Rhino 3D to Abaqus: A T-Spline Based Isogeometric Analysis Software Framework

Yicong Lai, Lei Liu, Yongjie Jessica Zhang, Joshua Chen, Eugene Fang, and Jim Lua

Abstract This chapter presents a novel software framework for T-spline based isogeometric analysis (IGA), interfacing between computer aided design (CAD) and finite element analysis environment for "design-through-analysis" concept. The framework is built on *Rhinoceros* 3D (Rhino) and *SIMULIA* Abaqus (Abaqus). This framework has three primary steps: creating CAD surface in Rhino with T-spline Plugin, converting surface T-spline representations into volumetric T-spline, and performing analysis with T-spline models in Abaqus through its user element subroutine. Both 2D and 3D examples are provided in the end to demonstrate our T-spline based IGA software framework.

1 Introduction

Non-uniform rational B-spline (NURBS) is the standard mathematical representation for geometry in most commercial computer aided design (CAD) software. Finite element analysis (FEA) has been widely used in most commercial computer aided engineering (CAE) software. Geometry designed in CAD software must undergo an approximating discretization process known as meshing, so that it can be used for analysis in CAE software. A seminal effort to integrate CAD and CAE, termed isogeometric analysis (IGA) [1], gained great attention as a new "design-through-analysis" methodology. It enabled direct analysis of the designed geometry without meshing.

T-spline [2] was introduced as a generalization of NURBS allowing local refinement. The introduction of the *Autodesk* T-Splines Plugin for Rhino in 2004 (http://www.tsplines.com/) provides designers the tool to create and manipulate T-spline geometries. T-spline basis functions have been incorporated into IGA [3], further generalizing the "design-through-analysis" framework. Analysis-suitable

Y. Lai • L. Liu • Y.J. Zhang (✉) • J. Chen
Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail: jessicaz@andrew.cmu.edu

E. Fang • J. Lua
Global Engineering and Materials, Inc, Princeton, NJ 08540, USA

Y. Bazilevs, K. Takizawa (eds.), *Advances in Computational Fluid-Structure Interaction and Flow Simulation*, Modeling and Simulation in Science, Engineering and Technology, DOI 10.1007/978-3-319-40827-9_21

T-splines in IGA were characterized [4], and various methods were developed to construct volumetric T-splines for IGA [5–9]. Open-source NURBS based IGA implementations [10, 11] were introduced, but they are primarily used in research. Recent efforts have increased accessibility of T-spline based IGA to industry. Data structures for T-spline based IGA were developed [12, 13]. Moreover, NURBS based IGA was implemented in Abaqus [14]. Although the improvements of IGA are vast, it is still at the beginning stage in industry. So far, T-spline based IGA implementations have not yet been available in many commercial software like Abaqus.

In this chapter, we present a novel CAD-CAE software framework for T-spline based IGA. Since Rhino and Abaqus are built for engineering design and analysis specifically, here we develop a T-spline based IGA software framework upon them. Users can benefit from the strengths of both software. This framework allows:

- Boundary value problem (BVP) specification on T-spline geometry;
- Volumetric T-spline construction from surface representation;
- Efficient and compact trivariate T-spline data structure; and
- Abaqus T-spline IGA user element subroutine based on Bézier extraction.

The reminder of this chapter is organized as follows. Section 1 overviews the framework and the pipeline. Section 3 describes surface T-spline geometry creation and BVP specification. Section 4 presents T-spline data structure. Section 5 explains how volumetric T-splines are constructed from surface T-splines. Section 6 presents our T-spline Abaqus user subroutine. Finally, section 7 shows numerical results and draws conclusions.

2 Software Framework and Pipeline Overview

To integrate Rhino with Abaqus, the framework uses Rhino with the T-spline plugin to create and manipulate T-spline geometries, and uses Abaqus for analysis through its user element subroutines (UEL/UELMAT). The framework incorporates two self-developed plugins as shown in Fig. 1, the Rhino plugin in the grey and blue blocks and the Abaqus plugin in the red block. This framework requires the following software environment:

- *Rhinoceros* 3D, Version 5 or newer;
- *Autodesk* T-splines Plugin for Rhino, Version 3.4 or newer;
- *SIMULIA* Abaqus Unified FEA, Version 6.0 or newer; and
- *Intel* FORTRAN, Version 11 or newer.

As shown in Fig. 1, surface geometry is created in Rhino first. With the Rhino plugin, BVP problem is specified and the geometry is pre-processed, ending up with Abaqus input files. Depending on whether a 2D or 3D geometry is needed, users can choose whether to construct volumetric T-splines. For a 2D case, a Rhino .iga file is directly converted into Abaqus .inp file and .bezier file. For a 3D case, .TSM files

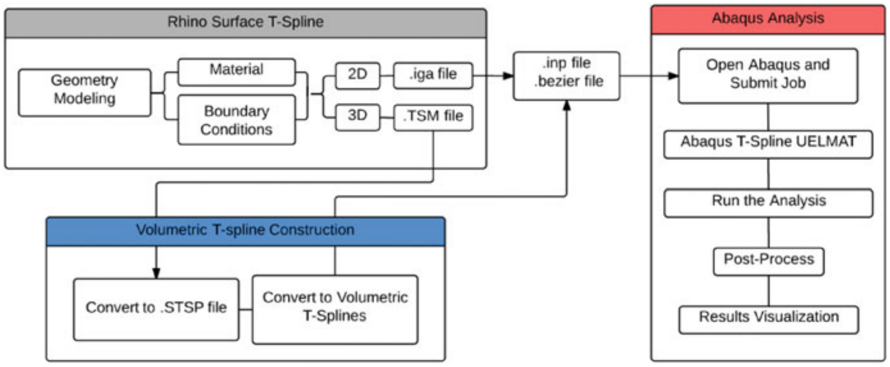

Fig. 1 Rhino 3D to Abaqus Software Pipeline Overview.

are saved and converted to our .STSP file for volumetric T-spline construction, and then Abaqus .inp file and .bezier file are generated for analysis. When the analysis is completed through the Abaqus plugin, a post-processing function is called in the Abaqus plugin to generate the .odb file for visualization. Based on the results, the user can go back to Rhino to refine or modify the geometry.

3 Geometry Generation and BVP Specification

Our software framework supports both surface and volumetric T-splines. Since Rhino only supports surface modeling, surface T-splines must undergo a conversion process to become volumetric T-splines. Currently, we support two types of volumetric T-spline geometries. The first type is a 3D geometry created by sweeping a surface T-spline patch, and the second type is the geometry with genus-zero cube topology which can be created using the conformal parametric mapping method[7]. Fig. 2 shows three T-spline surface models designed in Rhino. (a) and (b) show 2D open surfaces which can be used in analysis directly or converted to volumetric T-spline via sweeping. (c) shows a structure with genus-zero topology and six patches, which can be converted into volumetric T-spline via parametric mapping.

In Rhino, all the boundary conditions are applied on relevant control points. In the pre-processing step, the Rhino plugin extracts the information of these control points and writes to Abaqus .inp file. To apply boundary conditions, we use the T-Spline plugin selection node sets [15] to choose desired control points. The platform is currently limited to specifying Dirichlet boundary conditions, but extending to other boundary conditions is straightforward. For the material properties, simple isotropic material properties are supported currently. The users can specify the Young's Modulus E and the Poisson's Ratio ν.

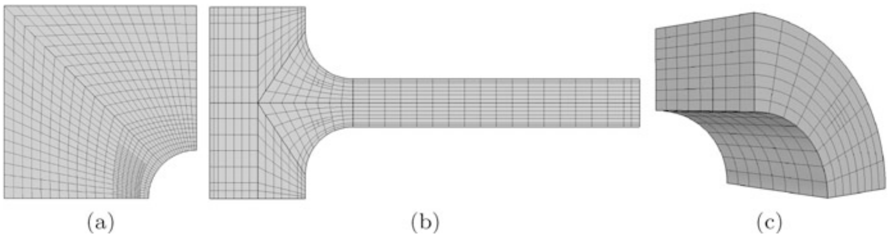

Fig. 2 (a) A 2D quarter of a plate with a hole; (b) a 2D notch structure prepared for sweeping; and (c) a quarter cylinder with genus-zero cube topology.

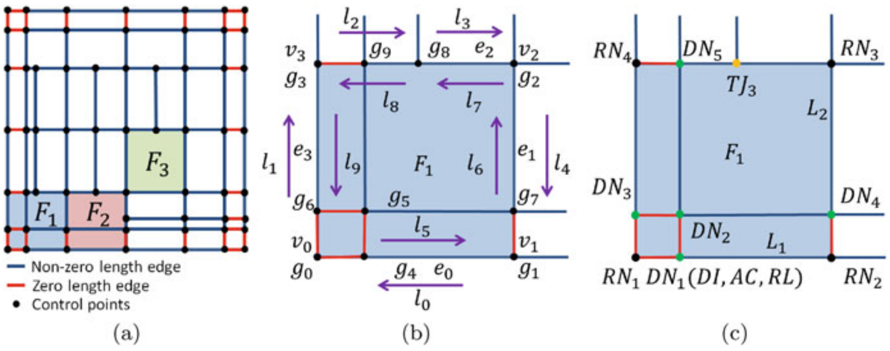
F_1
F_2
F_3
Non-zero length edge
Zero length edge
Control points
(a)
l_2 l_3 v_3 g_9 g_8 e_2 v_2 g_3 g_2 l_8 l_7 l_1 e_3 l_9 F_1 l_6 e_1 l_4 g_6 g_5 g_7 l_5 v_0 v_1 g_0 g_4 e_0 g_1 l_0
(b)
RN_4 DN_5 RN_3 TJ_3 L_2 F_1 DN_3 DN_4 DN_2 L_1 RN_1 $DN_1(DI, AC, RL)$ RN_2
(c)

Fig. 3 (a) A T-mesh with three representative faces/elements, the corner face F_1, the edge face F_2, and the interior face F_3; (b) a corner face in *.TSM* data format. Abbreviations are: *l-link, v-vertex, g-grip, F-face, e-edge*; and (c) a corner element in *.STSP* data format. Abbreviations are: *AC-Associate Corner, DI-Duplicated Node Index, DN-Duplicated Node, F- Element, L-Parametric Length, RL-Relative Location, RN-Regular Node, TJ-T-Junction.*

4 T-Spline Data Structure

In this section, we first explain the commercial *.TSM* data structure used in Rhino. Then, we introduce our *.STSP/.VTSP* data structure and explain the conversion between them.

4.1 TSM Data Format

The *.TSM* file originated from "half edge" data structure[13] which contains parametric and physical information of a T-spline control mesh (T-mesh). Fig. 3(a) shows three representative faces or elements of a T-mesh. F_1 is a corner face, F_2 is an edge face, and F_3 is an interior face. Fig. 3(b) takes a corner face as an example to show detailed components in a *.TSM* face, including links, vertices, grips, faces, and edges.

Link {previous, next, opposite, vertex, face, edge, type}. *Links* tie the topology of a surface together. Every edge, face, or vertex refers to a link to determine its location. The first 3 values indicate the id of the previous, next, and opposite links. For example, in Fig. 3(b), the previous, next, and opposite links of l_5 are l_9, l_6, and l_0, respectively. Each link has an associated vertex, face, and edge which are the three values following the opposite link. For example, l_5 originates from v_0 and belongs to F_1, associating to e_0. The type indicates what kind of node the link starts from, either a regular node (Type 0), a T-junction (Type 1), or an L-junction (Type 2). For example, the type of l_5 is 0 since it starts from a regular node while the type of l_8 is 1 since it starts from a T-junction. Note that the T-junction can break one edge into multiple links. Here, the T-junction g_8 breaks the edge e_2 into l_7 and l_8.

Vertex {link, direction}. A *vertex* is the origin of a link. The direction that the link points away from the vertex is stored as one of the North (N), South (S), East (E), and West (W). For example, v_0, v_1, v_2, and v_3 are four vertices of the corner face F_1 in Fig. 3(b).

Grip {x, y, z, weight}. A *grip*, equivalent to a control point, is defined by its x, y, z coordinates and weight. A grip may be parent of one or more other grips. This kind of grips is also known as *compound grips*. For example, in Fig. 3(b), g_0 is a compound grip which is the parent of g_4, g_5, and g_6.

Face {start link, flag}. A *face* is used to represent an element in the T-mesh, and it is defined by a start link and a flag. The connectivity of a face is given by the counter-clockwise cycle of links at its borders, starting from its start link. The *flag* of a face stores various properties of this face like if the face is hidden or not.

Edge {link, interval}. An *edge* connects vertices in the T-mesh. It is defined by a link and an interval. Its link is one of the two links running along it, while the "interval" is its parametric length.

4.2 STSP/VTSP Data Format and Conversion

Let us first define four types of control points for our T-spline data structure, including the regular node, the duplicated node, the T-junction node, and the extraordinary node. A *regular node* is a control point with valence 4. A *duplicated node* is a duplication of its neighbor, see the green dots in Fig. 3(c). A *T-junction* is a control point that is analogical to the hanging nodes in classical finite elements, see the orange dot. For T-junction nodes, we record the edge it locates as well as its index. An *extraordinary node* is an interior control point with valence other than 4, and not a T-junction. In our data structure, regular node, T-junctions, and extraordinary nodes are used to describe the connectivity of elements. Regular nodes are often the parents of duplicated nodes. A corner regular node is the parent of three duplicated nodes while a boundary regular node is the parent of one duplicated node. For example, in Fig. 3(c), RN_1 is the parent of DN_1, DN_2, and DN_3, and RN_2 is the parent of DN_4. According to the algorithm introduced in [16], extraordinary nodes cannot be the parents of duplicated nodes.

The *.STSP/.VTSP* data structure is efficient in storing surface and volumetric T-spline information. This data structure is designed especially for the convenience of extracting knot vectors. It is compact, concise, and easy to interpret and fit into the commercial software. Our data format contains two types of data, the control point data and the element data. The former contains the basic information of T-spline control points, and the latter contains connectivity, parametric length, T-junctions, and duplicated nodes in the element.

1. **Control Point {x, y, z, weight, type}**. A *control point* is defined by its x, y, z coordinates, weight, and type. For the type, we use 0 to represent the regular node, 1 for duplicated node, 2 for T-junction node, and 3 for extraordinary node.
2. **Element {Corner ID$_{1-4}$, Parametric Length$_{1-2}$, T-junction ID$_{1-4}$, Number of Duplicated Nodes, (Duplicated Node ID, Associated Corner, Relative Location)$_{1-totalnumber}$}**. The first four values are the indices of the four corners of this element. The next two values specify the parametric length of the edges. The following four values are the T-junction indices for each edge. If there is a T-junction on one edge, we record its index at the corresponding position, otherwise we put "-1" there. For example, in Fig. 3(c), there is a T-junction on the third edge. Assuming its index is TJ_3, we record the T-junctions as "-1 -1 TJ_3 -1". Following the T-junctions, the next value indicates the total number of duplicated nodes, which is 5 for a corner element and 2 for an edge element. The remaining values specify the relative location of the duplicated nodes with respect to the associated corner of this element. A duplicated node has three consecutive values, the global index, the corner in the T-mesh to which it belongs, and the relative location of this node to the corner.

.TSM to .STSP Conversion. To obtain a logical order of the duplicated nodes, we introduce an *STSP* plane to help extract the T-spline information from the *.TSM* patch in Rhino. Fig. 4(a) shows the *STSP* plane with the center point labeled as 8. This plane is a designed pattern to store the relative location of duplicated nodes with respect to their associated corners. The edge interval length between a duplicated node and its associated corner is zero. This information is further used in extracting knot vectors. For open surface topologically equivalent to a unit square, there are four types of corner elements. Thus, the *STSP* plane has four subsquares, each of which can be applied to one certain type of corner element or edge element to store the duplicated nodes information. For example, the green square in Fig. 4(a) can be applied to the corner element in Fig. 4(b) while the orange square can be applied to the corner element in Fig. 4(c). In Fig. 4(b), there are three nodes that have duplicated nodes, g_0, g_1, and g_3. Since g_0 has three duplicated nodes g_4, g_5, and g_6, we first put g_0 at the center of the *STSP* plane which is position 8. In this way, g_4, g_5, and g_6, are located at positions 3, 4, and 5 in the *STSP* plane, respectively. This indicates g_4 is associated with the corner 0 at the relative location of 3 in the *STSP* plane. Thus, we record the relative location of this duplicated node as "4 0 3". Similarly for g_5 and g_6, we record them as "5 0 4" and "6 0 5," respectively. We can apply the same rule to the remaining g_1 and g_3 with only one duplicated node each.

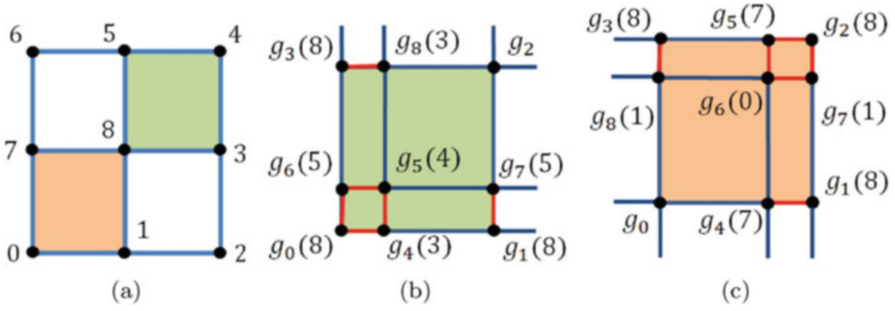

Fig. 4 (a) The *STSP* plane; (b) a corner element applied to the green square in (a); and (c) another corner element applied to the orange square in (a). The red edges have zero length.

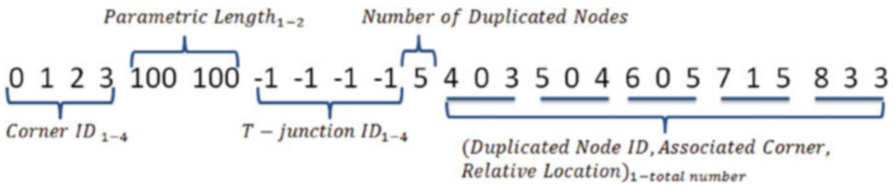
Parametric Length$_{1-2}$
Number of Duplicated Nodes
0 1 2 3 100 100 -1 -1 -1 -1 5 4 0 3 5 0 4 6 0 5 7 1 5 8 3 3
Corner ID$_{1-4}$
T − junction ID$_{1-4}$
(Duplicated Node ID, Associated Corner, Relative Location)$_{1-total\ number}$

Fig. 5 Sample element data line for the T-mesh element in Fig. 4(a).

We put g_1 and g_3 at the center of STSP plane and g_7 and g_8 can be recorded as "7 1 5" and "8 3 3," respectively. Similarly, Fig. 4(c) shows the relative location of all duplicated nodes in another corner element.

To convert the *.TSM* data to the *.STSP* data, we first follow the face link loop to find all the vertices and grips of this element. The four grips at the corners determine the connectivity of this element. They are also the first four values in our data format. With the links, we extract the edge interval associated with the links and record them as parametric length. Then we loop through each grip to order the information of duplicated nodes. We count the total number of duplicated nodes and then record the relative position using the *STSP* plane. Suppose this element has an edge length of 100 in both parametric directions and has no T-junction, its representation " 0 1 2 3 100 100 -1 -1 -1 -1 5 4 0 3 5 0 4 6 0 5 7 1 5 8 3 3 " is shown in Fig. 5.

After *.TSM* to *.STSP* data format conversion, we obtain all the information of the T-mesh. If only a 2D geometry is required for analysis, we directly extract the knot vectors and calculate the T-spline surface. If there are extraordinary nodes in the T-mesh. We use the interval duplication method [16] to deal with it. If 3D geometry is needed for analysis, we convert *.TSM* file to *.VTSP* file.

.TSM to .VTSP Conversion. We extend the designed pattern from *STSP* plane to *VTSP* cube in 3D case. Fig. 6(a) shows the *VTSP* cube and its labels. In 3D, we put each corner of this element to the center of the cube and for each duplicated node we record its relative location in the cube using the same way as in 2D case. As the example shown in Fig. 6(b), g_6 is associated with corner 1 in the element. After putting corner 1 to the center of the cube which is position 17, g_6

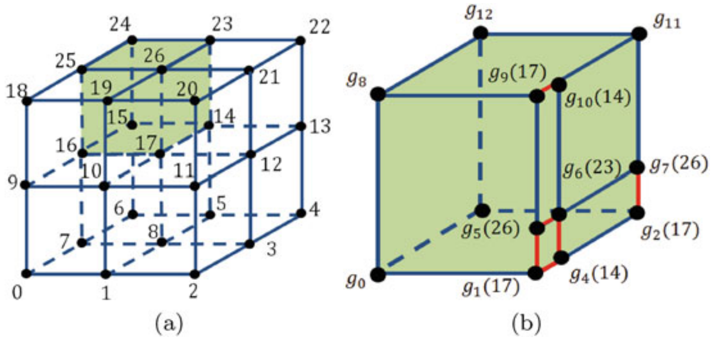

Fig. 6 (a) The *VTSP* cube; and (b) a corner element applied to the green cube in (a). The red edges have zero length.

is located at position 23 automatically. Thus we record the duplicated node g_6 as "6 1 23". Other duplicated nodes can be recorded in the same way. With everything recorded, the information is then used in knot vector extraction and volumetric T-spline construction.

5 Volumetric T-Spline Construction

After data structure conversion, we perform volumetric T-spline construction, using the parametric mapping algorithm [7]. The algorithm first identifies the eight corner nodes on the boundary surface and then maps the surface patch to a unit cube. The interior domain is adaptively subdivided using octree subdivision until each face or edge contains at most one T-junction. Two boundary layers are inserted between the input boundary and subdivision results using the pillowing technique to preserve the input boundary representation. Then, templates are applied around extraordinary nodes to build a valid T-mesh. Finally, the knot vectors are extracted and volumetric T-splines are constructed.

In our approach, instead of inserting two boundary layers, we pillow the 6 patches one by one, resulting in one boundary layer. The reason is that for CAD model with sharp edges, the original method may generate elements with bad quality and introduce unnecessary extraordinary nodes. Our modified method improves the quality and no new extraordinary node is introduced. We use a 16×16 cube to show the difference between these two methods. Fig. 7(a) shows the original input boundary representation. Fig. 7(b) shows the constructed T-splines in Bézier representations using the original algorithm, and Fig. 7(c) shows the results in Bézier representations using the modified algorithm. Note that in the modified method, the one layer inserted is used as the zero-length layer, so they are not extracted for Bézier representations. Finally, the Bézier operators [12] are calculated, which are used for IGA in Abaqus.

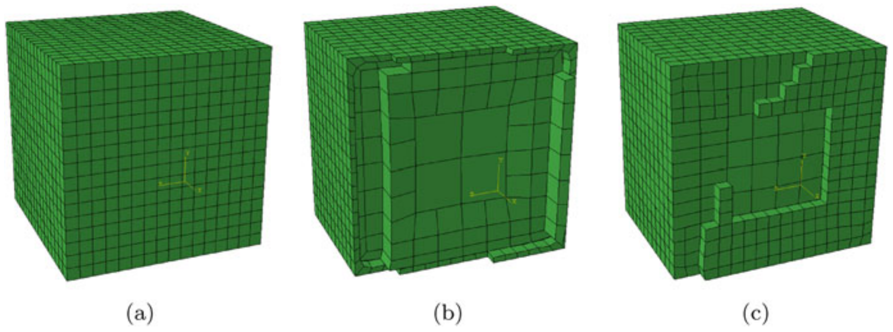

Fig. 7 Comparison between the original method and the modified method. (a) The boundary representation of the cube; (b) the constructed volumetric T-splines using the original method [7]; and (c) the T-splines using our modified method.

6 Abaqus IGA and Visualization

With the surface or constructed volumetric T-spline, the plugin generates an Abaqus .inp file and a .bezier file. The Abaqus .inp file contains T-spline control point coordinates, element connectivities, and the BVP information. The .bezier file is our self-defined file format containing information of T-spline control point weight and Bézier operators. Both files are read into the UELMAT through the User EXTERNAL DataBase (UEXTERNALDB) in Abaqus. The T-spline based UELMAT uses Bézier operator to compute the T-spline basis functions, we then build the stiffness matrix and force vector for the analysis[12].

Abaqus does not support the visualization of user-defined elements. So post-processing is necessary to project the results to a finite element mesh for visualization. In addition to Abaqus, other software like ParaView can also be used for visualization. For linear elastic problems, the displacement can be extracted from the .fil file generated by Abaqus. Then, the nodal values of each element are computed based on the analysis results.

7 Numerical Results and Conclusion

Using the three geometries in Fig. 2, linear elastic problems with a Young's modulus of 10,000 and a Poisson ratio of 0.3 are solved here, see Fig. 8. The first example shows the displacement and normal strain of a quarter of plate with a hole, which was designed directly in Rhino. The quarter plate is constrained from moving along the horizontal direction at the right edge and vertical direction at the bottom edge. The second example shows the displacement and normal strain of a notch structure under bending, which was created by sweeping the geometry in Fig. 2(b). The generated 2D notch model has one extraordinary node. After dealing with it

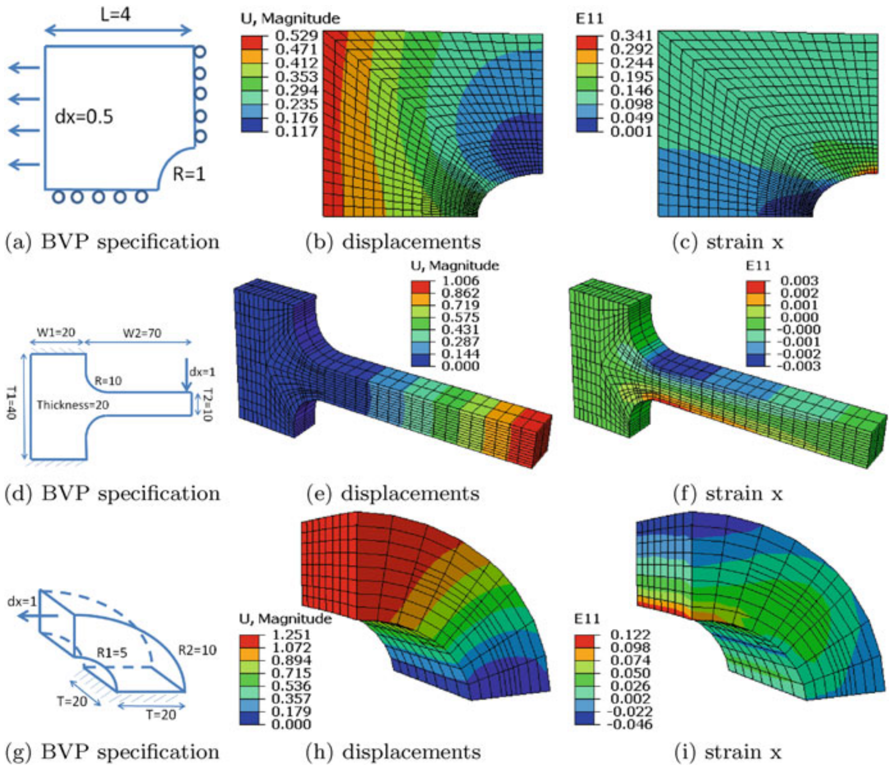

Fig. 8 Analysis results. (a-c) 2D plate with a hole; (d-f) 3D notch structure; and (g-i) 3D quarter cylinder.

with the interval duplication method [16], we sweep the geometry to obtain the volumetric T-spline. There is no further requirements for dealing the generated partial extraordinary nodes and we can directly obtain the Bézier extraction matrix. The geometry is fixed at the top and bottom while a load is applied at the tip of the notch structure. The third example shows the deformation and strain of a quarter cylinder, which was created using the conformal parametric mapping method [7]. The cylinder is fixed at one end and the load is applied at the other end.

In conclusion, this chapter presented a novel integrated CAD-CAE T-spline IGA software framework based on Rhino and Abaqus. The framework can solve both surface and volumetric T-spline problems. In particular, this framework realizes the transformation from Rhino surface T-splines to volumetric T-splines based on efficient *STSP/VTSP* data structures. Three examples were given to demonstrate our software framework. Generalizing this platform to arbitrary topology geometry is possible [6]. We are planning to implement this algorithm, enabling a more general set of geometries in the future.

Acknowledgements This project was supported by an ONR SBIR Phase I project. In addition, the research at CMU was supported in part by the PECASE Award N00014-14-1-0234.

References

1. Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. Comput. Methods Appl. Mech. Eng. **194**(39), 4135–4195 (2005)
2. Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A.: T-splines and T-NURCCs. ACM Trans. Graph. **22**(3), 477–484 (2003)
3. Bazilevs, Y., Calo, V.M., Cottrell, J.A., Evans, J.A., Hughes, T.J.R., Lipton, S., Scott, M.A., Sederberg, T.W.: Isogeometric analysis using T-splines. Comput. Methods Appl. Mech. Eng. **199**(5), 229–263 (2010)
4. Li, X., Zheng, J., Sederberg, T.W., Hughes, T.J.R., Scott, M.A.: On linear independence of T-spline blending functions. Comput. Aided Geom. Des. **29**(1), 63–76 (2012)
5. Zhang, Y., Wang, W., Hughes, T.J.R.: Solid T-spline construction from boundary representations for genus-zero geometry. Comput. Methods Appl. Mech. Eng. **249–252**, 185–197 (2012)
6. Wang, W., Zhang, Y., Liu, L., Hughes, T.J.R.: Trivariate solid T-spline construction from boundary triangulations with arbitrary genus topology. Comput. Aided Des. **45**(2), 351–360 (2013)
7. Zhang, Y., Wang, W., Hughes, T.J.R.: Conformal solid T-spline construction from boundary T-spline representations. Comput. Mech. **51**(6), 1051–1059 (2013)
8. Wang, W., Zhang, Y., Xu, G., Hughes, T.J.R.: Converting an unstructured quadrilateral/hexahedral mesh to a rational T-spline. Comput. Mech. **50**(1), 65–84 (2012)
9. Liu, L., Zhang, Y., Hughes, T.J.R., Scott, M.A., Sederberg, T.W.: Volumetric T-spline construction using Boolean operations. Eng. Comput. **30**(4), 425–439 (2014)
10. Vuong, A.V., Heinrich, C., Simeon, B.: ISOGAT: A 2D tutorial MATLAB code for isogeometric analysis. Comput. Aided Geom. Des. **27**(8), 644–655 (2010)
11. De Falco, C., Reali, A., Vázquez, R.: GeoPDEs: a research tool for isogeometric analysis of PDEs. Adv. Eng. Softw. **42**(12), 1020–1034 (2011).
12. Scott, M.A., Borden, M.J., Verhoosel, C.V., Sederberg, T.W., Hughes, T.J.R.: Isogeometric finite element data structures based on Bézier Extraction of T-splines. Int. J. Numer. Methods Eng. **88**(2), 126–156 (2011)
13. Asche, C., Berkhahn, V.: Efficient data structures for T-spline modeling. EG-ICE 2012 International Workshop: Intelligent Computing in Engineering, Technische Universität München, Germany (2012)
14. Elguedj, T., Duval, A., Maurin, F., Al-Akhras, H.: Abaqus user element implementation of NURBS based isogeometric analysis. In: 6th European Congress on Computational Methods in Applied Sciences and Engineering, Vienna (2012)
15. Scott, M.A., Hughes, T.J.R., Sederberg, T.W., Sederberg, M.T.: An integrated approach to engineering design and analysis using the Autodesk T-spline plugin for Rhino3d, ICES REPORT 14-33. The Institute for Computational Engineering and Sciences, The University of Texas at Austin (2014)
16. L. Liu, Y. Zhang, X. Wei, Handling extraordinary nodes with weighted T-spline basis functions. 24th International Meshing Roundtable. Procedia Engineering 124, 161–173 (2015)