ARL-MR-1065 ● Oct 2022

DEVCOM ARMY RESEARCH LABORATORY

A Script for Mapping Computed Tomography (CT) Values to a Finite Element (FE) Mesh for Use with a Biofidelic Skull Bone Material Model

by Brian Fagan

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-MR-1065 ● OCT 2022

A Script for Mapping Computed Tomography (CT) Values to a Finite Element (FE) Mesh for Use with a Biofidelic Skull Bone Material Model

Brian Fagan
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
October 2022	Memorandum Report	1 September–30 November 2021

4. TITLE AND SUBTITLE
A Script for Mapping Computed Tomography (CT) Values to a Finite Element (FE) Mesh for Use with a Biofidelic Skull Bone Material Model

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Brian Fagan

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLW-TB
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-MR-1065

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Head protection design and evaluation involves determining whether Soldier injury will occur under a given set of impact conditions. Skull fracture is a readily identifiable and prominent indicator of head injury. Recent work completed at the US Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory (ARL) has shown the potential for biofidelically modeling skull bone and the ability to introduce biovariability in the model. This material model's properties are determined based on the local bone volume fraction determined from corresponding computed tomography (CT) images. A script was developed within ARL to associate the CT image data with the spatial locations of a finite element mesh required by the skull bone material model of interest. With this tool, modelers will be able to more efficiently generate new models that use the biofidelic skull bone material model of interest.

15. SUBJECT TERMS
Terminal Effects, material mapping, skull mechanics, finite element model, behind-helmet blunt trauma

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Brian Fagan
Unclassified	Unclassified	Unclassified	UU	25	19b. TELEPHONE NUMBER (Include area code) (410) 278-3841

Contents

List of Figures

1. Introduction and Background

Head protection design and evaluation involves determining whether Soldier injury will occur under a given set of impact conditions. One such example—behind-helmet blunt trauma (BHBT)—occurs when the inner surface of the helmet deforms under a non-penetrating impact and strikes the head. Skull fracture is one prominent indicator of head injury that is readily identifiable. Skull fracture may have implications for injuries that are not specific to injury mechanisms in the brain, such as hemorrhaging due to soft tissue damage caused by the fracture, which can produce serious outcomes. Therefore, the ability to predict skull fracture can provide a tool for evaluating the effectiveness of current and future head protection concepts.

Modeling of bone failure is a complex topic that is currently an ongoing area of research (Alexander et al. 2020, 2021; Weerasooriya and Alexander 2021). Recent work completed at the US Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory (ARL) (Alexander et al. 2021) has shown the potential for biofidelically modeling skull bone and the ability to introduce biovariability in the model. One of the key features of this modeling effort is the ability to determine bone constitutive properties based on the bone volume fraction (BVF) at the given location in the skull. For biofidelic models, these values can be obtained through computed tomography (CT) imaging. One of the constraints that Alexander et al. (2021) and Weerasooriya and Alexander (2021) ran into while developing their model was mapping CT data (i.e., BVF) to the corresponding spatial location in the finite element (FE) mesh. In an earlier study (Weerasooriya and Alexander 2021), this process required a specialized software package (i.e., the FEA module of Mimics and 3-Matic); therefore, the meshing and property mapping work was contracted out.

In this report, a script developed internally at ARL to associate the CT image data with the spatial locations of an FE mesh required for the material model (Alexander et al. 2021) will be outlined.

2. Method

The script (see the Appendix) to associate CT image data (i.e., BVF) with the corresponding spatial location of an FE mesh was developed in MATLAB (version R2021a). The script has six main steps in the process to generate an FE mesh with mapped CT data (see Fig. 1).

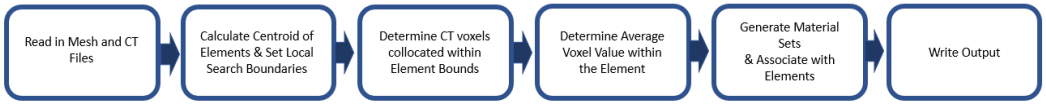

Fig. 1 Outline of script's function

First, it reads in both the CT image files and the FE mesh. Currently, the mesh is limited to Abaqus input decks and only tetrahedral (C3D4) and hexahedral (C3D8R) elements. This could be expanded on later, but it is sufficient for now. The mesh needs to be in the same spatial location and orientation as the CT images for this method to work. Second, it determines the centroid of each element and sets local voxel search bounds for later use. These localized bounds around the centroid of each element were implemented to increase the efficiency of the program (see Fig. 2). Next, within the limited voxel search bounds, it determines the voxels that fall within bounds of the element geometry. This was done by checking to ensure that the normal direction of each face (face normal directions are defined toward the inside of the element) and the vector from a node associated with that face and the voxel are aligned (i.e., the dot product of the two vectors is positive). Next, the average voxel value contained withing the element is computed. The fifth step is to generate the material sets and then associate each element with one of the material sets based on average voxel value. Similar to the biofidelic model (i.e., element-based mapping) used by Alexander et al. (2021), the script is setup to generate 100 material sets (spanning BVF = 0 for void and BVF = 1 for solid bone) each with its own BVF to capture the distribution in the model. This is editable in the script to either coarsen or refine the model's BVF distribution. Finally, the program writes to a new output file, formatted as an Abaqus input deck (.inp), a copy of the mesh along with the material sets.

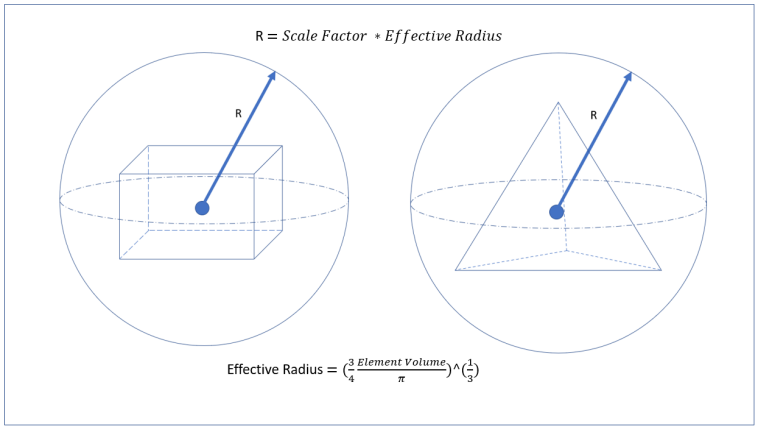

Fig. 2 Localized search around each element

3. Results

The script was tested on three meshes and corresponding CT images to generate FE meshes with mapped BVF values. First, a bone beam specimen (B01) from a set of bending experiments (Gunnarsson et al. 2021) was meshed with both tetrahedral and hexahedral elements to test the script. The script does a good job of mapping the bone density to the correct spatial location in the mesh (see Fig. 3). In Fig. 3, yellow represents BVF = 1 (solid bone) and dark blue represents BVF = 0 (void space).

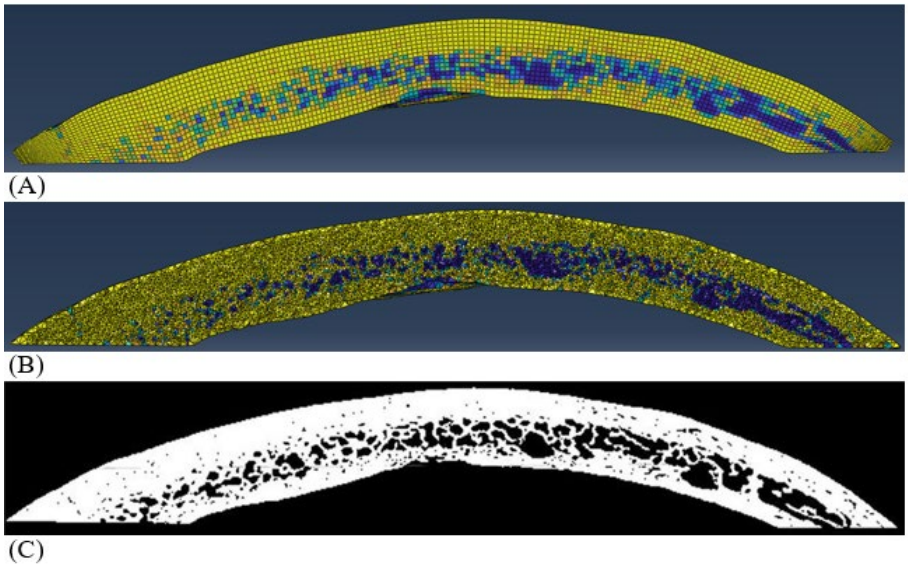

Fig. 3 (A) Mapped hexahedral mesh at the mid-plane of the beam. Yellow represents solid bone (BVF = 1) and dark blue represents a void (BVF = 0). (B) Mapped tetrahedral mesh at the mid-plane of the beam. (C) Micro-CT image at the mid-plane of specimen B01 (Gunnarsson et al. 2021).

The next mesh used to test the script was for a skull cap from a set of indentation experiments (Gunnarsson et al. 2019) composed of hexahedral elements. Once again, the script does a good job of mapping the bone density from the micro-CT images of the skull cap to the corresponding locations in the meshes (see Figs. 4 and 5).

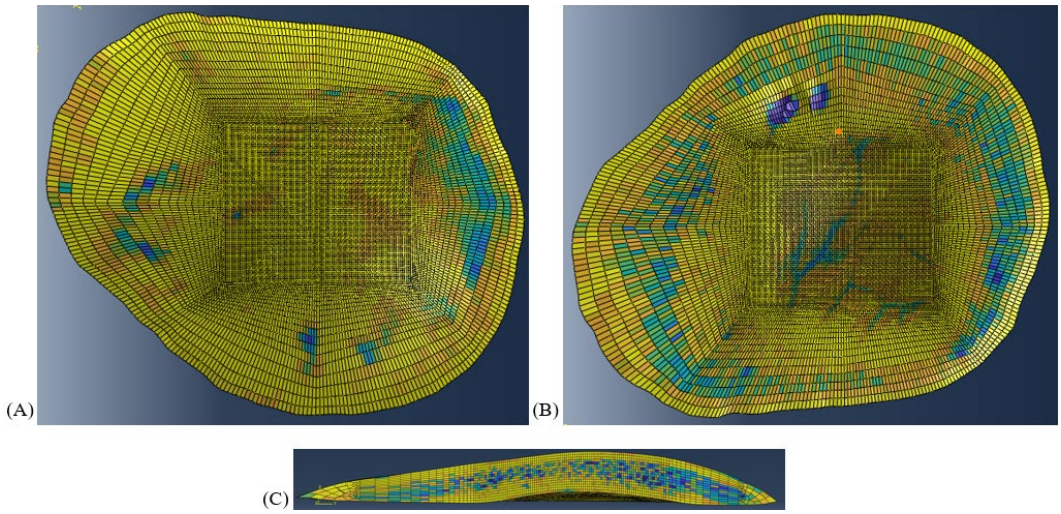

Fig. 4 Hexahedral mesh with BVF mapped from CT images. Yellow represents solid bone (BVF = 1) and dark blue represents a void (BVF = 0). (A) Top view. (B) Bottom view. (C) Cross section at the mid-plane.

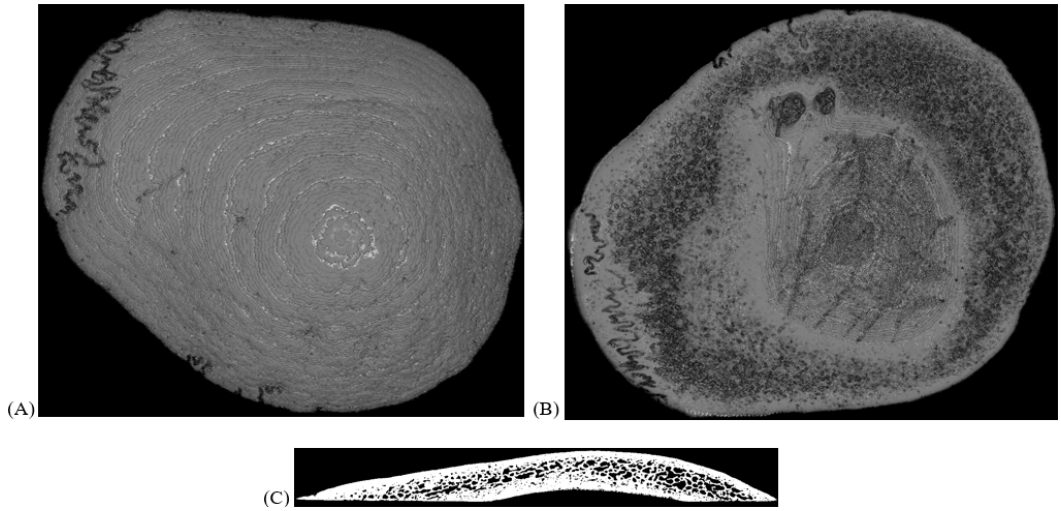

Fig. 5 (A) Top view of a volumetric reconstruction of the skull cap from CT images. (B) Bottom view of the same volumetric reconstruction. (C) Mid-plane CT image of the skullcap.

4. Conclusion

A script to spatially map BVF from micro-CT images to an FE mesh was developed. This script provides a valuable tool for the generation of highly biofidelic models that can take advantage of a constitutive model for bone and its failure, which is of great interest to the modeling community. These models will in turn assist in a more rapid analysis of BHBT and evaluating helmet performance.

With this tool, modelers will be able to efficiently generate new meshes or make alterations (e.g., change refinement of elements and element type) to current ones.

Previously, the generation of these meshes and the spatial mapping of BVF from micro-CT images had been contracted work but now this work can be done internally at ARL. Furthermore, this script allows modelers to expand the use of the bone constitutive model to FE models using hexahedral elements, which was previously unexplored due to limitations on the generation of new meshes.

5. References

Alexander SL, Baumer TG, Fagan BT, Weerasooriya T. Hybrid experimental modeling computational (HEMC) skullcap simulation: elemental to layer simplification and application to microstructural stochasticity. DEVCOM Army Research Laboratory (US); 2021. Report No.: ARL-TR-9296.

Alexander SL, McKee JP, Weerasooriya T. Micro-CT-based three-layer finite element model for quasi-static human skull impact. DEVCOM Army Research Laboratory (US); 2020. Report No.: ARL-TR-8962.

Gunnarsson CA, Alexander SL, Rafaels K, Walter T, Weerasooriya T. High-rate fracture of human skull. In: Grady M, Minary M, Starman L, Hay J, Notbohm J, editors. Mechanics of biological systems & micro- and nanomechanics, Vol 4. Proceedings of the Society for Experimental Mechanics Series; Springer, Cham; c2019. p. 91–94. doi: 10.1007/978-3-319-95062-4_20.

Gunnarsson CA, Alexander SL, Weerasooriya T. Bending response of human skull-beams as a function of loading rate and loading-tip geometry. DEVCOM Army Research Laboratory (US); 2021. Report No.: ARL-TR-9215.

MATLAB. Release R2021a. The Mathworks, Inc; 2021.

Weerasooriya T, Alexander SL. Mechanism and microstructure based concept to predict skull fracture using a hybrid-experimental-modeling-computational approach. J Mech Behav Biomed. 2021;121:104599.

Appendix. Script to Associate Computed Tomography (CT) Bone Volume Fraction (BVF) Data with the Corresponding Spatial Location of a Finite Element (FE) Mesh

The following script (CT_Mesh_Mapping.m) associates the CT image data with the spatial locations of an FE mesh.

```
clear all
clc

%Mesh File In
Mesh_File = 'Location of the mesh file';


%DICOM Folder
DICOM_Folder = 'Location of the folder of DICOM files of interest';


%Output Mesh File
Out_File = 'Output mesh file in the format: Location\output-mesh-
filename';


index1_correction = 0.0;
index2_correction = 0.0;
index3_correction = 0.0;

%voxel size (length is used and assuming equal sides)
voxel_size = 0.027118000000000;

%Maximum pixel value to be used in coversion to density
%255 for current bone beam specimens
%184 for current skullcap
max_pixel_value = 255;


%Read Mesh File
text = fopen(Mesh_File);

line = fgetl(text);

node_check = 0;
element_check = 0;

Nodes = [];
Element = [];
node_num = 0;
element_num = 0;

%Read and parse mesh file
while ~feof(text)

  % Get new line
  line = fgetl(text);

  % Set flags
  if node_check & strcmp('*',line(1))
```

```
    node_check = 0;
  end
  if element_check & strcmp('*',line(1))
      element_check = 0;
  end
  if strcmp('*NODE', line)
      node_check = true;
  end
  if strcmp('*ELEMENT, TYPE=C3D4', line)
      el_type = 'Tet';
      element_check = true;
  end
  if strcmp('*ELEMENT, TYPE=C3D8R', line)
      el_type = 'Hex';
      element_check = true;
  end

  % Build arrays
  if node_check
      split_line = str2num(line);
      if node_num >= 1
          Nodes(node_num,:) = split_line(2:4);
      end
      node_num = node_num + 1;
  end
  if element_check
      split_line = str2num(line);
      if element_num >= 1
          if strcmp(el_type, 'Tet')
              Elements(element_num,:) = split_line(2:5);
          elseif strcmp(el_type, 'Hex')
              Elements(element_num,:) = split_line(2:9);
          else
              assert(0,'Invalid Element Type');
          end
      end
      element_num = element_num + 1;
  end

end
disp('Completed reading mesh file')

% Convert to mm to match micro-CT units
%Nodes = Nodes; %For Skullcap
Nodes = Nodes.* 1e-3; %For current bone beam specimens

%Calculate Centroids of Elements
if strcmp(el_type, 'Tet')
    Element_X_Centroid = (Nodes(Elements(:,1),1)+Nodes(Elements(:,2),1)
+ Nodes(Elements(:,3),1) + Nodes(Elements(:,4),1))/4;
    Element_Y_Centroid = (Nodes(Elements(:,1),2)+Nodes(Elements(:,2),2)
+ Nodes(Elements(:,3),2) + Nodes(Elements(:,4),2))/4;
    Element_Z_Centroid = (Nodes(Elements(:,1),3)+Nodes(Elements(:,2),3)
+ Nodes(Elements(:,3),3) + Nodes(Elements(:,4),3))/4;
elseif strcmp(el_type, 'Hex')
```

```
    Element_X_Centroid = (Nodes(Elements(:,1),1)+Nodes(Elements(:,2),1)
+ Nodes(Elements(:,3),1) + Nodes(Elements(:,4),1) +
Nodes(Elements(:,5),1)+Nodes(Elements(:,6),1) + Nodes(Elements(:,7),1) +
Nodes(Elements(:,8),1))/8;
    Element_Y_Centroid = (Nodes(Elements(:,1),2)+Nodes(Elements(:,2),2)
+ Nodes(Elements(:,3),2) + Nodes(Elements(:,4),2) +
Nodes(Elements(:,5),2)+Nodes(Elements(:,6),2) + Nodes(Elements(:,7),2) +
Nodes(Elements(:,8),2))/8;
    Element_Z_Centroid = (Nodes(Elements(:,1),3)+Nodes(Elements(:,2),3)
+ Nodes(Elements(:,3),3) + Nodes(Elements(:,4),3) +
Nodes(Elements(:,5),3)+Nodes(Elements(:,6),3) + Nodes(Elements(:,7),3) +
Nodes(Elements(:,8),3))/8;
end

Centroids = [ Element_X_Centroid(:), Element_Y_Centroid(:),
Element_Z_Centroid(:)];
disp('Completed calculating centroids')

%Calculate Volume and effective radius
if strcmp(el_type, 'Tet')
    for i=1:size(Centroids,1)
       Elem_vol(i,1) = 1/6 * abs(det([Nodes(Elements(i,1),1),
Nodes(Elements(i,2),1), Nodes(Elements(i,3),1), Nodes(Elements(i,4),1);
Nodes(Elements(i,1),2),
Nodes(Elements(i,2),2),Nodes(Elements(i,3),2),Nodes(Elements(i,4),2);Nod
es(Elements(i,1),3),Nodes(Elements(i,2),3),Nodes(Elements(i,3),3),Nodes(
Elements(i,4),3);1,1,1,1]));
    end
elseif strcmp(el_type, 'Hex')
    for i=1:size(Centroids,1)
        V1 = 1/6 * abs(det([Nodes(Elements(i,1),1),
Nodes(Elements(i,4),1), Nodes(Elements(i,2),1), Nodes(Elements(i,6),1);
Nodes(Elements(i,1),2),
Nodes(Elements(i,4),2),Nodes(Elements(i,2),2),Nodes(Elements(i,6),2);Nod
es(Elements(i,1),3),Nodes(Elements(i,4),3),Nodes(Elements(i,2),3),Nodes(
Elements(i,6),3);1,1,1,1]));
        V2 = 1/6 * abs(det([Nodes(Elements(i,1),1),
Nodes(Elements(i,4),1), Nodes(Elements(i,5),1), Nodes(Elements(i,6),1);
Nodes(Elements(i,1),2),
Nodes(Elements(i,4),2),Nodes(Elements(i,5),2),Nodes(Elements(i,6),2);Nod
es(Elements(i,1),3),Nodes(Elements(i,4),3),Nodes(Elements(i,5),3),Nodes(
Elements(i,6),3);1,1,1,1]));
        V3 = 1/6 * abs(det([Nodes(Elements(i,4),1),
Nodes(Elements(i,6),1), Nodes(Elements(i,5),1), Nodes(Elements(i,8),1);
Nodes(Elements(i,4),2),
Nodes(Elements(i,6),2),Nodes(Elements(i,5),2),Nodes(Elements(i,8),2);Nod
es(Elements(i,4),3),Nodes(Elements(i,6),3),Nodes(Elements(i,5),3),Nodes(
Elements(i,8),3);1,1,1,1]));
        V4 = 1/6 * abs(det([Nodes(Elements(i,4),1),
Nodes(Elements(i,8),1), Nodes(Elements(i,7),1), Nodes(Elements(i,6),1);
Nodes(Elements(i,4),2),
Nodes(Elements(i,8),2),Nodes(Elements(i,7),2),Nodes(Elements(i,6),2);Nod
es(Elements(i,4),3),Nodes(Elements(i,8),3),Nodes(Elements(i,7),3),Nodes(
Elements(i,6),3);1,1,1,1]));
```

```
        V5 = 1/6 * abs(det([Nodes(Elements(i,4),1),
Nodes(Elements(i,3),1), Nodes(Elements(i,7),1), Nodes(Elements(i,6),1);
Nodes(Elements(i,4),2),
Nodes(Elements(i,3),2),Nodes(Elements(i,7),2),Nodes(Elements(i,6),2);Nod
es(Elements(i,4),3),Nodes(Elements(i,3),3),Nodes(Elements(i,7),3),Nodes(
Elements(i,6),3);1,1,1,1]));
        V6 = 1/6 * abs(det([Nodes(Elements(i,4),1),
Nodes(Elements(i,3),1), Nodes(Elements(i,2),1), Nodes(Elements(i,6),1);
Nodes(Elements(i,4),2),
Nodes(Elements(i,3),2),Nodes(Elements(i,2),2),Nodes(Elements(i,6),2);Nod
es(Elements(i,4),3),Nodes(Elements(i,3),3),Nodes(Elements(i,2),3),Nodes(
Elements(i,6),3);1,1,1,1]));
        Elem_vol(i,1) = V1 + V2 + V3 + V4 + V5 + V6;
    end
end

effective_rad = (3/4*Elem_vol/pi).^(1/3);

% Load CTs
[V,spatial,dim] = dicomreadVolume(DICOM_Folder);
V = squeeze(V);
disp('Completed loading CTs')

% Correct V orientation to match mesh
% Bone specimen
tmp1 = V(:,end:-1:1,:);
V = tmp1;
V = permute(V, [3 2 1]);

% Skullcap specimen
% tmp1 = V(end:-1:1,end:-1:1,:);
% V = tmp1;
% V = permute(V, [2 1 3]);



% Convert Bianirize gray scale 0 to 255 to a density scale from 0 to 1
gray_to_density = 1/max_pixel_value;

voxel_size = voxel_size;

%Calculate nearest centroid indices in the CT
Centroid_indices(:,1) = round((Centroids(:,1)-
index1_correction)./voxel_size, 0) + 1;
Centroid_indices(:,2) = round((Centroids(:,2)-
index2_correction)./voxel_size, 0) + 1;
Centroid_indices(:,3) = round((Centroids(:,3)-
index3_correction)./voxel_size, 0) + 1;

%TEST Volume average density
%Limit later searches for only the portion of the CT that contains
%speciment bounding box
i_min = floor(min(Nodes(:,1),[],'all')/voxel_size) + 1;
i_max = floor(max(Nodes(:,1),[],'all')/voxel_size) + 1;
```

```
j_min = floor(min(Nodes(:,2),[],'all')/voxel_size) + 1;
j_max = floor(max(Nodes(:,2),[],'all')/voxel_size) + 1;
k_min = floor(min(Nodes(:,3),[],'all')/voxel_size) + 1;
k_max = floor(max(Nodes(:,3),[],'all')/voxel_size) + 1;

%Calculate the number of voxels to check around the centroid based on
the
%voxel size and effective radius of the element. A scale factor of 1.5
for
%Hex elements and 2 for Tet elements. Scale factor is used in radius
check
%later
if strcmp(el_type, 'Tet')
  scale_factor = 2;
elseif strcmp(el_type, 'Hex')
  scale_factor = 1.5;
else
  assert(0,'Invalid Element Type');
end

voxels_around = ceil(scale_factor*max(effective_rad)/voxel_size);

Element_density = [];
for cur=1:size(Centroids,1)
    if rem(cur,250) == 0
        disp(cur)
    end

    i_cur_min = Centroid_indices(cur,1) - voxels_around;
    i_cur_max = Centroid_indices(cur,1) + voxels_around;
    j_cur_min = Centroid_indices(cur,2) - voxels_around;
    j_cur_max = Centroid_indices(cur,2) + voxels_around;
    k_cur_min = Centroid_indices(cur,3) - voxels_around;
    k_cur_max = Centroid_indices(cur,3) + voxels_around;

    if i_cur_min <= i_min
        i_cur_min = i_min;
    end
    if i_cur_max >= i_max
        i_cur_max = i_max;
    end

    if j_cur_min <= j_min
        j_cur_min = j_min;
    end
    if j_cur_max >= j_max
        j_cur_max = j_max;
    end

    if k_cur_min <= k_min
        k_cur_min = k_min;
    end
    if k_cur_max >= k_max
        k_cur_max = k_max;
    end
```

```
    Element_density(cur,1) = 0;
    num_voxels = 0;
    for i=i_cur_min:i_cur_max
        for j=j_cur_min:j_cur_max
            for k=k_cur_min:k_cur_max
                x = (i-1)*voxel_size + index1_correction;
                y = (j-1)*voxel_size + index2_correction;
                z = (k-1)*voxel_size + index3_correction;

                inElement = false;
                if ((Centroids(cur,1) - x)^2 + (Centroids(cur,2)-y)^2 +
(Centroids(cur,3)-z)^2)^(1/2) <= scale_factor*effective_rad(cur)
                    inElement = CheckInElement(el_type, Nodes,
Elements(cur,:), x, y, z, voxel_size);

                    if inElement
                        num_voxels = num_voxels +1;
                        Element_density(cur,1) = Element_density(cur,1)
+ double(V(i,j,k));
                    end
                end

            end
        end
    end
    Element_density(cur,1) =
Element_density(cur,1)*gray_to_density/num_voxels;

end

disp('Completed Mapping Densities')

%Number of Materials
NumMats=100;
BinCenter = 1/(2 *NumMats);

elem_mat_set = [];
count = zeros(NumMats,1);

%Calculate which material set each element belongs too
for i = 1:size(Element_density,1)
   for j = 1:NumMats
       %Special Case need to include max bound in the grouping(i.e. <=
instead of <)
       if j == NumMats
           if Element_density(i) >= BinCenter*2*(j-1) &&
Element_density(i) <= BinCenter*2*(j-1) + 2*BinCenter
               count(j) = count(j) + 1;
               elem_mat_set(j,count(j)) = i;
           end
       else
           if Element_density(i) >= BinCenter*2*(j-1) &&
Element_density(i) < BinCenter*2*(j-1) + 2*BinCenter
```

```
                count(j) = count(j) + 1;
                elem_mat_set(j,count(j)) = i;
            end
        end
    end
end

disp('Completed Calculating Material Sets')

%Print the CT density mapped file
file = fopen(Out_File,'w');
fprintf(file,'*HEADING\n');
fprintf(file,'Test file from matlab script\n');
fprintf(file,'** Units: mm\n');
fprintf(file,'*NODE\n');
for i = 1:size(Nodes,1)
   fprintf(file,'\t%i, %16.10E, %16.10E, %16.10E\n', i, Nodes(i, 1),
Nodes(i,2), Nodes(i,3));
end
if strcmp(el_type, 'Tet')
    fprintf(file,'*ELEMENT, TYPE=C3D4\n');
    for i = 1:size(Elements,1)
        fprintf(file,'\t%i, %i, %i, %i, %i\n', i, Elements(i,1),
Elements(i,2), Elements(i,3), Elements(i,4));
    end
elseif strcmp(el_type, 'Hex')
    fprintf(file,'*ELEMENT, TYPE=C3D8R\n');
    for i = 1:size(Elements,1)
        fprintf(file,'\t%i, %i, %i, %i, %i, %i, %i, %i, %i\n', i,
Elements(i,1), Elements(i,2), Elements(i,3), Elements(i,4),
Elements(i,5), Elements(i,6), Elements(i,7), Elements(i,8));
    end
end
for i =1:size(elem_mat_set,1)
    fprintf(file,'*ELSET, ELSET=Volume_0_MAT%i, UNSORTED\n',i-1);
    for j = 1:size(elem_mat_set,2)
       if j == size(elem_mat_set,2)
           fprintf(file,'%i\n',elem_mat_set(i,j));
       else
           if elem_mat_set(i,j) == 0
               break;
           end
           if rem(j,16) == 0
               fprintf(file,'%i\n',elem_mat_set(i,j));
           else
               if elem_mat_set(i,j+1) == 0
                   fprintf(file,'%i\n',elem_mat_set(i,j));
               else
                   fprintf(file,'%i,',elem_mat_set(i,j));
               end
           end
       end
    end
    fprintf(file,'*SOLID SECTION, ELSET=Volume_0_MAT%i,
MATERIAL=Material%i\n', i-1, i-1);
```

```
end
fprintf(file,'\n');
for i =1:NumMats
    fprintf(file,'*MATERIAL, NAME=Material%i\n', i-1);
    fprintf(file,'*DENSITY\n');
    fprintf(file,'%8f\n', BinCenter*2*(i-1) + BinCenter);
end

fclose(file);

disp('Completed Writing Inp file')
disp('Finished')
```

CheckInElement()

This function is called within the script CT_Mesh_Mapping and determines if the given voxel location is withing the bounds of the specified element.

```
function [inElement] = CheckInElement(el_type, Nodes, Elements, x, y,
z,voxel_size)
%Check if voxel is within the element

inElement = false;

% For tetrahedral elements
if strcmp(el_type, 'Tet')

    %Calculate the directions between nodes
    r12 = Nodes(Elements(2),:) - Nodes(Elements(1),:);
    r13 = Nodes(Elements(3),:) - Nodes(Elements(1),:);
    r14 = Nodes(Elements(4),:) - Nodes(Elements(1),:);
    r42 = Nodes(Elements(4),:) - Nodes(Elements(2),:);
    r43 = Nodes(Elements(4),:) - Nodes(Elements(3),:);

    %Calculate element face normals (positive to inside)
    n1 = cross(r12, r13);
    n2 = cross(r14, r12);
    n3 = cross(r13, r14);
    n4 = cross(r43, r42);

    %Normalize n vectors
    n1 = n1./norm(n1);
    n2 = n2./norm(n2);
    n3 = n3./norm(n3);
    n4 = n4./norm(n4);

    %Calculate direction to the voxel from required nodes
    vox_loc = [x y z];
    r1v = vox_loc - Nodes(Elements(1),:);
    r4v = vox_loc - Nodes(Elements(4),:);

    dot1 = dot(n1, r1v);
```

```
        dot2 = dot(n2, r1v);
        dot3 = dot(n3, r1v);
        dot4 = dot(n4, r4v);

        face1check = false;
        face2check = false;
        face3check = false;
        face4check = false;

        % Determine if the voxel is to the positive side of the element face
        if dot1 > 0
           face1check = true;
        end
        if dot2 > 0
           face2check = true;
        end
        if dot3 > 0
           face3check = true;
        end
        if dot4 > 0
           face4check = true;
        end

        % Determine if the voxel is withing the element
        if face1check && face2check && face3check && face4check
            inElement = true;
        end

    % For hexahedral elements
    elseif strcmp(el_type, 'Hex')

        %Calculate the directions between nodes
        r12 = Nodes(Elements(2),:) - Nodes(Elements(1),:);
        r14 = Nodes(Elements(4),:) - Nodes(Elements(1),:);
        r15 = Nodes(Elements(5),:) - Nodes(Elements(1),:);
        r78 = Nodes(Elements(8),:) - Nodes(Elements(7),:);
        r73 = Nodes(Elements(3),:) - Nodes(Elements(7),:);
        r76 = Nodes(Elements(6),:) - Nodes(Elements(7),:);

        %Calculate element face normals
        n1 = cross(r12, r14);
        n2 = cross(r15, r12);
        n3 = cross(r14, r15);
        n4 = cross(r76, r78);
        n5 = cross(r73, r76);
        n6 = cross(r78, r73);

        %Normalize n vectors
        n1 = n1./norm(n1);
        n2 = n2./norm(n2);
        n3 = n3./norm(n3);
        n4 = n4./norm(n4);
        n5 = n5./norm(n5);
        n6 = n6./norm(n6);
```

```
    %Calculate direction to the voxel from required nodes
    vox_loc = [x y z];
    r1v = vox_loc - Nodes(Elements(1),:);
    r7v = vox_loc - Nodes(Elements(7),:);

    dot1 = dot(n1, r1v);
    dot2 = dot(n2, r1v);
    dot3 = dot(n3, r1v);
    dot4 = dot(n4, r7v);
    dot5 = dot(n5, r7v);
    dot6 = dot(n6, r7v);

    face1check = false;
    face2check = false;
    face3check = false;
    face4check = false;
    face5check = false;
    face6check = false;

    % Determine if the voxel is to the positive side of the element face
    if dot1 > 0
       face1check = true;
    end
    if dot2 > 0
       face2check = true;
    end
    if dot3 > 0
       face3check = true;
    end
    if dot4 > 0
       face4check = true;
    end
    if dot5 > 0
       face5check = true;
    end
    if dot6 > 0
       face6check = true;
    end

    % Determine if the voxel is withing the element
    if face1check && face2check && face3check && face4check &&
face5check && face6check
        inElement = true;
    end

else
    assert(0,'Invalid Element Type');
end

end
```

List of Symbols, Abbreviations, and Acronyms

ARL	Army Research Laboratory
BHBT	behind-helmet blunt trauma
BVF	bone volume fraction
CT	computed tomography
DEVCOM	US Army Combat Capabilities Development Command
FE	finite element
FEA	finite element analysis

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
24 (PDF)	DEVCOM ARL FCDD RLW B C HOPPEL P GILLICH FCDD RLW T R FRANCART FCDD RLW TB S ALEXANDER R BANTON T BAUMER A BROWN B FAGAN A GOERTZ A GUNNARSSON C HAMPTON R KARGUS D KRAYTERMAN M KLEINBERGER E MATHEIS J MCDONALD P MCKEE K RAFAELS S SATAPATHY M TEGTMEYER C WEAVER T WEERASOORIYA S WOZNIAK T ZHANG