

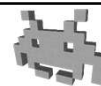
Course Plan



- 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** ●●

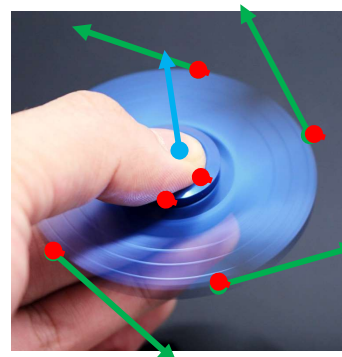
32

Points, Vectors, Versors ...in a spinner



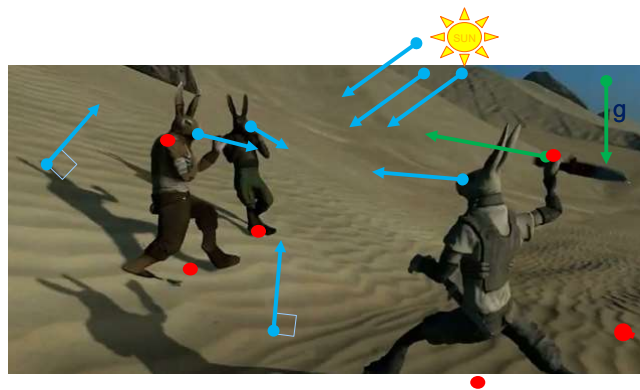
Examples of...

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



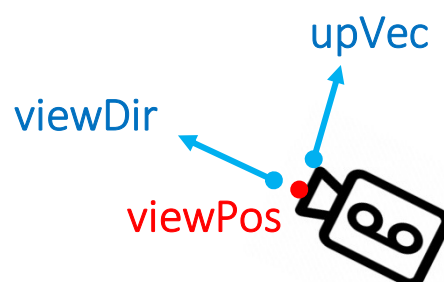
36

Points, Vectors, Versors ...in this screenshot



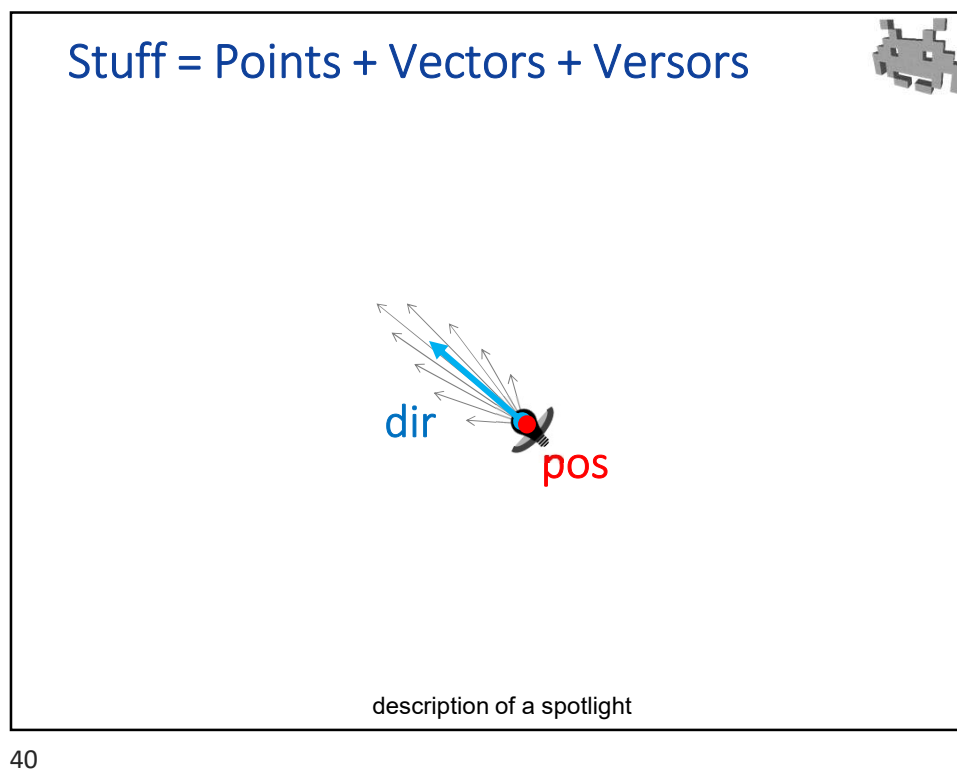
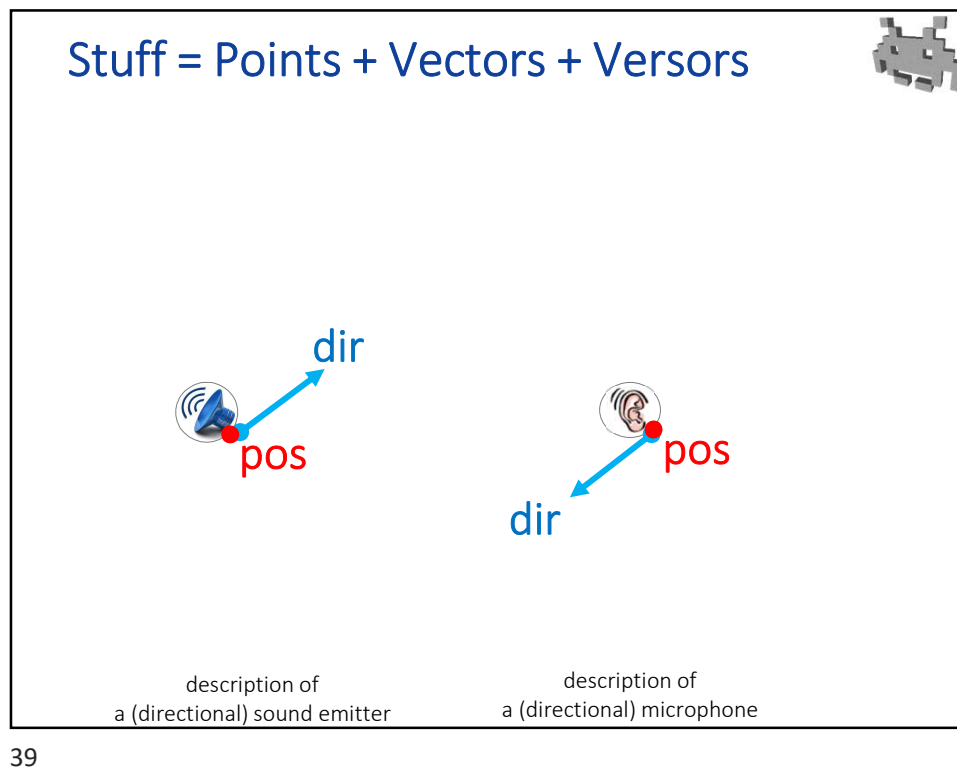
37

Stuff = Points + Vectors + Versors



Description of the camera

38



How to interpolate between...

But easily generalizes to > 2

- ...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 :

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

which is just a shortcut to express:

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

Linear interpolation

Multiplying a point with a scalar?
Summing two points?
Are these operations even legal?

Just legal operations (to-do: check)

NEXT LECTURE

41

LERP vs SLERP (of versors)

Linear interpolation:

$\mathbf{d} = \text{lerp}(\mathbf{d}_0, \mathbf{d}_1, \frac{2}{3})$

Then, renormalize:

Spherical interpolation:

$\mathbf{d} = \text{slerp}(\mathbf{d}_0, \mathbf{d}_1, \frac{2}{3})$

Not the same result!

- But, close enough
- Even closer when:
 - $\mathbf{d}_0, \mathbf{d}_1$ similar OR t close to $\frac{1}{2}$
- Is it worth the extra computation cost? 🤔

42

The formulas

- LERP + normalization:

$$(1 - t) \mathbf{d}_0 + t \mathbf{d}_1 \quad \left. \vphantom{(1 - t) \mathbf{d}_0 + t \mathbf{d}_1} \right\} \text{ aka "NLERP"}$$

then re-normalize

- or SLERP:

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

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

43

SLERP: notes

- Applies to any 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 final result

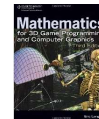
44

Point and vector algebra (summary 4/7)



Products between vectors, or between versors

- Dot product (or inner product)
 - Output: a scalar
- Cross product (or vector product)
 - Output: a vector (note: not a versor)



Section 2.2



Section 2.3

45

Point and vector algebra (summary 5/7)



- Dot product, useful to:
 - test orthogonality (if orthogonal then $\text{res} == 0$)
(between vectors, and/or versors alike)
 - sign tells: angle $<$ or $> 90^\circ$
(between vectors, and/or versors alike)
 - 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

46

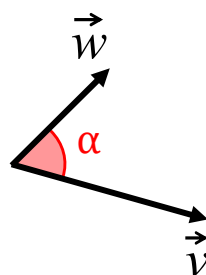
Point and vector algebra (summary 6/7)



- 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 (vector cross vector): 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 vector × 2D vector: (signed) sin of angle

47

Products and angles



$$\begin{aligned}\vec{v} \cdot \vec{w} &= \|\vec{v}\| \cdot \|\vec{w}\| \cdot \cos(\alpha) \\ \|\vec{v} \times \vec{w}\| &= \|\vec{v}\| \cdot \|\vec{w}\| \cdot |\sin(\alpha)|\end{aligned}$$

48

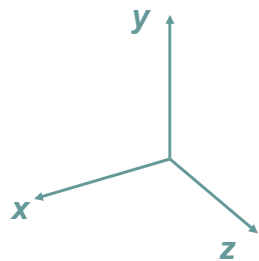
Note: Generalization to N - Dimensions



- Everything seen in this lecture generalizes in 2D (for 2D games), or even in $N > 3$ dimensions
 - Exception: the cross product is only defined in 3D
 - But in 2D, the problem of finding a vector/vector orthogonal to one (just one!) given vector/vector is easy: *"swap coordinates, flip one* sign"*
 (x,y) orthogonal to $(-y,x)$, and also to $(y,-x)$
- *: which coordinate you flip determines if you rotate 90° clockwise or counterclockwise: try!

49

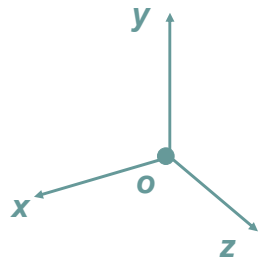
recap: Vector Base



- Axes: set of n lin. ind. vectors $(\mathbf{x}, \mathbf{y}, \mathbf{z})$
- Any vector \mathbf{v} can be expressed in exactly 1 way as a linear combination of these vectors
- The weights are the coord of \mathbf{v} in that base

50

recap: Reference Frame (or Space)



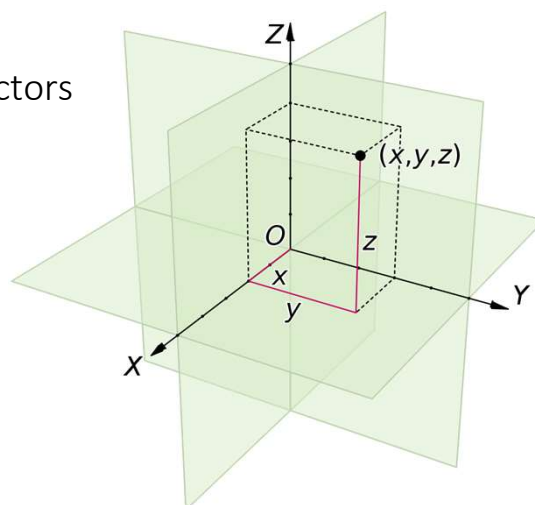
- n axes (**vectors**) (vector base)
+
1 origin (**point**)
- Any vector \mathbf{v} :
one linear comb. of the
axes
- Any point \mathbf{p} :
origin + one linear
comb. of axes

51

Recap: Orthonormal Frames Or Cartesian Frame



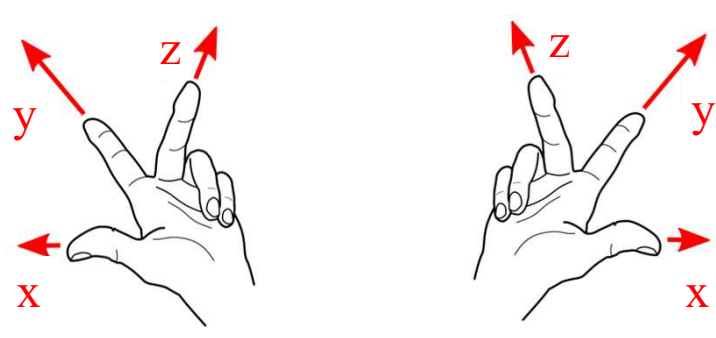
- Axes are unit vectors
and reciprocally
orthogonal



52

Recap: Handed-ness of a (Cartesian) frame

- They can be right- or left-handed



$x \times y = z$

$x \times y = z$
regardless!

Use the same hand to *imagine* a cross product

53

Still no standards in 3D games

personal opinion:
the most standard one,
among
3D modellers too

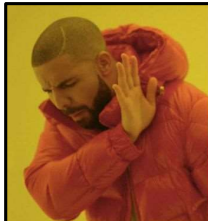
- **Unity**: left-handed: X-right, Y-up, Z-forward
- **Unreal**: left-handed: X-forward, Y-right, Z-up
- **3ds-Max**: right-handed, Z-up
- **Blender**: left-handed, Z-up
- **most VR systems**: right-handed, Y-up
- **OpenGL**: (clip space) right-handed, Y-up
- **DirectX**: (clip space) left-handed, Y-down

54

Pro-tip: try making your code assumption free!



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



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



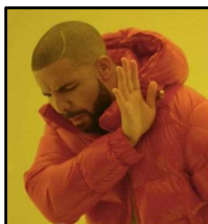
```
Vector3 pos = new Vector3 ( ... );  
  
pos += Vector3.right * 2.5;
```

56

Pro-tip: try making your code assumption free!



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



```
FVector pos = FVector( ... );  
  
pos.X += 2.5f; // maybe ??  
pos.Y += 2.5f; // hmm...??
```

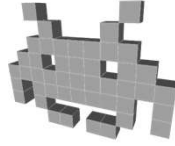


```
FVector pos ( ... );  
  
pos += FVector::RightVector * 2.5f;
```


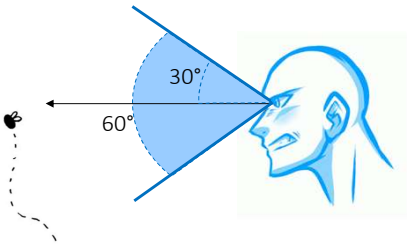
57

Master Videogames Univerona

Points, Vectors, Versors: mini task and exercises




Marco Tarini



59

Point to point distance



“When the player in position \mathbf{p} is closer than k to a powerup in pos \mathbf{q} , then the powerup is collected”

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

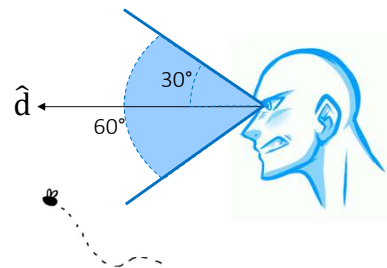
```
vec3 p,q;  
scalar k;  
if ( dot(p-q,p-q) < k*k ) then /*collect*/
```

62

Vision cones

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

- Hypotheses: no occlusions
- Trace:
 - For angles α, β in $0..90^\circ$: $\alpha < \beta \Leftrightarrow \cos(\alpha) > \cos(\beta)$
 - Find cosine of angle between view direction and the vector connecting \mathbf{q} to \mathbf{p}
 - Determine if this cosine is $> \cos(60^\circ/2)$



64

Orthonormal base completion

“I have a only two axes $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ of an orthonormal bases, how do I find the third vector $\hat{\mathbf{z}}$?”

- Data: $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ versors
- Hypotheses: $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are already orthogonal
- Variant: $\hat{\mathbf{y}}$ is not exactly orthogonal to $\hat{\mathbf{x}}$, but I want to change it the least to make it orthogonal ($\hat{\mathbf{x}}$ is to be kept constant) (see next problem)

65

Vector orthogonalization



“Find a versor \hat{u}' that is ortogonal to a given \hat{n} such that it is as similar as possible to a given versor \hat{u} ”

66

Vector orthogonalization



“Find a versor \hat{u}' that is ortogonal to a given \hat{n} such that it is as similar as possible to a given versor \hat{u} ”

Solution: $\hat{u}' = \hat{n} \times \hat{u} \times \hat{n}$, then renormalize it

In code (examples in different languages):

```
vec3 n,u;  
u = cross( cross( n , u ) , n );  
u = normalize( u );
```

GLSL or HLSL

```
FVector n,u;  
u = FVector::CrossProduct( FVector::CrossProduct(n,u),n );  
u.Normalize();
```

C++, with UE

```
Vector3 n,u;  
u = Vector3.Cross( Vector3.Cross( n , u ) , n );  
u = u.normalized;
```

C#, with Unity

67

Ray-sphere intersection



“I shoot a laser from \mathbf{p} to direction $\hat{\mathbf{d}}$. Do I hit a sphere in position \mathbf{q} of radius r ? Where?”

- Data: \mathbf{p}, \mathbf{q} points, r scalar, $\hat{\mathbf{d}}$ versor
- Trace:
 - Hit-point is \mathbf{s} on laser ray:
 $\mathbf{s} = \mathbf{p} + k \hat{\mathbf{d}}$, for some unknown scalar $k \geq 0$
 - Hit-point is \mathbf{s} on sphere:
 $\|\mathbf{q} - \mathbf{s}\| = r \iff (\mathbf{q} - \mathbf{s}) \cdot (\mathbf{q} - \mathbf{s}) = r^2$
 - Combine the two equations (substitute \mathbf{s} in second), solve for k (it's a 2nd degree equation), test that k exists and that it is >0

68

Shooting a walking target (with a finite speed bullet) 1/2



“I shoot a bullet from \mathbf{p} with velocity $\vec{\mathbf{v}}$. At which time the bullet will be the closest to a target currently in position \mathbf{q} and moving with velocity $\vec{\mathbf{w}}$? Where will bullet and target be, at that point?”

- Data: \mathbf{p}, \mathbf{q} points, $\vec{\mathbf{v}}$ and $\vec{\mathbf{w}}$ vectors
- Hypothesis: nothing accelerates (everything keeps moving at a constant speed)

69

Shooting a walking target (with a finite speed bullet) 2/2



Trace

- Position of bullet at time t : $\mathbf{p} + t \vec{\mathbf{v}}$
- Position of target at time t : $\mathbf{q} + t \vec{\mathbf{w}}$
- Squared distance between the two at time t :

$$\| (\mathbf{p} + t \vec{\mathbf{v}}) - (\mathbf{q} + t \vec{\mathbf{w}}) \|^2$$

=

$$\| (\mathbf{p} - \mathbf{q}) + t (\vec{\mathbf{v}} - \vec{\mathbf{w}}) \|^2$$

- Work on formulas (remember that $\|\vec{\mathbf{v}}\|^2 = \vec{\mathbf{v}} \cdot \vec{\mathbf{v}}$)
find derivative for dt ,
equate derivative to 0, extract t

70

Ray-Plane intersection



“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 (magnitude and verse are not 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

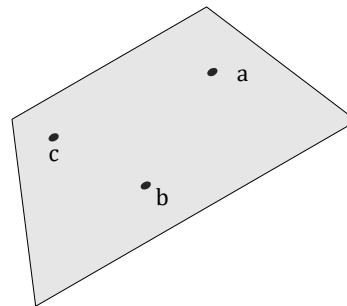
71

Sub problem: surface normal



“I have three points on a plane: find the normal \hat{n} of this plane (a versor)”

- Trace:
find any two
different vectors
on the plane
(i.e. parallel to it)



72