Truss model data analysis in Autodesk Revit

A A Vedernikova¹, R A Shishmarev²

¹ Saint Petersburg State University of Architecture and Civil Engineering, 4 Vtoraya Krasnoarmeiskaya ul., Saint Petersburg, 190005, Russia
² weblog “InfoBIM”, http://infobim.ru/

E-mail: parallelogram13@gmail.com¹, mr.shruslan@gmail.com²

Abstract. The article presents the process and results of truss model data analysis using the Revit API. Methods for obtaining model data are presented, as well as an example of their application: calculation of the truss using the finite element method. Fragments of the program code for the plug-in are given. The calculation results in the plug-in correspond to the calculation results in specialized programs. This article is useful for architects and designers using the Revit API and Dynamo.

1. Introduction

Working with BIM often requires data analysis to perform particular design tasks [1]. For example, technical expertise on tower cranes [2], mining implicit 3D modelling patterns [3], daylighting simulation and analysis [4], organization of Autodesk Revit software interaction with applications for structural analysis [5], analysis of the overall sustainability of the building [6]. Engineers usually use plugins for such tasks.

The article considers an analysis of the model of the truss frame (Fig. 1): obtaining and preparing geometry and finite element analysis to obtain forces in the truss elements. C# programming language and the Revit API were used for this. Many engineers use programming tools in their work. This article shows that the capabilities of BIM are not limited to modelling. Engineering calculations can also be performed without the use of additional software.

![Figure 1. Truss frame.](image-url)
2. Materials and methods
The plugin was based on the calculation of truss elements by the finite element method for bar systems described in [7], briefly outlined in part 3. C# and the Revit API were used. The data for the calculation was obtained by special methods for objects in the Revit API. The plugin methods code is presented in part 3.

3. The plugin creating process and results of its work
The plugin extracts initial data (truss geometry) from the model in Revit, filters necessary data, and sets missing parameters (loads, boundary conditions).

The plugin framework consists of three stages. The first stage gets the geometry of the truss elements which contain lines in base (Fig. 2). Every line has coordinates of the end and start points. Therefore, all truss frame lines are obtained using two methods GetLine and GetLines (Fig. 3). After that, a list of all points is formed using GetXYZs (Fig. 4) to obtain truss joints based on it. Then an additional GetJoints method sorts the list of all points to get unique points (Fig. 5).

One of the goals is dividing the truss cords into finite elements because often they are drawn with one line. The lines that need to be divided into finite elements are found using the obtained nodes. The result is a list of 25 lines.

The second stage is sorting. The nodes are sorted by three coordinates: first by Z, then by X and Y (Fig. 4). After that elements are sorted. An alternately scanning nodes and finding correspondences between nodes and lines method was developed for that. The order of the lines corresponds to the minimum sum of the node numbers. The first line is the line between nodes 0 and 1, the second between 0 and 2, the next between 1 and 2, and so on. As a result, the necessary sorted list of lines and nodes are obtained.

Figure 2. Truss frame and line base.
```java
class FrameLines {
    // Get the lines from element
    public Curve GetLine(Element el) {
        Curve beam_c;
        LocationCurve loc = el.Location as LocationCurve;
        beam_c = loc.Curve;
        return beam_c;
    }

    // Get the lines from elements
    public List<Curve> GetLines(List<Element> elements) {
        List<Curve> truss_curves = new List<Curve>();
        foreach (Element el in elements) {
            Curve curve = this.GetLine(el);
            truss_curves.Add(curve);
        }
        return truss_curves;
    }
}
```

Figure 3. Class `FrameLines` and method `GetLine`, `GetLines`.

```java
// Get all points from all curves
public List<XYZ> GetXYZs(List<Curve> all_curves) {
    List<XYZ> points = new List<XYZ>();
    foreach (Curve c in all_curves) {
        XYZ p_s = c.GetEndPoint(0);
        XYZ p_e = c.GetEndPoint(1);
        points.Add(p_s);
        points.Add(p_e);
    }
    return points;
}
```

Figure 4. Obtaining truss frame nodes coordinates.

```java
// Get all joints from all XYZ
public List<XYZ> GetJoints(List<XYZ> all_xyz) {
    List<XYZ> joints = new List<XYZ>();
    joints.Add(all_xyz[0]);
    foreach (XYZ p in all_xyz) {
        int count = 0;
        foreach (XYZ n_p in joins) {
            if (this.IsCompared(p, n_p)) {
                count++;
            }
        }
        if (count == 0) {
            joints.Add(p);
        }
    }
    return joins;
}
```

Figure 5. Obtaining truss frame nodes coordinates.
The third stage is calculation. Finite element method is used to calculate the truss frame [7]. It is assumed that the truss has pin-joints. Loads are nodal.

The array of forces:

\[
\begin{bmatrix}
F_{x1}, F_{z1}, F_{x2}, F_{z2}, \ldots, F_{xn}, F_{zn}
\end{bmatrix}
\]  

Here \( F \) is the load along the axis indicated in the index, \( n \) is the node number. Zero is set if there is no load in a particular direction and a particular node. Each node of the truss top cord is loaded with a concentrated load of 10 kN for the calculation example.

The list of boundary conditions is set similarly to the list of loads but instead of the values of the loads, the positions for the fixed nodes are set to 1, for not fixed – to 0. The nodes of the truss lower cord are assumed to be fixed so the boundary conditions list contains \([1, 1]\) and \([0, 1]\) respectively for these nodes.

The stiffness of each element is defined as \( A \cdot E \), where \( A \) is the cross-sectional area, \( E \) is the elastic modulus of steel. The area can be extracted from the model or set conditionally. \( E = 2 \cdot 10^9 \text{ MPa} \).

The result is several data arrays: the coordinates of the truss nodes, the length of the elements, the lists of loads, boundary conditions.

The vector of internal nodal forces of a structure in the general coordinate system \(X^0, Y^0, Z^0\) [7]:

\[
\begin{bmatrix}
\{S^0_1\} = \left[ K^0_a \right] \cdot \left[ A \right] \cdot \left[ \left[ K^0_a \right] \cdot \left[ A \right] \right]^{-1} \cdot \{P^0\} = \left[ K^0_a \right] \cdot \{A \} \cdot \{Z^0\}
\end{bmatrix}
\]  

\( \{S^0_1\} \) – vector of internal nodal forces of the structure, consisting of blocks (cells) of vectors of internal nodal forces of finite elements (FE);

\( \left[ K^0_a \right] \) – quasi-diagonal design stiffness matrix, consisting of stiffness matrix blocks FE in the general coordinate system;

\( \{A\} \) – correspondence matrix (nodes) of the structure, consisting of blocks of correspondence matrices FE;

\( \{P^0\} \) – nodal load vector;

\( \{Z^0\} \) – vector nodal displacements of the construction.

This part of script gives truss element stresses. It multiplies matrices using the Matrix library (C#) or Numpy library (Python). A calculation was made in another program (SCAD 21.1) to confirm the calculation as fig. 6 and 7 shows.

Figure 6. SCAD 21.1 calculations results [kN].
Figure 7. Plugin calculation results [kN].

4. Conclusion
This article will be useful to engineers and designers working with the API and looking for ways to speed up and simplify their workflow. The Revit API has special methods for retrieving data, and this article provides examples for its application. It is possible to extract data for structural analysis as a result of the analysis of the Revit model data. For example, FEM calculation that sets the cross-section of truss elements.

Perhaps in the future such plugins will be fully developed by large software vendors, but today there is a need for designers to have tools for solving particular problems and to be able to adapt it to their needs.

References
[1] Mladzievskiy E P 2020 Academy 2(53) pp 9–10
[2] Sharmanov V V, Mamaev A E and Simankina T L 2019 Vestnik nauchnogo tsentra 2 pp 76–86
[3] Yarmohammadi S and Castro-Lacouture D 2018 Automation in Construction 93 pp 91–111
[4] Kota S, Haberl J S, Clayton M J and Yan W 2014 Energy and Buildings 81 pp 391–403
[5] Zotkin S P, Ignatova E V and Zotkina I A 2016 Procedia Engineering 153 pp 915–919
[6] Xiongjue W 2016 Perspectives in Science 7 pp 317–322
[7] Chernyy A N and Chernov S A 2010 Truss finite element method calculation (Ulyanovsk: UISTU) p 30