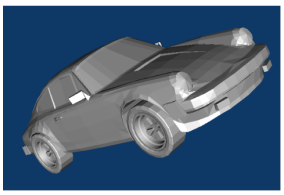
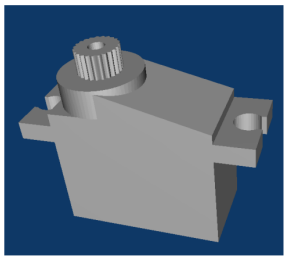
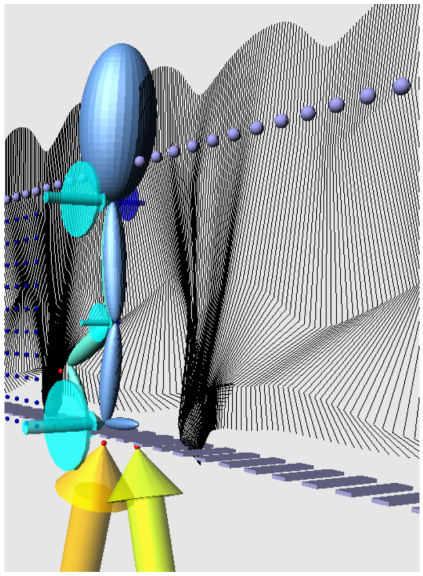
Solvere 4D Multibody Dynamics
Post-processor Manual

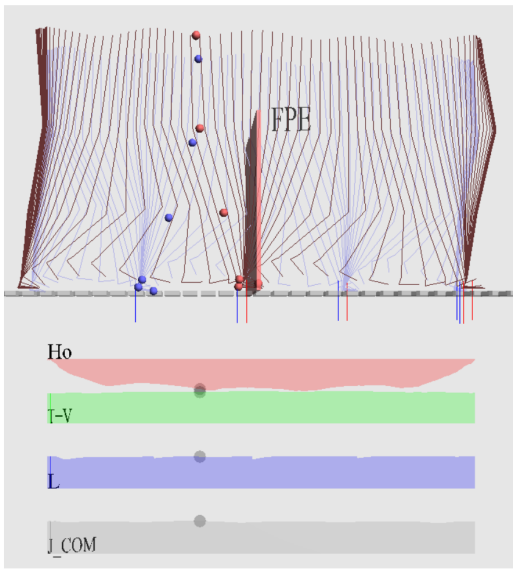
Matthew Millard
University of Waterloo
200 University Ave West
Waterloo ON, N2L 3G1, Canada
mjhmilla@engmail.uwaterloo.ca

October 9, 2008









Abstract

Solvere4D is a powerful, open-source, freely available post-processor that makes it easy for anyone to generate 3D animations of their data. Solvere4D is particularly designed for people who are in multibody dynamics, or kinesiology. Solvere4D can help you easily create intuitive research-grade 3D animations that include integrated plotting, force/torque visualization and graphical kinetic and kinematic history displays to enhance the insight gained from your data. Play back controls are embedded to allow you to play/pause the animation, speed or slow the rate of play back and randomly access different times in the animation using a slider. Solvere4D also includes powerful command-line tools that allow you to automate the creation of your 3D animations. With Solvere4D you can quickly and efficiently generate animations that allow you to see the equivalent of 20 or more data plots in one intuitive animation. "Solvere" is Latin for "to free" or "to release", accordingly Solvere4D is offered as free, open-source software under the GNU GPL Licence. This means to you can learn how Solvere4D works and add your own code to it to improve it.

Contents

1 Introduction

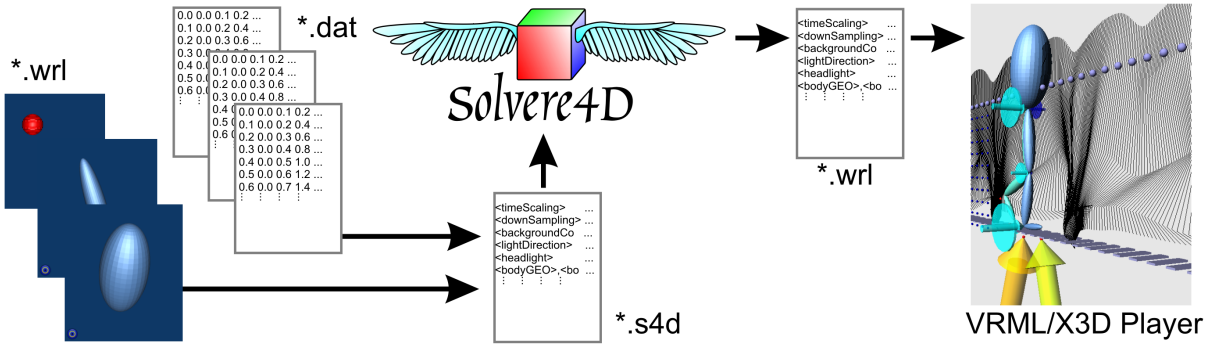


Figure 1: Solvere takes data files (*.dat), geometry files (*.wrl) a configuration file (*.s4d) and turns it into a 3D animation file (*.wrl). This animation file can be viewed using a number of different VRML/X3D players.

Solvere4D will take your exported data, body geometry, a configuration file and turn it into a 3D animation file as shown in Fig. 1. This animation file can then be played by any number of different VRML/X3D viewers. The remainder of this section will describe how to run Solvere4D and how to view 3D VRML movies. Section 2 onwards will describe how to use all of Solvere4D's features and how to contribute to Solvere4D.

1.1 Playing VRML/X3D Files

Solvere4D creates 3D animation files, or Virtual Reality Markup Language (VRML) files. These files on their own are not useful, but they contain commands that a VRML/X3D viewer can translate into 3D animations. There are many different VRML viewers available to you:

Software	Windows	Linux	Mac
Cosmo Player	X		
BSContact	X	X	
BS Contact J	X	X	X
Octaga Player	X	X	X
InstantPlayer	X	X	X
Cortona3D	X		X
Vivaty Player	X		
SwirlX3D	X		
FreeWRL		X	X
OpenVRML		X	X
Xj3D	X	X	X
Orbisnap	X	X	X
Demotride	X		

The preceding list can be found with hyper-links to each program at `http://cic.nist.gov/vrml/vbdetect.html`. I have tried out BSContact, Octaga, InstantPlayer, Cortona3D, SwirlX3D and FreeWRL. The best player by far is BSContact's player for the Windows platform, next I would recommend Octaga. The main difference between the players is how much of the VRML specification they implement. Some will not display text making it impossible to view the animations current time among other things. Some will not implement a "touchSensor" node, making it impossible to press the "Play/Pause", "Slow Down/Speed up" buttons.

If you cannot use BSContact it may take you some time to find a good VRML viewer. Once you do have a good VRML viewer you can look at the 3D world for the first time:

1. Open "Examples/experimental_data/fpe_research.wrl".
2. Use the controls pictured in Fig. 2 to run the animation:
   - Play/Pause: Press the blue button. By default the animation is paused to begin. You may need to click on the screen above the time controls before clicking the blue button to start.
   - Increase rate of play: Press the up triangle
   - Decrease rate of play: Press the down triangle
   - Set the current animation time: Click and drag on the grey time pointer that runs along the time slider
   - Note: the rate of play is displayed on the left of the display, while the current animation time is displayed in hours:minutes:seconds:milliseconds format.
3. Navigate around in the world:
   - Use the arrow keys to move forward/backwards and look left/right
   - Try using the other navigation methods to move around. The viewer should have a way of changing the navigation mode (right-click, then select the "Movement" menu in BSContact). There are several different modes available in VRML worlds:
     (a) Walk
     (b) Slide
     (c) Examine
     (d) Fly
     (e) Pan
     (f) Game-like

     Try all of them to see what they do, and what kinds of movement they afford.
4. Open "Examples/simulation_data/airWalkHat.wrl" and run it to see force and torque vector animations, and an animation that involves more camera movement.

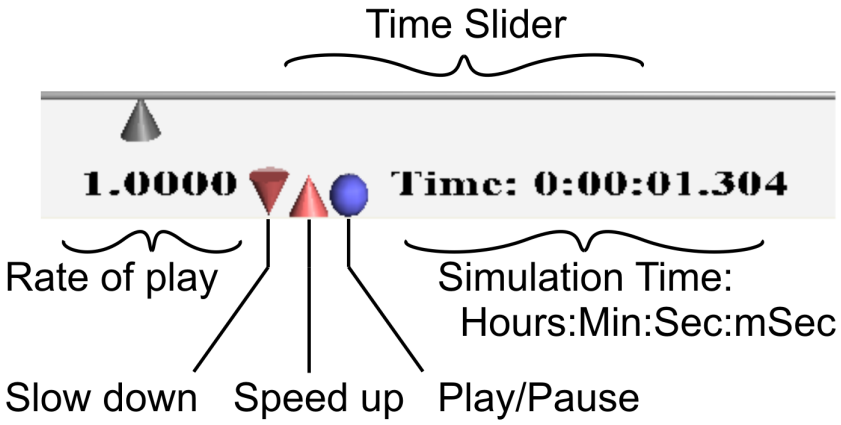


Figure 2: The animation can be played/paused, the rate of play adjusted (sped up and slowed down) and the current animation time can be manually adjusted using the controls that Solvere4D adds to the animation.

1.2 Running Solvere4D

Solvere4D can be run in two ways:

1. **GUI:** Double clicking "runAnimation.bat" on a Windows machine will launch the GUI. If you are not using windows, open up the *.bat file and make the necessary syntax adjustments to run it on your OS.

2. **Command-line:** Issuing a "java Solvere4D.Solvere4D C:/full/path/to/configfile/test.s4d" from within Solvere4D's "build/classes" directory will cause Solvere4D to read the specified *.s4d configuration file and create an animation titled "test.wrl" in the same directory as the *.s4d file.

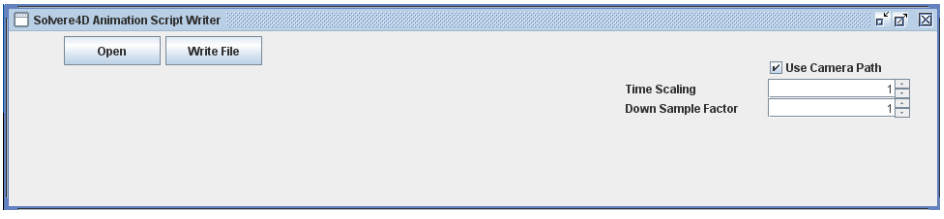


Figure 3: Solvere4D simple GUI. Some basic parameters like the time scale, down-sampling and the choice of using the camera path can be chosen here. Pressing the 'Write File' button will cause Solvere4D to write a 3D animation file (*.wrl) to the same directory that the configuration (*.s4d) file resides in.

You should try launching Solvere4D's GUI, and using it to write a VRML file using the steps below. A screen shot of the simple GUI is shown in Fig. 3.

1. Double click the "runAnimation.bat" file
2. Use the "Open File" dialogue to navigate to the "Manual/Examples/experimental_data" folder and choose the "fpe_research.s4d" file.
3. Press the "Write File" button
4. Go to the "Manual/Examples/experimental_data" folder and look at the "fpe_research.wrl" files timestamp: it should be from just a minute or two ago. View the *.wrl file using your VRML viewer of choice

You should also try to run Solvere4D from the command line. This is a particularly useful skill to learn because you can use other programs such as Matlab to run Solvere4D as detailed in Appendix C. This can allow you to completely automate the production of 3D visualizations of your data. I have used this technique to generate 1480 3D videos of experimental data (similar to the "fpe_research.wrl" files) using Matlab and Solvere4D in the span of a few hours. To run Solvere4D from the command line simply follow these steps:

1. Open a command shell. A command shell can also be called a command prompt, terminal or window in case you have not heard the term 'shell'. In Windows follow these steps
   (a) Press "Start"
   (b) Select "Run"
   (c) Type "cmd" in the little window
   (d) Press "OK"
2. Move the command shell to Solvere4D's "build/classes" folder. In Windows use the following steps:
   (a) Open a Windows folder
   (b) Navigate to Solvere4D's "build/classes" folder
   (c) Select the text in the "Address" bar and press "Ctrl C" to copy it
   (d) In the command prompt type the text "cd ", which stands for "change directory"
   (e) Use the mouse to put the cursor after the "cd " in the command shell, then use the mouse to right-click. A menu will pop up, select "Paste". This should put your whole address in the line.
   (f) Press "Enter". For example:

   ```
   cd C:\Solvere4D\build\classes
   ```

3. In the command shell type the text

   ```
   java Solvere4D.Solvere4D
   ```

4. If you press "Enter" now, you will launch the GUI. You'd rather learn to use the script, so get the full path to you *.s4D file and paste it in. For example

```
java Solvere4D.Solvere4D C:\Solvere4D\Manual\Examples\experimental_data
```

5. Add a slash and the name of the *.s4d configuration file. For example:

```
java Solvere4D.Solvere4D
"C:\Solvere4D\Manual\Examples\experimental_data\ fpe_research.s4d"
```

   The carriage-return should be ignored above, as this text should be on one line in the command shell.

6. Now press enter. If all goes well the text below will pop up, and one or two seconds later your new *.wrl file will be created and ready to use.

```
Copyright Matthew J.H.Millard 2008
  Solvere4D is offered under the GPL V3
  for more information visit http://www.gnu.org/
```

1.3 Generating Geometry Files

Solvere4D needs VRML97 files containing 3D representations of the bodies you wish to animate. If you have never drawn a 3D shape before do not worry, it is not too difficult. Many 3D CAD packages can generate these files, and a free 3D sculpting program known as Blender export VRML97 `http://www.blender.org/`. If your CAD package cannot write a VRML97 file it can probably export a *.stl file. Blender can import *.stl files and then export a VRML97 file for you.

For mere convenience it is worth knowing the commands to draw the basic VRML primitives (sphere, cylinder, cone and box) for your own use. All of the primitive shapes in VRML are drawn for you in 'Examples/primitive_geometry' folder - open up the files in any text editor to see how each of these shapes are specified. Note: all units are in meters. Simple spheres were used to animate the positions of optical markers on a person as shown in the "fpe_research.wrl" file — even simple markers can be quite effective at displaying motion.

As you produce more sophisticated geometry you will want to ensure that the centroid of the *.wrl part is at the same location as it is in your data — else the part will be animated with the wrong initial orientation or perhaps with an offset. The position of the parts centroid can be edited in Blender or in the CAD package you happen to be using.

2 Writing Solvere *.s4d Configuration Files

The following subsections will explain in detail how to write the *.s4d configuration file that Solvere4D requires to write a 3D animation of your data. Solvere4D uses this configuration file and the data files to animate your data. This configuration file points Solvere4D to the files that hold the data you wish to animate as well as telling Solvere4D how to display it. There are some general rules about how these data files should be formatted in order for Solvere4D to use them:

1. All data files should be in the same folder as the *.s4d configuration file.
2. All data files must be tab delimited.
3. Every data file that includes time as the first column must have the same time as every other file.

2.1 Pre-amble Tags

Tag	Arguments	Description
<timeScaling>	> 0	This scales the simulation time. A scaling of 2.0 would turn a 10 sec animation into a 20 second animation.
<downSampling>	Integer > 0	The integer value here will down sample *all* of the data you insert into the animation. A value of 1 will not downsample the data, a value of 2 will take every second point, 3 every 3^{rd} point, etc ...
<backgroundColour>	R# G# B# #:0.0-1.0	The values placed after the letters 'R', 'G', and 'B' correspond to an colour in Red-Green-Blue colour space. Decimal values of range 0-1 are used here. See appendix for a short primer on the RGB colour space.
<lightDirection>	<X# Y# Z#> #:0.0-1.0	If you would like a white light shone on your scene, include this followed by a vector direction. For example to shine a light along the X axis you would enter $< X1.0Y0.0Z0.0 >$.
<headlight>	Integer: [0,1]	A value of 0 will turn off the headlight, a 1 will turn it on

Examples of pre-amble tags for the *.s4d configuration file can be found below and in Appendix B.

<timeScaling>, 1.0
<downSampling>, 2
<backgroundColour>, <R0.90 G0.90 B0.90>
<lightDirection>, <X-0.5 Y-0.5 Z-1>
<headlight>, 0

2.2 <bodyGEO>,<bodiesMOV>: Kinematic Animation Tag

This simple tag will take geometry that you would like to animate, and move it through space in the manner you desire. Currently Solvere4D only accepts VRML97/VRML 2.0 (but not VRML 1.0) files to specify its geometry. Many 3D CAD packages can generate these files, and a free 3D sculpting program known as Blender export VRML97 `http://www.blender.org/`. With a little work Solvere4D can be upgraded to accept X3D files, which is the latest upgrade to VRML97. For mere convenience it is worth knowing the commands to draw the basic VRML primitives (sphere, cylinder, cone and box) for your own use. All of the primitive shapes in VRML are drawn for you in 'Examples/primitive_geometry' folder - open up the files in any text editor to see how each of these shapes are specified. Note: all units are in meters.

To make your geometry move, you must supply a text file that contains the position and orientation information about your part. The data file must be tab-delimited have the following format:

Column	1	2	3	4	5	6	7	8	9	10	11	12	13
	Time	X	Y	Z	r11	r12	r13	r21	r22	r23	r31	r32	r33

Where r11,r12,r13 ...r33, are the entries of a rotation matrix entered row wise so that r12 refers to the entry in row 1, column 2. An example of such a file can be found in the file 'Examples/simulation_data/HatPosOrien.dat'. Typically this data is generated by your simulation, or gathered from experiments.

Once your geometry text is ready, and the data file required to move the part you need to make an entry in the *.s4d configuration file so that Solvere4D can include it in the final animation. For geometry filed called 'redsphere.wrl' and a movement file called 'M1.dat', and another 'M2.dat' the entry would look like this:

<bodyGEO>,<bodiesMOV>
redsphere.wrl, M1.dat
redsphere.wrl, M2.dat
<\bodyGEO>,<\bodiesMOV>

Any subsequent geometry and movement file pairs should be added on a separate line as shown in Appendix B.

2.3 <forceTorque>,<genForceTorquePlots>: Force/Torque Animation Tag

The animation of forces and torques is automatically done for you in Solvere4D, you only need to provide the force and torque histories in a file and decide how you would like them displayed. Force vectors are drawn as an arrow and torques as an arrow with a disc with colour being dictated by the orientation of the vector as shown in Fig. 4. Force and torque vectors have their size scaled with the their respective magnitudes; it is up to you to choose the scaling that is most appropriate for your data.

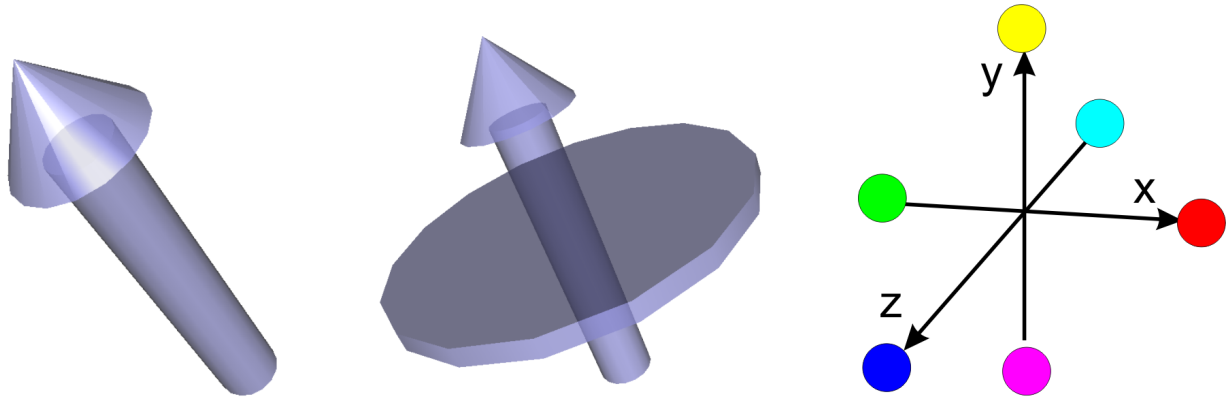


Figure 4: Forces are represented with an arrow, torques as an arrow with a disc. The colour of the vectors changes as a function of the orientation - the colour of the major axis is shown above. Orientations between the cardinal directions will have an interpolated colour.

Forces and torques are loaded into Solvere4D using a tab delimited file. The expected columns for the file are shown below.

Column	1	2	3	4	5	6	7
	Time	F_X	F_Y	F_Z	T_X	T_Y	T_Z

Once your data file is ready, you need to tell Solvere4D how to render your force/torque vectors using a series of statements. An example of this is shown below:

<**forceTorque**>,<**genForceTorquePlots**> normF=1000.0 normT=500.0 normD=1.0
toeRightForces.dat, <**f**0 **w**0 **c**0 **R**0.0 **G**0.0 **B**0.5 **T**0.5> <**t**0 **w**1 **c**1 **R**1.0 **G**0.5 **B**0.0 **T**0.5>

heelRightForces.dat, <f0 w0 c0 R0.0 G0.0 B1.0 T0.5> <t0 w1 c1 R0.0 G0.5 B 1.0 T0.5> <\forceTorque>,<\genForcePlots>,<\genTorquePlots>

The very first line contains the settings that control how the forces entered in the *.dat file are turned into size representations:

Setting	Arguments	Description
normF=	# > 0	All force vectors will be normalized using the value entered in normF
normT=	# > 0	All torque vectors will be normalized using the value entered in normT
normD=	# > 0	The normalized versions of the force and torque vectors will be multiplied by normD in order to set the vectors final size

If you look at the example above you will note that the forces are scaled such that a 1m long vector represents a force of 1000 N, and a torque vector with a 1m diameter disk would have a torque of 500 Nm. Note that Solvere4D does not care about the units, they were used in the previous sentence for clarity only.

In addition to simply rendering the force and torque vectors, you can have a persistent 3D plot of the force and torque vectors drawn in your animation. When used sparingly, this feature can be very useful for examining the time history of the forces and torques in more detail than is possible with the animation alone. The tags following the file entry control way in which the force/torque histories are displayed. For example, the tags that control how the force/histories are displayed for the first file are:

<f0 w0 c0 R0.0 G0.0 B0.5 T0.5> <t0 w1 c1 R1.0 G0.5 B0.0 T0.5>

The flags shown above have the following meaning:

Flag	Arguments	Description
f	0 or 1	Render force histories(1) or do not(0)
t	0 or 1	Render torque histories(1) or do not(0)
w	0 or 1	Render as wire frame (1) or as a solid (0)
c	0 or 1	Render in one colour (1) or using the space-mapped colours used for the vectors (0)
R	0.0-1.0	Red
G	0.0-1.0	Green
B	0.0-1.0	Blue
T	0.0-1.0	Transparency: 0 is solid, 1 is transparent

An example of a force/torque file can be found in the 'Examples/simulation_data' along with

the required entries in the *.s4d configuration file.

2.4 <camera>: Viewer Position Animation Tag

The location of the 'camera' defines the viewpoint that you see on the screen. You can define the 3D trajectory of the camera to control precisely the view you see by including a file identical in format to that required to animate a rigid body:

Column	1	2	3	4	5	6	7	8	9	10	11	12	13
	Time	X	Y	Z	r11	r12	r13	r21	r22	r23	r31	r32	r33

By including a 3D trajectory for the camera you can view a moving mechanism conveniently (like a car or a walking person), however you are not restricted to stick to it: you can use the arrow keys to navigate relative to this path. This will allow you to zoom into a particular area of interest, or move around to a more appropriate angle while the animation is running. The default navigation is 'walk', which you can do using the arrow keys. There are several other modes of navigation which you can use: walk, slide, examine, fly, pan, game-like and jump. If you do not include the <camera> tag then your view will start up with the view at (0,0,10.0 m) looking down the negative Z axis.

The syntax required to include a trajectory file named 'cameraPosOrien.dat' in the *.s4d configuration file is shown below. An example of a camera trajectory file can be found in both the 'Examples/experimental_data' folder and the 'Examples/simulation_data'.

```
<camera>
 cameraPosOrien.dat
<\camera>
```

2.5 <markers>: Static Markers Tag

The static scene that does not move is important to clarify both the viewers and the figures orientation in 3D space. A tag has been developed solely for the purpose of including large arrays of spheres, cylinders, boxes and cones. To include an array of shapes an entry into the *.s4d configuration file is necessary, along with the inclusion of a file that contains a list of spatial locations for the markers. An example entry in the *.s4d file for each of the shapes is shown below:

```
<markers>
 gridMarkers.txt,    <sphere r0.02              R0.6 G0.6 B0.9 T0.0>
```

```
 xzGridMarkers.txt, <cylinder r0.01 h0.05     R0.0 G0.0 B0.8 T0.0>
 xyGridMarkers.txt, <cone r0.01 h0.05         R0.0 G0.0 B0.8 T0.0>
 floorMarkers.txt,  <box  x0.05 y0.01 z0.2  R0.6 G0.6 B0.8 T0.0>
<\markers>
```

The flags for each of the markers have the following meaning:

Flag	Arguments	Description
R	0.0-1.0	red
G	0.0-1.0	green
B	0.0-1.0	blue
T	0.0-1.0	transparency
		Sphere
r	# > 0	radius (m)
		Cylinder
r	# > 0	radius (m)
h	# > 0	height (m)
		Cone
r	# > 0	radius (m)
h	# > 0	height (m)
		Box
x	# > 0	x dimension of the box (m)
y	# > 0	y dimension of the box (m)
z	# > 0	z dimension of the box (m)

The markers are placed according to the locations specified in the *.dat file — a tab-delimited file — that precedes the marker geometry description. The marker location file has the following format, with all units in meters:

Column	1	2	3
	X	Y	Z

An example of both the marker *.dat files and the corresponding entries in the *.s4d file can be found in both the ‘Examples/experimental_data’ folder and the ‘Examples/simulation_data’.

2.6 <plot3D>: 3D Plotting Tag

Solvere4D features an integrated plotting tool that will allow you to combine your animations with data plots to save you from switching from multiple windows. The plotting utility has been de-

signed under the assumption that the animation best affords a display of qualitative rather than quantitative data, so numbers are not displayed on the graphs. An example of the type of output that the 3D plotting utility creates can be seen in Fig. 5 and also by running the file 'Examples/experimental_data/fpe_research.wrl'. The plotting utility comes with the built in capability of displaying a label for the graph, and also a marker that follows the current data point of the plot.

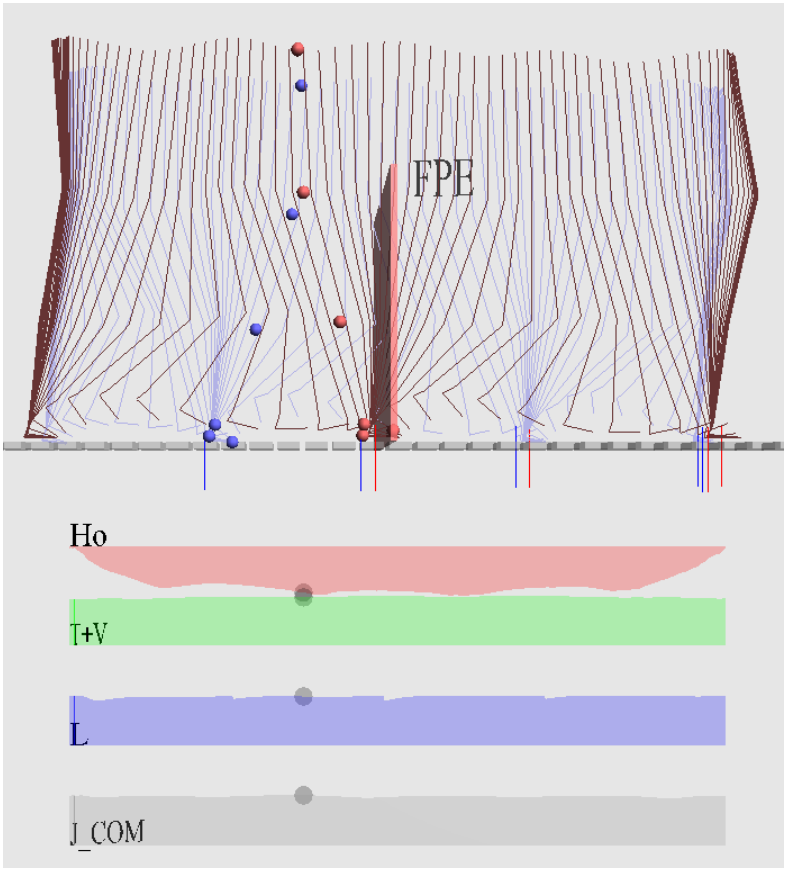


Figure 5: Four integrated data plots are shown in the figure labeled 'Ho', 'T+V', 'L', and 'J_COM'. These four plots are relevant to the underlying calculation that governs the location of the 'FPE' wall — thus saving the viewer from switching between multiple windows.

To create a 3D plot the data for the plot must be stored in a text file and an entry in the *.s4d configuration file must be made so that Solvere4D renders it properly. The plot data file should have the following format:

Column	1	2	3	4	5	6	7
	Time	X_1	Y_1	Z_1	X_2	Y_2	Z_2

Where (X_1, Y_1, Z_1) is by default associated with the reference axis, and (X_2, Y_2, Z_2) is the data point. The data marker (the grey dots in Fig. 5 will follow the path laid out by the series of

(X_2, Y_2, Z_2) points. An example entry in the *.s4d file to include and render a plot is shown below, and also in Appendix B.

<plot3D>
ankleLeftTorquePLOTY.dat,<w1 R0.0 G0.0 B0.5 T0.5 s0.075 m1 R1.0 G0.0 B0.0 "Ankle Torque" R0.0 G0.0 B0.0>
kneeLeftTorquePLOTY.dat, <w1 R0.0 G0.0 B0.5 T0.5 s0.075 m1 R1.0 G0.0 B0.0 "Knee Torque" R0.0 G0.0 B0.0>
hipLeftTorquePLOTY.dat, <w1 R0.0 G0.0 B0.5 T0.5 s 0.075 m1 R1.0 G0.0 B0.0 "Hip Torque" R0.0 G0.0 B0.0>
<\plot3D>

The plot tag entry is the most elaborate tag in the entire *.s4d configuration file. The following table will define, element by element, in the order they appear (from left to right), what each of the flags in the entries controls:

Flag	Arguments	Description
w	0 or 1	Wireframe (0), or solid (1)
R G B		RGB colour the plot is rendered in
s	# > 0	The scale of the plot label & marker
m	0 or 1	Add an animated marker (1) or not (0)
R G B		RGB colour of the marker
"Plot Label"	text	The name of the plot, in quotations
R G B		RGB colour of the label

If the use of the "R G B" flags to specify colour are unfamiliar to you please refer to Appendix A. An example of both the plot3D *.dat files and the corresponding entries in the *.s4d file can be found in both the 'Examples/experimental_data'.

2.7 <stickFigures>: Stick Figures Tag

Static arrays of lines can be printed to the screen using the <stickFigures> tag. This capability can be used to print the kinematic history of an animation to combine the advantages of a plots persistent data with the richness of an animation. This feature is used in 'Examples/experimental_data/fpe_research.wrl' and is shown in Fig. 5 to illustrate the kinematic history of the person as they jump. This feature was also used to show the location of foot contact (the blue vertical line) and the location where foot contact was predicted to occur (shown as a red vertical line).

To include stick figure drawings in your animation, you must include a data file that contains the geometry of the stick figures along with a corresponding entry in the *.s4d configuration file. The data file should have the following format:

Column	1	2	3	4	5	6	7	8 ...
	X_1	Y_1	Z_1	X_2	Y_2	Z_2	X_3	...

Any number of points can be included to draw a single stick figure. All of the points in one row are interpreted as belonging to a single stick figure, with subsequent stick figure coordinates being placed on a new line. The rows within one file must all contain the same number of coordinates — if you need to draw another stick figure that has a different number of points, then you must put that information in a new file. An example of the *.s4d entry to include a stick figure is shown below and in Appendix B.

<**stickFigures**>
motionL_SF.dat, <**R**0.7 **G**0.7 **B**0.9>
motionR_SF.dat, <**R**0.4 **G**0.2 **B**0.2>
FPE_SF.dat, <**R**1.0 **G**0.0 **B**0.0>
HFP_SF.dat, <**R**0.0 **G**0.0 **B**1.0>
<**stickFigures**>

The stick figure lines can be rendered in one colour, dictated by the values placed after the "R G B" flags. For more information refer to Appendix A.

Flag	Arguments	Description
R G B		RGB colour the stick figure lines

An example of both the plot3D *.dat files and the corresponding entries in the *.s4d file can be found in both the 'Examples/experimental_data'.

2.8 <movingLabels>: Moving Labels Tag

Occasionally it is useful to be able to label different portions of a particular animation, and to have those labels follow element of interest. To that end the <movingLabels> tag was created. The label that follows the red wall in the file 'Examples/experimental_data/fpe_research.wrl' was created using this tag.

To apply a moving label to an animation you need to have a data file that specifies where the label should be as a function of time. As with every external file used in Solvere4D this file should be tab delimited and it should have the same number of rows as every other text file that includes a "Time" column. The file should have the following format:

Column	1	2	3
Time	X	Y	Z

An entry in the *.s4d configuration file is required to command Solvere4D to include the label in the animation. An example of this entry is shown below:

<**movingLabels**>
FPELabel.dat, <"FPE" **s**0.25 **R**0.10 **G**0.10 **B**0.10>
<**movingLabels**>

An explanation for each of the flags used in the label tag are shown below:

Flag	Arguments	Description
"Label Text Here"	Text	The label text
s	# > 0	The scale of the plot label. A "1" will make 1 m tall lettering
R G B		RGB colour the plot is rendered in

3 Contributing to Solvere4D

Solvere4D is free software that is offered under the GNU Public License or GPL for short. That means that you, the user, have access to the source code. You may read the code in order to understand how Solvere4D works and improve it if you like. I have carefully documented every function and class and named variables in a helpful manner to make Solvere4D's implementation as understandable as possible. Solvere4D is written using Java, to make it compatible with different platforms. Solvere4D functions works in the following manner, and has different files associated with each function:

1. Read the *.s4d file and corresponding data files.
   - Solvere4D.java, *readAnimationData* function
   - SolvereUtilities.java, the following functions
     - *parseNumber*
     - *parseRGBT*
   - TextParser.java
2. Store the information required to build each element in a class, that functions like a structure. There is a one-to-one correspondence between *.s4d tags and these storage classes.
   - BodyData.java
   - ForceTorqueData.java
   - Plot3D.java
   - MarkerData.java
   - StickFigure.java
   - Label3D.java
3. Alter the data as necessary so that a VRML viewer can display it.
   - SolvereUtilities.java, with the following functions:
     - *convertToQuat*
     - *getAxisAngle*
     - *getColourMapping*
     - *getTriangularArray*
4. Write the data to a VRML file. This is done by reading in a VRML file for each tag element and replacing data in the default file with data specific to the users animation.
   - Solvere4D.java, *writeAnimationData* function
   - SolvereUtilities.java, with the following functions:
     - *appendColourTag*
     - *appendHeader*

  - *appendMovingLabel*
  - *appendOrientationTag*
  - *appendPlot*
  - *appendRouteColour*
  - *appendRouteRotation*
  - *appendRouteTranslation*
  - *appendScaleTag*
  - *appendStaticMarker*
  - *appendStickFigure*
  - *appendTimeControls*
  - *appendTranslationTag*
  - *getTextFile*
  - *mergStrings*
  - *replaceTags*
- All of the 'lib_' files in 'build/classes/WRL_SYNTAX'

Every function and every file is commented. If you're interested in adding a function to Solvere4D I suggest you go and download Sun's free Java developer environment and start stepping through the code to get a better feel for how it works. If you'd like to contribute but do not know what to contribute, here are a list of things I would like to add to Solvere4D, but have not yet:

1. Update Solvere4D to export X3D. This would come in 2 parts:

   (a) Create a series of 'lib_' files just as in 'build/classes/WRL_SYNTAX', but for *.x3d files and in a folder called 'build/classes/X3D_SYNTAX'

   (b) Update Solvere4D to output an X3D file - this involves upgrades to the 'readAnimationFile' function in Solvere4D.java and the 'append ...' functions in 'SolvereUtilities.java'.

2. Add the capability to display deformable geometry, and eventually muscle.

A Colour in Solvere4D

Many tags in Solvere4D allow you to set the colour of a particular element. In Solvere4D it is expected that you enter the colour using its Red-Green-Blue (RGB) colour space value. This involves entering 3 numbers that correspond to the right mix of red, green and blue in order to create the colour you are interested in. The table below contains some basic colour entries that you can use for your convenience:

RGB Text	Common Name
R0.0 G0.0 B0.0	Black
R0.0 G0.0 B1.0	Blue
R0.0 G1.0 B0.0	Green
R0.0 G1.0 B1.0	Cyan
R1.0 G0.0 B0.0	Red
R1.0 G0.0 B1.0	Magenta
R1.0 G1.0 B0.0	Yellow
R1.0 G1.0 B1.0	White
R0.1 G0.1 B0.1	Dark Gray
R0.5 G0.5 B0.5	Medium Gray
R0.9 G0.9 B0.9	Light Gray

Further adjusting these values will allow you to create what ever hue you want. For example, if you wanted a Sienna Brown you would use red with some green and blue: ”R0.63 G0.32 B0.18”. The internet is a great resource for finding RGB colour values, though the amount of each colour is usually specified as an integer from 0-255, that is as an 8 bit number. Thus Sienna Brown would be 160 82 45. To convert the values to 0-1 decimal values, simply divide each entry by 255. An excellent reference can be found in the document titled “Loreti_rgb.pdf” that is in Solvere4D’s “Manual” folder.

B Example s4d Configuration Files

The following shows: airWalkHat.sd4

```
<timeScaling>,     1.0
<downSampling>,     2
<backgroundColour>,  <R0.90 G0.90 B0.90>
<lightDirection>, <X-0.5 Y-0.5 Z-1>
<headlight>, 0
<bodyGEO>,<bodiesMOV>
    hat.wrl,     HatPosOrien.dat
    thighL.wrl,    HipLeftJointPosOrien.dat
    shinL.wrl,    KneeLeftJointPosOrien.dat
    footL.wrl,    AnkleLeftJointPosOrien.dat
    thigh.wrl,    HipRightJointPosOrien.dat
    shin.wrl,    KneeRightJointPosOrien.dat
    foot.wrl,    AnkleRightJointPosOrien.dat
    contact.wrl,    heelRightPosOrien.dat
    contact.wrl,    heelLeftPosOrien.dat
    contact.wrl,    toeRightPosOrien.dat
    contact.wrl,    toeLeftPosOrien.dat
<\bodyGEO>,<\bodiesMOV>
<forceTorque>,<genForceTorquePlots> normF=1000.0 normT=500.0 normD=1.0
    toeRightForces.dat,  <f0 w0 c0 R0.0 G0.0 B0.5 T0.5> <t0 w1 c1 R1.0  G0.5  B0.0   T0.5>
    heelRightForces.dat, <f0 w0 c0 R0.0 G0.0 B1.0 T0.5> <t0 w1 c1 R0.0  G0.5  B 1.0   T0.5>
    leftGRF.dat,         <f0 w0 c0 R1.0 G0.2 B0.2 T0.5> <t0 w1 c1 R1.5  G1.0  B0.0   T0.5>
    hipLeftTorque.dat,   <f0 w1 c1 R0.0 G1.0 B1.0 T0.5> <t0 w1 c1 R0.33 G0.0  B0.33  T0.5>
    hipRightTorque.dat,  <f0 w1 c1 R0.5 G0.5 B1.0 T0.5> <t0 w1 c1 R0.33 G0.33 B0.0   T0.5>
    kneeLeftTorque.dat,  <f0 w1 c1 R0.5 G0.5 B1.0 T0.5> <t0 w1 c1 R0.67 G0.0  B0.67  T0.5>
    kneeRightTorque.dat, <f0 w1 c1 R0.5 G0.5 B1.0 T0.5> <t0 w1 c1 R0.67 G0.67 B0.0   T0.5>
    ankleLeftTorque.dat, <f0 w1 c1 R0.5 G0.5 B1.0 T0.5> <t0 w0 c1 R1.0  G0.0  B1.0   T0.25>
    ankleRightTorque.dat,<f0 w1 c1 R0.5 G0.5 B1.0 T0.5> <t0 w1 c1 R1.0  G1.0  B0.0   T0.5>
<\forceTorque>,<\genForcePlots>,<\genTorquePlots>
<camera>
    cameraPosOrien.dat
<\camera>
<markers>
    gridMarkers.txt,  <sphere r0.02                    R0.6 G0.6 B0.9 T0.0>
    floorMarkers.txt,  <box  x0.05 y0.01 z0.2     R0.6 G0.6 B0.8 T0.0>
    xyGridMarkers.txt, <sphere r0.01                   R0.0 G0.0 B0.8 T0.0>
<\markers>
#<plot3D>
    ankleLeftTorquePLOTY.dat,<w1 R0.0 G0.0 B0.5 T0.5  s0.075 m1 R1.0 G0.0 B0.0 "Ankle Torque" R0.0 G0.0 B0.0>
```

```
    kneeLeftTorquePLOTY.dat, <w1 R0.0 G0.0 B0.5 T0.5  s0.075 m1 R1.0 G0.0 B0.0 "Knee Torque" R0.0 G0.0 B0.0>
    hipLeftTorquePLOTY.dat,  <w1 R0.0 G0.0 B0.5 T0.5  s0.075 m1 R1.0 G0.0 B0.0 "Hip Torque" R0.0 G0.0 B0.0>
<\plot3D>
<stickFigures>
    stickfigure.dat, <R0 G0 B0 >
<\stickFigures>
#<movingLabels>
    larryLabel.dat, <"Larry" s0.5 R0.0  G0.0  B1.0>
<\movingLabels>

Anything that is not betwetween a recognized tag such as those above,
int the angled brackets, is treated as a comment.

If the tags in the square brackets have any other character (such as the
"#" in front of the moving Label) then the whole block is ignored. At
the moment there is no way to comment out a single entry for a given
tag, you just have to move it outside from its tagged region.
```

fpe_research.s4d

```
<timeScaling>,     1.0
<downSampling>,     4
<lightDirection>,  <X-0.5 Y-0.5 Z-1>
<headlight>, 0
<backgroundColour>, <R0.9 G0.9 B0.9>
<bodyGEO>,<bodiesMOV>
    redsphere.wrl,    M1.dat
    redsphere.wrl,    M2.dat
    redsphere.wrl,    M3.dat
    redsphere.wrl,    M4.dat
    redsphere.wrl,    M5.dat
    redsphere.wrl,    M6.dat
    bluesphere.wrl,    M7.dat
    bluesphere.wrl,    M8.dat
    bluesphere.wrl,    M9.dat
    bluesphere.wrl,    M10.dat
    bluesphere.wrl,    M11.dat
    bluesphere.wrl,    M12.dat
    redplane.wrl,    FPEX.dat
<\bodyGEO>,<\bodiesMOV>
<camera>
    cameraPosOrien.dat
<\camera>
<plot3D>
    Ho.dat,         <w0 R0.5  G0.0  B0.0  T0.75  s0.15 m1 R0.0 G0.0 B0.0 "Ho"  R0.0  G0.0  B0.0>
    TpV.dat,        <w0 R0.0  G0.5  B0.0  T0.75  s0.15 m1 R0.0 G0.0 B0.0 "T+V" R0.0  G0.0  B0.0>
    L.dat,          <w0 R0.0  G0.0  B0.5  T0.75  s0.15 m1 R0.0 G0.0 B0.0 "L"   R0.0  G0.0  B0.0>
    JCOM.dat,       <w0 R0.25 G0.25 B0.25 T0.75  s0.15 m1 R0.0 G0.0 B0.0 "J_COM" R0.0  G0.0  B0.0>
<\plot3D>
<movingLabels>
    FPELabel.dat, <"FPE" s0.25 R0.10  G0.10  B0.10>
<\movingLabels>
<markers>
    floorMarkers.txt, <box x0.09 y0.03 z0.2 R0.6 G0.6 B0.6 T0.50>
<\markers>
<stickFigures>
    motionL_SF.dat,    <R0.7 G0.7 B0.9>
    motionR_SF.dat,    <R0.4 G0.2 B0.2>
    FPE_SF.dat,        <R1.0 G0.0 B0.0>
```

HFP_SF.dat,
<\stickFigures>
<R0.0 G0.0 B1.0>

C Generating Animations using Solvere4D From Matlab

Although Solvere4D cannot be called directly from Matlab, a *.bat file that runs Solvere4D can be called from Matlab. Matlab can be used to write and run a *.bat file (which is just a text file with an extension of *.bat) that will run Solvere4D. While the data files may be new for every animation you create, you might have a number of static files (*.wrl geometry files, marker location files, the *.s4d configuration file) that you need in order to create your animation file.

```
%%
% Write the VRML data files
%%

if flag_genVRMLFiles == 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %
 % 1. Copy over the files in the support directory.
 %   This includes all files EXCEPT the *.dat files
 %
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    tempDir = pwd;

    aniDir = ['C:\mjhmilla_TEMP\FPE\Animation'];
    cd(aniDir);
    supDir = [aniDir,'\suppfiles'];

    subDir = ['S',num2str(i)];
    expDir = expName;
    trialDir = ['T',num2str(k)];

    %%
    %Get into the proper directory
    %%

    if isdir(subDir)
         cd(subDir);
    else
   mkdir(subDir);
         cd(subDir);
    end

  if isdir(expDir)
         cd(expDir);
    else
```

```
    mkdir(expDir);
         cd(expDir);
     end

  if isdir(trialDir)
         cd(trialDir);
     else
    mkdir(trialDir);
         cd(trialDir);
     end

  fileDir = pwd;

%Now we are in the directory we would like to
  %have the support files in. Time to get DOS to
  %copy them over for us

     dos(['xcopy ',supDir, ' /e /Y']);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2. Generate your data and write it to the appropriate
%    *.dat files using the following command for each
%    matrix of data:
%
%     dlmwrite(['filename.dat'], matrixName, '\t');
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 3. Create the *.bat file required to run Solvere4D
%    then call it from DOS to create our new animation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


     goToS4D = 'cd C%s\\Solvere4D\\build\\classes';

     runRunSolver4D = ['java Solvere4D.Solvere4D', ...
                       ' C:/exampleFolder/exampleFile.s4d'];
     goToHomeDir = pwd;
     goToHomeDir = 'cd ' + curDir;
```

```
%Write the *.bat file
    fid = fopen('createFPE.bat', 'w');
    fprintf(fid, goToS4D,':');
    fprintf(fid, '\n');
    fwrite(fid, runSolvere4D);
    fwrite(fid, goToHomeDir);
    fclose(fid);

%Run the *.bat file to get Solvere4D to create the animation
    dos('createFPE.bat');

%Rename the animation file and copy it to another directory
    dos(['copy exampleFile.wrl ', 'C:\AnotherFolder\']);
end

%%
% Done Writing the VRML data files
%%
```