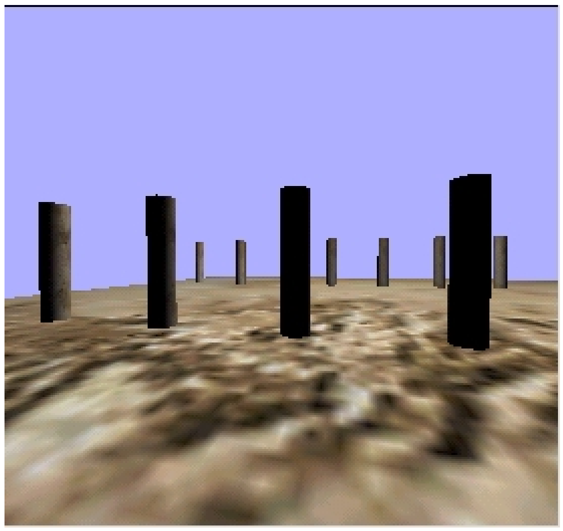
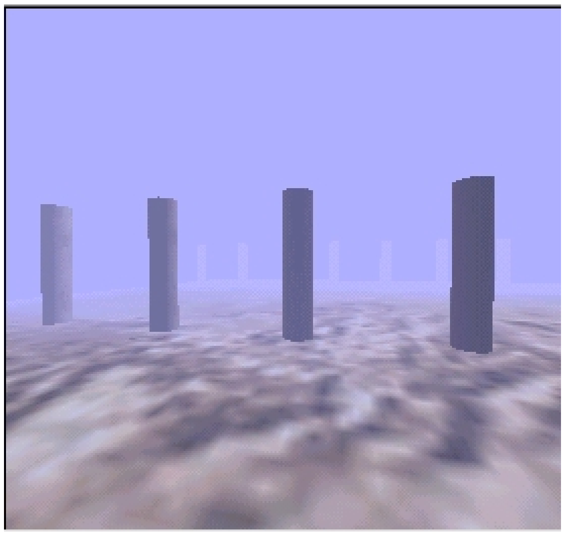
Implementing Fog in Direct3D

Douglas Rogers
NVIDIA Corportaion
drogers@nvidia.com

Fog is a special effect that is used to blend a scene into a predefined color to give the impression that "fog" is present in the scene.   Fog is usually implemented as it appears in the physical world. The farther objects are, the more they are obscured by fog. Objects that are close to the observer are clearer.



Unfogged Scene



Fogged Scene

Fog can also be useful to reduce the complexity in a scene. In the above right image, the pegs in the distance might not need to be rendered.

Fog Types

There are two ways to implement fog, vertex-based fog and table-based fog. In addition, there are additional attributes that control the application of fog.

- Distance Calculation (Range or Plane Based)
- Drop Off (Linear, Exponential or Exponential Squared)
- Z or W based

<u>Vertex Fog</u>

Using vertex fog, a fog coverage value is calculated for each vertex of a triangle, then interpolated within the triangle.

Table (Pixel) Fog

Table fog, also called pixel fog, is calculated independent of vertices. Ideally, fog is calculated on a pixel basis, but many device drivers emulate this feature in software with vertex fog values. It is called table fog because traditionally tables have been used to look up fog values.

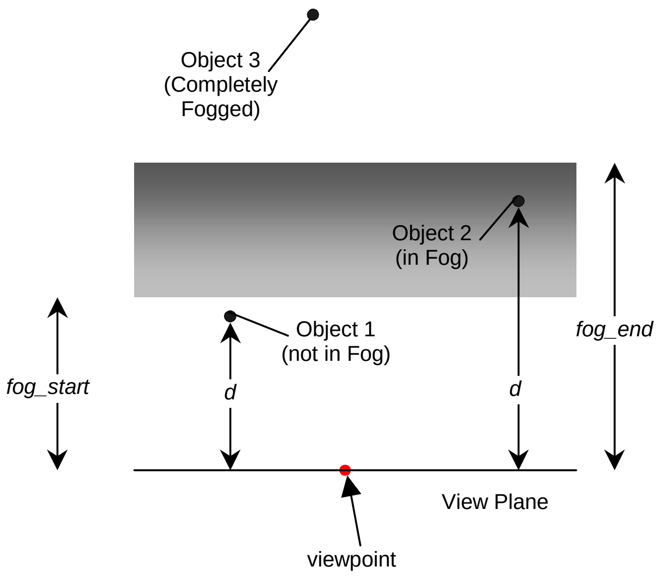
Fog Distance:

In addition to fog type, you must specify how the fog distance is calculated. The two methods for specifying fog distance are range and plane based. With linear fog (explained below), fog has a start (*fogstart)* and an end (*fogend).* Objects that are closer than *fogstart* have no fog applied, objects that are greater than *fogend* are completely fogged. When objects are in between, fog is applied by linearly interpolating from *fogstart* to *fogend.*

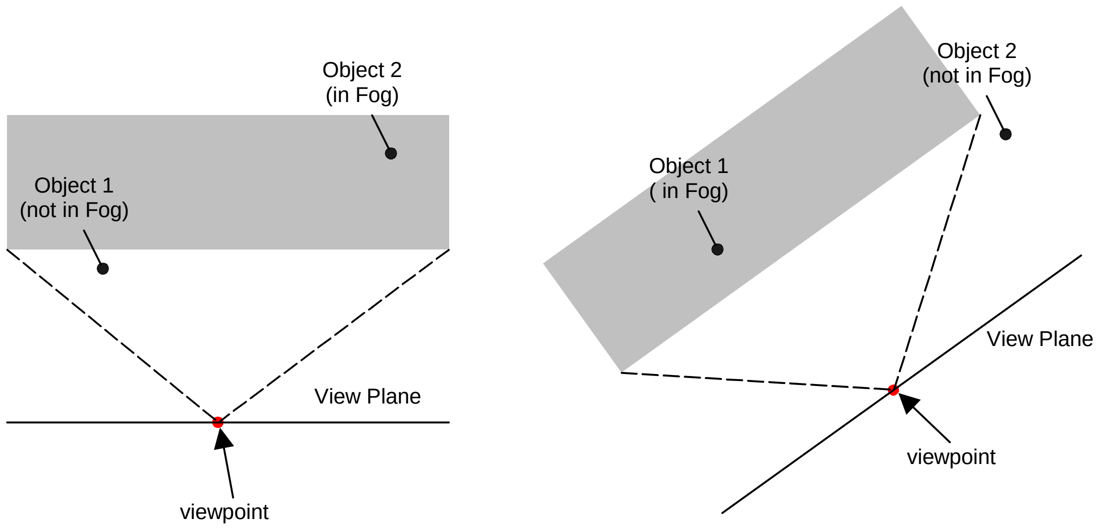
Plane Based Fog

Plane based fog is calculated as the distance from the *viewplane* to the object, shown here as *d.*

There can be unwanted artifacts when using plane based fog, however. When an observer rotates about a point, an object may jump in and out of the fog area. Notice that *object1* and *object2* reverse their presence in the fog area depending on the orientation of the observer.



Plane Based Fog

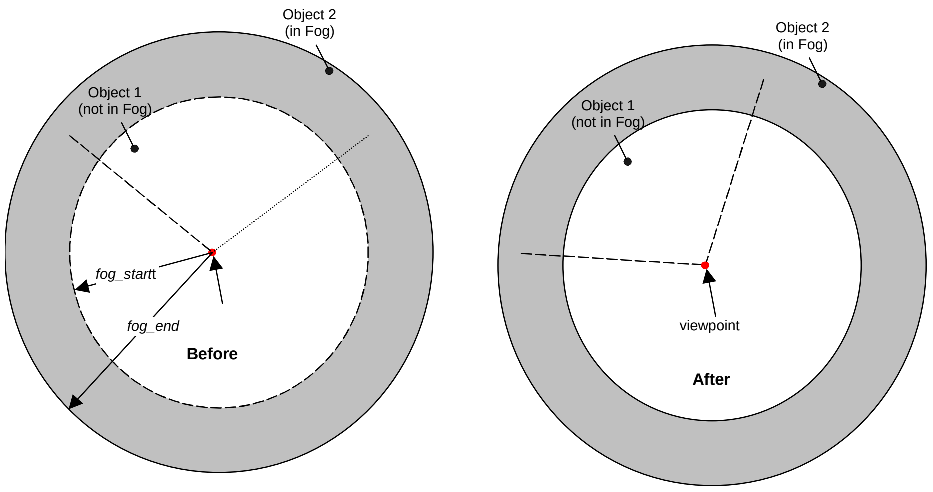


Before **After**

Plane Based Fog Artifact

Range Based Fog

Range based fog addresses the visual artifact that plane based fog introduces. Range based fog diminishes radially from a point, rather than linearly from a plane so objects do not enter and leave the fog area depending on the orientation of the observer. Range based fog is more expensive to calculate than plane based fog. **Range based fog cannot be used with table fog.**



Range Based Fog

You enable range based fog with the following call:

```
pd3dDevice->SetRenderState( D3DRENDERSTATE_RANGEFOGENABLE,, TRUE);
```

The following entry from D3DIM.DOC regarding range based fog may be misleading:

"If the current device supports range-based fog, it will set the D3DPRASTERCAPS_FOGRANGE capability flag in the **dwRasterCaps** member of the **D3DPRIMCAPS** structure when you call the **IDirect3DDevice3::GetCaps** method. To enable range-based fog, set the D3DRENDERSTATE_RANGEFOGENABLE render state to TRUE."

D3DPRASTERCAPS_FOGRANGE is not checked by D3D because the range is calculated in software from the matrices. So even if the D3DPRASTERCAPS_FOGRANGE capability bit is not set, you can enable range based fog because the range values are calculated in software.

Visibility Dropoff
You must specify how the fog visibility drops off with distance. There are three formulas to specify fog visibility drop-off. These are linear, exponential and exponential squared. The formula are indicated below:

LINEAR $f = \frac{fogend - d}{fogend - fogstart}$

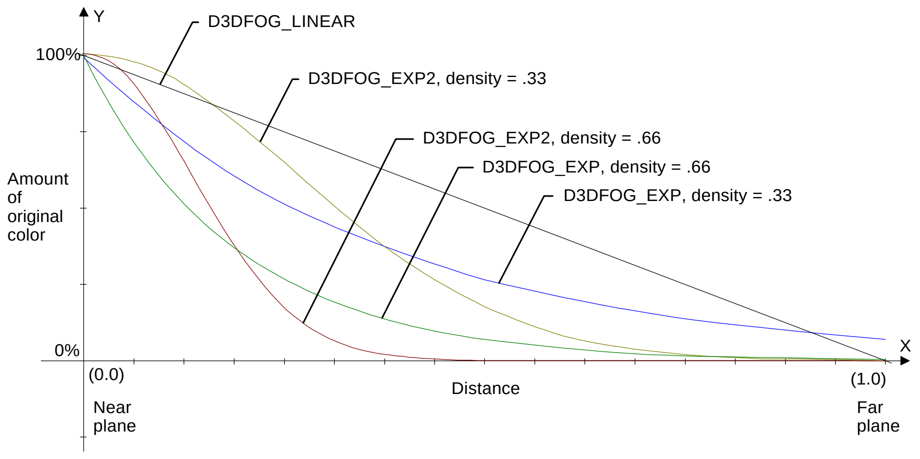
Linear fog interpolates from the *fogstart* to *fogend*. *d* is the distance from the observer to the object.

EXP $f = \frac{1}{e^{(d \times fogdensity)}}$

Exponental fog produces a rapid transition for fog. This is controlled only by *fogdensity* and distance, *fogstart* and *fogend* are not used.

EXP2 $f = \frac{1}{e^{(d \times fogdensity)^2}}$

Exponental fog produces a sharp transition for fog. This is controlled only by *fogdensity* and distance, *fogstart* and *fogend* are not used.



Fog Dropoff (from D3DIM.doc)

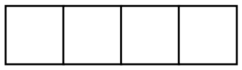
How Fog is Applied

Fog is turned on by enabling it with:

```
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGENABLE, TRUE );
```

The fog color that is applied is a 32 bit color value set with:

```
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGCOLOR,   fogcolor);
```



fogcolor

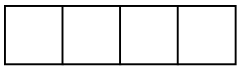
The formula for combining *fogcolor* with a pixel is:

$$pixel = f \cdot C + (1 - f) \cdot fogcolor \qquad \text{Eq. 1}$$

where *f* is the fog value, C is the pixel before fogging and pixel is the final pixel.
Note that when *f* = 1.0, no fog is applied; when *f* = 0.0, full fog is applied. This excludes the alpha blending of the color.

Vertex Fog

As mentioned before, vertex fog is applied on a per vertex basis. The fog applied to a vertex is specified in the 'alpha' field of the specular color component for the vertex. This *f* value is used the pixel fogging calculation Eq. 1. The value 1.0 is represented as 0xFF and means that there is no fog, only the pixel color *C*.



***specular color* (*f* = fog component)**

To enable vertex fog, table fog is set to NONE:

```
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEMODE,    D3DFOG_NONE );
```

<u>Performing Your Own Vertex Fogging Calculations</u>

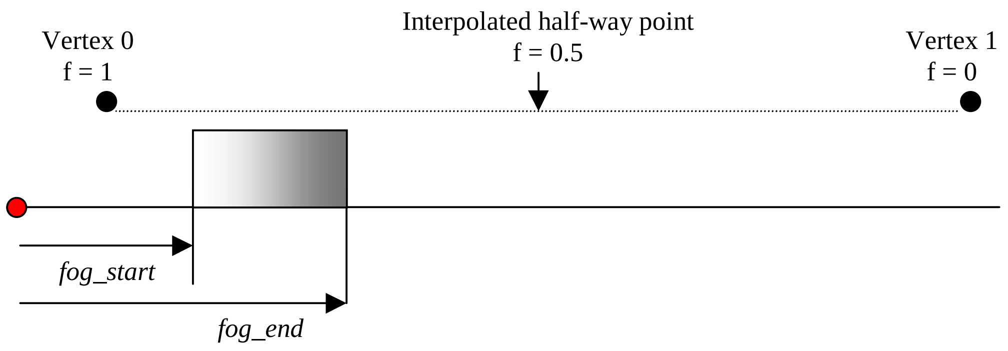
If you perform your own fogging calculations, you must calculate the fog intensity at each vertex. Place the fog value in the alpha component in the specular color. Turn off the vertex fog mode in the light state and the table mode fog in the render state.

```
// turn off all D3D fogging
pd3dDevice->SetLightState( D3DLIGHTSTATE_FOGMODE,  D3DFOG_NONE);
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEMODE,    D3DFOG_NONE );
```

The graphics hardware will linearly interpolate the fog values that you have specified, and apply the fog accordingly.

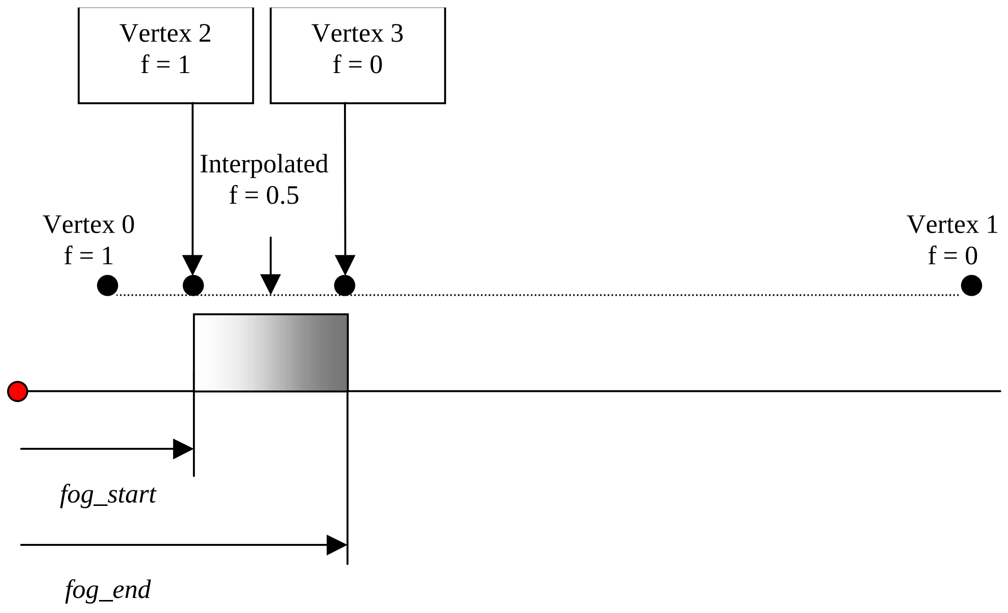
When performing your own fogging calculations, you must send the RHW value to the device driver. This value is used to perform perspectively correct fogging calculations.

Note: The interpolation is done without regard to *fogstart* and *fogend.* This may produce some unwanted visual problems. If your vertices straddle the fog range, linear interpolation may not be what you expect.



Unexpected Fog Values with Vertex Fog

Here vertex 0 is closer than the *fogstart,* so *f* is set to one (no fog). Vertex 1 is beyond the *fogend*, so *f* is set to zero (completely fogged). When this is interpolated, the halfway point will be half fogged, but this is not necessarily halfway into the fog region. Since you cannot specify fog values out side of the region zero and one, this generates visual artifacts. If you wish, you can solve this by clipping your objects to the fog planes and introducing new vertices at the *fogstart* and *fogend* boundaries. Below vertex two and vertex three are introduced at the fog boundaries to allow the vertex fog to be correctly interpolated.



Introducing Vertices to Correct Vertex Fog Interpolation Errors

You can tesselate your geometry to a greater degree to reduce this problem, too.

Using D3D to Perform Vertex Fogging

If you are using Direct3D to perform vertex fogging, it is controlled through the light state interface. D3D will calculate the fog values relative to *fogstart* and *fogend* and place them in the fog component of the specular color. D3D currently only supports linear based vertex fog so you must turn off table mode fog, specify linear vertex fog and set the vertex *fogstart* and *fogend* values. *Fogdensity* is only used for exponential fog mode so does not need to be specified for linear fog.

```
For DirectX 6.0
// disable table fog
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEMODE,    D3DFOG_NONE );

// select linear vertex fog
pd3dDevice->SetLightState( D3DLIGHTSTATE_FOGMODE, D3DFOG_LINEAR);
pd3dDevice ->SetLightState(D3DLIGHTSTATE_FOGSTART,        *(DWORD *)&vertex_fogstart);
pd3dDevice ->SetLightState(D3DLIGHTSTATE_FOGEND,        *(DWORD *)&vertex_fogend);
// not currently used
pd3dDevice ->SetLightState(D3DLIGHTSTATE_FOGDENSITY,    *(DWORD *)&vertex_fogdensity );
```

D3DLIGHTSTATE_FOGSTART, D3DLIGHTSTATE_FOGSTART and D3DLIGHTSTATE_FOGDENSITY require floating point values, but the interface to SetLightState is a DWORD value, so we cast the parameter to a DWORD.

For DirectX 7.0, a vertex based fog renderstate has been added.

```
pd3dDevice->SetRenderState(D3DRENDERSTATE_FOGVERTEXMODE,  D3DFOG_LINEAR);
pd3dDevice->SetRenderState(D3DRENDERSTATE_FOGSTART, *(DWORD *)&vertexfogstart);
```

```
pd3dDevice->SetRenderState(D3DRENDERSTATE_FOGEND, *(DWORD *)&vertexfogend);
  pd3dDevice->SetRenderState(D3DRENDERSTATE_FOGDENSITY, *(DWORD *)&vertexfogdensity);
```

Because D3D does not introduce extra vertices and fog values cannot be less than zero or greater than one, vertex fogging will suffer from the interpolation problem described above when using D3D's linear vertex fogging calculations.

D3D Code to set the vertex fog values:

```
// computes range or plane distance
D3DVALUE ComputeDistance(D3DVECTOR &v)
{
    if (dwFlags & D3DPV_RANGEBASEDFOG)
   {
        D3DVALUE x,y,z;
        x = v.x * m._11 + v.y * m._21 + v.z * m._31 + m._41;
        y = v.x * m._12 + v.y * m._22 + v.z * m._32 + m._42;
        z = v.x * m._13 + v.y * m._23 + v.z * m._33 + m._43;
        return SQRTF(x * x + y * y + z * z);
   }
   else
       return v.x * m._13 + v.y * m._23 + v.z * m._33 + m._43;
}



#define D3DFE_SET_ALPHA(color, a) ((char*)&color)[3] = (unsigned char)a;

fogfactor = 255.0 / (fogend - fogstart);

// return value from ComputeDistance is passed to FOG_VERTEX

inline void FOG_VERTEX(D3DVALUE d)
{
   int f;

   if (d < fogstart)
      D3DFE_SET_ALPHA(Specular, 255)
   else
   if (d >= fogend)
      D3DFE_SET_ALPHA(Specular, 0)
   else
   {
      D3DVALUE v = (fogend - d) * fogfactor;
      f = FTOI(v);
      D3DFE_SET_ALPHA(Specular, f)
   }
}
```

Table (Pixel) Fog

Table fog parameters are set through the SetRenderState interface. Range based fog is not supported when using table mode and must be disabled. Disable vertex fog as well. The RIVA 128, RIVA128ZX and the RIVA TNT all emulate table fog using vertex based fog. The GeForce 256 supports table fog in hardware.
To use table based fog, you must select which drop off method to use linear, exponential or exponential squared. Table fog cannot be combined with range based fog.

```
// disable vertex fog
pd3dDevice->SetLightState(  D3DLIGHTSTATE_FOGMODE,    D3DFOG_NONE );

// must disable range based fog
pd3dDevice->SetRenderState( D3DRENDERSTATE_RANGEFOGENABLE,  FALSE );
//
// TableFogMode =   D3DFOG_EXP or   D3DFOG_EXP2 or   D3DFOG_LINEAR
//
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEMODE,    TableFogMode );

// table)fogstart and table_fogend are not used with exponential fog
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLESTART, *(DWORD *)&table_fogstart);
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEEND, *(DWORD *)&table_fogend);

// table_fogdensity  is not used with linear table fog
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEDENSITY, *(DWORD *)&table_fogdensity);
```

Z based or W based Fog using D3D fogging.

When a vertex is passed to D3D to be transformed, the projection matrix is used. Based on this matrix, screen coordinates and a homogeneous W are created. The screen *Z* (*sz)* coordinate has a range from zero to one and *W* is specified (in eye space) as a scaled distance to the observer. For more information about projection matrices, see my papers "W-Buffering" and " Z-buffering, Interpolation and More W-buffering".

Z Based Fog

When the fog distance is calculated by D3D, the projection matrix is examined and a determination is made whether to use Z based or W based fog. If the projection matrix is affine, then Z based fog is calculated otherwise W based fog is implemented.

$$\begin{bmatrix} - & - & - & 0 \\ - & - & - & 0 \\ - & - & - & 0 \\ - & - & - & 1 \end{bmatrix}$$

Affine Matrix
('-' is don't care.)

The projection matrix is determined to be affine if the fourth column is [0, 0, 0, 1]. When this occurs, Z based fog is used and the values for *fogstart* and *fogend* range from [0, 1].

W Based Fog

Normally, you will want to use W based fog with a standard projection matrix. However, there are some restrictions on the projection matrix that you provide to D3D to use for your transformations.

If the projection matrix includes a (3,4) coefficient that is not 1, you must scale all coefficients by the inverse of the (3,4) coefficient to produce a compatible matrix. If you don't provide a compliant matrix, fog effects will not be applied correctly. Theoretically, the capability D3DPRASTERCAPS_WFOG must be available to use this mode, but it does not appear that it is checked in D3D, so any non-affine projection matrix should use W fog.

$$M = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & Q & s \\ 0 & 0 & -Qz_n & 0 \end{bmatrix}$$

Incorrect Projection Matrix for Accurate W Based Fog

$$M' = \begin{bmatrix} c/s & 0 & 0 & 0 \\ 0 & c/s & 0 & 0 \\ 0 & 0 & Q/s & 1 \\ 0 & 0 & -Qz_n/s & 0 \end{bmatrix}$$

Correct Projection Matrix for Accurate W based Fog

The reason to scale the matrix is so the W value is actual distance to the object. Using matrix *M*, W = s * Z, not W = Z which is what we want.

When you use W based fog, *fogend* and *fogstart* are specified in world coordinates.

Fog Index Calculation

Computing the distance *d* that is used in the above calculations, in D3D, is basically:

```
Float d, Wnear, Wfar;

// m is the projection matrix
WNear = m._44 - m._43 / m._33 * m._34;
WFar  = (m._44 - m._43)  /  (m._33 - m._34) * m._34 + m._44;

 if  (( 1.0f == Wnear )  &&  ( 1.0f == Wfar ) )   // affine matrix
{
    // use clamped Z for affine projection
   d = sz; // screen z clamped to [znear, zfar]
}
else     // use W for non-affine projection
   d = 1.0/ RHW;  // reciprocal of homogeneous W
```

Example

As an example of how to set the fog parameters, here is the code from the MFCFOG example.

```
inline DWORD FtoDW( FLOAT f ) { return *((DWORD*)&f); }

pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGENABLE, TRUE );
pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGCOLOR,  g_dwFogColor );

in DirectX 6.0
if( g_bUsingTableFog )
{
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLESTART,   FtoDW(g_fFogStart) );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEEND,     FtoDW(g_fFogEnd) );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEDENSITY, FtoDW(g_fFogDensity) );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEMODE,    g_dwFogMode );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGMODE,         D3DFOG_NONE );
}
else
{
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGSTART,       FtoDW(g_fFogStart) );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGEND,         FtoDW(g_fFogEnd) );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGMODE,        g_dwFogMode );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_RANGEFOGENABLE, g_bRangeBasedFog );
   pd3dDevice->SetRenderState( D3DRENDERSTATE_FOGTABLEMODE,   D3DFOG_NONE );
 }
```

This is table of the possible fog modes available in D3D and the parameters required to achieve the mode. Use this table when D3D is perform the fogging calculations.

Direct3D Calculated Fog

Lighting Mode / Parameters	Vertex Z based Plane	Vertex Z based Range	Vertex W based Plane	Vertex W based Range		Table Z based Plane	Table Z based Range	Table W based Plane	Table W based Range
D3DLIGHTSTATE_FOGMODE	LINEAR	LINEAR	LINEAR	LINEAR		NONE	NONE	NONE	NONE
D3DLIGHTSTATE_FOGSTART	[0, 1]	[0, 1]	world	World		-	-	-	-
D3DLIGHTSTATE_FOGEND	[0, 1]	[0, 1]	world	World		-	-	-	-
D3DLIGHTSTATE_FOGDENSITY [2]	n/u	n/u	n/u	n/u		-	-	-	-
D3DRENDERSTATE_FOGTABLEMODE[3]	NONE	NONE	NONE	NONE		LINEAR, EXP or EXP2	LINEAR, EXP or EXP2	LINEAR, EXP or EXP2	LINEAR, EXP or EXP2
D3DRENDERSTATE_FOGTABLESTART	-	-	-	-		[0, 1]	[0, 1]	world	world
D3DRENDERSTATE_FOGTABLEEND	-	-	-	-		[0, 1]	[0, 1]	world	world
D3DRENDERSTATE_FOGTABLEDENSITY	-	-	-	-		[0, 1]	[0, 1]	[0, 1]	[0, 1]
D3DRENDERSTATE_RANGEFOGENABLE[7]	FALSE	TRUE	FALSE	TRUE		FALSE	TRUE	FALSE	TRUE
Perspective Projection Matrix and D3DPRASTERCAPS_WFOG[6]	No	No	Yes	Yes		No	No	Yes	Yes
Notes:	5, 9	5, 9	4, 9	4, 9		5, 8	1	4, 8	1

Notes:
1. Direct3D performs range-based fog only when using vertex fog with the Direct3D transformation and lighting engine. This is not a legal mode.
2. D3DLIGHTSTATE_FOGDENSITY is not used because the only vertex fog mode is linear which does not use the density component.
3. Setting D3DRENDERSTATE_FOGTABLEMODE to D3DFOG_NONE enables vertex fog.
4. W based fog. Projection matrix must be normalized about the entry (3, 4). See W Based Fog above. Fog parameters are in world coordinates.
5. Z based fog. Fog parameters range between zero and one.
6. It does not appear that D3DPRASTERCAPS_WFOG is checked for by D3D, it may actually be forced to enabled. No means an affine matrix
7. D3DPRASTERCAPS_FOGRANGE is not checked by D3D because all the vertex range calculations are done in software and set in the vertex.
8. D3DPRASTERCAPS_FOGTABLE capability must exist.
9. D3DPRASTERCAPS_FOGVERTEX capability must exist.

Key:

[0, 1]	values range from zero to one, inclusive
world	values are specified in world coordinates
n/u	not used in the current implementation.

Application Calculated Fog

Lighting Mode / Parameters	Vertex		Table Z based Plane	Table Z based Range	Table W based Plane	Table W based Range
D3DLIGHTSTATE_FOGMODE	NONE		NONE	NONE	NONE	NONE
D3DLIGHTSTATE_FOGSTART	-		-	-	-	-
D3DLIGHTSTATE_FOGEND	-		-	-	-	-
D3DLIGHTSTATE_FOGDENSITY	-		-	-	-	-
D3DRENDERSTATE_FOGTABLEMODE	NONE		LINEAR, EXP or EXP2	LINEAR, EXP or EXP2	LINEAR, EXP or EXP2	LINEAR, EXP or EXP2
D3DRENDERSTATE_FOGTABLESTART	-		[0, 1]	[0, 1]	world	world
D3DRENDERSTATE_FOGTABLEEND	-		[0, 1]	[0, 1]	world	world
D3DRENDERSTATE_FOGTABLEDENSITY	-		[0, 1]	[0, 1]	[0, 1]	[0, 1]
D3DRENDERSTATE_RANGEFOGENABLE$_7$	-		FALSE	TRUE	FALSE	TRUE
Perspective Projection Matrix and D3DPRASTERCAPS_WFOG$_2$	-		No	No	Yes	Yes
Notes:	1		3	3, 4	3	3, 4

Notes:

1. Fog values are placed in the alpha component of the specular color and linearly interpolated. RHW must be set for the vertex. Range and distance calculations have already been made.
2. You must still set the perspective projection matrix so that Wnear and Wfar are initialized.
3. This is emulated with vertex fog in the RIVA128, RIVA128ZX and RIVATNT
4. Range Based Table Fog is not supported