Chapter in *Creating and Animating the Virtual World*, Editors: Nadia Magnenat Thalmann and Daniel Thalmann, Publisher: Springer-Verlag Tokyo, 1992, pp. 35-52.

NPSNET: PHYSICALLY-BASED MODELING ENHANCEMENTS TO AN OBJECT FILE FORMAT

Michael J. Zyda*, James G. Monahan, David R. Pratt
Naval Postgraduate School
Department of Computer Science
Monterey, California 93943-5100 USA
Email: zyda@trouble.cs.nps.navy.mil
*contact author

Abstract

The Naval Postgraduate School (NPS) has actively explored the design and implementation of real-time three-dimensional simulators on low-cost, readily accessible graphics workstations. Many of the simulator platforms have had tremendous success due to the fact that a common object format was used. Prototyping time is dramatically reduced when the tedious and often repetitious task of object design is replaced with the simpler task of modifying an existing object description file. The current level of support that the NPS Object File Format (NPSOFF) provides is descriptions for lights, lighting, material characteristics, the expected graphics drawing primitives (lines, polygons, surfaces,...), and provisions for texturing and special lighting effects (spotlights, decaling,...). The objectives of this work are the enhancement of the basic NPSOFF structure with information necessary for accurate physically-based rendering in real-time; to construct a library of functions specifying an object's physical properties and the internal/external forces controlling the object and to develop a tool to rapidly design and test an object's dynamic characteristics.

The Basic Object File Format

In the past, the various areas of real-time, three-dimensional (3D) visual simulator research at NPS have had specific scope and purpose for a unique vehicle platform (Zyda 1991c). The visual simulators developed in the Graphics and Video Laboratory include the FOG-M missile simulator, the VEH vehicle simulator, the Airborne Remotely Operated Device (AROD), the Moving Platform Simulator series (MPS-1, MPS-2 and MPS-3), and the High Resolution Digital Terrain Model (HRDTM) system. Simulation design and implementation techniques optimized the technology of workstations in use. Advances in workstation hardware and software have always lead to more accurate simulations with each successive generation. An area of concern was to prevent an iconoclastic attitude between existing simulation projects and to facilitate the rapid prototyping of new simulator platforms. It soon became evident that future simulator development would demand a more unified protocol for object/scene description, rendering, and manipulation.

Advances in hardware capabilities such as lighting and texturing were painfully absent from these early simulations. There was no ability to quickly modify and port various objects between platforms; object renderings and control modifications were tedious for the platform's author let alone a follow-on design team. The NPSOFF initial research was designed to solve these problems by introducing an editable ASCII file with the information necessary to render an object along with various support routines to show, manipulate and save NPSOFF objects (Zyda 1991a, Zyda 1991b).

The version 1.0 NPSOFF consisted of lights, light model, material (color) and drawing subprimitive (lines, polygons, surfaces) definition tokens along with some administrative tokens for file maintenance and readability. The rendering of an object was accomplished in 3 steps: 1) pre-render parsing of the ASCII file into a dynamically allocated structure of object definition opcodes, 2) pre-render definition of lights and light models, 3) traversing the opcode list, drawing only the graphic primitives and selecting the "currently active" light, light model or material definition. Step 3 is the only one required each time through the display loop.

Additions To The Basic Object File Format

Further enhancements to NPSOFF included tokens to select textures, decaling, 2-sided lighting, spotlights and other rendering attributes. While current NPSOFF objects *looked* just like the real world objects that they were simulating, unfortunately, many of the NPSOFF object simulations did not *behave* realistically. As each NPSOFF object was nothing more than a description of its "skin", it was usually animated by implicitly specifying changes in linear position/velocity and orientation. NPSOFF objects could quite literally become "...faster than a speeding bullet, more powerful than a locomotive..." and defy many more laws of physics that we implicitly, if not explicitly, understand.

Incorporating Physical Realism

More recent research at NPS, specifically the Autonomous Underwater Vehicle (AUV), has taken a current NPSOFF submarine object and animated it under the constraints of accurate hydrodynamic laws of motion (Jurewicz 1989). The result is an amazingly realistic, both visually and physically, simulation of one specific NPSOFF object. A small drawback of the AUV simulation is that the physically-based modeling (PBM) representation of the dynamics is hardcoded. Adding/adjusting the AUV's dynamics is not a simple task, and the integration of a physically different submarine model would require software maintenance by a knowledgeable AUV programmer.

Related Work

bolio

bolio is an integrated graphical simulation platform developed by David Zeltzer et al at MIT's Media Lab (Brett 1987, Sturman 1989, Zeltzer 1989). The project's goal has been to provide an environment that animates objects governed by a network of constraints (dynamic and kinematic). The bolio file format is similar in nature to NPSOFF in that the top level file contains ASCII keyword/ value pairs specifying object characteristics. While bolio identifies additional binary data structure files, NPSOFF remains completely ASCII. The product development at NPS is almost exclusively experimental research and it was felt that a 100% human readable file format was needed during platform prototyping. When the final project design has been accepted, each NPSOFF file can be converted into binary to reduce I/O.

While bolio has demonstrated exceptional realism with the constraint-based movement of a *few articulated bodies*, the NPSOFF and the NPS simulation network (NPSNET) programs have been more concerned with the real-time animation of a *legion of 3D icons* (Zyda 1991c). Only with recent advances in workstation hardware has there been a capability to render a multitude of minimally articulated vehicles, in real-time. The vehicular nature of most NPSNET objects has lead toward a more interactive form of object-control, rather than bolio's use of kinematically specified task-level manipulations. Also, the constraints in NPSOFF are used more as a specification of an object's *physical capabilities*, rather than a notation for an object's *desired behavior.*

Virya

Virya is a graphical editor for specifying an articulated object's motion-control characteristics, designed by Jane Wilhelm's group at UCSC (Wilhelms 1986, Wilhelms 1987). A user can assign to each body's degree of freedom (DOF), one or more controlling functions (forces or torques vs. time or positions vs. time). These functions can exist in one of many control states such as position or dynamics control, frozen or relaxed. The control functions are cubic spline curves delineated by control points maintained in an ASCII file format.

Notion

Additional motion control work by Jane Wilhelm's group describes a technique that allows a user to depict an object's behavior based on internal sensors (provocation detectors), effectors (propulsion mechanisms) and mappings (connections and nodes) between them (Wilhelms 1990). Connections provide data transfer/modification from sensors to effectors, while nodes permit multiple connections from many sources of input/output. This technique has been demonstrated with an interactive, workstation-based system called Notion which allows a user to specify and view an object's behavior-derived motion.

Dynamic Constraints

Barzel and Barr present an approach to controlling rigid bodies with dynamic constraints (Barr 1987, Barzel 1988a, Barzel 1988b). These constraints are instanced and then sustained throughout the animation using inverse dynamics. The resultant "constraint" forces determine the object's motion. Rather than construct "constraining" forces, we are more interested in specifying "controlling" forces, similar to Barzel/Barr's use of external forces to guide objects prior to constraint initiation.

This paper describes an approach for enhancements to NPSOFF which bestows an object with physical characteristics and provides mechanisms to govern the object's motion given a list of known internal and external forces acting on the object. We have developed a rudimentary algorithm for the automatic maintenance of multiple objects' current placement and orientation in real time. Using a tool developed at NPS called the NPSOFF Mover Tool, a designer can view NPSOFF objects from all perspectives, including those from an object's point of view. After a set of forces is added and adjusted in location/affect, the designer is then able to "test-drive" an object to verify its force characteristics. Constraints on the force actuators and object movement are easily added or changed. The modified NPSOFF object is saved back to a file and is ready for integration into any simulation utilizing the NPSOFF library of object and force functions.

The following sections describe: basic dynamics theory for object animation, the use of a layered approach to the creation and application of force definitions, force control and action control in NPSOFF, the capabilities and performance of the NPSOFF Mover Tool development and testing simulator. The final section concludes with a description of future work to increase the accuracy and realism of the physically-based modeling while lowering the final complexity of user-specified object movement.

The Dynamics Of Object Animation

The use of dynamics in rigid-body simulations requires a delicate understanding and balancing of geometric and algorithmic complexities. If we are interested in modeling the precise physical interactions of simple objects, we can afford the computational expense of dynamics simulation. Increasing an object's structural complexity and having it interact with a greater number of peer objects, strains many dynamics algorithms to the point where they are unusable for real-time simulation. It is clear that the use of dynamics to simulate Newtonian mechanics is essential for most forms of motion (ballistic, robotic, ambulatory and piloted). The following sections attempt to provide broad insight into simplifying the task of dynamics integration. Additional amplification is available in two exceptional references, Jane Wilhelm's dynamics tutorial (Wilhelms 1988) and Goldstein's mechanics theory text (Goldstein 1980).

Non-deforming Forces

Initial Conditions

The object's current position and orientation are calculated based on a set of current initial conditions:

$$\left[p_{x_0}, p_{y_0}, p_{z_0}\right] \quad \text{(position)}$$

$$\left[\theta_{x_0}, \theta_{y_0}, \theta_{z_0}\right] \quad \text{(orientation)}$$

$$\left[\frac{\delta}{\delta t}(p_{x_0}), \frac{\delta}{\delta t}(p_{y_0}), \frac{\delta}{\delta t}(p_{z_0})\right] \quad \text{(linear velocity)}$$

$$\left[\frac{\delta}{\delta t}(\theta_{x_0}), \frac{\delta}{\delta t}(\theta_{y_0}), \frac{\delta}{\delta t}(\theta_{z_0})\right] \quad \text{(angular velocity)}$$

and initial time t_0.

Often, the initial velocity values are not needed as most objects begin life motionless. Nevertheless, the ability to create an object, such as an airplane, in all phases of its movement description requires a provision for non-zero initial velocities.

Newton's Laws

$$F_{xyz} = m \times a_{xyz}$$

or simply "A given force acting on a given mass will accelerate it."

More specifically,

$$F_{xyz} = m \times \frac{\delta^2}{\delta t^2}(p_{xyz})$$

$$T_{xyz} = m \times \frac{\delta^2}{\delta t^2}(\theta_{xyz})$$

where

F_{xyz}	= the net force directed at the object's center of mass
T_{xyz}	= the net torque directed at the object's center of mass
m	= object mass
$\delta^2/\delta t^2(p_{xyz})$	= linear acceleration component
$\delta^2/\delta t^2(\theta_{xyz})$	= angular acceleration component

Local Frame of Reference

Each non-deforming force vector is converted into two collision coordinate system vectors; one that affects a torque (tangential) and one that affects a translation (radial) in each of the XY, YZ and XZ planes respectively (Figure 1). Since each component of the movement force 6-vector (three tangential forces and three radial forces) is mutually exclusive, they are summed to generate a *cumulative* object frame movement force 6-vector.

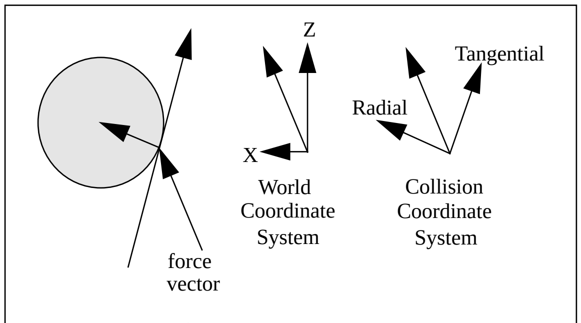

Figure 1 Force Components

The movement force 6-vector specifies an object-frame acceleration 6-vector (the three tangential force components create three rotation acceleration components as the remaining three radial force components create three translational accelerations). For example, in the XZ plane, only the X and Z components create linear motion and torque about the Y axis. Each force adds its effects to the object's six object frame of reference accelerations along and around each of the three object's axes. An object-frame velocity 6-vector is calculated using constant acceleration over the integration time interval δt. Both 6-vectors are then mapped into their world frame counterpart 6-vectors. These world frame accelerations and velocity 6-vectors are then used in a modified Euler integration (Spiegal 1988).

$$\frac{\delta}{\delta t}(Fp_{xyz}) = \frac{\delta}{\delta t}(Ip_{xyz}) + \left(\frac{\delta^2}{\delta t^2}(Ip_{xyz}) \times \delta t\right)$$

$$\frac{\delta}{\delta t}(F\theta_{xyz}) = \frac{\delta}{\delta t}(I\theta_{xyz}) + \left(\frac{\delta^2}{\delta t^2}(I\theta_{xyz}) \times \delta t\right)$$

These two equations calculate final linear/angular velocities, given current velocities and accelerations over a time interval δt.

$$Fp_{xyz} = Ip_{xyz} + \left(\frac{\delta}{\delta t}(Fp_{xyz}) \times \delta t\right) + \left(0.5 \times \frac{\delta^2}{\delta t^2}(p_{xyz}) \times (\delta t)^2\right)$$

$$F\theta_{xyz} = Ip_{xyz} + \left(\frac{\delta}{\delta t}(F\theta_{xyz}) \times \delta t\right) + \left(0.5 \times \frac{\delta^2}{\delta t^2}(\theta_{xyz}) \times (\delta t)^2\right)$$

These two equations calculate final linear/angular positions, given current positions/velocities and predicted velocity averages at sample time.

where

$\delta/\delta t(Fp_{xyz})$	= final linear velocity component
$\delta/\delta t(F\theta_{xyz})$	= final angular velocity component
Fp_{xyz}	= final position component
Ip_{xyz}	= initial position component
$F\theta_{xyz}$	= final orientation component
$I\theta_{xyz}$	= initial orientation component
δt	= time interval since last integration

The modified Euler method was selected for its simplicity and iterative speed. Each object's force list is updated once per rendering loop, therefore nullifying the additional precision provided by second order and higher methods of integration. Obviously, Euler's method will lose accuracy as each object is subjected to rapidly changing forces. A future implementation will sample the force updates in parallel, track the relative changes in linear/angular accelerations and switch to a higher order integration method, such as Runge-Kutta, under a rapidly moving scenario.

Each global force (such as gravity) affects the object at its center of mass causing only linear acceleration. Since the movement does not involve rotations, it can be added after the net effect of all local forces is determined.

Deforming Forces

A deforming force affects the object in one of three ways. Each polygon in the object has an associated *break* and *bend threshold* token specified in newtons/meter^2. Using the relationship that a force dissipates its kinetic energy inversely over the square of the distance from the force origin to the polygon, a dissipated force per unit polygon surface area value is calculated. If the force is strong enough to break the polygon, the original polygon token is removed from the object token list and replaced with a list of smaller triangular polygonal shard tokens (Figure 2). Triangles are used to guarantee planar polygons.

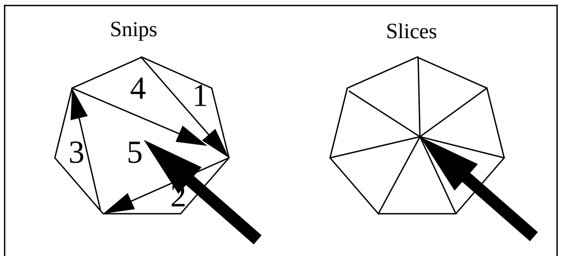

Figure 2 Breaking Force

The shards are initially determined by "snipping" off the corners of a multi-sided convex polygon, thus spiraling inward until the remaining quadrangle is divided in two. The rationale is to 1) prevent identical "pizza slice" shards as explosions are rarely symmetrical and 2) generate (*n-2*) versus *n* fragments from an *n*-sided polygon. Any remaining shards are broken along their hypotenuse, as needed.

If the force is only strong enough to bend the polygon, the polygon token is removed from the object token list and replaced with a new bendable polygon that tracks a moving point of bending force impact (Figure 3). The bending force is modeled using Hooke's Law and a spherical spring that seeks to return the moving vertex back to the polygon's actual point of impact.

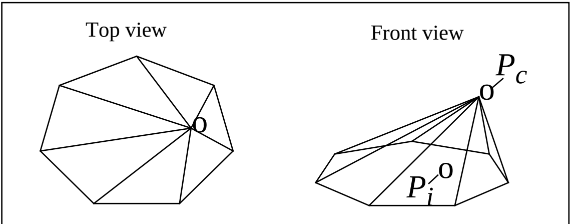

Figure 3 Bending Force

$$F_{xyz} = -(k_s \times \Delta p_{xyz} + k_d \times \Delta v_{xyz})$$

where

Pi	= initial point of bending force impact
Pc	= current point of bending force impact
F_{xyz}	= linear bending component
k_s	= spring constant
k_d	= damping constant
Δp_{xyz}	= difference between the position components of the initial and current points of impact
Δv_{xyz}	= difference between the velocity components of the initial and current points of impact

If the force is neither strong enough to break or bend a polygon, then it may only push a polygonal shard.

A Layered Approach

The initial objective of our work was to provide a structured mechanism for object behavior control that would allow varying degrees of *user* and *designer* involvement with the simulation. The end user is concerned with realism: visual accuracy and similarity of interface (i.e *look* and *feel*). The simulated object's appearance and movement must closely resemble its real world counterpart. Objects that instantly accelerate to highly unnatural velocities provide a temporary sensation of giddiness, bordering upon the comical. A lack of expected visual clues from the objects' Newtonian interactions, such as the lack of gravity effects, reduces the allure of the virtual reality. A similar degradation in simulator immersion is also evident when the user is expected to identify and interface with an inordinate number of forces, which are manipulating several objects.

Method

The desired implementation would allow the art design team to create a specific object hierarchy with realistic shape, colorings and positioning data. The engineering section would then add the subobject physical attributes (mass, center of mass and object elasticity) and affecting force descriptions (force position, or point of affect in the object's frame of reference, along with the force direction unit vector, magnitude and type of force). Reasonable defaults for omitted physical attributes are assumed and/or calculated from other specifications. The analysis team would then specify mappings between subobject movement and the forces affected by such movement. The end user is then able to control a given object in a realistic manner with realistic results by manipulating a set of *control subobjects* linked to local forces. The result is an adjustable "focus" in specifying high-level object motion in a range of control modes: directly, indirectly through local *force* control, and even more indirectly with *subobject* control.

The final objective was to design a suite of tools, so that a single user, with even a limited background in Newtonian mechanics, could rapidly design and test an object's physical characteristics.

Descriptions Of Each Layer

Similar to Barr's view of Teleological Modeling (Barr 1988), the approach is for each layer to provide a control description to the layer below. The lowest layer consists of rendering descriptions (drawing primitives, materials and lighting controls to color their skins). The second layer consists of the object's physical characteristics and a set of force descriptions (a list of forces and their influence upon specific objects). The third layer consists of action descriptions (a mapping of an object's movements to changes in a set of force descriptions) (Figure 4).

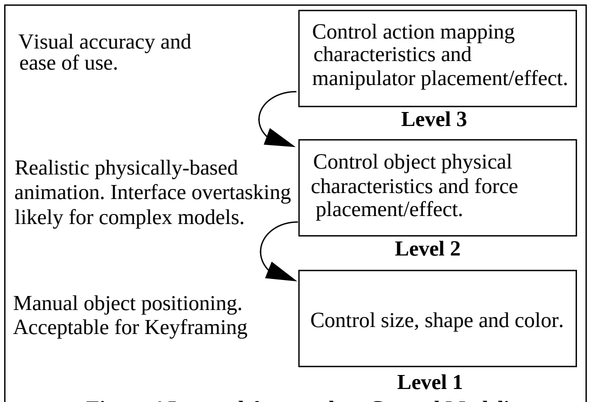
Visual accuracy and ease of use.
Control action mapping characteristics and manipulator placement/effect.
Level 3
Realistic physically-based animation. Interface overtasking likely for complex models.
Control object physical characteristics and force placement/effect.
Level 2
Manual object positioning. Acceptable for Keyframing
Control size, shape and color.
Level 1

Figure 4 Layered Approach to Control Modeling

Primitive Additions

As described, the version 1.0 NPSOFF file supported only drawing subprimitives such as lines, polygons, and meshed surfaces. Other common generalized surface primitives (cones, cylinders, spheres and parallelepipeds) were usually calculated off-line, with their polygonal data stored into an NPSOFF file. The major disadvantage to this approach was that these NPSOFF files were extremely large and difficult to edit. As scenarios requiring different colors on a primitive were rare at best, the inclusion of a set of parametrized primitive descriptions was required. For example, a cylinder token is specified by a height, radius, and quality factor that indicates the maximum number of polygons or mesh points to use in the rendering. The advantages were automatic minimal polygonal computation based on ranging data from a specified viewpoint, a simple mechanism for multiple object resolution creation and known mass/center of mass values, to name a few.

Objects And Force Control

The models in the various NPS simulators have quite an eclectic background. Some came from non-organic sites such as NASA and MIT, requiring conversion from other file formats. Many others were designed in-house and more often than not, the various models were rarely scale compatible. The first set of extensions to the NPSOFF language, (Table 1), included tokens to specify and convert between the various units of measure (Level 1).

TABLE 1: UNITS OF MEASURE

Token Function	Argument Type(s)
units of dimension	char
units of force	char
units of mass	char

Object Modeling Requirements

The next set of extensions, (Table 2), included tokens to specify the capabilities and constraints of a force acting upon an object (Level 2).

TABLE 2: FORCE CHARACTERISTICS

Token Function	Argument Type(s)
name	char
type	char
origin	float, float, float
origin constraints (low)	float, float, float
origin constraints (high)	float, float, float
direction	float, float, float
magnitude	float
magnitude constraints	float, float
asleep	yes or no

The next set of extensions, (Table 3), included tokens to specify the physique, initial conditions, and motion boundary conditions of an object (Level 2). A default bounding volume is calculated as the object is read into memory. Provisions for specifying a smaller (or larger) bounding description (spherical, rectangular or ellipsoid) were added to facilitate parallel research efforts in collision detection. As much of the research at NPS involves the simulation of piloted vehicles, the inclusion of a vehicle viewpoint was a requirement.)

TABLE 3: OBJECT CHARACTERISTICS

Token Function	Argument Type(s)
initial position	float, float, float
position constraints (low)	float, float, float
position constraints (high)	float, float, float
initial rotation	float, float, float
rotation constraints (low)	float, float, float
rotation constraints (high)	float, float, float
initial linear velocity	float, float, float
linear velocity constraints (low)	float, float, float
linear velocity constraints (high)	float, float, float
initial rotation velocity	float, float, float
angular velocity constraints (low)	float, float, float
angular velocity constraints (high)	float, float, float
mass	float
center of mass	float, float, float
elasticity	float
bounding volume radius	float
bounding volume length	float
bounding volume width	float
bounding volume height	float
viewpoint from object	float, float, float

The concept of a “sleeping” or suspended force is to leave the force attached to the object but remove its effect. The rationale for this was for simplification during the force definition and analysis phase. A given force can be isolated by leaving it the sole “awake” or active force. The unacceptable alternative is to nullify the other forces by restricting their magnitudes and/or directions.

Objects And Action Control

What we have now is a “marionette” object that is manipulated by pulling and pushing “force” strings and rods. For objects with a simple description of force effectors, this is quite acceptable as each force can be visualized as a controlling “object” rather than a force. Realistically, most objects’ movements are *described* by a complex network of forces, yet *controlled* from a small number of input sources. Level 3’s objective will be to identify a small set of *control* objects and provide a mapping from their movement (translations/rotations) changes to a set of force description (origin/direction/magnitude) changes.

We will want to specify a set of *dynamic* controlling forces and then provide an abstraction for altering their effect based on user input changes. As early key-frame animation control systems identified key object positions and then interpolated the in-between positions as a function of time, so should we specify key controlling force effects and then interpolate the in-between effect components as functions of a user input position/orientation. This approach will provide for a natural migration from *simulated* input sources to *actual hardware/sensor* input sources.

For example, a jet object has three forces that describe the effects of two ailerons and one stabilizer. The action induced by their movement is controlled by one input source, the pilot's stick. The user will describe simple mappings for the stick's lateral rotation (affecting the aileron forces) and longitudinal rotation (affecting the stabilizer force). Adding additional mappings for throttle/rudder pedal positions and we will have an airplane that is fully controllable with changes in the *input device's position and rotation* (Figure 5).

As the construction of Layer 3 is part of continuing research, the following is a *specification* for a possible NPSOFF file mapping from subobject movement (translations/rotations) changes to a set of force description (origin/direction/magnitude) changes:

```
defmapping sample_map_name
 sample_object              name
 sample_force               name
 rot_to_force_origin        matrix
 rot_to_force_vector        matrix
 trans_to_force_origin      matrix
 trans_to_force_vector      matrix
defend
```

Each object and force has an initial (neutral) state specified in their respective descriptions. Each respective 3x3 matrix would transform a 3-vector (controlling object rotation and translation changes) into another 3-vector (force origin and vector incremental updates). Additional mappings would require velocity information as well.

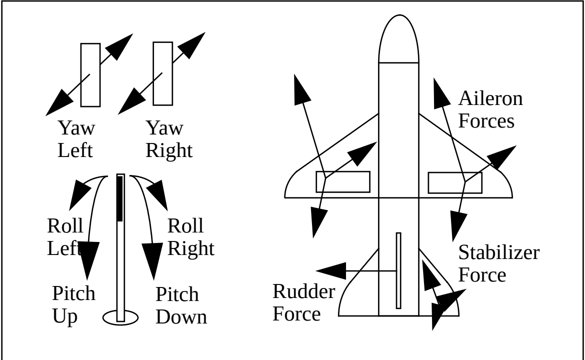

Figure 5 Linking Subobject Movement to Forces

The NPSOFF Mover Tool

The objective of the NPSOFF Mover Tool is to provide an environment to design and test the dynamics of NPSOFF objects. An NPSOFF object without physical characteristics is read into memory from disk and the object is measured for future calculations. A default mass, mass center, elasticity and object viewpoint are calculated. The user is then able to "fine tune" any of these approximations based on known data. The user then specifies the initial values for object position, orientation and velocity. At the lowest layer of the tool's control, the user is able to continually update the object's movement by indicating the linear/angular direction and speed. This would be acceptable for specifying instances for a key-framing sequencer, but we are more interested in providing mechanisms to accelerate the object just like its real world counterpart. The mechanism of choice is a force description.

Application

The user controls a set of forces that are in turn, controlling the object's movement. A force is positioned around an object and its range of effect is specified. For example, our jet fighter object is re-read into the Mover Tool and the engine forces are added via a force interface (Figure 6). A separate force vector is positioned in the center of each exhaust nozzle, initially directed forward (direction of the *reactive* force) and parallel to the turbine housing, with a zero newton magnitude. The force's magnitude is constrained by a non-negative range of thrust values. The point of effect is also given a small range of values along the axis parallel to the direction of thrust to account for the change in thrust position when the engine is operated in afterburner and additional fuel is combusted behind the main combustion chamber. Similar forces are added to the wings for lift and drag forces created by the various control surfaces (flaps, ailerons, spoilers,...).

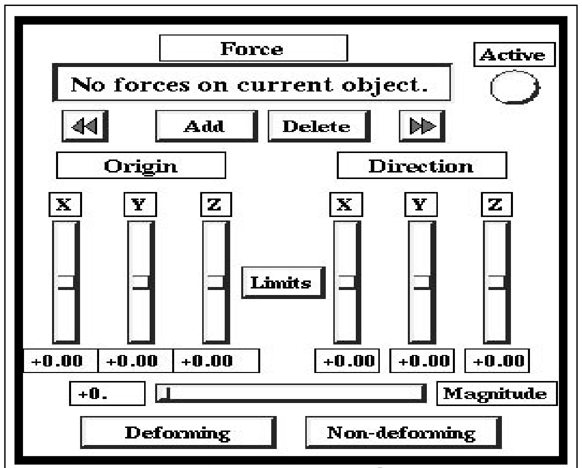

Figure 6 Force Panel

Additional pseudo forces, such as parasitic drag can be added to tune the realism. The result is a vehicle that will maneuver with amazing realism. The object can be “test flown” in isolation or with other testbed objects to verify the object’s force parameters. The object is saved to a file when the desired physical model is accepted. The NPSOFF file force descriptions are always editable with any ASCII text editor.

The following is a sample NPSOFF file force description for the plane’s left engine:

```
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
```

Postdesign

The same functions that are used to animate objects in the NPSOFF Mover Tool are embedded in the NPSOFF function library. In addition to the functions that read in an object file, ready it for display, and display it each time through the display loop, are a host of new functions that: add/delete objects and forces from the animation environment, navigate the object/force lists, alter the object/force parameters, and start/stop the animation process.

Off File Sample

The following is a fragment of an NPSOFF file description of an SU-25 Frogfoot Soviet ground attack aircraft.

```
/* These are ALL of the required units of measure. */
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 choic-
  es are available. */
mass kilos
defend

/* These are ALL of the required object characteristics.*/
defphysics
/* This object's initial position is (X,Y,Z) in meters rel-
  ative 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 lev-
  el 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 per
  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

/* 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.
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 light-
   ing/shading characteristic.
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 acti-
   vated. */
setmaterial su25mat0

/* The next definition specifies a triangular polygon.
defpoly
/* Normal, number of vertices and vertex coordi-
   nates. */
0.875439 -0.483329 0.00
3
40.142231 6.476233 1.029732
39.087486 4.565808 -1.076130
40.142231 6.476233 -1.029732

/* The remaining primitive definitions go here. */
```

Off File Integration Sample

The following C-code fragments demonstrate the various phases of NPSOFF file integration with the force/object functions.

```
/* Initializing the environment. */
initialize_environment();

/* Let's add gravity. */
add_global_force();
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 NPSOFF file) can be modified. We can check if a particular object characteristic has changed (e.g. by polling an input device), add 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);

/* If we want to suspend all forces on an object, */
suspend_object(objectptr);

/* or to allow all active forces to influence an object. */
wakeup_object(objectptr);

/* Adding and modifying a force. */
add_local_force(objectptr);
/* or */
add_global_force();

/* Any characteristics (specified or not in the NPSOFF 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. */
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);

/* Functions are provided to update all or individual objects' physics in the environment. The update process applys applicable forces, modifies velocities and updates location and orientation information.*/
update_environment();
update_only_this_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_only_this_object(objectptr);
objmatrixptr = object_matrix(objectptr);
```

Performance

The following tables are used to illustrate the cost associated with this physically-based modeling technique. All tests were performed on a Silicon Graphics 4D/240VGX workstation (single-thread). Test case group A involves a small, average and large polygon count object with small, average and large non-deforming force lists (Table 4). Test case group B involves small, average and large sets of an average polygon count object with small, average and large non-deforming force lists (Table 5). Test case group C involves a small, average and large polygon count object with a small deforming force list, during the explosion phase (Table 6). All numbers are in frames per second.

TABLE 4: OBJECT SIZE VERSUS NUMBER OF FORCES (Frames/second)

	Small **Polygon Count** (6) Object	*Average* **Polygon** Count (165) Object	*Large* **Polygon** Count (960) Object
Single **Non-deforming** Force	17.117	16.547	11.174
Small **Set of** Non-deforming Forces (5)	20.083	18.166	11.864
Medium **Set of** Non-deforming Forces (10)	19.923	18.063	11.519
Large **Set of** Non-deforming Forces (20)	19.525	17.876	10.765

TABLE 5: NUMBER OF OBJECTS VERSUS NUMBER OF FORCES (Frames/second)

	Average **Number** of Objects (5)	*Large* **Number** of Objects (10)
Small **Set of** Non-deforming Forces (5)	11.861	8.441
Average **Set of** Non-deforming Forces (10)	11.525	8.179
Large **Set of** Non-deforming Forces (2**0)**	10.935	8.025

TABLE 6: OBJECT SIZE VERSUS A DEFORMING FORCE (Frames/second)

	Small **Polygon** Count (6) Object	*Average* **Polygon** Count (165) Object	*Large* **Polygon** Count (960) Object
Small **Set of Deforming** Forces	16.535	15.493	10.454

A note of interest in case A - the frame rate actually increases from one force to five forces and then decreases from then on. We are reclaiming idle CPU time and improving graphics-CPU overlap.

In the case of non-deforming forces, the frame rate decreases linearly with the *total number of non-deforming forces* attached to all objects, (Figures 7 and 8). In the case of deforming forces, the frame rate decreases linearly with the *number of initial polygons* in the pre-destroyed object (Figure 9).

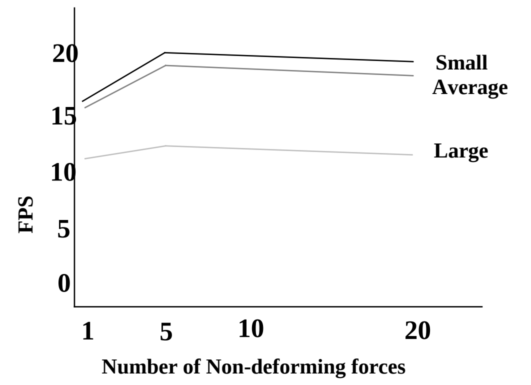

Figure 7 Object Size and Non-deforming Forces

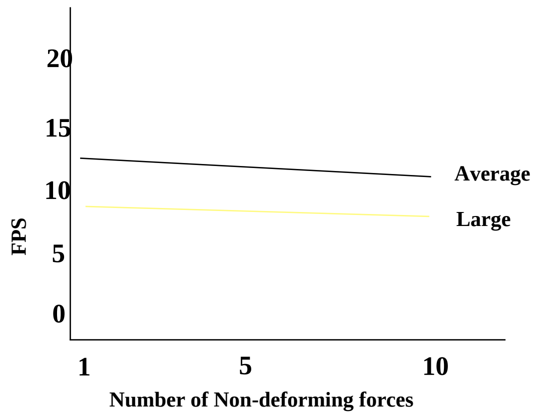

Figure 8 Number of Objects and Non-deforming Forces

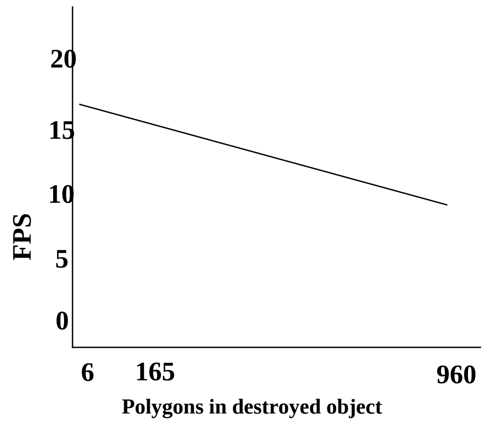

Figure 9 Object Size and Deforming Forces

Conclusions And Future Work

The use of physically-based modeling is still in its infancy at NPS. Previous simulations were able to "fake" or "downplay" the expected visual clues from an object's physical interactions. As hardware and software technology afford us with greater *capability* in animation realism, we are obligated to strive for more accurate physical modeling, but not at the expense of increased *user workload* to specify and control the animation process. The extensions to NPSOFF present a simplified mechanism for building models with physical characteristics and adding controlling functions that are as complex as necessary given the current hardware support.

Future work includes the implementation of the Action Control layer using the Layer 3 specification. Generation of the mapping function matrices could be achieved quite easily by selecting the object/force pair and then taking "snapshots" of a series of object motion/force description couplings. Each coupling would then be displayed in a 2D graph (object component vs. force component) for any desired function smoothing/modification. Addition, deletion, and modification mechanisms would function similarly to identical object functions in key-framing systems.

Further refinements to the integration process would include parallelization of the force sampling process and the addition of an adaptive algorithm for more accurate positioning of objects with rapidly fluctuating forces.

References

(Barr 1987)
A. H. Barr, “Dynamic Constraints”, A*CM SIGGRAPH ‘87 Tutorial Notes: Topics in Physically-Based Modeling*, 1987

(Barr 1988)
A. H. Barr, “Teleological Modeling”, *ACM SIGGRAPH ‘88 Course Notes #27: Developments in Physically-Based Modeling, Section E*, August, 1988

(Barzel 1988a)
R. Barzel and A. H. Barr, “A Modeling System Based on Dynamic Constraints,” *ACM SIGGRAPH ‘88 Conference Proceedings*, Vol 22 No 4, August 1988, pp. 179-188

(Barzel 1988b)
R. Barzel and A. H. Barr, “Controlling Rigid Bodies with Dynamic Constraints,” *ACM SIGGRAPH ‘88 Course Notes #27: Developments in Physically-Based Modeling, Section E*, August, 1988

(Brett 1987)
C. Brett, S. Pieper, D. Zeltzer, “Putting it all Together: An Integrated Package for Viewing and Editing 3D MicroWorlds,” *Proc. 4th Usenix Computer Graphics Workshop*, October 1987

(Goldstein 1980)
H. Goldstein, *Classical Mechanics*, Second Edition, Addison-Wesley, Reading, MA, 1980

(Jurewicz 1989)
T. Jurewicz, “A Real Time Autonomous Underwater Vehicle Dynamic Simulator,” M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1989

(Spiegel 1988)
M. Spiegel, “*Applied Differential Equations, Third Edition*,” Prentice Hall, Inc., Englewood Cliffs, N.J. 1988, pp. 1-26

(Sturman 1989)
D. Sturman, D. Zeltzer, and S. Pieper, “The Use of Constraints in the bolio System,” *ACM SIGGRAPH ‘89 Course Notes: Implementing and Interacting with Realtime Microworlds*, Boston, MA, July 31, 1989

(Wilhelms 1986)
J. Wilhelms, “Virya - A motion Control Editor for Kinematic and Dynamic Animation,” *Proceedings of Graphics Interface 86*, May, 1986, pp. 141-146

(Wilhelms 1987)
J. Wilhelms, “Using Dynamic Analysis for Realistic Animation of Articulated Bodies,” *IEEE Computer Graphics and Applications*, Vol 7 No 6, June 1987, pp. 12-27

(Wilhelms 1988)
J. Wilhelms, “Dynamics for Computer Graphics: A Tutorial,” *Computing Systems*, USENIX Association, Winter, 1988, pp. 63-93. also *UCSC Computer and Information Science Technical Report* UCSC-CRL-87-5

(Wilhelms 1990)
J. Wilhelms and R. Skinner, “An Interactive Approach to Behavior Control,” *ACM SIGGRAPH ‘90 Course Notes: Developments in Physically-Based Modeling*, July, 1990

(Zeltzer 1989)
D. Zeltzer, S. Pieper, and D. Sturman, “An Integrated Graphical Simulation Platform,” *Proceedings of Graphics Interface 89*, June 19-23, 1989, London, Ontario, pp. 266-274

(Zyda 1991a)
M. Zyda, “Book 7, Computer Graphics”, *Naval Postgraduate School Course Notes CS 4470: Computer Graphics*, 2 April 1991

(Zyda 1991b)
M. Zyda, “Book 9, Computer Graphics”, *Naval Postgraduate School Course Notes CS 4470: Computer Graphics*, 31 May 1991

(Zyda 1991c)
M. Zyda and D. Pratt, “NPSNET: A 3D Simulator for Virtual World Exploration and Experimentation”, *Society for Information Display 1991 International Symposium Digest of Technical Papers*, 31 May 1991, pp. 361-364