

Open Source Multibody Aeroelastic Modeling, Simulation, and Video Rendering

G. Douglas Baldwin*
Baldwin Technology Company, LLC
Port Washington
New York 11050

Abstract

Multibody simulation and video animation are both powerful tools for analyzing, communicating, and promoting advanced vertical flight concepts. By combining these two activities, the rendered videos have the credibility of being physics based, and the multibody simulation results can be presented in a real world setting. This paper reports on the integration of two complimentary open source tools to create a general purpose multibody modeling, simulation, and video rendering environment that can be used for real-time pilot-in-the-loop or batch mode simulation and analysis. The two free open source tools that were integrated are MBDyn and Blender.

Introduction

A technically powerful and potentially cost effective method for exploring innovative rotorcraft concepts is to build a series of virtual multibody dynamic models of increasing complexity, and then simulate and analyze the models' behavior in a variety of operating environments. With multibody techniques, the hard work of faithfully representing elemental physics is accomplished by subject matter experts and delivered to engineers as a useful analytical tool. The potential cost effectiveness of this technique can be overwhelmed, however, by the need to add new elemental features to the simulation environment after the virtual models are built. When dealing with innovative concepts, one does not always know a priori the full set of elemental features needed, and in fact this knowledge can be a product of the engineering process and develop over time. The choice of a multibody analytical tool is a critical decision, particularly if the tool has economic or legal encumbrances that drive the cost of feature enhancement.

One innovative rotorcraft design that can benefit from multibody dynamic simulation and analysis is the 9400 pound gross weight Mono Tiltrotor – Scaled Demonstrator (MTR-SD) illustrated in Figure 1. The configuration has several design features that would benefit from a multibody treatment, including: the tilting coaxial proprotor; the pitch axis suspended cargo pod; and the optionally hinged wing panels which can fold to eliminate download in hover and aerodynamically deploy into an efficient fixed wing cruise configuration. This aircraft design is the subject of published Government research reports [1, 2], contractor reports [3, 4, 5, 6, 7] and conference papers [8, 9, 10]. Further Government funded research and development is anticipated.

In preparation for future contracted work, Baldwin Technology Company, LLC (BTC) developed a general purpose multibody dynamic graphical design and simulation environment that combines the MBDyn text based multibody dynamic software with the Blender artistic graphical design and rendering environment. Both of these design tools and the integration software that was written by BTC are open source and freely available.

Subject

Quoting from its webpage, “MBDyn is the first and possibly the only free, general-purpose MultiBody Dynamics analysis software... MBDyn features the integrated multidisciplinary analysis of multibody, multiphysics systems, including nonlinear mechanics of rigid and flexible constrained bodies, smart materials, electric networks, active control, hydraulic networks, essential fixed-wing and rotorcraft aerodynamics.” [11] A recent validation of this tool against proprietary tools and wind tunnel data was presented at the AHS Forum 2008, entitled “Modeling a Stiff-Inplane Tiltrotor Using Two Multibody Analyses: a Validation Study” [12]. Furthermore, MBDyn can operate in a parallel and real-time computational environment. Thus, MBDyn incorporates the features appropriate for rotorcraft simulation, and has been validated for this purpose.

The Blender website is even more bold in its assertion of leadership. Its website includes the statement, “Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License” [13]. Indeed, high definition animated movies with computer generated actors have been created using Blender.

* Managing Director. doug@baldwintechology.com



Figure 1: Mono Tiltrotor Scaled Demonstrator (MTR-SD) Rendering.

While it is true that the Blender software tool was developed by and for the use of the digital arts community, it incorporates an underlying mathematics model that fits perfectly with MBDyn's data representation. Each 3D Blender Object has a 3D location vector and a 3D rotation matrix that can be used as proxies for MBDyn nodes, and Blender can animate these matrices over a time domain simulation. Furthermore, Blender has an open and documented Application Programming Interface (API) for writing Python scripts to interface with this data model, and for interacting with the model in realtime pilot-in-the loop simulation [14].

One of the first individuals to recognize the compatibility between MBDyn and Blender was John Kollman of OSEngineer.com. He developed a set of tools he calls Blended MBDyn for implementing a limited subset of MBDyn functionality inside of Blender [15]. This effort showed the possibility of having a point-and-click environment for specifying the MBDyn model, and then automatically importing the MBDyn results to visually represent the 12-degree of freedom behavior of all the multibody nodes. After importing the model's dynamic behavior, the native power of Blender's materials properties, textures, lighting, and shading can be used to produce realistic video renderings.

Purpose

The specific motivation behind integrating Blender with MBDyn comes from the desire to deliver a physics based motion to the rendered videos of the MTR-SD design. While it has heretofore been possible to produce realistic animations based on a prescribed motion that represents observed and predicted behavior, this integration of Blender and MBDyn takes the next step of directly coupling the rendering to a physics based analysis. Furthermore, all of the analytical products, charts, and graphs that are traditionally produced by multibody analysis can be derived from these free open source code

simulations, and any analytical results can be made available without restrictions for independent verification and validation of both the model and the underlying analytical code.

Scope

All the above is simply prologue to the work performed and reported in this paper. A full integration of MBDyn and Blender has been accomplished using the object oriented Python scripting language [16, 17]. The graphical user interface (GUI) provides context sensitive, hierarchical menus. Aeroelastic, system level models can be rapidly developed, analyzed, and rendered by clicking on the Blender Objects, selecting from a menu the MBDyn attributes to be applied to each Object, and then tailoring those attributes. The software architecture approach taken was to enhance native Blender Objects by assigning them MBDyn attributes, and then use these attributes to generate the text based input file required by MBDyn. An expert MBDyn user may then elect to edit this text based input file if they like before initiating the analysis. The analysis can be performed in batch or pilot-in-the-loop interactive mode, with the pilot inputs saved for future batch execution. The temporal 6-degree of freedom results are automatically imported into Blender to animate the Objects in real time or as a batch file, and the animation can be rendered in high quality at a user specified number of frames per second providing a 12-degree of freedom video representation of the model.

Furthermore, this Python script has a future proof, open, object oriented architecture that easily facilitates the inclusion of new MBDyn attributes as they are added to the MBDyn code.

Plan

The fundamental goal was to have a graphical user interface (GUI) that would be familiar to current Blender users, and use this GUI to produce MBDyn input and output files that would be familiar to MBDyn users. In this regard, the goal of implementing full production capable functionality with unaltered MBDyn or Blender software code was achieved.

The software development plan that was executed included the following steps: 1) Write a prototype software procedure in the Python programming language that enables Blender to graphically represent a fully specified MBDyn model, generate an MBDyn input file for batch execution, and then import the MBDyn batch results into Blender for graphical display. 2) Architect and write an object oriented Python script with class specifications for all MBDyn model attributes, and test this script against the MBDyn supplied example models. 3) Publish the script under the free open source software license [18] and solicit feedback from current MBDyn

users. User feedback was then incorporated into a subsequent code release to make the code production ready.

A Government research contract award is anticipated in which this code will be used to advance the understanding of the MTR-SD in performance of mission task elements specified in the "Aeronautical Design Standard: Performance Specification: Handling Qualities Specification for Military Rotorcraft" (ADS-33E-PRF).

Research Methodology

Background material regarding both Blender and MBDyn is useful in understanding the significance of this work, and how best an engineer can benefit from these tools. Blender has a very large and active user base that freely generates user tutorials and provides helpful guidance. While some of the user documentation and tutorials are in written form, the most useful and more current support is provided in narrated video tutorials posted to the Internet. The Blender software lends itself to video instruction, as does the integration of Blender with MBDyn.

MBDyn, on the other hand, is completely text based. The user must have some capacity for spatial visualization to envision in "the mind's eye" the multibody model, and then write a specification for this model using keywords and numerical values, all properly delimited with commas and semicolons. After batch execution of the model, MBDyn delivers its results in tabulated numerical format. These tabular results must be plotted or otherwise represented to communicate meaning. MBDyn lends itself to formal classroom instruction with assigned problem sets.

The combination of MBDyn and Blender provides the opportunity for video instruction while communicating multibody dynamics engineering concepts. In fact, experienced Blender users with no knowledge of MBDyn have successfully used video instruction to build and run example MBDyn models. Any attempt to provide written instruction in this paper on the use of MBDyn and Blender would be inferior to and less interesting than video instruction. For that reason, video tutorials have been produced and more will be produced and posted to <http://www.baldwintechology.com/blender.html>.

A key point is that the Blender script does allow a user to create a multibody model that has no physical analogue and would fail to execute in MBDyn. In this regard, the graphical interface is no different than the text interface to MBDyn. A key difference, however, is that the model produced in Blender will always be geometrically accurate and geometrically consistent. The user no longer needs to envision the geometric orientation of each element, and faithfully transcribe this orientation into mathematical form. Furthermore, the user in the

graphical Blender environment can rapidly interact with and test a variety of changes to the model.

Excerpts from video tutorials will be presented at the technical session. For the benefit of software developers and users who would like to customize these scripts, the remainder of this methodology section of the paper will discuss the software design.

Blender's Python API

Blender provides a rich Python API [14] for directly interacting with all aspects of its 3-D environment. The MBDyn script primarily uses only three classes of Blender API methods. The first class is the Object API which interfaces with the position and orientation data associated with each Blender Object. Object data is used to populate the MBDyn input file; the MBDyn output file is used to animate Objects for visual display of simulation results. The second class is the Mesh API, with is used to alter the visual representation of each mesh Object according to its MBDyn attributes. The third class is the Draw API which is used for drawing GUI menus.

Blender scripts are constrained to using the Draw API for GUI functionality for the following reason. The Python scripting language is compiled into Blender and is available as a single threaded process. One benefit of constraining Python to a single process thread is that it eliminates the possibility of having two parallel processes both altering a Blender model simultaneously. A byproduct of this single threaded implementation is that external GUI programs cannot directly communicate with a Blender Python process. The class Draw API has been found to be completely satisfactory for implementing MBDyn GUI functionality, with the benefit of loosely enforcing conformity with Blender's standard user interface.

Software Architecture

MBDyn functionality in Blender is delivered in the form of two Python scripts [18]. The front-end script, named `mbdyn_gui.py`, provides the primary interface to the user and includes the routines for performing an automatic full database backup triggered by each change to the database, and for performing automatic restoration of the database when the Blender file is reopened and the script is restarted. The back-end script, named `mbdyn.py`, contains all class definitions and class methods. These classes define the database architecture, and also define the methods for generating context sensitive user menus, the methods for populating the database, and the methods for writing the MBDyn input file.

Classes are defined for each of the following MBDyn entities: Element, Constitutive, Shape, Driver, Matrix, Friction, Function, and Drive. A description of the

unique meaning and purpose of each entity is as described in the MBDyn documentation [19]. However, all of these class definitions have the following features in common within the script.

Each class definition is organized into three methods. The first method (`__init__`) generates an instance of an MBDyn entity and initializes its attributes. The common attributes for all entities are as follows: `*.type` of entity; `*.name` of this particular instance of an entity; `*.users` to enumerate the number of other entities in the MBDyn database that are linked to this entity; `*._links` to list the pointers of all other entities that this entity links to; and `*._args` which provides a place to hold scalar, string, or Boolean data values associated with this entity. Also, the MBDyn database must be passed from method to method which is the purpose served by the `*.database` attribute.

The second method (`modify`) generates the GUI data entry blocks for the user to enter scalar, string, or Boolean data, and to select links to other entities. This method is performed immediately following entity initialization, and this newly initialized entity is stored in the persistent database only upon successful completion of this second method.

The third method (`write`) generates the text for the MBDyn input file. Each MBDyn entity has a text input format that is specified by the MBDyn documentation, and this specification is complied with by the 'write' method. Some 'write' methods generate the actual text lines which are append to the Mbdyn input file, while other 'write' methods simply generate text strings that are returned to their calling method for eventual insertion into the MBDyn input file.

The Element class has a fourth method (`draw`) which modifies the appearance of the Blender object to provide a visual representation of the entity. For example in the case of an aerodynamic body, this visual representation is parameter driven to provide feedback to the user of the wing's chord, span, taper, sweep, and twist. The video tutorials provide clear examples of how this 'draw' method is applied to the various MBDyn elements.

The other classes of objects that are defined in the `mbdyn.py` file are named Common, Menu, Frame, and MBDyn. The Common class is simply a collection of methods that can be applied to the MBDyn entities. For instance, the `'name_check'` method is used to ensure that each entity has a unique name. As another example, the `'finalize'` method is performed at the conclusion of every `'modify'` method to update the MBDyn database.

The Menu class provides the methods for generating the context sensitive menus. This class is best understood by first learning the Blender Python API calls for generating a tree menu. A lot of programming gymnastics were employed to force the Blender Python API to behave appropriately.

The final two classes are Frame and MBDyn. The Frame class provides the ability to create fixed or moving 3-D reference frames, and to assign MBDyn nodes to those reference frames. Reference frames seem to be unnecessary in the Blender environment for most MBDyn models, however this feature of the MBDyn text based input file is retained.

The class named MBDyn initializes the object database and holds lists of all the entities generated by all of the preceding classes. When the `mbdyn_gui.py` script starts, its first action is to call for the initialization of the MBDyn database, and then populate this database with any saved data. This class includes a 'write' method which is the primary method for generating the MBDyn input file, and it calls all of the other entity 'write' methods for their strings and lines of text.

An alert reader may have noticed that there is no class definition for Nodes. How can this be? A node is the fundamental entity for any multibody dynamic model. The answer is that the Blender Objects are used to represent nodes, and a database of Objects is maintained by the Blender program. In fact, these Objects are linked to the MBDyn entities and are what give these MBDyn entities spatial meaning. Depending on the linked relationships between the Blender Objects and the MBDyn entities, these Objects are used by the 'write' method to create nodes and displacements from nodes.

Results and Discussion

The primary result of this work is a free open source software, production capable, graphical environment for general purpose multibody dynamic simulation based on MBDyn and Blender. Most all of the MBDyn entities are enabled in this environment, including 2-node and 3-node aeroelastic beams, dynamic inflow and other rotor wake models, and a variety of joints including clamp and total joint, revolute and spherical hinge, and deformable hinge. The environment supports both batch and interactive execution of MBDyn models. For instance, batch mode can be used to incrementally and interactively build a model and examine its behavior as elements are added or changed. Once the model is complete, a pilot-in-the-loop mode may be employed to send streaming keyboard and joystick driver inputs during MBDyn execution, and display in the 3-D environment the dynamic behavior of the model driven by the streaming output of MBDyn's position and orientation data for each node. The streaming driver inputs are automatically captured in a log file and can be reused in subsequent batch runs to produce detailed simulation output data files for engineering presentation and analysis. The batch run can also be used to drive a high quality rendered animation of the model.

As an example of this technique, a video tutorial was produced to demonstrate building and flying a swashplate

driven, articulated blade, coaxial helicopter model [20]. The video tutorial shows many high productivity features such as using the 'duplicate' method to automatically duplicate a blade on the lower hub, and then 'duplicate' the entire lower rotor assembly to quickly and easily create the upper rotor assembly. The tutorial also demonstrates the iterative process of adding elements to the model and running short test simulations to check the behavior of the model. Once the model is complete, this video tutorial shows pilot-in-the-loop flight and a batch rendering of the animated behavior of the piloted aircraft.

As evidence of the value provided by this technique, two figures are provided below. Figure 2 shows an image from the video tutorial of the coaxial helicopter model. Any MBDyn element assigned to any Blender Object in this figure can be interrogated or changed with two clicks of the mouse. The first right click selects the Object to be interrogated, and the second right click brings up a menu of all MBDyn elements assigned to that Object and a list of MBDyn elements that could be added to the Object.

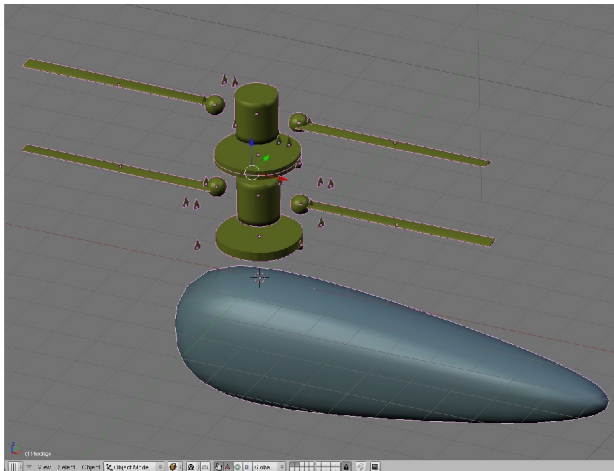


Figure 2: Blender 3-D representation of a coaxial helicopter with MBDyn attributes applied.

Figures 3 through 9 show the MBDyn input file that was automatically produced for the model shown in the prior figure. As can be seen, this input file is a very long and detailed specification of the model with lots of vectors, matrices, commas, and semicolons. A misplaced comma or an inverted matrix could cause MBDyn to have an execution error, or worse to execute and generate erroneous results. By using the Blender and MBDyn script, the coaxial helicopter model and input file was created in about an hour, and the whole design and flight simulation was done in a half day, all without a single typographical or mathematical error in the MBDyn input file. Note that this tutorial model was not intended to represent any specific design, and more design and engineering time would be needed to exactly place each Object and properly specify each parameter.

```
/* Label Indexes

Nodes:
0 - blade (Mesh, users=11)
1 - blade.001 (Mesh, users=11)
2 - blade.002 (Mesh, users=11)
3 - blade.003 (Mesh, users=11)
4 - fuselage (Mesh, users=10)
5 - hub (Mesh, users=14)
6 - hub.001 (Mesh, users=14)
7 - swash_lower (Mesh, users=5)
8 - swash_lower.001 (Mesh, users=5)
9 - swash_upper (Mesh, users=10)
10 - swash_upper.001 (Mesh, users=10)

Drives:
0 - Constant grav (Constant drive, users=1)
1 - Null drive (Null drive, users=10)
2 - Time < 3. (String drive, users=4)
3 - collective (File drive, users=2)
4 - cycle (Cosine drive)
5 - fore_aft (File drive, users=1)
6 - rampup_lower (Cosine drive, users=1)
7 - rampup_upper (Cosine drive, users=1)

Drivers:
0 - File (File, users=2)

Elements:
0 - Rotor (Rotor)
1 - Aerodynamic body (Aerodynamic body)
2 - Aerodynamic body.001 (Aerodynamic body)
3 - Aerodynamic body.002 (Aerodynamic body)
4 - Aerodynamic body.003 (Aerodynamic body)
5 - Air properties (Air properties)
6 - Axial rotation (Joint)
7 - Axial rotation.001 (Joint)
8 - Body (Body)
9 - Body.001 (Body)
10 - Body.002 (Body)
11 - Body.003 (Body)
12 - Deformable hinge (Joint)
13 - Deformable hinge.001 (Joint, users=1)
14 - Deformable hinge.002 (Joint)
15 - Deformable hinge.003 (Joint, users=1)
16 - Deformable hinge.004 (Joint)
17 - Deformable hinge.005 (Joint, users=1)
18 - Deformable hinge.006 (Joint)
19 - Deformable hinge.007 (Joint, users=1)
20 - Distance (Joint)
21 - Distance.001 (Joint)
22 - Distance.002 (Joint)
23 - Distance.003 (Joint)
24 - Driven (Driven)
25 - Driven.001 (Driven)
26 - Driven.002 (Driven)
27 - Driven.003 (Driven)
28 - Gravity (Gravity)
29 - In plane (Joint)
30 - In plane.001 (Joint)
31 - Revolute hinge (Joint)
32 - Revolute hinge.001 (Joint)
33 - Rigid (Rigid)
34 - Rigid.001 (Rigid)
35 - Rigid.002 (Rigid)
36 - Rigid.003 (Rigid)
37 - Rigid.004 (Rigid)
38 - Rigid.005 (Rigid)
39 - Rigid.006 (Rigid)
40 - Rigid.007 (Rigid)
41 - Rigid.008 (Rigid)
42 - Rigid.009 (Rigid)
43 - Rigid.010 (Rigid)
44 - Rigid.011 (Rigid)
45 - Rigid.012 (Rigid)
46 - Rigid.013 (Rigid)
47 - Rigid.014 (Rigid)
48 - Rigid.015 (Rigid)
49 - Rigid.016 (Rigid)
50 - Rigid.017 (Rigid)
51 - Rigid.018 (Rigid)
52 - Rigid.019 (Rigid)
```

Figure 3: MBDyn input file for the coaxial helicopter model.

```

53 - Rigid.020 (Rigid)
54 - Rigid.021 (Rigid)
55 - Rotor.001 (Rotor)
56 - Spherical hinge (Joint)
57 - Spherical hinge.001 (Joint)
58 - Spherical hinge.002 (Joint)
59 - Spherical hinge.003 (Joint)
60 - Total joint (Joint)
61 - Total joint.001 (Joint)
62 - fuselage (Body)

*/

begin: data;
  integrator: initial value;
end: data;

begin: initial value;
  initial time: 0.0;
  final time: 43.0;
  time step: 0.007;
  max iterations: 100;
  tolerance: 1e-06;
  derivatives tolerance: 2.0;
  derivatives coefficient: 0.001;
end: initial value;

begin: control data;
  default orientation: orientation matrix
  output meter: meter, 0., forever, steps, 5;
  structural nodes: 11;
  joints: 24;
  rigid bodies: 5;
  air properties;
  gravity;
  aerodynamic elements: 4;
  rotors: 2;
  file drivers: 1;
end: control data;

begin: nodes;
  structural: 0, dynamic,
    reference, global, 1.0, 0.0, 2.0,
    reference, global, matr,
      1.0, 0.0, 0.0,
      0.0, 1.0, 0.0,
      0.0, 0.0, 1.0,
    reference, global, null,
    reference, global, null;
  structural: 1, dynamic,
    reference, global, -1.0, 1.50995802528e-07, 2.0,
    reference, global, matr,
      -1.0, 8.74227765735e-08, 0.0,
      -8.74227765735e-08, -1.0, 0.0,
      0.0, 0.0, 1.0,
    reference, global, null,
    reference, global, null;
  structural: 2, dynamic,
    reference, global, -1.0, 1.50995802528e-07, 4.0,
    reference, global, matr,
      -1.0, 8.74227765735e-08, 0.0,
      -8.74227765735e-08, -1.0, 0.0,
      0.0, 0.0, 1.0,
    reference, global, null,
    reference, global, null;
  structural: 3, dynamic,
    reference, global, 1.0, 0.0, 4.0,
    reference, global, matr,
      1.0, 0.0, 0.0,
      0.0, 1.0, 0.0,
      0.0, 0.0, 1.0,
    reference, global, null,
    reference, global, null;
  structural: 4, dynamic,
    reference, global, 0.0, 0.0, 0.0,
    reference, global, matr,
      1.0, 0.0, 0.0,
      0.0, 1.0, 0.0,
      0.0, 0.0, 1.0,
    reference, global, null,
    reference, global, null;
  structural: 5, static,
    reference, global, 0.0, 0.0, 2.0,

```

Figure 4: MBDyn input file for the coaxial helicopter model (cont. from Figure 3).

```

reference, global, matr,
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0,
reference, global, null,
reference, global, null;
structural: 6, static,
reference, global, 0.0, 0.0, 4.0,
reference, global, matr,
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0,
reference, global, null,
reference, global, null;
structural: 7, static,
reference, global, 0.0, 0.0, 1.0,
reference, global, matr,
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0,
reference, global, null,
reference, global, null;
structural: 8, static,
reference, global, 0.0, 0.0, 3.0,
reference, global, matr,
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0,
reference, global, null,
reference, global, null;
structural: 9, static,
reference, global, 0.0, 0.0, 1.0,
reference, global, matr,
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0,
reference, global, null,
reference, global, null;
structural: 10, static,
reference, global, 0.0, 0.0, 3.0,
reference, global, matr,
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0,
reference, global, null,
reference, global, null;
end: nodes;

begin: drivers;
  file: 0, fixed step, 1964, 2, 0.0, 0.007, pad zeros, no,
  /home/doug/temp/rotor_File;
end: drivers;

begin: elements;
  rotor: 0, 4, 5, induced velocity, no;
  aerodynamic body: 1, 0,
    2.29999995232, 0.0, 0.0,
  matr,
    -1.78813962748e-07, -5.96046412227e-08, 1.00000011921,
    -1.00000011921, 1.19209282445e-07, -5.96046412227e-08,
    -2.38418579102e-07, -1.00000011921, -1.78813962748e-07,
  4.5,
  const, 0.333000004292,
  const, 0.0,
  const, 0.0,
  linear, -0.0349065555556, 0.104719666667,
  1;
  aerodynamic body: 2, 1,
    2.29999995232, -4.27566106964e-07, 0.0,
  matr,
    -2.38418564891e-07, -5.96046447754e-08, -1.0,
    -1.0, 9.13911861744e-08, 2.38418579102e-07,
    9.13911577527e-08, 1.0, -5.9604687408e-08,
  4.5,
  const, 0.333000004292,
  const, 0.0,
  const, 0.0,
  linear, -0.0349065555556, 0.104719666667,
  1;
  aerodynamic body: 3, 3,
    2.29999995232, 0.0, 0.0,
  matr,
    -5.96046518808e-08, 2.98023259404e-08, 1.0,

```

Figure 5: MBDyn input file for the coaxial helicopter model (cont. from Figure 4).

```

1.0, -2.38418607523e-07, 2.98023259404e-08,
2.08616256714e-07, 1.0, -5.96046518808e-08,
4.5,
const, 0.333000004292,
const, 0.0,
const, 0.0,
linear, 0.0349065555556, -0.104719666667,
1;
aerodynamic body: 4, 2,
2.29999995232, -4.27566106964e-07, 0.0,
matr,
2.38418607523e-07, 2.98023152823e-08, -0.999999940395,
0.999999940395, -1.50995788317e-07, 2.68220901489e-07,
-1.2119348014e-07, -1.0, -5.69544411633e-14,
4.5,
const, 0.333000004292,
const, 0.0,
const, 0.0,
linear, 0.0349065555556, -0.104719666667,
1;
air properties: std, SI,
temperature deviation, 0.0,
reference altitude, 0.0,
-1.0, 0.0, 0.0, null;
joint: 6, axial rotation,
5, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
4, 0.0, 0.0, 2.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
cosine, 0.0, 1.5, 20.0, half, 0.0;
joint: 7, axial rotation,
6, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
4, 0.0, 0.0, 4.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
cosine, 0.0, 1.5, -20.0, half, 0.0;
body: 8, 0, 30.0,
2.29999995232, 0.0, 0.0, diag, 60.0, 1.0, 60.0,
inertial, matr,
-1.78813962748e-07, -5.96046412227e-08, 1.00000011921,
-1.00000011921, 1.19209282445e-07, -5.96046412227e-08,
-2.38418579102e-07, -1.00000011921, -1.78813962748e-07;
body: 9, 1, 30.0,
2.29999995232, -4.27566106964e-07, 0.0, diag, 60.0, 1.0, 60.0,
inertial, matr,
-2.38418564891e-07, -5.96046447754e-08, -1.0,
-1.0, 9.13911861744e-08, 2.38418579102e-07,
9.13911577527e-08, 1.0, -5.9604687408e-08;
body: 10, 3, 30.0,
2.29999995232, 0.0, 0.0, diag, 60.0, 1.0, 60.0,
inertial, matr,
-5.96046518808e-08, 2.98023259404e-08, 1.0,
1.0, -2.38418607523e-07, 2.98023259404e-08,
2.08616256714e-07, 1.0, -5.96046518808e-08;
body: 11, 2, 30.0,
2.29999995232, -4.27566106964e-07, 0.0, diag, 60.0, 1.0, 60.0,
inertial, matr,
2.38418607523e-07, 2.98023152823e-08, -0.999999940395,
0.999999940395, -1.50995788317e-07, 2.68220901489e-07,
-1.2119348014e-07, -1.0, -5.69544411633e-14;
joint: 12, deformable hinge,
5,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,

```

Figure 6: MBDyn input file for the coaxial helicopter model (cont. from Figure 5).

```

Linear viscous, 10000.0;
joint: 13, deformable hinge,
0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
5,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
Linear viscous, 10000.0;
joint: 14, deformable hinge,
5,
hinge, matr,
-1.0, -8.74227765735e-08, 0.0,
8.74227765735e-08, -1.0, 0.0,
0.0, 0.0, 1.0,
1,
hinge, matr,
1.0, 1.74845553147e-07, 0.0,
-1.74845553147e-07, 1.0, 0.0,
0.0, 0.0, 1.0,
Linear viscous, 10000.0;
joint: 15, deformable hinge,
1,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
5,
hinge, matr,
-1.0, 8.74227765735e-08, 0.0,
-8.74227765735e-08, -1.0, 0.0,
0.0, 0.0, 1.0,
Linear viscous, 10000.0;
joint: 16, deformable hinge,
6,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
3,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
Linear viscous, 10000.0;
joint: 17, deformable hinge,
3,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
6,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
Linear viscous, 10000.0;
joint: 18, deformable hinge,
6,
hinge, matr,
-1.0, -8.74227765735e-08, 0.0,
8.74227765735e-08, -1.0, 0.0,
0.0, 0.0, 1.0,
2,
hinge, matr,
1.0, 1.74845553147e-07, 0.0,
-1.74845553147e-07, 1.0, 0.0,
0.0, 0.0, 1.0,
Linear viscous, 10000.0;
joint: 19, deformable hinge,
2,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
6,
hinge, matr,
-1.0, 8.74227765735e-08, 0.0,
-8.74227765735e-08, -1.0, 0.0,

```

Figure 7: MBDyn input file for the coaxial helicopter model (cont. from Figure 6).

```

0.0, 0.0, 1.0,
Linear viscous, 10000.0;
joint: 20, distance,
0, position, 0.0, 1.0, 0.0,
9, position, 1.0, 1.0, 0.0, from nodes;
joint: 21, distance,
1, position, 8.7422769468e-08, 0.999999940395, 0.0,
9, position, -1.0, -0.999999761581, 0.0, from nodes;
joint: 22, distance,
3, position, 0.0, -1.0, 0.0,
10, position, 1.0, -1.0, 0.0, from nodes;
joint: 23, distance,
2, position, -5.64259948987e-07, -1.00000011921, 0.0,
10, position, -1.0, 1.00000023842, 0.0, from nodes;
driven: 13, string, "Time < 3.",
existing: Joint, 13;
driven: 15, string, "Time < 3.",
existing: Joint, 15;
driven: 17, string, "Time < 3.",
existing: Joint, 17;
driven: 19, string, "Time < 3.",
existing: Joint, 19;
gravity: 0.0, 0.0, -1.0, 9.81000041962;
joint: 29, inplane,
9, 1.0, 0.0, 0.0,
0.0, -1.00000011921, -1.34358856485e-07,
5, offset, 1.0, 0.0, -1.0;
joint: 30, inplane,
10, 1.0, 0.0, 0.0,
0.0, -1.00000011921, -1.34358856485e-07,
6, offset, 1.0, 0.0, -1.0;
joint: 31, revolute hinge,
9, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
7, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0;
joint: 32, revolute hinge,
10, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
8, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0;
rotor: 55, 4, 6, induced velocity, dynamic inflow,
5.5, 40.0,
max iterations, 1,
tolerance, 1.0,
eta, 1.0,
correction, 1.0, 1.0;
joint: 56, spherical hinge,
0, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
5, 1.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0;
joint: 57, spherical hinge,
1, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
5, -1.0, 1.50995802528e-07, 0.0,
hinge, matr,
-1.0, 8.74227765735e-08, 0.0,
-8.74227765735e-08, -1.0, 0.0,
0.0, 0.0, 1.0;
joint: 58, spherical hinge,
3, 0.0, 0.0, 0.0,

```

Figure 8: MBDyn input file for the coaxial helicopter model (cont. from Figure 7).

```

hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
6, 1.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0;
joint: 59, spherical hinge,
2, 0.0, 0.0, 0.0,
hinge, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
6, -1.0, 1.50995802528e-07, 0.0,
hinge, matr,
-1.0, 8.74227765735e-08, 0.0,
-8.74227765735e-08, -1.0, 0.0,
0.0, 0.0, 1.0;
joint: 60, total joint,
4, position, 0.0, 0.0, 1.0,
position orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
rotation orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
7, position, 0.0, 0.0, 0.0,
position orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
rotation orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
position constraint, active, active, active, component,
null,
null,
file, 0, 1,
orientation constraint, active, active, active, component,
null,
file, 0, 2,
null;
joint: 61, total joint,
4, position, 0.0, 0.0, 3.0,
position orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
rotation orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
8, position, 0.0, 0.0, 0.0,
position orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
rotation orientation, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0,
position constraint, active, active, active, component,
null,
null,
file, 0, 1,
orientation constraint, active, active, active, component,
null,
null,
null;
body: 62, 4, 500.0,
0.0, 0.0, 0.0, diag, 1000.0, 1000.0, 1000.0,
inertial, matr,
1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0;
end: elements;

```

Figure 9: MBDyn input file for the coaxial helicopter model (cont. from Figure 8).

As was discussed in the introduction of this paper, the motivation for developing this general purpose design environment was to perform multibody dynamic analysis and design of the MTR-SD, and provide a physics based behavior for high quality animated renderings. A Government contract to perform this work is anticipated to start in Summer 2009.

Conclusions

A project to develop a free open source software based graphical environment for general purpose multibody dynamic modeling and simulation was undertaken. This project combined the MBDyn multibody dynamic simulation and analysis code with the Blender graphical design and rendering environment. The conclusions to be drawn from this work are as follows:

1. The project succeeded in maintaining a graphical user interface (GUI) that is familiar to current Blender users, and using this GUI to produce MBDyn input and output files that are familiar to MBDyn users.
2. Video tutorials teaching how to use this GUI environment to build and run multibody dynamic models have been produced and publicly released.
3. Aeroelastic models with 2-node and 3-node beams, and dynamic inflow and other rotor wake models, can be built in this GUI environment and operated with pilot-in-the-loop.
4. Batch execution using either a log file of pilot inputs or a prescribed control input can be run to produce detailed engineering data.
5. The motion output file of the MBDyn simulation can be used to produce high quality animated renderings of physics based behavior.
6. This work was motivated by the desire to create an extensible free open source software based engineering environment for modeling and simulating the multibody dynamic behavior of the Mono Tiltrotor Scaled Demonstrator (MTR-SD) aircraft design.

Acknowledgments

This work was performed under independent research and development by Baldwin Technology Company, LLC. The author gratefully acknowledges Dr. Pierangelo Masarati and Dr. Paolo Mantegazza of Politecnico de Milano for their guidance in using MBDyn, Dr. Jinwei Shen of the National Institute of Aerospace for testing the Blender and MBDyn scripts and recommending improvements that were incorporated into the code, and

to the Blender community for their assistance in learning the Blender API.

References

1. Baldwin, G. D., *Mono Tiltrotor Validation Activities*, U. S. Army Contract Number: W911W6-04-D-0004-0002, RDECOM TR 08-D-0069, July 2008.
2. Baldwin, G. D., *Mono Tiltrotor (MTR) Concept Evaluation*, U. S. Army Contract Number: W911W6-04-D-0004-0001, RDECOM TR 06-D-40, DTIC Accession Number ADB324612, 2006.
3. Assessment of the Mono Tiltrotor Scaled Demonstrator, Contract No: BTC001, Bell Helicopter Textron Incorporated, January 23, 2008.
4. Mavriplis, D. J., Computational Drag Study for the Mono Tiltrotor Scaled Demonstrator (MTR-SD), March 2008.
5. The Mono-Tiltrotor Final Project Report, U.S. Army Research Laboratory: Vehicle Technology Directorate, June 16, 2006.
6. MTR FY 2005 Development Final Comprehensive Report, Eagle Aviation Technologies Incorporated, May 18, 2006.
7. Leishman, J. G., Samscock, J., Mono Tiltrotor (MTR): Final Comprehensive Report of Research Performed During July 2005 – June 2006.
8. Baldwin, G. D., 'Mono Tiltrotor (MTR) Validation Activities', Proceedings of the 64th Annual National Forum of the American Helicopter Society, Montreal, Canada, April 29 - May 1, 2008.
9. Baldwin, G. D., 'Utility of a Mono Tiltrotor (MTR) Scaled Demonstrator', Proceedings of the 63rd Annual National Forum of the American Helicopter Society, Virginia Beach, VA, May 1-3, 2007.
10. Baldwin, G. D., "Preliminary Design Studies of a Mono Tiltrotor (MTR) with Demonstrations of Aerodynamic Wing Deployment", AHS International Specialists Meeting, Chandler, Arizona, January 23-25, 2007.
http://www.baldwintechology.com/MTR_AHS_Jan07.pdf
11. <http://www.aero.polimi.it/mbdyn/>, accessed 23 March 2009.
12. J. Shen, P. Masarati, D.J. Piatak, M.W. Nixon, J.D. Singleton, B. Roget, "Modeling a Stiff-Inplane Tiltrotor Using Two Multibody Analyses: a Validation Study", American Helicopter Society Annual Forum, April 29 - May 1, 2008.
13. <http://www.blender.org/>, accessed 23 March 2009.

14. <http://www.blender.org/documentation/248PythonDoc/index.html> , accessed 23 March 2009.

15. <http://www.osengineer.com/blendedmbdynintro.htm> , accessed 23 March 2009.

16. Lutz, M., *Learning Python, 3rd Edition*. O'Reilly, 2008.

17. Martelli, A., *Python in a Nutshell., 2nd Edition*. O'Reilly, 2006.

18. <http://sourceforge.net/projects/blenderandmbdyn> , accessed 23 March 2009.

19. Masarati, P., *MBDyn Input File Format*, Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano, 8 January 2009.
<http://www.aero.polimi.it/~masarati/MBDyn-input/mbdyn-input.pdf> accessed 23 March 2008.

20. Baldwin, G.D., Build and Fly Coaxial Heli Using Blender and MBDyn.
http://www.baldwintechology.com/video/piloted_coax_tutorial.mp4 accessed 23 March 2009.