
T1
T2
World Space

54

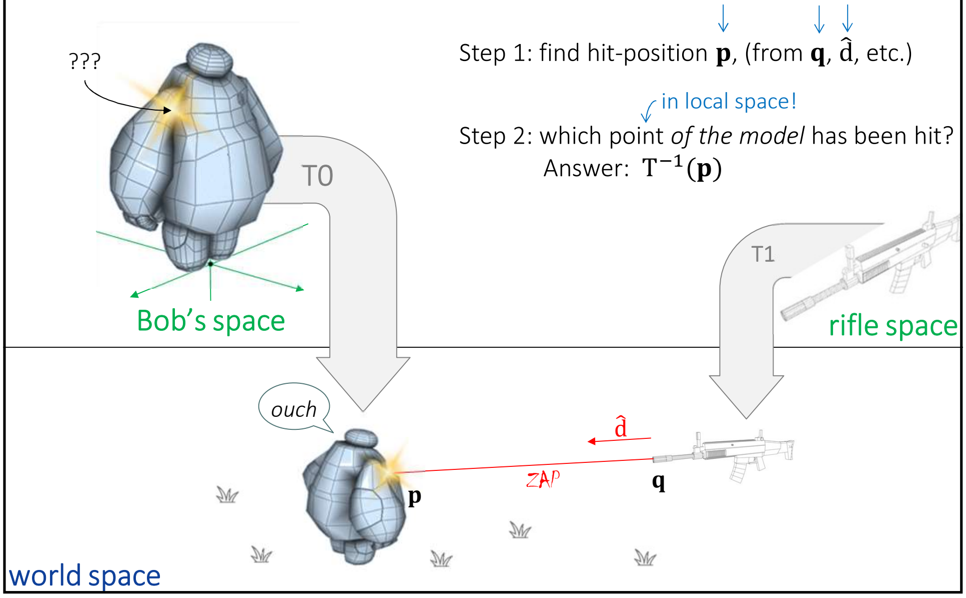

Why we need transform interpolation:
Animations
T_1
T_i
T_2
?
time 100
time 150
time 200

57

Why do we need to invert transforms : example 1
«Where has Bob been hit?»

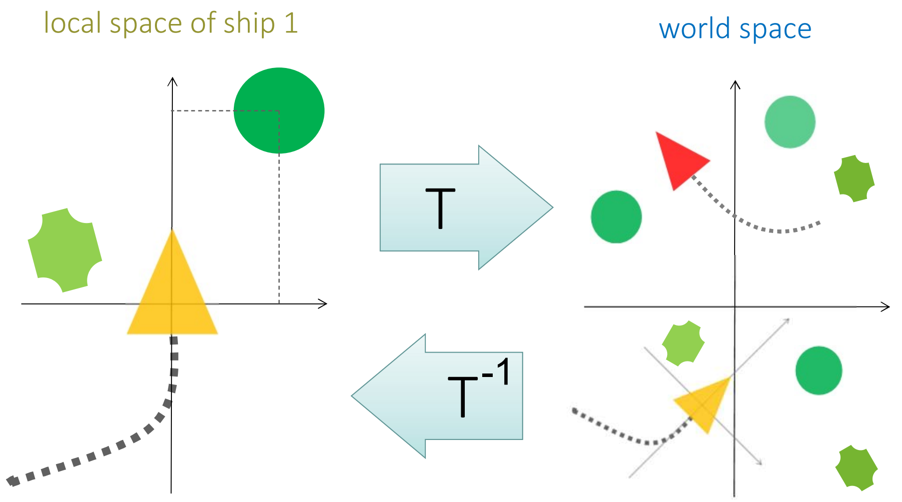
Step 1: find hit-position $\mathbf{p}$, (from $\mathbf{q}$, $\hat{\mathrm{d}}$, etc.) — in global space!

Step 2: which point *of the model* has been hit? — in local space!
Answer: $\mathrm{T}^{-1}(\mathbf{p})$



58

Why do we need to invert transformations: example 2
the AI point of view



59

Marco Tarini
Unviersità degli studi di Milano

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)
  - Issue: versors become vectors ☹ – length not preserved
- good to composite
  - With a 4x4 Matrix: matrix-matrix products (~128 scalar operations!)
  - Big problem: 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 happens
  - i.e. the interpolation between of 2 rigid transformations is not rigid
- intuitive to author / define
  - With a 4x4 Matrix: not very much. Need to specify all vectors axes.

61

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

- Translation : necessary
  - and trivial to do
- 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

62

keep the components *separated*

a Transformation = { a Rotation + a Scaling + a Translation ~~+ Shearing~~ }

no need!

uniform ~~or not~~ (no need!)

63

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!

65