TriGA v1.0 README

Luke Engvall

February 16, 2015

1 Introduction

TriGA (Triangular IGA) is a toolkit for extending Isogeometric Analysis to unstructured grids of higher order triangles and (in the future) tetrahedra. It consists of three main parts:

TriGA: TriGA contains the finite element subroutines for using rational beziers over cubic triangular elements. It also contains a simple 2D steady heat equation solver for demonstrating that the meshes are analysis suitable.
xmesh: Short for Exact Mesh, xmesh generates higher order triangular meshes based off of NURBS curves.
Mesh2D: Triga's meshing capabilities rely heavily on MESH2D, an excellent 2-D triangular mesh generator by Darren Engwirda. It can be found **here**, and is also included in it's entirety in the TriGA repository.

TriGA is a work in progress. Please send any unmeshable geometries or poorly behaved simulations to Luke Engvall at: **engvall@colorado.edu**. Additionally, any comments or suggestions for improvements are always welcome.

2 Getting Started

The only things needed to get started with TriGA should be the TriGA download and a recent version of MATLAB. For users interested in seeing what TriGA can do, it's recommended to start with running the scripts

xmeshdemo and trigademo. After that, section 2.3 has a short tutorial on creating a geometry file, meshing it, and running a simulation.

2.1 xmeshdemo

xmeshdemo provides a short introduction to what xmesh can do. It steps through several files and gives the user the option of seeing meshes for the following geometries. Each geometry showcases a particular ability of xmesh. When running xmesh, it should be noted that some geometries do take a couple of seconds to be meshed.

Geometry	Description
Unit Circle	Demonstrates the ability to exactly represent curved geometries with relatively few elements. Also shows xmesh's refinement capabilites.
Plate with hole	Geometry used in trigademo.
Nozzle	Demonstrates the ability to mesh long skinny geometry features.
Flange	Demonstrates the ability to mesh a complex connected face made up by multiple curves.
Airfoil	Shows the ability to mesh complicated curves.

2.2 trigademo

trigademo shows how meshes created by xmesh can be used for analysis. heat2d solves the steady 2D heat equation over two geometries and uses the method of manufactured solutions to analyze the convergence in each case. Note that triagedemo can take 5-10 minutes to run depending on the machine.

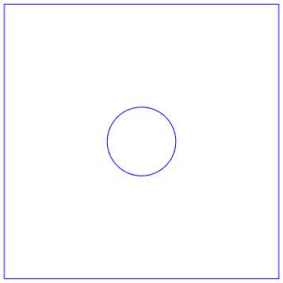
2.3 Tutorial

This tutorial will step through the workflow of creating a geometry, meshing, running a simulation and visualizing the results.

Create a Spline Geometry

The first step of the workflow is to define a geometry. Unfortunately at this point there is no GUI for drawing geometries. We hope to add this functionality at some point in the near future, but for now we are stuck creating our geometries by hand. This is done by specifying any number of NURBS curves and writing a spline file (see section 3.3 for the .spline file format specification). For this tutorial, we'll consider the case of a plate and a hole (Figure 1).

Figure 1: Plate with a hole.



TriGA operates mainly on gambit .neu files, and the argument to most functions is the filename of the file that you are working with. Start by creating a new directory in TriGA called tutorial. All the files from this example will be written to this directory. Open a plain text file, and save it to tutorial as 'platetutorial.spline'. This section will guide you through creating a .spline file that defines the problem geometry.

A plate with a hole will be defined by two NURBS Curves, one defining the exterior of the plate, and one defining the hole. It doesn't matter what order we define the curves in, we'll just start with the hole. A circle is exactly defined by a quadratic NURBS curve with the following control points and knot vector:

$$P = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & \sqrt{2}/2 \\ 0 & 1 & 1 \\ -1 & 1 & \sqrt{2}/2 \\ -1 & 0 & 1 \\ -1 & -1 & \sqrt{2}/2 \\ 0 & -1 & 1 \\ 1 & -1 & \sqrt{2}/2 \\ 1 & 0 & 1 \end{bmatrix}$$

$$KV = [0\ \ 0\ \ 0\ \ 0.25\ \ 0.25\ \ 0.5\ \ 0.5\ \ 0.75\ \ 0.75\ \ 1\ \ 1\ \ 1]$$

Figures 2 and 3 show the control points and the knot locations for the two curves that make up the mesh.

Figure 2: Curve Control Nets

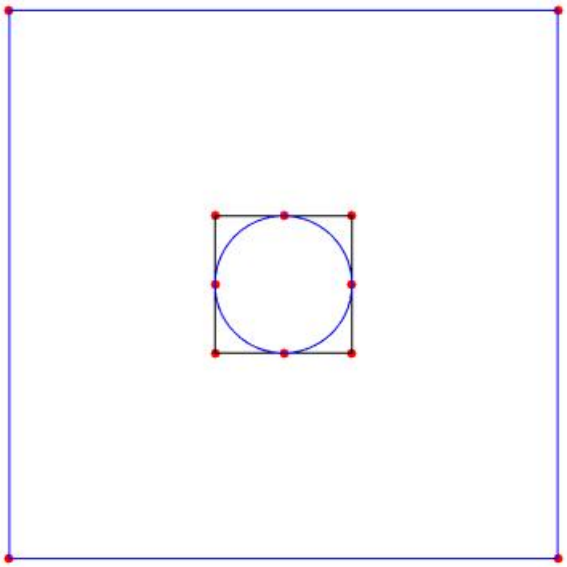
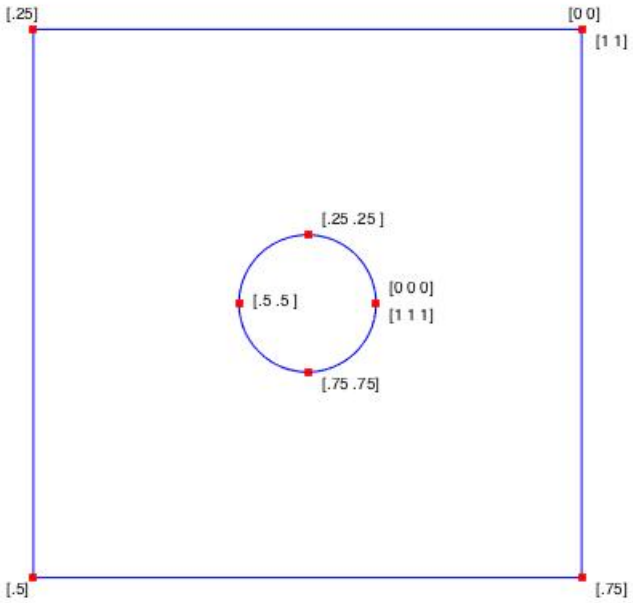


Figure 3: Curve Knot Locations.

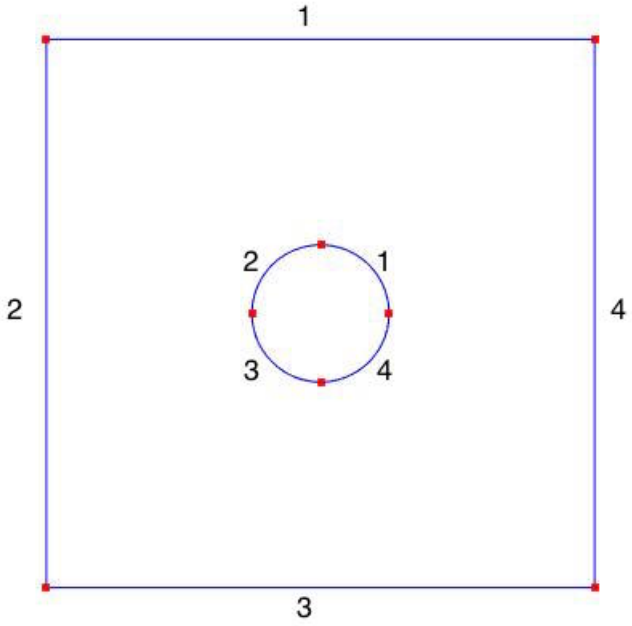


We also need to flag what type of boundaries each knot span of the curve lie on. There are four knot spans making up the circle, and we'll assume that they all lie in the same boundary set. Thus the boundary flag array is:

$$B = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}$$

Figure 4 shows the knot spans of each of the curves.

Figure 4: Curve Knot Spans.



The control points and knot vector for the curve that represents the outer edge of the plate are:

$$P = \begin{bmatrix} 4 & 4 & 1 \\ -4 & 4 & 1 \\ -4 & -4 & 1 \\ 4 & -4 & 1 \\ 4 & 4 & 1 \end{bmatrix}$$

$$KV = \begin{bmatrix} 0 & 0 & 0.25 & 0.5 & 0.75 & 1 & 1 \end{bmatrix}$$

Lets say that we want to apply one set of boundary conditions to the left and right sides of the plate, and a different set of boundary conditions to the top and bottom. Our boundary flag array is

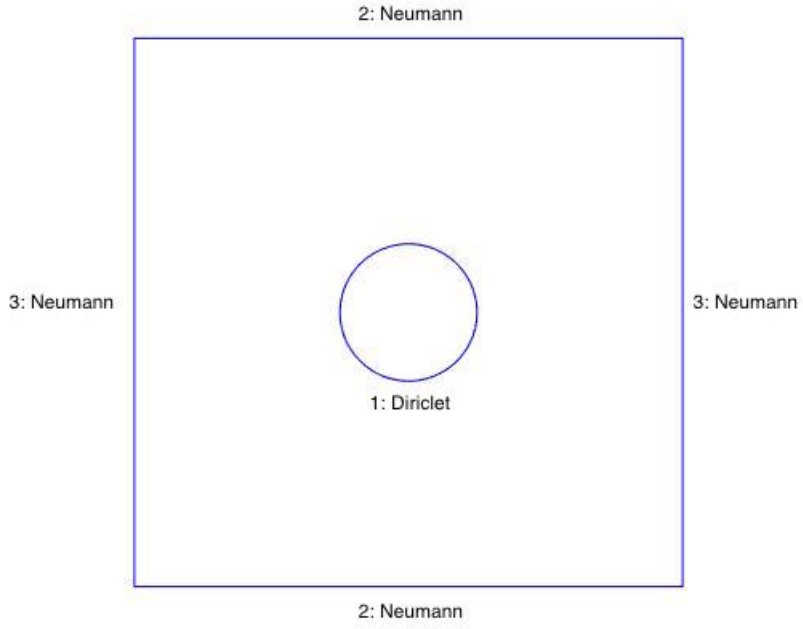
$$B = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 2 \\ 4 & 3 \end{bmatrix}$$

Finally we need to specify what type of boundary conditions exist on each set. Right now heat2d supports Diriclet and Neumann boundary conditions. As per .neu file format standards, these are specified by 6 and 7, respectively. Thus, if we want Neumann boundary conditions on the plate exterior and Dirclet on the inner circle we specify the BC matrix:

$$BC = \begin{bmatrix} 1 & 6 \\ 2 & 7 \\ 3 & 7 \end{bmatrix}$$

Figure 5 shows boundary condition sets.

Figure 5: Boundary Condition Sets.



Thus, our final .spline file looks like:

```
HEADER
NCURVES NSD
2       2
NCP     LKV     p
9       12      2
CONTROL POINTS
     1   0  1
     1   1  0.707106781186548
     0   1  1
    -1   1  0.707106781186548
    -1   0  1
    -1  -1  0.707106781186548
     0  -1  1
     1  -1  0.707106781186548
     1   0  1
KNOT VECTOR
0 0 0 .25 .25 .5 .5 .75 .75 1 1 1
BOUNDARY FLAGS
1 1
2 1
3 1
4 1
NCP  LKV     p
5    7       1
CONTROL POINTS
     4   4  1
    -4   4  1
    -4  -4  1
     4  -4  1
     4   4  1
KNOT VECTOR
0 0 .25 .5 .75 1 1
BOUNDARY FLAGS
1 2
2 3
3 2
4 3
BOUNDARY CONDITIONS
NBOUNDS
3
1   6
2   6
3   6
```

Create a mesh
Once a .spline file has been created, mesh the geometry using xmesh. From the TriGA directory, run:

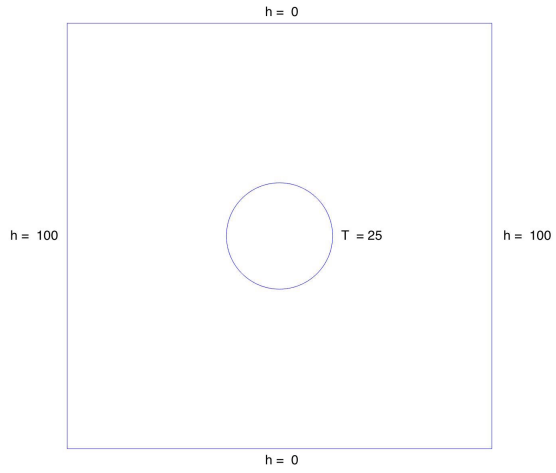
```
addpath xmesh
xmesh2d('./tutorials/platetutorial');
```

This creates a gambit neutral file in TriGA/tutorials called platetutorial.neu. Visualize the resulting mesh with:

```
showMesh('./tutorials/platetutorial');
```

Run a simulation
Consider the problem shown below, where we have no internal heat generation and unit conductivity, $\kappa = 1$.

Figure 6: Example Problem



We run this with:

```
heatGen =  @(x,y)0; % Internal heat Generation
kappa = 1; % Conductivity
BC = [25, 0, 100]; % Boundary Condition Values
D   = eye(2)*kappa; % Conductivity Matrix
heat2d('./tutorials/platetutorial',heatGen,BC,D);
```

heat2d will automatically write out the temperature results to platetutorial.neu. Visualize the results with:

```
showResults('./tutorials/platetutorial')
```

The default in showResults is to visualize the resulting temperature field with a very coarse heat map. To increase the resolution of this map, the user can input an h value to showMesh (The smaller the value of h, the finer the resolution). For example:

```
showResults('./tutorials/platetutorial',.1)
```

Refine the Mesh
If you need a finer mesh, xmesh can do uniform mesh refinement.

```
refineMesh('./tutorials/platetutorial')
```

This creates a new file platetutorialref.neu. Visualize the new mesh just like before:

```
showMesh('./tutorial/platetutorialref');
```

3 Documentation

The documentation in this section serves as a jumping off point for any users that want to understand more about how TriGA works or want to modify parts of the code for their own use. The documentation in this section is fairly high level and mainly serves to give an overview of the purpose of each function in the TriGA code. The code is also documented fairly well internally, and the comment header in each function gives documentatian on the usage of the function. Again, users should feel welcome to contact Luke Engvall with any comments or questions they have about the code and/or documentation.

3.1 TriGA

The TriGA directory contains functions for performing FEA on simple heat conduction problems over higher order triangular meshes. The table below provides an overview of the main functions and their purpose.

Name	Description
heat2d.m	Finite element solver that uses Galerkin's method for solving simple 2-D steady heat conduction problems
showResults.m	Visualizes the results from heat2d as a heat map over the domain
calcNorm.m	Calculates the L2 and H1 norms of a finite element solution that has an analytic solution
tri10.m	Finite element subroutine for the 10 node triangular element
tri10b.m	Finite element subroutine for the edges of the 10 node triangular element
quadData	Stores the location and weight data for Gaussian Quadrature

3.2 xmesh

xmesh is the core meshing routine called in TriGA. The xmesh directory contains the following functions that perform meshing.

Name	Description
xmesh2d	This is the main meshing subroutine. It reads in a .spline file, and outputs a mesh to a .neu file.
NURBS2Poly	Converts the NURBS curves to polygons to be fed to xmeshfaces.
xmeshface	The core meshing routine for generating the triangular mesh. Takes the polygonal approximation of the geometry from NURBS2Poly and generates a linear mesh.
meshBoundary	Performs Bezier extraction on the original NURBS curves to obtain the local control points for the boundary edges of the elements that lie on a global boundary. These are then used to modify the mesh from xmeshfaces so that it is geometrically exact.
evalNURBS	evaluates a NURBS curve given by P and KV at parameter values [xi]
gen_arrays	takes a 1xnel cell array of local control points and generates the global NODE and IEN arrays
refineMesh	Performs uniform subdivision of the triangles in a mesh to create a refined mesh.
showMesh	Displays the mesh generated by xmesh.

3.3 Input/Output Files

.spline Files xmesh2d operates on any number of closed NURBS curves and attempts to mesh one simply connected face defined by the input curves. If an input geometry contains a geometry that has multiple faces, xmesh2d may fail. Additionally, the NURBS curves must be closed. Boundary conditions The file format goes as follows:

```
HEADER
NCURVES           NSD
 <# of curves>    <# spatial dimensions>
 NCP                      LKV                    p
<# of control points 1>   <knot vector length 1>  <curve polynomial degree 1>
CONTROL POINTS
<NCP x 3 array containing the control points for curve 1>
KNOT VECTOR
<LKV x1 row vector containing the knot vector for curve 1>
BOUNDARY FLAGS
<# knot spans x 2 array containing the knot span number and its boundary set>

                                                        .
                                                        .
                                                        .


 NCP                       LKV                    p
<# of control points N>   <knot vector length N>  <curve polynomial degree N>
CONTROL POINTS
<NCP x 3 array containing the control points for curve N>
KNOT VECTOR
<LKV x1 row vector containing the knot vector for curve N>
BOUNDARY FLAGS
<# knot spans x 2 array containing the knot span number and its boundary set>
 BOUNDARY CONDITIONS
 NBOUNDS
 <# boundary condition sets>
 <NBOUNDSx2 matrix containing BC set and its corresponding boundary condition>
```

.neu Files
Once a mesh has been created from a .spline file, TriGA uses the gambit neutral file format for all its input and output. A writeup on the file format can be found **here**.