Neutral File Format,

by Eric Haines, Autodesk, 1050 Craft Road, Ithaca, NY 14850
email: erich@acm.org

Draft #1, 10/3/88
Version 2.7, as of 5/22/90 - added information on hither, light color
Version 3.0, 12/17/90 - minor information changes
Version 3.1, 11/2/92 - more minor information changes
Version 3.9, 12/4/96 - minor changes in introduction

The NFF (Neutral File Format) is designed as a minimal scene description language. The language was designed in order to test various rendering algorithms and efficiency schemes. It is meant to describe the geometry and basic surface characteristics of objects, the placement of lights, and the viewing frustum for the eye. Some additional information is provided for aesthetic reasons (such as the color of the objects, which is not strictly necessary for testing the efficiency of rendering algorithms).

At present the NFF file format is used in conjunction with the SPD (Standard Procedural Database) software, a package designed to create a variety of databases for testing rendering schemes. For more information about SPD see "A Proposal for Standard Graphics Environments," IEEE Computer Graphics and Applications, vol. 7, no. 11, November 1987, pp. 3-5. See IEEE CG&A, vol. 8, no. 1, January 1988, p. 18 for the correct image of the tree database (the only difference is that the sky is blue, not orange).

The SPD package (along with images of the databases) is available from:

        http://www.acm.org/tog/resources/SPD/

NFF is meant for testing efficiency schemes and so has minimal support for lighting and shading. For a reasonable scene description language, see VRML.

By providing a minimal interface, NFF is meant to act as a simple format to allow the programmer to quickly write filters to move from NFF to the local file format. Presently the following entities are supported:

```
     A simple perspective frustum
     A background color description
     A positional (vs. directional) light source description
     A surface properties description
     Polygon, polygonal patch, cylinder/cone, and sphere descriptions
```

Files are output as lines of text. For each entity, the first field defines its type. The rest of the line and possibly other lines contain further information about the entity. Entities include:

```
"v"  - viewing vectors and angles
"b"  - background color
"l"  - positional light location
"f"  - object material properties
"c"  - cone or cylinder primitive
"s"  - sphere primitive
"p"  - polygon primitive
"pp" - polygonal patch primitive
```

These are explained in depth below.

Viewpoint location. Description:

```
    "v"
    "from" Fx Fy Fz
    "at" Ax Ay Az
    "up" Ux Uy Uz
    "angle" angle
    "hither" hither
    "resolution" xres yres
```

Format:

```
    v
    from %g %g %g
    at %g %g %g
    up %g %g %g
    angle %g
    hither %g
    resolution %d %d
```

The parameters are:

```
    From:  the eye location in XYZ.
    At:    a position to be at the center of the image, in XYZ world
           coordinates.  A.k.a. "lookat".
    Up:    a vector defining which direction is up, as an XYZ vector.
    Angle: in degrees, defined as from the center of top pixel row to
           bottom pixel row and left column to right column.
    Hither: distance of the hither plane (if any) from the eye.  Mostly
           needed for hidden surface algorithms.
    Resolution: in pixels, in x and in y.
```

Note that no assumptions are made about normalizing the data (e.g. the from-at distance does not have to be 1). Also, vectors are not required to be perpendicular to each other.

For all databases some viewing parameters are always the same:
Yon is "at infinity."
Aspect ratio is 1.0.

A view entity must be defined before any objects are defined (this requirement is so that NFF files can be displayed on the fly by hidden surface machines).

Background color. A color is simply RGB with values between 0 and 1:

```
    "b" R G B
```

Format:

```
    b %g %g %g
```

If no background color is set, assume RGB = {0,0,0}.

Positional light. A light is defined by XYZ position. Description:

```
    "l" X Y Z [R G B]
```

Format:

```
    l %g %g %g [%g %g %g]
```

All light entities must be defined before any objects are defined (this requirement is so that NFF files can be used by hidden surface machines). Lights have a non-zero intensity of no particular value, if not specified (i.e. the program can determine a useful intensity as desired); the red/green/blue color of the light can optionally be specified.

Fill color and shading parameters. Description:

```
    "f" red green blue Kd Ks Shine T index_of_refraction
```

Format:

```
    f %g %g %g %g %g %g %g %g
```

RGB is in terms of 0.0 to 1.0.

Kd is the diffuse component, Ks the specular, Shine is the Phong cosine power for highlights, T is transmittance (fraction of contribution of the transmitting ray). Usually, 0 <= Kd <= 1 and 0 <= Ks <= 1, though it is not required that Kd + Ks == 1. Note that transmitting objects (T > 0) are considered to have two sides for algorithms that need these (normally objects have one side).

The fill color is used to color the objects following it until a new color is assigned.

Objects: all objects are considered one-sided, unless the second side is needed for transmittance calculations (e.g. you cannot throw out the second intersection of a transparent sphere in ray tracing).

Cylinder or cone. A cylinder is defined as having a radius and an axis defined by two points, which also define the top and bottom edge of the cylinder. A cone is defined similarly, the difference being that the apex and base radii are different. The apex radius is defined as being smaller than the base radius. Note that the surface exists without endcaps. The cone or cylinder description:

```
    "c"
    base.x base.y base.z base_radius
    apex.x apex.y apex.z apex_radius
```

Format:

```
    c
    %g %g %g %g
    %g %g %g %g
```

A negative value for both radii means that only the inside of the object is visible (objects are normally considered one sided, with the outside visible). Note that the base and apex cannot be coincident for a cylinder or cone. Making them coincident could be used to define endcaps, but none of the SPD scenes currently make use of this definition.

Sphere. A sphere is defined by a radius and center position:

```
    "s" center.x center.y center.z radius
```

Format:

```
    s %g %g %g %g
```

If the radius is negative, then only the sphere's inside is visible (objects are normally considered one sided, with the outside visible). Currently none of the SPD scenes make use of negative radii.

Polygon. A polygon is defined by a set of vertices. With these databases, a polygon is defined to have all points coplanar. A polygon has only one side, with the order of the vertices being counterclockwise as you face the polygon (right-handed coordinate system). The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined by using just the first three vertices. Description:

```
    "p" total_vertices
    vert1.x vert1.y vert1.z
    [etc. for total_vertices vertices]
```

Format:

```
    p %d
    [ %g %g %g ] <-- for total_vertices vertices
```

Polygonal patch. A patch is defined by a set of vertices and their normals.

With these databases, a patch is defined to have all points coplanar. A patch has only one side, with the order of the vertices being counterclockwise as you face the patch (right-handed coordinate system). The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined. Description:

```
    "pp" total_vertices
    vert1.x vert1.y vert1.z norm1.x norm1.y norm1.z
    [etc. for total_vertices vertices]
```

Format:

```
    pp %d
    [ %g %g %g %g %g %g ] <-- for total_vertices vertices
```

Comment. Description:

```
    "#" [ string ]
```

Format:

```
    # [ string ]
```

As soon as a "#" character is detected, the rest of the line is considered a comment.