Spherical Coordinates and Axes Rotation in NWN.

Dan Franklin (Genji)

Jan 3, 2003

1 Content

There are several ways to pin down a point in 3-space. The most common is the triplet (x, y, z) where each is a length of distance from the origin $(0, 0, 0)$. There are other ways of relaying where this point is in the space; for example in two space we have polar coordinates: (r, Θ). r is the distance from the origin to the point in question and Θ is the angle between that line and the X axis, as in Figure 1.

Remembering our trig we see that the length r is the hypotenuse of the triangle with sides length x and y. The distance $x = r\cos\Theta$ and $y = r\sin\Theta$. So we can map each polar coordinate to a regular rectangular one with these two functions. Since this angle is drawn on a flat plane, or a flat piece of paper as you look down at it, I will call this the yaw angle. Yaw is the movement of a plane caused by the rudder, and basically makes a plane turn left or right while remaining flat.[1] With that in mind, you say what about pitch and roll? Well that's where the meat of this article comes from.

Polar coordinates can't give us a height on the point, so to include the rest of three space we add another angle ϕ. A spherical coordinates is the triplet (r, Θ, ϕ) where r is the distance to the origin, Θ is the same as polar, it is the angle from the X axis in the XY plane, and ϕ is the angle from the line through r to the XY plane. See Figure 2. I'll call the ϕ angle the pitch angle since that is the degree of up or down we have from the origin. We see

$$r = \Phi\sin\phi \tag{1}$$
$$\Theta = \Theta \tag{2}$$
$$z = \Phi\cos\phi \tag{3}$$

Then since $x = r\cos\Theta$ and $y = r\sin\Theta$ in polar, we have that

$$x = \Phi\sin\phi\cos\Theta \tag{4}$$
$$y = \Phi\sin\phi\sin\Theta \tag{5}$$
$$z = \Phi\cos\phi \tag{6}$$

[1]There's more to it than that of course, as when a plane turns while flat, one wing gets air going over it faster than the other and you change the amount of lift under each as a result. This causes the plane to move in a roll direction as well.

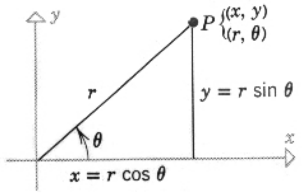

Figure 1: *Polar Coordinates*

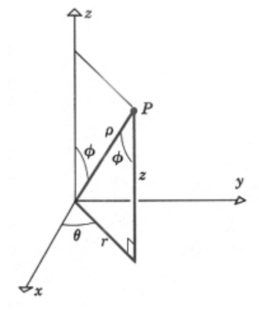

Figure 2: *Spherical Coordinates.*

and we have equations that will translate between spherical coordinates and common rectagular coordinates.

So given any point in three space we can find another point that is r distance away on yaw angle Θ and pitch angle ϕ. Now that we have this set up, let's look at changing elements of the triplet while keeping one constant. Constant r gives us a sphere, so given an origin point you can find any location on the sphere of radius r around it. Constant Θ gives us a plane perpendicular to the XY plane. Constant ϕ gives us cones, above and below depending on whether ϕ is negative or not. See Figure 3. You can find any other point in respect to an origin point.

But what about treating the origin as an axis and then translating our location by turning that axis?

Rotating around an axis is nothing new, we already rotated around the Z-axis in our polar coordinates example that gave us yaw. So we'll generalize what went on with that situation and make equations that will do the work for us. If we look at the XY plane by itself in two space we see that any point p in that plane can be given in relation to any other rotated axes with the exact same origin. See Figure 4.

By keeping the axis constant and reevaluating where the point would be for another axis, we can rotate the point. From the figure we see

$$x = r\cos(\alpha + \Theta) \tag{7}$$
$$y = r\sin(\alpha + \Theta) \tag{8}$$
$$x' = r\cos\alpha \tag{9}$$
$$y' = r\sin\alpha \tag{10}$$

From trig identities we can rewrite x and y and substitute in x' and y' with

$$x = r\cos\alpha\cos\Theta - r\sin\alpha\sin\Theta \tag{11}$$
$$= x'\cos\Theta - y'\sin\Theta \tag{12}$$

and

$$y = r\sin\alpha\cos\Theta + r\sin\Theta\cos\alpha \tag{13}$$
$$= y'\sin\Theta + x'\sin\Theta \tag{14}$$

These are the rotation equations for rotating a point around the Z axis (or yaw). For each axis we can find these equations and, for sake of brevity, we have
Rotation around the Y axis, (pitch):

$$x = x'\cos\Theta + z'\sin\Theta \tag{15}$$
$$z = z'\cos\Theta - x'\sin\Theta \tag{16}$$

and Rotation around the X axis, (roll): e

$$y = y'\cos\Theta - z'\sin\Theta \tag{17}$$
$$z = z'\cos\Theta + y'\sin\Theta \tag{18}$$

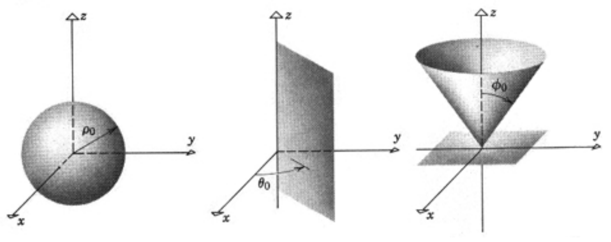

Figure 3: *Spherical Coordinates with One Held Constant.*

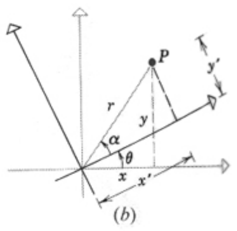

Figure 4: *Rotating the XY plane around the Z axis.*

As an example of this, and to be able to see it in action, here are some functions for NWN in nwscript that utilize these equations:

GetNearLocation uses Spherical Coordinates and normalizes the "X" axis to be whatever the current facing is of the location. The three rotation functions use the above rotation equations to rotate a given point around the origin point passed in.

```
// ——————————————————————————[ Function Prototypes ]

// this function returns a location using spherical coordinates
// with the X axis being the current facing of the location passed in.
// The location returned will be (r,offFacing_yaw,offFacing_pitch) where r = distance away,
// offFacing_yaw is the angle looking down at the location
// and offFacing_pitch is the angle looking at the side.
// offFacing_yaw: 0.0 is same direction you already are facing, -90.0 is exactly on
// the creature's right, 90.0 is on the left, and 180.0 is behind them.
// offFacing_pitch: 0.0 is same direction you already are facing, -90.0 is
// straight down, 90.0 is straight up, -180.0 is behind them.
// offFacingOnArrival can be used to turn the object granted this location by the amount specified.
// E.g. To get a location that is 10.0 meters on the creature's 11 oclock:
// location example = GetNearLocation(GetLocation(OBJECT_SELF),30.0,10.0);
// To get a location that is 4 meters behind you and up in the air by 45 degrees:
// location example2 = GetNearLocation(GetLocation(OBJECT_SELF),-180.0,4.0,0.0,45.0);
// issues? Genji@thegenji.com 12-31-02 updated.
location GetNearLocation(location loc,float offFacing_yaw, float distanceFrom,
    float offFacingOnArrival = 0.0,float offFacing_pitch = 0.0);
// take the location passed in and return a location derived by rotating the
// position around the origin passed in as the X axis.
location GetRotatedByAxisX(location loc, float angle, location origin);
// take the location passed in and return a location derived by rotating the
// position around the origin passed in as the Y axis.
location GetRotatedByAxisY(location loc, float angle, location origin);
// take the location passed in and return a location derived by rotating the
// position around the origin passed in as the Z axis.
location GetRotatedByAxisZ(location loc, float angle, location origin);
```

```
// ———————————————————————————————[ Function Implementations ]

location GetNearLocation(location loc,float offFacing_yaw, float distanceFrom,
    float offFacingOnArrival = 0.0,float offFacing_pitch = 0.0)
{

    float currentFacing = FixFacing(GetFacingFromLocation(loc));
    vector pos = GetPositionFromLocation(loc);
    float newX,newY,newZ;
    // return spherical coordinates with the facing of the location = the positive x axis.
    newX = pos.x + (distanceFrom * sin(90.0 - offFacing_pitch)* cos(currentFacing + offFacing_yaw));
    newY = pos.y + (distanceFrom * sin(90.0 - offFacing_pitch) * sin(currentFacing + offFacing_yaw));
    newZ = pos.z + (distanceFrom * cos(90.0 - offFacing_pitch));

    return Location(GetAreaFromLocation(loc),Vector(newX,newY,newZ),currentFacing +
offFacingOnArrival);
}

location GetRotatedByAxisX(location loc, float angle, location origin)
{
    // $x' = x, y' = y\cos\Theta - z\sin\Theta, z' = y\sin\Theta + z\cos\Theta$
    float currentFacing = FixFacing(GetFacingFromLocation(loc));
    vector originPos = GetPositionFromLocation(origin);
    vector pos = GetPositionFromLocation(loc);
    float newY,newZ;
    //adjust for origin, rotate, then add back in.
    newY = (pos.y - originPos.y) * cos(angle) - (pos.z - originPos.z) * sin(angle) + originPos.y;
    newZ = (pos.y - originPos.y) * sin(angle) + (pos.z - originPos.z) * cos(angle) + originPos.z;

    return Location(GetAreaFromLocation(loc),Vector(pos.x,newY,newZ),currentFacing);
}

location GetRotatedByAxisY(location loc, float angle, location origin)
{
    // $x' = z\sin\Theta + x\cos\Theta, y' = y, z' = z\cos\Theta - x\sin\Theta$
    float currentFacing = FixFacing(GetFacingFromLocation(loc));
    vector originPos = GetPositionFromLocation(origin);
    vector pos = GetPositionFromLocation(loc);
    float newX,newZ;
    //adjust for origin, rotate, then add back in.
    newX = (pos.z - originPos.z) * sin(angle) + (pos.x - originPos.x) * cos(angle) + originPos.x;
    newZ = (pos.z - originPos.z) * cos(angle) - (pos.x - originPos.x) * sin(angle) + originPos.z;

    return Location(GetAreaFromLocation(loc),Vector(newX,pos.y,newZ),currentFacing);
}

location GetRotatedByAxisZ(location loc, float angle, location origin)
{
    // $x' = x\cos\Theta - y\sin\Theta, y' = x\sin\Theta + y\cos(\Theta), z' = z$
    float currentFacing = FixFacing(GetFacingFromLocation(loc));
    vector originPos = GetPositionFromLocation(origin);
    vector pos = GetPositionFromLocation(loc);
    float newY,newX;
    //adjust for origin, rotate, then add back in.
    newX = (pos.x - originPos.x) * cos(angle) - (pos.y - originPos.y) * sin(angle) + originPos.x;
    newY = (pos.x - originPos.x) * sin(angle) + (pos.y - originPos.y) * cos(angle) + originPos.y;

    return Location(GetAreaFromLocation(loc),Vector(newX,newY,pos.z),currentFacing);
}
```

For a working example of this in an already set up area, assuming you have NWN, check out:

`http://www.thegenji.com/nwn/portal_gen.zip`