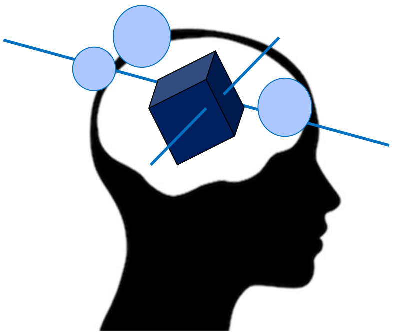
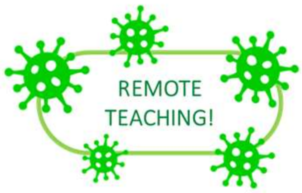
3D videogames

Points, Vectors, Versors (recap)

Marco Tarini





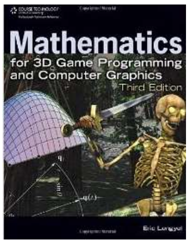
2

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. 8: Game 3D Animations
lec. 9: Game 3D Audio
lec. 10: Networking for 3D Games
lec. 11: Artificial Intelligence for 3D Games
lec. 12: Game 3D Rendering Techniques

4

Suggested reading



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

6

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

7

Point, Vectors, Versors

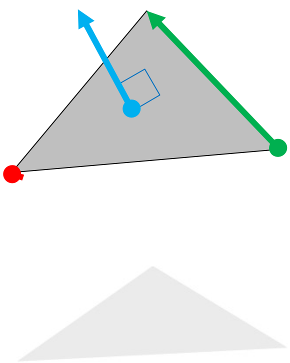
	represents:	example:	imagine it as...
Point	A position A location	Where a character is The center of a sphere	a small floating dot :-D
Vector	A displacement The difference between 2 points. The vector that connects them.	The velocity of a thrown knife The gravity acceleration How to reach the head of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

8

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

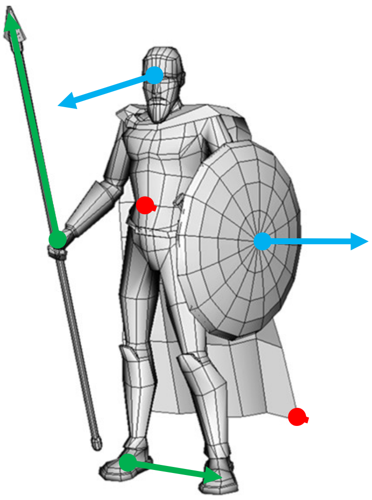


9

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



10

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

18

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, ...)

19

Caveat: one type, multiple semantics

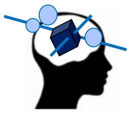
- Many libraries/engines choose can opt to use the same data type for 3D points, 3D vectors, 3D versors, (plus, sometimes: colors, and more)
  - alternatively, a library can use different types, e.g. Vector, Point, Versor
- Still, they should not be considered the same thing
  - that's nothing new: likewise, 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* ?

20

Point, vector, versor *algebra*

Refer to
the CG course
and the book

- *Hint:* before going on, make sure to 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...)

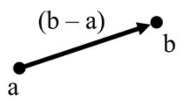
- And, to familiarize with their **rules** such as

(1) invariance (associativity?, commutativity?)
(2) distributivity? (between operations)
(3) inverse operation? itentity element? absorbing element?

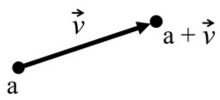
24

Point and vector algebra (summary 1/7)

- Difference:
  point – point = vector



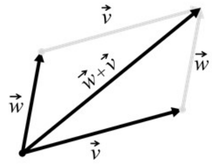
- Addition:
  point + vector = point



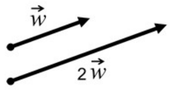
25

Point and vector algebra (summary 2/7)

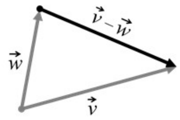
- Linear operations for vectors
  - addition (vector + vector = vector)



  - product with a scalar (scaling)
    (vector * scalar = vector)



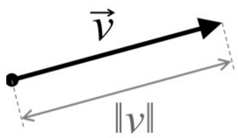
  - therefore: interpolation
    $\text{mix}(\vec{v_0}, \vec{v_1}, t) = (1 - t)\,\vec{v_0} + t\,\vec{v_1}$
  - therefore: opposite (flip verse)
    (how to: multiply by – 1)
  - therefore: difference
    (vector – vector = vector)



26

Point and vector algebra (summary 3/7)

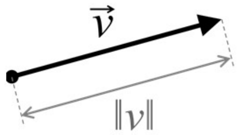
- Norm (for vectors)



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

27

Point and vector algebra (summary 4/7)



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

28

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: ← 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

29

Interpolation in general - notes

- Very used in Computer Graphics (e.g. rendering, animation)
- Terminology:
  - a x + b 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 x + b y + c 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

30

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")…

NEXT LECTURE

31