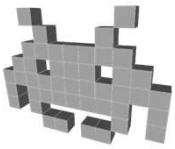
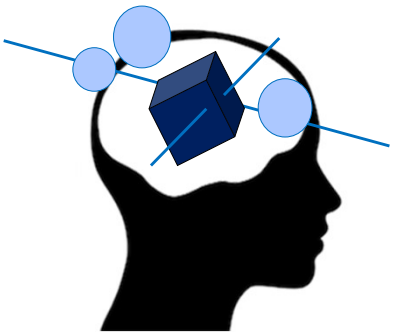
3D VideoGames - UniMi

Points, Vectors, Versors (recap)

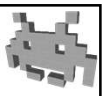




Marco Tarini

3

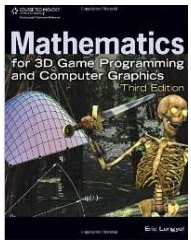
Course Plan



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. 9: Game **Materials**
lec. 8: Game **3D Animations**
lec. 10: **Networking** for 3D Games
lec. 11: **3D Audio** for 3D Games
lec. 12: **Rendering Techniques** for 3D Games
lec. 13: **Artificial Intelligence** for 3D Games

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

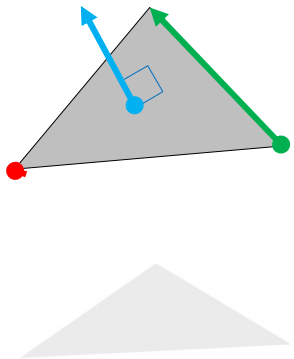
	represents:	example:	imagine it as…
Point	A position A location	Where a character is The center of a sphere	a small floating dot :-D
Vector	A displacement The difference between 2 points. The vector that connects them.	The velocity of a thrown knife The gravity acceleration How to reach the head of a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

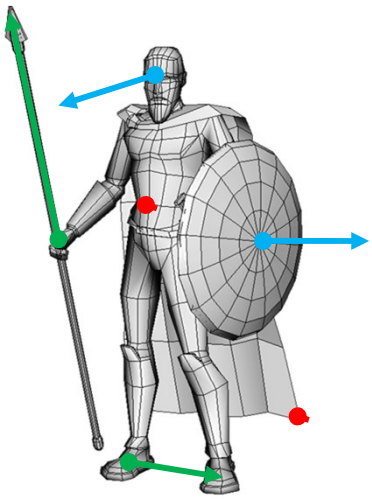


8

Points, Vectors, Versors ...in a character model

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

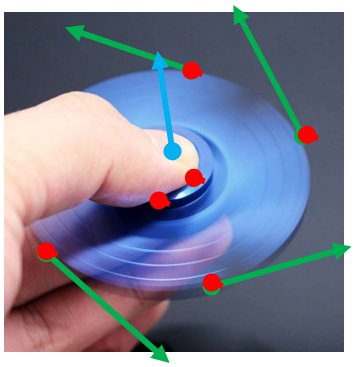


9

Points, Vectors, Versors ...in a spinner

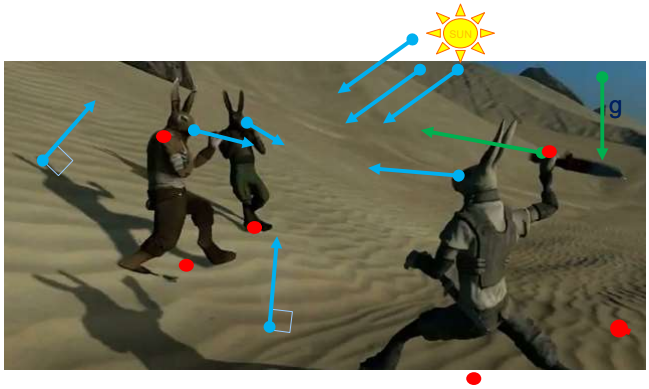
Examples of...

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



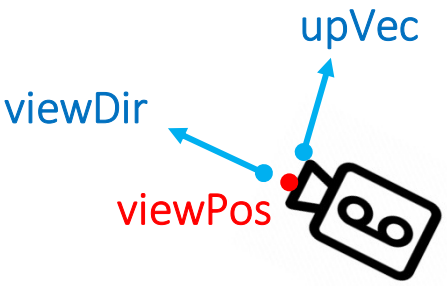
12

Points, Vectors, Versors
...in this screenshot



14

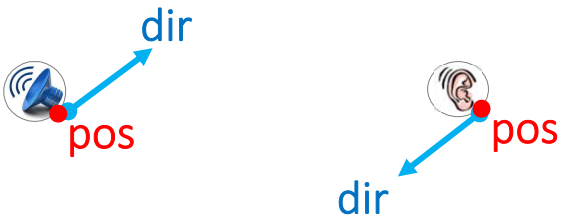
Stuff = Points + Vectors + Versors



Description of the camera

15

Stuff = Points + Vectors + Versors

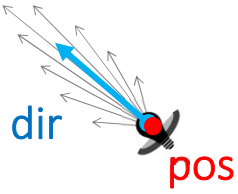


description of
a (directional) sound emitter

description of
a (directional) microphone

16

Stuff = Points + Vectors + Versors



description of a spotlight

17

Points, Vectors, Versors: Internal representation

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

  // methods:
  …
}
```

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

  // methods:
  …
}
```

18

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; Vector3d: Eigen; Point3d: VcgLib; Vector3: three.js

shader languages: GLSL, HLSL; C++ libraries: GLM, Eigen, VcgLib; JavaScript library: three.js

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

The *algebra* of points, vectors, versors (and scalars)

- make sure you understand each of 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…)

23

The *algebra* of points, vectors, versors (and scalars)

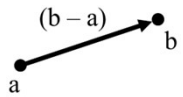
- also, familiarize with the way the operations behave, i.e. rules such as

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

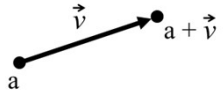
24

Point and vector algebra (summary 1/7)

- Difference:
  point – point = vector



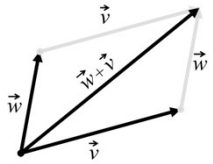
- Addition:
  point + vector = point

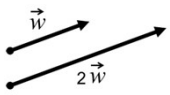


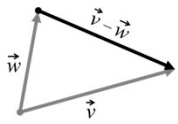
26

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

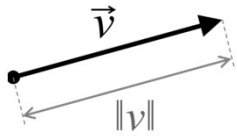






27

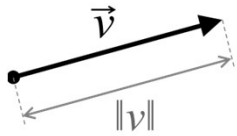
Point and vector algebra (summary 3/7)



- 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)

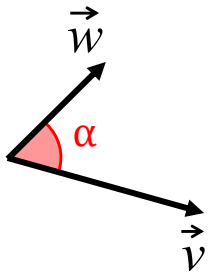


- Normalization
  - Input: a vector. Result: a versor
  - how to: scale the vector by (1.0 / length)

29

Point and vector algebra (summary 5/7)

- Dot product (or inner product)



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

30

Point and vector algebra (summary 5/7)

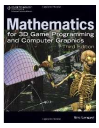
- Dot product (or inner product)
  - Output: a scalar
  - Alternative notations:

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

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

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

See…



Section 2.2

31

Point and vector algebra (summary 5/7)

- Dot product, useful to:
  - dot is zero: vectors are orthogobnal
    (works between vectors, and/or versors)
  - positive dot: acute angle
    negative dot: obtuse angle
    (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

Point and vector algebra (summary 6/7)

- 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:
  - *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

a proper interpolation

specifically linear

33

Interpolation in general - notes

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

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

Only legal operations with an easily defined geometric meaning (to-do: check)

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

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

which you may also want to write as:

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

Summing… two points ??

Scaling… a point ??

We usually don't need any such operation.
But it's equivalent, mathematically.

35

How to interpolate between…

But easily generalizes to > 2

Linear interpolation

- …two vectors $\mathbf{v}_0$ and $\mathbf{v}_1$:

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

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

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

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

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

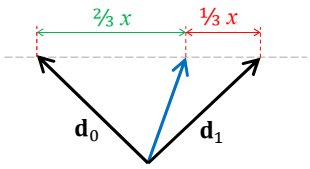
then renormalize the result (it's no longer unitary).

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

36

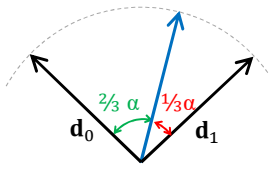
LERP vs SLERP (of versors)

Linear interpolation:



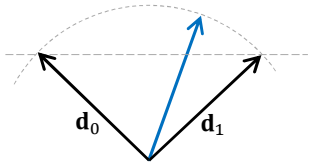
$\mathbf{d} = \mathrm{lerp}(\mathbf{d}_0, \mathbf{d}_1, ⅔)$

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? 🤔

37

The formulas

- LERP + normalization:

$$\left.\begin{array}{c}(1-t)\,\mathbf{d_0} + t\,\mathbf{d_1} \\ \text{then re-normalize}\end{array}\right\}\text{aka “NLERP”}$$

- or SLERP:

angle between $\mathbf{d}_0$ and $\mathbf{d}_1$

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

38

SLERP: notes

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

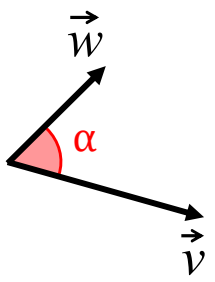
39

Point and vector algebra (summary 7/7)

- 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 triangle (in 3D)
  - find normal of a triangle in 3D (remember to 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 × 2D versor: (signed) sin of angle

40

Products and angles



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

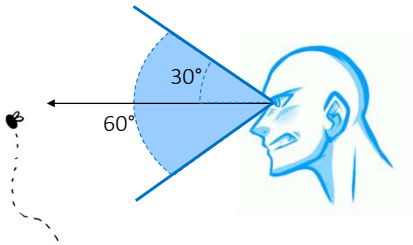
$$\|\vec{v} \times \vec{w}\| = \|\vec{v}\| \cdot \|\vec{w}\| \cdot |\sin(\alpha)|$$

41

3D videogames

Points, Vectors, Versors: mini task and exercises Part I

Marco Tarini



55

Points, Vectors, Versors: mini problems

- The following are examples of spatial problem problems that need to be solved in 3D games
  - They can be solved simply using point/vector/versor algebra
  - Many game engines libraries implement functions for many of them

For some of them, the solution will be given in full here. In other, only a trace of the solution is given

- General schema for finding a solution:
  - identify input and output (and their types)
  - write the equations driven by your intuitive/spatial understanding of the operations
  - identify the unknowns
  - manipulate the equations according to the rules
  - extract the unknowns
  - everything is ready to code it!

56

Point to point distance (trivial)

"When the player in position p is closer than k to a powerup in pos q, then the powerup is collected"

- Data: p, q points, k scalar
- Test: $\|\mathrm{p}-\mathrm{q}\| < k$
- Optimize vers: $\|\mathrm{p}-\mathrm{q}\|^2 < k^2$
- Pseudo-code example:

```
vec3 p,q;
scalar k;
if ( dot(p-q,p-q) < k*k ) then /*collect*/
```

58

Ray-Plane intersection Ver0

"I shoot a laser from p in direction $\hat{\mathrm{d}}$ toward a plane which contains points q and has normal $\hat{\mathrm{n}}$.
Which point q do I hit?"

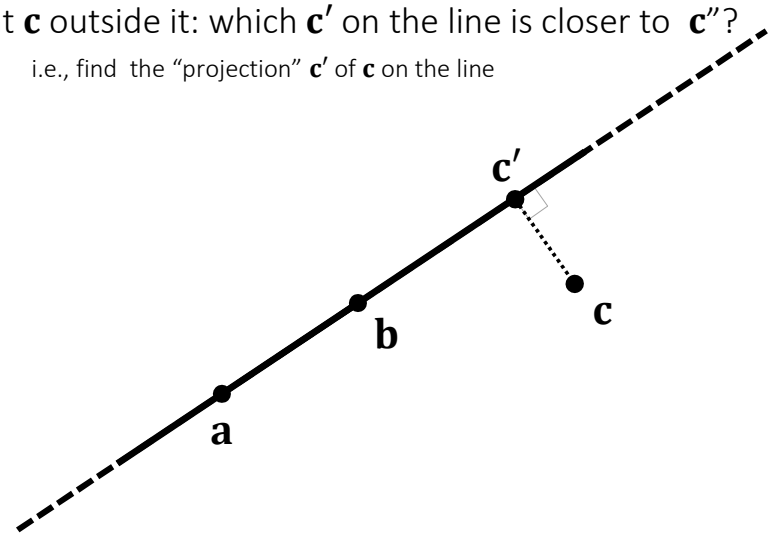
- Trace:
  - Define q as a point on the laser (see Ray-Sphere inters.)
  - Define q as a point on the plane
    (hint: the vector connecting it to any other point on the plane is orthogonal to $\vec{\mathrm{n}}$)
  - Combine the two equations into one
  - Extract the only incognita

59

Projection of a point on a line

"Consider a line passing through points $\mathbf{a}$ and $\mathbf{b}$ and a point $\mathbf{c}$ outside it: which $\mathbf{c}'$ on the line is closer to $\mathbf{c}$"?

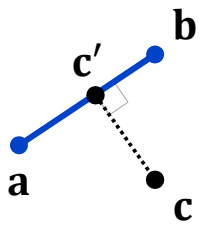
i.e., find the "projection" $\mathbf{c}'$ of $\mathbf{c}$ on the line



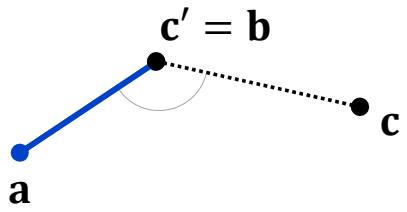
60

Projection of a point on a segment

"Which $\mathbf{c}'$ point on a segment connecting point $\mathbf{a}$ and $\mathbf{b}$ is closer to a third point $\mathbf{c}$"?



case 1



case 2

61

Line-Line "intersection"

"Given two 3D lines, find the two points on both lines that are as close as possible to each other"

(they are the same point, if the lines intersect!)

- Input: a point on line "A" $\mathbf{p}_A$ and its direction $\hat{\mathbf{d}}_A$
  a point on line "B" $\mathbf{p}_B$ and its direction $\hat{\mathbf{d}}_B$
- Output: two points $\mathbf{q}_A$ and $\mathbf{q}_B$

62

Ray-Plane intersection Ver1

"I shoot a laser from $\mathbf{p}$ in direction $\hat{\mathbf{d}}$ toward a plane which contains points $\mathbf{a}$ $\mathbf{b}$ $\mathbf{c}$. Which point $\mathbf{q}$ do I hit?"

- Hypotheses: $\mathbf{a}$ $\mathbf{b}$ $\mathbf{c}$ are not colinear (not on a line)
- Trace:
  - Find vector $\vec{\mathbf{n}}$ orthogonal to plane, use cross product (question for later: are magnitude and verse important?)
  - Define $\mathbf{q}$ as a point on the laser (see Ray-Sphere inters.)
  - Define $\mathbf{q}$ as a point on the plane (hint: the vector connecting it to any other point on the plane is orthogonal to $\vec{\mathbf{n}}$)
  - Combine the two equations into one
  - Extract the only incognita

63

Sphere-sphere intersection (trivial)

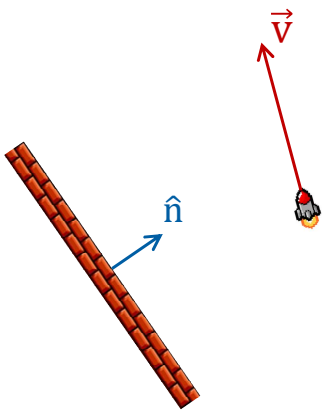
"Given two spheres with center in $\mathbf{c}_0$ and $\mathbf{c}_1$ and radii r_0 and r_1: do they intersect? Do they touch?"

- Hint:
  - remember that working with *squared norms* is more efficient (and more accurate) than working with vector norms

64

The missile and the wall (trivial)

"A missile is moving at constant velocity $\vec{\mathbf{v}}$ (meter per sec), in the general proximity of a large (infinite) wall with normal $\hat{\mathbf{n}}$. After some time t (sec), how much closer to (or farther from) the wall is it?"



65

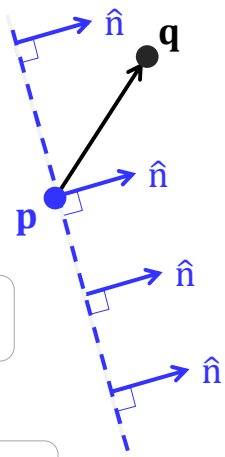
Plane VS Point test

- Input: a point $\mathbf{q}$
  and a plane given by:
  - its normal: $\hat{\mathrm{n}}$
  - a point on it at random: $\mathbf{p}$
- *Q*: on which side of the plane is $\mathbf{q}$?
- *A*: it's the sign of

$$\hat{\mathrm{n}} \cdot (\mathbf{q} - \mathbf{p}) =$$
$$\hat{\mathrm{n}} \cdot \mathbf{q} - \vec{n} \cdot \mathbf{p} =$$
$$\hat{\mathrm{n}} \cdot \mathbf{q} + k =$$
$$(n_x, n_y, n_z, k) \cdot (q_x, q_y, q_z, 1)$$

$k = -\vec{n} \cdot \mathbf{p}$
(minus distance of plane from origin)

a 4D vector representing the plane:
a more convenient representation for a plane

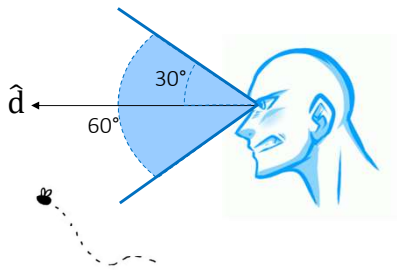


66

Vision cones

"A guard has eyes in position q and looks in direction $\hat{\mathrm{d}}$. Does it spot a fly in position p , if his cone of vision is 60° wide?"

- Hypotheses: no occlusions



- Trace:
  - For angles α, β in 0..90°: $\alpha < \beta \leftrightarrow \cos(\alpha) > \cos(\beta)$
  - Find cosine of angle between view direction and the vector connecting q to p
  - Determine if this cosine is > $\cos(60°/2)$

67