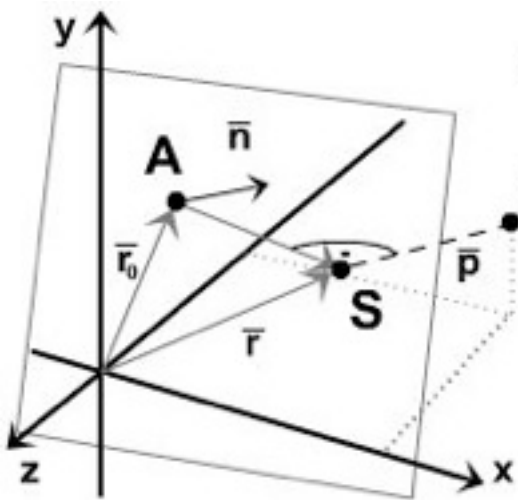


Projection Plane Based Real 3D

All children, from the age of 6 dreams of an own 3D engine. Even I did that. That's why i have written this tutorial, to make those people`s work easier who couldn't succeeded in doing so.

PART 1. THE THEORY

What is it about? We have a point in the 3 dimensional space, and we would like to project it perpendicularly to an arbitrary plane, and we have to determine the projected points position relative to a given point on the plane. We can define a plane in many ways, but in this case the best way is to define it by a normal vector and a base point, so it will be much more easier to move it. In addition, we can at once use the normal vector as the position vector of the line going through the chosen point in the 3D space.



Plane: $(\bar{r}-\bar{r}_0)\bar{n}=0$

$\bar{r}\bar{n}-\bar{r}_0\bar{n}=0$

$(\bar{p}_0+t\bar{n})\bar{n}-\bar{r}_0\bar{n}=0$

$t=\frac{\bar{r}_0\bar{n}-\bar{p}_0\bar{n}}{\bar{n}\bar{n}}$

$S=\bar{p}=\bar{p}_0+t\bar{n}$

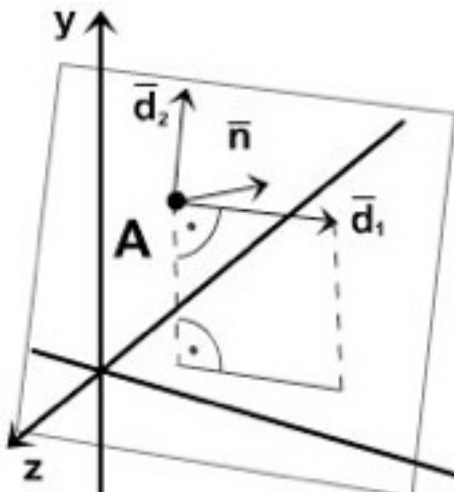
Line: $\bar{p}=\bar{p}_0+t\bar{v} \quad / \bar{v}=\bar{n}$

$\bar{p}=\bar{p}_0+t\bar{n}$

$\bar{r}=\bar{p} \quad / \text{intersection}$

$/ \text{if } t>0 \text{ not visible}$

We can define the intersection point by using the parametric equations of the line (normal to the plane, going through the point P) and the plane. However, we can't determine a relative position directly yet, because we only know one point related to the plane. We need two more points related to the plane with which we will be able to determine the position of S.

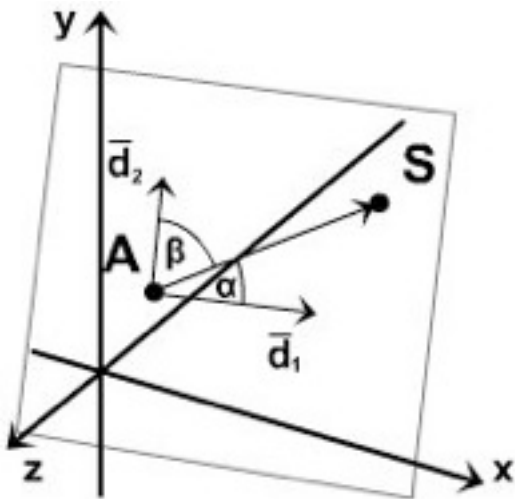


$\bar{d}_{1xz}=\bar{n}_{xz} \times \begin{pmatrix} \cos\varphi/2 & \sin\varphi/2 \\ -\sin\varphi/2 & \cos\varphi/2 \end{pmatrix} = \bar{n}_{xz} \times \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$

$\bar{d}_{1y}=0$

$\bar{d}_2=\bar{n} \cdot \bar{d}_1 = \begin{vmatrix} \bar{i} & \bar{j} & \bar{k} \\ \bar{n}_i & \bar{n}_j & \bar{n}_k \\ \bar{d}_{1i} & \bar{d}_{1j} & \bar{d}_{1k} \end{vmatrix}$

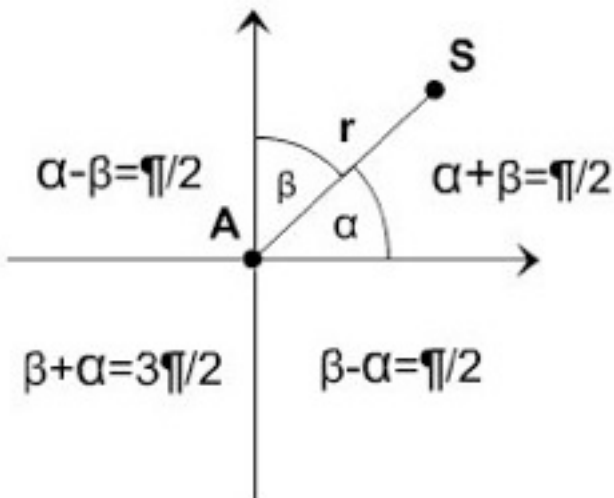
The simplest way is to rotate the plane's normal onto the plane to such a position where it is parallel to xz plane thus providing a fix axis. The other vector can be calculated by the vectorial multiplication of the normal and the previously created vector. If we have all this, then we only need to calculate the relative position of S.



$$\alpha = \overline{SA} \nabla \overline{d_1} \quad \cos \alpha = \frac{\overline{SA} \overline{d_1}}{|\overline{SA}| |\overline{d_1}|}$$

$$\beta = \overline{SA} \nabla \overline{d_2} \quad \cos \beta = \frac{\overline{SA} \overline{d_2}}{|\overline{SA}| |\overline{d_2}|}$$

We only need to determine the angles between SA vector, and the directional vectors, and we have to change into the planes A originated, 2 dimensional coordinate system.



$$x = \cos \alpha r$$

$$y = \cos \beta r$$

$$\text{if } \beta > \pi/2 \quad y^* = -1$$

PART 2. ALL THIS IN PRACTICE

Everything has been good and easy so far, but how the hell can we implement this? So, it's no easy matter, i suffered for 3 weeks, until i managed to structure the code in a simple and clean form.

We need two classes, the first - and the most important - is the Vector class, since we only use vectors during the projection, and the second is the Plane class, in which we can carry out other, plane-related functions.

But let's see the Vector class first. It must have all the used vector operations.

Vector Class Source Code :

```
//
```

```

// Vector Class
//
Vector = function (i, j, k)
{
    this.i = i;
    this.j = j;
    this.k = k;
    this.r = Math.sqrt(i*i+j*j+k*k);
};

//
// Standard Summa
//
Vector.prototype.Sum = function (ve)
{
    var i=this.i+ve.i;
    var j=this.j+ve.j;
    var k=this.k+ve.k;
    return new Vector(i,j,k);
};

//
// Standard Substraction
//
Vector.prototype.Sub = function (ve)
{
    var i=this.i-ve.i;
    var j=this.j-ve.j;
    var k=this.k-ve.k;
    return new Vector(i,j,k);
};

//
// Dot (vectorial) Multiplication of two vectors
// Standard 3x3 Matrix Determinant Calculation
//
Vector.prototype.DotMul = function(ve)
{
    var i = this.j*ve.k-this.k*ve.j;
    var j = -(this.i*ve.k-this.k*ve.i);
    var k = this.i*ve.j-this.j*ve.i;
    return new Vector(i, j, k);
};

//
// Scalar Multiplication of two Vectors
//
Vector.prototype.VectMul = function(ve)
{
    return this.i*ve.i+this.j*ve.j+this.k*ve.k;
};

//
// Multiplication with a Scalar
//
Vector.prototype.SkalMul = function(skal)
{
    var i = this.i*skal;
    var j = this.j*skal;
    var k = this.k*skal;
    return new Vector(i, j, k);
};

//
// Calculating angle between two vectors
//
Vector.prototype.Angle = function(ve)
{
    var vmult = this.i*ve.i+this.j*ve.j+this.k*ve.k;
    var rmult = this.r*ve.r;
    var alpha = Math.acos(Math.round(vmult)/Math.round(rmult));
    return alpha;
};

```

```
};
```

Plane Class Source Code

```
//  
// Vector Class  
//  
Vector = function (i, j, k)  
{  
  this.i = i;  
  this.j = j;  
  this.k = k;  
  this.r = Math.sqrt(i*i+j*j+k*k);  
};  
  
//  
// Standard Summa  
//  
Vector.prototype.Sum = function (ve)  
{  
  var i=this.i+ve.i;  
  var j=this.j+ve.j;  
  var k=this.k+ve.k;  
  return new Vector(i,j,k);  
};  
  
//  
// Standard Substraction  
//  
Vector.prototype.Sub = function (ve)  
{  
  var i=this.i-ve.i;  
  var j=this.j-ve.j;  
  var k=this.k-ve.k;  
  return new Vector(i,j,k);  
};  
  
//  
// Dot (vectorial) Multiplication of two vectors  
// Standard 3x3 Matrix Determinant Calculation  
//  
Vector.prototype.DotMul = function(ve)  
{  
  var i = this.j*ve.k-this.k*ve.j;  
  var j = -(this.i*ve.k-this.k*ve.i);  
  var k = this.i*ve.j-this.j*ve.i;  
  return new Vector(i, j, k);  
};  
  
//  
// Scalar Multiplication of two Vectors  
//  
Vector.prototype.VectMul = function(ve)  
{  
  return this.i*ve.i+this.j*ve.j+this.k*ve.k;  
};  
  
//  
// Multiplication with a Scalar  
//  
Vector.prototype.SkalMul = function(skal)  
{  
  var i = this.i*skal;  
  var j = this.j*skal;  
  var k = this.k*skal;  
  return new Vector(i, j, k);  
};  
  
//  
// Calculating angle between two vectors  
//
```

```
Vector.prototype.Angle = function(ve)
{
  var vmult = this.i*ve.i+this.j*ve.j+this.k*ve.k;
  var rmult = this.r*ve.r;
  var alpha = Math.acos(Math.round(vmult)/Math.round(rmult));
  return alpha;
};
```

And thats all folks! I've further tuned the final version to make things more spectacular, download it and examine it to your hearts content.

(real3d.swf)

(real3d fla)

[Article on gotoandplay.it](#)

2004.12.