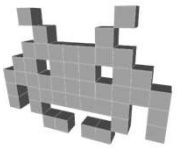
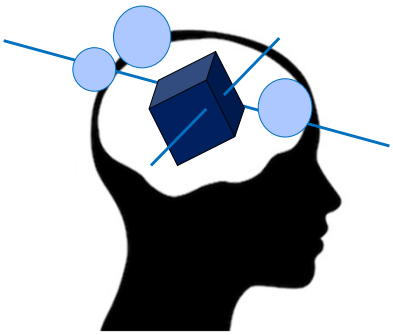
3D VideoGames - UniMi

Points, Vectors, Versors (recap)

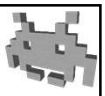




Marco Tarini

3

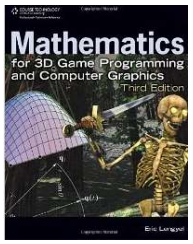
Course Plan



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

5

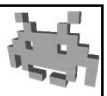
Point, Vectors, Versors and Spatial Transformation

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

6

Point, Vectors, Versors



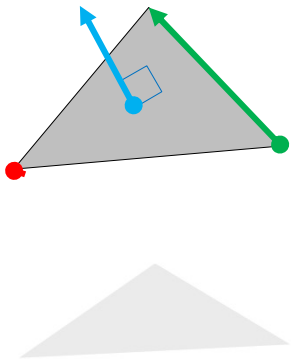
	represents:	example:	imagine it as...
Point	A position A location	Where a character is The center of a sphere	a small floating dot :-D
Vector	A displacement The difference between 2 points. The vector that connects them.	The velocity of a thrown knife The gravity acceleration How to reach the head of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

7

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

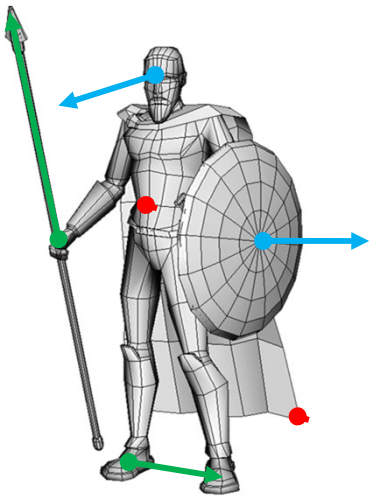
Examples of...

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



8

Points, Vectors, Versors …in a character

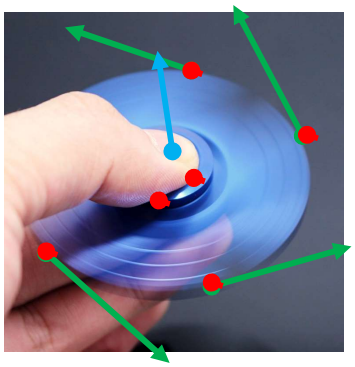


Examples of…

- 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

9

Points, Vectors, Versors …in a spinner

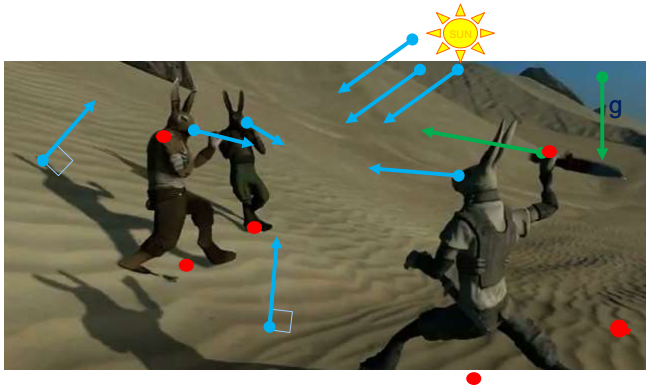


Examples of…

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

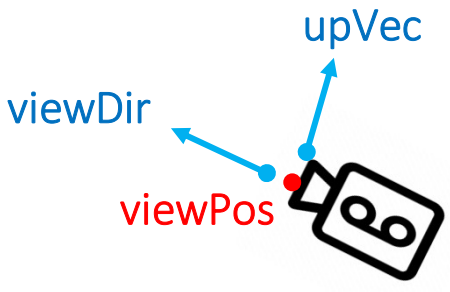
12

Points, Vectors, Versors
...in this screenshot



14

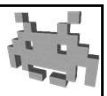
Stuff = Points + Vectors + Versors



Description of the camera

15

Stuff = Points + Vectors + Versors

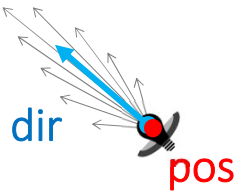


description of
a (directional) sound emitter

description of
a (directional) microphone

16

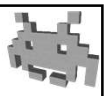
Stuff = Points + Vectors + Versors



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

- same class for points, vectors, and versors
- this is done in many libs & languages, e.g.:

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

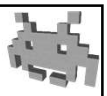
class FVector
http://api.unrealengine.com/INT/API/Runtime/Core/Math/FVector/

- and also: GLSL, HLSL, GLM, Eigen, VcgLib , three.js, …

vec3 (GLSL, HLSL, GLM) — shader languages
Vector3d (Eigen) — C++ libraries
Point3d (VcgLib) — C++ libraries
Vector3 (three.js) — JavaScript library

20

Caveat (about coding): one type, multiple semantics

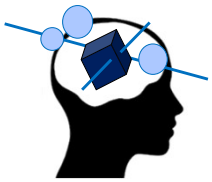


- Libraries/engines/languages can opt to use the same data type for 3D points, 3D vectors, 3D versors, (plus, sometimes: colors, and more)
  - alternatively, a library can use different types, e.g. Vector, Point, Versor
- Still, they should not be considered the same thing
  - that's nothing new: likewise,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

21

Point, vector, versor *algebra*

make sure to understand each operation in 3 different ways



intuitive
know how to imagine it spatially, in 3D

operational
know how to compute from data

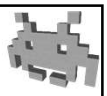
syntactic
know how to write down, (in paper or code)

& rules:
know how to manipulate equations

22

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 / intuitively

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…)

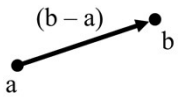
- And, to familiarize with their **rules** such as

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

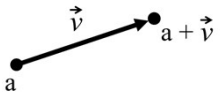
25

Point and vector algebra (summary 1/7)

- Difference:
point – point = vector

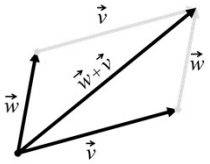


- Addition:
point + vector = point

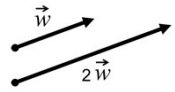


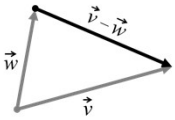
26

Point and vector algebra (summary 2/7)



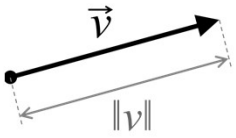
- Linear operations for vectors
  - addition (vector + vector = vector)
  - product with a scalar (scaling)
    (vector * scalar = vector)
  - therefore: interpolation
    $\text{mix}(\overrightarrow{v_0}, \overrightarrow{v_1}, t) = (1-t)\,\overrightarrow{v_0} + t\,\overrightarrow{v_1}$
  - therefore: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





27

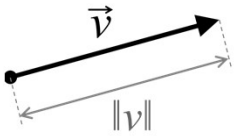
Point and vector algebra (summary 3/7)



- 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:

28

Point and vector algebra (summary 4/7)

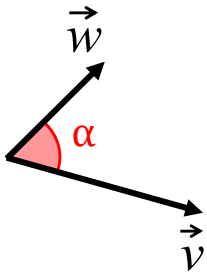


- Normalization
  - Input: a vector. Result: a versor
  - how to: scale the vector by (1.0 / length)

29

Point and vector algebra (summary 5/7)

- Dot product (or inner product)



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

30

Point and vector algebra (summary 5/7)

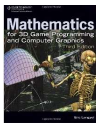
- Dot product (or inner product)
  - Output: a scalar
  - Alternative notations:

$$\vec{v} \cdot \vec{w}$$

$$< \vec{v}, \vec{w} >$$

$$(v^{\mathrm{T}} \vec{w})$$

See...



Section 2.2

31

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 us: is the angle < or > 90° (works between vectors, and/or versors)
  - 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

32