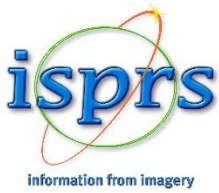
ISPRS Scientific initiative 2021

(Final report)



to

Pushing forward the development of software tools for IndoorGML

Team

Dr. Abdoulaye Abou Diakite (Principal Investigator) – UNSW (Australia)
Dr. Lucía Díaz-Vilariño (Co-Investigator) – U-Vigo (Spain)
Dr. Filip Biljecki (Co-I) – NUS (Singapore)
Prof. Sisi Zlatanova (Co-I) – UNSW (Australia)
Prof. Ki-Joune Li (Co-I) – Pusan University (South Korea)
Prof. Ümit Işıkdağ (Co-I) – Mimar Sinan Fine Arts University (Turkey)
Scott Simmons (Co-I) – OGC (USA)

1. Context

In recent years, the interest in 3D indoor models is increasing. Most of the indoor 3D models have been made available as IFC models. However, these models are very complex, containing many details, which often leads to privacy issues. IndoorGML (Lee et al., 2014) is one of the standards for describing 3D indoor space but with the purpose to support Location Based Services (LBS). It contains a relatively simple geometry (space-based) and semantic and provides mechanism for aggregation, which allows to protect sensitise property information.

IndoorGML relies on solid scientific concepts and offers a high flexibility with extension mechanisms. It provides a geometric, topological, and semantic description of the indoor which facilitates specifically applications like indoor navigation or facility management. Accepted as an Open Geospatial Consortium (OGC) standards since January 2015, it is actively developed and extended by several universities. A new version, IndoorGML 2.0 is currently under development to enhance and comply with user requirements.

However, despite its solid conceptual basis, IndoorGML is suffering from a lack of practical tools and remains largely an academic development. So far, few open-source initiatives have been undertaken to address this gap. The STEM Lab[1] from the Pusan University (Korea), the GRID team[2] from the University of New South Wales (Australia) and the 3DGeoInfo team[3] from TU Delft (The Netherlands) have been working on tools, STEM being the main contributor to the IndoorGML software ecosystem. OGC has led a pilot project on indoor navigation[4], which also contributed to some developments. Unfortunately, most of those initiatives are resulting from separated and disconnected projects, leading to scattered outputs.

2. Objectives

The aim of this project is to push forward the accessibility and use of IndoorGML in the geospatial community by introducing a simple, robust, and open-source tool named *ifc2indoorgml*, that allows easy generation of IndoorGML models from IFC architectural models. With the involvement of ISPRS members from 3 working groups of Commission IV (WG IV/1, WG IV/5 and WG IV/10), in addition to members of the standards working group (SWG) IndoorGML and OGC representative, the goal is to significantly advance the international standard for its broader use in 3D indoor modelling, LBS and beyond.

3. Workflow and implementation

There are three main steps involved in our workflow to implement the *ifc2indoorgml* tool (Figure 1):

[1] github.com/STEMLab
[2] github.com/grid-unsw
[3] github.com/tudelft3d/indoorjson
[4] docs.ogc.org/per/18-089.html

Figure 1: The main steps of the workflow.

1. IFC import to LCC: this part consists in the deployment of an IFC model parser which will ingest the input files and collect geometric, topological, and semantic information from them. The collected information needs to be organised in a structured way for efficient operations on the entities. For this reason, we will use the Linear Cell Complex (LCC) (Damiand, 2022).
2. Data processing: The data organised in the LCC is processed to derive adapted IndoorGML information. For example, the stored geometry will be used to compute the nodes, the semantic data will be used to classify the CellSpace entities properly (e.g., into CellSpace or NavigableSpace, etc.). But mostly, the topological relationships between the CellSpaces are maintained in the LCC.
3. IndoorGML export: once the IndoorGML information is generated, IndoorGML files can be exported. We took into consideration the differences between the versions 1 and 2 (which is still a beta version until its final release).

Figure 2: The open-source libraries used for implementing the ifc2indoorgml tool.

The ifc2indoorgml tool has been fully implemented in C++ and it relies on other existing open-source libraries. Each of the 3 components of the workflow relies on one major library:

- IFC++ (Gerold, 2022): this is the library used in the first part of the workflow. IFC++ is an open source IFC implementation for C++, originally developed at the Bauhaus University Weimar and available on GitHub. It provides C++ class models, as well as a reader and writer for IFC files in STEP format. It relies on other C++ libraries (e.g., Carve, OpenSceneGraph, etc.) to handle robustly Constructive Solid Geometry (CSG) operations that may be necessary for explicitly representing some IFC entities.
- CGAL (The CGAL Project, 2022): this library is partly used in the first step (for the LCC) and is the main one that is used for all the required data processing. The Computational Geometry Algorithms Library (CGAL) is a software project that provides easy access to efficient and reliable geometric algorithms in the form of a C++ library. It is used in various areas needing geometric computation, such as GIS, CAD, medical imaging, etc. It offers data structures and robust geometric predicates, which are necessary for storing, managing, and processing the date imported from IFC files.
- RapidXML (Kalicinski, 2009): this was used to handle the import and export of IndoorGML files. It is a lightweight XML library for C++.

4. Results

The source code of the ifc2indoorgml tool has been made available on GitHub[5]. Instructions are provided to guide users for the compilation steps required to build it from scratch. Binaries for 64bits Windows operating system are also provided for easier access. A simple user interface (UI) is provided to simplify the use of the tool. The UI is built on the one provided by the LCC demo that comes with the CGAL library package, which has been customised for the purpose of the project (see Figure 3).

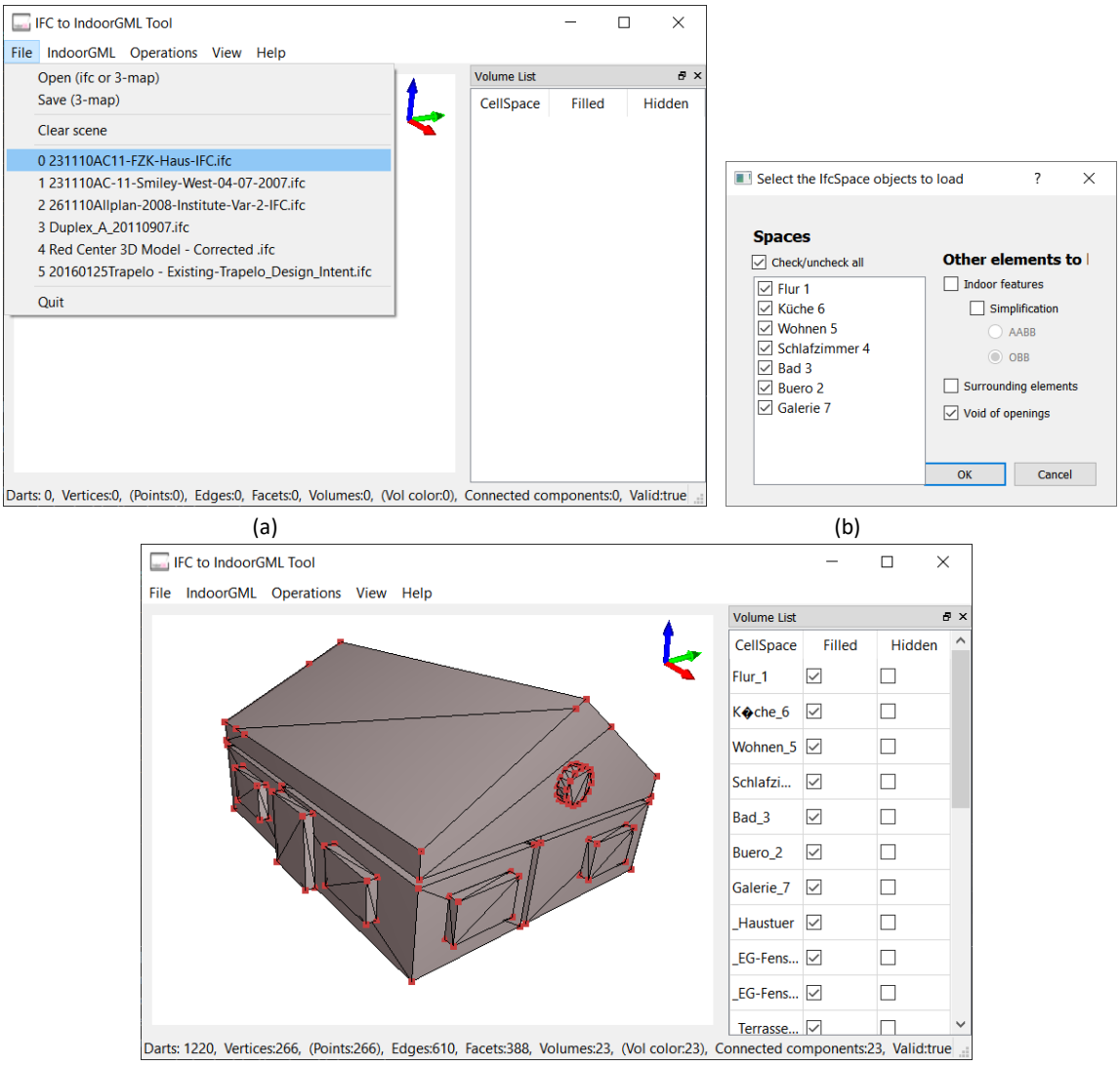


Figure 3: ifc2indoorgml user interface and example of loading an IFC file. (a) Options under the File button. (b) IfcSpace entities detected in the selected model. (c) All the spaces loaded as cells of the LCC.

The File tab contains button to open an ifc file, or a 3-map file (an XML format for LCC models for saving/loading other formats than IndoorGML). Alternatively, previously loaded models are listed in a history and can be directly reloaded (Figure 3-a). When an IFC model is selected, the parser queries the

[5] github.com/grid-unsw/ifc2indoorgml

IfcSpace entities that is contains and offer to the user options to discard some of them or keep all of them (Figure 3-b). The user can also choose to load indoor features such as furnishing elements, with their original geometries or simplified, either as Axis-Aligned Bounding Boxes (AABB) or as Oriented Bounding Boxes (OBB). It is also possible to load features surrounding the spaces (walls, slabs, etc). All these options are unchecked by default, except for the void of the openings, which is critical for the generation of correct IndoorGML dual space (network) (see Figure 3-b). The loaded model is just composed of the selected spaces, and it should form a valid LCC. This is indicated in the stats at the bottom of the window, along with the number of entities in the LCC (number of faces, volumes, etc.). More details on those notions can be found in (Damiand, 2022).

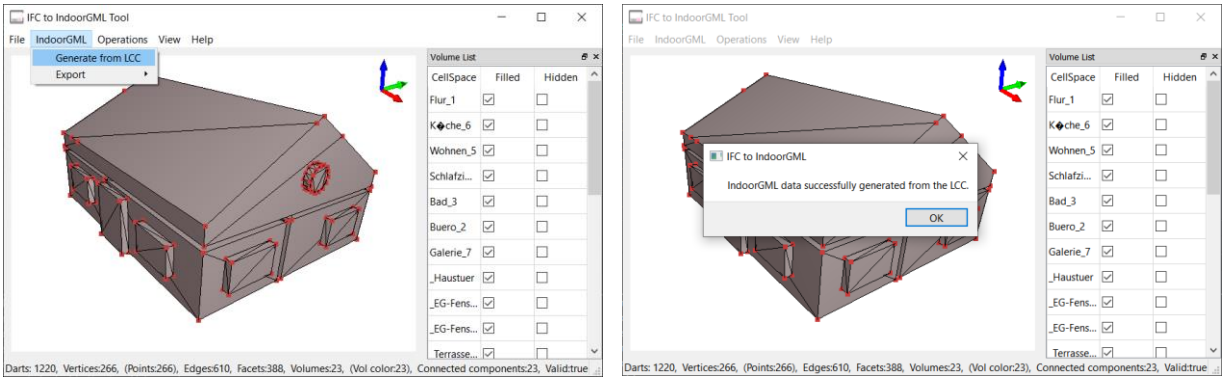


Figure 4: Generation of IndoorGML data from the LCC.

The IndoorGML tab regroups operations specific to the standard. For now, only two options are available, one for generating the IndoorGML data from the LCC loaded in the scene (Figure 4) and another one for exporting a selected IndoorGML version. A pop-up message will let the user know if the generation of IndoorGML data is successfully done or not. If it is successful, it is already possible to see the network generated for the dual space of IndoorGML, by un-filling the volumes of the LCC (hiding the faces). This is done either by clicking on the keyboard shortcut '**W**' or by un-ticking 'Filled' options of the volumes, on the volume list on the right (Figure 5). A list of all keyboard shortcuts can be found in Annex 2.

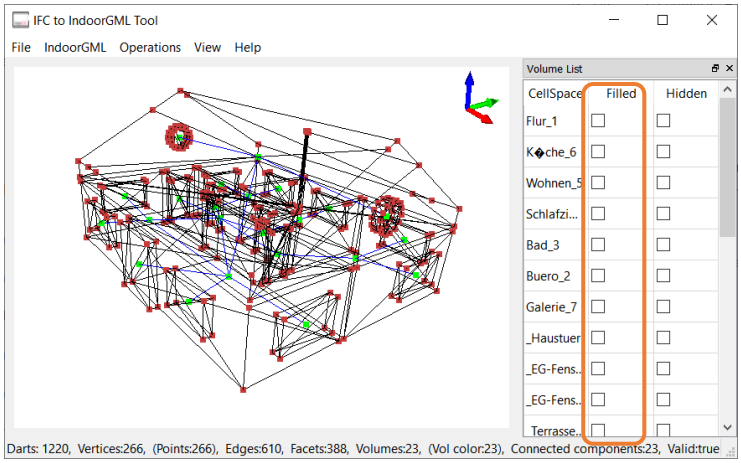


Figure 5: Hiding the volumes' faces to see the generated network.

Other general operations on the LCC are implemented under the Operations tab (e.g., merging of coplanar 2-cells (faces) to reduce tessellation, triangulation of 2-cells, deletion of volumes from the model, etc.). Figure 6 illustrates an example of tessellation reduction that allows to reduce the number of edges visible on the model, which helps see the network clearer. Note that most of these operations are sensitive to geometric and topological issues, and they may fail (or crash the application) when the geometry from the input model has some errors (wrongly oriented faces, unclosed or flat volumes, etc.).

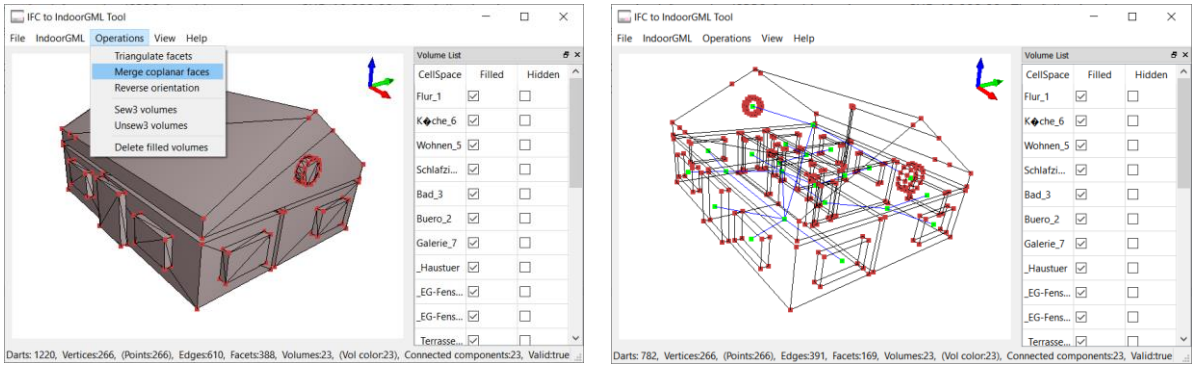


Figure 6: Simplification operation on the LCC.

The View tab has an option to reset the zoom on the loaded scene for now, while the Help provides some details on the project. It also allows to enable or disable the Volume list on the right-hand side. The latter is provided to help the scene manipulation by listing all the CellSpaces that could be retrieved in the input data. The user can also fill or un-fill the volumes to switch between plain and wireframe views, or simply hide/unhide them.

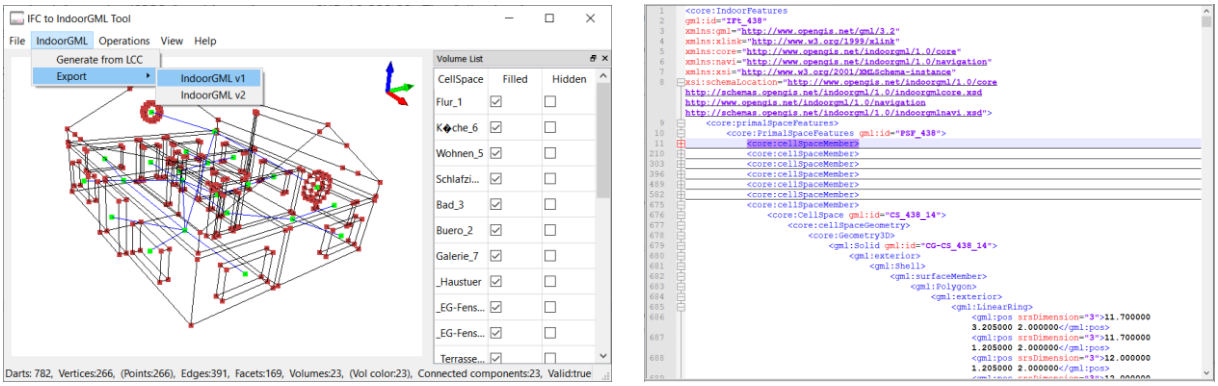


Figure 7: IndoorGML export.

The current version of the ifc2indoorgml tool offers the possibility to export both IndoorGML v1 and v2 (beta). This lead to a file explorer for choosing the name and receiving folder of the file to generate and then a '.gml' file is created (see Figure 7).

5. Expenses and outputs

As the main work in the project was about software development, the budget has been fully used for the development time. All the rest, including the extra-development time and the related communications are covered by in-kind contributions from the investigators.

Outputs:

- Open-source code available at github.com/grid-unsw/ifc2indoorgml.
- Conference paper (extended abstract) accepted and to be published in the Archives proceedings of the ISPRS congress in Nice 2022. More technical details about the implementation can be found there.

References:

- CGAL Project (The), 2022. CGAL Editorial Board. cgal.org (28 March 2022).
- Damiand, G., 2022. Linear cell complex. CGAL User and Reference Manual, 5.4 edn, CGAL Editorial Board. doc.cgal.org/latest/Linear_cell_complex/index.html.org (28 March 2022).
- Gerold, F., 2022. IFC++, Open source IFC implementation. IFC++ - Open source IFC implementation. ifcquery.com (08 March 2022).
- Kalicinski, M., 2009. Rapidxml. rapidxml.sourceforge.net (09 March 2022).
- Lee, J., Becker, T., Nagel, C., Kolbe, T. H., Sisi, Z., Li, K.-J., 2014. OGC®IndoorGML, Version 1.0.

Annex 1

ifc2indoorgml- Installation Guide

Dependencies: (the indicated versions are the tested ones, but lower versions may still work)

- CMake (>= 3.1)
- Qt5 (>=5.10)
- CGAL (>=5.3)
- OpenSceneGraph (>= 3.6)
- IFC++

For all these tools, it is critical to make sure that their system version used (32bits or 64bits) is consistent. This means that if your system is 64bits, whether you download them already compiled (binaries) or you compile them from their sources.

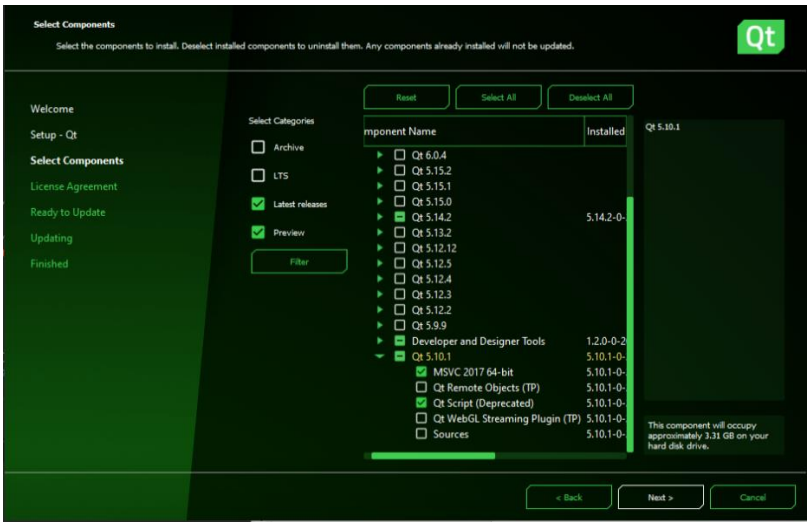
Step 1: Install CMake

https://cmake.org/download/

It is common to use cmake in the terminal, but it may be handy to have the UI, mainly for an easier configuration of the projects to build.

Step 2: Install Qt5 (<= 5.10) binaries

Go to https://www.qt.io/download-qt-installer and download the Qt download assistant. Select a version depending on your system's preferred compiler (e.g. in my case, on a Windows 10 64-bit, I use MSVC 2017 64-bit). Also, make sure to select **QtScript** among the listed components (even though it is deprecated). **Versions that do not start with 5.X.X are not part of Qt5.**



Step 3: Install CGAL (>=5.3)

https://www.cgal.org/download.html

CGAL is a header-only library, which means that you don't have to compile/build any resource to use it. Therefore, I would recommend downloading the sources on GitHub (https://github.com/CGAL/cgal/releases) – e.g. CGAL-5.3-library.zip. CGAL has some dependencies on other libraries (Qt5, GMP and MPFR). While it should automatically detect the Qt5 installed, binaries of GMP and MPFR libraries for Windows (64bits) are provided in the same GitHub page that provides the sources. More guidance on installing CGAL can be found here: https://doc.cgal.org/latest/Manual/general_intro.html.

Step 4: Install OpenSceneGraph

http://www.openscenegraph.org/index.php/download-section/stable-releases

After installing OSG, it is important to set its relevant folders in the system **environment PATHS**. Make sure you have these following properly set:

1. the **include** folder in the OSG installation folder (e.g. C:\Tools\OpenSceneGraph-3.6.0\include)
2. the folder containing the OSG **release** libraries (e.g., osg.lib). Optionally, you could add the directory with the **debug** libraries too (e.g., osgd.lib).

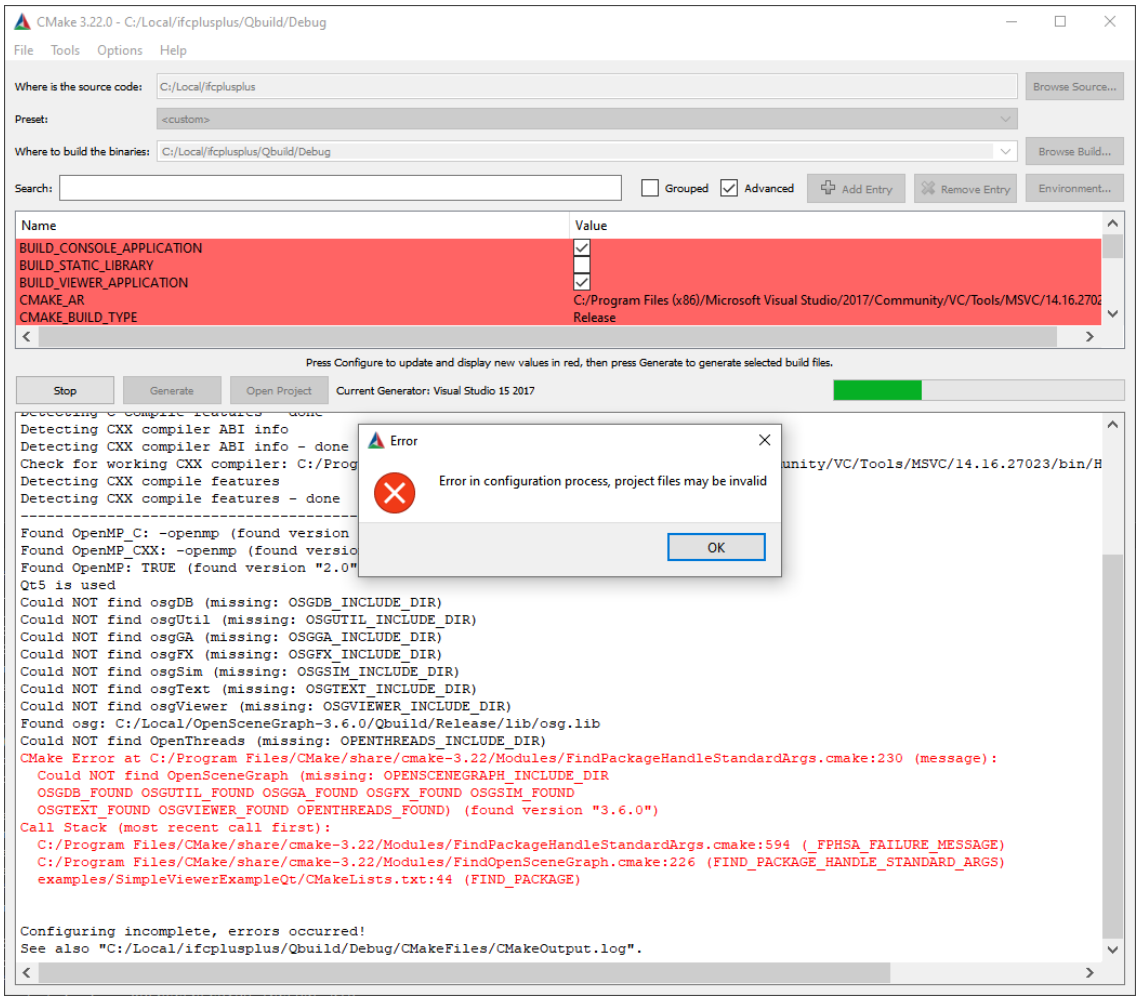
On Windows, you can simply add the corresponding directories to the "path" environment variable without having to set a specific variable name for each of them. This has not been tested for Linux and Mac, but the same behaviour is expected. Furthermore, chances are that those variables will be handled automatically if you use a package manager of your OS (e.g. apt-get install, homebrew, etc.).

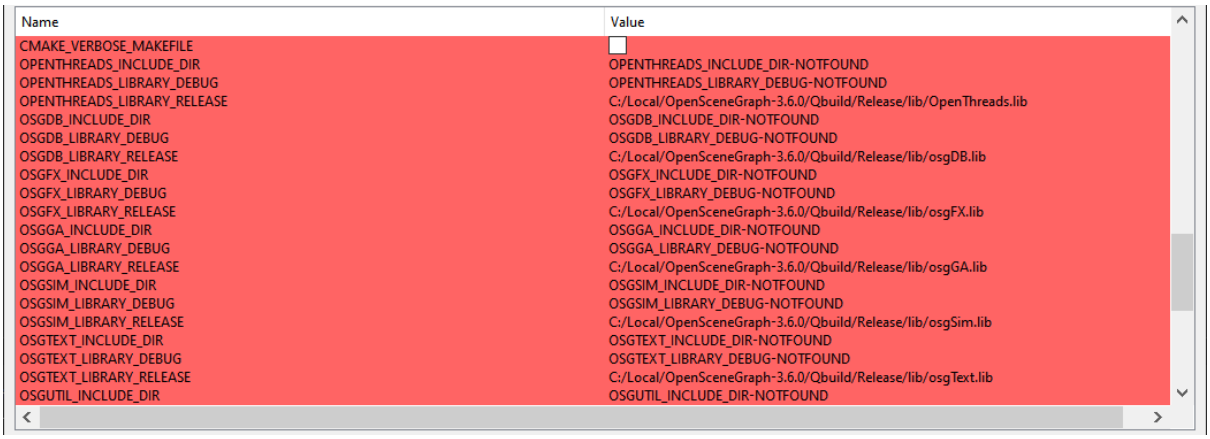
Step 5: Install IFC++

https://github.com/ifcquery/ifcplusplus

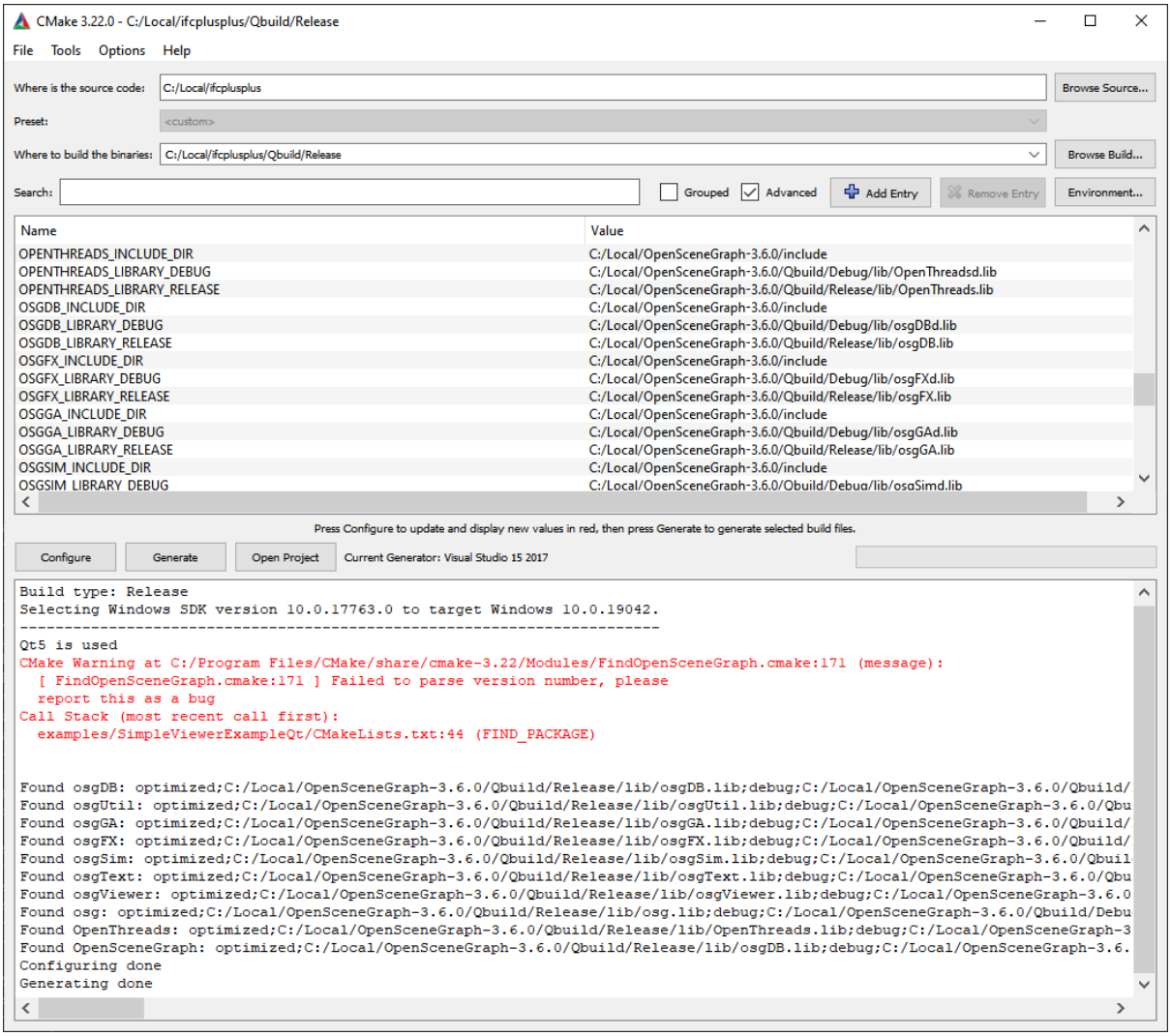
This library is not much documented unfortunately. A building/installation guide is provided (see **Build IFC++.pdf** document), but I could not successfully replicate it. Instead, after installing Qt5 and OSG (IFC++ depend on them), I could compile the two necessary components of the library: **IfcPlusPlus** and **Carve**. The CMakeLists of both components can be found in the main folder (in /IfcPlusPlus and external/Carve). If you intend to contribute to the code of the project, it may be handy to build both the Debug and Release versions of the IFC++ library, by setting up the CMAKE_BUILD_TYPE variable accordingly.

- **IfcPlusPlus**: run the CMakeLists.txt wich CMake. You may see a similar error message when some dependency components are not found:





In the cases above, Qt5 was successfully found, but not the OSG components (osgDB, osgUtil, etc.). Simply change their corresponding values to point to the right files and folders as indicated in the image below. Although Debug and Release versions of OSG are provided, only the release version would be enough, as the debugging of ifc2indoorgml would not depend on it.



Notes: The message in red in the above image is simply a warning. It will not affect the rest of the process and can be ignored. Also,

Once the configuration and generation are done with CMake, you should be able to build the IFC++ libraries and obtain the following files (on Windows): **IfcPlusPlus.dll, IfcPlusPlus.exp, IfcPlusPlus.lib.** For other OS, you may just have the **.lib** file, or **.a**. Also the files will have a 'd' at the end of their names for their Debug versions (e.g. IfcPlusPlus**d**.lib).

- Same process as above should be followed with the CMakeLists.txt file in the external/Carve folder, to obtain the single file **carve.lib** (or **carved.lib** for the Debug version).

Step 6: Install ifc2indoorgml

https://github.com/grid-unsw/ifc2indoorgml

If all the above dependencies are properly installed, it should be fairly easy to build ifc2indoorgml. A simple run of CMake should be enough to generate the sources and build them. If it fails, it is probably

because the libraries above are not visible to CMake. In that case, you can either ensure that all the folders containing the sources and the above libraries are included in the environment PATH of your system, or you could simply add them manually to CMake, just like we did in Step 5. Below is a snapshot of the folders that needed to be visible in my case for CMake to automatically configure and generate the build files:

- **Qt5**

Qt5_DIR	C:/Local/Qt/5.10.1/msvc2017_64/lib/cmake/Qt5
Qt5Core_DIR	C:/Local/Qt/5.10.1/msvc2017_64/lib/cmake/Qt5Core
Qt5Gui_DIR	C:/Local/Qt/5.10.1/msvc2017_64/lib/cmake/Qt5Gui
Qt5OpenGL_DIR	C:/Local/Qt/5.10.1/msvc2017_64/lib/cmake/Qt5OpenGL
Qt5Script_DIR	C:/Local/Qt/5.10.1/msvc2017_64/lib/cmake/Qt5Script
Qt5Svg_DIR	C:/Local/Qt/5.10.1/msvc2017_64/lib/cmake/Qt5Svg
Qt5Widgets_DIR	C:/Local/Qt/5.10.1/msvc2017_64/lib/cmake/Qt5Widgets

- **CGAL** (and related dependencies: Boost, GMP and MPFR; GMPXX can be ignored)

⌄ Boost	
Boost_DEBUG	☐
Boost_INCLUDE_DIR	C:/Local/boost_1_66_0
⌄ CGAL	
CGAL_Boost_USE_STATIC_LIBS	☐
CGAL_CTEST_DISPLAY_MEM_AND_TIME	☐
CGAL_DEV_MODE	☐
CGAL_DIR	C:/Local/CGAL/CGAL-5.3
CGAL_TEST_DRAW_FUNCTIONS	☐
CGAL_WITH_GMPXX	☐
› CMAKE	
⌄ GMP	
GMP_INCLUDE_DIR	C:/Local/CGAL/CGAL-5.3/auxiliary/gmp/include
GMP_LIBRARY_DEBUG	C:/Local/CGAL/CGAL-5.3/auxiliary/gmp/lib/libgmp-10.lib
GMP_LIBRARY_RELEASE	C:/Local/CGAL/CGAL-5.3/auxiliary/gmp/lib/libgmp-10.lib
› GMPXX	
⌄ MPFR	
MPFR_INCLUDE_DIR	C:/Local/CGAL/CGAL-5.3/auxiliary/gmp/include
MPFR_LIBRARIES	C:/Local/CGAL/CGAL-5.3/auxiliary/gmp/lib/libmpfr-4.lib
MPFR_LIBRARIES_DIR	C:/Local/CGAL/CGAL-5.3/auxiliary/gmp/lib

- **OSG**

OPENTHREADS	
OPENTHREADS_INCLUDE_DIR	C:/Local/OpenSceneGraph-3.6.0/include
OPENTHREADS_LIBRARY_DEBUG	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Debug/lib/OpenThreadsd.lib
OPENTHREADS_LIBRARY_RELEASE	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Release/lib/OpenThreads.lib
OSG	
OSG_INCLUDE_DIR	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Release/include
OSG_LIBRARY_DEBUG	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Debug/lib/osgd.lib
OSG_LIBRARY_RELEASE	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Release/lib/osg.lib
OSGTEXT	
OSGTEXT_INCLUDE_DIR	C:/Local/OpenSceneGraph-3.6.0/include
OSGTEXT_LIBRARY_DEBUG	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Debug/lib/osgTextd.lib
OSGTEXT_LIBRARY_RELEASE	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Release/lib/osgText.lib
OSGUTIL	
OSGUTIL_INCLUDE_DIR	C:/Local/OpenSceneGraph-3.6.0/include
OSGUTIL_LIBRARY_DEBUG	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Debug/lib/osgUtild.lib
OSGUTIL_LIBRARY_RELEASE	C:/Local/OpenSceneGraph-3.6.0/Qbuild/Release/lib/osgUtil.lib

- **IFC++**

Carve_LIBRARIES	C:/Local/ifcplusplus/Qbuild/x64/Debug/external/Carve/Debug/carved.lib
IFCPP_LIBRARIES	C:/Local/ifcplusplus/Qbuild/x64/Debug/IfcPlusPlus/Debug/IfcPlusPlusd.lib

Annex 2

ifc2indoorgml – Keyboard shortcuts

Key(s)	Description
+	Increase size of edges
-	Decrease size of edges
C	Switch clipping plane display mode
E	Toggles edges display
M	Toggles mono color
N	Inverse direction of normals
O	Toggles 2D mode only
R	Toggles random face colors
S	Switch between flat/Gouraud shading display
T	Toggles text display
U	Move camera direction upside down
V	Toggles vertices display
W	Toggles faces display
PgUp	Decrease light (all colors, use shift/alt/ctrl for one rgb component)
PgDown	Increase light (all colors, use shift/alt/ctrl for one rgb component)
Ctrl++	Increase size of vertices
Ctrl+-	Decrease size of vertices
Alt+C	Toggle clipping plane rendering on/off
Standard viewer keys	
Space	Changes camera mode (observe or fly)
A	Toggles the display of the world axis
F	Toggles the display of the FPS
G	Toggles the display of the XY grid
H	Opens this help window
Return	Starts/stops the animation
Left	Moves camera left
Up	Moves camera up
Right	Moves camera right
Down	Moves camera down
Shift+?	Toggles the display of the text
Ctrl+Q	Exits program
Alt+Return	Toggles full screen display
Camera paths are controlled using the F1..F12 keys (noted *Fx* below):	
Fx	Plays path (or resets saved position)
Alt+*Fx*	Adds a key frame to path (or defines a position)
Alt+*Fx*+*Fx*	Deletes path (or saved position)