Room-Element-Aggregation to Enhance the Quality of Observed 3D Building Information

Christian MANTHE and Christian CLEMEN, Germany

Key words: room topology, room element aggregation

SUMMARY

This paper shows how an intuitive way of modeling a building with respect to a described room-elements-aggregation algorithm leads automatically to a building {*face-room*}-graph. As a result of creating that graph rooms and building elements (parts of the wall-space) are accessible. The accessibility of the 3-dimensional elements relieves to add other descriptive information to enhance the level of detail or informative quality of a building model.

If the geometric description of a building is considered the accessibility of building elements enables to add relative observations through walls like the wall-thickness to fit a room in a building model in an over determined way. The thickness of a wall can be estimated or measured as relative information form a layman. With that information we can reduce the effort to get a controlled model.

To handle the geometry of an over determined building model a given surface-based parameterization model *POPA3d* (**P**lanes from **O**bservations in a **P**robabilistic data model made for the **A**djustment of **3D** building models information model) developed at the Chair of Engineering Surveying and Adjustment Techniques (Technische Universität Berlin) can be used.

Room-Element-Aggregation to Enhance the Quality of Observed 3D Building Information

Christian MANTHE and Christian CLEMEN, Germany

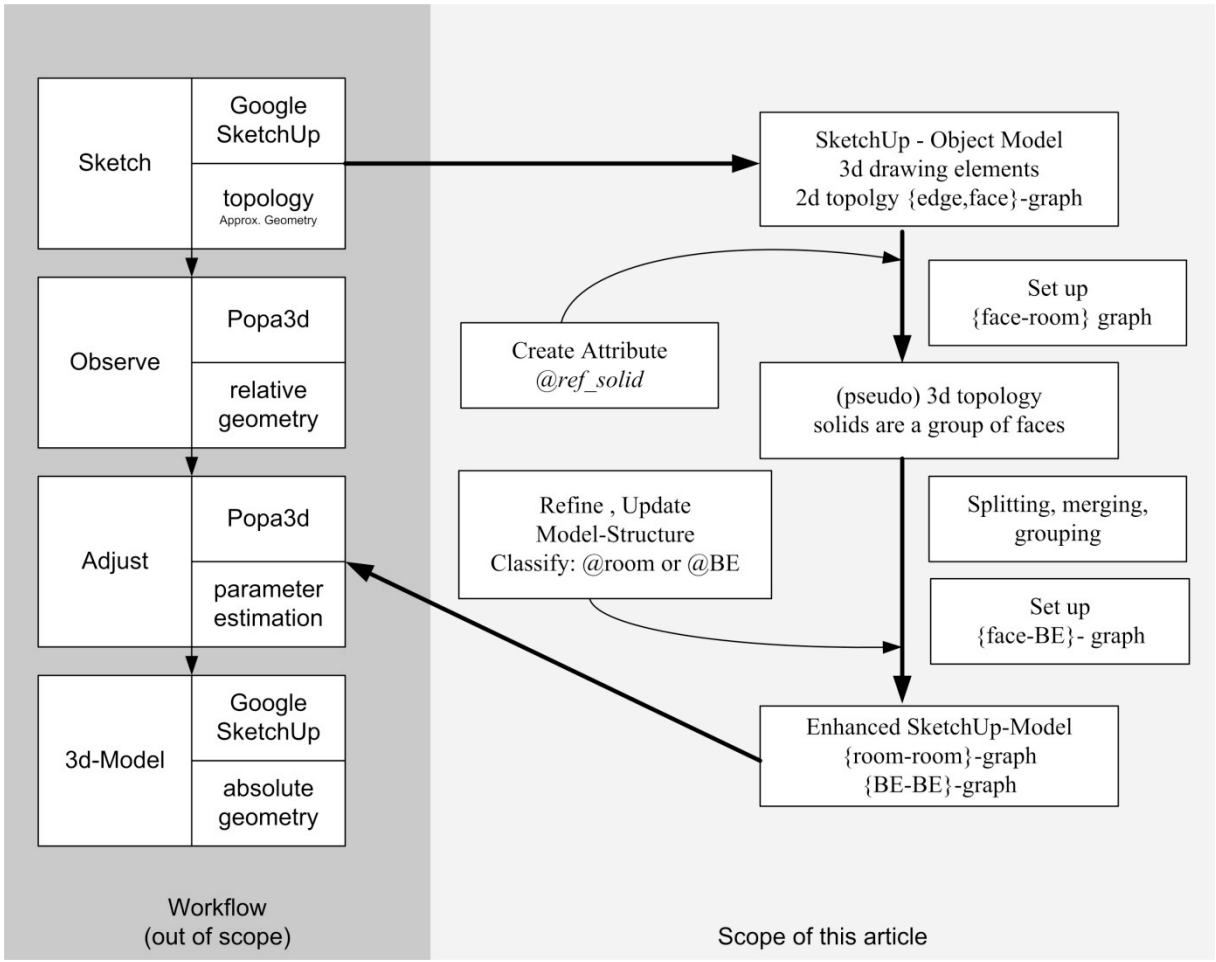
1. INTRODUCTION

Digital Building Models provide the base information for planning, constructing, renovating and maintenance of buildings. In practice, the geometric information is edited within a CAD-software which stores the digital drawing in a layered structure. Measures (relative geometry) are derived from the drawing (absolute geometry) automatically as a computational result. Alphanumeric data are stored separately. This procedure contrasts with an engineering surveyor's way of thinking: "Geometry" is more than just the position of a drawing element. Geometry depicts position and shape of an object and provides an appropriate reference frame for any other type of building information. For technically reliable as-built documentations the (geometric) building survey is very important. The survey of a building consists of observed (not calculated) measures, implicit or explicit constraints (like parallelism and perpendicularity) and the datum (relation to a coordinate system).

Popa3d is an abbreviation for "**P**lanes from **O**bservations in a **P**robabilistic data model made for the **A**djustment of **3d** building models". *Popa3d* provides a novel method, that reverses the traditional work flow sketch→observe→adjust→3D model and an engineering surveyor's way of thinking is expressed in both, data model and software. The process starts with a three-dimensional sketch that defines the topological building structure. Observations, measured between topological elements, are modeled stochastically and are attached to the sketch. These observations provide the basis for the calculation of the building's geometry (absolute geometry). This computation uses **L**east **S**quares **A**djustment (LSA). In a last step the datum, a fixation to the reference frame is specified.

Based on the research of Dr.-Ing. Frank Gielsdorf at the Chair of Engineering Surveying and Adjustment Techniques (Technische Universität Berlin) *Popa3d* constitutes a new conceptual and logical data model. The central idea of this method is, that the survey is not only stored for data capture but is an integral part of the building model throughout the whole life-cycle. The data model connects relative and absolute geometry via the topological elements of the building structure. Resulting in fewer parameters for depicting the absolute geometry, not point-coordinates but surface-parameters parameterize the geometry. These surface-parameters can be calculated on demand (estimated) with a LSA at any time. The LSA, as the central algorithm, is designed as a Gauss-Helmert Model and implemented and tested in newly developed software. For the practical work-flow it is important to automatically detect invalid elements or relations in the 3d model and visualize the located inconsistencies to the user.
With the novel method the time-consuming work of data acquisition for an as-built documentation can be achieved more efficiently and reliable. Moreover it supports the validation of existing 3d building models with stochastic analysis.

The concepts of Popa3d are described in (Gielsdorf & Gründig, 2002) (Clemen & Gielsdorf, 2008)



This paper addresses the challenge to model a three-dimensional topology by creating, merging, editing solids in a building model. As stated above the workflow of data capture begins with a three-dimensional sketch of the building structure. For this task the three-dimensional drawing tool Google SketchUp is used. The surveyor uses Graphical User Interface (GUI) of SketchUp as a digital, three-dimensional field book on site.

The SketchUp off-the-shelf model consists of vertices, edges and faces. While the geometry is represented in three dimensions, using x,y,z – coordinates in a local reference system, the topology is only two-dimensional. This has two main drawbacks:

- *Rooms* are not identifiable. A room, in this sense, is an accessible, visible and closed part of an building

- *Building Elements* (BE) are not identifiable. A Building Element (BE) represents a physical part of the building construction, made out of a material and may be pre-fabricated or build during construction or renovation.

Both, Rooms and Building Elements are geometrically modeled as a set of polygons. The article at hand describes a way, how to enable SketchUp to store and edit a 3d-topology. The aim is to establish a {face-room}-graph and {face-BE}-graph automatically or, if this is not possible, semi-automatically.

The developed concepts are implemented with Ruby (Thomas, 2009) – Plug-ins and SketchUp API (Google) that can be easily deployed in both, SketchUp graphical user interface (GUI) and document object model (DOM). This work mainly uses two techniques in order to extend the SketchUp functionality:

- *Modifying* the values of a given set of entities (coordinates of edges or faces and built in properties) with Ruby scripting.
- *Adding* problem related (dynamic) attributes with Ruby scripting. Dynamic attributes are designed by a third party developer (we) but are loaded, displayed and stored by the SketchUp Application. A dynamic attribute consists of a triple *{@Dictionary, @Name, @Value}* and may be attached to any build-in entity type like vertex, edge or face.

2. THE OBJECT MODEL

2.1 Sketch Up off-the-shelf model

The classical problem of 3d-CAD is how to insert a 3d-situation on a 2d-screen. The SketchUp GUI extensively uses the concept of sweeping. The operator starts to draw 2d lines or polygons in an arbitrary (drawing-) plane and then, afterward, extrudes this figure perpendicular to the (drawing-) plane at a certain shifting value.

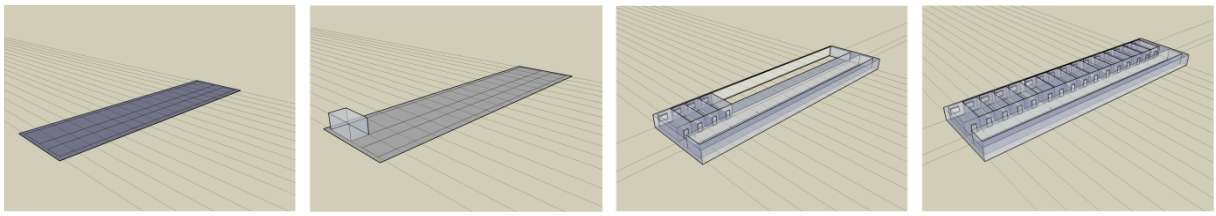


Figure 1 drawing in three dimensions (construction lines, sweep, copy)

Since positioning the 2d-mouse in the 3d-scene is not easy to achieve, the operator can help himself by adding construction lines to the drawing or using CAD-snapping techniques for the three-dimensional positioning of the drawing elements. As the model becomes more complex and doors or windows are added to the sketch, the probability of inserting invalid drawing elements increases. Google SketchUp supports several validation rules that are useful for three-dimensional field books. Self intersecting polygons are detected and corrected automatically, for instance. However additional functionality should be added.

2.2 Checking the structure of the drawing elements

In order to grade up the application to a useful 3d-fieldbook, a plug-in was implemented that visualizes and labels invalid drawing elements. The following validation rules for the building model are checked

- a vertex is connected to three edges
- a vertex is connected to three linear independent faces (planes)
- an edges is bounded by two different vertices (ensured by SketchUp)
- an edge is linked to two linear independent faces (planes)
- a face is bounded by at least three edges.
- valid Euler-Characteristic

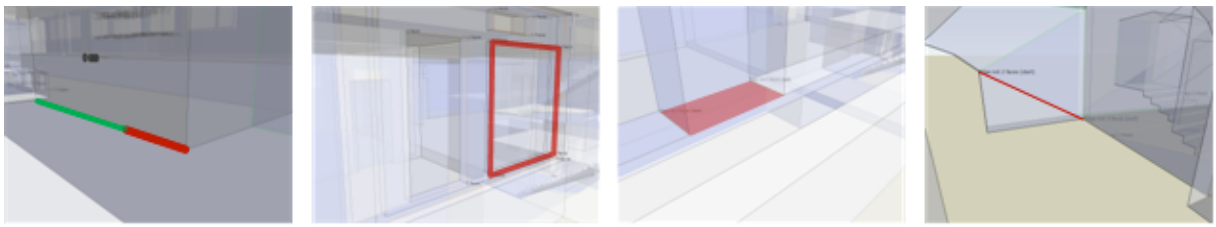


Figure 2 Visualisation of invalid drwaing elements

A room is a polyhedral object. For these objects an ruler characteristic (Hatcher, 2002) is computable by using the euler's formula (Mäntylä, 1976) (Mortenson, 2006). The Euler formula

$$v-e+f = 2s - 2h + r$$

checks the number of the room related vertices (v) , edges (e) , faces (f) , loops (r) and passages (holes (h) through the entirely connected object (s)). The Euler formula is necessary, but not sufficient.

An early detection of invalid drawing elements and proper visualization on screen is very important because posterior corrections could induce other (cascading) mistakes.

2.3 Adding topology to SketchUp

The necessity of an explicitly specified topology is not only a question of accelerating computational geometry as stated in [ISO19107]. Moreover, when using a 3d-model as a field book, topology is need for the

- *identification* of primitive elements (vertices, edges, faces, solids)
- attachment of surveying *observations* (distances, polar survey in relation to the building structure)

- *calculation* of geometric properties
- *semantic* enrichment for automatically derived geometric constraints

Using the SketchUp-API the {*vertex-edge-face*}-graph can be traversed on runtime. However the relationships between elements are not accessible as a linked *key-keyref* pair. Since identifiers (id) should be in place, serializable (binary, XML, database) and not changing over the entities lifetime, each vertex, edge and face of the building structure is attached with an attribute *@id*. By adding references *@ref* to edges and faces an accessible, two-dimensional topology is established. SketchUp does not support the entity type solid off-the-shelf. By adding an *@ref_solid* reference to each face entity, a (pseudo) 3d-topology is establish, although no solids are stored in the model.

2.4 Creating an Object Model with SketchUp – a solid oriented workflow

While drawing the three-dimensional sketch, it is hard to take the whole building under consideration. A workflow that suits a surveying procedure generates the building as a set of rooms. By drawing each room for its own, more attention is paid to the currently observed object. The building is hierarchically assembled from rooms. First the outer shell of the building is modeled including at least one passage. Using the doors and windows as gluing elements, diverse single rooms (*Figure 3*) can be fitted together (see *Figure 5* and *Figure 6* and finally *Figure 4*). SketchUp supports the exact three-dimensional positioning of composites (rooms) and tracks topological changes in the drawing elements.

.

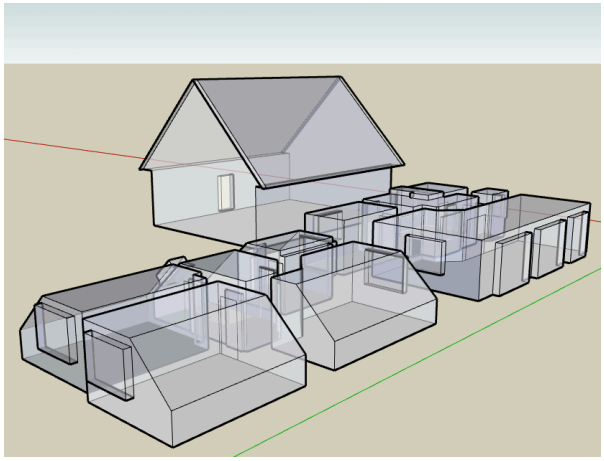


Figure 3 rooms of a building

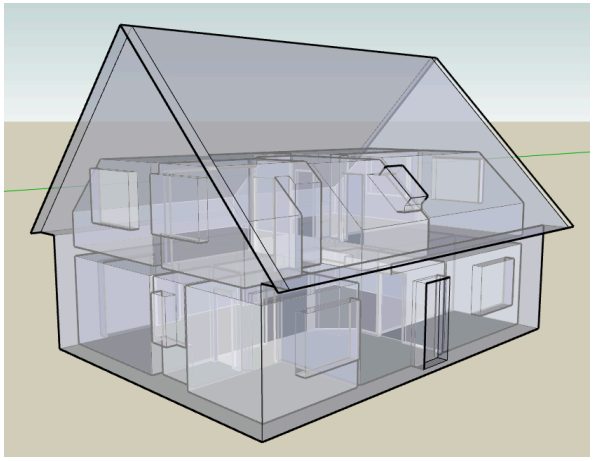


Figure 4 composite of rooms

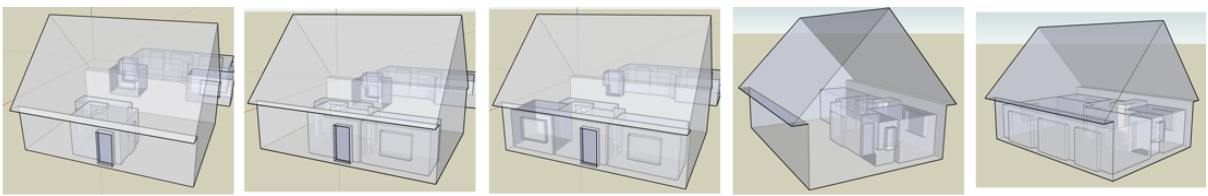


Figure 5 assemby of rooms forming the ground floor

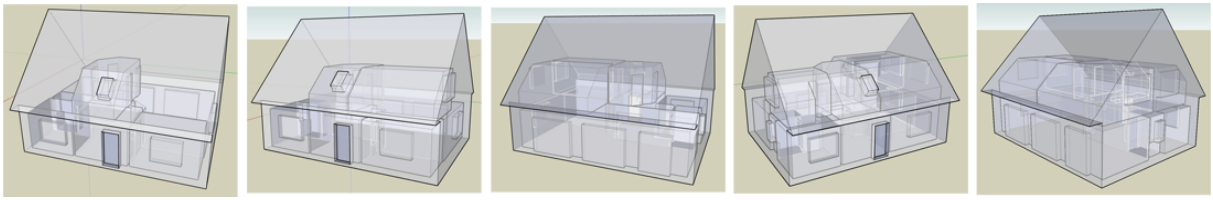


Figure 6 assemby of rooms forming the upper floor

3. ROOM-ELEMENT-AGGREGATION

An aggregation or object composition is a way to combine simple objects or data types into more complex ones (Wikipedia). As stated above, a pseudo three-dimensional topology is set up by adding one or two attributes *@ref_solid* to each face. Solids are indirectly identifiable as a set of faces having equal values for the attribute *@ref_solid*.

Faces representing a window or door are either classified as *passage-face* or, if representing a wall, as *wall-face*.

3.1 Room Element Aggregation application Sketchup

SketchUp uses an object oriented data model. Each edge knows about its start and end vertex. A face aggregates edges, loops and provides access to its plane and normal vector. A loop knows about its related edges. Using the Google SketchUp API (Google, 2010) a ruby script is written to detect, collect and label all needed faces and edges of a room.

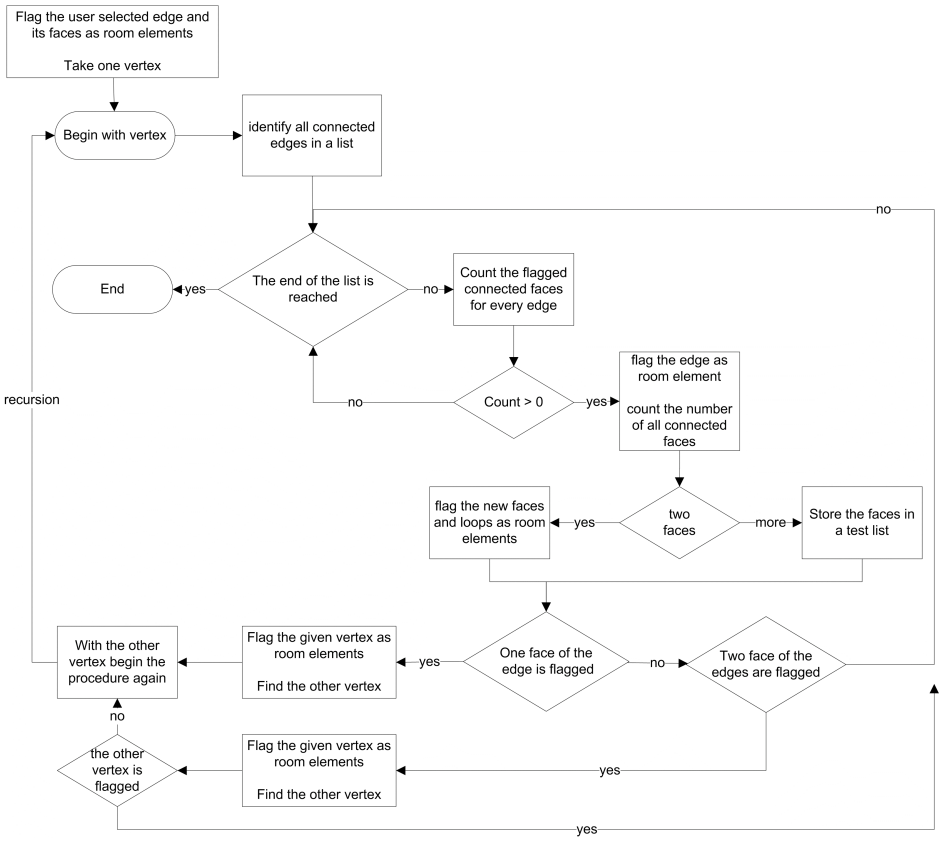


Figure 7 progam overview

Like shown in the program overview (Figure 7) the algorithm starts with one user-selected edge (Figure 8 or Figure 10). That selected edge has to

- be part of the geometry bounding the solid under consideration
- be connected with two faces, belonging to the solid under consideration

One vertex of the start edge represents the root of recursion (*Figure 9* and *Figure 11*). After retrieving all connected edges of that vertex each edge will be inspected. Edges connected to one or two flagged faces are related to the regarded room. In case of *Figure 9* two new edges lie at the bottom room face. That bottom face is not flagged after the first step. But the new edges are connected with the flagged faces alongside the room. This is the

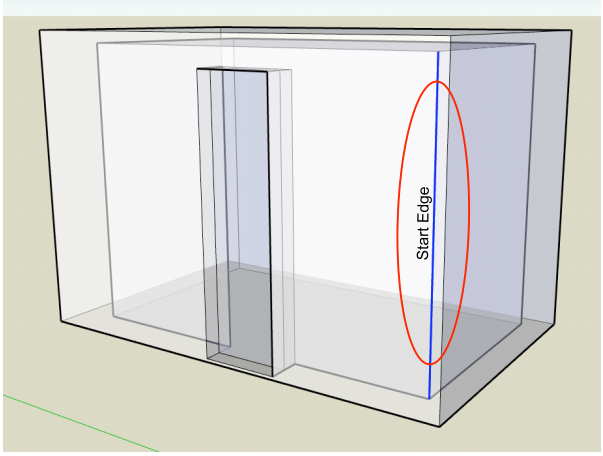


Figure 8 start situation for the interior

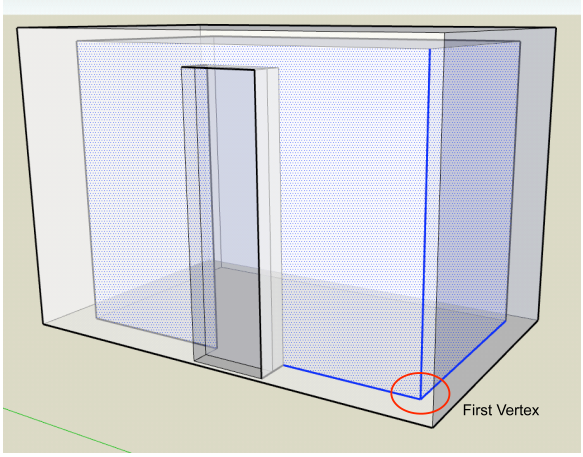


Figure 9 application at the first vertex

reason why the two new edges and corresponding faces, here the bottom face, are related to the new room and will be flagged. If the edge was as room-edge inspected the procedure starts by using the opposite vertex of inspected edge as a starting point.

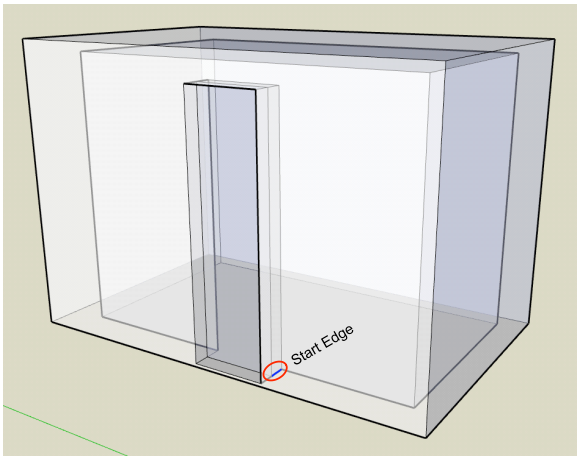


Figure 10 second start situation for the interior

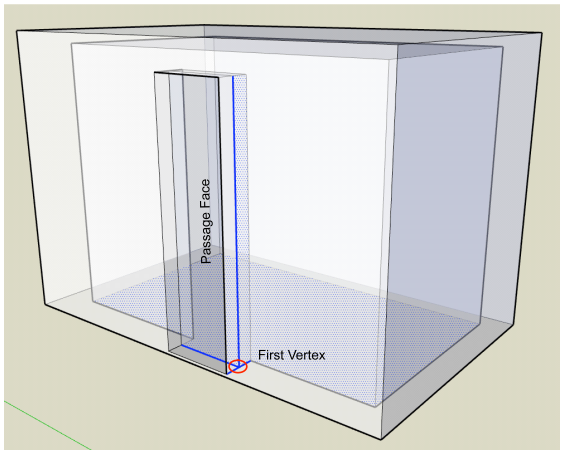


Figure 11 application at the first vertex

In the case of *Figure 10* we start with an edge next to a *passage-face* where the first vertex response 3 new edges (*Figure 11*).
One of these edges goes out of the regarded room. Both retrieved faces of that edge are not attributed as room elements so that edge belongs not to the regarded room.
The other two of the edges in figure 10 are adjacent to three faces. So the application has the choice between one outside face and one associated *passage-face* regarded to the considered room. If all edges related to a passage face belong to the considered room the *passage-face* is a component of the room. That feature (*passage-face-test*) helps to find the right face in situation *Figure 11*. By now, at the first vertex, we have not enough edge information to make a decision, so both faces will stored in a temporary test-list. After attributing all edges of a room we can come to a decision. For every face in the test temporary test-list we have to make the *passage-face-test* to get all elements of the treated room.

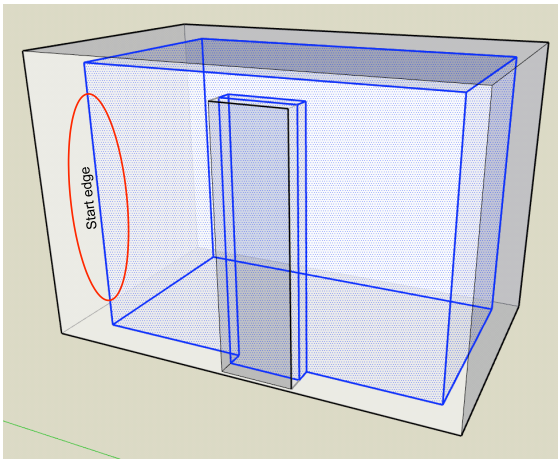


Figure 12 room aggregation for the interior

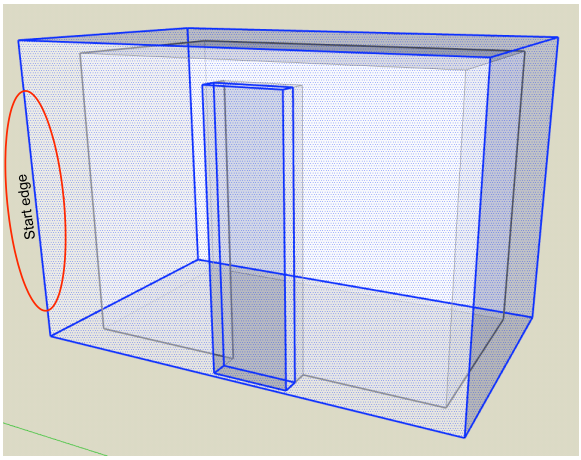


Figure 13 room aggregation for the exterior

As a final result for both discussed situations (*Figure 8* and *Figure 10*) we get *Figure 12* as result. Please note, that the same algorithm is used in order to find the exterior *Figure 13*. Therefore only the starting edge must be in the exterior.

3.2 Introduction of a Room Topology

When the starting edge is a surrounding element of the *passage-face* between two rooms the developed algorithm returns all elements of both rooms (see *Figure 14* and *Figure 15*). That gives the possibility to pass automatically through the whole building to create interior {room-face}-graph by setting the *@ref_solid* attributes to the flagged faces.
At the beginning all exterior faces has to be attributed. After that the edges of all *passage-faces* have to be extract. We can attribute all faces of the next room when the algorithm starts with one *passage-face-edge* to the next room because all faces with no *@ref_solid* attribute inclusive the used passage face belongs to the next room (other *@ref_solid-value*). This can be done for all *passage-faces* of a room and all passage-faces for the next rooms until every room-face has one *@ref_solid-value*. The result of that is a {*room-face*}-graph is all rooms of a building.

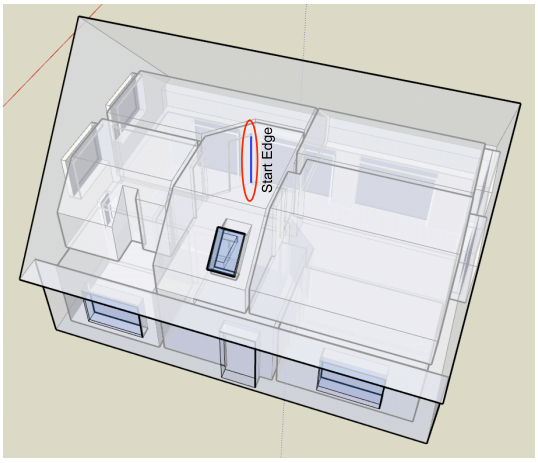


Figure 14 start edge between two rooms

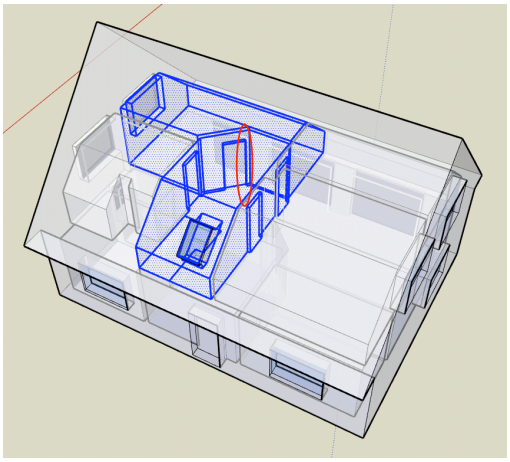


Figure 15 result all elemens in both rooms selected

4. THE WALLSPACE

If the interior {*room-face*}-graph is generated, every face "knows" the room of which it is the boundary. It is easily possible to set up and traverse a {*room-room*}-graph. Until now most faces are related to one room. Only the *passage-faces* are related to two rooms. Because of the distinction between only two kinds of faces (*passage-faces* and *wall-faces*) all the other faces must be wall-faces. Assembling all (!) faces that are linked to only one room, the "*wall-space*" is identified. For now the physical building is represented as a single block, consisting only of one single solid of type BE. In order to increase topological granularity the *wall-space* can be cut into pieces. The resulting {*BE-face*}-graph and {*BE-BE*}-graph or {*BE-room*}-graph enables the navigation on a semantic level. Until now the specification of building elements (*BE*) must be done semi-automatically by manually drawing a separation-face and applying the above described algorithms. Complete automation of this procedure is an interesting and challenging topic for further research.

5. OUTLOOK

This work shows how a semantic modularization on building models can be used in order to make the surveying process easier for a layman and the modeling process faster. Then it is shown how a pseudo 3D-topology can be introduced. Therefore a room aggregation algorithm was developed and implemented in SketchUp. Solids are classified as rooms and building elements.

Further research must be undertaken concerning graphical user support in order to accelerate the user's input time. Establishing building elements in a 3d-model is crucial but cannot be realized without hypothesis about the building structure, since the interior elements and junctions are not visible.

The overall sketch-observe-adjust-3d-model workflow should be integrated in only one user interface. A good candidate for this is Google SketchUp. Although having only a basic geometric data model, the graphical user interface allows intuitive interaction. With the open the Ruby-API (and C++-API) it is possible to extend Sketch-Up's functionality to proper engineering software.

REFERENCES

Clemen, C., & Gielsdorf, F. (2008). Architectural Indoor Surveying. An Information Model for 3D Data Capture and Adjustment. *American Congress on Surveying and Mapping Conference (ACSM).* Spokane.

Gielsdorf, F., & Gründig, L. (2002). *Geometrical Modeling for Facility Management Systems.* Washington, D.C. USA: FIG XXII International Congress.

Google. (n.d.). *Objects Diagram Google SketchUp Ruby API*. Retrieved Januar 21, 2010, from http://code.google.com/intl/de-DE/apis/sketchup/docs/diagram.html

Hatcher, A. (2002). *Algebraic Topology.* Cambrige University Press.

Mäntylä, M. (1976). *An Introduction to Solid Modeling.* Rockville, Maryland, USA: Computer Science Press.

Mortenson, M. E. (2006). *Geometric Modeling.* New York: Industial Press Inc. .

Norbert Paul, A. B. (2009, October). Geometrical and Topological Approaches in Building Information Modelling. *Journal of Information Technology in Construction (ITcom Vol. 14)* , pp. 705-723.

Thomas, D. (2009). *Programming Ruby 1.9.* Raleigth, North Caroline, USA: Pragmatic Bookshelf.

CONTACTS

Christian Manthe
Technische Universität Berlin
Department for Geodesy and Geoinformation Science
Straße des 17. Juni 135
10623 Berlin
GERMANY
Tel. +49 (30) 314 24147
Email: Christian.manthe@tu-berlin.de

Christian Clemen
Technische Universität Berlin
Department for Geodesy and Geoinformation Science
Straße des 17. Juni 135
10623 Berlin
GERMANY
Tel. +49 (30) 314 26483
Email: christian.clemen@tu-berlin.de