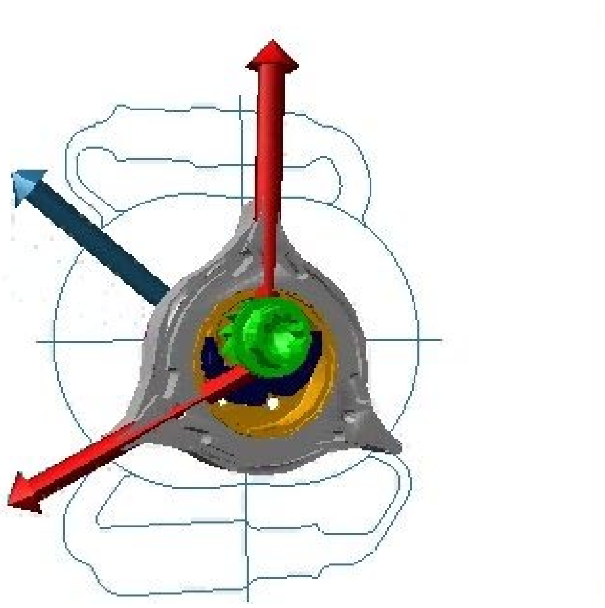
Applying Timed, Staggered Pressure Forces In a Wankel-Principle ADAMS Model.

Sam McDonald, TRW VSSI
Rohit Tangri, TRW VSSI
Ravi Guttal, Automated Analysis Corp.
Steve Mornelli, MDI

1998 European ADAMS Users' Conference
November 18, 1998



ABSTRACT

It is sometimes said, "Timing is everything!", and ADAMS is a good tool for helping to find causes of timing problems in mechanisms where timing is critical. In this paper, a mechanism model is looked at with ADAMS. The modeling of the Wankel-Principle mechanism in ADAMS, as well as the application of the pyrotechnic pressure forces, the VARIABLEs used to trigger them, and the contact forces between the axle and housing are discussed.

This paper shows three techniques that have been useful in modeling multiple state variable triggered pressure forces driving a mechanism in an ADAMS model. The three techniques described here are:

1. Converting the pressure to a force.
2. Capture the beginning time(t0) of the triggered pressure.
3. Modeling the contact between the pawl and the axle.

The Model

The model is based on using the Wankel principle to drive a piston around a housing, which then uses a pawl to engage a ratchet with a gear set on the axle to transfer the pistons rotary action to an axle.

The Wankel-principle allows a coupling between the piston rotation and the housing rotation that is not necessarily 1:1. Equation 1 shows the formula.

$$\theta_a = C * \theta_p$$

Equation 1.

Where θ_a is the rotational angle of the axle, and θ_p is the rotational angle of the piston, and C is the Wankel gear ratio. This axle is connected to the seatbelt webbing, which is then tensioned due to this action. The driving pressures are provided in the form of pressure-time curves that are independently triggered based on one of the piston tip displacements. The model is used to check the effects of differing the trigger point for the initiation of the pressures, and to investigate the forces arising from initial pawl contact with the axle.

Pressure Forces

The equation describing the force arising from uniform pressure on a flat plate is:

$$F_p = A * P$$

Equation 2.

Where F_p is the force on the plate due to the pressure, *A* is the area of the plate and *P* is the pressure. In this model, the pressure is a given spline, and the area will come from the model graphics.
To get a good representation of the forces arising on a given body, the body surface itself needs to be modeled. In ADAMS, this can be done with many different graphics, but the most versatile graphic is the SHELL graphic.

The SHELL graphic is a faceted surface defined by specifying the coordinates of the vertices, and then defining each facet by connecting the vertices. The .SHL file (the file that normally defines a SHELL) looks like Figure 1.

```
4 1 1.00000
0.0     0.000  0.000
1.0     0.000 0.000
0.0    1.000 0.000
1.0    1.000 0.000
3 1 2 4
3 1 4 3
```

Figure1.

The first line in Figure 1 lists the number of vertices, (4) the number of elements/facets (2) and a conversion factor that gets applied to the vertices. The next four lines give the coordinates for vertices one through four, and the last two lines specify the two facets. The first facet has 3 sides, and connects vertices 1, 2, and 4. The shell in the above example would be flat in the x-y plane, with the facets normals in the Reference Markers Z direction.

When a stereo lithography file is read into ADAMS, it becomes a shell file. Thus, Shell files are the easiest way to get your information into VIEW. Figure 2 shows the shell file used to model the piston surface on which the pressure acts in this model.

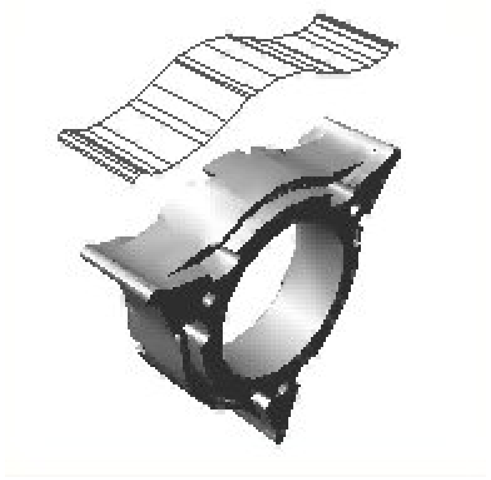


Figure 2.

The idea behind using the shell is to sum up the individual forces on each of the individual elements. The first step in the summation is to find the normal for each shell facet, and the find the area for each shell facet. These parameters do not change with time, and can be computed once during the initialization flag. The pressure is then coded as a special state variable that can be ‘turned on’ when a control variable reaches a certain point. The variable triggering process will be discussed in more detail in the next section of the paper.

The shell is defined with respect to a Reference Marker on the piston, and the shell is fixed with respect to that marker. The marker is also fixed with respect to the piston, and so the normals are invariant over time with respect to the piston frame. The areas of each facet are also time invariant. Only the pressure will change with time. Thus, a force vector for the shell can be determined during the initial pass through the GFOSUB and used throughout the analysis, as long as the GFORCE's Reference Marker is on the piston.

If the bounding shell (or polyline) is known, then the method can be extended to determine the pressure (with a few assumptions and the ideal gas law.) throughout the process. This is the logical next step, but was not done in this model.

State Variable Timing

Many times it is desired to determine when an event happens during the analysis and use the time when the event happens as a variable in some function. It is not easy to do this using only modeling elements from VIEW, or an ADAMS dataset (though it is possible). We desired to make the process easy to set up, easy to understand, easy to use, and easy to verify that it was working correctly.

It was decided to use variables as the method of data input. VARIABLEs are bare FUNCTIONS that can have initial conditions, can depend on any other state variables, and can use a VARSUB to handle the complicated things. The technique we use is to require three variables:

1. T_0 VARIABLE which is to capture and hold the desired value.

2. Triggering VARIABLE.

3. Value VARIABLE, provides the value that will be held by T0

The value variable was initially visualized as being set equal to time, but could be any ADAMS State variable. Currently the variables are set up manually in View, but the desire is to create an accompanying macro and dialog box to finish making the setup simple.

The variables will be set up transparently to the VIEW user (Sorry, dataset buffs) and will be modifiable in the View environment. It is assumed that the user will be utilizing results output, which should automatically trap for output of all three variables.

When the value of the triggering VARIABLE goes negative at the end of a converged integration step, the Value VARIABLE gets its instantaneous value trapped and stored. If the Triggering VARIABLE is going negative and the Value VARIABLE has not been set, then the current value of the Value VARIABLE is used. The storage of the Value VARIABLE only happens in a SENSUB.

The subroutines work in the following fashion. The VARSUB calls a subroutine in which the data is stored. This parking subroutine will return the stored value if the VARIABLE has been triggered before, return the current time if it has not been triggered, or return the current value if the current iteration shows potential triggering. The parking subroutine is also called by the SENSOR subroutine that will check to make sure that the value stored will be from a converged output step. This is a common precaution that must be used in some form whenever storing away state variables in a subroutine for later use, to ensure that bad data from an unconverged step will not ruin the simulation information from that point.

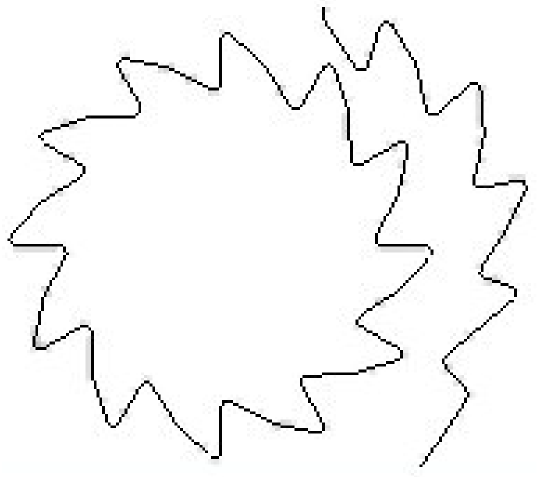


Figure 3.

Ratchet-Pawl Contact

Initially the contact between the axle and the pawl was modeled with a rotational SFORCE using a standard type of rotational stiffness, multiplied by a step dependent on one of the variables mentioned above. This worked, but induced unrealistic oscillations into the piston motion. It was decided to go back to detailed modeling of the contact. The surface of the axle was modeled with a closed polyline, and the pawl was modeled with an open polyline. The contact forces were modeled with one GFORCE per part, with the vertices of the I part contacting the lines of the J part. Thus, the polyline to polyline contact required two GFORCEs. The polylines are shown in figure 3.

The GFORCES are easy to define with the macro given in the appendix a. This macro requires the user to specify two GFORCE names, pick the two polylines that are to come into contact, and then select(create) a contact array. The contact array is the same as that used in the ADAMS 9.1 contacts. It contains the information needed for the impact function, as well as the coulomb friction information.

Conclusions

The model was first correlated with one set of pressure curves by modifying a viscous torsion damper that was used to represent the friction in the mechanism. The timing of the triggering of two of the pressure curves was then verified. A further check on the correlation will be to run the model with the same friction coefficient and different pressure curves, and then verify that the results match.

Appendix A
Macros to define polyline-polyline contact

Setup2.cmd

```
Macro read file="cf_pp_g.cmd" macro=mac cre=yes
interface menubar read menubar=.gui.main.mbar file="new.mnu"
file command read file="xcplpl.cmd"
```

cf_pp_g.cmd

```
!USER_ENTERED_COMMAND contact_poly_poly
!HELP_STRING write the gforce statements for the surface contact.
!WRAP_IN_UNDO yes

!$CONTACT_NAME_I:T=new_general_force:a
!$CONTACT_NAME_J:T=new_general_force:a
!$POLYLINE_1:T=POLYLINE
!$POLYLINE_2:T=POLYLINE
!$CONTACT_ARRAY:T=ADAMS_ARRAY

defaults model model=($polyline_1.parent.parent)

if cond=(db_exists("sfm_contact_counter"))

else
     variable create variable=sfm_contact_counter  &
       integer = 0
end


defaults model part=($polyline_1.PARENT)

variable create variable=junk string = (eval($polyline_1.parent))

if cond=(db_exists(junk//".poly_rm"))

   var cre var=ply1 str=($polyline_1.parent.poly_rm.remarks)

else

  if cond=(sfm_contact_counter==2)

    marker create marker=poly_rm   &
       location = 0,0,0  &
       orientation = 0,0,0 &
       relative_to = ($’polyline_1’.parent) &
       comment = sfm_contact_array_3

    variable modify variable=sfm_contact_counter integer=3
    data_element create array ic_array &
     array= sfm_contact_array_3        &
     numbers = ($’polyline_1’.location)

  else

    marker create marker=poly_rm   &
       location = 0,0,0  &
       orientation = 0,0,0 &
       relative_to = ($’polyline_1’.parent) &
       comment = sfm_contact_array_1

    variable modify variable=sfm_contact_counter integer=1

    data_element create array ic_array &
     array= sfm_contact_array_1        &
     numbers = ($’polyline_1’.location)

  end
    variable create variable=ply1
str=($polyline_1.parent.poly_rm.remarks)
end


defaults model part=($polyline_2.PARENT)
variable create variable=junk2 string=(eval($polyline_2.parent))

if cond=(db_exists(junk2//".poly_rm"))

   variable create variable=poly2
str=($polyline_2.parent.poly_rm.remarks)
else

   if cond=(sfm_contact_counter==2)

      marker create marker=poly_rm   &
         location = 0,0,0  &
         orientation = 0,0,0 &
         relative_to = ($’polyline_2’.parent) &
         comment = sfm_contact_array_3

      variable modify variable=sfm_contact_counter integer=3

      data_element create array ic_array &
       array= sfm_contact_array_3        &
       numbers = ($’polyline_2’.location)

   else

      marker create marker=poly_rm &
         Location = 0,0,0          &
         orientation = 0,0,0       &
         relative_to = ($’polyline_2’.parent) &
         comment = sfm_contact_array_2

      data_element create array ic_array &
```

```
       array = sfm_contact_array_2 &
       numbers = ($’polyline_2’.location)

      variable modify variable=sfm_contact_counter int=2

   end

   variable create variable=ply2
str=($polyline_2.parent.poly_rm.remarks)
end


 force create direct general_force &
 general = $contact_name_i &
 i_marker_name=($polyline_1.parent.poly_rm) &
 j_part_name=($polyline_2.parent) &
 ref_marker_name=($polyline_2.parent.poly_rm) &
 user_function=1,1

 force modify direct gen &
 general = $contact_name_i &
 user=7,($contact_name_i.i_marker_name.adams_id), &
($contact_name_I.ref_marker_name.adams_id),($contact_array.adams_id), &
(eval(stoo(ply1)).adams_id),(eval(stoo(ply2)).adams_id)


 force create direct general_force &
 general = $contact_name_j &
 i_marker_name=($polyline_2.parent.poly_rm) &
 j_part_name=($polyline_1.parent) &
 ref_marker_name=($polyline_1.parent.poly_rm) &
 user_function=1,1

 force modify direct gen &
 general = $contact_name_j &
 user=7,($contact_name_J.i_marker_name.adams_id), &
($contact_name_j.ref_marker_name.adams_id),($contact_array.adams_id), &
(eval(stoo(ply2)).adams_id),(eval(stoo(ply1)).adams_id)


variable delete variable = ply1
variable delete variable = ply2
variable delete variable = junk
variable delete variable = junk2
```

<u>New.mnu</u>
….
…..

```
MENU1  TRW
     MENU2  Special_contacts
          BUTTON3  Polyline
                 CMD=interface dialog display dialog=.gui.xcplpl
          BUTTON3  Other
                 CMD=interface dialog display dialog=.gui.atry
MENU1  Help
.....
```

xcplcpl.cmd

```
!
interface dialog_box create  &
   dialog_box_name = .gui.xcplpl  &
   help_text = "Contact Poly Poly"  &
   location = 650.0, 132.0  &
   height = 230.0  &
   width = 421.0  &
   units = pixel  &
   horiz_resizing = attach_left  &
   vert_resizing = attach_top  &
   title = "Contact Poly Poly"  &
   iconifiable = no  &
   start_commands =   &
                    "  int fie set fie=$_self.f_CONTACT_NAME_I
str=(eval(UNIQUE_FULL_NAME(\"General_Force\")))"  &
   execution_commands = "contact_poly_poly  &",  &
                        "   ‘CONTACT_NAME_I = $f_CONTACT_NAME_I‘  &",
&
                        "   ‘CONTACT_NAME_J = $f_CONTACT_NAME_J‘  &",
&
                        "   ‘POLYLINE_1 = $f_POLYLINE_1‘  &",  &
                        "   ‘POLYLINE_2 = $f_POLYLINE_2‘  &",  &
                        "   ‘CONTACT_ARRAY = $f_CONTACT_ARRAY‘",  &
                        "if con=(\"$_2\" != \"\")",  &
                        "  int fie set fie=$_2
str=\"$f_CONTACT_NAME_I\"",  &
                        "  int fie set fie=$_3
str=\"$f_CONTACT_NAME_J\"",  &
                        "end"  &
   decorate = yes  &
   resizable = yes  &
   grab_all_input = no
!
interface label create  &
   label_name = .gui.xcplpl.l_CONTACT_NAME_I  &
   location = 4.0, 4.0  &
   height = 25.0  &
   width = 154.0  &
   units = pixel  &
   horiz_resizing = attach_left  &
   vert_resizing = attach_top  &
   justified = left  &
   text = "Contact Name I"
!
interface field create  &
```

```
   field_name = .gui.xcplpl.f_CONTACT_NAME_I  &
   location = 160.0, 2.0  &
   height = 25.0  &
   width = 257.0  &
   units = pixel  &
   horiz_resizing = expand  &
   vert_resizing = attach_top  &
   scrollable = no  &
   editable = yes  &
   required = yes  &
   execute_cmds_on_exit = no  &
   number_of_values = 1  &
   object_type = new  &
   type_filter = general_force
!
interface label create  &
   label_name = .gui.xcplpl.l_CONTACT_NAME_J  &
   location = 2.0, 78.0  &
   height = 25.0  &
   width = 160.0  &
   units = pixel  &
   horiz_resizing = attach_left  &
   vert_resizing = attach_top  &
   justified = left  &
   text = "Contact Name J"
!
interface field create  &
   field_name = .gui.xcplpl.f_CONTACT_NAME_J  &
   location = 164.0, 78.0  &
   height = 25.0  &
   width = 257.0  &
   units = pixel  &
   horiz_resizing = expand  &
   vert_resizing = attach_top  &
   scrollable = no  &
   editable = yes  &
   required = yes  &
   execute_cmds_on_exit = no  &
   number_of_values = 1  &
   object_type = new  &
   type_filter = general_force
!
interface label create  &
   label_name = .gui.xcplpl.l_POLYLINE_1  &
   location = 4.0, 31.0  &
   height = 25.0  &
   width = 154.0  &
   units = pixel  &
   horiz_resizing = attach_left  &
   vert_resizing = attach_top  &
   justified = left  &
   text = "Polyline 1"
!
interface field create  &
   field_name = .gui.xcplpl.f_POLYLINE_1  &
   location = 160.0, 29.0  &
   height = 25.0  &
   width = 257.0  &
   units = pixel  &
   horiz_resizing = expand  &
   vert_resizing = attach_top  &
   scrollable = no  &
   editable = yes  &
   required = yes  &
   execute_cmds_on_exit = no  &
   number_of_values = 1  &
   object_type = old  &
   type_filter = polyline
!
interface label create  &
   label_name = .gui.xcplpl.l_POLYLINE_2  &
   location = 2.0, 105.0  &
   height = 25.0  &
   width = 160.0  &
   units = pixel  &
   horiz_resizing = attach_left  &
   vert_resizing = attach_top  &
   justified = left  &
   text = "Polyline 2"
!
interface field create  &
   field_name = .gui.xcplpl.f_POLYLINE_2  &
   location = 162.0, 105.0  &
   height = 25.0  &
   width = 257.0  &
   units = pixel  &
   horiz_resizing = expand  &
   vert_resizing = attach_top  &
   scrollable = no  &
   editable = yes  &
   required = yes  &
   execute_cmds_on_exit = no  &
   number_of_values = 1  &
   object_type = old  &
   type_filter = polyline
!
interface label create  &
   label_name = .gui.xcplpl.l_CONTACT_ARRAY  &
   location = 2.0, 164.0  &
   height = 25.0  &
   width = 160.0  &
   units = pixel  &
   horiz_resizing = attach_left  &
   vert_resizing = attach_top  &
   justified = left  &
   text = "Contact Array"
!
interface field create  &
   field_name = .gui.xcplpl.f_CONTACT_ARRAY  &
   location = 162.0, 164.0  &
   height = 25.0  &
   width = 257.0  &
   units = pixel  &
   horiz_resizing = expand  &
   vert_resizing = attach_top  &
   scrollable = no  &
   editable = yes  &
   required = yes  &
   execute_cmds_on_exit = no  &
   number_of_values = 1  &
   object_type = old  &
   type_filter = adams_array
!
interface push_button create  &
   push_button_name = .gui.xcplpl.OK  &
   location = 155.0, 197.0  &
   height = 25.0  &
   width = 76.0  &
   units = pixel  &
   horiz_resizing = attach_right  &
   vert_resizing = attach_bottom  &
   label = "OK"  &
   commands = "interface dialog execute dialog=$_parent undisplay=yes"
&
   default = true
!
interface push_button create  &
   push_button_name = .gui.xcplpl.Apply  &
   location = 245.0, 197.0  &
   height = 25.0  &
   width = 76.0  &
   units = pixel  &
   horiz_resizing = attach_right  &
   vert_resizing = attach_bottom  &
   label = "Apply"  &
   commands = "interface dialog execute dialog=$_parent undisplay=no"
!
interface push_button create  &
   push_button_name = .gui.xcplpl.Cancel  &
   location = 335.0, 197.0  &
   height = 25.0  &
   width = 76.0  &
   units = pixel  &
   horiz_resizing = attach_right  &
   vert_resizing = attach_bottom  &
   label = "Cancel"  &
   commands = "interface dialog undisplay dialog=$_parent"
!
interface separator create  &
   separator_name = .gui.xcplpl.sep_1  &
   location = 4.0, 74.0  &
   height = 2.0  &
   width = 413.0  &
   units = pixel  &
   horiz_resizing = attach_left  &
   vert_resizing = attach_top
```