Point, vector, versor *algebra*

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

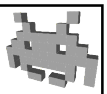
- **intuitive / spatial:** what does it do conceptually

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

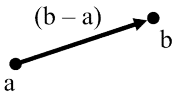
- **syntactic:** how to write them down
  (1) on paper (math-notation)
  (2) in a programming language (Unity C# lib, Unreal C++ lib, GLSL…)

- Refer to the CG course / the book

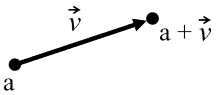
Point and vector algebra 1/7 (summary)



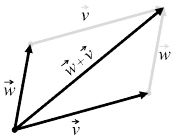
- Difference:
  point – point = vector



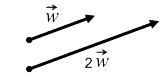
- Addition:
  point + vector = point

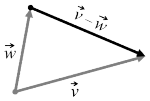


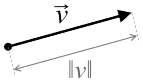
Point and vector algebra 2/7 (summary)



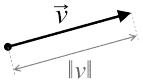
- Linear operations for vectors
  - addition (between vectors)
  - product with a scalar
  - therefore: interpolation (between vectors)
  - opposite (flip verse)
  - therefore: difference (between vectors)
- Norm
  - aka length / magnitude / Euclidean norm
- Normalization
  - Input: a vector. Result: a versor







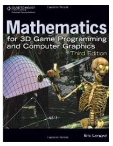
Point and vector algebra 3/7 (summary)



- 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
  - scale the vector by 1.0 / its length

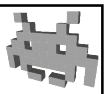
Point and vector algebra 4/7 (summary)

- Dot product (or innter product)
  - See Chapter 2.2
- Cross product (or vector product)
  - See Chapter 2.3



(exercises in class)

Point and vector algebra 5/7 (summary)

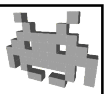


- Dot product, good for:
  - orthogonality test (if res == 0…)
    (between vectors, and/or versors)
  - sign: angle < or > 90
    (between vectors, and/or versors)
  - versor dot vector: project over versor
  - find cosine of angle (between versors)
  - get squared length (vector dot itself)
  - similarity measure (between versors), in -1 +1

Point and vector algebra 6/7 (summary)

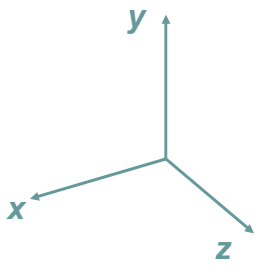
- Cross product, good for:
  - find orthogonal vectors
  - construct orthonormal basis
  - among 2D versors: find signed sin of angle
  - find (double) area of 3D triangle
  - find normal of 3D triangle
  - collinearity test (if res = (0,0,0) …)

Point and vector algebra 7/7 (summary)



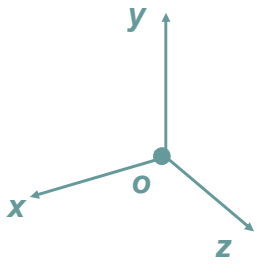
- Interpolate (linearly) beween pairs of
  - points, vectors, versor (each with its own)
  - Versors how-to: (crude version)
    do the usual linear combination, then renormalize
- mix(point , point , scalar-weight) → point
  - and likewise for other types
  - weight == 0.5 → you are just averaging
- Terminlogy: (libraries, game engines…)
  - interpolate = mix = blend = lerp

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

Recap: reference frame (or space)

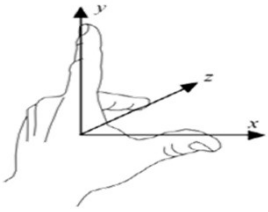


- *n* axes (vectors) (vector base)
  +
  1 origin (point)
- Any vector ***v*** : one linear comb of the axes
- Any point ***p*** : origine + one linear comb. of axes

Recap: Handed-ness of a frame

- Orthonormal frames: axes are unit vectors and reciprocally orthogonal.
- Such space can be right- or left-handed

$x \times y = z$



$x \times y = z$

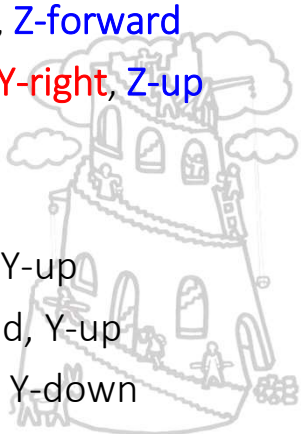
regardless!

Remember to use the same hand to *imagine* a cross product

Still no standards in 3D games

personal opinion: most standard, among 3D modellers too

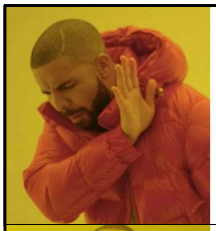
- Unity: left-handed: X-right, Y-up, Z-forward
- Unreal: left-handed: X-forward, Y-right, Z-up
- 3DMax: right-handed, Z-up
- Blender: left-handed, Z-up
- most VR systems: right-handed, Y-up
- OpenGL: (clip space) right-handed, Y-up
- DirectX: (clip space) left-handed, Y-down



Pro-tip: try making your code assumption free!

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

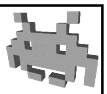
unity



```
Vector3 pos = new Vector3 ( … );

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

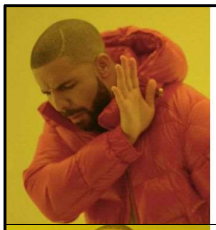
```
Vector3 pos = new Vector3 ( … );

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

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;
```