

Verification of a Multiphysics Toolkit against the Magnetized Target Fusion Concept

*Scott Thomas**

*International Space Systems, Inc.
555 Wynn Drive, NW, Suite 440
Huntsville, Alabama 35816*

Eric Perrell†

Caroline Liron

Robert Chiroux†

Jason Cassibry†

*University of Alabama in Huntsville
Propulsion Research Center
Tech Hall, S234
Huntsville, Alabama 35899*

Robert B. Adams†

*National Aeronautics and Space Administration
George C. Marshall Space Flight Center
NP10/Advanced Concepts Office
MSFC, AL 35812*

ABSTRACT

In the spring of 2004 the Advanced Concepts team at MSFC embarked on an ambitious project to develop a suite of modeling routines that would interact with one another. The tools would each numerically model a portion of any advanced propulsion system. The tools were divided by physics categories, hence the name multiphysics toolset. Currently most of the anticipated modeling tools have been created and integrated. Results are given in this paper for both a quarter nozzle with chemically reacting flow and the interaction of two plasma jets representative of a Magnetized Target Fusion device. The results have not been calibrated against real data as of yet, but this paper demonstrates the current capability of the multiphysics tool and planned future enhancements

INTRODUCTION

In the spring of 2002 the Advanced Concepts department at MSFC embarked on an ambitious project to develop an integrated set of analytical tools for conceptual design of spacecraft. This project included development of a collaborative engineering environment to integrate the efforts of teams of engineers and scientists designing these spacecraft. It quickly became apparent that it would be prohibitive in

terms of time and cost to develop design routines for all of the myriad advanced propulsion concepts suitable to space exploration. When the Advanced Concepts team considered the list of propulsion concepts, from liquid propellant propulsion, to solar sails and from nuclear thermal to fusion propulsion they discovered that a common set of physics routines should be able to model the physical processes of most of the propulsion concepts. The sections below define each of the component models and integration of the components in the CEE to yield a multiphysics model of the propulsion system in question.

*Systems Engineer, member AIAA.

Senior Space Systems Engineer.

†Systems Engineer, Advanced Propulsion Technologies, senior member AIAA

COMPONENT MODELS

Integration

The main routines to integrate into the logic flow were a fluids routine, a solid structural routine, and a solid thermal conduction routine. The routines used to

fill these needs are detailed below. The Preliminary Analysis of Revolutionary Exploration Concepts (PARSEC) collaborative engineering environment (CEE) was used to apply the logic flow to the quarter-nozzle analysis. The input and output described in this section are restricted to those necessary for the integration of the component models. Input and output specific to a single component will be detailed in that component's section of this report.

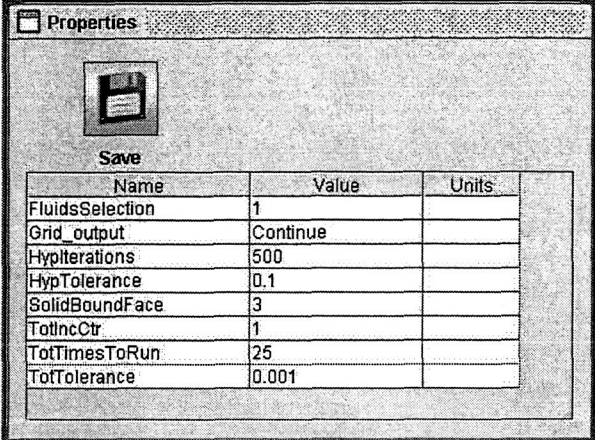
First, a fluids code was selected by the user, choosing between the continuum fluid mechanics code, HYP, or the particle-in-cell (PIC) code. In the quarter-nozzle analysis, the HYP code was used. The HYP code read in the fluids grid and wall surface temperature as integration input. The code then ran until a user-specified tolerance was reached or a maximum number of component iterations was completed. This was done to allow the fluids code to approach steady-state output. The HYP code integration output was the heat flux at the wall and the pressure at the wall.

Second, the solid thermal conduction code read in the solid grid and the heat flux at the wall from the HYP code and calculated the resulting nodal temperatures. The thermal conduction code output was a new wall surface temperature distribution.

Next, the structures code read in the solid grid and the pressure at the wall. A resulting solid deformation was calculated, and output as a new solid grid.

Finally, the new solid grid was read in to a grid resizing component. This component calculated the deformation of the new solid grid with respect to the old solid grid. This displacement was then equally distributed between the nodes of the old fluid grid to create a new fluid grid the aligned with the deformed solid. This simple algorithm applied either an expansion or contraction of the fluid based on the solid deformation.

Integrating these tools into the multiphysics toolkit in the PARSEC CEE are global variables and processes. The global variables are available to import into any of the processes for logic control, decision making, exit criteria, etc. The global variables for the multiphysics toolkit are shown in **Figure 1**.



Name	Value	Units
FluidsSelection	1	
Grid_output	Continue	
HypIterations	500	
HypTolerance	0.1	
SolidBoundFace	3	
TotIncCtr	1	
TotTimesToRun	25	
TotTolerance	0.001	

Figure 1 Global Variables

The use of these global variables will be explained in detail as each process in the logic flow is described.

The overall toolkit in the PARSEC CEE is shown in **Figure 2**.

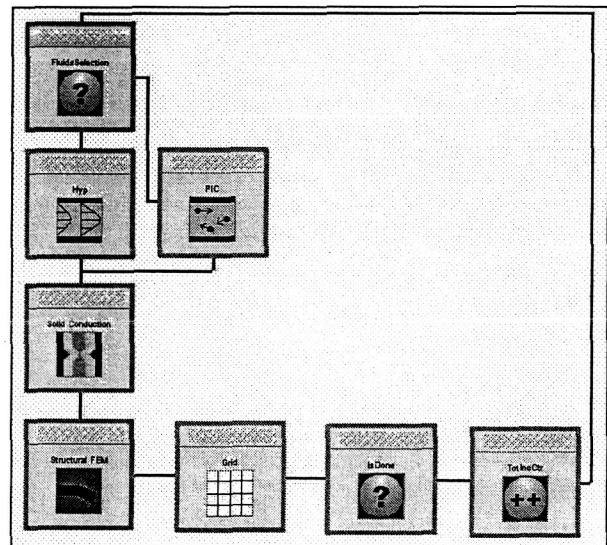


Figure 2 Multiphysics Toolkit

Each box in the toolkit represents a process available to perform a discipline analysis. All the processes currently available in this multiphysics toolkit are also shown in the process window as seen in **Figure 3**.

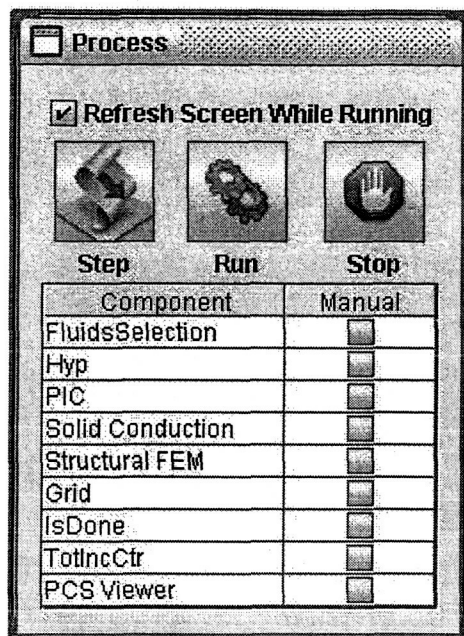


Figure 3 Process Window

Any process may be selected and launched separately using the “Step” button. Launching an entire global loop simply requires selecting the process with which to begin and pressing the “Run” button.

The proper logic flow for this layout begins in the top left corner of Figure 2 and progresses in a counter-clockwise fashion. The FluidsSelection process (Figure 4) uses the *FluidsSelection* global variable in a comparator. The comparator lets the user set an upper and lower bound. It then compares the input to those bounds and outputs either *LessThan*, *Between*, or *GreaterThan* as an exit code. This exit code then allows the user to direct the logic flow in the proper direction.

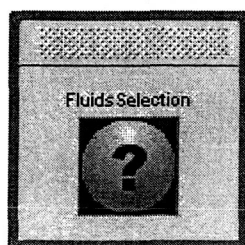


Figure 4 FluidsSelection Process

For the quarter-nozzle analysis, the *FluidsSelection* value of 1 causes control to pass to HYP (Figure 5)

instead of using a value of 2, which would have passed control to PIC (Figure 6) instead.

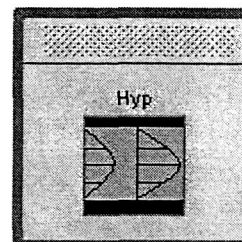


Figure 5 HYP Process

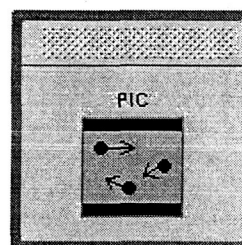


Figure 6 PIC Process

HYP uses the *HypIterations* global variable as input to determine how many component iterations to run per global iteration. *HypTolerance* is a placeholder for future use that will allow HYP to relinquish control when the maximum residual reaches the desired threshold. HYP also uses *SolidBoundFace* as input to determine which boundary will communicate with the solid processes.

Both the HYP and PIC process, once completed, pass control to the Solid Conduction process (Figure 7). This process also uses *SolidBoundFace* as input to determine communication requirements. It then passes control to the structures process (Figure 8).

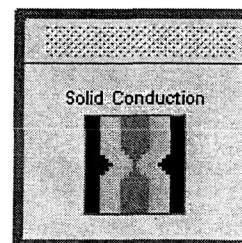


Figure 7 Solid Conduction Process

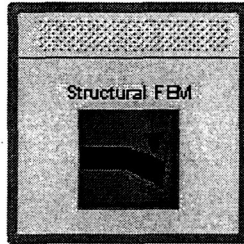


Figure 8 Structural FEM Process

The structures process also uses *SolidBoundFace* as input to determine communication requirements. It then passes control to the grid process (Figure 9). The grid process uses *TotTolerance* as input to determine if the solid deformation is below the desired threshold. The grid process uses *Grid_output* as output to alert the user to the status of the grid. If the grid has converged, the output will be "Converged" and control will pass back to the screen, ending the global iterations. Otherwise, the output will be "Continue" and control will pass to another comparator, *IsDone* (Figure 10).

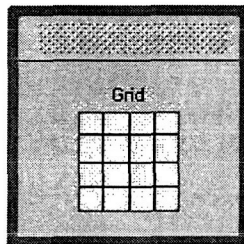


Figure 9 Grid Process

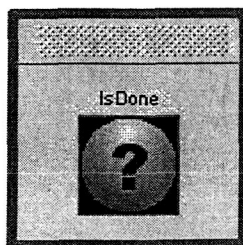


Figure 10 IsDone Process

The *IsDone* process uses *TotIncCtr* and *TotTimesToRun* as input. If the value of *TotIncCtr* is equal to or greater than *TotTimesToRun*, then control passes back to the screen. Otherwise, control is passed to the *TotIncCtr* process (Figure 11).

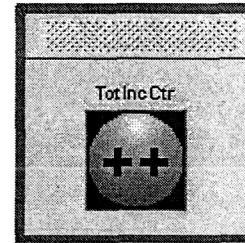


Figure 11 TotIncCtr Process

This process takes the *TotIncCtr* global variable and increments its value by one. Control is then passed back to the *FluidsSelection* process, and the next global iteration has begun.

This global iteration logic flow will continue until one of the exit criteria are met and control is not passed to the next process. The user can interrupt this flow by using the "Stop" button. The current process will be terminated and control will be passed back to the screen.

All the above processes output data in the simple legacy Visualization Toolkit (VTK) file format¹. The PARSEC Construction Set (PCS) was written using VTK routines. The PCS Viewer process (Figure 12), while currently not in the logic flow loop, allows easy viewing of the results.

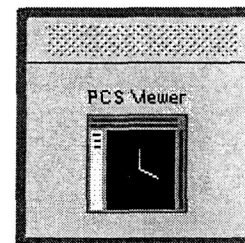


Figure 12 PCS Viewer Process

Future modification will allow this process to be placed in the logic loop, thus allowing real-time viewing of the results for each global iteration.

Continuum Fluid Mechanics (HYP)

Rick

Thermal Conduction

The solid thermal conduction code is based on a structured finite difference scheme that is first order accurate in time and second order accurate in space. The formulation is based on natural coordinates in a computational domain. The node coordinates may then be transformed to the physical space using a proper transformation technique. The formulation allows for transient, anisotropic analysis through time-step and spatial thermal conductivity input. A heat flux can be applied at each surface node, or an equal heat flux can be applied to any surface. At present, the density and specific heat are a single input, not allowing for either to be a function of temperature.

After reading the inputs, the spatial thermal diffusivity² is calculated according to Equation 1):

$$\alpha_{xi} = \frac{k_{xi}}{\rho c_p} \quad (1)$$

Then the spatial Fourier number is calculated according to Equation 2):

$$Fo_{xi} = \frac{\alpha_{xi} d_t}{d_{xi}^2} \quad (2)$$

The node temperature at the current time step is calculated explicitly from the surrounding node temperatures at the previous time step. The formulae³ for calculation differs for the interior nodes (Equation (3))), surface nodes (Equation 4)), edge nodes (Equation 5)), and corner nodes (Equation 6)).

$$T_2(i, j, k) = Fo_{xi}(T_1(i+1, j, k) + T_1(i-1, j, k)) + Fo_{eta}(T_1(i, j+1, k) + T_1(i, j-1, k)) + Fo_{zeta}(T_1(i, j, k+1) + T_1(i, j, k-1)) + (1 - 2(Fo_{xi} + Fo_{eta} + Fo_{zeta})) * T_1(i, j, k) \quad (3)$$

$$T_2(i, j, k) = 2Fo_{xi}\left(q_{xi}(i, j, k)\frac{d_{xi}}{k_{xi}} + T_1(i+1, j, k)\right) + Fo_{eta}(T_1(i, j+1, k) + T_1(i, j-1, k)) + Fo_{zeta}(T_1(i, j, k+1) + T_1(i, j, k-1)) + (1 - 2(Fo_{xi} + Fo_{eta} + Fo_{zeta})) * T_1(i, j, k) \quad (4)$$

$$T_2(i, j, k) = 2Fo_{xi}\left(q_{xi}(i, j, k)\frac{d_{xi}}{k_{xi}} + T_1(i+1, j, k)\right) + \quad (5)$$

$$2Fo_{eta}\left(q_{eta}(i, j, k)\frac{d_{eta}}{k_{eta}} + T_1(i, j+1, k)\right) + Fo_{zeta}(T_1(i, j, k+1) + T_1(i, j, k-1)) + (1 - 2(Fo_{xi} + Fo_{eta} + Fo_{zeta})) * T_1(i, j, k)$$

$$T_2(i, j, k) = 2Fo_{xi}\left(q_{xi}(i, j, k)\frac{d_{xi}}{k_{xi}} + T_1(i+1, j, k)\right) + \quad (6)$$

$$2Fo_{eta}\left(q_{eta}(i, j, k)\frac{d_{eta}}{k_{eta}} + T_1(i, j+1, k)\right) + 2Fo_{zeta}\left(q_{zeta}(i, j, k)\frac{d_{zeta}}{k_{zeta}} + T_1(i, j, k+1)\right) + (1 - 2(Fo_{xi} + Fo_{eta} + Fo_{zeta})) * T_1(i, j, k)$$

The stability criteria can easily be seen in the last term of Equation 3). For unconditional stability, this last term must be positive, which requires the sum of the directional Fourier numbers be less than 1/2. Also note that Equation 4) is for the lower xi-surface, and that 5 similar equations are used for the remaining surfaces. Similarly, Equation 5) is for the edge that shares the lower xi-surface and the lower eta-surface, with 11 similar equations for the remaining edges. Finally, Equation 6) is for the corner that shares the lower xi-, eta-, and zeta-surfaces, with 7 similar equations for the remaining corners.

These equations are iterated through according to the input time-step, and the resulting nodal temperatures are written to the output file.

Structures

The initial calculation within the structures code is to determine the transformation matrix for a triangular element oriented arbitrarily in space. This technique is drawn heavily from Zienkiewicz¹. The following development assumes the three nodes are oriented counterclockwise, positive outward, within the element and located based on Cartesian coordinates globally common to all elements. The resulting transformation matrix will be used to orient the element into a common plane with all other transformed elements to ensure accurate compilation of the global stiffness matrix.

The first step is to calculate the component distances between the first and second nodes. The vector's scalar length between nodes one and two is calculated. The components are then normalized. This produces the direction cosines for the "local" coordinate system's "x"

axis and defines that axis along the node 1-2 side of the element.

$$X_{21} = X_2 - X_1 \quad 2.1.1$$

$$Y_{21} = Y_2 - Y_1 \quad 2.1.2$$

$$Z_{21} = Z_2 - Z_1 \quad 2.1.3$$

$$L_{21} = \sqrt{X_{21}^2 + Y_{21}^2 + Z_{21}^2} \quad 2.1.4$$

$$L_x = \frac{X_{21}}{L_{21}} \quad 2.1.5$$

$$M_x = \frac{Y_{21}}{L_{21}} \quad 2.1.6$$

$$N_x = \frac{Z_{21}}{L_{21}} \quad 2.1.7$$

Having determined the "x-prime" axis direction cosines a similar process can be applied to the node 1-3 side of the element. It is important to note that this is not typically the "local" coordinate system's "y" axis, but it does define another vector.

$$X_{31} = X_3 - X_1 \quad 2.1.8$$

$$Y_{31} = Y_3 - Y_1 \quad 2.1.9$$

$$Z_{31} = Z_3 - Z_1 \quad 2.1.10$$

With two vectors multiplication of the first into the second produces a third vector, normal to both. This resulting vector does constitute the "local" coordinate "Z" axis and along with it the "Z" axis direction cosines can be determined.

$$XZ = Y_{21} \cdot Z_{31} - Y_{31} \cdot Z_{21} \quad 2.1.11$$

$$YZ = Z_{21} \cdot X_{31} - Z_{31} \cdot X_{21} \quad 2.1.12$$

$$ZZ = X_{21} \cdot Y_{31} - X_{31} \cdot Y_{21} \quad 2.1.13$$

$$L_{31} = \sqrt{XZ^2 + YZ^2 + ZZ^2} \quad 2.1.14$$

$$L_z = \frac{XZ}{L_{31}} \quad 2.1.15$$

$$M_z = \frac{YZ}{L_{31}} \quad 2.1.16$$

$$N_z = \frac{ZZ}{L_{31}} \quad 2.1.17$$

At this point the "local" coordinate "x" and "z" axes have been defined, along with the normalized components that define their direction cosines. Multiplication of the "z" vector to the "x" vector, or more specifically the direction cosines of "z" into "x", the "Local" coordinate "y" axis is defined along with its direction cosines.

$$L_y = M_z \cdot N_x - N_z \cdot M_x \quad 2.1.18$$

$$M_y = N_z \cdot L_x - L_z \cdot N_x \quad 2.1.19$$

$$N_y = L_z \cdot M_x - M_z \cdot L_x \quad 2.1.20$$

This process is repeated for each element in succession and the direction cosine matrix, and its transpose, for each element is used in various locations throughout the "MPTFEM" code.

$$[DCM] = \begin{bmatrix} L_x & M_x & N_x \\ L_y & M_y & N_y \\ L_z & M_z & N_z \end{bmatrix}$$

2.1.21

$$[DCM]^T = \begin{bmatrix} L_x & L_y & L_z \\ M_x & M_y & M_z \\ N_x & N_y & N_z \end{bmatrix}$$

2.1.22

After assembling the transformation matrix for each element the coordinates of each element's nodes will need to be transformed to that element's "local" coordinate system. This process first translates the element so as to define node one at the local origin. The transformation matrix can then be used to rotate the element such that its node 1-2 edge define its local "x" axis and the element as a whole lies in the "local" coordinate "xy" plane.

Translation of the element is accomplished by setting node one at the origin and subtracting its coordinates from the other two nodes. The translation of node three is shown as an example.

$$N3_x = X_3 - X_1$$

2.2.1

$$N3_y = Y_3 - Y_1$$

2.2.2

$$N3_z = Z_3 - Z_1$$

2.2.3

Assembling the node translated coordinates in matrix form and multiplying it into the transpose of the transformation matrix produce a matrix of "local" nodal coordinates.

$$\begin{bmatrix} N1'_x & N1'_y & N1'_z \\ N2'_x & N2'_y & N2'_z \\ N3'_x & N3'_y & N3'_z \end{bmatrix} = \begin{bmatrix} N1_x & N1_y & N1_z \\ N2_x & N2_y & N2_z \\ N3_x & N3_y & N3_z \end{bmatrix} \times \begin{bmatrix} L_x & L_y & L_z \\ M_x & M_y & M_z \\ N_x & N_y & N_z \end{bmatrix}$$

Assuming the element has been transformed to the local coordinate system determination of the element's area is fairly straightforward. The algorithm within the MPTFEM code was written for a triangular element in an arbitrary orientation and has been left intact for future applications. For an arbitrary orientation if the "x" coordinates of nodes one and two are identical a "divide by zero" error would develop during the slope calculation. In this case the program automatically

swaps node three and one for the area calculation and restores them after its completion.

$$SLOPE = \frac{N1_y - N2_y}{N1_x - N2_x}$$

2.3.1

$$HEIGHT = \frac{-SLOPE \cdot N3_x + N3_y + SLOPE \cdot N2_x}{\sqrt{1 + SLOPE^2}}$$

2.3.2

$$BASE = \sqrt{(N1_x - N2_x)^2 + (N1_y - N2_y)^2}$$

2.3.3

$$AREA = \frac{1}{2} \cdot BASE \cdot HEIGHT$$

2.3.4

The elasticity matrix for a 3D element can be easily drawn from reference material.

$$[ESM] = \frac{E \cdot AREA \cdot T}{(1 - \mu^2)} \begin{bmatrix} 1 & \mu & \mu & 0 & 0 & 0 \\ \mu & 1 & \mu & 0 & 0 & 0 \\ \mu & \mu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-\mu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-\mu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-\mu}{2} \end{bmatrix}$$

2.4.1

Where: E = Modulus of elasticity

AREA = Element Area

T = Element Thickness

μ = Poisson's Ratio

The gradient matrix for a triangular plate element, arbitrarily oriented in three-dimensional space, is essentially a two-dimensional matrix that is expanded to accommodate a three dimensional structure. Although the gradient matrix is calculated based on an element's "local" coordinates, and thus out of plane terms are zero, the expanded gradient matrix has been left intact for future development. The purpose of the gradient matrix is to relate nodes displacements to strains. The gradient matrix combined with the elasticity matrix generates an element's stiffness matrix.

$$[B] = \frac{1}{2 \cdot \text{AREA}} \rightarrow$$

2.5.1

$$\rightarrow \begin{bmatrix} N2'_y - N3'_y & 0 & 0 & N3'_y - N1'_y & 0 \\ 0 & N3'_x - N2'_x & 0 & 0 & N1'_x - N3'_x \\ 0 & 0 & 0 & 0 & 0 \\ N3'_x - N2'_x & N2'_y - N3'_y & 0 & N1'_x - N3'_x & N3'_y - N1'_y \\ 0 & 0 & N2'_y - N3'_y & 0 & 0 \\ 0 & 0 & N3'_x - N2'_x & 0 & 0 \end{bmatrix}$$

Once the element's elasticity matrix and gradient matrix are determined calculation of the element stiffness matrix is straightforward. This results in a 12x12 matrix.

$$[EStM'] = [B]^T \cdot [ESM] \cdot [B]$$

2.6.1

Due to the gradient matrix having been formed from "local", transformed nodal coordinates the resulting element stiffness matrix was defined in the "local" coordinate system as well. Use of the previously determined transformation matrix for the element allows transformation of the element stiffness matrix back into global coordinates. It is important to note that the element stiffness matrix is a 12x12 matrix while the transformation matrix is a 3x3 matrix. This can be overcome by repeating the transformation matrix four times.

$$[T]_{12 \times 12} = \begin{bmatrix} L_x & M_x & N_x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ L_y & M_y & N_y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ L_z & M_z & N_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & L_x & M_x & N_x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & L_y & M_y & N_y & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & L_z & M_z & N_z & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & L_x & M_x & N_x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & L_y & M_y & N_y & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & L_z & M_z & N_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L_x & M_x & N_x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L_y & M_y & N_y \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L_z & M_z & N_z \end{bmatrix}$$

2.6.1

After expanding the transformation matrix the stiffness matrix can be transformed easily.

$$[EStM] = [T]_{12 \times 12}^T \cdot [EStM'] \cdot [T]_{12 \times 12}$$

2.6.2

In order to assemble the global stiffness matrix from the element stiffness matrices it is necessary to enter the stiffness terms in the proper locations. For this type of element each node has three degrees of freedom. With this in mind a model consisting of 30 nodes would have 90 degrees of freedom. The purpose of the global stiffness matrix is to define the relationships between each degree of freedom. For instance, the global stiffness matrix would have a term to relate the "x" motion of node one to the "z" motion of node seven, and so forth. More specifically a triangular element is defined by nodes 1, 7 and 9, in counterclockwise order, it would be necessary to relate degrees of freedom 1, 2, 3, 19, 20, 21, 25, 26 and 27 to each other. The interaction between these nine degrees of freedom generate 81 stiffness relationship terms to be added to the global stiffness matrix in the proper locations. Since combinations of these nodes appear in other elements it would not be unusual for an entry to be added to a previously entered relationship in the global stiffness matrix. The example equation shown demonstrates the entry to the global stiffness matrix for the interaction noted above, the "x" motion of node one to the "z" motion of node seven.

$$BIGK[1,21] = BIGK[1,21] + ESStM[1,9]$$

2.8.1

After completing an element's entries to the global stiffness matrix the next element and its associated nodes are read and the above logic is repeated until all entries to the global stiffness matrix are complete.

After assembly of the global stiffness matrix it must be modified to account for any applied constraints. Constraints to the stiffness matrix would simulate locations where the model is fixed. Constraints are applied to degrees of freedom as opposed to nodes as not all degrees of freedom at a node are constrained. For instance a fixed node would have its x, y and z motions constrained while a sliding node might have only its z motion constrained. Each degree of freedom that is constrained must generate a modification to the global stiffness matrix that is covered in two parts. As an example we will assume the "z" motion of node five is constrained.

First the diagonal within the global stiffness matrix is set to unity.

$$BIGK[15,15] = 1.0$$

2.9.1

Next the column and row terms including this degree of freedom are zeroed out. This degree of

freedom can have no affect on other degrees of freedom because it is locked.

$$BIGK[15, (ALL)] = 0.0$$

2.9.2

And:

$$BIGK[(ALL), 15] = 0.0$$

2.9.3

A simple but important step is to form the loads array. This array will be needed along with the assembled and modified global stiffness matrix to produce the primary result, the deflection matrix. The input file provides a series of entries to define any load to be applied to each degree of freedom. For illustration we will assume that a load of 250 lbf is applied, in the negative "z" direction, on node twenty.

$$R[60] = -250.0$$

2.10.1

The solution that is produced is a direct result of the loads applied the structure, which is represented by its modified stiffness matrix. In actuality the equation that is used to determine the solution is normally stated with the loads array as the solution of the multiplication of the global stiffness matrix to the displacement array.

$$[GlobalStiffnessMatrix] \times [DisplacementArray] = [LoadsArray]$$

2.11.1

Which in terms of this program would be stated:

$$[BIGK] \times [DEF] = [R]$$

2.11.2

In order to determine the displacement matrix "[DEF]" the stiffness matrix must be inverted.

$$[DEF] = [BIGK]^{-1} \times [R]$$

2.11.3

It should be noted that inverting the stiffness matrix for a large matrix is a problem requiring so much bookkeeping that a computer algorithm is the only practical method for doing so. Fortunately many subroutines have already been written to perform this operation. In this program such a routine is drawn from McKinley². The size of the stiffness matrix in this program is 900 by 900 degrees of freedom. Compared

to larger commercially available finite element programs this would be considered small.

The final step of the program is to determine the state of stress in each element. The component matrices necessary to calculate the state of stress have already been determined. The matrices include the element elasticity matrix, the gradient matrix and the displacement matrix. The element stress is calculated in the local element system therefore the elasticity matrix is transformed into local coordinates, the gradient matrix is defined in local coordinates and the displacement matrix, recently solved, is transformed into local coordinates. The result produces the stress state in the local element coordinate system.

$$[\sigma'] = [ESM'] \times [B] \times [DEF']$$

2.12.1

Particle in Cell
Jason

$$c = \alpha \Delta s. \quad (1)$$

RESULTS

Quarter-Nozzle, Liquid Propellant Engine

The quarter-nozzle, liquid propellant engine analysis was performed using the PARSEC CEE. The geometry for the quarter-nozzle analysis can be seen in **Figure 13**. The wall geometry can be seen in **Figure 14**. The fluid geometry can be seen in **Figure 15**.



Figure 13 Quarter-nozzle Geometry

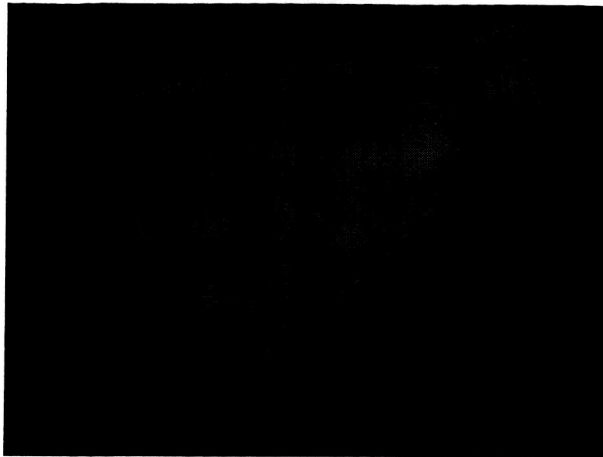


Figure 14 Quarter-nozzle Wall Geometry



Figure 15 Quarter-nozzle Fluid Geometry

Note that the fluid geometry has been set up with a central cathode for future use with electromagnetic effects.

The analysis run in the PARSEC CEE was comprised of 32 global iterations, with 250 iterations of HYP per global iteration. The solid wall properties used were for typical steel due to the author's familiarity with its response. The wall modulus of elasticity was 205 GPa, the Poisson's ratio was 0.30, the density was 7870 kg/m³, the specific heat was 599 J/(kg K), the thermal conductivity in all three axes was 51.9 W/(m K), and the initial wall temperature was 300K. The input fluid temperature was 600K, the input fluid pressure was 1013250 Pa (10 atm), and the input fluid velocity was 83 m/s in the axial direction.

The resulting temperature for the quarter-nozzle can be seen in **Figure 16**. Note that due to the relatively high thermal conductivity of the wall and running the analysis to a quasi-steady state solution, it appears as nearly equal temperature in **Figure 16**. The temperature for the wall alone can be seen in **Figure 17**, which more clearly displays the temperature contour. The temperature for the fluid can be seen in **Figure 18**, which clearly shows the reduction in temperature through the nozzle due to expansion.

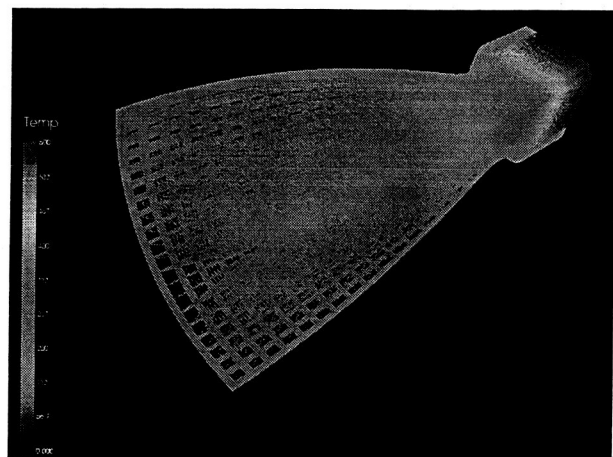


Figure 16 Quarter-nozzle Temperature

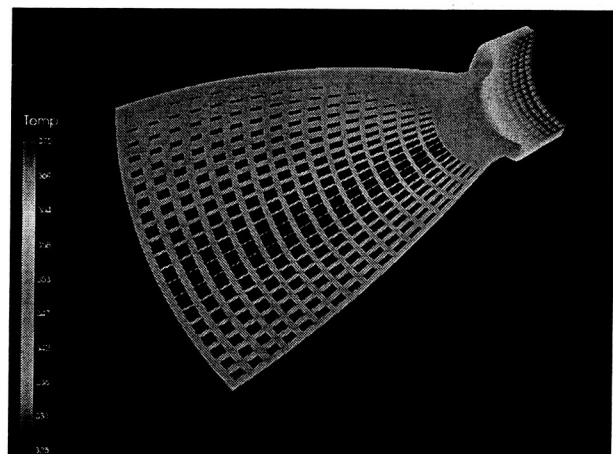


Figure 17 Quarter-nozzle Wall Temperature

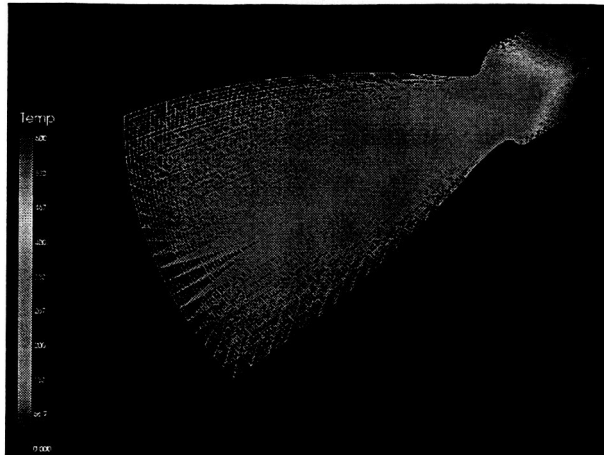


Figure 18 Quarter-nozzle Fluid Temperature

Future analysis plans are to modify the wall material to use a proper nozzle material in place of steel. The chemical capabilities of the HYP code will be used to model combustion, allowing output of species concentrations in the fluid flow. The PCS Viewer will be updated to allow displaying output of vector fields to better report the fluid analysis.

MTF Jet Impingement

Rick

CONCLUSIONS/FUTURE WORK

The results above demonstrate the power and capability of the multiphysics approach. Considerable effort has been expended to develop this toolset. Currently the multiphysics team is at the early stages of integration of the disparate physics modules. Future efforts include validation and verification of the toolset against experimental and other computational data;. Development of post processing routines to calculate net performance is required to integrate the multiphysics results with the design tools for other vehicle subsystems. Finally development of optics, neutronics, dynamic structures, and incompressible flow modules are necessary to expand the capability of the multiphysics tool to other propulsion systems.

¹ The Visualization Toolkit is open-source data visualization software. Contact Kitware at www.kitware.com for more information.

² Incropera, Frank P., Dewitt, David P., Fundamentals of Heat and Mass Transfer, Fourth Ed., John Wiley & Sons, 1996.

³ Chung, T.J., Computational Fluid Dynamics, First Ed., Cambridge University Press, 2002.