Vectors

Basics of vectors

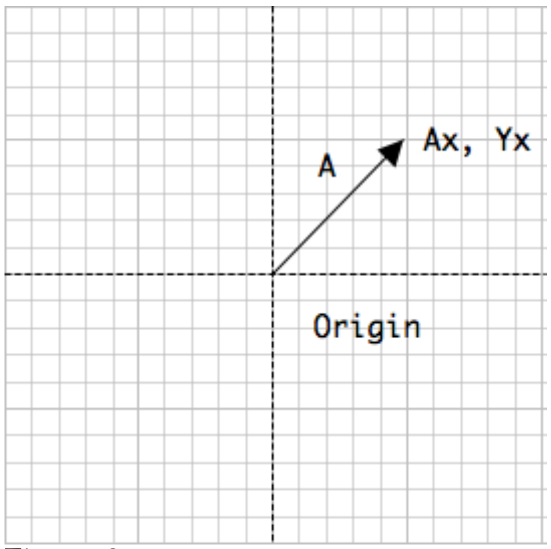


Figure 8.

A vector is a number pair that describes a spatial relationship. Vectors may be polar or Cartesian, although the latter are more common. They are usually defined by two numbers[1], which describe how the vector looks if it starts at the origin. The numbers are then the x and y coordinates of the point. Vectors may be labeled by a letter (generally upper case), A letter under an arrow, or two letters with or without an arrow- in that case the vector AB is the vector from point A to Point B. Vectors may also be indicated by the coordinates in a pair of brackets, either as a row or a column.

$$\vec{\mathbf{A}} \qquad \overrightarrow{\mathbf{AB}} \qquad [A_x, A_y] \qquad \begin{bmatrix} A_x \\ A_y \end{bmatrix}$$

You can think of the space defined by a vector as a path to travel, so a vector defines a direction and distance[2]. The distance or length of the vector is the magnitude. It can be easily calculated by the hypotenuse formula. Magnitude is often indicated by |A|. The tangent of the angle of travel is the ratio of the X and Y coordinates, so the angle can be found by the arctangent. In fact, this is just the Cartesian to polar conversion.

[1] Or three or more, we'll come to that in a bit.

[2] A vector has the same semantic meaning as "three miles north". It's a concept we understand, although it's not much use until we know where to start.

Keeping the association of motion and vectors in mind tells us how (and why) to add vectors. The result should tell where we wind up, it is as if one vector were stuck at the point of the other:

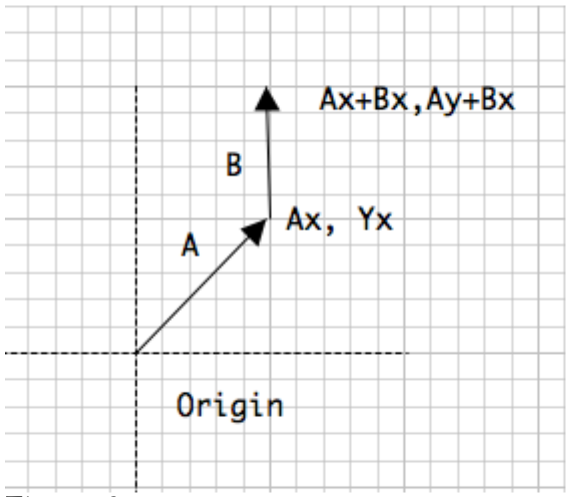


Figure 9.

The new X coordinate is the sum of the vectors' X and the new Y coordinate is the sum of the vectors' Y. These points define a new vector, of course, and you can draw it in if you like.

Simple addition accounts for 90% of our work with vectors. Vectors are used to represent anything that has magnitude and direction. Forces, velocity, stock market trends, line art in Illustrator and so on. Vectors work in three dimensions if we just add a z component. The magnitude becomes sqrt(X^2+Y^2+Z^2). Everything else is the same.

There are times when we use a vector to define a direction, such as the axis of rotation for a 3D object. In that case we think of the magnitude as infinite (defining a line) or we set it to 1. This is called a unit vector.

A line segment is similar to a vector-- it is the line between two points. The line from point A to point B is often called line AB. Both A and B are defined by coordinates. You can convert a line segment to a vector by subtracting the start from the end. A line can also be represented as a starting point and a vector. In that case the form is P+uD, where P is a point, D is a unit vector and u is the length of the line[3].

When we are dealing with objects scattered around a coordinate system, it is often handy to translate the coordinates so our starting point is 0,0. This is done by subtracting the global coordinates of the starting point from all of the coordinates we will work with. When we are done, we need to add those values back to the results.

[3] This form is called parametric notation. A lot of sources use t instead of u.

Putting vectors to work.

A little knowledge can be dangerous, but the properties of vectors covered so far can be quite useful in Max & Jitter patches. In particular, vectors can be used to control motion in the GL world.

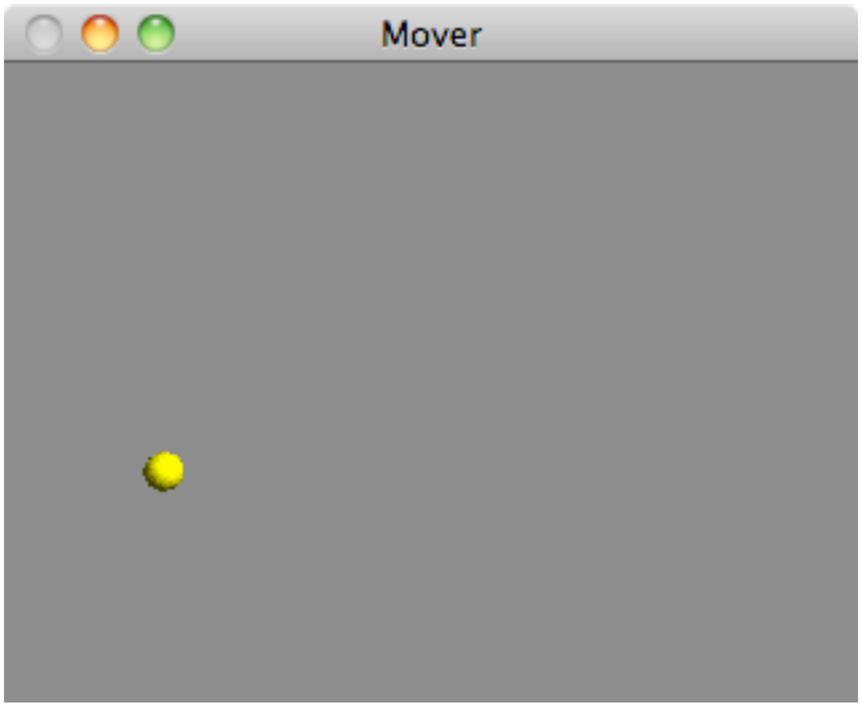
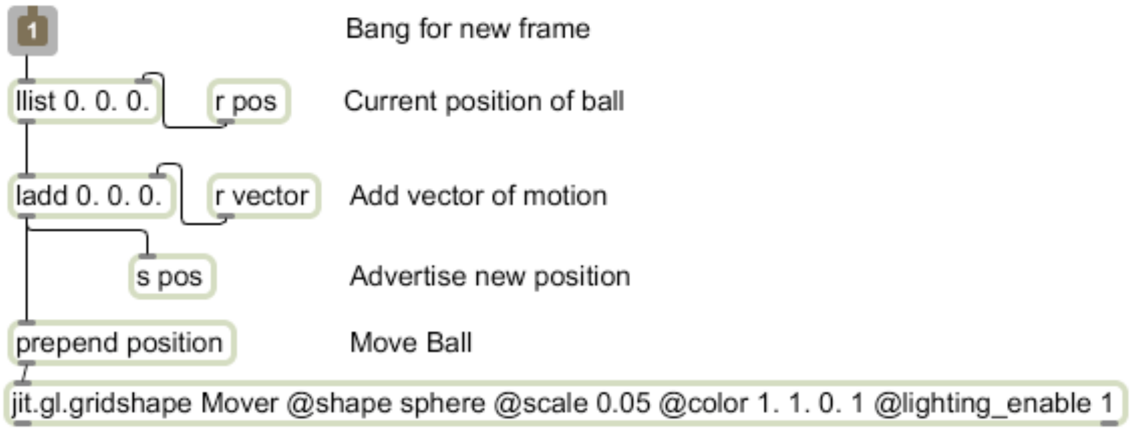


Figure 10.

Figure 10 shows a simple 3D window, with a single sphere generated by jit.gl.gridshape. We will use vectors to put the sphere in motion.



```
1                                        Bang for new frame
llist 0. 0. 0.    r pos                  Current position of ball
ladd 0. 0. 0.     r vector               Add vector of motion
          s pos                          Advertise new position
prepend position                         Move Ball
jit.gl.gridshape Mover @shape sphere @scale 0.05 @color 1. 1. 0. 1 @lighting_enable 1
```

Figure 11.

Figure 11 shows the mechanism for drawing the ball. This is contained in a subpatch named draw_ball. The current location is held in an llist object. For each new frame this list is sent to an Ladd object containing a vector that defines the motion[4]. The result is used to position the sphere and is sent back to the top llist object for the next frame. You can easily see how the sphere would move. If the Ladd object contained the motion vector 0.1 0. 0., successive frames would produce positions of:

```
0.    0.   0.
0.1   0.   0.
0.2   0.   0.
0.3   0.   0.
```

and so on. This would move the sphere to the right.

Figure 12 shows one method for creating the motion vector.

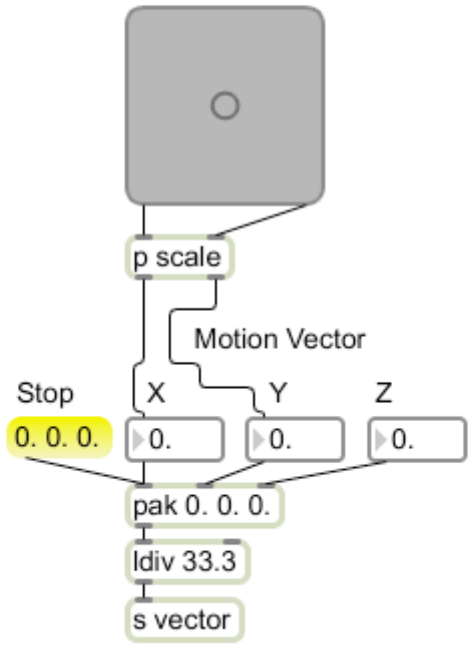


Figure 12.

The top object in Figure 12 is a pictslider. It is an X-Y controller which produces a pair of numbers that range from 0 to 127. The scale subpatch remaps these to the range of -1 to 1. This is done with a pair of expr objects containing $f1/127. – 1. (It would be very simple to connect a physical joystick here—see the controllers tutorial.) The X and Y values are packed into a vector along with Z. This vector is divided by 33.3 to convert the values shown to gl units per second.

The send (s vector) object sends the vector to the mechanism shown in figure 11. When the control in the pictslider is moved from the center position, the ball will move in the direction indicated, with a speed proportional to the distance the slider is from the center.

[4] Physicists call this vector velocity. Velocity is speed and direction, perfect for vector representation.

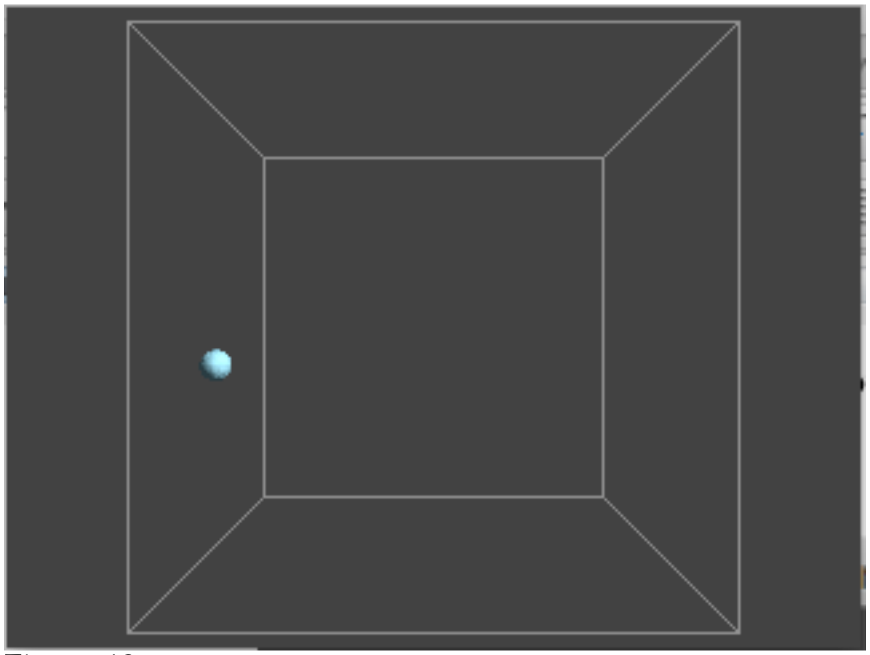


Figure 13.

Figure 13 shows the output of a more complex patch. When this is running, the blue ball bounces around inside the box. If we like, the ball will gradually slow down and come to a halt, and we have the further option of turning on gravity to make the ball settle on the box bottom.

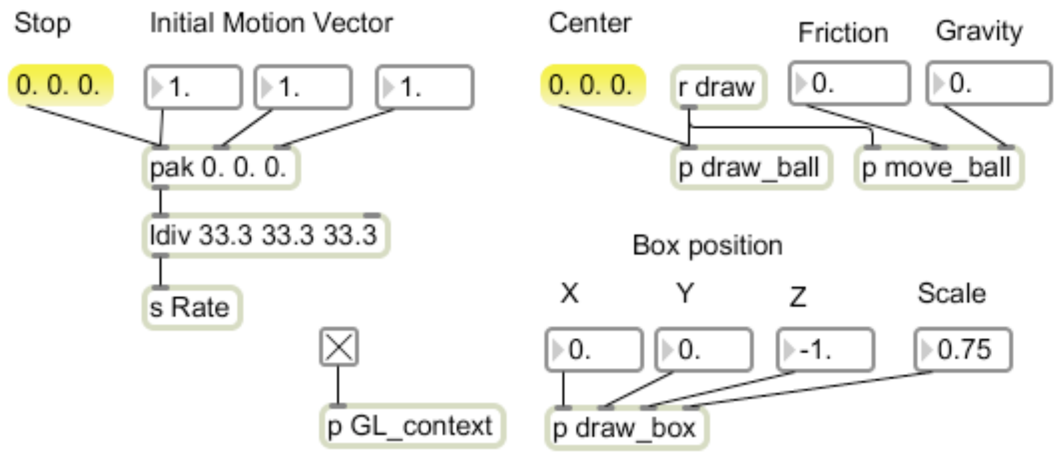


Figure 14.

Figure 14 shows the master patch. The essential functions are encapsulated:

- GL context contains the usual window and render objects needed for any jit.GL display.
- Draw_box contains a jit.GL.plato object that will generate a cube with poly_mode set to 1 1 so only the edges show. Note that the position and size (scale) of the box may be adjusted.
- Move_ball calculates the motion vector for the ball.
- Draw_ball draws the blue ball .

The Draw_ball subpatch is the same as figure 11. (This patch can be used to move anything) If the motion vector is fixed, the ball will move forever in the same direction. The core of the problem is to detect when the ball has moved too far. Figure 15 shows the move_ball subpatch.

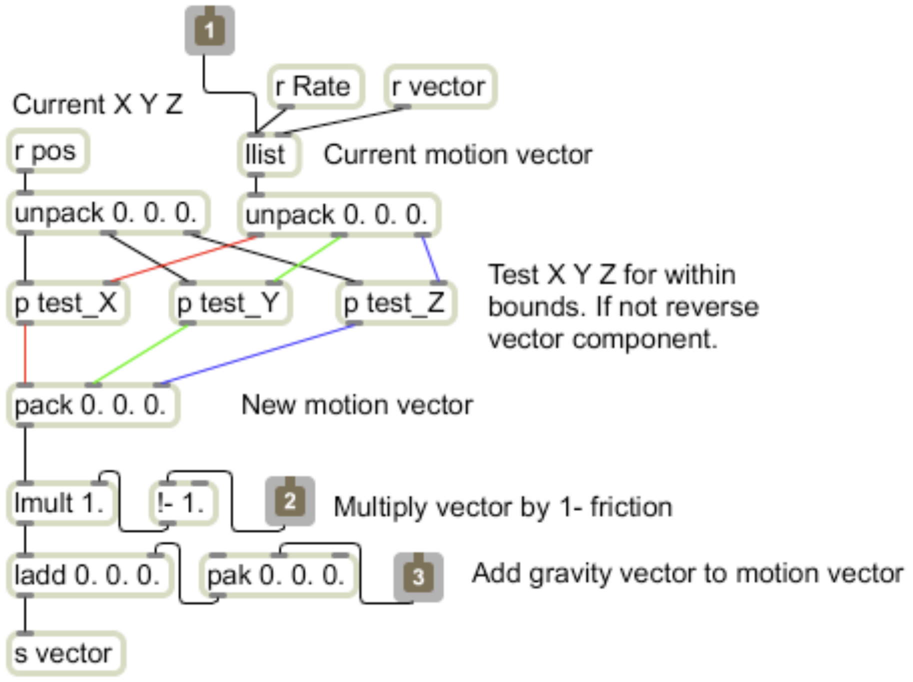


Figure 15. Move_ball subpatch

There are two major vectors involved here. These define the current position and the current motion. This patch tests the current position to see if the ball is moving out of bounds. If so, the motion vector must be modified.

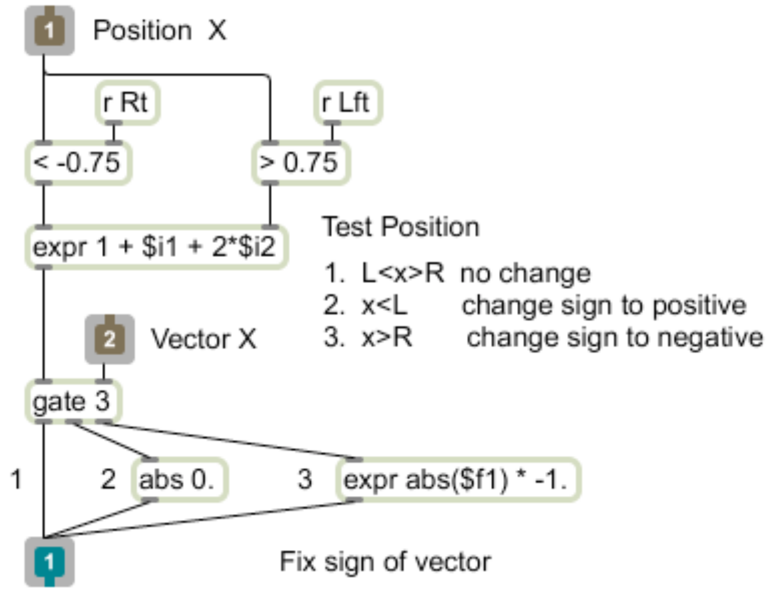


Figure 16.

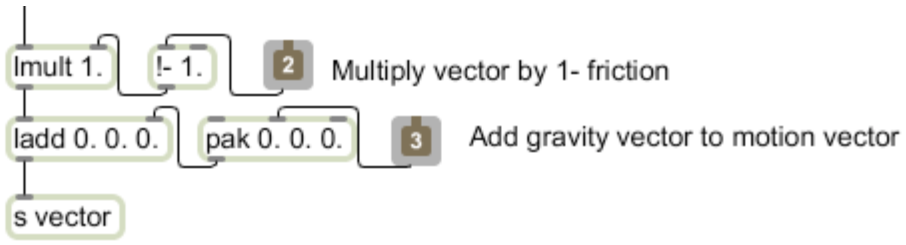
Figure 16 shows how the position is tested and the vector modified if necessary. There are three possible cases:

1. X falls between the left and right limits and no change is needed.
2. X is less than the left limit—the sign of the X component must be made positive.
3. X is greater that the right limit—the sign of the X component must be made negative.

The tests are performed by the < and > objects. (Note that the default test values can be changed if the bounding box is changed.) These objects report a 1 if the test is true and 0 if the test is not true. In case one neither test will be true, in the other cases, one or the other will be true. There is no condition where both would be true.

The expression 1 + $f1 + 2*$f2 will add up to 1 if neither test is true as both comparison objects will report 0. If the left hand test is true the expression will add up to 2. If the right hand test is true the expression will yield 3.

The X component of the motion vector is routed through a gate controlled by the top of the patch. If the component is sent from outlet 1, there is no change. If the component is sent from outlet 2, it passes through an abs object—this returns the absolute value of the X component. (Positive value) The expression on outlet 3 does the same thing, but multiplies the result by -1 so the sign is guaranteed negative.


lmult 1.
!- 1.
2
Multiply vector by 1- friction
ladd 0. 0. 0.
pak 0. 0. 0.
3
Add gravity vector to motion vector
s vector
Figure 17.
The identical process is applied to the Y and Z components.

The motion vector may be further modified by the actions shown in figure 17.

- Friction will gradually reduce the velocity, so multiplying the motion vector by a value slightly less than 1 will slow the ball down.
- Gravity is also a vector (it has direction and acceleration). The motion of any object is always determined by the sum of the velocity vector and gravity vector. The result of a negative y component added to each frame is movement of the object toward the bottom of the box.

This simple use of vectors will produce interesting and realistic action within the GL world.

Advanced Vectors

(Patches using these functions will appear eventually.)
Sometimes we start with a vector that is not at the origin, as in B of the above drawing. Before we do any math on the vector, we must subtract the coordinates of the vector's starting point (vector origin). Sometimes we will see equations that include this operation with terms like A_x-A_0 . The following operations assume the vector origin is [0 0].

Scaling

A number that is not a vector is a scalar. We can't add a scalar to a vector, but we can multiply them. 3 (a scalar) times [2 2] (a vector) is [6 6]. Each component of the vector is multiplied by the scalar. The operation is called (what else?) scaling.

Dot Product

We can multiply two vectors in a way that produces a scalar. This is the scalar product or dot product[5]. With a bit of geometry, I could show that $A \cdot B = A_xB_x+A_yB_y$. I could also show that it is equal to the product of the magnitudes of the two vectors times the cosine of the angles between them. $A \cdot B = AB\cos\theta$. This lets me find the angle between two vectors by
$\theta = \text{acos}((A_x*B_x+A_y*B_y) \;/\; (\text{sqrt}(A_x*A_x+A_y*A_y)* \text{sqrt}(B_x*B_x+B_y*B_y)))$

For three dimensional vectors, the dot product is $A \cdot B = A_xB_x + A_yB_y + A_zB_z$. The cosine formula still holds, but you need to use the z terms in calculating dot product and length.

Normal

A vector normal is another vector that is at right angles to the original. Some math books indicate the normal to vector A as $\widehat{A}$. In two dimensions, the normal is easy-- you just exchange the coordinate values and change the sign of either one. (Any vector has two normals, in opposite directions.) So the normal to [2 4] is [4 -2]. We use the normal to define a direction, so the magnitude is not important. Any math involving the normal is simplified if we adjust the length of the normal to be 1. This is the unit normal, found by dividing each coordinate value by the magnitude. So to find the normal to a vector A:

$N_x = A_y / \text{sqrt}(A_x*A_x+A_y*A_y)$
and
$N_y = -A_x / \text{sqrt}(A_x*A_x+A_y*A_y)$

The dot product of a vector and its normal is always 0. The dot product of any pair of perpendicular lines is 0.

[5] Also known as inner product.

Cross Product

We don't find the normal to a vector in three dimensions, we find the normal to a plane.
Two vectors (and their origin) define a plane, and the normal to the plane is found by an operation known as the vector cross product. The cross product[6] is written with an 'x' for the multiplication sign as A x B = N

$N_x = A_yB_z - A_zB_y$
$N_y = A_zB_x - A_xB_z$
$N_z = A_xB_y - A_yB_x$

This operation is essential to determining how a vector is reflected from a plane. This is involved in lighting as well as modeling dynamics like a bouncing ball where the surfaces are not aligned to the axes.

Intersection

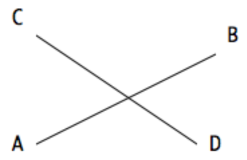


Figure 10.
Given two line segments AB and CD, find the point of intersection.
First calculate a denominator as (Dy-Cy)*(Bx-Ax) - (Dx - Cx)*(By-Ay).
If the denominator is 0, give up, because the lines are parallel.

Next
Uab = ((Dx-Cx)*(Ay - Cy) - (Dy - Cy)*(Ax - Cx))/denominator
Ucd = ((Bx-Ax)*(Ay - Cy) - (By - Ay)*(Ax - Cx))/denominator

If 0 <= Uab <= 1 and <= Ucd <= 1 the lines intersect.
The point of intersection is:
INTx = Ax + Uab*(Bx-Ax)
INTy = Ay + Uab*(By-Ay)

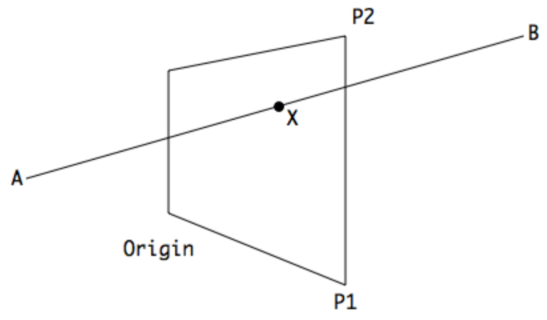


Figure 11.

[6] Cross product may be called outer product.

The intersection of a line AB and plane is a bit more complex. A plane is defined by three points, the origin, P1 and P2.[7] The cross product of P1 and P2 is the normal, N.

The vector of the line AB is (B-A). If (B-A) ·N = 0, the line does not intersect the plane.

The intersection point X = A + (B-A)* (P1 -A) ·N/(B-A) ·N

To find out if the intersection is within a specific polygon on the plane, consider a line from the point to a point on the plane known to be outside the polygon. Test this line for intersections with the edges of the polygon. If there are an odd number of intersections, the point is within the polygon.

Reflection

When modeling a bouncing ball, the reflections off of the top and ends of the frame are simple, we only need change the sign of one of the components of the motion vector. When the surface is angled, the situation is a bit more complex. First we must test that the vector and surface are intersecting as above. For the most accurate model, we will also need the point of intersection.

First find the unit normal to the wall, N. The reflection R of the vector V is symmetrical about the normal. The formula that expresses that is
R = V-2(V·N)N.

That's a dot product in there, which produces a scalar, so the component by component math is easy

$R_x = V_x - 2(V \cdot N)N_x$
$R_y = V_y - 2(V \cdot N)N_y$
$R_z = V_z - 2(V \cdot N)N_z$

This works in either 2 or 3 dimensions.

[7] If necessary, translate the plane and line so one of the points on the plane is the origin.