how do you reflect a vector over another vector?

Asked 9 years ago Active 9 years ago Viewed 3k times

1

I'm using AS3 to program some collision detection for a flash game and am having trouble figuring out how to bounce a ball off of a line. I keep track of a vector that represents the ball's 2D velocity and I'm trying to reflect it over the vector that is perpendicular to the line that the ball's colliding with (aka the normal). My problem is that I don't know how to figure out the new vector (that's reflected over the normal). I figured that you can use Math.atan2 to find the difference between the normal and the ball's vector but I'm not sure how to expand that to solve my problem.

actionscript-3 vector physics

Share Follow

asked Feb 14 2013 at 23:04

avorum
2,033 9 34 48

3 Answers

Active | Oldest | Score

5

Vector algebra - You want the "bounce" vector:
vec1 is the ball's motion vector and vec2 is the surface/line vector:

```
// 1. Find the dot product of vec1 and vec2
// Note: dx and dy are vx and vy divided over the length of the vector (magnitude)
var dpA:Number = vec1.vx * vec2.dx + vec1.vy * vec2.dy;

// 2. Project vec1 over vec2
var prA_vx:Number = dpA * vec2.dx;
var prA_vy:Number = dpA * vec2.dy;

// 3. Find the dot product of vec1 and vec2's normal
// (left or right normal depending on line's direction, let's say left)
var dpB:Number = vec1.vx * vec2.leftNormal.dx + vec1.vy * vec2.leftNormal.dy;

// 4. Project vec1 over vec2's left normal
var prB_vx:Number = dpB * vec2.leftNormal.dx;
var prB_vy:Number = dpB * vec2.leftNormal.dy;

// 5. Add the first projection prA to the reverse of the second -prB
var new_vx:Number = prA_vx - prB_vx;
var new_vy:Number = prA_vy - prB_vy;
```

Assign those velocities to your ball's motion vector and let it bounce.

PS:
vec.leftNormal --> vx = vec.vy; vy = -vec.vx;
vec.rightNormal --> vx = -vec.vy; vy = vec.vx;

To clarify, vec1.vx would be the x component of the vector. And also, vec1.dx = vec1.vx / vec1.length? – avorum Feb 15 2013 at 0:03

1 I really recommend the book *Advanced Game Design with Flash By Rex Van Der Spuy*. There is an excellent introduction chapter on vectors besides many other interesting stuff, and it's for flash. – chadiik Feb 15 2013 at 2:14

It's *vector algebra*. There's no such thing as *vector physics*. Vectors are abstract mathematical objects and no laws of physics apply to them. – Hristo Iliev Feb 15 2013 at 13:23

1

The mirror reflection of any vector **v** from a line/(hyper-)surface with normal **n** in any dimension can be computed using projection tensors. The parallel projection of **v** on **n** is: **v**|| = (**v** . **n**) **n** = **v** . **nn**. Here **nn** is the outer (or tensor) product of the normal with itself. In Cartesian coordinates it is a matrix with elements: `nn[i,j] = n[i]*n[j]` . The perpendicular projection is just the difference between the original vector and its parallel projection: **v** - **v**||. When the vector is reflected, its parallel projection is reversed while the perpendicular projection is retained. So the reflected vector is:

v' = -**v**|| + (**v** - **v**||) = **v** - 2 **v**|| = **v** . (**I** - 2 **nn**) = **v** . **R**(**n**), where
R(**n**) = **I** - 2 **nn**

(**I** is the identity tensor which in Cartesian coordinates is simply the diagonal identity matrix *diag(1)*)

R is called the reflection tensor. In Cartesian coordinates it is a real symmetric matrix with components `R[i,j] = delta[i,j] - 2*n[i]*n[j]` , where `delta[i,j] = 1` if `i == j` and `0` otherwise. It is also symmetric with respect to **n**:

R(-**n**) = **I** - 2(-**n**)(-**n**) = **I** - 2 **nn** = **R**(**n**)

Hence it doesn't matter if one uses the outward facing or the inward facing normal **n** - the result would be the same.

In two dimensions and Cartesian coordinates, `R` (the matrix representation of **R**) becomes:

```
    [ R00  R01 ]   [ 1.0-2.0*n.x*n.x     -2.0*n.x*n.y ]
R = [          ] = [                                 ]
    [ R10  R11 ]   [    -2.0*n.x*n.y  1.0-2.0*n.y*n.y ]
```

The components of the reflected vector are then computed as a row-vector-matrix product:

```
v1.x = v.x*R00 + v.y*R10
v1.y = v.x*R01 + v.y*R11
```

or after expansion:

```
k = 2.0*(v.x*n.x + v.y*n.y)
v1.x = v.x - k*n.x
v1.y = v.y - k*n.y
```

In three dimensions:

```
k = 2.0*(v.x*n.x + v.y*n.y + v.z*n.z)
v1.x = v.x - k*n.x
v1.y = v.y - k*n.y
v1.z = v.z - k*n.z
```

Finding the exact point where the ball will hit the line/wall is more involved - see here.

Share Follow

edited May 23 2017 at 12:17
Community Bot
1 1

answered Feb 15 2013 at 10:40
Hristo Iliev
68.8k 11 124 175

0

Calculate two components of the vector.

One component will be the projection of your vector onto the reflecting surface the other component will be the projection on to the surface's normal (which you say you already have). Use dot products to get the projections. Add these two components together by summing the two vectors. You'll have your answer.

You can even calculate the second component A2 as being the original vector minus the first component, so: A2 = A - A1. And then the vector you want is A1 plus the reflected A2 (which is simply -A2 since its perpendicular to your surface) or:

Ar = A1-A2

or

Ar = 2A1 - A which is the same as Ar = -(2A2 - A)

If [Ax,Bx] is your balls velocity and [Wx,Wy] is a unit vector representing the wall:

A1x = (Ax*Wx+Ay*Wy)*Wx;

A1y = (Ax*Wx+Ay*Wy)*Wy;

Arx = 2*A1x - Ax;

Ary = 2*A1y - Ay;

Share Follow

edited Feb 15 2013 at 3:56

answered Feb 14 2013 at 23:25
Octopus
7,517 5 41 63
