Evaluation of an open-source finite element package, WARP3D

David F. Dolphin

B.Eng in Mechanical Engineering

March 2009

The University of Limerick Department of Mechanical and Aeronautical Engineering

Evaluation of an open-source finite element package, WARP3D

David Dolphin
ID: 0548693
david.dolphin@csn.ul.ie
B.Eng in Mechanical Engineering

Supervisor: Prof. Noel O'Dowd

Final Year Project report submitted to the University of Limerick, March 2009

I declare that this is my work and that all contributions from other persons have been appropriately identified and acknowledged.

Abstract

WARP3D is an Open Source Finite Element solver. This report compares the basic commands involved in modeling a WARP3D problem to their ABAQUS counterparts. Post-processing software was written in Matlab to visualise results obtained from WARP3D.

Acknowledgements

I’d like to thank Noel O’Dowd and Hugh O’Brien, for their unwavering support.

Contents

List of Figures

Chapter 1

Introduction

Finite Element Analysis is used increasingly in all fields of engineering to cut down on development and prototyping time. It enables engineers to quickly compute otherwise laborious and tedious Finite Element Equations.

WARP3D is an open source Finite Element solver. As it is open source, the Fortran source code is readily available under the GNU Public License. This enables programmers and engineers to modify the application in any way they deem necessary. It is also free software; it is not necessary to purchase a License to use the software.

WARP3D has been available since 1995 and currently supports many of the features available in other solvers, including, but not limited to: mesh-tieing, fracture growth analysis, parallel execution, and domain-integral computations.

The aim of this project is to investigate the foundation features of WARP3D and compare them to the existing commercial package, Simulia ABAQUS. We also intend on viewing the output from WARP3D in a meaningful way.

It is presumed that the audience of this report are familiar with the Finite Element Method. As a result this report does not cover the math involved in Finite Element Analysis.

It is presumed that the audience are familiar with Simulia ABAQUS, and can model simple problems using the CAE package, submit a model for analysis, and view results.

Finally, it is presumed that the audience are familiar with Mathworks Matlab, can execute commands, and are familiar with basic graphing and file manipulation functions.

Chapter 2

Literature Review

WARP3D is capable of outputting results in the MSC/PATRAN format (MSC Software Corporation 2009) . As a result WARP3D output can be read into Patran.

The PATRAN format can be read by a variety of commercial post-processing applications, including ABAQUS and ANSYS. However, in both cases a translator license is required.

Chapter 3

System Description

The hardware used for this project were x86 compatible PC's.

WARP3D release 15.9 was used. It requires Microsoft Windows (2000 or later), Linux, SGI Linux, or HP-UX (11.x or later) (Com n.d.).

Matlab 6.5 (release 13) was used. It requires 128Mb of RAM (256Mb recommended) and Microsoft Windows (98 or later), OSX (10.1.4 or later), or Linux (Kernel 2.2.1 or later). In order to perform the graphing functions in Matlab an OpenGL compatible graphics card is required. Further details can be found on the manufacturers' website (Mathworks 2002) .

Simula ABAQUS version 6.6-2 was used. It requires 256Mb of RAM (512Mb recommended) and a Microsoft Windows (2000 SP3 or later), Linux, HP-UX (11.22 or later), IRIX (6.6.8 or later), or AIX (5.2 or later). Further details can be found on the manufacturers' website (Dassault Systemes, Simulia Corp. 2006) .

ABAQUS consists of several sub-packages. Those within the scope of this project include; ABAQUS/CAE, ABAQUS/Standard, ABAQUS/Explicit and ABAQUS/Viewer.

ABAQUS/CAE is a pre- and post-processer for results, it is used to sketch models, mesh them and then view results (stresses, strains, displacements, &c.).

ABAQUS/Standard is a general purpose solver used for solving linear and nonlinear problems.

Chapter 4

Comparison of File Formats

4.1 Breakdown of an ABAQUS input file

The following is an ABAQUS input file, using a square element with loading and unloading in the X direction.

Description of the problem

```
*HEADING
Element type:CPS8  Mesh of   1 x   1 elements
```

Define nodes and assign them to the ALLNODES set.

```
*NODE, NSET=ALLNODES
     1,0.000000E+00,0.000000E+00
     2,0.500000E+00,0.000000E+00
     3,0.100000E+01,0.000000E+00
     4,0.000000E+00,0.500000E+00
     5,0.100000E+01,0.500000E+00
     6,0.000000E+00,0.100000E+01
     7,0.500000E+00,0.100000E+01
     8,0.100000E+01,0.100000E+01
```

Define node sets used for loading and constraints.

```
*NSET, NSET=ALLN  , GENERATE
     1,    8,     1
*NSET, NSET=XAXIS,GENERATE
 1,3,1
*NSET, NSET=YAXIS
 1,4,6
*NSET,NSET=NDISPX
3,5,8
```

Define element type, here 2D quad (8 node) element is used. Nodes are also assigned to the elements.

```
*ELEMENT, TYPE=CPS8
```

```
     1,      1,      3,      8,      6,      2,      5,      7,      4
```

Assign elements to element sets.

```
*ELSET, ELSET= ALLE
  1
```

Define material properties and the element sets they apply to.

```
*SOLID SECTION,ELSET=ALLE, MATERIAL=CERAMIC
*MATERIAL, NAME=CERAMIC
*ELASTIC
400,0.3
```

Define symmetrical boundary conditions.

```
*BOUNDARY
XAXIS,    2
YAXIS,    1
```

Define displacement amplitude.

```
*AMPLITUDE, NAME=DISP           , VALUE=RELATIVE, TIME=STEP ...
   TIME
    0.000,      0.000,    100.000,      0.100
```

Define loading. The four numbers represent initial time increment, total time, minimum time increment and maximum time increment.

```
*STEP, INC=   10
*STATIC
    1,   1,      0.0001,      1.00
```

Apply nodal displacements

```
*BOUNDARY, AMPLITUDE= DISP
NDISPX,  1,  1,     1.000
```

Generate results.

```
*NODE PRINT, FREQ=    0
*EL PRINT,FREQ=    1
S,E
*OUTPUT,FIELD,VARIABLE=PRESELECT,FREQ=1
*NODE OUTPUT
RF
*OUTPUT,HISTORY,VARIABLE=PRESELECT,FREQ=1
*ELEMENT OUTPUT,ELSET=ALLE
S,E
*RESTART, WRITE, FREQ=  1,OVERLAY
*END STEP
```

4.2 Breakdown of a WARP3D input file

Some differences exist between the ABAQUS and WARP3D input formats. Blocking is defined for multi-processor work and there are no node or element sets.

Define the name of the problem:

```
structure square
```

Define any materials used in the problem, giving their engineering properties.

```
material a515_grade_70_steel
    properties mises  e 30000 nu 0.3 yld_pt 60.0 n_power 10.,
                       rho 1.0
```

Define the number of nodes and elements used in the problem.

```
number of nodes     8
number of elements     1
```

Define element types. In this example our element is an 8 node 3D isoparametric element (l3disop), as WARP3D does not support 2D elements with small strain kinematic formulation (linear). The previously defined material is assigned using standard 2x2x2 Gauss integration.

Here only one element has been defined, but it is possible to define multiple elements, each with different properties.

```
elements
     1  type  l3disop nonlinear  material  a515_grade_70_steel,
                   order 2x2x2  short  bbar
```

Coordinates are indexed and defined in an x-y-z manner.

```
coordinates
     1   0   0   0
     2   1   0   0
     3   1   1   0
     4   0   1   0
     5   0   0   1
     6   1   0   1
     7   1   1   1
     8   0   1   1
```

The incidences reference how each node is placed in an element. WARP3D expects to the defining elements of the cube in a predefined order, dependant on element type.

```
incidences
     1     5     1     4     8     6     2     3     7
```

Blocking maps elements to block domains. This is necessary for parallel execution, as each block may be executed on a different processor.

```
blocking
     1     1     1
```

Define constraints and displacements of the problem, specifying the plane to constrain and whether we are constraining (u=0) or displacing (u=1).

```
constraints
    plane x=0 u=0
    plane y=0 v=0
    plane z=0 w=0
    plane x=1 u=1
```

In this example forces are not applied, but displacements, yet WARP3D requires a loading step, therefore the dummy force has no magnitude.

```
loading dummy
  nodal loads
    1  force_y 0.0
```

Specify our loading and unloading conditions.

```
 loading bend
  nonlinear
    step 1-5  dummy  1
    step 6-10 dummy 0
```

Next specify our problem settings, including number of iterations, stabilisation factors, time steps and convergence test tolerances.

```
   nonlinear analysis parameters
   solution technique direct sparse
   maximum iterations 5
   minimum iterations 1
   maximum linear iterations 10
   preconditioner type hughes-winget
   convergence test norm res tol 0.05 maximum residual tol 0.5
   time step  1.0e10
   trace solution on lpcg_solution off
   linear stiffness iteration one off
   adaptive solution on
   batch messages off
   bbar stabilization factor 0.0
   extrapolate on
```

Finally we specify our output formats. For post processing the important commands are the *output patran neutral* and *output patran formatted nodal displacements stresses strains*.

```
   compute displacements for loading bend for step 1-10
   output  displacements nodes 1-8
   output displacements eleme   nts 1
   output strains steps 1-10
   output wide eformat noheader strains 1
   output wide eformat noheader stresses 1
   output patran neutral
   output patran formatted nodal displacements stresses strains
c
stop
```

Chapter 5

Software Development

One of the goals of this project was to view results created by WARP3D. The application WarpViz was created to view WARP3D output in Matlab.

The WARP3D and PATRAN mesh formats are similar to the Polygon file format (Bourke 2009) (PLY) also know as the Stanford Triangle Format (Stanford Computer Graphics Laboratory 1994). The PLY format can be natively read by MeshLab, i-sight (Sourceforge 2007) and Wolfram Mathemathica (Wolfram Mathematica Documentation Center 2009) and interfaces exist for a variety of platforms, including Matlab. The Matlab PLY interface was used as reference when createing the WarpViz application (Burkardt 2007) .

In order to view results we first need our WARP3D problem to output a Patran neutral file. This is parsed with the process_neutral_file function. An array of node co-ordinates and element incidences are returned as two arrays.

Matlab plots 3D polygons so the cube elements must be redefined as a series of faces. The patran element incidences are parsed into a series of polygon vertexes using the nodes_to_faces function.

For displacements the WARP3D problem must output Patran ASCII formatted nodal displacements, stresses and strains.

The Patran displacements file is parsed into 3 by x array of x-y-z displacement values using the read_disp function. In order to visualise the displacements the magnitude of the displacement vector is found with the displacement_magnatuide function.

The read_strain and read_stress functions parse the Patran strain and stress files respectively. They output 22 by x and 26 by x arrays respectively, using the nomenclature referenced in the WARP3D manual, Figure 2.10 (Com n.d.).

These functions are all called by the warp_viz function and fig file, which provide a graphical user interface for the text based functions.

Chapter 6

Software Evaluation

6.1 Usability

Usability research was conducted on existing Finite Element software. A critical paper on Human Computer Interaction, and the Psychology of Computer interaction was referenced throughout this project, Larry L. Constantine's "What do users want? Engineering Usability into Software"(Constantine 1995). This will enable engineers to quickly adapt to WarpViz.

It was discovered that one of the key stumbling blocks in learning to use a piece of engineering software was the lack of text feedback. Users were required to remember the functions associated with countless icons and buttons.

The Access rule states that "Good systems are useable without help or instruction". To address this text based labels were used, clearly defining what each function performed.

The Reuse principal states "[maintain] consistency with purpose rather than merely arbitrary consistency". To combat this, WarpViz uses the Matlab figure GUI and maintains all functionality. Users already familiar with graphing in Matlab will be able to simply rotate meshes, print or export results, change views and modify titles.

The Simplicity principal states "Make simple, common tasks simple to do". Within the WarpViz dropdown menu users can easily access common functions to toggle various visualisation options, including mesh visibility, deformation display and contour shading options.

According to Card, S.K., "Users tend to break a large task into a series of unit tasks within which behaviour is highly integrated"(Card, Moran & Newell 1983). WarpViz addresses this by breaking the task of viewing results into a series of optional steps. For example, if users do not need to view stress information then stress files do not need to be loaded.

6.2 Accuracy

WarpViz output was compared to ABAQUS/Viewer output for similar problems and result were conclusive, stress peaks where shown to occur in the same regions and deformed meshed were identical.

Chapter 7

Results

The following results were taken from WARP3D sample problem 42, an Elastic-Plastic steel bar undergoing rolling.

The input file describes the problem as ”[. . .] simulates rolling of a 2x2x10 inch bar of steel using a contact cylinder. The cylinder initially pushes into the bar a half an inch, then moves across the bar, flattening it. Large residual stresses are left in the bar after rolling is complete.”

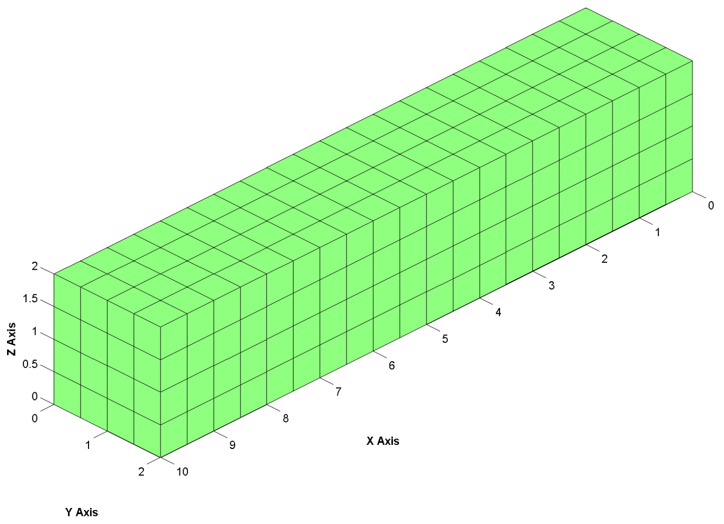

Figure 7.1: Bar Mesh - Step 0

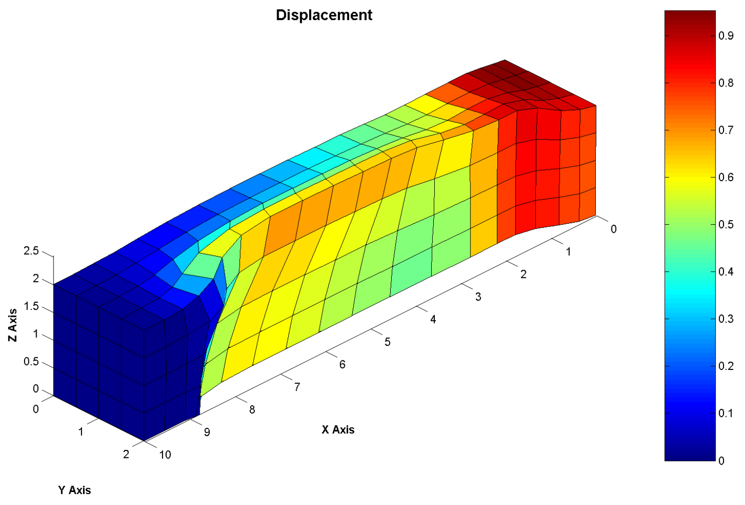

Figure 7.2: Bar Mesh - Displacements - Deformed Mesh - Step 150

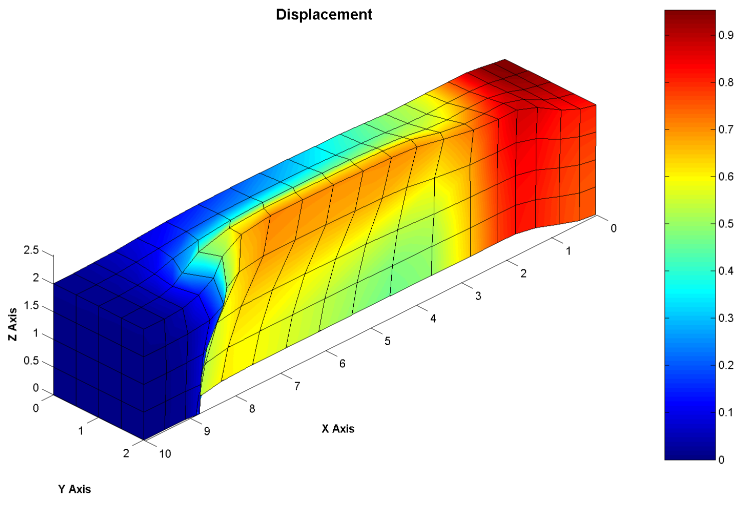

Figure 7.3: Bar Mesh - Displacements - Interpolated contours - Step 150

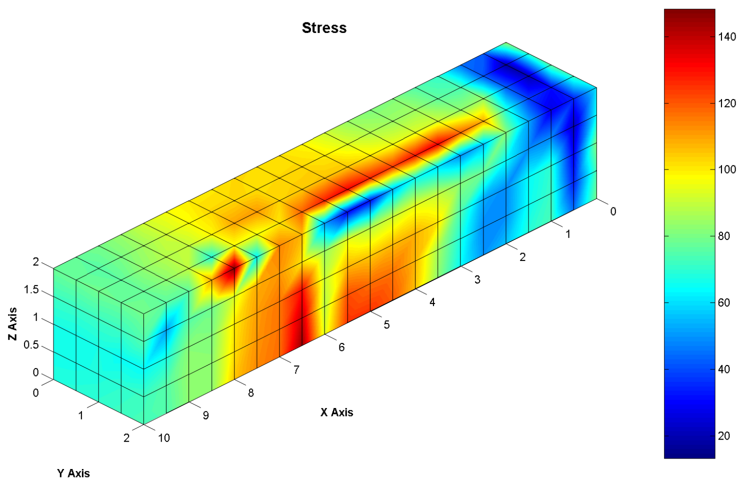

Figure 7.4: Bar Mesh - Stesses - Interpolated contours - Step 150

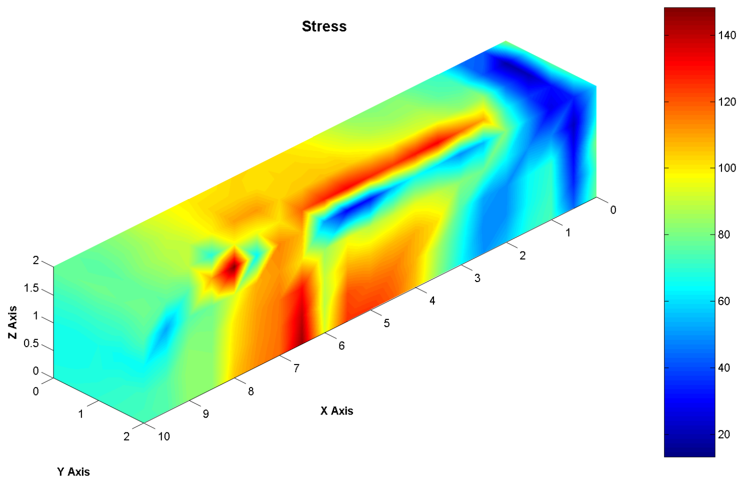

Figure 7.5: Bar Mesh - Stesses - Interpolated contours - No Grid - Step 150

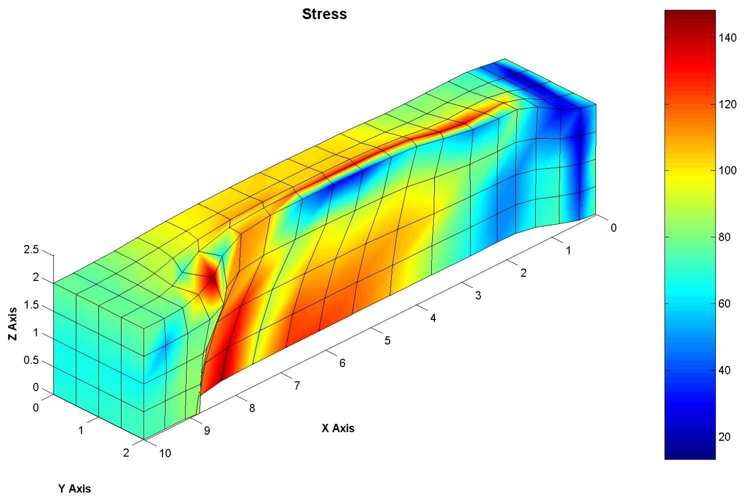

Figure 7.6: Bar Mesh - Stesses - Interpolated contours - Deformed mesh - Step 150

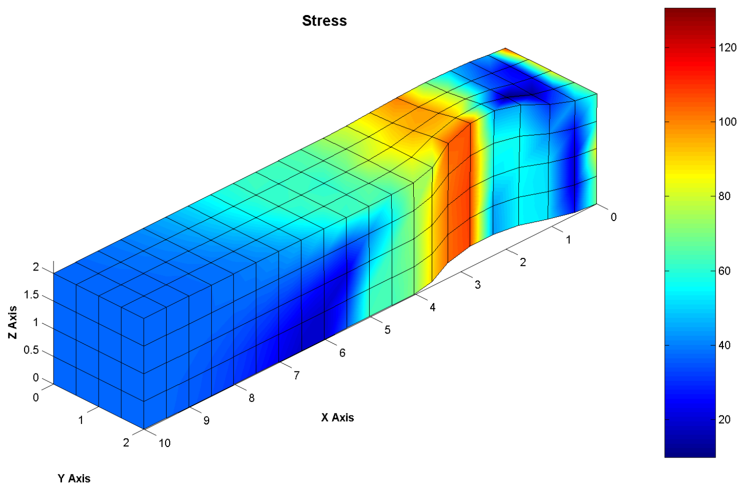

Figure 7.7: Bar Mesh - Stesses - Interpolated contours - Deformed mesh - Step 60

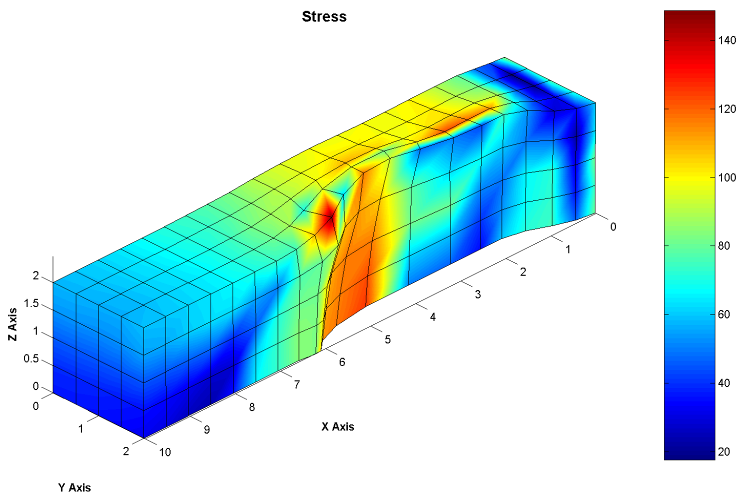

Figure 7.8: Bar Mesh - Stesses - Interpolated contours - Deformed mesh - Step 100

Chapter 8

Discussion

We compared the basic structure of an input file for both ABAQUS and WARP3D. As only ABAQUS has a complete GUI we can only fairly compare their text input methods and results.

The output of WarpViz covers the basic features found in other FEA result visualisation packages. It enables users to pan, zoom, rotate, toggle views, plot contours, edit labels and titles, insert annotations, export to a variety of formats, and print results.

Should free pre-processing tools be developed for WARP3D then an entirely free FEA package would be achievable. This would enable those who are unable to pay for high licenses fees (students, small firms, freelance engineers, &c.) to obtain high quality FEA results.

Chapter 9

Conclusions and Further Work

In some problems visual corruption occurs, elements which should be hidden at the back of the mesh are brought to the front. This has been identified as an issue with the Matlab painters renderer (Solutions 2007).

Matlab uses three methods of graph rendering, painters, opengl and zbuffer. The renderer used in WarpViz is painters, due to its ability to output vector images used in printed output. OpenGL and Z-Buffer rendering were tested but both produce low quality pixel images.

It may be possible to solve this visual corruption if WarpViz takes the camera view angle into account when rendering, and only renders faces which are visible (WarpViz currently renders all faces and corruption occurs due to face placement in the z-plane).

It is noted that even when corrupted images are displayed on a vector output (.pdf) or on the screen, images are correctly exported to pixel formats (.png). While pixel formats are not ideal for print this may be an acceptable compromise.

The deformation scale factor can be modified in the source, adjusting the variable handles.scale. A GUI slider could be developed to improve usability here.

WarpViz only takes into account l3disop WARP3D elements. Additional patch/-surf/trisurf commands and new parser libraries will need to be developed to view meshes with other element types.

The read_strain, read_stress and read_disp functions are written for ASCII formatted files, not binary formatted files. Further libraries could be written to produce the same output used by read_strain, read_stress and read_disp.

While the read_strain function will parse a Patran Strain step file, it has not been interfaced with a graphing function. A Young's Modulus function could be developed, mapping stress against strain.

WarpViz only processes one set of Stain and Displacement step functions at a time. It may be possible to view animations if multiple steps were rendered in sequence.

WarpViz is not compatible with Octave, the third party open source clone of Matlab. If WarpViz was modified to be compatible then a fully open source Finite Element processing and visualisation environment would be available. Currently the setup relies upon Matlab, which is closed source.

WarpViz is only a postprocessor of results and requires manual input file creation, or output from another application. A simple GUI based modeller and meshing application which created WARP3D input files would greatly increase the usability of WARP3D.

Patwarp (included in the WARP3D source directory) can be used to convert from a patran input to a WARP3D input and should be referred to if future pre-processor work is to be completed in this area.

Appendix A

User Manual

A.1 How to run simple WARP3D problem (using Microsoft Windows)

- Download the WARP3D compressed archive (.zip) from `http://cee-ux49.cee.uiuc.edu/cfm/warp3d.html`. File name `warp3d_distribute_###.zip` where `###` represents the release number.
- Extract archive to a suitable location (`C:\warp3d`).
- Open the command prompt, click Start – Run – cmd
- Change current working directory to the location of the extracted files (`cd c:\warp3d`).
- Change directory to the run_windows directory.
- Type `warp3d`. WARP3D runs in interactive mode, accepting typed commands and printing results to the terminal.
- To pass a pre-created input file to warp3d for processing use the ¡ operator (`warp3d < ..\example_problems_serial\test_1`). The results are printed to the terminal.
- To save results to a file for later analysis use the ¿ operator (`warp3d < ..\example_pro`
  This .txt file can be viewed with notepad.

Note: Patran and packet files are saved in the folder which the WARP3D command is executed from, which may not be the folder output is saved to.

A.2 How to view WARP3D output using WarpViz

Note: The WARP3D problem executed must provide Patran formatted output. Example problem 14 outputs both a neutral file, and the ASCII formatted displacements and stresses.

- Copy the WarpViz.zip file from the CD which accompanies this report, or download it from `http://www.skynet.ie/~tyrion/fyp/`
- Extract it to a suitable location (`C:\WarpViz`).
- Open Matlab and set the current working directory to the WarpViz directory. This can be done using the location bar in the toolbar, or by typing the command `cd c:\warpviz\`.
- Type warp_viz to execute the WarpViz GUI.
- Click FEA View – Open Patran Neutral File. Navigate to the location of your saved .neutral file (it is located in the folder WARP3D was executed from). In this example the file is called box_gird.neutral .
- To view displacements click FEA View – Open Patran Disp. Step file. Navigate to the location of your saved step files. In this example the file is called wnfd00020 (warp3d nodes formatted displacements, step 00020. Section 2.12.3 of the WARP3D Manual(Com n.d.) deals with the naming of step files).
- To view stresses click FEA View – Open Patran Stress Step file. Navigate to the location of your saved step files. In this example the file is called wnfs00020.
- Using the remaining FEA View menu options it is possible to toggle deformation, the mesh and smooth contours.

A.3 Print Results from WarpViz

- While viewing a model click File – Print preview. The default graph placement is sub-optimal and should be adjusted.
- Click Page setup.
- To rotate the page click Paper – Landscape. Paper sizes can be changed here too.
- Return to the Size and Position tab and click, in sequence, Fill Page, Fix aspect ratio, Center.
- Click OK, then Print.
- Select printer/Operating System specific options.
- Click Print.
- Export WarpViz results as an image
- While viewing a model Click File Export.
- Navigate to the directory you wish to save the file in. The default directory is the Matlab working directory, in this case the WarpViz directory.
- From the Save as type drop down choose the desired format.
- Enter a filename and click Save.

Note: JPEG is a lossy pixel based format. When printing, choose a lossless (.png) or vector (.ai or .eps) format.

Note 2: When Exporting Matlab uses the Print options (Landscape/portraight) and constrains the X pixel dimension to 1166 pixels. To obtain higher quality exported images set the paper orientation to Landscape and rotate the outputted image in an image manipulation package (mspaint).

Appendix B

Source Code

B.1 warp_viz.m

```
function varargout = warp_viz(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @warp_viz_OpeningFcn, ...
                   'gui_OutputFcn',  @warp_viz_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})...
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end


% --- Executes just before warp_viz is made visible.
function warp_viz_OpeningFcn(hObject, eventdata, handles, ...
    varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of ...
    MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to warp_viz (see VARARGIN)

% Choose default command line output for warp_viz
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes warp_viz wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command ...
    line.
function varargout = warp_viz_OutputFcn(hObject, eventdata, ...
    handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

% ------
function warpMenu_Callback(hObject, eventdata, handles)

% ------
function open_neutral_Callback(hObject, eventdata, handles)

[input_file,pathname] = uigetfile( ...
    {'*.neutral', 'Patran Neutral File (*.neutral)';
    '*.*', 'All Files (*.*)'}, 'Pick a File');

if pathname == 0
    return
end

[handles.orig_elements nodes] = process_neutral_file(strcat(...
    pathname,input_file));
handles.faces = nodes_to_faces(nodes);

handles.elements = handles.orig_elements;   % This is to ...
    preserver the loaded information
handles.disp = 0;                           % View displacements
handles.disp_mag = 0;                       % Display a contour
handles.scale = 0.5;                        % Scale/factor of ...
    deformation
handles.bar = 0;                            % Show a color bar
handles.title_text = '';                    % Title of the graph
handles.edge = 'k';                         % Show edges / Mesh....
     Default color can be changed here too
```

```
handles.face_smooth = 'flat';                % Smooth color ...
    transitions in countour or pixel edges
handles.face_smooth_tog = 0;                 % Smooth color ...
    transitions in countour or pixel edges

plotData(handles);

guidata(hObject, handles);

% ———
function open_disp_Callback(hObject, eventdata, handles)

[input_file,pathname] = uigetfile( ...
    {'*.*', 'All␣Files␣(*.*)'}, 'Pick␣a␣File');

if pathname == 0
    return
end

handles.pre_disp(:,:) = read_disp(strcat(pathname,input_file)); ...
   % Element Displacements
handles.disp_mag = displacement_magnatuide(handles.pre_disp); % ...
    magnituide of displacement vector

handles.elements = handles.orig_elements .* ((handles.pre_disp *...
    handles.scale)+1); % Map Displacement

handles.title_text = 'Displacement';
handles.disp = 1;
handles.bar = 1;

plotData(handles);

guidata(hObject, handles);

% ———
function open_stress_Callback(hObject, eventdata, handles)

[input_file,pathname] = uigetfile( ...
    {'*.*', 'All␣Files␣(*.*)'}, 'Pick␣a␣File');

if pathname == 0
    return
end

stress = read_stress(strcat(pathname,input_file));

```

```
handle.stress_von_mises = stress(:,8);
handles.disp_mag = handle.stress_von_mises;

handles.title_text = 'Stress';
handles.disp = 0;
handles.bar = 1;

plotData(handles);

guidata(hObject, handles);

% ———
function toggle_mesh_Callback(hObject, eventdata, handles)

if handles.edge == 'k'
    handles.edge = 'none';
else
    handles.edge = 'k';
end
plotData(handles);
guidata(hObject, handles);

% ———
function toggle_smooth_Callback(hObject, eventdata, handles)

handles.face_smooth

if handles.face_smooth_tog == 0
    handles.face_smooth = 'interp';
    handles.face_smooth_tog = 1;
else
    handles.face_smooth = 'flat';
    handles.face_smooth_tog = 0;
end
plotData(handles);
guidata(hObject, handles);

% ———
function toggle_deform_Callback(hObject, eventdata, handles)

if handles.disp == 0
    handles.elements = handles.orig_elements .* ((handles....
        pre_disp * handles.scale)+1);
    handles.disp = 1;
else
    handles.elements = handles.orig_elements;
    handles.disp = 0;
```

```
end
plotData(handles);
guidata(hObject, handles);

% ---------- GRAPH PLOTTING FUNCTION ------------
function plotData(handles)

cla(handles.axes1);
axes(handles.axes1);

patch('Vertices',handles.elements,'Faces',handles.faces, '...
    FaceColor', handles.face_smooth, 'FaceVertexCData', handles....
    disp_mag, 'EdgeColor', handles.edge);
axis image;
view(135,30);

xlabel('X Axis', 'FontWeight', 'Bold');
ylabel('Y Axis', 'FontWeight', 'Bold');
zlabel('Z Axis', 'FontWeight', 'Bold');

title(handles.title_text, 'FontWeight', 'Bold', 'FontSize', 14);

if handles.bar ~= 0
    colormap(jet(100));
    col = colorbar;
end;
```

B.2 process_neutral_file.m

```
function [elements,nodes] = process_neutral_file(neutfile)
% Processes a Partan neutral file and asignes the
% elements and nodes to two seperate arrays

fid=fopen(neutfile);

tline = fgetl(fid);
tline = fgetl(fid);
element_no = str2num(fgetl(fid));
tline = fgetl(fid);

% element_no holds the header line which specifies the number of
% elements and nodes. These numbers are used next in the loops

for i=1:element_no(5),
    tline = fgetl(fid);
    a(:,i) = fscanf(fid, '%f'); % Scan for floats
    tline = fgetl(fid);
end

for j=1:element_no(6),
    tline = fgetl(fid);
    tline = fgetl(fid);
    nodes(j,:) = str2num(fgetl(fid)); % Convert the string to an...
         array
end

fclose(fid);

elements = transpose(a);
elements(:,4) = [];
```

B.3 nodes_to_faces.m

```matlab
function [faces] = nodes_to_faces(nodes)
% Convert an array of patran nodes into an array of Matlab patch...
    faces

old_rows = size(nodes);

for i=1:old_rows(1,1),
    j = (i - 1) * 6;

    faces(1 + j,1) = nodes(i,1);
    faces(3 + j,1) = nodes(i,1);
    faces(5 + j,1) = nodes(i,1);

    faces(1 + j,2) = nodes(i,2);
    faces(3 + j,2) = nodes(i,2);
    faces(6 + j,1) = nodes(i,2);

    faces(1 + j,3) = nodes(i,3);
    faces(4 + j,1) = nodes(i,3);
    faces(6 + j,2) = nodes(i,3);

    faces(1 + j,4) = nodes(i,4);
    faces(4 + j,2) = nodes(i,4);
    faces(5 + j,2) = nodes(i,4);

    faces(2 + j,1) = nodes(i,5);
    faces(3 + j,4) = nodes(i,5);
    faces(5 + j,4) = nodes(i,5);

    faces(2 + j,2) = nodes(i,6);
    faces(3 + j,3) = nodes(i,6);
    faces(6 + j,4) = nodes(i,6);

    faces(2 + j,3) = nodes(i,7);
    faces(4 + j,4) = nodes(i,7);
    faces(6 + j,3) = nodes(i,7);

    faces(2 + j,4) = nodes(i,8);
    faces(4 + j,3) = nodes(i,8);
    faces(5 + j,3) = nodes(i,8);
end
```

B.4 read_disp.m

```
function [el] = read_disp(patran_disp_file)
% Reads and parses the Partan strain file into a 3 by x array of...
    displacements

[element el(:,1) el(:,2) el(:,3)] = textread(patran_disp_file, '...
    %d %f %f %f', 'headerlines', 4);
```

B.5 displacement_magnatuide.m

```
function [disp_mag] = displacement_magnatuide(disp)
% Calculate the magnatude of the XYZ displacement vector

disp_size = size(disp);

for i=1:disp_size(1,1),
    disp_mag(i) = sqrt(disp(i,1)^2 + disp(i,2)^2 + disp(i,3)^2);
end

disp_mag = transpose(disp_mag);
```

B.6 read_stress.m

```
function [el] = read_stress(patran_stress_file)
% Reads and parses the Partan stress file into a 26 by x array ...
    of stresses

[element el(:,1) el(:,2) el(:,3) el(:,4) el(:,5) el(:,6) el(:,7)...
    el(:,8) el(:,9) el(:,10) el(:,11) el(:,12) el(:,13) el(:,14)...
    el(:,15) el(:,16) el(:,17) el(:,18) el(:,19) el(:,20) el...
    (:,21) el(:,22) el(:,23) el(:,24) el(:,25) el(:,26)] = ...
    textread(patran_stress_file, '%d %f %f %f %f %f %f %f %f %f %...
    f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f', '...
    headerlines', 4, 'whitespace', ' \b\n\r\t');
```

B.7 read_strain.m

```
function [el] = read_strain(patran_strain_file)
% Reads and parses the Partan strain file into a 22 by x array ...
    of strains

[element el(:,1) el(:,2) el(:,3) el(:,4) el(:,5) el(:,6) el(:,7)...
    el(:,8) el(:,9) el(:,10) el(:,11) el(:,12) el(:,13) el(:,14)...
    el(:,15) el(:,16) el(:,17) el(:,18) el(:,19) el(:,20) el...
    (:,21) el(:,22)] = textread(patran_strain_file, '%d %f %f %f ...
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f', '...
headerlines', 4, 'whitespace', ' \b\n\r\t');
```

References

Bourke, P. (2009), 'Ply - polygon file format', http://local.wasp.uwa.edu.au/~pbourke/dataformats/ply/. [Online, accessed 2009-01-18].

Burkardt, J. (2007), 'Ply-io', http://people.sc.fsu.edu/~burkardt/m_src/ply_io/ply_io.html. [Online, accessed 2009-02-06].

Card, S., Moran, T. & Newell, A. (1983), *The psychology of human-computer interaction*, Lawrence Erlbaum Associates.

Com (n.d.), *WARP3D User-Theory Manual*, 15.9 edn. http://cern49.cee.uiuc.edu/cfm/documents/WARP3D_v15.9_manual.pdf.

Constantine, L. (1995), 'What do users want? Engineering usability into software', *Windows Tech Journal* **4**(12), 30–39.

Dassault Systemes, Simulia Corp. (2006), 'System requirements and tested configurations for ABAQUS version 6.6 & 6.6-ef products', http://www.abaqus.com/support/v66/v66_sysRqmts.html. [Online, accessed].

Mathworks (2002), 'Matlab support - system requirements - current release 13 - Windows', http://www.mathworks.com/support/sysreq/release13/index.html. [Online, accessed 2009-03-17].

MSC Software Corporation (2009), 'Patran', http://www.mscsoftware.com/products/patran.cfm. [Online, accessed 2009-01-17].

Solutions, M. T. (2007), '1-1b33h - why does the painter's renderer in matlab display 3-d graphics incorrectly?', http://www.mathworks.com/support/solutions/data/1-1B33H.html?solution=1-1B33H. [Online, accessed 2009-03-10].

Sourceforge (2007), 'i-sight : Mesh visualization and scientific plotting tool', http://i-sight.sourceforge.net/. [Online, accessed 2009-02-05].

Stanford Computer Graphics Laboratory (1994), ‘The stanford 3d scanning repository’, http://www-graphics.stanford.edu/data/3Dscanrep/#file_format. [Online, accessed 2009-01-18].

Wolfram Mathematica Documentation Center (2009), ‘Ply (.ply)’, http://reference.wolfram.com/mathematica/ref/format/PLY.html. [Online, accessed 2009-01-18].