1994

NPSOFF: An Object Description Language for Supporting Virtual World Constructio

Zyda, Michael J.

Zyda, Michael J., Wilson, Kalin P., Pratt, David R., Monahan, James G. and Falby, John S.

þÿ NPSOFF: An Object Description Language for Supporting Virtual World

In *Computer & Graphics*, Vol. 17, No. 4, pp. 457-464.

NPSOFF: An Object Description Language for Supporting Virtual World Construction

Michael J. Zyda*, Kalin P. Wilson, David R. Pratt,
James G. Monahan and John S. Falby

Naval Postgraduate School
Department of Computer Science
Monterey, California 93943-5100 USA
Email: zyda@trouble.cs.nps.navy.mil
*contact author

Keywords: graphics description languages, virtual world construction, virtual worlds.

Abstract

The development of interesting virtual world systems requires the modeling of many different, usually complex, graphical objects. How these objects are represented in the system plays a major part in determining the capabilities of the system. We present an efficient, flexible and extensible object description language used in virtual world system development at the Naval Postgraduate School.

Introduction

Research in the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School concentrates on real-time, 3D virtual worlds [1][2]. These worlds are typically populated with a large number of graphical objects, usually consisting of several hundred polygons and several lighting materials and textures. During research on early systems it became increasingly clear that we needed an application independent method to describe, manipulate and manage objects. We needed to extend the immediate mode IRIS GL^{TM} [3] to allow storage of 3D objects while maintaining a close tie to the library to take advantage of its flexible and efficient interface to the IRIS graphics workstation. Whereas systems like PHIGS+ [4] and GKS [5] treat objects as closed lists, we wanted to have easy access to our objects, to treat them as first-class 3D objects. Unlike GL, Starbase^{TM} [6] and RenderMan^{TM} [7] which have no notion of retained objects (other than closed display lists in the case of GL and Starbase), we wanted objects to be retained for future use. However, we did not want to proliferate multiple databases of objects for different applications. An in-house file format was developed that allowed the storage and retrieval of our 3D objects. This file format was the basis of NPSOFF.

NPSOFF: Overview

NPSOFF provides a powerful application-independent method for describing graphical objects which is easy to use and extensible. Modeling and prototyping of objects is efficient. NPSOFF is tightly coupled to the rendering hardware and software of the Silicon Graphics Inc. IRIS workstation, helping to facilitate the efficiency of our virtual world applications. The language system consists of “tokens” that represent primitive graphical concepts. These tokens are combined in an ASCII file to represent an object. The object can then be referenced by an application in an abstract manner without needing to know the details of how the object is composed. The level of abstraction that NPSOFF provides offers numerous advantages that are discussed below. Objects can have varying levels of complexity to represent a wide range of graphical objects and environments. NPSOFF also serves as a standard for application development. This makes supporting general purpose tools plausible and extremely useful.

Functional Description

NPSOFF is implemented in C on Silicon Graphics Inc. IRIS workstations. The tokens that comprise NPSOFF generally represent IRIS GL functions.

The NPSOFF language is easy to read and understand. Since NPSOFF objects are stored in ASCII files, the only tool needed to make or edit an NPSOFF object is a text editor. Also, the ASCII format allows off-line editing independent of an application. The ASCII format also provides a level of device independence. NPSOFF objects can be shared between machines that implement the standard IRIS GL library. Objects are filtered and tokens which are not supported on the current machine are disregarded.

In early versions of the language, tokens corresponded almost one for one to GL functions. Later versions added more abstraction and flexibility. The language tokens simplify the interface to the GL library by labeling components and encapsulating some of its complexity.

NPSOFF extends the GL interface by allowing many system settings to be named. Naming system definitions allows us to build libraries of commonly used settings like materials and textures which can be shared across applications.

NPSOFF tokens belong to one of three categories: definition tokens, display or execution tokens, or characteristics/composition tokens. Definition tokens define graphics system settings:. lights (normal and spot), lighting models (normal and two-sided), materials, textures, and colors. Definition tokens are named to allow later access. Figure 1.0 shows the definition of a light, lighting model, material and texture.

Display or execution tokens make up the bulk of NPSOFF and represent a change in graphics system state or graphics primitives. They are stored in a sequential display list in the order that they appear in an NPSOFF file. Example tokens that change the system state are: setmaterial, setlight, settexture. Each token has a name argument that corresponds to an earlier definition. In the case of setlight, the named light is associated with one of seven possible light numbers [3]. These state tokens make it easy to manipulate the graphics pipeline. Complex lighting and shading effects can be accomplished with NPSOFF in a simple and straightforward manner.

The graphics primitives which can be changed by NPSOFF are: polygons, surfaces (polygon with vertex normals), triangular meshes and lines. Figure 2.0 shows the general format for the graphics primitive token definitions. Additional display tokens perform manipulations of the system matrix stack, providing a means to transform NPSOFF objects and components of objects within the object definition file itself. The tokens that change the system stack are loadmatrix, multmatrix, pushmatrix, popmatrix, rotate, scale and translate.

The third category of NPSOFF tokens allows the user to define object characteristics and composition. They allow a high level of abstraction and support complex graphics techniques. Two of the main abstractions are composite objects and polygon decaling. Objects can be named and may contain nested object definitions. The nested definitions can contain any display tokens. This structure allows multiple objects to be treated as a single object. It also minimizes the duplication of primitive definitions. Objects are defined with the ***defobject*** token and displayed with the ***callobjec*****t** token. This structure is flexible and useful. Figure 3.0 shows a composite object made of three copies of a previously defined unit cube. The resulting object will be barbell shaped.

Non-graphical object characteristics tokens are the subject of current and future research. We discuss these tokens below.

Figure 4.0 shows a typical NPSOFF object definition file of a cube with side length 10 units and texture coordinates. While this object definition uses utilizes only a small subset of the capabilities of NPSOFF, it illustrates that the language is simple, straightforward and easy to read. The KISS principle is very important here.

Using an NPSOFF object is also simple. Essentially the user needs to use only three function calls to access and display an object. There are many more programmatic entry points to NPSOFF but many of them deal with in-memory manipulation that is not needed for standard use. They are used primarily by tools that build or manipulate NPSOFF objects. Figure 5.0 shows a typical use of an NPSOFF object

Physical Modeling Support

In the past, simulations developed in the Graphics and Video Laboratory have each handled physically-based modelling (PBM) independently and internally. The latest extension of the NPSOFF system is an object-based PBM system [8]. These enhancements give NPSOFF objects physical characteristics and provide mechanisms to control an object's motion given a list of internal and external forces on the object. Objects are handled in an enclosed reference called the "environment". All objects that participate in the NPSOFF PBM system are members of the environment.

The NPSOFF PBM system models object rigid-body dynamics using a Newtonian framework. Forces are specified as either local to an object or global to all objects. Local forces act in a relative collision frame of reference generating tangential and radial force components that affect an object. Global forces act on an object's center of mass and are added after local forces.

An object can be given many physical properties using the ***defphysics*** token. These properties include the object's initial location and location constraints in the environment, initial orientation and orientation constraints, initial linear and angular velocities and constraints on each, the object's mass and center of mass, the object's ability to absorb forces (elasticity), the dimensions of a bounding volume and a local viewpoint for the object. Each object can also use its own system of measurement. The ***defunits*** token allows the user to specify the units of measurement for dimensions, force magnitude and mass. This capability was incorporated to accommodate the use of object models from various sources. The PBM system uses reasonable constant or calculated defaults for all physical characteristics so none of the properties is required to be present when object physical characteristics are defined.

Forces are defined and added to an object's force list with the ***defforce*** token. Two types of forces are supported: deforming and non-deforming. Deforming forces are used for object explosions and bending. Non-deforming forces are used to alter an object's linear and angular velocities. Forces can be specified as awake or asleep. This allows the selective application of previously defined forces. The characteristics of a force defined with ***defforce*** are: type (deforming/non-deforming), origin relative to object center and origin constraints, force direction vector, magnitude and magnitude constraints and force state (asleep/awake).

The run-time interface of the NPSOFF PBM system is simple and flexible. Once the PBM environment has been initialized, the user can add or delete objects from the environment, add and modify global forces, modify object physical characteristics, add and modify object force characteristics and modify object and force states. The environment is updated by iterating over each object's force list. Each deforming force is applied, if present, otherwise each non-deforming force is applied. After local forces are applied, global forces are applied to an object. Finally the result of all applied forces is used to update an object's physical characteristics. To display the objects, the user can ask the PBM system for a transformation matrix to be applied to the system stack. This matrix is multiplied onto the system stack and then the object is displayed.

The NPSOFF PBM system adds rigid-body dynamics to NPSOFF objects. It provides us with a simple environment to model object dynamics and interaction. This is our first step to add more physical reality to our applications. An example NPSOFF object definition file showing physical tokens and a C code fragment showing part of the run-time interface are given in Figures 6.0 and 7.0.

Advantages

NPSOFF provides many advantages to the researchers in the NPS Graphics and Video Laboratory.

1. NPSOFF allows an application independent description of graphical objects. Objects can change without application recompilation. Objects can be designed and maintained by general purpose tools. Collections of objects can be built and shared with other researchers.

2. NPSOFF adds a level of abstraction that greatly simplifies application development. Beginning users can quickly master the use of NPSOFF because it has a simple syntax and interface. Interface to GL functions is managed by the language system for the user. Also, by having a large collection of common objects, developers can concentrate on how objects should be used rather than designing and rendering the objects.

3. NPSOFF provides a simple, object based, run-time interface to an object. Functions such as read_object(), display_object() and delete_object() all operate on individual objects in memory.   Many functions are provided so flexible manipulations are possible.

4. The stand-alone, reusable nature of NPSOFF objects encourages the use of common libraries of definition tokens. For example, consider a single object that is composed of only ***defmaterial*** tokens (a material object). This object (file) would act as a single source of material definitions. Other object definitions would not need to define materials that exist in the material object. Only setmaterial tokens would be needed in most definitions. The user would then read and ready_for_display the material object before any other objects. Other items with definition tokens, such as textures and lights, lend themselves to this concept as well.

Support Tools

The wide use of NPSOFF in our laboratory has led to a variety of tools to aid in the design and maintenance of NPSOFF objects. These tools include: The OFF calculator, NPSME - a material editor, NPSTE - a texture editor, NPSICON - a model builder and The OFF Mover Tool - a physically based design editor.

The OFF calculator allows in memory manipulation of NPSOFF objects using a simple command line interface. Using the OFF calculator, objects can be transformed (transformation applied to all primitives), primitives can be added to an object, graphical objects (spheres, boxes, etc.) can be added to an object, objects can be concatenated and translation between different units of measurement can be accomplished. The OFF calculator enables the user to quickly and easily make modifications to an object that affect the entire object.

NPSME is a material editor that helps manage libraries of materials. Material definitions can be selected from the library for viewing and editing. The material that is being edited is displayed on an object. Each of the material components can be manipulated for the desired effect. The edited material can be saved to the library of materials and the library saved to an NPSOFF file. The material editor helps us to maintain a large collection of material definitions used by NPSOFF objects in our applications. The ability to interactively design and modify material definitions is very important to rapid application development.

NPSTE is a texture editor. It helps to manage libraries of NPSOFF texture definitions. See Figure 1.0 for an example texture definition. NPSTE can use images in many formats as textures. Portions of an image can be copied and used as a texture image. Textures can be viewed on any NPSOFF object using either the texture coordinates specified in the object or automatically generated coordinates. Textures can be edited using a simple pixel editor. Finally, a texture definition can be saved in a library of textures and the library saved as an NPSOFF file. The texture editor lets developers interactively create, select and view textures independent of a developing application. Having a collection of common textures that a developer can select from has obvious advantages also.

NPSICON is an interactive object design tool that allows a developer to design or modify NPSOFF objects using a set of predefined building blocks. It is designed to be used primarily to build vehicular models. Objects can be edited and transformed in many ways and then saved to an NPSOFF file. NPSICON allows rapid prototyping of vehicular objects for use in applications. It also allows developers to modify existing models quickly and easily.

The “OFF Mover Tool” provides an environment in which users can design and test physical dynamics of NPSOFF objects. Mover reads in an object, assigning default values for any physical characteristics not present. The user can then adjust all physical characteristics of the object. Forces can be defined and added to an object’s force list using interactive controls. Once the object’s initial conditions, constraints and characteristics are set and the forces acting on the object are specified, the dynamics can be “turned on”. The user can observe the effects of the forces and make necessary adjustments. Once the user is satisfied, the object can be saved to an NPSOFF file with all defining tokens automatically inserted. The Mover tool provides a simple, interactive environment to view and adjust an object’s basic dynamic behavior.

Future Directions

Current and future projects at NPS will seek to extend and improve NPSOFF. In particular, we seek to add non-graphical characteristics to object definitions. Also, work is being done to improve the overall design of the NPSOFF system.

The virtual world applications developed in the Graphics and Video Laboratory require objects to have non-graphical characteristics. These characteristics include physical properties (e.g. mass and center of gravity), sound (effects for movement and weapons fire) and animation.

Future work on the PBM system will add support for defining inter-object relationships and constraints. This would allow the composite object structure to be extended to where each subobject has physical properties and affects the behavior of the whole object. Also, the notion of linked objects needs to be explored in the context of NPSOFF. This will allow the realistic modeling of such things as vehicle controls (e.g. aircraft stick movement changes control surface which changes forces on whole aircraft).

We are currently adding sound support to our virtual world applications. Many sounds are object based. Support for sound tokens within NPSOFF would allow flexible use of sound effects in our applications. Putting the sound control within NPSOFF would provide a standard use of sounds and facilitate the collection of sound definitions just as we collect materials and textures.

Another area that future research will address is animation support within NPSOFF. Support for continuously animated portions of an object (vehicle antennae) or constraint management of sub-objects (doors, arms, etc.) would be very useful to our researchers. Such a system would benefit from the standardization that NPSOFF provides and offer many more capabilities to developers.

The NPSOFF system is object-based in its design and use but is implemented in a non-object oriented language. Modifying or extending the current system is time consuming and error prone. Adding a new token to the language requires the modification of at least five source files. We are currently redesigning NPSOFF to be truly object-oriented and implementing it in C++. The main benefits of moving to an object-oriented implementation will be increased extensibility through inheritance and polymorphism and better maintainability.

Conclusion

In this paper we have described an object description language that is being used in virtual world construction for the NPSNET project [1]. NPSOFF provides the developer a standard, application independent, system for modeling graphical objects in an object-based manner. Current versions of the system support most graphical properties for objects. Future extensions will add physical properties, sound and animation support to the language system.

Acknowledgments

We wish to acknowledge the sponsors of our efforts, in particular Major David Neyland, USAF, of ARPA/ASTO, John Maynard and Duane Gomez of the Naval Ocean Systems Center, San Diego, Stan Goodman of USA STRICOM and Col. Dolahite, USA of the Headquarters Department of Army AI Center, Washington, D.C.

References

1. Zyda, Michael J. and Pratt, David R. “NPSNET: A 3D Visual Simulator for Virtual World Exploration”, SID Digest, May 1991, pp 361-363

2. Zyda, Michael J., Pratt, David R., Monahan, James G. and Wilson, Kalin P., “NPSNET: Constructing a 3D Virtual World”, *in Computer Graphics, Special Issue on the 1992 Symposium on 3D Interactive Graphics*, 29 March - 1 April 1992, pp. 147-156.

3. Silicon Graphics Computer Systems Inc., Graphics Library Reference Manual, C edition, IRIS-4D series, 1990.

4. PHIGS+ Committee, Andries van Dam, chair, “PHIGS+ Functional Description, Revision 3.0,” *Computer Graphics*, 22(3), pp. 125-218, July 1988.

5. International Standards Organization, *International Standard Information Processing Systems - Computer Graphics - Graphical Kernel System for Three Dimensions (GKS-3D) Functional Description*, ISO Document Number 8805:1988(E), American National Standards Institute, New York, 1988.

6. *Starbase Graphics Techniques and Display List Programmer’s Guide*, Hewlett-Packard Company, Fort Collins, Colorado, 1991.

7. Steve Upstill, *The RenderMan Companion*, Addison-Wesley, Reading, Mass., 1990.

8. Zyda, Michael J., Monahan, James G. and Pratt, David R. “NPSNET: Physically-Based Modeling Enhancements to an Object File Format,” chapter in *Creating and Animating the Virtual World*, Ed: Nadia Magnenat-Thalmann, Daniel Thalmann, Pub: Springer-Verlag Tokyo, 1992, pp. 35-52

```
/* define a light called whitelight */
deflight whitelight
ambient 0.0 0.0 0.0
lcolor 1.0 1.0 1.0
position 0.0 0.707106 0.707106 0.0
defend
/* last value in position line indicates if */
/* light is local or at infinity. 0.0 = infinity */
/* position is vector if light is not local */

deflmodel mylightmodel
ambient 0.0 0.0 0.0
localviewer no
attenuation 1.0 0.0 0.0 /* k0 k1 k2 */
twoside no
defend

defmaterial shinygold
emission 0.0 0.0 0.0
ambient 0.4 0.2 0.0
diffuse 0.9 0.5 0.0
specular 0.9 0.9 0.0
shininess 20.0
alpha 1.0
defend

deftexture wood
imagefile wood.rgb
minfilter mipmap_bilinear
magfilter bilinear
wrap repeat
endtexture
```

Figure 1.0 - Example definition tokens

```
/* define a line */
defline
ncoords /* number of coordinates */
x0 y0 z0
x1 y1 z1
...

/* define a polygon (with texture coords in []) */
defpoly[t]
nx ny nz /* normal vector */
ncoords
x0 y0 z0 [s0 t0]
x1 y1 z1 [s0 t0]
...

/* draw a surface (optional texture coords) */
defsurface[t]
ncoords
x0 y0 z0 nx0 ny0 nz0 [s0 t0]
x1 y1 z1 nx1 ny1 nz1 [s1 t1]
...

/* draw a triangular mesh */
deftmesh
ncoords
x0 y0 z0 nx0 ny0 nz0
x1 y1 z1 nx1 ny1 nz1
...
```

Figure 2.0 - Example primitives tokens

```
/* define a composite object made of cubes */
defobject barbell
/* draw copy of a unit cube translated left */
pushmatrix
translate -5.0 0.0 0.0
callobject cube
popmatrix

/* draw a copy of the cube stretched */
pushmatrix
scale 10.0 1.0 1.0
callobject cube
popmatrix

/* draw a copy of the cube translated right */
pushmatrix
translate 5.0 0.0 0.0
callobject cube
popmatrix
endobject
```

Figure 3.0 - Example composite object

```
/* program cube_ex.c */
long cubewin; /* window id */
OBJECT *cube; /* pointer to NPSOFF object */

void init()
{
/* open a window and read the object */
cubewin = winopen("cube Display");
RGBmode();
doublebuffer();
gconfig();

/*read an object file and assign a pointer */
cube = read_object("cube.off");

/* scan the object for definition tokens */
ready_object_for_display(cube);
}

main()
{
float black[4] = {0.0, 0.0, 0.0, 0.0};

init();

while(1) /* let window manager kill us */
{
 winset(cubewin);
 clear(black);
 display_object(cube);
 swapbuffers();
}
}
```

Figure 5.0 - Example object use code

```
/* define the light for the object */
deflight whitelight
 ambient 0.0 0.0 0.0
 lcolor 1.0 1.0 1.0
 position 0.0 0.707106 0.707106 0.0
defend

/* define the lighting model */
deflmodel mylightmodel_normal
 ambient 0.0 0.0 0.0
 localviewer no
 attenuation 1.0 0.0 0.0
 twoside no
defend



/* turn on the lights */
setlight whitelight 0 on


/* turn on the lighting model */
setlmodel mylightmodel_normal on

/* define the whitecube material */
defmaterial whitecube
 emission 0.0 0.0 0.0
 ambient 0.2 0.2 0.2
 diffuse 1.0 1.0 1.0
 specular 1.0 1.0 1.0
 shininess 10.0
 alpha 1.0
defend

/* define the blue cube material */
defmaterial bluecube
 emission 0.0 0.0 0.0
 ambient 0.0 0.1 0.1
 diffuse 0.0 0.5 0.75
 specular 0.0 0.5 0.75
 shininess 10.0
 alpha 1.0
defend

defmaterial greencube
 emission 0.0 0.0 0.0
 ambient 0.0 0.2 0.0
 diffuse 0.0 1.0 0.0
 specular 0.0 1.0 0.0
 shininess 10.0
 alpha 1.0
defend

 /* Draw a cube with each face a different color */
 /* Begin Box's Definition */
 origin 0.0 0.0 0.0
```

Figure 4.0 - Example Object File

```
/* Back Face */
setmaterial whitecube
defpolyt
 0.0 0.0 -1.0
 4
 -5.0 5.0 -5.0 0.5 0.5
 5.0 5.0 -5.0 0.0 0.5
 5.0 -5.0 -5.0 0.0 0.0
 -5.0 -5.0 -5.0 0.5 0.0

/* Front Face */
setmaterial bluecube
defpolyt
 0.0 0.0 1.0
 4
 5.0 5.0 5.0 1.0 1.0
 -5.0 5.0 5.0 0.5 1.0
 -5.0 -5.0 5.0 0.5 0.0
 5.0 -5.0 5.0 1.0 0.0

/* Top Face */
setmaterial greencube
defpolyt
 0.0 1.0 0.0
 4
 -5.0 5.0 -5.0 0.25 0.25
 -5.0 5.0 5.0 0.75 0.25
 5.0 5.0 5.0 0.75 0.75
 5.0 5.0 -5.0 0.25 0.75

/* Bottom Face */
setmaterial redcube
defpolyt
 0.0 -1.0 0.0
 4
 -5.0 -5.0 -5.0 0.0 0.0
 5.0 -5.0 -5.0 2.0 0.0
 5.0 -5.0 5.0 2.0 2.0
 -5.0 -5.0 5.0 0.0 2.0

/* Left Face */
setmaterial yellowcube
defpolyt
 -1.0 0.0 0.0
 4
 -5.0 -5.0 -5.0 0.0 0.0
 -5.0 -5.0 5.0 0.5 0.0
 -5.0 5.0 5.0 0.5 1.0
 -5.0 5.0 -5.0 0.0 1.0

/* Right Face */
setmaterial magentacube
defpolyt
 1.0 0.0 0.0
 4
 5.0 -5.0 -5.0 1.0 0.0
 5.0 5.0 -5.0 1.0 1.0
 5.0 5.0 5.0 0.0 1.0
 5.0 -5.0 5.0 0.0 0.0
/* End Box's Definition */
```

Figure 4.0 - Example Object File (cont.)

```
/* These are ALL of the required units of measure. */

/* Note: each subtoken is separate and none are required (defaults used). */

defunits
    /* All lengths are in meters. Other length choices are available. */
 dimension meters
   /*All force magnitudes are in newtons. Other force choices are available*/
 force newtons
   /* All mass amounts are in kilograms. Other mass choices are available. */
 mass kilos
defend

/* These are ALL of the required object characteristics. */
/* Note: each subtoken is separate and none are required (defaults used). */

defphysics

 /* This object's initial position is (X,Y,Z) in meters relative from the
 environments's center. Unless otherwise specified, all triples are X,Y,Z
 respective. */

 location 0.00 0.00 0.00
 /* The object's position is constrained to a one meter level square, relative from
 the object's initial position. */

 location_lower -1.00 -0.00 -1.00
 location_upper 1.00 0.00 1.00
 /* This object's initial orientation (Roll, Yaw, Pitch) in degrees. */

 orientation 0.00 0.00 0.00
 /* The object's orientation is unconstrained. */

 orientation_lower 0.00 0.00 0.00
 orientation_upper 360.00 360.00 360.00
 /* The object's initial linear velocity in meters/second. */

 linear 0.00 0.00 0.00
 /* The object's linear velocity is constrained to: 0.00 to 1000.0 longitudinal,
 +/- 1000.0 vertical and +/- 500.0 latitudinal. */

 linear_lower 0.00 -1000.00 -500.00
 linear_upper 1000.00 1000.00 500.00
 /* The object's initial angular velocity in degrees/second. */

 angular 0.00 0.00 0.00
 /* The object's angular velocity is constrained to: +/- 10.00 longitudinal, vertical
 and latitudinal. */

 angular_lower -10.00 -10.00 -10.00
 angular_upper 10.00 10.00 10.00
 /* The object's center of mass and amount in kilos. */

 mass_amount 25000.00
 mass_center 0.00 0.00 0.00
 /* The object's ability to absorb local forces. (0.0 is perfectly inelastic) */

 elasticity 0.80
```

Figure 6.0 - Example NPSOFF file with physical tokens

```
/* The dimensions of the object's bounding volume (e.g. for collision detection).
 The volume dimensions are calculated if this data is omitted. */

 bv_radius 30.00
 bv_latitude 15.00
 bv_longitude 20.00
 bv_vertical 8.0
 /* The location of the object's local viewpoint.

 setviewpoint 0.00 38.241650 0.00
defend

/* These are ALL of the required force characteristics for this force.

Note: each subtoken is separate and none are required (defaults used). */

defforce left_jet_engine

 force_type non-deforming
 force_origin -4.0 0.5 -0.8
 force_origin_low 0.0 0.0 0.0
 force_origin_high -4.5 0.5 -0.8
 force_direction -1.0 0.0 0.0
 force_magnitude 8000.0
 force_magnitude_constraints 0.0 10000
 asleep no
defend

/* Additional forces (right_engine, left_aileron,...) would follow here. */

/* The next two definitions specify a polygon lighting/shading characteristic.

 Note: each subtoken is separate and none are required (defaults used). */

defmaterial su25mat0
 emission 0.00 0.00 0.00
 ambient 0.047059 0.086275 0.047059
 diffuse 0.235294 0.431373 0.235294
 specular 0.00 0.00 0.00
 shininess 0.00
 alpha 1.00
defend

defmaterial su25mat1
 emission 0.00 0.00 0.00
 ambient 0.047059 0.094118 0.047059
 diffuse 0.235294 0.470588 0.235294
 specular 0.00 0.00 0.00
 shininess 0.00
 alpha 1.00
defend

/* The remaining defmaterials go here. */

/* A particular lighting/shading characteristic is activated. */
setmaterial su25mat0

/* Primitive drawing tokens would follow */
```

Figure 6.0 - Example NPSOFF file with physical tokens (cont)

```
/* INTEGRATION SAMPLE

The following code fragments demonstrate the various phases of OFF file integration with the
force/object functions.
*/

/* Initializing The Environment */

initialize_environment();

/* Let’s add gravity. */

 add_global_force(); /* allocates force and positions current forceptr */
 strcpy(current_global_forceptr->name,”gravity”);
 modify_force_origin(current_global_forceptr,0.0,1.0,0.0);
 modify_force_direction(current_global_forceptr,0.0,-1.0,0.0);



/* Adding And Modifying An Object */

objectptr = read_object(“sample_filename”);
ready_object_for_display(objectptr);
add_object_to_environment(objectptr);

/* Any characteristics (specified or not in the OFF file) can be modified.
We can check if a particular object characteristic has changed (e.g. by polling
an input device), and adjust it prior to calculating the objects’ motion. */

modify_object_position(objectptr,px,py,pz);
modify_object_position_lower(objectptr,lx,ly,lz);
modify_object_position_upper(objectptr,ux,uy,uz);
modify_object_rotation(objectptr,rx.ry,rz);
modify_object_rotation_lower(objectptr,lx,ly,lz);
modify_object_rotation_upper(objectptr,ux,uy,uz);
modify_object_linear_velocity(objectptr,vx,vy,vz);
modify_object_linear_velocity_lower(objectptr,lx,ly,lz);
modify_object_linear_velocity_upper(objectptr,ux,uy,uz);
modify_object_angular_velocity(objectptr,vx,vy,vz);
modify_object_angular_velocity_lower(objectptr,lx,ly,lz);
modify_object_angular_velocity_upper(objectptr,ux,uy,uz);
modify_object_mass(objectptr,mass,mx,my,mz);
modify_object_bounds(objectptr,radius,latitude,longitude,vertical);

/* If the object needs to be removed, we delete it. */

delete_object_from_environment(objectptr);
delete_object(objectptr);

/* If we want to suspend all forces on a particular object, */

suspend_object(objectptr);

/* or to re-allow all active forces to influence the object. */

wakeup_object(objectptr);
```

Figure 7.0 - PBM system run-time use

```
/* Adding And Modifying A Force */

add_local_force(objectptr);

/* or */

add_global_force();


/* Any characteristics of a force (specified or not in the OFF file) can be modified. */
/* We can check if a particular force characteristic has changed (e.g. by polling
 an input device), add adjust it prior to calculating the objects' motion.
 A pointer to a force can be retrieved from an object's force list or the environment*/

modify_force_origin(forceptr,ox,oy,oz);
modify_force_origin_lower(forceptr,lx,ly,lz);
modify_force_origin_upper(forceptr,ux,uy,uz);
modify_force_direction(forceptr,ox,oy,oz);
modify_force_magnitude(forceptr,magnitude);
modify_force_magnitude_constraints(forceptr,lower,upper);
modify_force_type(forceptr,type);

/* If the force needs to be removed, */

delete_local_force(objectptr,forceptr);

/* or */

delete_global_force(forceptr);

/* If we want to suspend a force, */

suspend_force(forceptr);

/* or to re-allow this force to influence the object. */

wakeup_force(forceptr);

/* Updating The Object's Physics
 Functions are provided to update all objects in the environment or
 individual objects. The update process applies applicable forces,
 modifies velocities and updates location and orientation information.
*/
update_environment();
update_object(objectptr);

/* An object can be displayed by the environment or individually by the user.
 Also a transformation matrix for the object can be requested.
*/
display_environment()
display_object(objectptr);
objmatrixptr = object_matrix(objectptr);
```

Figure 7.0 - PBM system run-time use (cont)