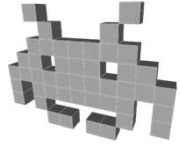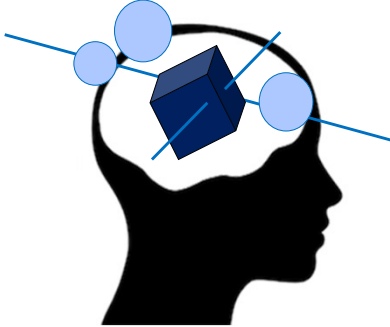3D VideoGames - UniMi

Points, Vectors, Versors:
(final notes on programming)

Marco Tarini

48

Points, Vectors, Versors:
Internal representation

- *n*-tuple of scalar values (*n* is the dimension)
  - for us (usually): *n* = 3 (at times, 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 to represent points, vectors, and versors

50

Caveat (about coding):
one type, multiple semantics

- Many libraries/engines/languages 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 <u>not</u> 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?
  - the ones we have seen in their *algebra* !

51

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/

| vec3 | Vector3d | Point3d | Vector3 |

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

shader languages   C++ libraries   JavaScript library

52

Classes for Points / Vectors / Versors
A few examples in C++ libraries

- **GLM** library (*Graphics*) : class `vec3`
- **UNREAL ENGINE** library (*Videogames*) : class `VectorF`
- **Eigen lib** (*Linear Algebra*) : class `Vector3d`
- **VCG-Lib** (*Geometry processing*) : class `Point3f`
- **Point Cloud Lib** (*Geometry Processing*) : class `ON_3dVector`
- **openMesh** for (*Geometry processing*) : class `VectorT`
- **cgall** for (*Geometry Processing*) : class `Vector3`
- **CinoLib** (*Geometry Processing*) : class `vec3d`
- **OpenCV** for (*Computer Vision*) : class `Point3f`
- **bullet** for (*Physical Simulation*) : class `btVector3`
- **ODE** for (*Physical Simulation*) : class `dVector3`

53

Classes for Points / Vectors / Versors:
Other examples in C++ like languages

- **GLSL** shader language from Chronos : type `vec3`
- **HLSL** shader language from DirectX : type `float3`
- **unity** , C# *(videogames)*: class `Vector3`
- **three.js** , JavaScript *(graphics)* : class `Vector3`

54

Programming with vector algebra: the code looks like the expressions

- Concept ("on paper"):

$$\mathbf{p} = \mathbf{p} + k\,\hat{\mathbf{d}}$$

- Code:
  - Data types:

```
Point3D dragonPos = …;
Versor3D dir = …;
float k = …;
```
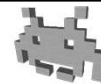
  - Beginner's style code:

```
dragonPos.x = dragonPos.x + dir.x * k ;
dragonPos.y = dragonPos.y + dir.y * k ;
dragonPos.z = dragonPos.z + dir.z * k ;
```

  - What you should do:

```
dragonPos.add( dir.scale(k) );
```

or (depending on the language)

```
dragonPos += dir * k ;
```

55

Semantics associated to X,Y,Z: 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
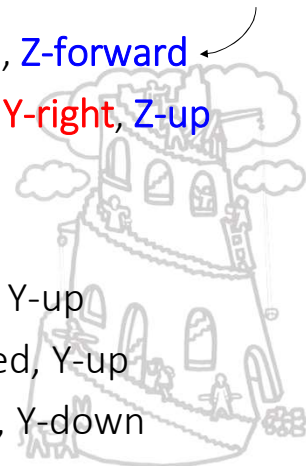
56

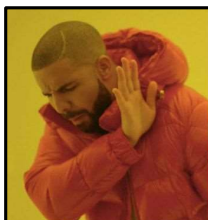## Pro-tip: try making your code assumption free!

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

 unity

```
Vector3 pos = new Vector3 ( … );

pos.x = pos.x + 2.5;   // maybe ??
pos.y = pos.y + 2.5;   // hmm…??
```
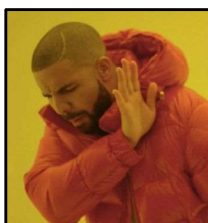
```
Vector3 pos = new Vector3 ( … );

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

58

Pro-tip: try making your code assumption free!

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

 UNREAL ENGINE

```
FVector pos = FVector( … );

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

```
FVector pos ( … );

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

59