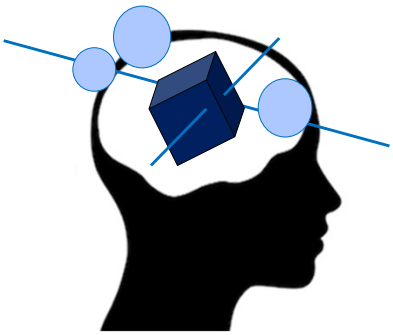
3D VideoGames - UniMi

Points, Vectors & Versors algebra



Marco Tarini

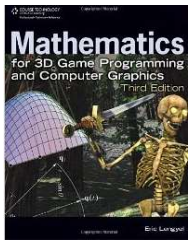
6

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

7

Suggested reading



Mathematics for 3D Game Progr. and C.G. (3rd ed)
Eric Lengyel
Chapters 2, 3, 4

8

Scalars, Points, Vectors, Versors & associated oprerations

They are the basic building blocks

- In the computation, for all videogame tasks
  - rendering
  - physic symulations
  - AI
  - 3D sound
  - …
- in the data structures of all 3D Assets
  - meshes, animations, etc

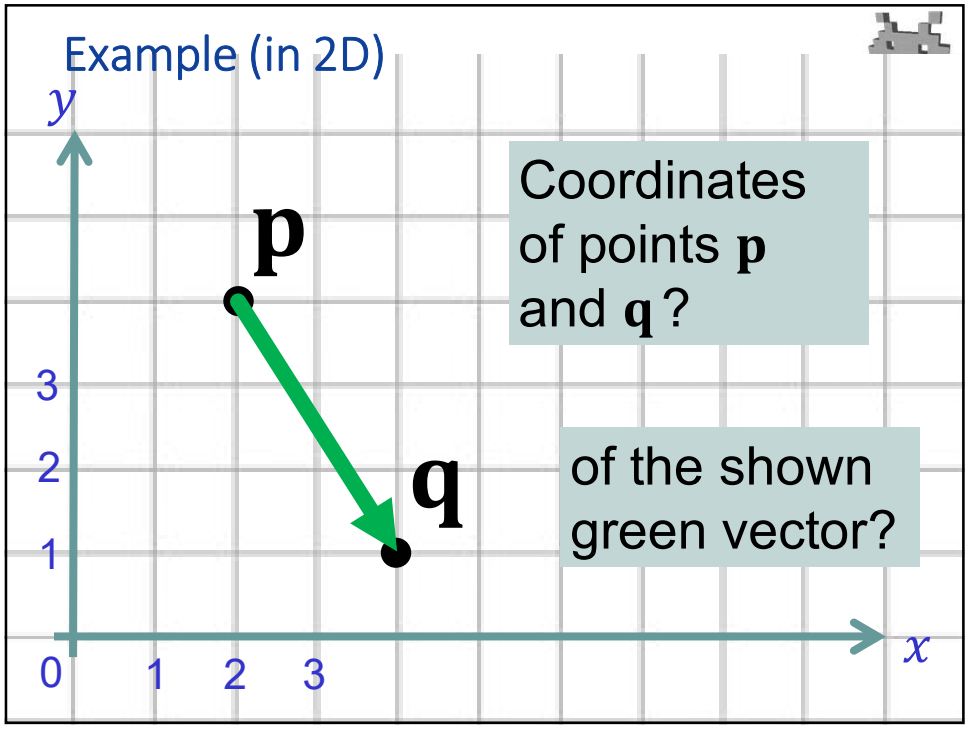
9

Point, Vectors, Versors

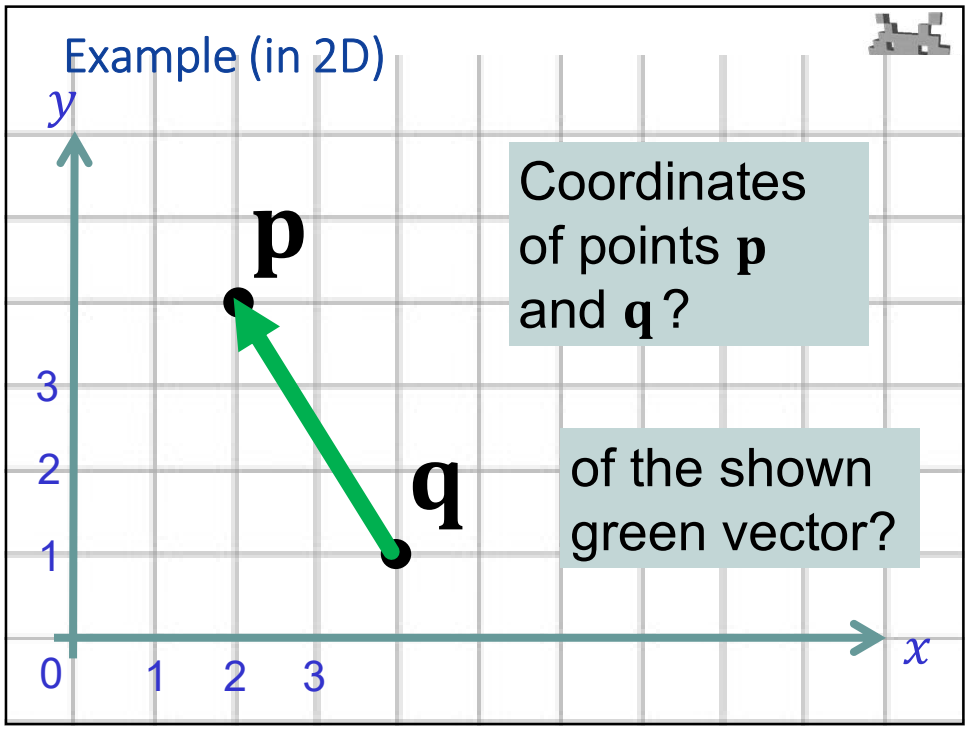
	represents:	example:	imagine it as...
Point	A position A location	Where a character is The center of a sphere	a small floating dot :-D
Vector	A displacement The difference between 2 points. The vector that connects them.	The velocity of a thrown knife The gravity acceleration How to reach the head of a character from its neck	a small arrow :-D *(length is relevant)*
Versor aka unit vector (as length = 1) aka normal aka direction aka normalized vector	A direction A facing	The view direction of a character The facing of a plane in 3D (i.e. its "normal") The direction of a line, or a ray A rotation axis	the same :-D *(its length is irrelevant)*

10

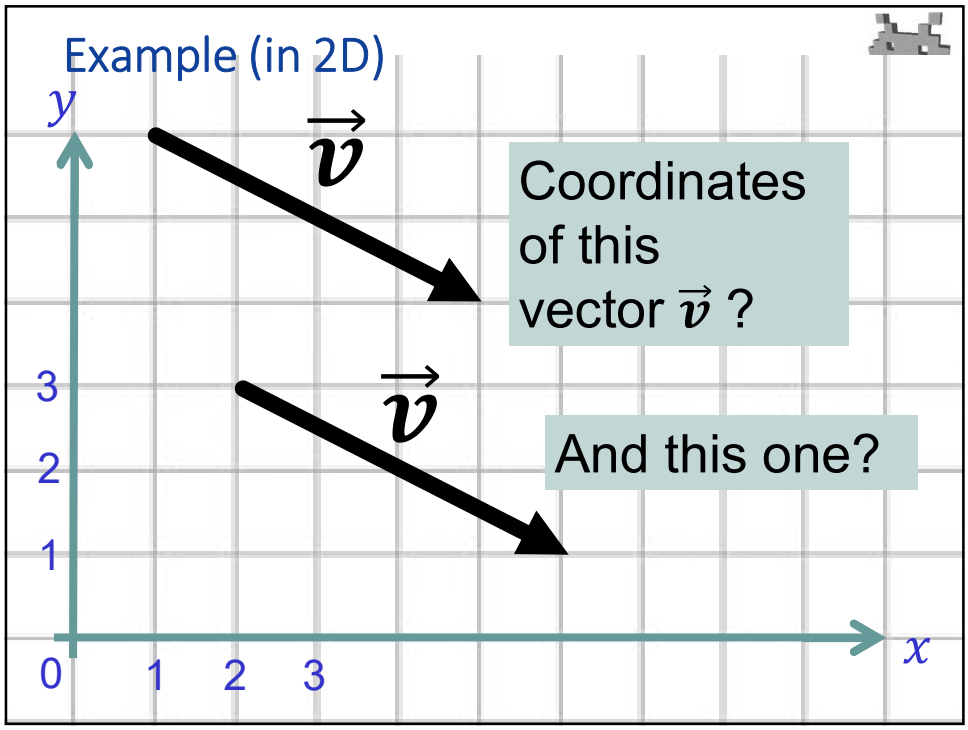
Example (in 2D)



12

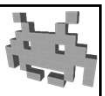

Example (in 2D)
y
x
0
1
2
3
p
q
Coordinates of points p and q ?
of the shown green vector?

13


Example (in 2D)
y
x
0
1
2
3
$\vec{v}$
$\vec{v}$
Coordinates of this vector $\vec{v}$?
And this one?

14

Points, Vectors, Versors: internal representation



Thinking in terms of programming languages,
it's helpful to imagine them as three *distinct* types of object

coords

deltas

no direct write-access
(must ensure unitarity)

```
class Point {
  // fields:
  float x,y,z;

  // methods:
  …
}
```

```
class Vector {
  // fields:
  float dx,dy,dz;

  // methods:
  …
}
```

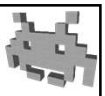
```
class Versor {
  // fields:
  const float dx,dy,dz;

  // methods:
  …
}
```

- Each supporting their own *distinct* operations & methods
- For now. In practice, many (all?) existing 3D libs choose to use the same class for the three of them (see later)

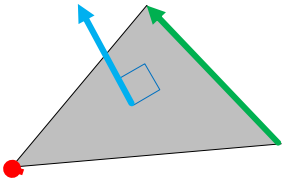
16

Points, Vectors, Versors …on a 3D floating tirangle



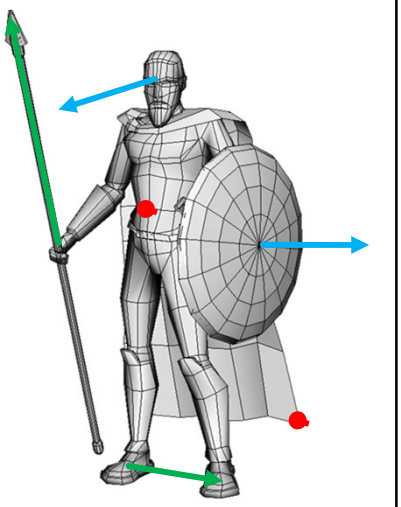
Examples of…

- point:
  - one vertex of the triangle
- vector:
  - one side of the triangle
- versor:
  - the «normal» of the triangle



17

Points, Vectors, Versors …in a character model

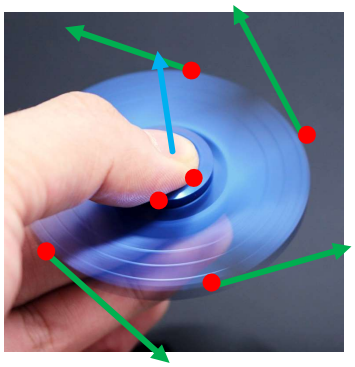


Examples of…

- points:
  - the pos of the navel
  - the pos of lewer-left tip of the hood
- vectors:
  - the vector connecting the L foot to the R foot
  - the vector from the hand to the tip of the lance
- versors:
  - the gaze direction
  - the facing of the shield

18

Points, Vectors, Versors …in a spinner

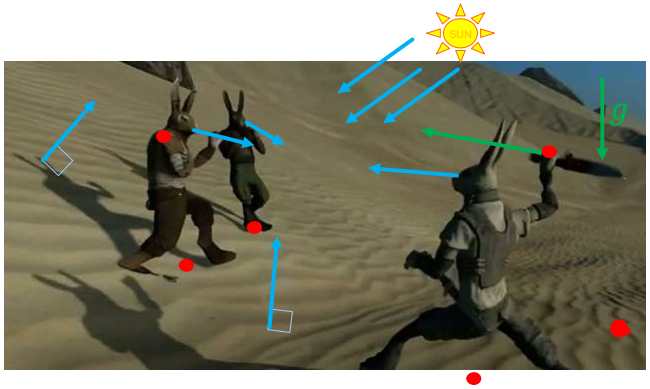


Examples of…

- points:
  - points of contact between finger-spinner
- vectors:
  - linear velocities of these four points
- versors:
  - rotation axis (direction of)

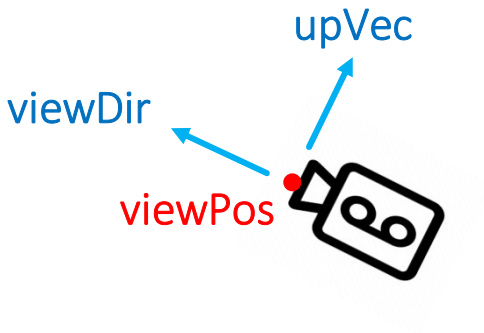
22

Points, Vectors, Versors
...in this screenshot



23

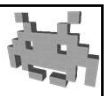
Stuff = Points + Vectors + Versors



Description of a camera
(its so called «extrinsic parameteres»)

24

Stuff = Points + Vectors + Versors

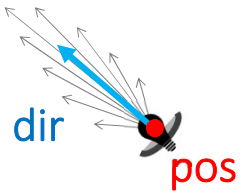


description of
a (directional) sound emitter

description of
a (directional) microphone

25

Stuff = Points + Vectors + Versors



description of a spotlight

26

The *algebra* of points, vectors, versors (and scalars)

- make sure you understand each of operation in 3 different ways:

  - intuitive / spatial: what does it do conceptually / visually in 3D

  - operational: how to compute the result, starting from
    (a) the coordinates of the operand(s)
    (b) when appropriate, also
    the angle between the operands, their lengths, etc

  - syntactic: how to write them down
    (a) on paper (mathematical notation)
    (b) in a programming language (Unity C# lib, Unreal C++ lib, GLSL…)

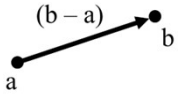
- also, familiarize how to manipulate equations, i.e. rules such as

(a) commutativity? associativity? (of each operation)
(b) distributivity? (between pairs of operations)
(c) inverse operation? identity element? absorbing element?

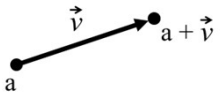
28

Point and vector algebra (summary 1/7)

- Difference:
  point – point = vector

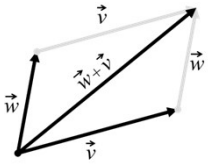


- Addition:
  point + vector = point

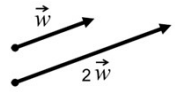


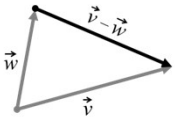
29

Point and vector algebra (summary 2/7)



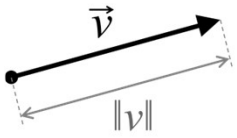
- Linear operations for vectors
  - addition (vector + vector = vector)
  - product with a scalar (scaling)
    (vector * scalar = vector)
  - therefore: interpolation
    $\text{mix}(\overrightarrow{v_0}, \overrightarrow{v_1}, t) = (1-t)\,\overrightarrow{v_0} + t\,\overrightarrow{v_1}$
  - therefore: opposite (flip verse)
    (how to: multiply by – 1)
  - therefore: difference
    (vector – vector = vector)





30

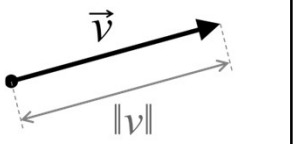
Point and vector algebra (summary 3/7)



- Norm (for vectors)
  - aka length / magnitude /
    Euclidean norm / 2-norm
  - distance between points:
    length of vector (**a** – **b**) = distance between **a** and **b**
  - Rules: triangle inequality:

31

Point and vector algebra (summary 4/7)



- Normalization
  - input: a vector. Result: a versor
  - how to: scale the vector by (1.0 / length)

32

Point and vector algebra (summary 5/7)

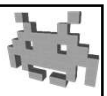
- Interpolate between pairs of <*something*> :
  - mix(point , point , *t*) → point
  - mix(vector , vector , *t*) → vector
  - mix(versor , versor , *t*) → versor
- *t* is a scalar «weight»
  - *t* = 0 → pick the first one
  - *t* = 1 → pick the second one
  - *t* ∈ (0,1) → get something in between, for example:
  - *t* = 0.5 → just average the two
  - *t* = 0.1 → use almost the first, with just a bit of the second in it
  - *t* < 0 or t > 1 → extrapolate
- Terminology: (in libraries, game engines…)
  - interpolate = mix = blend = lerp

a proper interpolation

specifically linear

33

Interpolation in general terms - notes



- Very used in Computer Graphics (e.g., rendering, animation)
- Terminology:
  - $a\,\mathbf{x} + b\,\mathbf{y}$: a linear combination of **x** and **y**
  - if $a+b=1$ and $a,b \in [0,1]$: a (linear) interpolation of **x** and **y**
  - if $a+b=1$ but $a,b \notin [0,1]$: a (linear) extrapolation of **x** and **y**
  - a , b : the used weights
  - $a + b = 1$: weights are a partition of unity
- Generalizes to > 2 objects ($a\,\mathbf{x} + b\,\mathbf{y} + c\,\mathbf{z}$)
- When interpolating 2 objects, we can just specify one weight t.
  - The other is given by difference. $a = t$, $b = 1-t$
  - The interpolation is often written in programming languages as `mix( x, y, t )` (or similar ways). Rember: t=0 → pick the first
- It's a general concept! All sorts of things can be interpolated
  - Intuition: interpolation = a mix between objects (e.g.: colors, statuses…)
  - Let's analyze case of Points, Vectors, Versors

34

How to interpolate between…

but easily generalizes to > 2

linear interpolation

- …two vectors $\mathbf{v}_0$ and $\mathbf{v}_1$:

$$(1 - t)\,\mathbf{v}_0 + (t)\,\mathbf{v}_1$$

only legal operations, with an easily defined geometric meaning (to-do: check)

- …two points $\mathbf{p}_0$ and $\mathbf{p}_1$:

$$\mathbf{p}_0 + t\,(\mathbf{p}_1 - \mathbf{p}_0)$$

which you may also want to write as:

$$(1 - t)\,\mathbf{p}_0 + (t)\,\mathbf{p}_1$$

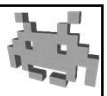
summing… two points ??

scaling… a point ??

we usually don't need any such operation, but it's equivalent, mathematically.

35

How to interpolate between…



But easily generalizes to > 2

Linear interpolation

- …two vectors $\mathbf{v}_0$ and $\mathbf{v}_1$:

$$(1 - t)\,\mathbf{v}_0 + (t)\,\mathbf{v}_1$$

- …two points $\mathbf{p}_0$ and $\mathbf{p}_1$:

$$\mathbf{p}_0 + t\,(\mathbf{p}_1 - \mathbf{p}_0)$$

- …two versors $\mathbf{d}_0$ and $\mathbf{d}_1$:

$$(1 - t)\,\mathbf{d}_0 + (t)\,\mathbf{d}_1$$

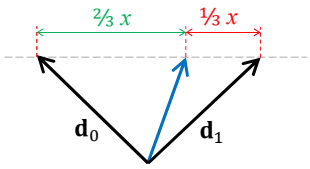
then renormalize the result (it's no longer unitary).

Or, use "spherical interpolation" (aka "slerp")…

36

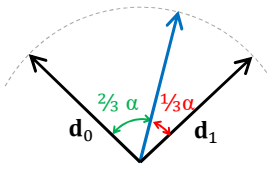
LERP vs SLERP (of versors)

Linear interpolation:



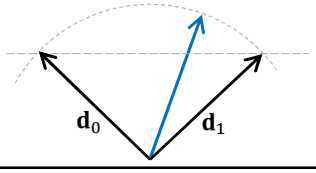
$\mathbf{d} = \mathrm{lerp}(\mathbf{d}_0, \mathbf{d}_1, ⅔)$

Spherical interpolation:



$\mathbf{d} = \mathrm{slerp}(\mathbf{d}_0, \mathbf{d}_1, ⅔)$

then, renormalize:

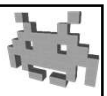


Not the same result!

- But, close enough
- Even closer when:
  $\mathbf{d}_0$, $\mathbf{d}_1$ similar OR t close to ½
- Is it worth the extra computation cost? 😉

37

The formulas



- LERP + normalization:

$$(1 - t)\,\mathbf{d_0} + t\,\mathbf{d_1}$$
then re-normalize

} aka "NLERP"

- or SLERP:

angle between $\mathbf{d}_0$ and $\mathbf{d}_1$

$$\frac{\sin((1-t)\,\alpha)}{\sin(\alpha)}\mathbf{d_0} + \frac{\sin(t\,\alpha)}{\sin(\alpha)}\mathbf{d_1}$$

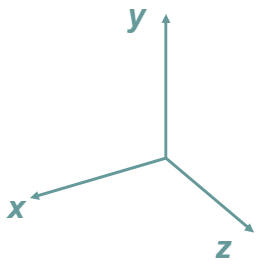
38

SLERP: notes

- Applicable to any versor (unit vector) including 2D, 3D, and quaternions (see later)
- SLERP can even be used on general vectors:
  - Compute magnitudes of vectors
  - Compute directions of vectors (divide by magnitude, i.e., normalize)
  - new direction = SLERP of the directions (unit vectors)
  - new magnitude = LERP of the magnitudes (scalars)
  - multiply new dir with new mag to get the final result

39

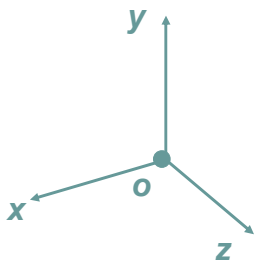
recap: Vector Base



- Axes: set of *n* lin. ind. vectors (***x***,***y***,***z***)
- Any vector ***v*** can be expressed in exactly 1 way as a linar combination of these vectors
- The weights are the coord of ***v*** in that base

40

recap: Reference Frame (or Space)

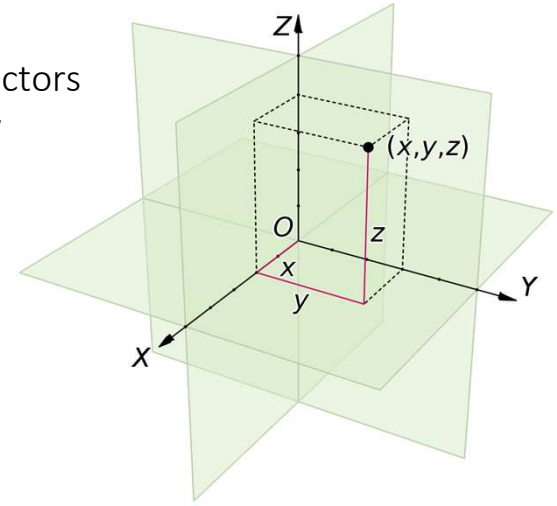


- *n* axes (vectors) (vector base)
  +
  1 origin (point)
- Any vector ***v*** :
  one linear comb. of the axes
- Any point ***p*** :
  origin + one linear comb. of axes

41

Recap: Orthonormal Frames Or Cartesian Frame

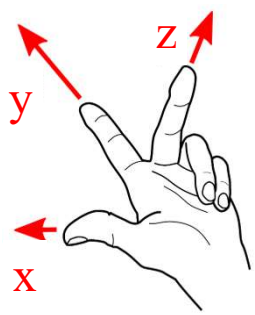
- Axes are unit vectors and reciprocally orthogonal



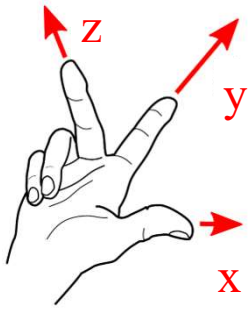
42

Recap: Handed-ness of a (Cartesian) frame

- They can be right- or left-handed



$x \times y = z$

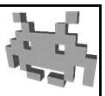


$x \times y = z$
regardless!

Use the same hand to *imagine* a cross product

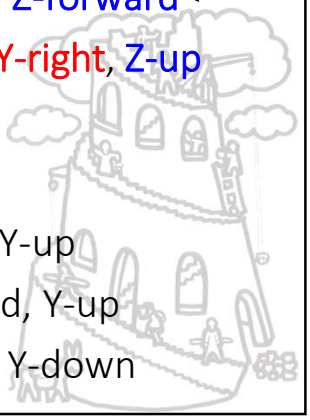
43

Semantics associated to X,Y,Z axes: still no standards for 3D games



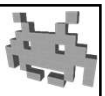
personal opinion:
the most standard one,
among
3D modellers too

- Unity: left-handed: X-right, Y-up, Z-forward
- Unreal: left-handed: X-forward, Y-right, Z-up
- 3ds-Max: right-handed, Z-up
- Blender: left-handed, Z-up
- most VR systems: right-handed, Y-up
- OpenGL: (clip space) right-handed, Y-up
- DirectX: (clip space) left-handed, Y-down



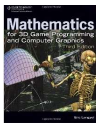
44

Point and vector algebra
Products: additional reading



To be continued!

Products between vectors and/or versors

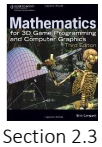


Section 2.2

- Dot product (or inner product)
  - Output: a scalar

NEXT LECTURE!

- Cross product (or vector product)
  - Output: a vector



Section 2.3

45