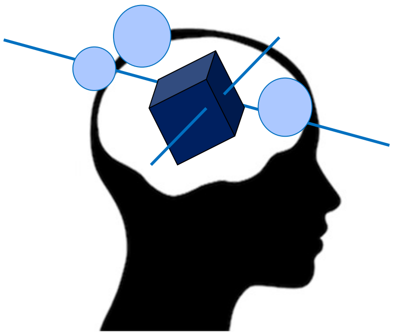
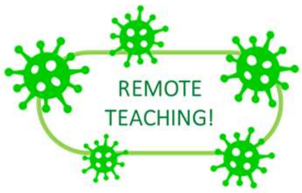
3D videogames

Points, Vectors, Versors (recap)



Marco Tarini

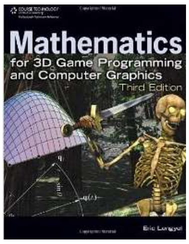


2

Course Plan

3

This lecture and the next two



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

7

Point, Vectors, Versors and Spatial Transformation

They are the basic data-type of 3D Games

- In the computation, for all modules
  - rendering engine
  - physics engine
  - AI
  - 3D sound
  - …
- In the data structures of all 3D Assets
  - See prev. lecture for the list

8

Point, Vectors, Versors

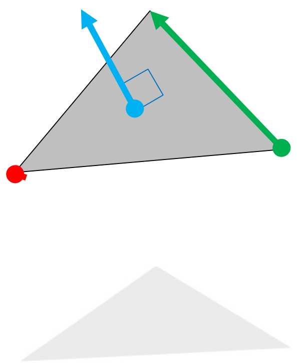
	represents:	example:	imagine it as...
Point	A position A location	Where a character is The center of a sphere	a small floating dot :-D
Vector	A displacement The difference between 2 points. The vector that connects them.	The velocity of a thrown knife The gravity acceleration How to reach the head of 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)*

9

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

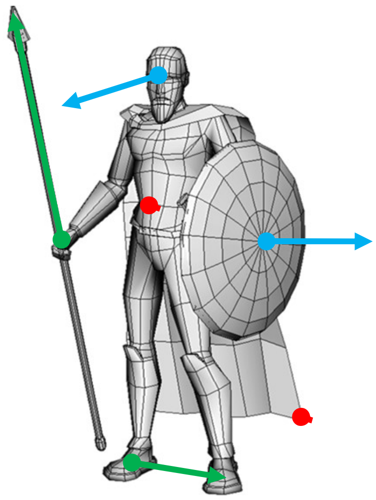


10

Points, Vectors, Versors
…in a character

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

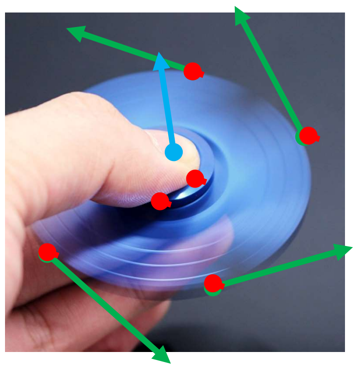


11

Points, Vectors, Versors
…in a spinner

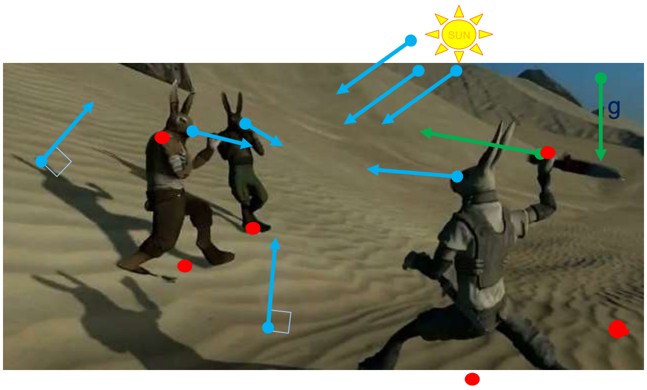
Examples of…

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



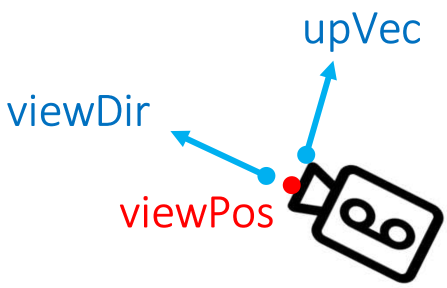
13

Points, Vectors, Versors
...in this screenshot



14

Stuff = Points + Vectors + Versors



Description of the camera

15

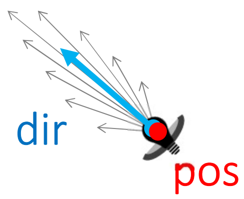
Stuff = Points + Vectors + Versors

description of
a (directional) sound emitter

description of
a (directional) microphone

16

Stuff = Points + Vectors + Versors



description of a spotlight

17

Points, Vectors, Versors: Internal representation

- *n*-tuple of scalar values (*n* is the dimension)
  - with $n = 3$ (rarely, 2 or 4)
  - they are the Cartesian coordinates of the point/vector
  - e.g.:

```
class Vector3 {
  // fields:
  float coords[3];

  // methods:
  …
}
```

or:

```
class Vector3 {
  // fields:
  float x, y, z;

  // methods:
  …
}
```

- note: the same structure is often used for points, vectors, and versors

19

Points, Vectors, Versors: Internal representation

- one class for points, vectors, and versors
- E.g. done by:

unity — class Vector3
https://docs.unity3d.com/ScriptReference/Vector3.html

UNREAL ENGINE — class FVector
http://api.unrealengine.com/INT/API/Runtime/Core/Math/FVector/

- (and also by: GLSL, HLSL, Eigen, GLM, …)

20

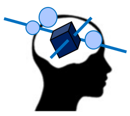
Caveat: one type, multiple semantics

- Many libraries/engines choose can opt to use the same data type for 3D points, 3D vectors, 3D versors, (plus, sometimes: colors, and more)
  - alternatively, a library can use different types, e.g. Vector, Point, Versor
- Still, they should not be considered the same thing
  - that's nothing new:
    likewise, we use the same scalar data types ("float", "doubles")
    with widely different semantics (e.g. "weight", "volume", "temperature"…).
- It is up to us to *operate* on them accordingly
  - e.g.: not ok to sum a *temperature* with a *surface*
  - e.g.: ok to divide a *weight* by a *volume* (and get a *specific weight*)
- which operation does make sense on points, vectors, versors?
  - that is, what is their *algebra* ?

21

Point, vector, versor *algebra*

- *Hint:* before going on, make sure to know / understand each of the following operation in 3 different ways:



  - **intuitive / spatial:** what does it do conceptually / visually

  - **algebraic / code:** how to compute the result, starting from
    (1) the coordinates of the operand(s)
    (2) and, additionally, (for products)
    the angle between the two operands, and their the lengths

  - **syntactic:** how to write them down
    (1) on paper (mathematical notation)
    (2) in a programming language (Unity C# lib, Unreal C++ lib, GLSL…)
- Refer to the CG course / the book

23

Point, vector, versor *algebra*

- *Hint:* also, familiarize with the way the operations behave, i.e. rules such as

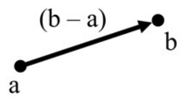
(1) commutativity? associativity? (of each operation)
(2) distributivity? (between pairs of operations)
(3) inverse operation? identity element? absorbing element?

- Refer to the CG course / the book

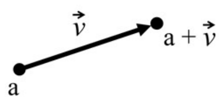
24

Point and vector algebra (summary 1/7)

- Difference:
  point – point = vector



- Addition:
  point + vector = point

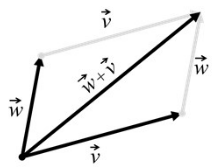


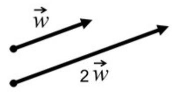
26

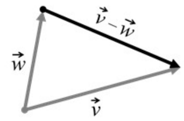
Marco Tarini
Unviersità degli studi di Milano

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)

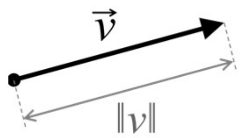






27

Point and vector algebra (summary 3/7)



For vectors:

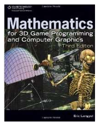
- Norm
  - aka length / magnitude / Euclidean norm
  - distance:
    length of vector (a – b) = distance between a and b
  - triangle inequality
- Normalization
  - Input: a vector. Result: a versor
  - how to: scale the vector by (1.0 / length)

28

Point and vector algebra (summary 4/7)

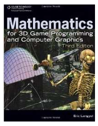
Products between vectors, or between versors

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



Section 2.2

- Cross product (or vector product)
  - Output: a vector (note: not a versor)



Section 2.3

(exercises in class)

29

Point and vector algebra (summary 5/7)

- Dot product, useful to:
  - test orthogonality (if orthogonal then res == 0)
    (between vectors, and/or versors alike)
  - sign tells: angle < or > 90°
    (between vectors, and/or versors alike)
  - versor dot vector: project vector along axis
  - versor dot versor: cosine of angle
  - versor dot versor: a similarity measure (in -1 +1)
  - any vector dot itself: its squared length

30

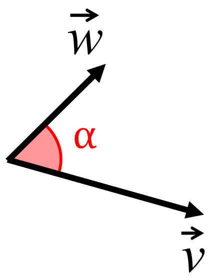
Point and vector algebra (summary 6/7)

MORE ABOUT THUS NEXT TIME

- Cross product, useful to:
  - find orthogonal vectors
  - therefore: construct orthonormal basis
  - collinearity test (if colinear then res == (0,0,0))
  - find (double) area of a 3D triangle
  - find normal of a 3D triangle (renormalize it)
  - norm of (versor cross versor): module of sin of angle
  - analogue in 2D: 2D vector “cross” 2D vector = scalar (how to: extend with Z=0, get Z of result)
  - 2D versor cross 2D versor: (signed) sin of angle

31

Products and angles



$$\vec{v} \cdot \vec{w} = \|\vec{v}\| \cdot \|\vec{w}\| \cdot \cos(\alpha)$$

$$\|\vec{v} \times \vec{w}\| = \|\vec{v}\| \cdot \|\vec{w}\| \cdot \sin(\alpha)$$

32

Point and vector algebra (summary 7/7)

- 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
  - $t \in (0,1)$ → get something in between, for example: ← a proper interpolation
  - $t = 0.5$ → 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 ← specifically linear

33

Interpolation in general - notes

- Very used in Computer Graphics (e.g. rendering, animation)
- Terminology:
  - $a\,\mathrm{x} + b\,\mathrm{y}$: 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 weights $a + b = 1$: weights are a partition of unity
- Generalizes to > 2 objects ($a\,\mathrm{x} + b\,\mathrm{y} + c\,\mathrm{z}$)
- In interpolations of 2, we can just give one weight t.
  - The other is is given by difference. $a = t$, $b = 1-t$
- General! All sort of objects can be interpolated
  - Intuition: interpolation = a mix between objects
  - 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 $\mathbf{v}_1$:

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

Multiplying a point with a scalar? Summing two points? Are these operations even legal?

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

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

  which is just a shortcut to express:

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

Just legal operations (to-do: check)

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

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

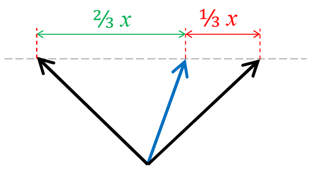
  then renormalize the result (it's no longer unitary).

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

35

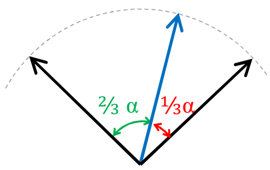
LERP vs SLERP (of versors)

Linear interpolation:



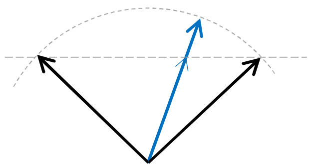
$\mathbf{d} = \mathrm{lerp}(\mathbf{d}_0, \mathbf{d}_1, ⅔)$

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? 🤔

36

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 $\mathbf{d}_1$

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

37

SLERP: notes

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

38

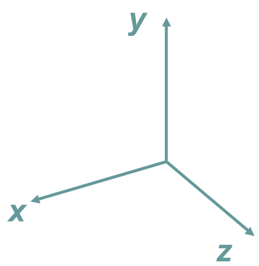
Note: Generalization to N - Dimension

- Everything seen in this lecture generalizes in 2D (for 2D games), or even in N>3 dimensions
- Exception: the cross product is only defined in 3D
  - But in 2D, the problem of finding a vector orthogonal to one (just one!) given vector is trivial: "swap coordinates, flip one* sign" (x,y) orthogonal to (-y,x)

    *: which coordinate you flip determines if you rotate 90° clockwise or counterclockwise: try!

39

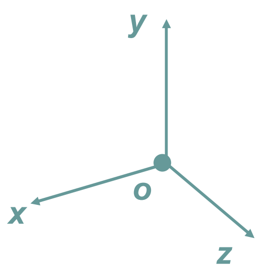
recap: Vector Base



- 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)

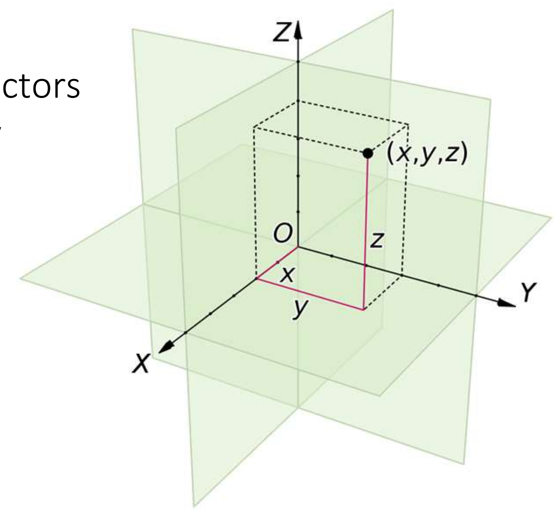


- *n* axes (vectors) + 1 origin (point) ← (vector base)
- Any vector ***v*** : one linear comb. of the axes
- Any point ***p*** : origin + one linear comb. of axes

41

Recap: Orthonormal Frames Or Cartesian Frame

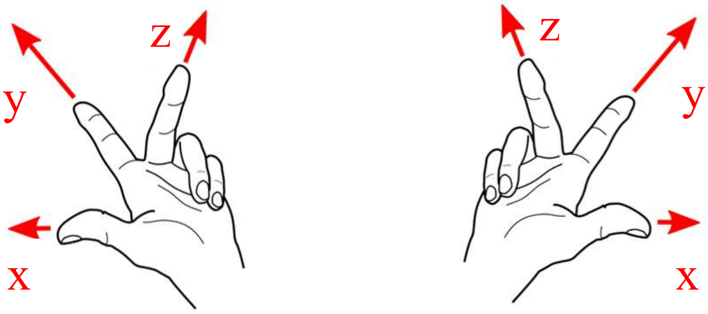
- Axes are unit vectors and reciprocally orthogonal



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

Still no standards in 3D games

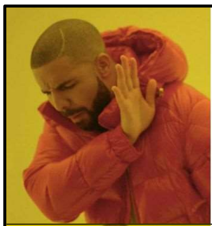
personal opinion:
the most standard one,
among
3D modellers too

- 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

Pro-tip: try making your code assumption free!

E.g.: to move a pos 2.5 units "to the right":



```
Vector3 pos = new Vector3 ( … );

pos.x = pos.x + 2.5;   // maybe ??
pos.y = pos.y + 2.5;   // hmm…??
```

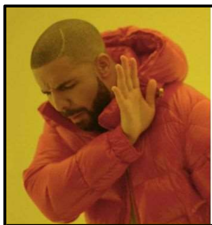
```
Vector3 pos = new Vector3 ( … );

pos += Vector3.right * 2.5;
```

46

Pro-tip: try making your code assumption free!

E.g.: to move a pos 2.5 units "to the right":



```
FVector pos = FVector( … );

pos.X += 2.5f;   // maybe ??
pos.Y += 2.5f;   // hmm…??
```

```
FVector pos ( … );

pos += FVector::RightVector * 2.5f;
```

47