Altair

HyperWorks

Altair HyperMesh 2019 Tutorials

HM-8070: Create Spline Surfaces on Tria Elements

HM-8070: Create Spline Surfaces on Tria Elements

In this tutorial you will create a Tcl script that creates spline surfaces from the nodes of selected tria elements.

Tools

The Tcl commands `if`, `foreach`, and `incr` will be used to add logic to the script. The command `hm_getentityvalue` is used to extract information from HyperMesh entities, based on data names.

Data names are generic references to the information that physically define an entity in the HyperMesh environment. An example of this is the x-, y-, and z-coordinates that define a node location in three-dimensional space. The available data names for each can be found in the HyperMesh Reference Guide Data Names topic.

Data names are accessed using the `hm_getentityvalue` command. This command uses the data names available for an entity to return the particular value of interest. The command will return a value that is either a string or a numeric value, depending on the command syntax and the value stored in that particular data name field. The basic syntax of the command is:

`hm_getentityvalue` *`entity_type id data_name flag`*

where *`entity_type`* is the requested entity type (elements, loads, nodes, etc…), *`id`* is the entity ID, the *`data_name`* is the data field name of interest, and *`flag`* is either 0 or 1 depending on whether the command should return a numeric value (0) or a string (1).

To retrieve the x-component of a force with ID 12, the following command can be used:

```
set force_x [hm_getentityvalue loads 12 "comp1" 0]
```

Note that to assign the value from the command to a variable, the command is placed within square brackets.

Exercise

Create a Tcl script that creates spline surfaces from the nodes of selected tria elements. This requires that the script read data from the element entities. To create the spline surfaces, retrieve the 3-node IDs of the tria elements.

1. Define the process.
2. Determine the data names to use to extract the element type and node IDs.
3. Create the Tcl script and add logic as necessary.
4. Test the script.

Step 1: Define the process.

The script should automate the following process:

- Prompt the user to select a number of tria elements to create spline surfaces from.
- Make sure the user has selected one or more elements.

- If a selected element is not a tria, skip that element.
- Extract the node IDs of each element.
- Create the spline surface from the nodes.
- Report on the number of spline surfaces created.

Step 2: Determine the data names to use to extract the element type and node IDs.

The following table lists several relevant data names for tria elements:

config	The number, "103"
node1	first node (node pointer)
node2	second node (node pointer)
node3	third node (node pointer)

Steps 3-14: Create the Tcl script and add logic as necessary.

A Tcl script to perform this function might be similar to the following:

Step 3: Open a text file and save the file as HM8070.tcl.

Step 4: Allow the user to select the desired elements and then add those loads to a list

The `*createmarkpanel` command is used to allow the user to graphically select the elements from the HyperMesh interface and add them to the mark. The command below adds the elements to mark 1. Once the elements have been added to mark 1, the element ids are assigned to a list called `elems_list`, using the TCL command `set`. Add the following 2 lines to the file `HM8070.tcl`:

```
*createmarkpanel elems 1 "Select tria elements to create surfaces";
set elems_list [hm_getmark elems 1];
```

Step 5: Begin an if loop which checks to see if the variable `elems_list` has values. If it does, proceed with the macro.

Before continuing with the macro, we should check to make sure that the variable `elems_list` has values in it. This is done by using an `if` loop. In the `if` loop below, we are checking that the variable `elems_list` is not empty. Add the following line to the TCL file to initialize the `if` loop:

```
if {$elems_list != ""} {
```

Step 6: Initialize a variable which counts the number of times the `foreach` loop is entered.

The variable `success_count` is initialized and set to 0. This variable is used to count the number of times the `foreach` loop (defined in Step 7) is entered. We will use this variable at the end of the script. Add the following line to the TCL script:

```
set success_count 0;
```

Step 7: Use a `foreach` loop to iterate through each element in the list `elems_list` and then set a variable `config` which stores the element configuration. This is extracted using the `hm_getentityvalue` command and the appropriate data name.

Using a `foreach` loop, each element in the list `elems_list` will be iterated through. Within the `foreach` loop, each load is referenced by `elem_id` and then the variable `config` is defined. This variable is set to the result of the `hm_getentityvalue` which uses the element data name `config` to report the configuration of the element. A tria element will have an element configuration of 103 while a quad element will have a configuration of 104. Add the following 2 lines to the TCL file:

```
foreach elem_id $elems_list {
    set config [hm_getentityvalue elems $elem_id "config" 0];
```

Step 8: Begin an `if` loop which checks to see if the variable `config` has a value of 103. If it does, proceed with the macro.

Using an `if` loop, the variable `config` is checked to see if it doesn't have a value of 103. A value of 103 means that the element configuration is a tria element. If the value is not equal to 103, the `continue` statement is used to move outside of the foreach loop. If the value is the `config` variable is 103, then the macro is continued. Add the following lines to the TCL script:

```
    if {$config != 103} {
        continue;
    }
```

Step 9: Set 3 variables which contain the node id of each of the nodes used to define the tria element.

Three variables are defined (`node1`, `node2`, and `node3`) which represent the 3 nodes that define the tria element. These 3 nodes will be used to create the spline surface. Using the `hm_getentityvalue` command and the element data names node1, node2, and node3 along with the pointer id, the node id is retrieved and assigned to the appropriate variable. Add the following 3 lines to the TCL script:

```
set node1 [hm_getentityvalue elems $elem_id "node1.id" 0];
set node2 [hm_getentityvalue elems $elem_id "node2.id" 0];
set node3 [hm_getentityvalue elems $elem_id "node3.id" 0];
```

Step 10: Set the appropriate mode to create the surface.

Using the `*surfacemode` command, the surface mode can be set according to the following:

1 – mesh keep surface

2 – mesh delete surface

3 – mesh without surface

4 – surface only

In this example, we only want to create a surface, so mode 4 is used. Add the following line to the TCL script:

```
*surfacemode 4;
```

Step 11: Create a node mark which contains the 3 nodes defined in Step 9 and then use the `*splinesurface` command to create a spline surface using the nodes in the mark.

Using the `*createmark`, mark 1 for nodes is created and it contains the 3 nodes defined in the variables `node1`, `node2`, and `node3`.

```
*createmark nodes 1 $node1 $node2 $node3;
*splinesurface nodes 1 0 1 1;
```

Step 12: Increase the variable `success_count` which is used to count the number of times the `foreach` loop is entered. Then, close the `foreach` loop.

Using the `incr` command, the variable `success_count` is increased. Following this command, a } is used to close the `foreach` loop. Add the following 2 lines to the TCL script:

```
        incr success_count;
    }
```

Step 13: Clear the node and element marks, and then use the `hm_usermessage` command to report the number of spline surfaces created.

Using the command `*clearmark`, mark 1 for the nodes and elements is cleared. Following these commands, the `hm_usermessage` command is used to report the number of spline surfaces created. The variable `success_count` is used to do this. Because this variable was increased each time the `foreach` loop was entered and the element configuration was 103, this variable kept a count of the number of spline surfaces that were created. Add the following 3 lines to the TCL script:

```
    *clearmark nodes 1;
    *clearmark elems 1;
    hm_usermessage "$success_count splines created."
```

Step 14: Add an `else` statement which compliments the `if` statement which checked to see if the `elems_list` variable was empty. If it is empty, the `else` statement is executed.

The `else` statement compliments the `if` statement defined in Step 5 which checks to see if the `elems_list` variable is empty. If it is empty the `else` statement is executed. Inside the `else` statement, the `hm_errormessage` command is used to report to the user that no elements were selected. Following the `hm_errormessage` command, the `if` statement is closed using a }. Add the following 3 lines to the TCL script file:

```
} else {
    hm_errormessage "No elements selected";
}
```

Step 15: Test the script.

1. From the menu bar, select ***File > Open > Model*** and then load the file, `spline-tcl.hm`.
2. From the menu bar, select ***View > Command Window*** display the Command Window at the bottom of the screen.
3. Click and drag to open the Command Window from the bottom edge of the screen.
4. Use the source command to execute the script. For example:

   ```
   source HM8070.tcl
   ```

It is often necessary to debug Tcl scripts using the Command Window. This allows you to run the Tcl script and easily review error messages, as well as print out debug information. Additional details can be found in the *Creating Tcl Scripts and Running Tcl Scripts* sections.

5. Select a few of the tria elements and observe the spline surfaces that are created.

There are several important things to notice.

- Only first order tria elements are considered. It is possible to add `if/elseif` logic to support other element configurations.
- The data names for the nodes associated with an element are pointers. A pointer is used to directly access another data name. This means they "point" to the data names available for nodes. In order to retrieve any data from a pointer, the data name requested for the particular pointer must also be supplied. The additional data names are separated by a period or dot (.).
- The `*entityhighlighting` and `hm_commandfilestate` commands are used to speed up the execution of the script. The `*entityhighlighting` command disables highlighting entities when the `*createmark` command is used. The `hm_commandfilestate` command controls if commands are written out to the command file. It is always important to "reset" these commands after a script is complete or before exiting due to an error.