LAB University of Applied Sciences
Civil and Construction engineering, Lappeenranta
Double Degree Programme in
Civil and Construction Engineering

Kosavchenko Daria

BIM Geometry Creation from Point Clouds

Thesis 2020

Abstract

Kosavchenko Daria
BIM Geometry Creation from Point Clouds, 64 pages, 4 appendices.
LAB University of Applied Sciences
Civil and Construction Engineering, Lappeenranta
Double Degree Programme in
Civil and Construction Engineering
Thesis 2020
Instructors: Lecturer Heikki Vehmas, LAB University of Applied Sciences;
Managing Director Patrik Laakso, Nodetec Oy.

Over the past two centuries, scientific and technological progress has made such a large qualitative and quantitative leap that it could not but affect the development, methods and techniques of studying and learning most scientific disciplines. Now building information modeling is widely used in the construction industry. With the development of technology, economics, and competition in the world, reverse engineering is becoming an advanced method for studying objects of their structure, construction, and functioning. This method, using the most modern methods of obtaining spatial information about the object, allows returning the lost or not existing, comprehensive information about the subject of research.

The essence of this work is to study reverse engineering based on three-dimensional scanning data, methods for creating three-dimensional models, and automation of this process using Tekla plug-ins. The development was carried out using Tekla Open API in C# using Visual Studio. The Tekla Open API provides an interface for third-party applications to interact with model and drawing objects in Tekla Structures.

In the result of the research, several methods of designing from the point cloud were identified, from manual modelling to automatic one. This study is the beginning of a path in automating the process of converting a point cloud into structural models. In general, this approach shows improvement of BIM modelling processes and will be useful in future use.

Keywords: Tekla, Tekla Structures, BIM, application for Tekla Structures, Tekla Open API, IFC, Cyclone, pointcloud.

Contents

Appendices

Definitions

Point cloud (point model) is a set of points obtained as a result of 3D scanning of a real-world object and representing the surface of this object in a three-dimensional coordinate system.

A polygon mesh (model) is a set of topologically connected simple geometric two-dimensional primitives that describe the surface of an object. These primitives are polygons, which are shapes with straight sides (3 or more sides) defined by three-dimensional space points (vertices) and connecting lines (edges).

Bezier curves are a way to define a curve from reference points. The curve is described by an array of control points, which must be at least two. To draw a curve, you need to draw two imaginary lines, one of which will have the coordinates P_0 and P_1, and the other — P_1 and P_2. Then the extreme points of these lines start moving continuously to the next points. The third imaginary line is drawn with the starting point moving continuously along the first line and the end point moving along the second line. On this imaginary line, a point is drawn that gradually moves from the beginning of the line to the very end. The curve that this point will describe is called the Bezier curve.

A Boolean operation is used to denote comparison operations between sets. A Boolean operation is performed by creating a Boolean composite object from two existing objects — these objects are called operands and must necessarily intersect in some area of space. Operands are represented as separate objects during the entire stage of editing a Boolean composite object, which allows you to select and modify them if necessary.

NURBS are surfaces or curves whose shape is described by inhomogeneous rational B-splines. Depending on the type of NURBS, surfaces can be constructed using control vertexes or control points that lie on the surface. These surfaces are an ideal tool for creating organic shapes: they are easy to work with, have good interactive control, allow you to make seamless surfaces and remain smooth even on a curved surface.

Solid-state modeling is a method based on the principle of forming a complex model from elementary volumes: the main parallelepiped, cube, sphere, cylinder, cone, wedge, torus, as well as, as a result of rotation, squeezing out areas and using logical operations of combining, subtracting and intersecting.

1 Introduction

Over the past two centuries, scientific and technological progress has made such a great leap in quality and quantity that it could not but affect the development, methods and methods of study and knowledge of most scientific disciplines. This also applies to engineering (applied) geodesy and ground (applied) photogrammetry, which provide various spheres of human activity with methods, techniques and organization of geodetic and photogrammetric works for solving engineering and other tasks. One of these tasks is reverse engineering, which is a set of methods, methods and technical solutions for determining the shape, size, design, functioning and other characteristics of real-world objects.

With the development of technology, economics, and competition in the world, reverse engineering is becoming an advanced method for studying objects of their structure, construction, and functioning. This method, using the most modern methods of obtaining spatial information about the object, allows returning the lost or not existing, comprehensive information about the subject of research. The main positive point of using reverse engineering is not duplication of the object, but its study in order to Refine and release a new, more perfect product. The improvement of something based on the experience of past years has always been, is and will be the basis of scientific, technical, and any other progress. Therefore, the improvement of this direction directly entails the development of other Sciences and disciplines.

Currently, the most common method for obtaining a model of an existing project is manual modeling based on drawings and measurements made on the site. Now the technological process makes it possible to use a more convenient and fast method of modeling - creating point clouds of a building structure. This topic is widely discussed and improved for use in the construction business. This method will not only speed up the modeling process, but also eliminate the use of human labor at the entire stage of object modeling, which will improve the economic side of this issue.

2 BIM (Building Information Modelling)

Building information modeling (BIM) is a new look at the process of optimizing design and construction. Using BIM technology, an information model is created that provides an accurate vision of the project as a whole. Building information modeling technologies are a qualitatively new approach in architectural and construction design, which consists in building a three-dimensional virtual model of a building in digital form and carries complete information about the future object.

The use of BIM technology in building design involves the collection and complex processing of technological, architectural, design, and economic information about the building, so that the construction object and everything related to it are considered as a whole. A three-dimensional model of a building is closely linked to an information database, so changing at least one parameter of a building object also changes all related systems and objects, including drawings, specifications, visualizations, and a calendar schedule.

The principles of BIM, the:

- 3D-modeling;
- automatic drawing production;
- intelligent object parameterization;
- sets of design data corresponding to the objects;
- distribution of the construction process by time stages.

Advantages of using BIM:

- reducing design time;
- reducing project implementation costs;
- increased productivity due to the ease of obtaining information;
- improving the consistency of construction documentation;
- availability of specific information about material manufacturers and quantitative characteristics for evaluation and tendering.

Building information modeling (BIM) as a way to achieve efficiency from the construction industry by providing an improved way of working throughout the asset lifecycle simplification of this lifecycle is shown in figure 1 and shows the main functions in this process. It is at the center of this cycle that data about digital assets in the form of a model is located.

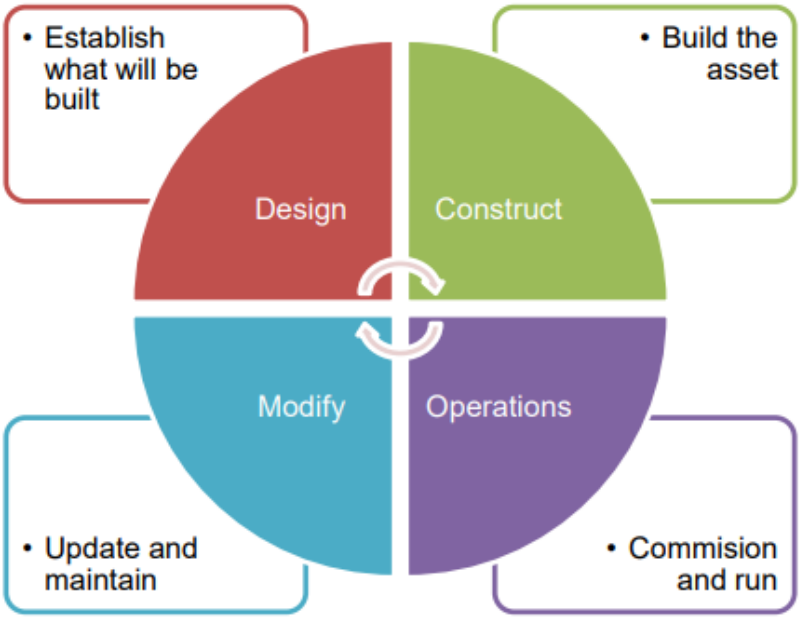


Figure 1.Types of Tekla Structures expansions.

3 Modelling from pointcloud

Point cloud (point model) is a set of points obtained as a result of 3D scanning of a real-world object and representing the surface of this object in a three-dimensional coordinate system (Fig.2.). In addition, the point model can be obtained from an already designed digital model that has no analog in the real world. Cloud points are usually represented by XYZ coordinates, which can be simply written in a file, but there are formats that have a slightly different representation.

Point clouds provide fast visualization of a real-world object. At the same time, they are successfully used for measurement and control of objects, 3D printing, visual visualization of hard-to-reach places or large extended objects, creation of three-dimensional and mathematical models, image recognition, automated

analysis, reconstruction and operation, as well as are the basis for reverse engineering of real-life objects. [4]

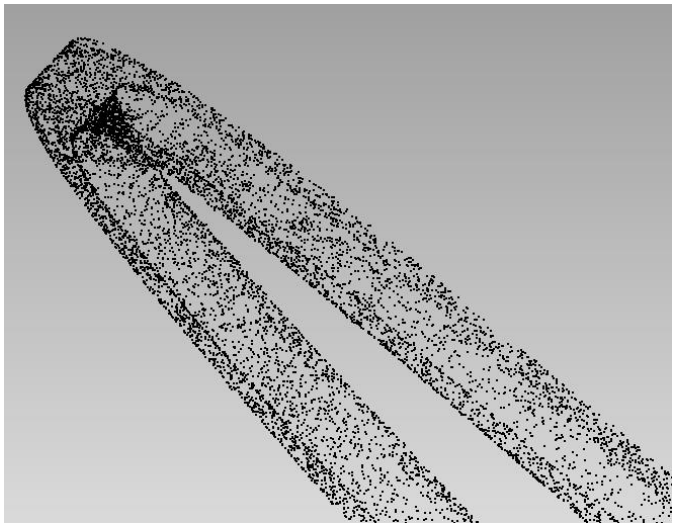


Figure 2. A cloud of points on the surface of an object.

Figure 2 shows an example of a simple point cloud that contains only one cloud. This type of cloud does not require merging the clouds together. But in most cases, a point cloud consists of a set of clouds scanned from different points. After scanning, you need to register and combine the scan data together.(Fig.3)

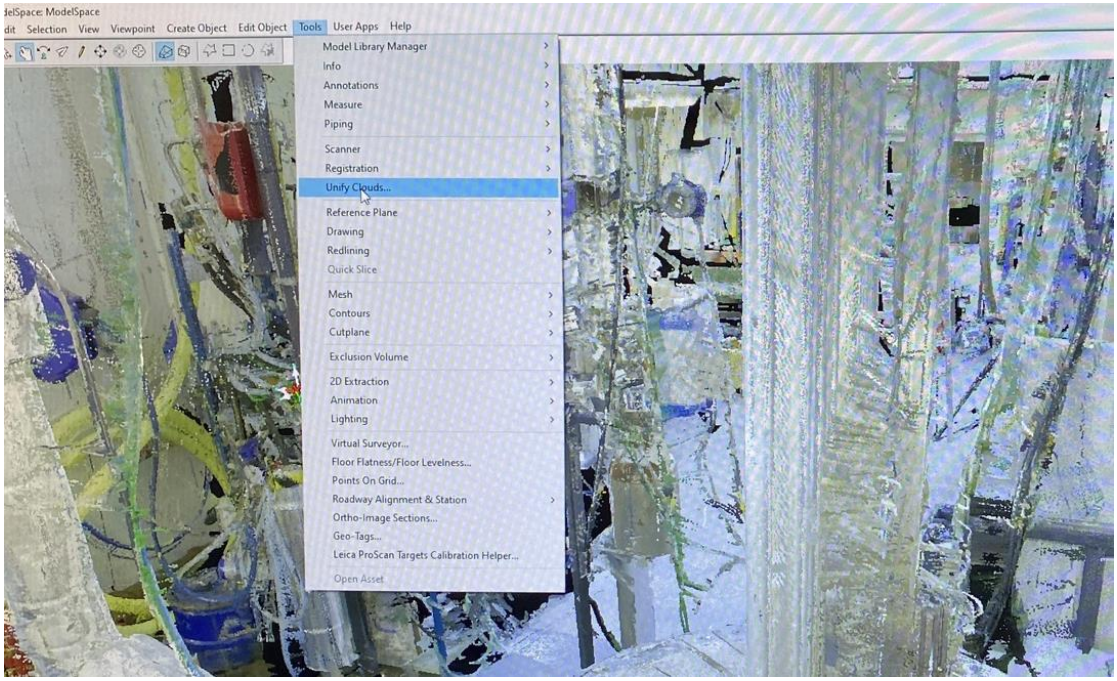


Figure 3. Merging multiple point clouds.

3.1 Registration of point clouds

With traditional static scanning, measurements can only be taken of the environment that is visible from the scan position (i.e. line of sight). Therefore, occlusions occur where objects in the environment block the view from the scanner. To minimize these multiple setups are used to ensure a required level of coverage. To allow these multiple scans to be brought together into their correct position relative to each other a registration is performed.

Data-Driven (Cloud-to-Cloud) form of registering point data is generally performed by the iterative closest point (ICP) algorithm [5] which has become the dominant approach.[6] It works by iteratively translating and rotating with six degrees of freedom a free dataset to a fixed one until the transformation converges within a required tolerance. The advantage of this form of registration is the target-less nature meaning that the process is fully automated. However, the scans need to be reasonably positioned for the process to converge correctly and quickly.[7] To improve the registration by reducing ambiguity in the matching phase the initial conditions can be improved by the user providing matching tie points. This is done by providing at least 3 common points between pairs of scans

or by taking the results of a registration by targets. Then the ICP can be performed as a fine or local registration step.[8]

3.2 Segmentation

For reconstruction of building elements, the main focus is on computational geometry algorithms. Segmentation was used to extract a three-dimensional representation of building elements. Distance measurement data segmentation is a long-used method for classifying data with the same characteristics and grouping them together. Hoover et al. (1996) provide a General overview of this topic [16]. To date, it has been possible to bring together various approaches to segmentation that were used at that time, and present a method for evaluating these algorithms. The main tasks of segmentation research were described as the development of an algorithm that can handle a wide range of geometries and select a reliable measure of points belonging to the same cluster. [18]

3.3 Modelling

Combining traditional surveying with 3D scanning currently does not result in a product that is optimal for the process of BIM due to the historical use of non-parametric CAD software to create 2D survey drawings. Therefore, a process shift is required in workflows and modelling procedures of the stakeholders who do this work to align themselves with the new information-rich object-oriented 3D deliverables of BIM.

Typically, digital modeling is performed to provide a representation or simulation of an object that may not exist in reality. However, geomatics seeks to model objects as they exist in reality.

Currently, this process is largely manual and is recognized by many as time-consuming, tedious, subjective and requires skills [8]. Manual process of documenting buildings from point clouds, just like when creating 2D CAD plans. This requires the user to use the cloud as a guide in the design software to effectively track the geometry, requiring high knowledge input to interpret the scene, and add rich semantic information that really makes BIM a valuable process.

In the case of reverse engineering, the first step in analyzing this type of data is usually to reconstruct the surface and triangulate it in order to obtain a basic approximation in the form of a polygon grid. Triangulation can be used later to approximate such a surface by higher-order functions, such as parametric or Nurbs surfaces.

The point cloud modelling process can be divided into two large groups: creating models using a polygon grid or solid-state modelling. Using both methods is effective. The choice of modelling method depends on the type of object. If you need a model that carries information about the mass, volume, and center of gravity, then solid modelling can provide this. If this information is not important and there is a way to indicate this property by another method, then you can use polygon modelling. For example, for modelling ground-level surfaces.

For example, for modeling a model intended as a reference, as in renovation projects in which the exact location of structures is necessary to model a new part. As in the project of an additional part of the underground Parking, there was no need to use a model other than the reference one. There was no need to create drawings of the old part, so in this project, you could use the polygon grid to create the shell of the building, to use it as an accurate reference point for the location of parts of the structures.

And if we need to set the object's mass, volume, or need to create or modify the design. In this case, we use solid modeling to exclude additional inclusions in the program for converting the surface to a solid model. In this situation, using a polygon grid makes it difficult and slows down the modeling process. This will require more knowledge and effort.

3.3.1 The polygon grid

A polygon mesh (model) is a set of topologically connected simple geometric two-dimensional primitives that describe the surface of an object. These primitives are polygons, which are shapes with straight sides (3 or more sides) defined by three-dimensional space points (vertices) and connecting lines (edges) (Fig.3). The inner region of a polygon is called a face.[15]

Typically, in a polygon grid, different faces share vertices and edges, so their topology is implemented. In this case, they are called common vertices or common faces. The outer edges of the grid are called boundary edges.[15]

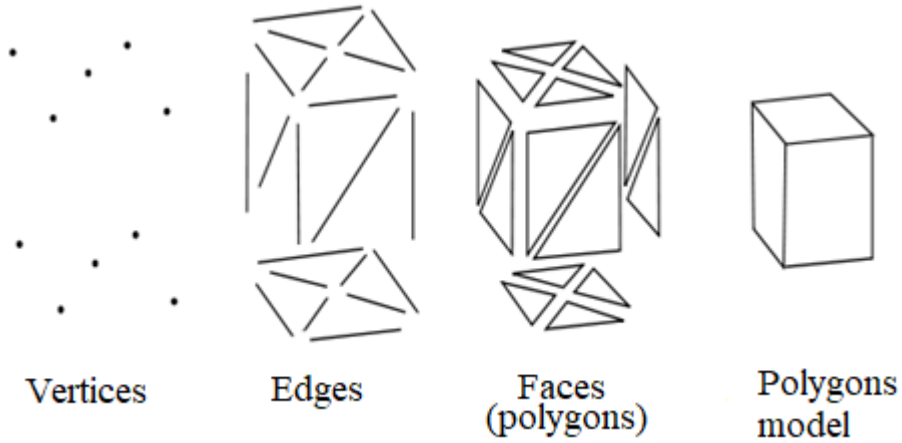


Figure 4.Components of a polygon grid.

The front side of a polygon face is graphically represented by a vector perpendicular to it, called the face normal (Fig.5).

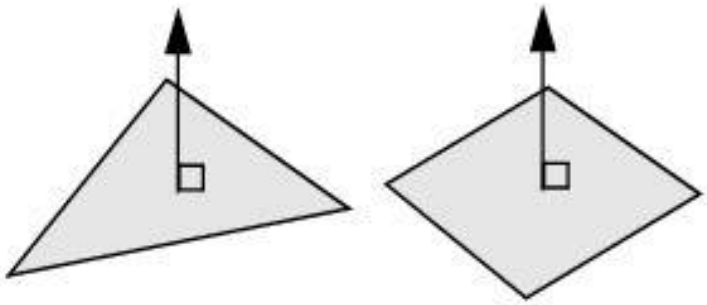


Figure 5.The normal to the surfaces.

The order of enumeration of the vertices surrounding the face determines its direction in which direction the face is facing, and in which - the wrong side. This fact may be important, since polygons are visible only from their front side.[15]

The polygon model is more accurate and closer to the original object, the smaller the size of the polygons, and the greater their number.

Therefore, these polygonal models are divided into high poly and low poly, and since the amount of data in polygonal models greatly affects the performance of technical equipment, these two types of polygonal models have different

applications. Low-poly models are used mainly where high performance and speed of model visualization in the program are needed, for example in games. But it is possible to increase the detail of the model by tessellation. Tessellation is a method by which it is possible to increase the number of polygons in a three-dimensional model using Bezier curves. (Fig.6)

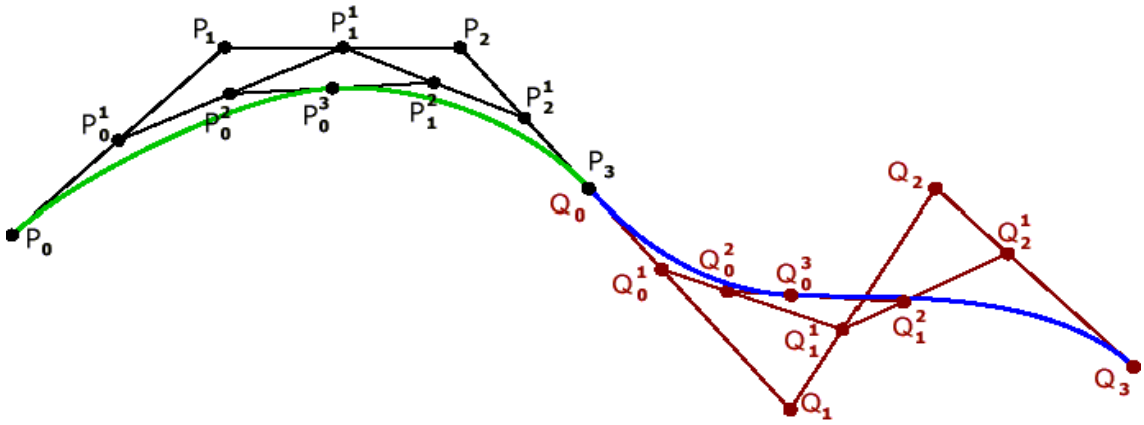


Figure 6. A spline made up of Bezier curves.

Polygonal modelling is modelling only the surface of an object, so it is related to hollow modelling. This is manifested in the fact that when editing such a model if delete part of the polygons, a hole is formed in the surface, through which it is possible to see the entire inner part of the model. That is, such a model, unlike solid-state models, does not have information about its volume. Therefore, the disadvantages of polygonal models include the fact that it is impossible to get information about the physical properties of an object, such as mass, volume, center of gravity, etc. from such models[16]

3.3.2 Solid model

Solid-state modelling is the most advanced and reliable method for creating a copy of a real object.[4] The model is based on the principle of forming a complex model from elementary volumes: basic parallelepiped, cube, sphere, cylinder, cone, wedge, torus, as well as, as a result of rotation, squeezing out areas and using logical operations of combining, subtracting, and intersecting. (Fig.7, Fig.8, Fig.9).

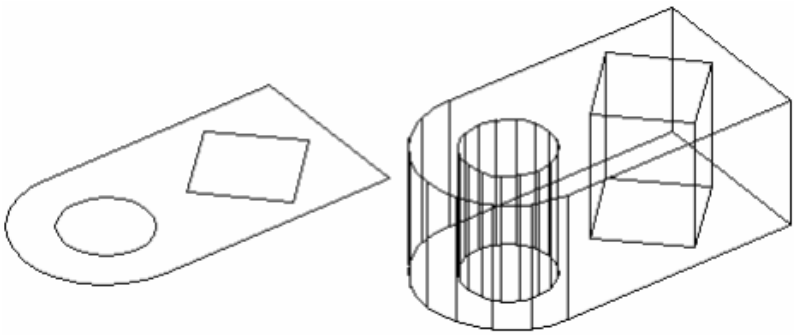


Figure 7. The principle of forming a complex model by squeezing out areas.

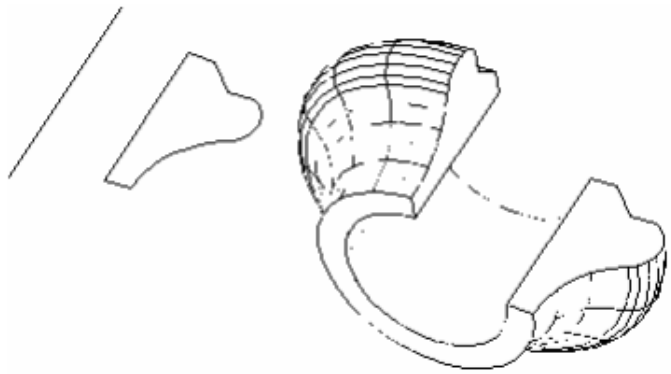


Figure 8. The principle of forming a complex model as a result of rotation.

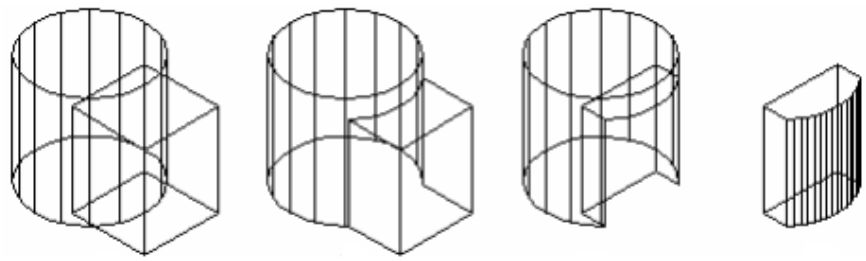


Figure 9. The principle of forming a complex model from elementary volumes.

A solid-state model is constructed by creating a complete, non-discontinuous set of surfaces that are cross-linked and form a regular closed geometric volume, that is, a solid body contains an internal volume that is bordered by an external surface. Changing one of them leads to changing the others. [21] A Fundamental property of solid-state modelling is the preservation of the topology of the elements of the body. The relationship of elements - their mutual location. Preservation of connections of elements of a body provides isolation and

consistency of the volume of a solid body. The relationships established between the elements are stored together with the geometric information in the database.

Solid-state modelling methods based on Boolean operations are particularly convenient for calculating the surface and weight characteristics of bodies, calculating stresses, and simulating machining operations. [20] In the latter case, metal cutting operations (turning, milling, drilling, etc.) can be easily described using a Boolean difference. (Fig.10).

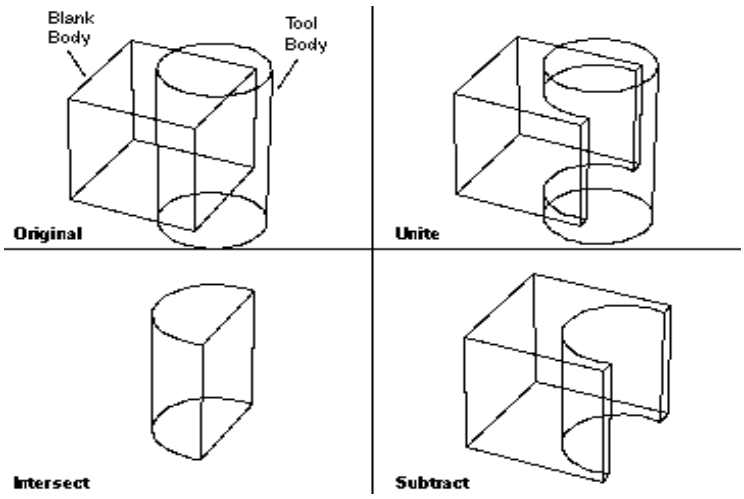


Figure 10. A Boolean difference.

It consists of the construction of complex solid-state models, from the basic simplest constituent elements, called solid-state primitives, which are determined by the shape, size, anchor point, and orientation. Each primitive is defined by a certain shape (a parallelepiped, a cylinder of variable cross-section, a ball, a cube, a sphere, a cone, a wedge, etc.), a reference point, an initial orientation, and variable dimensions. The construction of basic primitives consists of creating bodies by expanding (squeezing, rotating, and moving along a curve) two-dimensional regions in three-dimensional space. This creates a volume (a closed space defined by the parameters of the sketch and extrusion along the curve) – a solid body. Editing such models is reduced to changing the parameters of the sketch and extrusion, after which the body model is changed.

Methods for creating solid-state models are divided into two classes:

3.3.3 The Method of constructive representation (C-Rep).

It consists of building complex solid-state models from the basic simplest component elements, called solid-state primitives, which are defined by shape, size, anchor point, and orientation. Each primitive is defined by a certain shape (parallelepiped, cylinder of variable cross-section, ball, cube, sphere, cone, wedge, etc.), anchor point, initial orientation, and resizable dimensions. The construction of basic primitives consists of creating bodies by means of unfolding(squeezing, rotating, and moving along a curve) two-dimensional regions in three-dimensional space. This creates a volume, a closed space defined by the sketch and extrusion parameters along the curve – a solid. Editing such models is reduced to changing the sketch and extrusion parameters, and then the body model is rebuilt.

The model of constructive geometry is a binary tree graph $G=(V, U)$, where V is the set of vertices – the basic elements of the form – primitives from which the object is constructed, and U is the set of edges that denote the set – theoretic operations performed on the corresponding basic elements of the form. Each primitive model is specified by a set of attributes $A=$, where x,y,z – coordinates of the anchor point of the local coordinate system entity to the coordinate system of the synthesized object, ax, ay, az - rotation angle of the primitive around the corresponding coordinate axes, $Sx, Sy, ..., Sn$ - metric parameters of the object.

The tools for building Creep are Boolean operations based on algebraic set theory. Boolean operations are the main tool for building the C-Rep model when defining relationships between neighboring primitives. [19,20]

Most often used operations:

- The join operation ($\cup$) defines the space inside the outer boundary of a composite form derived from two bodies with a common area. The Union of two arbitrary circles A and B in Fig.11 is a shaded area $A \cup B$. thus, the union operation defines the resulting composite form as a single element. The same figure shows the application of an equivalent operation for two solid-state primitives (cylinder P and cuboid Q). There

is also a cross-section of the Union $P \cup Q$, to emphasize that a new solid form was formed, not like either a cylinder or a cuboid n.

- The difference operation (—) defines the space bounded by the surface from one form and the outer boundary of the common area of the two forms. In Fig.11, the shaded area $A - B$ shows the result of the Boolean difference operation on the circles A and B. Below, the same figure shows a cylinder with a groove $P - Q$, which is the difference between two solids P and Q. for Example, a model of a plate with a hole in it can be obtained by subtracting the cylinder from the parallelepiped.

- The intersection operation ($\cap$) defines the space within the boundaries of the shared object area. The intersection of the circles A and B is represented in Fig.11 by the shaded area $A \cap B$, and the intersection of the bodies P and Q is represented by the solid form $P \cap Q$.

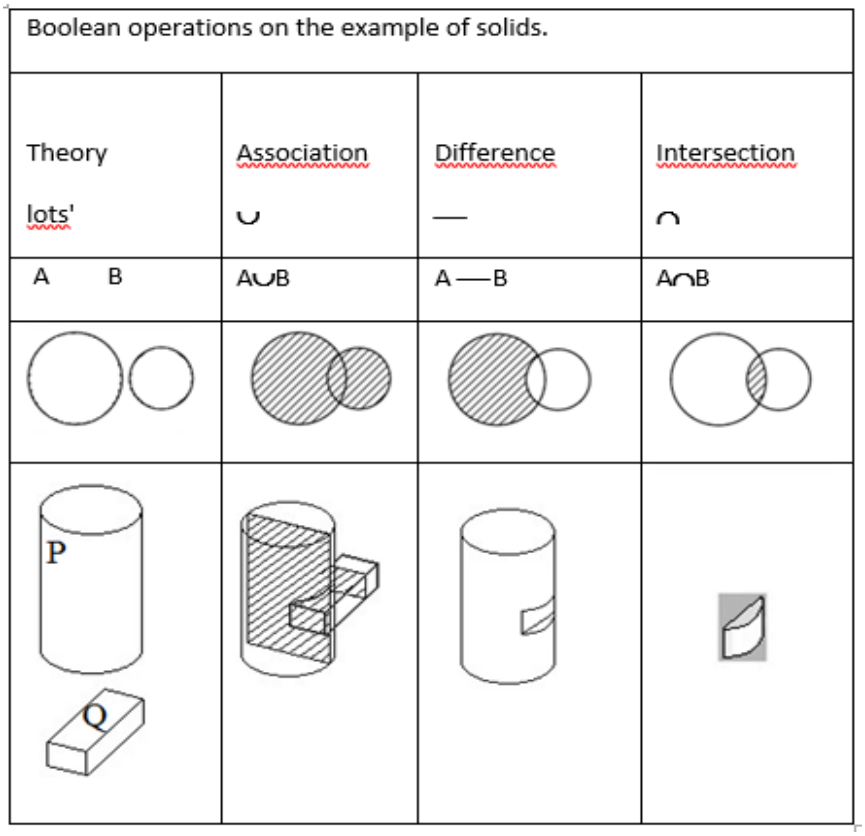


Figure 11. Boolean operations on the example of solids.

3.3.4 Boundary representation (surface) method (B-Rep).

The boundary representation method (B-Rep) also operates on primitives associated with Boolean operations. (Fig.12) Boundary representation-setting the boundary elements of the part-parametrically described surfaces (faces-face), edges (borders or intersections of faces), vertices (part/point on the curve that forms the edge), describing the body. This data is supplemented with information about the topology of the primitive and the features of its geometry.

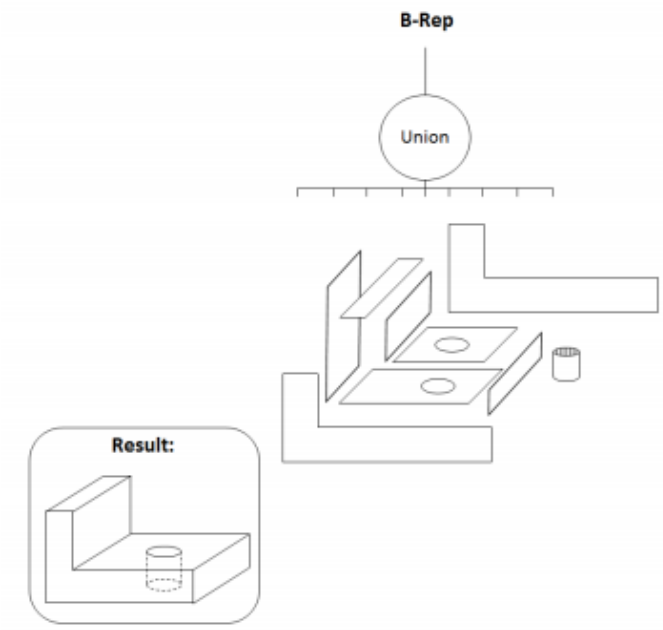


Figure 12. B-rep example representations for a combined shape.

In this method, the axis and contour of the future object are set, and then the area occupied by the contour is constructed. Then remove the hidden lines and paint the surfaces with their properties. It is the edges and faces that form the three-dimensional boundary surface of the bulk body. This is the only method that allows to create an accurate, not a perfect representation of a geometric solid. [19,20]

In this approach, the user is required to specify the contours or boundaries of the object, as well as sketches of surfaces of different types, and specify the lines of connection between these surfaces, so that you can establish a mutual correspondence. The B-Rep method is relevant in the formation of complex structures that are very difficult to recreate using the C-Rep method. Also, the

advantage of systems with B-Rep is the simplest change of the boundary representation in this frame model and its reverse change. The reason for this is that the description of the boundaries is similar to the description of the frame model. (Fig.13)

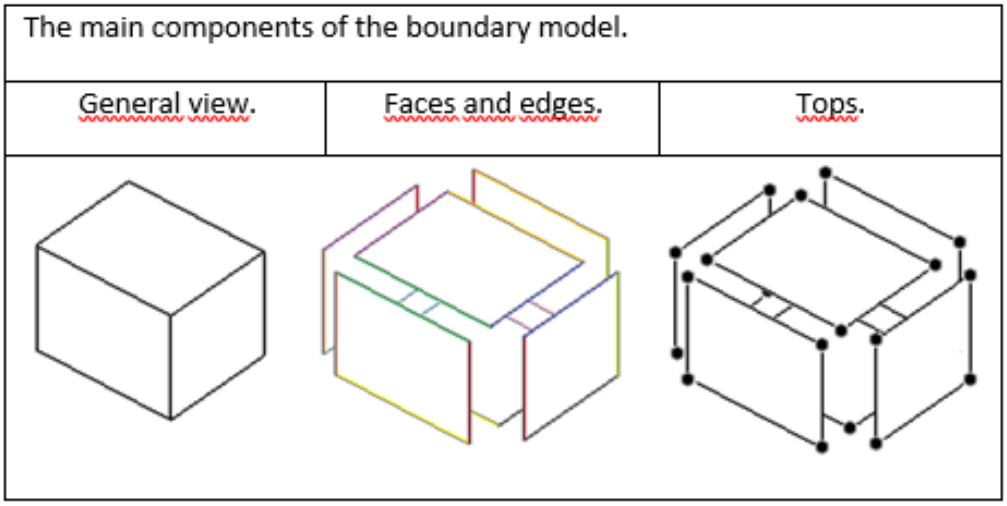


Figure 13.The main components of the boundary model.

Advantages and disadvantages of C-Rep and B-Rep methods[19,20]:

- A system with C-Rep representation has advantages in the initial formation of the model, since it is quite simple to build a volume model of the correct form from volume primitives using Boolean operations. In addition, this method provides a more compact description of the model in the database.
- However, the B-Rep representation is relevant when creating complex forms, which are very labor-intensive to recreate using the C-Rep method.
- In the C-Rep method, the model is stored as a combination of data and logical procedures, which requires less memory, but the amount of computation is large when the model is reproduced. On the other hand, the model c B-Rep does not store an accurate description of the model boundaries, it needs more memory, powerful hardware and SOFTWARE, but requires almost no calculations to recreate the image.
- The relative advantage of systems with B-Rep is the comparative ease of converting the boundary representation to the corresponding frame

model and back. The reason for this availability is that the description of borders is similar to the description of the framework model, which makes it easier to convert the model from one form to another.

3.4 Nurbs-surfaces

A non-uniform rational Bezier spline, NURBS (Non-Uniform Rational Bezier Spline) is a special case of Bezier curves. This is a special technology designed for the creation of smooth organic forms and models, based on a complex mathematical apparatus.[22] in total, there are about 1,500 equations to describe all geometric elements, from the simplest curves to complex surfaces. Because of the structure of NURBS, surfaces are always smooth (they do not have sharp edges inherent in polygons), so they are widely used in organic modelling (similar to the creation of plant forms), to create models of animals, people, machines, etc. NURBS surfaces do not consist of a grid of rectangles; the division of surfaces into polygons occurs only at the stage of visualization in the software and involves the use of an optimal algorithm to preserve smoothness. Therefore, at any approximation, the smoothness of the surface is observed.[22] Modelling based on NURBS curves is more flexible and allows to create any of the most bizarre models. However, NURBS models are more massive, difficult to edit, and take longer to visualize.[22]

Any NURBS model is a set of NURBS surfaces formed by NURBS curves. These curves are described by mathematical formulas, as a result, there is no need to remember each point of the curve, it is enough to know the coordinates of its beginning and end and the mathematical formula describing the curve. This allows creating complex curved surfaces with a small number of control vertices and easily get rid of rough-cut objects, giving them a smooth curved shape by simply increasing the detail. If I select one of these NURBS surfaces, such as a sphere, and zoom in on it, it will be seen that all its edges are made up of curves, and the individual components of the NURBS surface are rectangular patches. (Fig. 14)

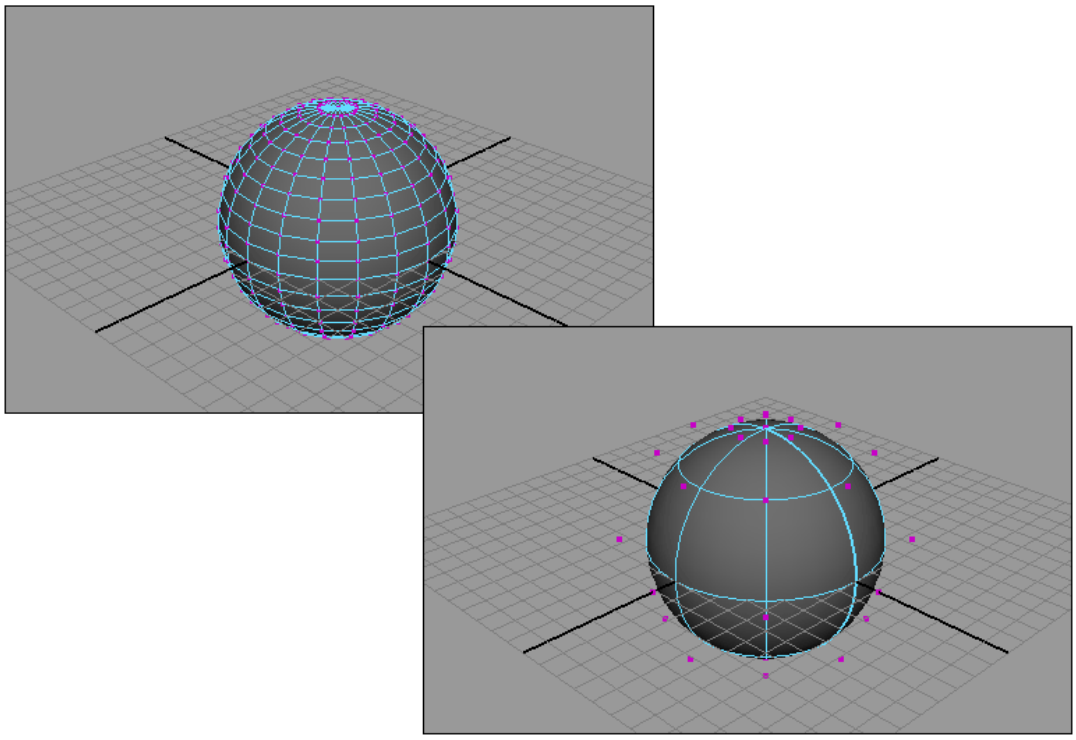


Figure 14.The level of detail of the polygonal ball (left) and NURBS-ball (right) - the number of vertices in the first ball is an order of magnitude greater.

The simplest type of NURBS-surfaces are NURBS-primitives represented by objects Sphere (sphere), Cube (cube), Cylinder (cylinder), Cone (cone), Plane (plane) and Torus (torus). Like other types of primitives, NURBS primitives are usually used as a basis for forming more complex models.[22]

4 Manual BIM geometry

Today the most practical modelling method based on the point cloud design is manual. This method is not entirely practical, which slows down the overall design process. To do this, compare the work of manual and automatic modelling and understand how they work.

This work uses manual modelling using the example of software such as Revit, the latest version of which supports point cloud data. This program is only used for creating solid-state models. This program does not have a function for processing and working with the point cloud. for this purpose, the Cyclone program was used, which is described below.

4.1 Modelling in Revit

Revit is Autodesk's main parametric modelling tool for building design. It consists of design tools for modelling a building in a 3D environment with automatically linked 2D views and data views for non-spatial data, such as cost. Therefore, it has three workflows that support different parts of the design process: architecture, structures, and MEP.

To start, upload the point cloud to Revit in the format (*. rkp). Specify an indexed point cloud project file with the .rcp extension .the. rcp File includes several .rcs-format point cloud files. (Fig.15)[17]

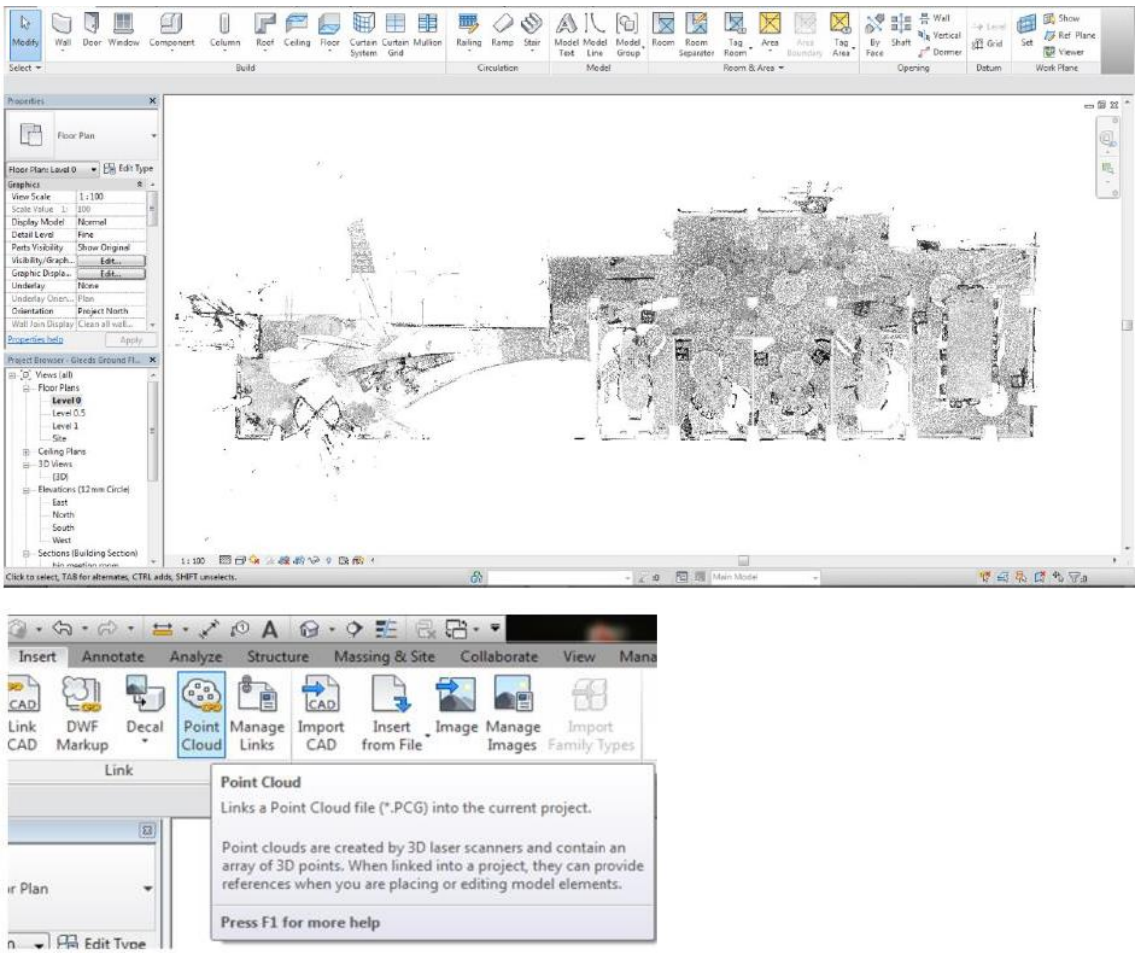


Figure 15. Point cloud to Revit.[17]

Point cloud:

- typically behaves like a model object in Revit;
- displayed in various types of modelling (for example, in 3D, on plans, and in sections);
- split by plans, sections, and 3D view borders, which makes it easy to isolate sections of the cloud.
- there is an opportunity to select, move, rotate, copy, delete, display symmetrically, and so on.
- represents an object that can be snapped to via assumed plane snaps or direct point snaps in the point cloud.

Once the point clouds are loaded in Revit it is useful to set up levels to create 2D cross-sectional floorplans where the floors are as in Figure 16. These levels can also act as modelling constraints to snap geometry start and termination points to. The 2D floorplans that are generated by levels allow for the modelling of floor-to-ceiling elements such as walls and columns as they appear clearly as voids in the point cloud.

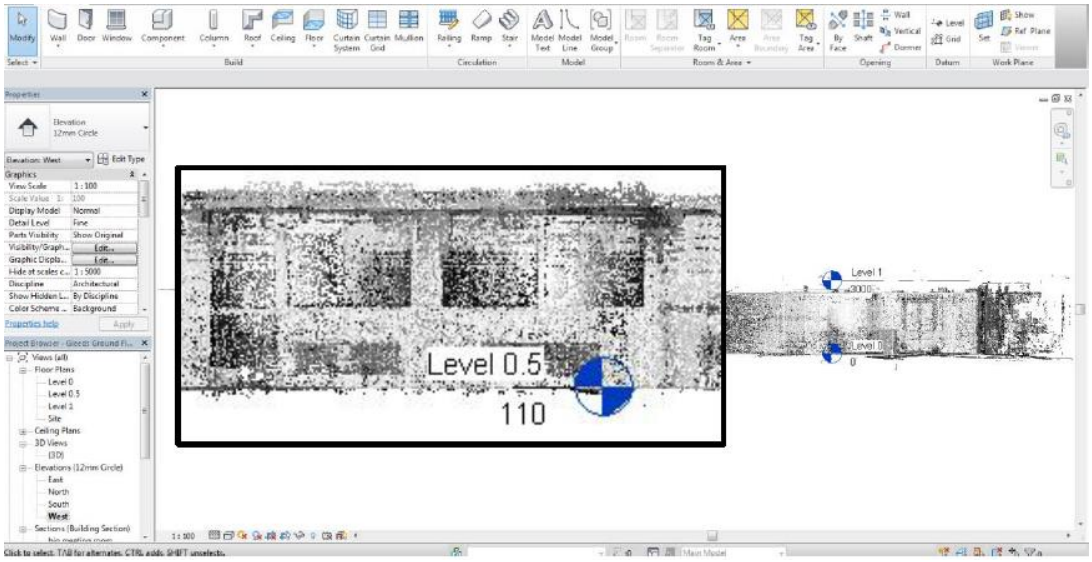


Figure 16. Levels that define floorplan views in Revit; zoomed detail in box.[17]

The modelling involved using stock elements that are part of the standard Revit library to build up the 3D parametric model of the ground floor that had been scanned, the final result of which can be seen in Figure 17. The decision to use stock elements was chosen as this case study focused more on learning the

process of modelling with Revit's new point cloud handling functionality. Also it should be noted that although Autodesk trialled an automated point cloud modelling plugin for Revit, it was pulled and so there is no automated modelling functionality from point clouds in Revit as things stand.

The snap feature makes it easier to create a model based on point cloud data. Tools for creating or changing geometry in Revit (for example, "Wall", "Line", "Grid", "Rotate", and "Move") can be linked to implicit flat surfaces that are dynamically defined in the point cloud, or directly to points in the point cloud.

- Snap to a plane - Revit only detects planes that are perpendicular to the current working plane (in plan, section, or 3D view) and are in close proximity to the cursor. However, after the work plane is detected, it is used as a global reference until the view is zoomed in or out.

- Binding directly to a point-binding directly to the point cloud has a low priority in the binding hierarchy. This means that if a binding to a plane is detected, it will always be displayed first. If you want to skip binding to the plane and use direct binding, press the tab key to go through the possible binding options.

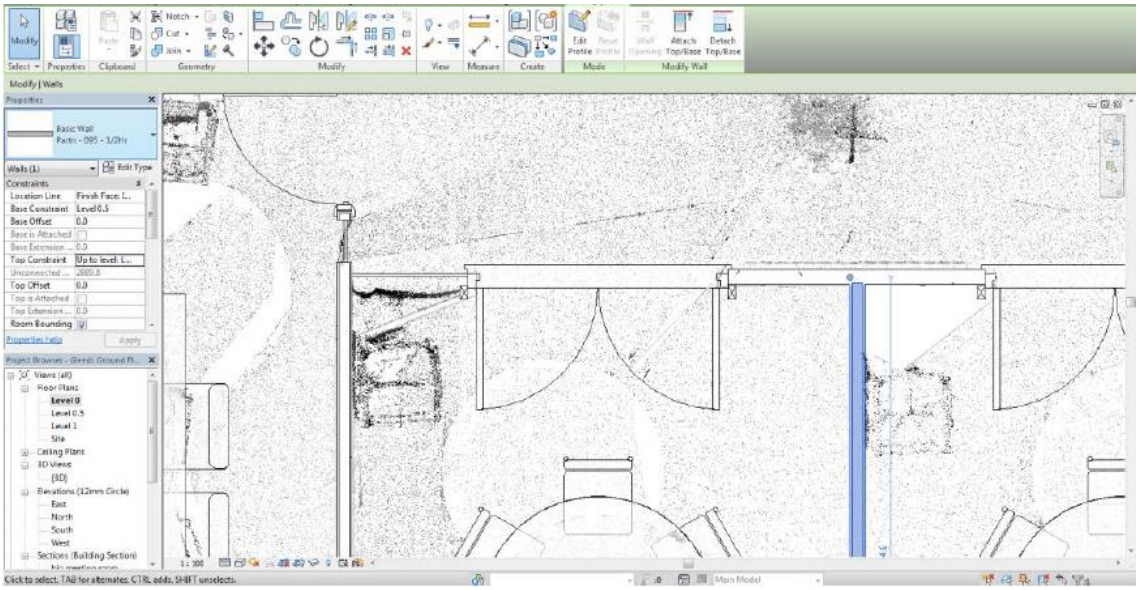


Figure 17. Modelling from a point cloud in Revit plan view. Initial point cloud (top) and with partition wall object being modelled (bottom).[17]

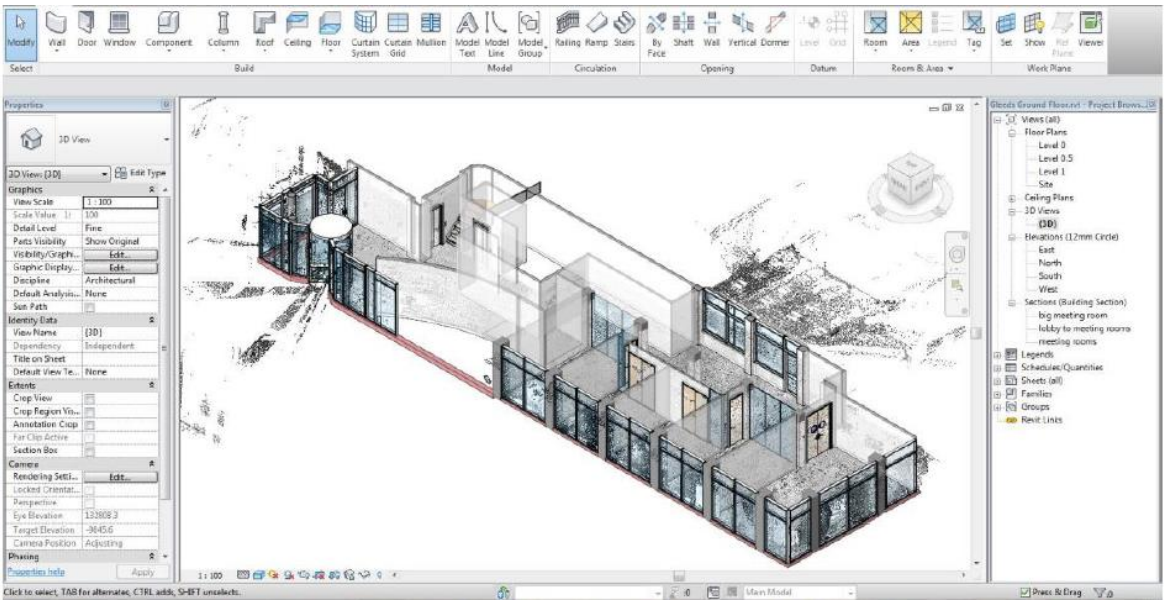


Figure 18. Revit 3D model view with point cloud shown.[17]

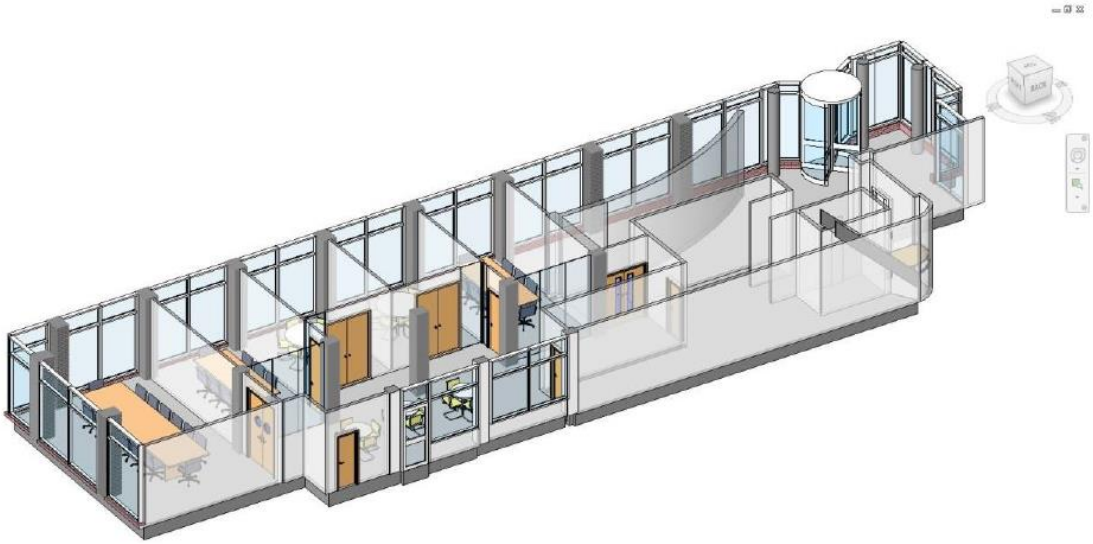


Figure 19. Final model in Revit of Ground Floor Office.[17]

5 Automatic BIM

This thesis considered methods for automating the modelling process almost without using the manual method. Now this task is the most urgent for reverse engineering. There are very few automated solutions of this kind, they are not widely used and their functionality is limited.

Why is the manual work bad? Formally, everything is fine: the problem is solved. However, when working with a large object, a person may simply not notice a relatively small deviation, which, however, will go beyond the tolerance. The second point is time. The third point is the human factor: there are always deviations, the employee will independently determine whether they meet the standards or not, what to do with them. There is always the possibility of an error. The fourth point is that such an analysis cannot be performed on the site; it will require sufficiently productive equipment.

5.1 Modelling in Cyclone

The ideas and approaches taken to aid the problem of geometry construction have two strands: the creation of imagined geometry for virtual worlds and the reconstruction of geometry as it exists in the real world from measured data. Both commercial and academic spheres of research have investigated the automated reconstruction of geometry from point clouds, especially as interior modelling has risen in prominence with the shift to BIM requiring rich parametric models.

Currently, a large number of software products are known for processing the results of ground-based laser scanning and 3-D modelling. Therefore, the task of choosing such programs correctly is relevant. To solve this problem, a number of criteria should be identified, which makes it possible to comprehensively evaluate a particular software:

- Editing point clouds
- Orientation of point models
- Building a Mesh surface
- Creating NURBS (Non-Uniform Rational B-Spline, C-Rep, NURBS) - a universal way to represent spline curves and surfaces in CAD, which has the basic geometric properties of Bezier curves and surfaces and allows (unlike the latter) to accurately represent many canonical curves and surfaces).
- Creating a three-dimensional model using primitives
- Creating profiles and sections
- Creating orthophoto images.

- Texturing

The Cyclone program was chosen for the work, which is currently one of the most widely used universal programs.

5.2 Cyclone

The most versatile program for point cloud processing and scanner management. Cyclone consists of separate modules (Fig.20) embedded in a single software shell. Various modules are designed to solve the individual tasks of the General data processing process of three-dimensional laser scanning. Cyclone-Scan is a module for controlling the scanner operation. The user can configure scanning density, data filtering, create custom macros, and scan.

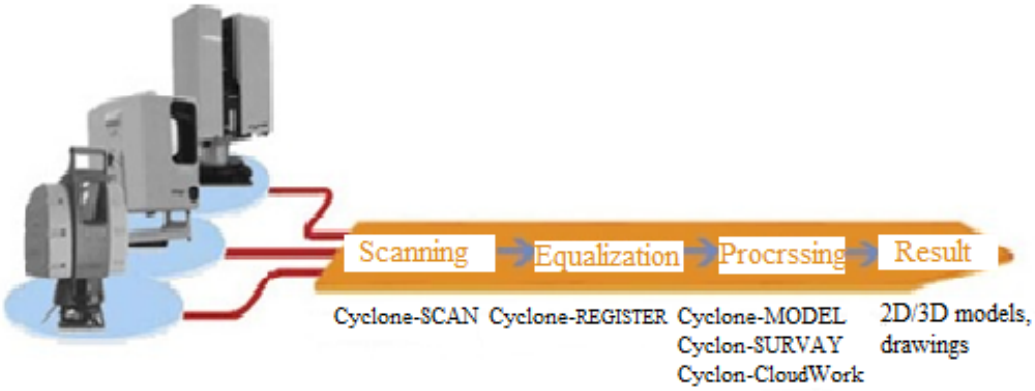


Figure 20. Diagram of the Cyclone program modules.

The Cyclone-Register has all the functions for quick and precise adjustment of the point clouds taken from different stations of the survey. The Cyclone-Register automatically recognizes standard sighting marks, and also links scans by characteristic binding points without sighting marks.

The Cyclone-Model module makes it possible to process point clouds, turning them into objects for export and import into project automation systems (CAD). There are powerful tools to support complex topographic models with the ability to smooth irregular networks of surface triangles, which allows to reduce the amount of data while maintaining accurate geometry. It is quite easy to create sections along a polyline and a line of equal heights (isolines) directly from the point cloud.

5.3 Scanning the object

The scan was performed with an active Leica RTC360 photogrammetric 3D scanner. The construction structure of the farm was chosen as the object under study, since it has many surfaces of regular geometric shape.[23]

Photogrammetric scanning is usually performed on several routes. For correct scanning, you need to move the scanner around the object in a certain range of distance from it. This can be observed in the scanner software window. When you exit the range, an audio signal will sound, in which case you need to zoom in or out of the scanner. Since the object has a fairly complex geometry, it was necessary to make several scans: 2 large scans on both sides of the object, and 3 more for internal and small areas.

As a result of scanning, we have a preliminary visualization of the future model (Fig. 21). It is necessary to process the scans, which are a set of images from the scanner with preliminary internal orientation. To do this, we set the program to perform the following actions sequentially: global registration, precise gluing, and filtering of small objects. All these steps are performed almost automatically. In this software, there is an option also to combine scans and combine them into a single model.

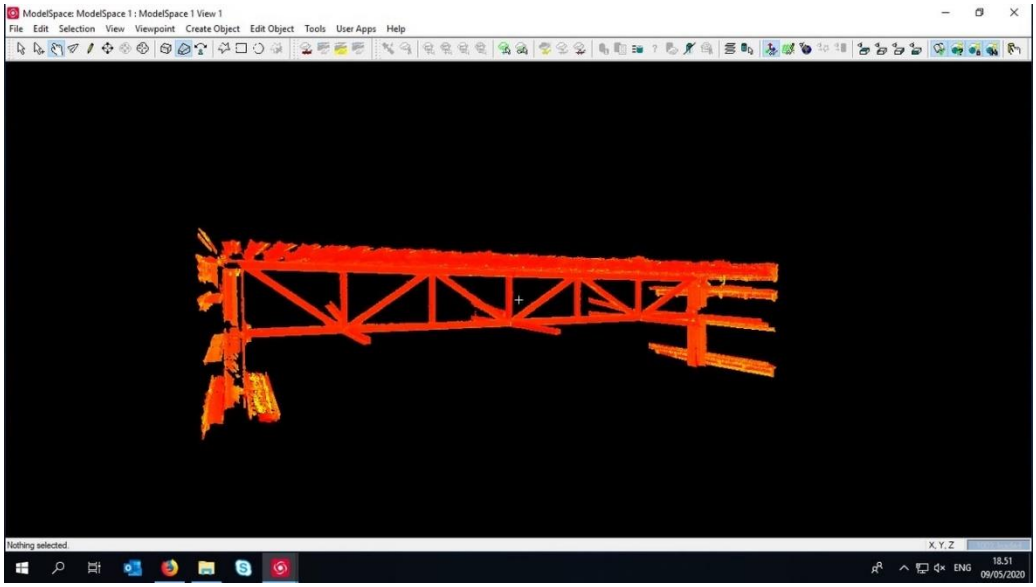


Figure 21. The point cloud in Cyclone.

5.4 Structuring point clouds

Point clouds prodused through the laser scanning process can be either unstructured and structured. In unstructured point clouds, the spatial relationship between points representing real-world objects is not reflected in the structure of the file that contains the point cloud. A comparison with raster models aids understanding here. Raster data models define the spatial proximity of values by using a regular grid, in which the same number of columns and rows are used to store the values in each cell. Unstructured point clouds can be stored in an array that has millions of columns and only one row.

To be able to process point cloud data efficiently and extract information from it, it is essential to structure the data. Spatial proximity operators and analysis are vastly more effective and efficient on structured data. For instance, «nearest neighbor» analysis methods – to analyze epyvariation of the density of the points over a particular space – will execute far more efficiently on structured data than they will on the unstructured data. In the structured data, the space for performing specific processes can be delimited, while with unstructured data, it is necessary to process all of the points sequentially.

5.5 Filtering

As with any measurement device, sensors that generate point clouds may present noise and outliers as false points. Noises and outliers are generated by a combination of many factors, including the type and design of the LiDAR sensor, scanning conditions, and the environment being scanned. Measurement errors, sudden movement of the sensor, geometrical discontinuities due to occlusions, and varying densities across the scanned space are some of the sources contributing to the generation of false points.

Removing this noise and any errant outliers from the meaningful by filtering is essential for accurate, usable results. For filtering methods, it is important to consider the density, depth, distribution of points, and distances between them in clusters. For instance, in distance-based methods, all points with mean distances are outside of an assumed distance mean and standard deviation are considered as outliers and removed from data. Filtration of emissions in the cyclone program

is carried out almost manually, by selecting and deleting points. It is also possible to reduce the number of points automatically using the internal functions of the program.

5.6 Registration in Cyclone

The object is registered in the Cyclone semi-automatically, selecting the necessary sets of scans, after which the model is combined into a full 3D model.

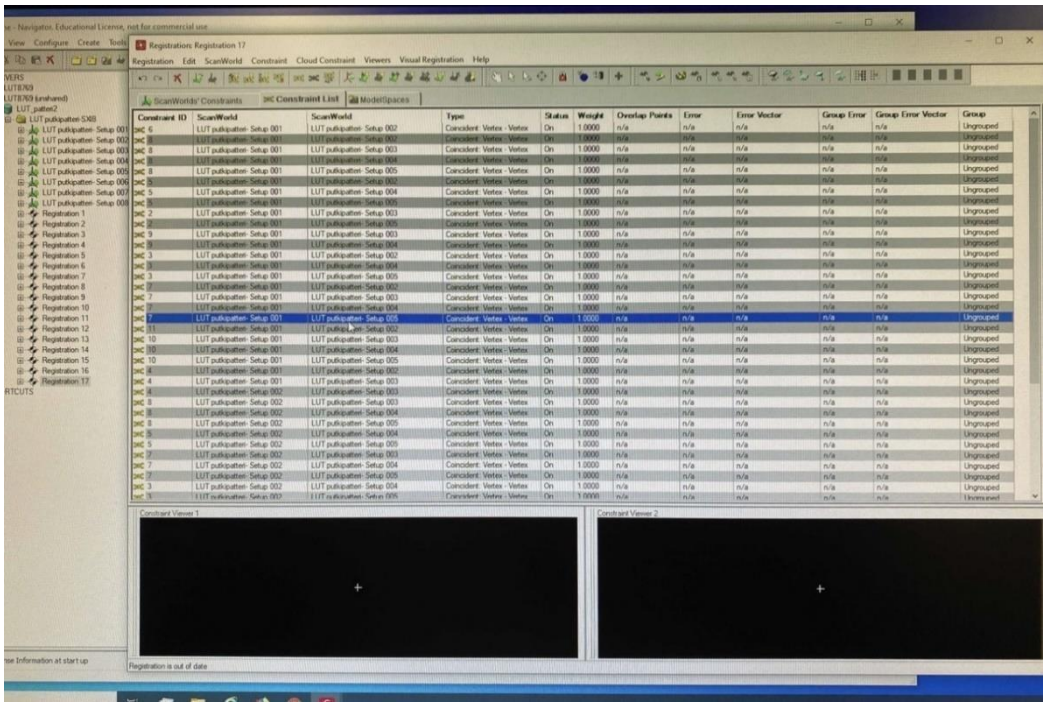


Figure 22. Registration in Cyclone.

5.7 Recognition

Starting the process of recognizing surfaces, edges, and vertexes that describe the body. In the Cyclone software, this process is performed by the “create object: patch" function, which prompts you to select the desired set of points for surface recognition. (Fig. 23).

There is a Function to define a number of standing points. This enables to quickly and accurately create 3D models from point clouds. The function works as follows: one or more points are selected, and then using the algorithm of best placement, the program automatically finds neighboring points that satisfy the

condition of construction of the corresponding figure. Moreover, the algorithm works until the newly found points satisfy the figure's build condition. In this way, you can draw cylinders (pipes), planes, and smoothed surfaces. The algorithm is based on the least-squares method and all statistics (MSE, mean and maximum deviation from the mean, etc.) show the reliability of the model construction.

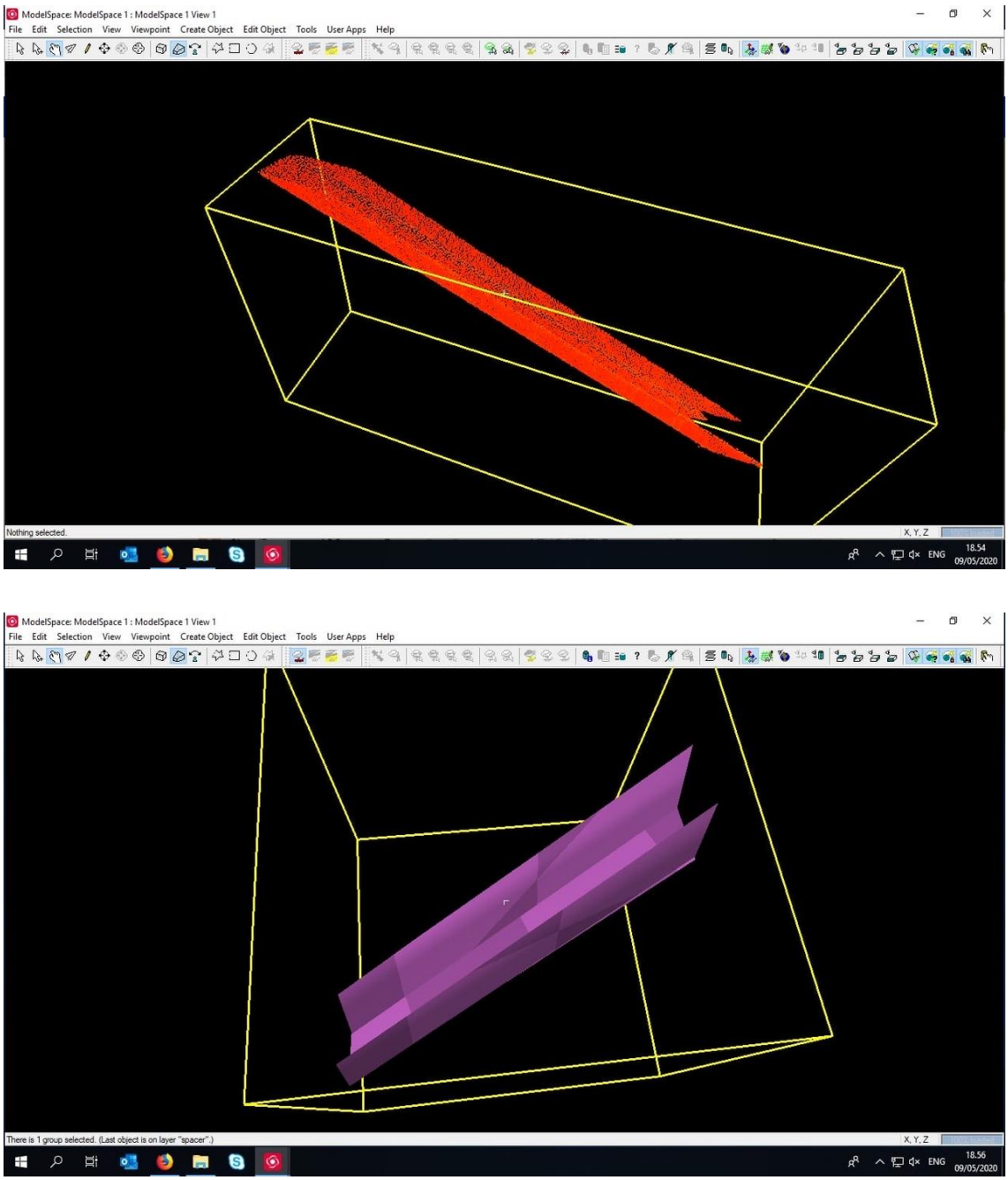


Figure 23. Recognition of surfaces in Cyclone.

The program automatically selects the values of the necessary parameters and as a result, of the recognition process, a good result is obtained. However, after

recognition, you must either extend the surface manually, or create a new set of patches that will be automatically recognized. Thus, to approximate a section of a polygon grid with a surface, only the area that corresponds most closely to the shape being entered is used. After correctly placing all surfaces, and obtaining a closed surface model, it can be saved in one of the many available formats, or transformed into a solid-state CAD model (Fig.24), over which Boolean operations can be performed: subtractions, joins, and intersections. However, the functionality of solid-state modelling in Cyclone is not very large.

Therefore, it is possible to exchange parametric objects between Cyclone and packages of supported CAD applications, such as Tekla or Autodesk Revit, in which you can continue modelling.

5.8 Segmentation

Segmentation into a Cyclone can be done by creating layers for both a set of point clouds and ready-made surfaces, which is most convenient. When creating surfaces, you can combine them into objects, as shown in the figure. To do this, you need to create a new layer, name the object, and combine the set of surfaces into a single group that represents the object. (Fig. 24).

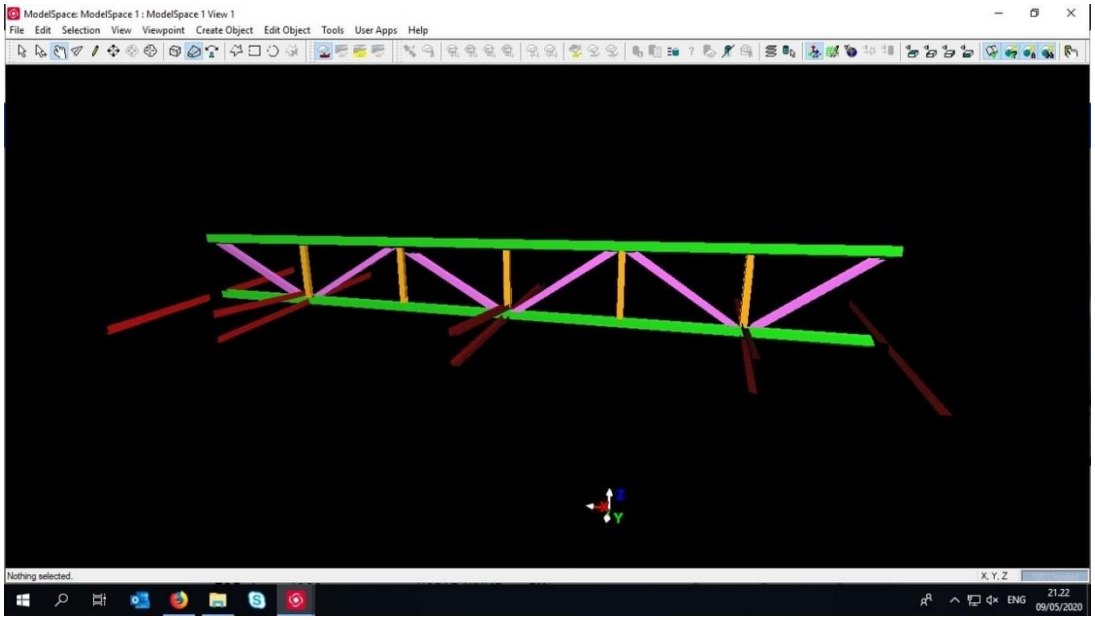


Figure 24. The division of surfaces into objects.

5.9 Modelling

Then the modelling process is transferred to the program, where you need to create a structural model using the original surfaces divided into certain groups. To do this, we will transfer this model in pts. format to Tekla Structures. Here, to define surfaces, you need a plugin that includes learning machine. (Fig. 25).

Tekla Structures software was selected for point cloud conversion and automation. This is convenient due to its automation functionality. It has an open application programming interface (API) - Tekla Open API ™. The Tekla open API provides an interface for third-party applications to interact with model and drawing objects in Tekla structures (Tekla Open API, 2019). The BIM handler module was written using C#. C# is an object-oriented mature programming language that is part of .NET Framework. This language was chosen because it is the most commonly used language in the Tekla Open API environment, although VBA can also be used (Tekla Open API, 2019). However, the structure of the BIM handler has been made so non-specific for the software. To achieve this, the BIM handler module includes a separate submodule that works with Tekla Structures. It was created as its own separate module for easy replacement with other modules for other design software.

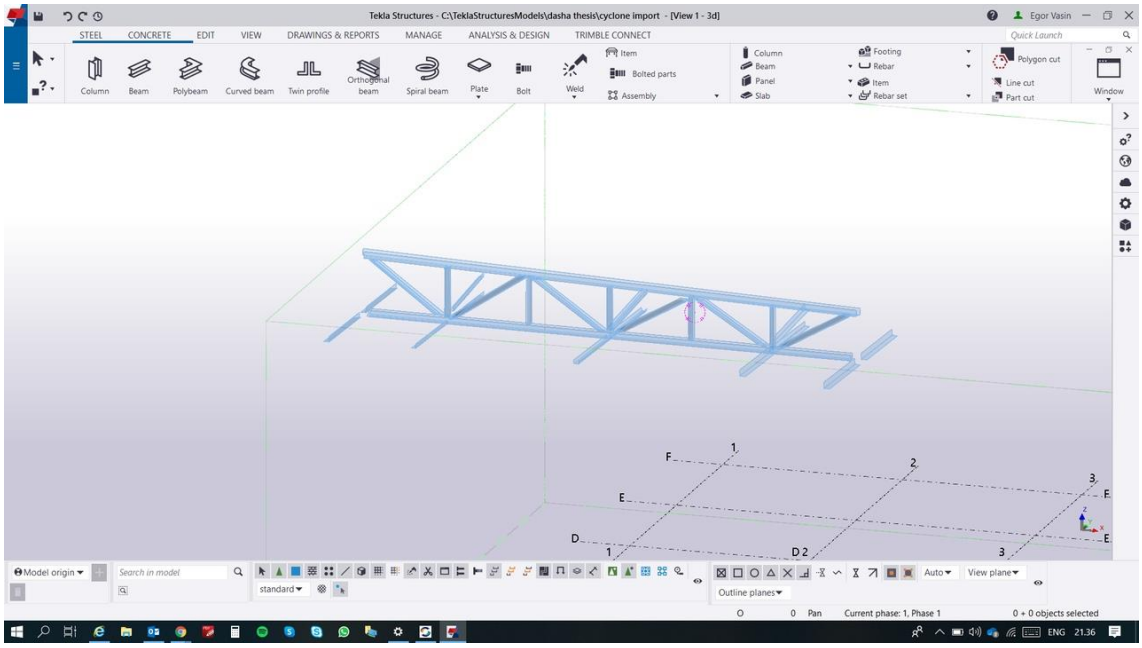


Figure 25. The model transferred into Tekla Structures.

6 Automating BIM geometry in Tekla

There are several types of Tekla Structures expansions, which could be divided into three types (Fig. 26):

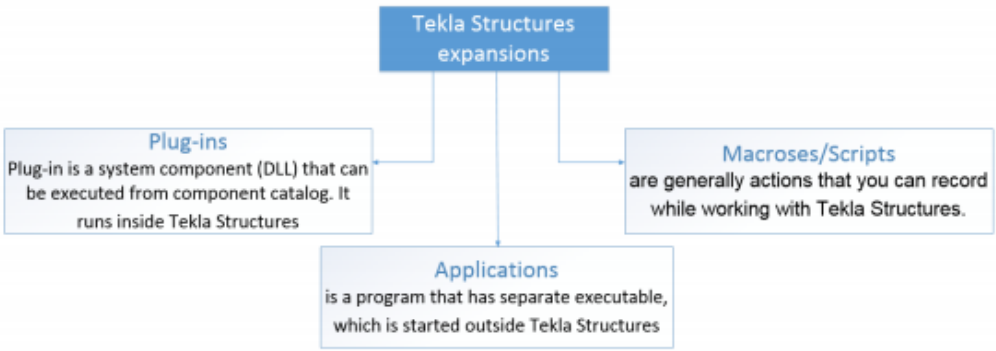


Figure 26.Types of Tekla Structures expansions.

The application is a program that has a separate, which is started outside Tekla Structures. [14] For example a sided application may be a Multi converter to other formats, it can export Tekla Structures objects in multiple file formats: IFC, STEP, IGES, OBJ ,STL, DGN, DWG, DXF, SKP. Objects can be exported by part mark, assembly mark, phase, object ID or for the selection. This extends the basic functionality of Tekla Structures. Plug-in is a system component (DLL) that can be executed from the component catalog. It runs inside Tekla Structures. For example a sided plug-in can help users to create the reinforcement cage for continuous beam or column faster and easily. Macros/Scripts are generally actions that you can record while working with Tekla Structures. Macros are basically C# (.cs) source files that are compiled at run-time, which could be edited; applications and plug-ins are compiled executables or DLLs. All these extensions are able to be formed by Tekla Tekla Open API(Application Programming Interface) [15]. Interface is a common boundary between two entities.Tekla Structures user interface enables human users to communicate with the software.Tekla's Open API, Application Programming Interface, facilitates interaction between Tekla Structures and other software. The Tekla Open API provides an interface for third-party applications to interact with model and drawing objects in Tekla Structures. It also allows the creation of plug-ins.

The main solution is a plug-in for Tekla and therefore relies on most of Tekla's functionality to handle most tasks (including loading the point cloud and geometry library) and essentially just adds some detection and fit algorithms along with a few other tools for scanning processing. The main function is surface fitting, in which the user selects 3 points to define the surface plane from which the area growth algorithm determines extents. The user then sets the tolerance and selects which parametric element of the object type in the Tekla model should be used. There is also the possibility of installing a massive wall, which is a useful way to model the surface of a wall that is not perfectly perpendicular and orthogonal. The disadvantage of this plugin is that it only handles the definition on one surface, which means that to model the entire volume; you must rely on one side of the wall, unless you install a mass wall on each surface and perform a Boolean function to properly merge the two solids.

6.1 Segmentation

After removing false points from clouds, point clouds are subdivided into clusters to extract building components from them. Depending on the level of detail required in a BIM, individual components are to be recognized. For example, in the case of a multi-storey building, each level must be recognized separately to extract the planar view of each storey before further details on each storey can be recognized, such as floors, ceilings, walls, or outlier and doors.

For segmentation and recognition of building components, there are a sufficient number of various algorithms and methods. For example, to identify an object, statistical algorithms are provided with an input point cloud alongside a mathematical model of the object to be recognized. The model is then fitted to the different areas of a point cloud, as the degree of confidence in fitting is evaluated. For example, to find a wall, the model of a plane is provided to the algorithms. The algorithms make a subset of the given point cloud by randomly selecting a fixed number of points from the cloud. They then calculate the coefficients of the shape which best fits the sample of points. Then, they check all the other points in the cloud against this model to separate the inliers and the outliers. This consensus of the entire cloud to a model built from randomly selected points is recorded – an iterative process that is repeated several times.

This yields a shape, which has the highest consensus from all the random samples.

These algorithms are robust and return the best parameters, even when a significant portion of the dataset is outliers. In cases where the wall has a protruding section, or the ceiling has an elevated portion, the algorithm can find planes that fit most of that surface.

One such method is the RANSAC method. RANSAC (RANDOM SAMple Consortium) is a sustainable method of estimating model parameters based on random samples. The advantage of this scheme is its resistance to noise in the original data.

A data processing task often occurs in which you need to define model parameters that must satisfy the source data. The source data can be divided into two types: good points that satisfy the models and noises that are random inclusions in the source data, i.e. outliers.

Consider the simplest example of how the algorithm works 2D a straight line into two points. (Fig.27) Taking the fact that there are emissions among the data, estimating parameters in a standard way, such as the least-squares method, will cause the wrong model to be calculated, as the model is built on the basis of all points. The RANSAC method uses only the two points needed to construct a straight line and builds the model, then checks how many points match the model using the evaluation function with the specified threshold.

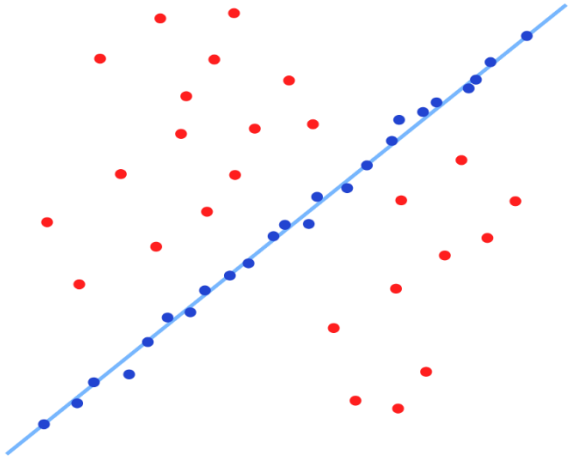


Figure 27. Proposed by the RANSAC algorithm is direct. Emissions do not affect the outcome.

The description of the RANSAC algorithm is found in Appendix 1.

RANSAC is effective with a majority of outliers present and is conceptually simple which leads to its popularity. The main issue with RANSAC is the computational load if no optimizations are applied, especially with reducing the search space of the random sampling, which on large amounts of data can be time-consuming otherwise. [24]

There are also machine learning methods that are used to recognize building components. In these methods, recognition systems are built, so objects are identified by learning from past recognition. In these approaches, models of construction objects to be recognized are stored and marked in the classification database. For building modeling, a classification database is compiled and obtained from CAD or BIM computer-aided design files. Then the appropriate algorithms are used to recognize construction objects that are located in the database.

Once a group of points is recognized as an object, all points that lie on this object are kept, and the remaining points are discarded. As a result, at the end of this step, a set of known objects represented by point-clouds are stored. (Fig.28)

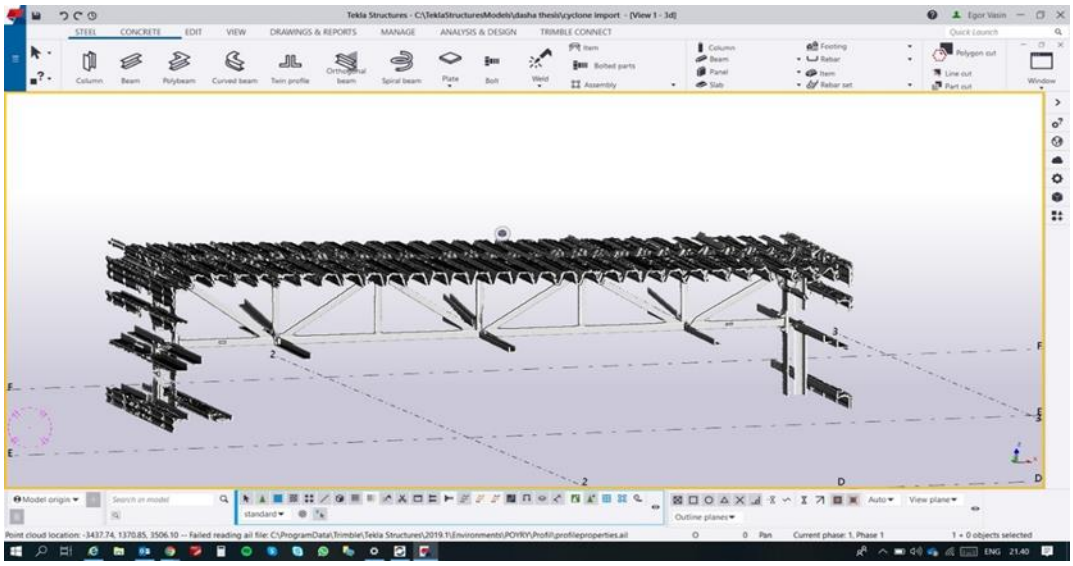


Figure 28.The point cloud into Tekla Structures.

6.2 Reconstruction of the polygon grid

Reconstruction of three-dimensional surfaces that form point samples is a well-known task. It allows you to customize scanned data, fill in surface holes, and remeshing existing models. There is a new approach that expresses surface reconstruction as a solution to the Poisson equation.

Point sampling is often heterogeneous, so this reconstruction of the shape surfaces of oriented points has a number of difficulties in practice. Positions and normals are usually noisy due to incorrect registration and inaccurate sampling. In addition, availability restrictions during scanning may result in some areas of the surface being stripped of data. Given these problems, reconstruction methods attempt to deduce the topology of an unknown surface, accurately match noisy data, and intelligently fill holes without overlapping.

Given a set of 3D points with oriented normals (denoted oriented points in the sequel) sampled on the boundary of a 3D solid, the Poisson Surface Reconstruction method [36] solves for an approximate indicator function of the inferred solid, whose gradient best matches the input normals. The output scalar function, represented in an adaptive octree, is then iso-contoured using an adaptive marching cubes.

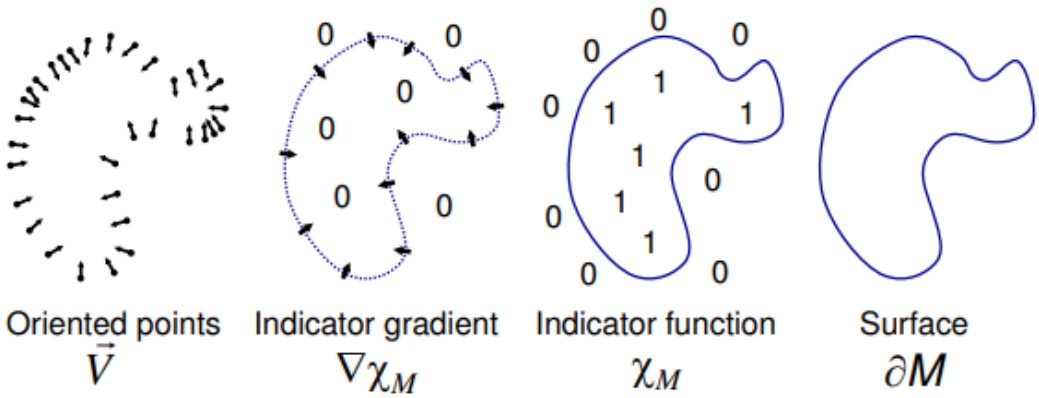


Figure 29. Intuitive illustration of Poisson reconstruction in 2D.

It is not a method that divides the surface into smaller regions, instead, it returns a global solution that considers all data at once, and therefore it is faster. It combines global (for example it solves a problem without resorting to heuristic spatial partitioning) and local fitting schemes. First, we compute indicator function χ^2, which defines whether points are situated inside $(\chi = 1)$ or outside of the model $(\chi = 0)$. (Fig. 29) Because there is an integral relationship between oriented points sampled from the model surface and our indicator function, we can reduce our problem and use samples of the gradient of the indicator function. The gradient of the indicator function is a vector where numbers are zero almost everywhere or they are equal to the surface normals if they are situated near the surface. Finally, we can reconstruct the indicator from this gradient field.[37]

We reconstruct the surface of the model by solving for the indicator function of the shape. (Fig.30)

$$\chi_M(p) = \begin{cases} 1 & \text{if } p \in M \\ 0 & \text{if } p \notin M \end{cases} \quad (1)$$

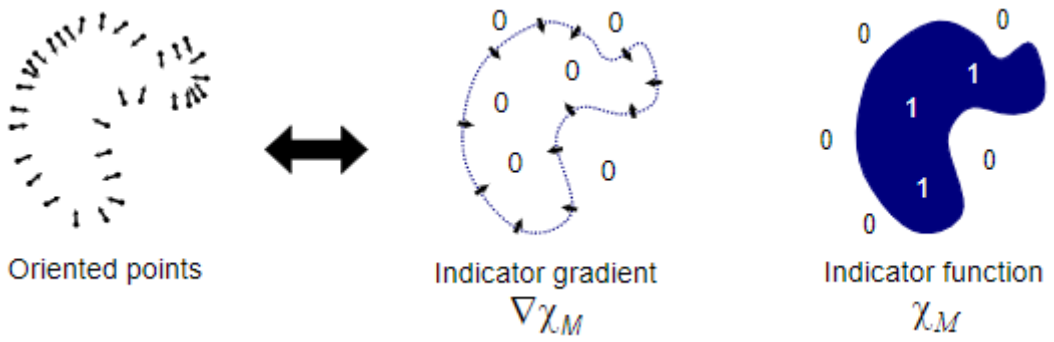


Figure 30. A relationship between the normal field and gradient of the indicator function.

Now we can transform the problem to standard Poisson problem by applying the divergence operator. $\Delta\chi = \nabla \cdot \nabla\chi = \nabla \cdot \vec{V}$ (2) We want to solve function χ^1, where $-\vec{V}$ is the smoothed normal field, which is the result of the computation of its gradient field. The equation is solved only near the surface, which is more efficient.[37]

There is a version of this algorithm that solves for a piecewise linear function on a 3D Delaunay triangulation instead of an adaptive octree. This algorithm takes as input a set of 3D oriented points. It builds a 3D Delaunay triangulation from these points and refines it by Delaunay refinement so as to remove all badly shaped (no isotropic) tetrahedral and to tessellate a loose bounding box of the input-oriented points. The normal of each Steiner point added during refinement is set to zero. It then solves for a scalar indicator function f represented as a piecewise linear function over the refined triangulation. It solves the Poisson equation Δf=div(n) at each vertex of the triangulation using a sparse linear solver. Eventually, the surface mesh generator extracts an isosurface with a function value set by default to be the median value of *f* at all input points. [37]

The Poisson Solution is found in Appendix 2.

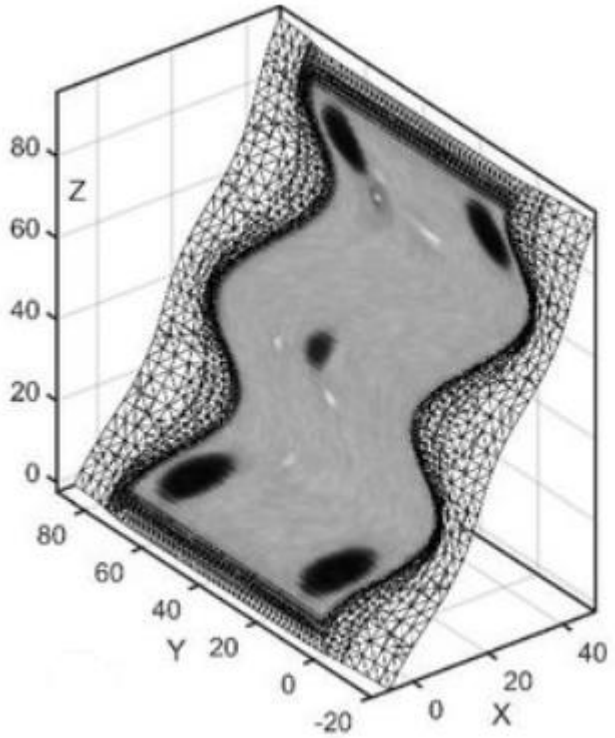


Figure 31. Poisson surface reconstruction.

6.3 The C# implementation

The C# implementation uses the TriangleMesh class introduced for ray tracing and the SparseMatrix class used for mesh parameterization. Here, the Dao structure is used to visualize the radius of triangles in the grid. The radius of polygons is not interpolated to get the vertex radii. The Mesh Triangle class has been extended to include the area and support calculating coefficients using ray tracing. Only new methods are enabled.

The TriangleMesh class algorithm overall Program Structure is found in Appendix 3.

6.4 Conversion of surfaces into objects

As mentioned above, Tekla Structures provides an open API, Tekla Open API, which allows users to create their own plugins and external applications on top of Tekla Structures. Plug-ins are components that are connected to the teklastructures application object model.when any of the component parameters changes, all the objects included in It are recreated according to the new input

data. To work correctly, the plugin must be compiled for a specific version of Tekla, otherwise Tekla will not be able to connect it.

This application is an independent program that can connect to Tekla using TeklaAPI if necessary. For it to work correctly, it must be compiled for a specific version of Tekla. However, external applications require Tekla to be open in order to be able to read and modify models [29]. The Tekla Open API includes eight DLL files. These libraries can be divided into three categories: user interface libraries, modeling libraries, and core libraries. Libraries provide basically the same model editing methods as the Tekla Structures user interface. User interface libraries include tools for creating the user interface and other visual tools for plug-ins and external applications. The dialog box library contains classes and methods for creating a user interface for plug-ins. The catalog library includes functionality for accessing instances of the Tekla Structures catalog, such as the profile or rebar catalog, as well as user interface components that can be used in the plugin UI. The modeling libraries contain classes for handling structural parts in model a. The model library contains classes for structural parts, and they are named the same as in the Tekla Structures user interface. These classes include methods for creating, modifying, and deleting them. In addition, the library provides tools for accessing all objects in model a. The drawing library contains similar classes and methods for processing drawings and objects within them. Finally, the analysis library provides basic classes that you can use to access analytical and project information.

The main libraries are used in other Tekla open API libraries. The Tekla Structures library contains basic and general types that are shared by model and drawing libraries, as well as methods for editing Tekla Structures parameters and environment variables. The plugin library includes functionality for creating plugins and abstracts for classes that need to be inherited in plugins. The data type library contains methods related to various data types that are used in other libraries. The program is an example of a method written in C# that modifies the profiles of selected beams in the Tekla Structures model to demonstrate the open Tekla API. The example contains three parts: getting the selected beams, changing the profiles of these beams, and returning the value if all the beams

were changed successfully. To get the selected objects, a ModelObjectSelector object is created when using method, all the selected objects are obtained from the object. If the object has the value beam, the profile changes according to the specified profile string. If the modification was not successful, information about it is stored in a variable named beamsProfilesModifiedSuccessfully. (Fig. 32).

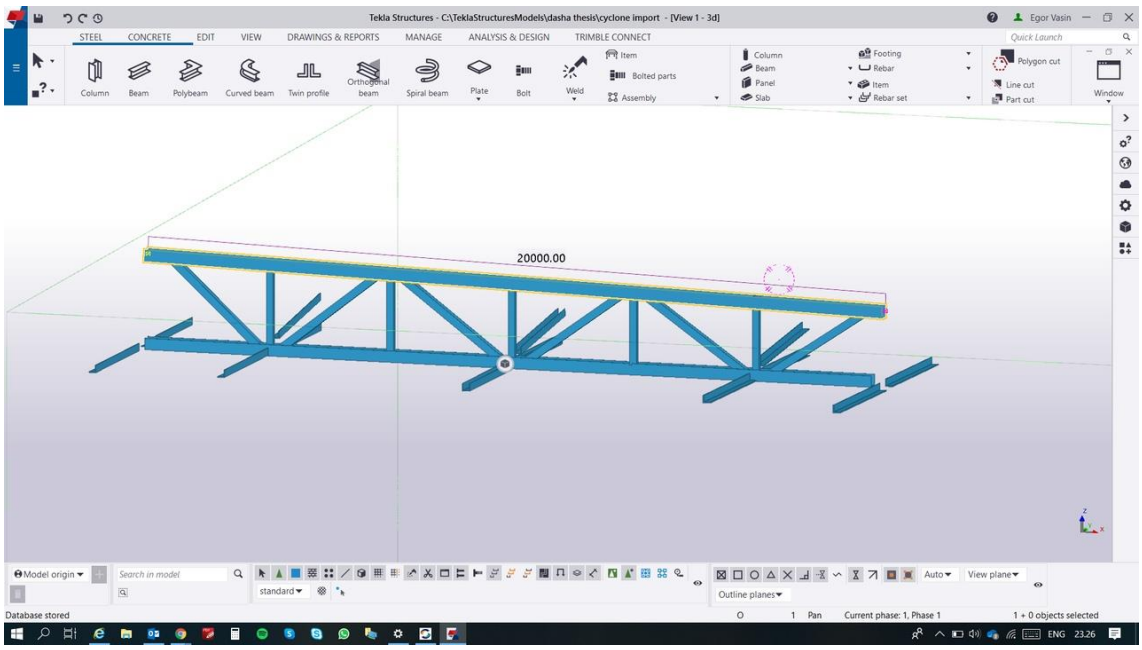


Figure 32. The surfaces converted to objects into Tekla Structures.

A machine learning project usually contains 4 main actions: acquisition, preparation, process, and report. These steps for implementing the machine learning module are described in the following chapters.

All information collected from the beams was collected and stored in a database. Then the desired data set was created. After that, the data set was preprocessed in a format that machine learning algorithms can use. The processing phase is the actual work performed by the algorithm. Ultimately, all information is delivered to users. Understanding input and output data is more important in any software system, and machine learning is no exception than knowing what happens in between. [34] after understanding the input and output data and finding the necessary bundle, it becomes easier to analyze all approaches and algorithms suitable for this situation and evaluate the necessary characteristics.

Thus the input signal to the algorithm is a ray that was obtained from other models for the program to represent an object with input data. Each ray in the model was added as a new instance containing the data received from them. Information collection has been automated to improve the user experience. All information received by the user is divided into two parts. The first is information about the most suitable beams. And the second information is necessary for modeling these beams.

Information for finding a particular profile beam has two main elements: geometry and BIM properties. Geometry includes relationships between structural elements, including information such as position, orientation, and profile. BIM properties include properties derived from the BIM model, such as material. To reconstruct a beam, the plugin needs to know all the object properties that are necessary for successful modeling. Since the goal was to make no difference between a custom modeled object and an automatically modeled object, it was not possible to exclude any properties.

Data preparation consisted of three stages: selecting the appropriate data, pre-processing the data, and converting the data. Here we used only the relevant properties of the beams in comparison with the creation of the database. Based on this analysis, feature vectors from the beam regions were constructed. These vectors contained geometric information about the design elements and properties of BIM. These properties cover all the necessary rules that were used to select the beams that will be implemented in the structural model. It is accepted that machine learning algorithms can only work with numeric data. But today there are quite a lot of algorithms for different types of data. However, data with mixed string and numeric values needs to be pre-processed, and usually string values are converted to numeric values. As explained earlier, Euclidean space was chosen for use with the algorithm and therefore for the need for transformation.The algorithm had one goal: to predict the best fit profile and orientation of the beam. To find the most appropriate beams, information about how the object is modelled and what parts are used in it does not matter. Instead of including dimension information, the most appropriate beam profile can be found by the outline of their surfaces. As mentioned above, the NN approach is

well suited for finding the best matches. For this reason, the closest neighbors (k-NN) were used in this study, where the Euclidean distance metric is used to measure distances. k-NN is a classification algorithm that classifies each example based on the majority of k nearest neighbors in the training set. [34] This method is widely used in various fields, such as pattern recognition and data mining. [30]

According to Hastie et al. (2009), NN methods use these observations in the training set closest in input space to$x$ to generate a forecast of output Y. k-NN can be defined as follows [31]:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \quad (3)$$

where $Nk(x)$ is the neighborhood of x defined by the k closest points to xi in training sample.

The main benefit to use k-NN is its simplicity and efficiency. While being one of the simplest machine learning algorithms, it can be very powerful in incorrect situations. Furthermore, k-NN has shown remarkable performances on data with a large example size. [32] However, selecting the k may affect the performance. [32] The algorithm was implemented with C#. C# is one of the most popular programming languages for scientific computing and it is an appealing choice for data analysis. [33] In this study, machine learning algorithms were created in Visual Studio. Visual Studio is a C# module that includes a wide range of machine learning algorithms for both supervised and unsupervised problems. [33]

The k-NN classification algorithm overall program structure is found in Appendix 4.

7 Future development

Thus, there are several ways to continue research on this topic, and they can be divided into three groups: testing the Tekla plugin in real projects with the accompanying use of Cyclone, continuing to develop plugins using various coding methods, and expanding the scope of the plugin for other types of constructs. The most natural way to continue research is to test the Tekla plugin

in real projects with cloud processing in cyclone. By comparing the design and its effectiveness between a typical design process and a design process that uses a plug-in, you can say that this really increases the speed of modeling. Improving design efficiency as well as improving design quality are the main reasons for introducing new design methods, and it is very important that these methods actually improve design without compromising efficiency or quality. In this paper, we have considered two types of point cloud design: manual modeling and automated modeling using programs and additional developments. The manual process is the most commonly used and easy to use. This process does not require knowledge of third-party programs and additional developments. Huge disadvantages of this process: the need for a large amount of time and the relative accuracy of building structures in comparison with the real object. The automated process greatly simplifies the work and reduces design time and increases the accuracy of modeling. The Cyclone software simplifies the tasks of registering segmentation and filtering point clouds. This program is perfect for these functions. However, the process of converting point clouds to objects is not accurate enough. This process is still almost manual, with the exception of the patch-based simulation function. Plug-ins will significantly speed up and facilitate the modeling process. This will also improve the accuracy of the designed structures. The disadvantages of this method are the need for sufficient knowledge and experience of coding using mathematical methods. You also need to create multiple plugins for different types of construction, or make the plugin more complex by combining all the necessary methods. This is quite a difficult task. In any case, plug-in development can continue. New functions and methods can be added. Many algorithms and methods for converting point clouds are currently known. They should be actively developed and used for various types of construction, various structures and materials. You can also create methods for both segmentation and point cloud filtering, as well as create a simpler method for displaying the grid and converting surfaces to objects. You can also create plug-ins based on solid-state modeling. Finally, it is worth investigating whether this approach works with other BIM programs than Tekla Structures. It can be assumed that there are no barriers to using the same approach with other BIM programs, since the plug-in methods were built to be independent of the BIM software. However, most of these potential future studies

are natural pathways for development, and once promising results are obtained, they are likely to be widely applied.

8 Conclusion

As previous research has shown, automation using machine learning can be implemented in BIM processes. The purpose of this study was to consider plug-ins as a new method for more efficient design in BIM models using point clouds as input data. The proposed approach was based on searching for a better and more automatic design method from the point cloud. The goal was also to determine whether the point cloud can be converted to a structural model using machine learning plug-ins and Tekla Structures. The study showed that there are quite a few methods and depending on the complexity of the project, there is a choice. Machine learning and plug-ins are the most automated, but the difficulty lies in writing plug-ins for different project variants. The results show that the strategy provides a new solution for the scope of BIM applications in the field of building design. The methods proposed in this thesis and the codes are examples for further work in this field. Based on the lack of experience in coding, it was not possible to achieve absolute success. This work requires a wider range of knowledge and experience. It is also necessary to analyze and compare all possible and available encoding methods. Based on this work, it is possible to delve into the nuances of coding and achieve the desired results. The observed and potential advantages of point cloud modelling using methods make this topic worthy of further study, since the results are expected to improve the traditionally low productivity of the construction industry.

Figures

References

1. Tekla Structures [web source] URL:www.tekla.com (date of reference 20.12.2019)

2. Help. Tekla Structures 2019

3. Дико С.М. Базы данных. Проектирование и создание: Учебно-методический комплекс – Изд.центр ЕАОИ.2008

4. Назаров А.С. Фотограмметрия: учеб. пособие для студентов вузов – ТетраСистемс, 2006

5. Besl, P.J., McKay, N.D., 1992. A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14, 239–256. doi:10.1109/34.121791, pp. 239–256.

6. Pfeifer, N., Böhm, J., 2008. Early stages of LiDAR data processing, in: Li, Z., Chen, J., Baltsavias, E. (Eds.), Advances in Photogrammetry, Remote Sensing and Spatial Information Sciences. pp. 169–184.

7. Mitra, N.J., Gelfand, N., Pottmann, H., Guibas, L., 2004. Registration of point cloud data from a geometric optimization perspective, in: Scopigno, R., Zorin, D. (Eds.), Proceedings of the 2004 Eurographics, p. 22.

8. Besl, P.J., McKay, N.D., 1992. A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14, pp. 239–256

9. Barnea, S., Filin, S., 2013. Segmentation of terrestrial laser scanning data using geometry and image information. ISPRS Journal of Photogrammetry and Remote Sensing 76, pp. 33–48

10. Mortenson, M.E., 1997. Geometric Modelling, 2nd Ed. ed. John Wiley & Sons, Inc., New York, NY, USA.

11. Larsen, K.E., Lattke, F., Ott, S., Winter, S., 2011. Surveying and digital workflow in energy performance retrofit projects using prefabricated elements. Automation in Construction 20, pp. 999–1011

12. Tekla Open API. Self Learning Exercises. Tekla Structures 2019; Trimble Solutions Corporation.

13. Tekla open API [web source] URL:https://www.tekla.com/openAPI, (date of reference 20.12.2019) the article in the compilation.

14. Tekla Open API. Developer's Guide. Tekla Structures 2019; Trimble Solutions Corporation.

15. Tekla Structures 2019. Open API Release Notes; Trimble Solutions Corporation

16. Hoover, A., Jean-Baptiste, G., Jiang, X., Flynn, P.J., Bunke, H., Goldgof, D.B., Bowyer, K., Eggert, D.W., Fitzgibbon, A., Fisher, R.B., 1996. An experimental comparison of range image segmentation algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 673–689

17. Capturing and Processing Survey Data Towards Automation for High Quality 3D Models to Aid a BIM Process, Charles Patrick Hugo Thomson.2016

18. Sareen, K.K., Knopf, G.K., Canas, R., 2011. Hierarchical data clustering approach for segmenting colored three-dimensional point clouds of building interiors.

19. Boundary Representation Modelling Techniques 2006th Edition, Ian Stroud

20. Response Surface Methodology: Process and Product Optimization Using Designed Experiments, 4th Edition, Raymond H. Myers, Douglas C. Montgomery, Christine M. Anderson-Cook

21. Solid-State Physics: Introduction to the Theory, James Patterson, Bernard Bailey

22. Distance Calculation Between a Point and a NURBS Surface, Defense Technical Information Center, 2000

23. Leica Geosystems, 2014. Leica CloudWorx for Revit. URL: http://www.leica-geosystems.com/

24. Schnabel, R., Wahl, R., Klein, R., 2007. Efficient RANSAC for Point-Cloud Shape Detection. Computer Graphics Forum 26, pp.214–226.

25. Poux F., Hallot P., Neuville R., R. Billen, 2016. Smart point cloud: definition and remaining challenge. pp.20-21

26. Barnea, S., & Filin, S., 2013. Segmentation of terrestrial laser scanning data using geometry and image information. pp. 33-48.

27. Brodu, N., Lague D., 2012. 3D Terrestrial LiDAR data classification of complex natural scenes using a multi-scale dimensionality criterion: applications in geomorphology, pp.121-134.

28. Farella, E.M, 2016. 3D Mapping of underground environments with a hand-held laser scanner. Proc. SIFET annual conference, Lecce, Italy.

29. Tekla Structures Glossary. Retrieved form Tekla Structures Support website: https://teklastructures.support.tekla.com/system/files/files/TeklaStructuresGlossary.pdf, accessed on December 17, 2019.

30. Shakhnarovich, G., Darrell, T. and Indyk, P. (2006). Nearest-Neighbor Methods in Learning and Vision: Theory and Practice. MIT Press.

31. Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction (2nd ed.). Springer.

32. Zhang, S., Li, X., Zong, M., Zhu, X., and Cheng, D. (2017). Learning k for kNN Classification, ACM Transactions on Intelligent Systems and Technology, 8 (3), Article 43.

33. Murphy, K. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

34. Witten, I. H., Frank, E., and Hall, M. A. (2011). Data Mining : Pratical Machine Learning Tools and Techniques (3rd ed.). Elsevier.

35. GRINSPUN E., KRYSL P., SCHRÖDER P.: Charms: a simple framework for adaptive simulation. In SIGGRAPH (2002), pp.281–290.

36. Poisson Surface Reconstruction. Michael Kazhdan, Matthew Bolitho and Hugues Hoppe. Eurographics Symposium on Geometry Processing (2006).

37. KAZHDAN M.: Reconstruction of solid models from oriented point sets. SGP (2005), 73–82.

38. Fischler, M.A., Bolles, R.C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24, 381–395.

Appendix 1. The description of the RANSAC algorithm.

To the input of the algorithm comes:[38]

1. the set of source data *X*
2. a set of source data 2. a function *M* that allows you to calculate the parameters *Θ* of the model *P* from a set of data from *n* points
3. the function of evaluating *E* correspondence of points of the obtained model
4. threshold *t* for the evaluation function
5. number of iterations of the method *k*

The entire algorithm consists of a single loop, each iteration of which can be logically divided into two stages.

1. The first stage — the choice of points and the calculation model.
   - From the set of source points *X*, n different points are randomly selected.
   - Based on the selected points, the parameters *Θ* of the model *P* are calculated using the function *M*. the constructed model is usually called a hypothesis.
2. The second stage is the verification of the hypothesis.
   - For each point, its compliance with this hypothesis is checked using the evaluation function *E* and the threshold *t*
   - Each point is marked with an inlayer or outlier
   - After checking all points, it checks whether the hypothesis is the best at the moment, and if it is, it replaces the previous best hypothesis.

At the end of the cycle, the last best hypothesis is left.

The results of this method are:

1. The parameters Θ of model P

2. Source data Points marked with inlayers or outliers.

Assessment of source data:

The value of the t parameter should be determined depending on the specific requirements that depend on the data, in most cases, only after experimental evaluations. The number of iterations k can be determined prior to executing the algorithm using the theoretical estimation method. [38]

Let p be the probability that the RANSAC algorithm at some iteration, choosing n points on which to build the model, will take only inlayers from the original data set for calculations. In this situation, the model based on these points is likely to be accurate. Based on this, we can use probability p to estimate the accuracy of the algorithm. Let ω be the probability of selecting one inlayer from the total number of points, i.e. $\omega = I / T$, where I is the number of inlayers T is the total number of points. In most cases, the proportion of inlayers ω is unknown before the algorithm starts running, but it is almost always possible to give some rough estimate. The probability of an independent selection of n inlayers from the source data, in this case, is $q = C_I^n / C_T^n = I!(T-n)! / (T!(I-n)!)$, and the probability that at least one point from the set is an outlier, i.e. that an incorrect model will be constructed is $(1 - q)$. The probability that the algorithm will never select n inlayers during k iterations is $(1 - q)^k$. this situation means that the exact model will not be built, and the probability of this event is $(1 - p)$. [38] Thus

$$(1 - p) = (1 - q)^k$$

Express the number of iterations we need k

$$k = \frac{log(1-p)}{log(1-q)}$$

Appendix 2. The Poisson Solution.

Having defined the vector field, $\vec{V}$ we would like to solve for the function $\tilde{\chi} \in \mathscr{F}_{\mathcal{O},F}$ such that the gradient of $\tilde{\chi}$ is closest to $\vec{V}$, i.e. a solution to the

Poisson equation $\Delta\tilde{\chi} = \nabla \cdot \vec{V}$. One challenge of solving for $\tilde{\chi}$ is that though $\tilde{\chi}$ and the coordinate functions of $\vec{V}$ are in the space $\mathscr{F}_{\mathscr{O},F}$ it is not necessarily the case that the functions $\Delta\tilde{\chi}$ and $\nabla \cdot \vec{V}$ are. To address this issue, we need to solve for the function $\tilde{\chi}$ such that the projection of $\Delta\tilde{\chi}$ onto the space $\mathscr{F}_{\mathscr{O},F}$ is closest to the projection of $\nabla \cdot \vec{V}$. Since, in general, the functions F_o do not form an orthonormal basis, solving this problem directly is expensive. However, we can simplify the problem by solving for the function $\tilde{\chi}$ minimizing:

$$\sum_{o \in \mathscr{O}} \left\| \langle \Delta\tilde{\chi} - \nabla \cdot \vec{V}, F_o \rangle \right\|^2 = \sum_{o \in \mathscr{O}} \left\| \langle \Delta\tilde{\chi}, F_o \rangle - \langle \nabla \cdot \vec{V}, F_o \rangle \right\|^2. \tag{1}$$

Thus given the $|\mathscr{O}|$- dimensional vector v whose *o*-th coordinate is $v_o = \langle \nabla \cdot \vec{V}, F_o \rangle$, the goal is to solve for the function $\tilde{\chi}$ such that the vector obtained by projecting the Laplacian of $\tilde{\chi}$ onto each of the F_o is as close to v as possible. To express this in matrix form, let $\tilde{\chi} = \sum_o x_o F_o$, so that we are solving for the vector $x \in \mathbb{R}^{|\mathscr{O}|}$. Then, let us define the $|\mathscr{O}| \times |\mathscr{O}|$ matrix L such that Lx returns the dot product of the Laplacian with each of the F_o. Specifically, for all $o, o' \in \mathscr{O}$, the (o, o') - th entry of L is set to

$$L_{o,o'} \equiv \left\langle \frac{\partial^2 F_o}{\partial x^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial y^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial z^2}, F_{o'} \right\rangle \tag{2}$$

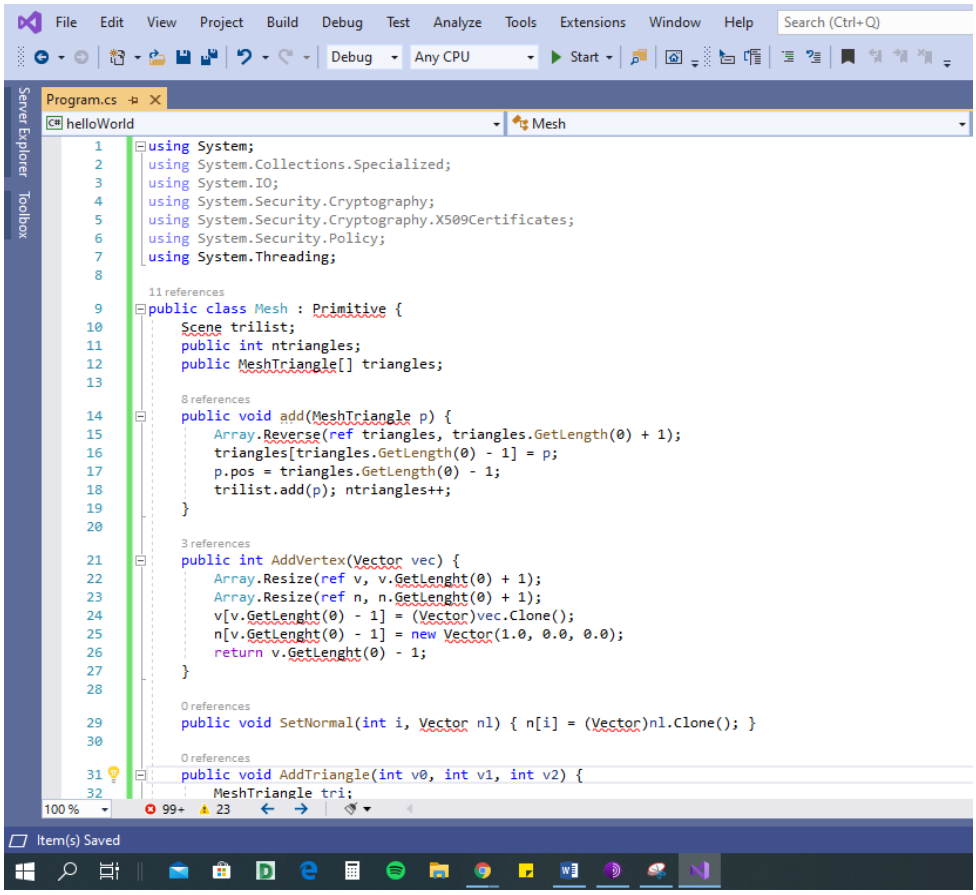
Thus, solving for $\tilde{\chi}$ amounts to finding

$$\min_{x \in \mathbb{R}^{|\mathscr{O}|}} \|Lx - v\|^2.$$

Note that the matrix L is sparse and symmetric. (Sparse because the F_o are compactly supported, and symmetric because $\int f''g = -\int f'g'$.) Furthermore,

there is an inherent multiresolution structure on $\mathscr{F}_{\mathcal{O},F}$, so we use an approach similar to the multigrid approach in [35], solving the restriction L_d of L to the space spanned by the depth d functions (using a conjugate gradient solver) and projecting the fixed-depth solution back onto $\mathscr{F}_{\mathcal{O},F}$ to update the residual.[36]

Appendix 3. The Mesh Triangle class algorithm. Overall program structure.



```
0 references
public void AddTriangle(int v0, int v1, int v2) {
    MeshTriangle tri;
    tri = new MeshTriangle(v[v0], v[v1], v[v2], n[v0], n[v1], n[v2]);
    add(tri);
}

0 references
public void SetTex(int i, Vector n) { //not used for radioity }

    public Mesh() {
        trilist = new SceneTree();
        triangles = new MeshTriangle[0];
        ntriangles = 0;
        v = new Vector[0]; n = new Vector[0];
    }

    public void PrepareForIntersect() { trilist.prepare_for_render(); }

    public void Mesh(Mesh m) {
        v = (Vector[])m.n.Clone(); n = (Vector[])m.n.Clone();
        trilist = (Scene)m.trilist.Clone();
        ntriangles = m.ntriangles;
        triangles = (MehTriangle[])m.triangles.Clone();
    }

    public void rcenter() {
        int i, n;
        Vector c = new Vector(0.0, 0.0, 0.0);
        v0 = triangles[i].v0; v1 = triangles[i].v1; v2 = triangles[i].v2;
        e1 = v1 - v0; e2 = v2 - v1; e3 = v0 - v2;
        if ((e1.norm() >= e2.norm()) && (e1.norm() >= e3.norm()))
        {
            v = (triangles[i].v1 + triangles[i].v0) / 2.0;
            p = m.AddVertex(v);
            tri = new MeshTriangle(v1, v2, m.v[p]); m.add(tri);
```

```
            v = (triangles[i].v1 + triangles[i].v0) / 2.0;
            p = m.AddVertex(v);
            tri = new MeshTriangle(v1, v2, m.v[p]); m.add(tri);
            tri = new MeshTriangle(m.v[p], v0, v1); m.add(tri);
        }
        else if ((e2.norm() >= e1, norm()) && (e2.norm() >= e3.norm()))
        {
            v = (triangles[i].v2 + triangles[i].v1) / 2.0;
            prop = m.AddVertex(v);
            tri = new MeshTriangle(v1, v2, m.v[p]); m.add(tri);
            tri = new MeshTriangle(m.v[p], v0, v1); m.add(tri);
        }
        else {
            v = (triangles[i].v2 + triangles[i].v1) / 2.0;
            prop = m.AddVertex(v);
            tri = new MeshTriangle(v1, v2, m.v[p]); m.add(tri);
            tri = new MeshTriangle(m.v[p], v0, v1); m.add(tri);
        }
    }
    return mbox;
}

//same as subdivide, but only subdivide triangles
//large than the current minimum area triangle
0 references
public Mesh EqualizeArea() {
    int i;
    double miniarea = 1e10;
    double checkarea = 1e12;
    double area;
    for (i = 0; i < ntriangles; i++)
    {
        area = triangles[i].area();
        if (area < miniarea) miniarea = area;
    }
    Mesh m = SubdivideArea(miniarea);
```

```
            if (area < miniarea) miniarea = area;
        }
        Mesh m = SubdivideArea(miniarea);
        while (checkarea > miniarea)
        {
            checkarea = -1.0;
            for (i = 0; i < m.ntriangles; i++)
            {
                area = m.triangles[i].area();
                if (area > checkarea) checkarea = area;
            }
            if (checkarea > miniarea) m = m.SubdivideArea(miniarea);
        }
        return m;
    }

    0 references
    public Mesh EqualizeArea(double maxarea) {
        double minarea = maxarea;
        double checkarea = 1e12;
        double area;
        Mesh m = SubdivideArea(minarea);
        while (checkarea > minarea)
        {
            checkarea = -1.0;
            for (int i = 0; i < m.ntriangles; i++)
            {
                area = m.triangles[i].area();
                if (area > checkarea) checkarea = area;
            }
            if (checkarea > minarea)
                m = m.SubdivideArea(minarea);
        }
        return m;
    }
```

```
    }

    0 references
    public Mesh SubdivideFrea(double minarea) {
        int i, p;
        MeshTriangle tri;
        Vector v0, v1, v2, e1, e2, e3, v;
        Mesh m = new Mesh();
        m.v = (Vector[])this.v.Clone(); m.n = (Vector[])this.n.Clone();
        for (i = 0; i < ntriangles; i++) {
            v0 = triangles[i].v0; v1 = triangles[i].v1; v2 = triangles[i].v2;
            if (triangles[i].area() <= minarea) {
                tri = new MeshTriangle(v0, v1, v2); m.add(tri);
                continue;
            }
            e1 = v1 - v0; e2 = v2 - v1; e3 = v0 - v2;
            if ((e1.norm() >= e2.norm()) && (e1.norm() >= e3.norm())) {
                v = (triangles[i].v1 + triangles[i].v0 / 2.0);
                p = m.AddVertex(v);
                tri = new MeshTriangle(v1, v2, m.v[p]); m.add(tri);
                tri = new MeshTriangle(m.v[p], v2, v0); m.add(tri);
            } else if ((e2.norm() >= e2.norm()) && (e2.norm() >= e3.norm())) {
                v = (triangles[i].v2 + triangles[i].v1 / 2.0);
                p = m.AddVertex(v);
                tri = new MeshTriangle(v2, v0, m.v[p]); m.add(tri);
                tri = new MeshTriangle(m.v[p], v0, v1); m.add(tri);
            } else {
                v = (triangles[i].v0 + triangles[i].v2 / 2.0);
                p = m.AddVertex(v);
                tri = new MeshTriangle(v0, v1, m.v[p]); m.add(tri);
                tri = new MeshTriangle(m.v[p], v1, v2); m.add(tri);
            }
        }
        return m;
    }
```

```
162     public Vector[] v, n;
163
        0 references
164     public override object Clone() { object o = new Mesh(this); return o; }
165
        0 references
166     public override string ToString() { return "Primitive::Mesh()"; }
167 }
168
169 Application.SetCompatibleTextRenderingDefault(false);
        0 references
170 Application.Run(new MainForm());
171 }
172
        0 references
173     public void IterateRadiosity() {
174     int i;
175     if ((!working) && (radiosity1 = null)) {
176         working = true;
177         for (i = 0; i < m.ntriangles; i++) {
178             radiosity[i] = radwork[i]; radwork[i] = emittance[i];
179         }
180         radmatrix.GaussSeidel(radwork, 10, 1e-3, radiosity);
181         for (i = 0; i < m.ntriangles; i++) radiosity[i] = radwork[i];
182         working = false;
183     }
184 }
185
    0 references
186 public void DisplayCallback() {
187     int i;
188     angel += 0.5;
189     IterateRadiosity();
190     GI.glClear(GI.GL_COLOR_BUFFER_BIT | GI.GL_DEPTH_BUFFER_BIT);
191     if (working) { Glut.glutSwapBuffers(); return; }
192     GI.glColor3d(1.0, 1.0, 1.0);
```

```
    0 references
186 public void DisplayCallback() {
187     int i;
188     angel += 0.5;
189     IterateRadiosity();
190     GI.glClear(GI.GL_COLOR_BUFFER_BIT | GI.GL_DEPTH_BUFFER_BIT);
191     if (working) { Glut.glutSwapBuffers(); return; }
192     GI.glColor3d(1.0, 1.0, 1.0);
193     GI.glMatrixMode(GI.GL_MODELVIEW);
194     GI.glLoadIdentity();
195     GI.glTranslated(0.0, 0.0, -2.5);
196     GI.glRotated(angle, 0.0, 1.0, 0.0);
197     if (wireframe) GI.glPolygonMode(GI.GL_FRONT_AND_BACK, GI.GL_LINE);
198     else GI.glPolygonMode(GI.GL_FRONT_AND_BACK, GI.GL_FILL);
199     if (m != null) ;
200     for (i = 0; i < m.ntriangles; i++) {
201         GI.glBegin(GI.GL_POLYGON);
202         if (wireframe) GI.glColor3d(1.0, 1.0, 1.0);
203         else GI.glColor3d(radiosity[i], radiosity[i], radiosity[i]);
204         GI.glVertex3d(m.triangles[i].v0.z
205             m.triangles[i].v0.y, m.triangles[i].v0.z);
206         GI.glVertex3d(m.triangles[i].v1.z
207             m.triangles[i].v1.y, m.triangles[i].v1.z);
208         GI.glVertex3d(m.triangles[i].v2.z
209             m.triangles[i].v2.y, m.triangles[i].v2.z);
210         GI.glEnd();
211     }
212     Glut.glutSwapBuffers();
213 }
214
    0 references
215 public void IdleCallback() {
216     if (!working) Glut.glutPostRedisplay();
217     else System.Threading.Thead.Sleep(250);
218     if (quitting) Glut.glutLeaveMainLoop();
219 }
```

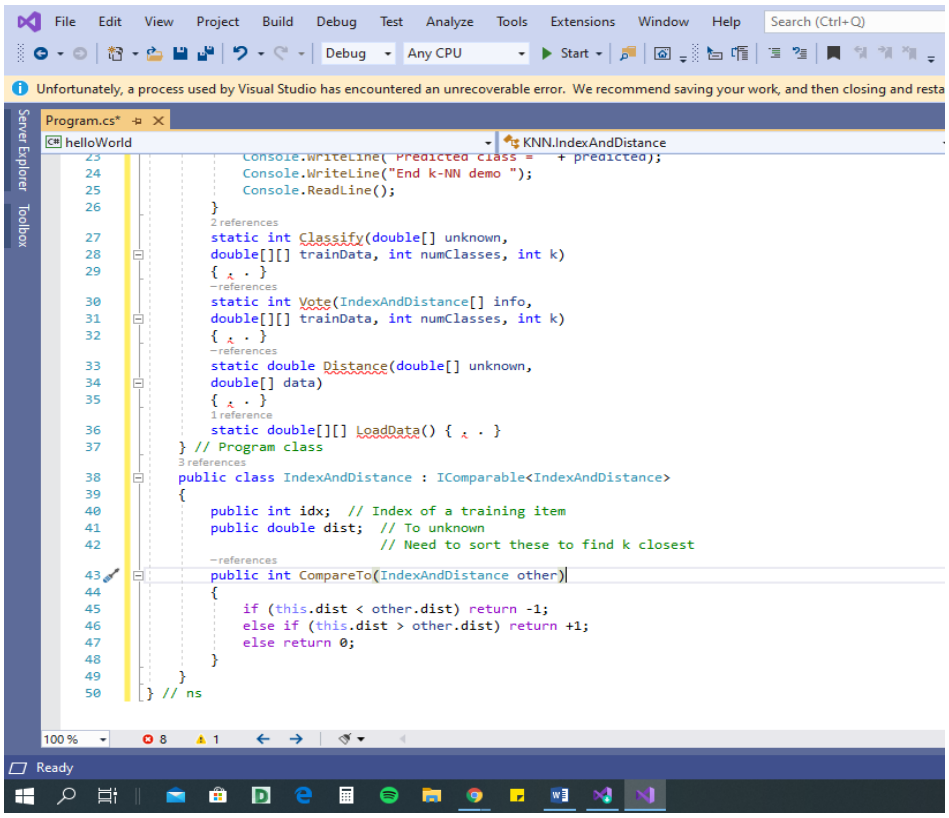
```
0 references
public void IdleCallback() {
    if (!working) Glut.glutPostRedisplay();
    else System.Threading.Thead.Sleep(250);
    if (quitting) Glut.glutLeaveMainLoop();
}
0 references
public MainForm()
{
    InitializeComponent();
    thread = new System.Threading.Thread(this.ThreadStart);
    thread.Start();
}

0 references
void MainFormFormClosed(object sender, FormCloseEventArgs e)
{
    quitting = true; Thread.Join();
}

0 references
void InitRadiosity() {
    int i;
    Random rnd = new Random();
    emittance = new double[m.ntrianhles];
    radiosity = new double[m.ntrianhles];
    radwork = new double[m.ntrianhles];
    for (i = 0; i < m.ntrianhles; i++) emittance[i] = 0.0;
    emittance[rnd.Next() % m.ntrianhles] = initial_power;
    for (i = 0; i < m.ntrianhles; i++) radwork[i] = emittance[i];
    radmatrix.GaussSeidel(radwork, 10, 1e-3);
    wireframe = false;
}

0 references
void CmdLoadClick(object sender, EventArgs e)
```

```
0 references
void CmdLoadClick(object sender, EventArgs e)
{
    working = true;
    openFileDialog1.Filter =
        "OFF (*.off)|*.off|Wavefront OBJ(*.obj)|*.obj|All files(*.*)|*.*";
    if (openFileDialog1.ShowDialog() == DialogResult.OK) {
        m = GeometryFile.Load(openFileDialog1.FileName);
        m.recenter(); m.rescale();
    }
    radiosity = null; wireframe = true; working = false;
}

0 references
void ComputerFormFactor() {
    double vij, fij, r, cosphi_i, cosphi_j;
    int i, j, k, hits;
    IntersectInfo info;
    MeshTriangle tri;
    Vector o,d;
    int ntri = m.ntriangles;
    formfactors = new SparseMatrix(ntri, ntri);
    radmatrix= new SparseMatrix(ntri, ntri);
    m.PrepareForIntersect();
    for (i = 0; i < ntri; i++) {
        ConsoleWrite("Form factor{0}/{1}\r", i + 1, ntri);


        void CmdRestarClick(object sender, EventArgs e)
        { working = true; InitRadiosity(); working = false; }

        void TxtRhoTextChanged(object sender, EventArgs e)
        { Double.TryParse(txtPower.Text,out rho); }

        void TxtRhoTextChanged(object sender, EventArgs e)
```

```
        void CmdRestarClick(object sender, EventArgs e)
        { working = true; InitRadiosity(); working = false; }

        void TxtRhoTextChanged(object sender, EventArgs e)
        { Double.TryParse(txtPower.Text,out rho); }

        void TxtRhoTextChanged(object sender, EventArgs e)
        { Double.TryParse(txtRho.Text, out initial_power); }

        void CmdSubdivClick(object sender, EventArgs e)
        {
            working = true; radiosity = null;
            m = m.Subdivide(); working = false; wireframe = true;
        }

        void CmdAreaClick(object sender, EventArgs e)
        {
            working = true; wireframe = true;
            Mesh newm = m.EqualizeArea(area);
            m = newm; working = false; radiosity = null;
        }

        void TxtAreaTextChanged(object sender,EventArgs e)
        { Double.TryParse(txtArea.Text, out area); }
    }
}
```

Appendix 4. The k-NN classification algorithm. Overall program structure.

```
using System;
namespace KNN
{
    class KNNProgram
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Begin k-NN classification demo ");
            double[][] trainData = LoadData();
            int numFeatures = 2;
            int numClasses = 3;
            double[] unknown = new double[] { 5.25, 1.75 };
            Console.WriteLine("Predictor values: 5.25 1.75 ");
            int k = 1;
            Console.WriteLine("With k = 1");
            int predicted = Classify(unknown, trainData,
            numClasses, k);
            Console.WriteLine("Predicted class = " + predicted);
            k = 4;
            Console.WriteLine("With k = 4");
            predicted = Classify(unknown, trainData,
            numClasses, k);
            Console.WriteLine("Predicted class = " + predicted);
            Console.WriteLine("End k-NN demo ");
            Console.ReadLine();
        }
        static int Classify(double[] unknown,
        double[][] trainData, int numClasses, int k)
        { . . }
        static int Vote(IndexAndDistance[] info,
        double[][] trainData, int numClasses, int k)
```


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Any CPU
Start
Unfortunately, a process used by Visual Studio has encountered an unrecoverable error. We recommend saving your work, and then closing and resta
Program.cs*
helloWorld
KNN.IndexAndDistance
Console.WriteLine("End k-NN demo ");
Console.ReadLine();
}
2 references
static int Classify(double[] unknown,
double[][] trainData, int numClasses, int k)
{ . . }
-references
static int Vote(IndexAndDistance[] info,
double[][] trainData, int numClasses, int k)
{ . . }
-references
static double Distance(double[] unknown,
double[] data)
{ . . }
1 reference
static double[][] LoadData() { . . }
} // Program class
3 references
public class IndexAndDistance : IComparable<IndexAndDistance>
{
public int idx; // Index of a training item
public double dist; // To unknown
// Need to sort these to find k closest
-references
public int CompareTo(IndexAndDistance other)
{
if (this.dist < other.dist) return -1;
else if (this.dist > other.dist) return +1;
else return 0;
}
}
} // ns
100 %
8
1
Ready
Server Explorer
Toolbox