Elastic collision between oblique wall and ball

Asked 15 years, 11 months ago Modified 15 years, 10 months ago Viewed 3k times

2

I'm having trouble computing reflection angles for a ball hitting an oblique wall. I'm using an algorithm lifted from this tutorial. It looks like this (in Actionscript 3), with p1 being the current velocity vector and p2 the normal of the wall:

```
private function getReflect2(p1 : Point, p2 : Point) : Point
{

    var wallvec : Point = getNorm(p2);
    var wallnorm  : Point = p2;

    var t : Number = dotProduct(wallvec, p1);
    var n : Number = dotProduct(wallnorm, p1);
    var vt : Point = new Point(wallvec.x * t, wallvec.y * t);
    var vn : Point = new Point(wallnorm.x * -n, wallnorm.y * -n);

    var vx : Number = dotProduct(new Point(1,0), vn) + dotProduct(new
Point(1,0), vt);
    var vy : Number = dotProduct(new Point(0,1), vn) + dotProduct(new
Point(0,1), vt);
    return new Point(vx, vy);
}
```

The function returns the new velocity vector, and does so correctly for collisions with perpendicular walls but not for oblique ones. The ball may hit the wall from both "sides" (ie. the normal may be jutting in the other direction).

Can anyone spot my error? Or suggest a better algorithm?

actionscript-3 collision-detection

Share
Improve this question
Follow

edited Dec 27, 2008 at 23:46
user1228

asked Dec 27, 2008 at 23:40

friism
19.3k ● 5 ● 82 ● 116

5 Answers

Sorted by: Highest score (default)

1

Vectors wallvec and wallnorm must have a magnitude of 1 for the function to work. I suspect that the function getNorm() handles the work for wallvec for you, but wallnorm seems to get an arbitrary magnitude.

Also, your implementation doesn't exactly match the algorithm linked on the page. In particular, the page uses P2 as the direction of the wall itself. It then normalizes the vector (which reduces the magnitude to 1 without changing direction) and copies it into wallvec. Wallnorm receives the normal of the wall (which is already at magnitude 1.)

My C implementation looks like this. Note that even though wnx and wny is the same direction as p2c and p2y, I still had to divide by the magnitude.

```
void solve (double p1x, double p1y, double p2x, double p2y, double *p3x, double
*p3y)
{
double wvx = -p2y / sqrt(p2x*p2x+p2y*p2y);
double wvy = p2x / sqrt(p2x*p2x+p2y*p2y);
double wnx= p2x/ sqrt(p2x*p2x+p2y*p2y);
double wny=  p2y/ sqrt(p2x*p2x+p2y*p2y);

double t = wvx*p1x+wvy*p1y;
double n = wnx*p1x+wny*p1y;

double vtx = wvx*t;
double vty = wvy*t;
double vnx = wnx*-n;
double vny = wny*-n;

*p3x=vnx+vtx;
*p3y=vny+vty;

}
```

Share Improve this answer Follow

answered Dec 28, 2008 at 0:37

Raymond Martineau

6,039 ● 1 ● 22 ● 12

0

I think the answer may be the part about "(ie. the normal may be jutting in the other direction)." If the normal is pointed "into" the wall, the reflection will be *through* the wall, not in the direction you expect.

Share Improve this answer Follow

answered Dec 28, 2008 at 0:00

Charlie Martin

112k ● 26 ● 196 ● 264

Normal direction doesn't affect the collision. (I just tested this.) – Raymond Martineau Dec 28, 2008 at 0:29

0

If it works for perpendicular, maybe `wallvec` is not right. Have you printed out `wallvec` and other values?

Edit: Could you provide all the vector values when `P1` is `(1, 0)` and `P2` is `(0, 1)`?

Share

Improve this answer

Follow

edited Dec 28, 2008 at 0:04

answered Dec 27, 2008 at 23:51

Eugene Yokota

95.5k ● 45 ● 217 ● 320

> Yup, I even have a debugger-mode displaying the various vectors. The only one that looks wrong is the outgoing one. – friism Dec 27, 2008 at 23:54

0

This isn't quite an answer to what you asked, but the last few lines are unnecessarily complicated. Since (1, 0) represents the *x*-axis,

```
dotProduct(new Point(1,0), vn)
```

can be replaced by vn.x. Thus, the last three lines of your function can be made a lot more clear by simply referring to the appropriate members of vn and vt:

```
return new Point(vn.x + vt.x, vn.y + vt.y);
```

I'm not familiar enough with ActionScript to know if there's a more convenient way to express vector addition, but it would be better to express the vector addition directly.

Share Improve this answer Follow

answered Jan 3, 2009 at 3:36

cgranade

11 ● 1

0

it's untested, but this should reflect your velocity

```
private function reflectVectors(p1 : Point, p2 : Point) : Point
{
    // get the normalized normal
    var wallNorm:Point = p2.clone();
    wallNorm.normalize(1);

    // v1 = v - 2 * (v.n).n
    var dot:Number = p1.x * wallNorm.x + p1.y * wallNorm.y;
    var diff:Point = wallNorm.clone();
    diff.normalize(dot * -2);
    return p1.add(diff);
}
```

Share Improve this answer Follow

answered Feb 6, 2009 at 22:52

Sean
509 ● 4 ● 12