

TUM Open Infra Platform: an open source package for simultaneous viewing and analysis of digital models in the civil engineering domain

Štefan Jaud^{1¶}, Helge Hecht¹, Jonas Schlenger¹, and Julian Amann¹

¹ Chair for Computational Modeling and Simulation, Technical University of Munich ¶ Corresponding author

DOI: [10.21105/joss.03061](https://doi.org/10.21105/joss.03061)

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Hugo Ledoux](#) ↗

Reviewers:

- [@aothms](#)
- [@CBenghi](#)
- [@abdoulayedialk](#)

Submitted: 23 February 2021

Published: 26 April 2022

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

The Architecture, Engineering and Construction (AEC) is in its transition from 2D design processes to 3D object-oriented modelling. Building Information Modeling (BIM) is steadily gaining importance, replacing the conventional Computer-Aided Design (CAD) practices and getting implemented in every aspect of the very complex software and stakeholder landscape (Jaud et al., 2019). As one of the main principles, BIM describes the idea of integrating all information relevant to the life cycle of a structure, such as a tunnel, bridge, building or road, in a digital (BIM) model. The digital model is to ensure, among other things (Amann, 2018):

- that all relevant data is available to all project participants;
- that all data is in a consistent state (data integrity should be guaranteed); and
- that the data can be used efficiently.

TUM Open Infra Platform (OIP) is an open source application for viewing and analysis of different BIM models used in the civil engineering field. OIP supports reading, visualization, navigating, and handling of:

- Industry Foundation Classes (IFC) models as specified in ISO 16739 (ISO, 2018); and
- Point Cloud Data (PCD) models as supported by the Cloud Compare library (Cloud-Compare, 2016).

Multiple models can be loaded at once and compared between each other (see Figure 1). Their absolute position is accounted for, so the models can be checked against one another for internal differences based on location of elements. Allowing a direct comparison between IFC and point cloud data is especially valuable in the context of ScanVsBIM approaches.

Additionally, OIP incorporates its own EXPRESS parser that consumes and evaluates data models (like IFC) specified with a schema following the ISO (2004) standard (see Figure 2). This enables automatic code generation for IFC early-binding library (Amann, 2018; Amann et al., 2018; Hecht & Jaud, 2019). As such, complete contents of IFC files produced following ISO (2016) can be interpreted and analysed. Moreover, type safety is guaranteed at compile time, thus reducing the risk for bugs and errors. This is achieved by a carefully designed schema-agnostic template library.

OIP serves as a prototypical playground for developments. The software architecture, features, and functionalities have been changed, added or removed as required along the way. Nowadays, OIP uses the IFC schema and CloudCompare's model as independent internal data models — explained in detail by Hecht & Jaud (2019). Major elements from previous developments were already realigned, while some (previously available) functionalities are still considered as work-in-progress.

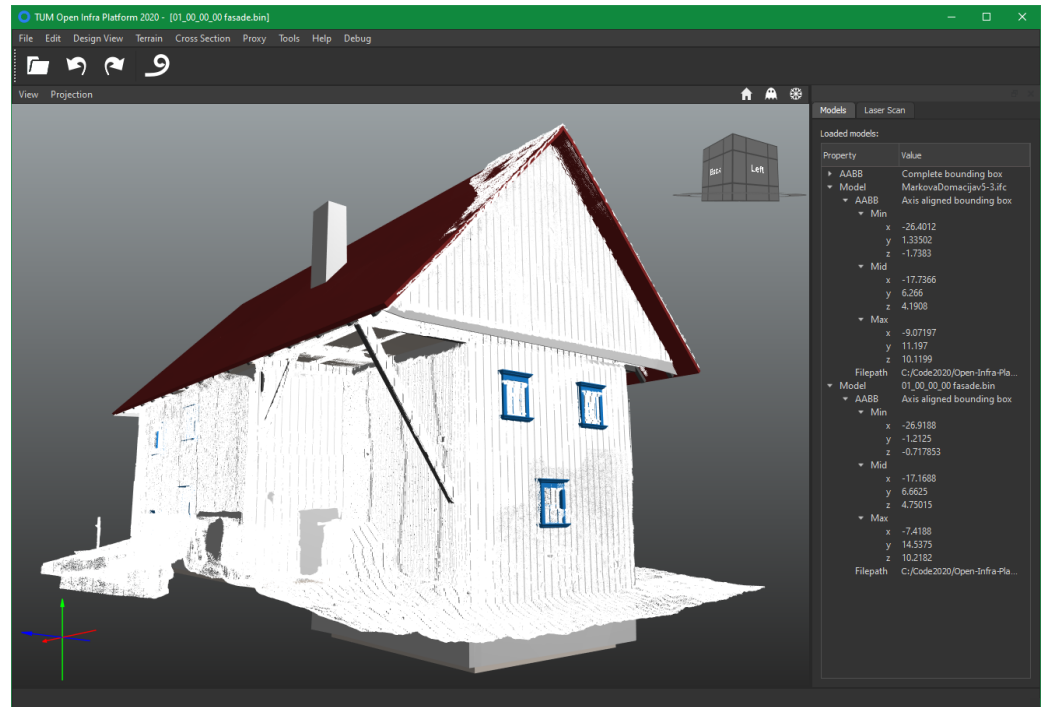


Figure 1: A PCD model together with an IFC model loaded in OIP. (PCD model courtesy of Grega Indof, IFC model courtesy of Laurens Oostwegel.)

EXPRESS Schema/ISO 10303-11

```

3576 ENTITY IfcBoundedSurface
3577 ABSTRACT SUPERTYPE OF (ONEOF
3578   (IfcBoundedSurfaceWithKnots))
3579 SUBTYPE OF (IfcBoundedSurface);
3576   lDegree : IfcInteger;
3577   vDegree : IfcInteger;
3578   controlPointsList : LIST [2..?] OF LIST [2..?] OF IfcCartesianPoint;
3579   surfaceForm : IfcBoundedSurfaceForm;
3580   uClosed : IfcLogical;
3581   vClosed : IfcLogical;
3582   selfIntersect : IfcLogical;
3583 DERIVE
3584   uUpper : IfcInteger := SIZEOF(controlPointsList) - 1;
3585   vUpper : IfcInteger := SIZEOF(controlPointsList[1]) - 1;
3586   controlPoints : ARRAY [0:uUpper] OF ARRAY [0:vUpper] OF IfcCartesianPoint;
3587   0:uUpper, 0:vUpper;
3588 END_ENTITY;

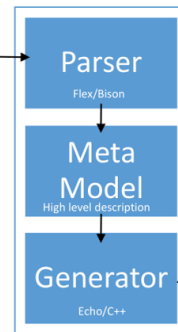
```

Clear text encoding/ISO 10303-21

```

1 ISO-10303-21;
2 HEADER;
3 FILE_DESCRIPTION(('IFC4x1', '2.1.1');
4 FILE_NAME('C:/Users/no68koc/Desktop/test.ifc', '2016-08-01');
5 FILE_SCHEMA(('IFC4x1'));
6 ENDSEC;
7 DATA;
8 #1=IFCPERSON($, 'User (FamilyName)', 'User (GivenName)');
9 #2=IFCORGANIZATION($, 'TUM', 'Chair of Computational Mechanics');
10 #3=IFCAPPLICATION(#2, 'RTM', 'TUM Open Infra Platform');
11 #4=IFCPERSONANDORGANIZATION(#1, #2, $);
12 #5=IFCOMERHISTORY(#4, #3, $, 'NOCHANGE', $, $, $, 0);
13 #6=IFCDIMENSIONALEXPONENTS(0, 0, 0, 0, 0, 0);
14 #7=IFCPROJECT('IAAGv9D1S9NAP9kC7B0pH', #5, 'IfcAlignedSurface');
15 #8=IFCUNITASSIGNMENT(#9, #10, #11, #12);
16 #9=IFCSIUNIT(*, 'LENGTHUNIT', $, 'METRE');
17 #10=IFCSIUNIT(*, 'PLANEANGLEUNIT', $, 'RADIAN');
18 #11=IFCSIUNIT(*, 'AREAUNIT', $, 'SQUARE_METRE');
19 #12=IFCSIUNIT(*, 'VOLUMEUNIT', $, 'CUBIC_METRE');
20 #13=IFCCARTESIANPOINT(0, 0, 0);
21 #14=IFCACIS2PLACEMENT3D(#13, $, $);
22 #15=IFCLOCALPLACEMENT($, #14);

```



C++11 Early Binding (ISO/IEC 14882:2011)

```

class IfcBoundedSurface
{
public:
    IfcBoundedSurface();
    IfcBoundedSurface(const IfcBoundedSurface&);
    IfcBoundedSurface& operator=(const IfcBoundedSurface&);
    virtual ~IfcBoundedSurface();
    virtual void setControlPoints(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setSurfaceForm(const IfcBoundedSurfaceForm&);
    virtual void setUClosed(const IfcLogical&);
    virtual void setVClosed(const IfcLogical&);
    virtual void setSelfIntersect(const IfcLogical&);
    virtual void setUUpper(const IfcInteger&);
    virtual void setVUpper(const IfcInteger&);
    virtual void setControlPointsList(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setDegree(const IfcInteger&);
    virtual void setVDegree(const IfcInteger&);
    virtual void setControlPointsList1(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList2(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList3(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList4(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList5(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList6(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList7(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList8(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList9(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList10(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList11(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList12(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList13(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList14(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList15(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList16(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList17(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList18(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList19(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList20(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList21(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList22(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList23(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList24(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList25(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList26(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList27(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList28(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList29(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList30(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList31(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList32(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList33(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList34(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList35(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList36(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList37(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList38(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList39(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList40(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList41(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList42(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList43(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList44(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList45(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList46(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList47(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList48(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList49(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList50(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList51(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList52(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList53(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList54(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList55(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList56(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList57(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList58(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList59(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList60(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList61(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList62(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList63(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList64(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList65(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList66(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList67(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList68(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList69(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList70(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList71(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList72(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList73(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList74(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList75(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList76(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList77(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList78(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList79(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList80(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList81(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList82(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList83(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList84(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList85(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList86(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList87(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList88(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList89(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList90(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList91(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList92(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList93(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList94(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList95(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList96(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList97(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList98(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList99(const std::vector<std::vector<IfcCartesianPoint>>&);
    virtual void setControlPointsList100(const std::vector<std::vector<IfcCartesianPoint>>&);
};

```

Figure 2: An overview over the EXPRESS parser's architecture (Amann, 2018).

Statement of Need

During the development of the IFC standard, a clear need for an independent software package capable of consuming and producing IFC files according to the newly developed schemas was determined. Additionally, an interface to other infrastructure data model standards like OKSTRA and LandInfra was needed (Amann et al., 2015; Amann & Borrmann, 2015a; BAsT, 2021; OGC, 2016). OIP fulfils this role by being open source, using CMake as a

build system generator, and being based on well-known libraries (like Qt, Eigen, Carve, and Boost). Additionally, the provided EXPRESS parser proved useful for automatically producing source code and data models from newly developed schemas when needed (Amann, 2018; Vilgertshofer et al., 2017).

There are similar open source libraries and viewers available. As a matter of fact, the geometry handling of IFC content has been taken and improved upon from the versatile *IFC++* library (*IFC++*, 2021). Note that this is not an exhaustive list — a more thorough comparison has been conducted by Hecht & Jaud (2019) and Valero et al. (2020).

On the one hand, *IfcOpenShell*, *IFC++*, *IFC.js* and *XBim toolkit* focus primarily on handling IFC content and geometries (*IFC++*, 2021; *IFC.js*, 2021; *IfcOpenShell*, 2021; Lockley et al., 2017). They provide an API to implement against (in various programming languages) together with a viewer. However, they focus mainly on the stable versions of the IFC standard (like IFC2x3 and IFC4), while OIP focuses primarily on the newer developments. Additionally, OIP supports DirectX versions 11 and 12 simultaneously through the use of the BlueFramework open-source library that serves as a basis for the rendering engine.

On the other hand, *ParaView*, *Point Cloud Library* and *CloudCompare* focus on handling PCD (*CloudCompare*, 2016; *ParaView*, 2021; *Point Cloud Library*, 2021). These provide more functionality for PCD analysis than OIP, but cannot handle BIM models. With the emergence of the Scan2BIM and ScanVsBIM research field (deriving from or merely comparing PCD and BIM models), a tool that supports both PCD and IFC data is a welcome addition to the research processes. Valero et al. (2020) selected OIP among many other software solutions as best suitable for the development of a Scan+BIM platform.

Recently, Blender received a *BlenderBIM Add-on* and can now support both IFC and PCD models, using *IfcOpenShell* and *ParaView* as supporting libraries, respectively (*BlenderBIM Add-on*, 2021). The add-on currently supports only the IFC4 version of the IFC data model, which is a major drawback if one wishes to compare PCD and IFC data of infrastructure objects. OIP bridges this gap by providing a viewer for both recent IFC versions and PCD (Valero et al., 2020).

Research Projects

The origins of this software date back to the first projects expanding IFC for infrastructure at the Technical University of Munich (Amann et al., 2014, 2015; Amann & Borrmann, 2015a, 2015b; Singer & Amann, 2014). There, first implementations of roads' geometric concepts and interfacing between multiple standards (like IFC, OKSTRA, and LandXML) were explored (see Figure 3).

Further on, many functionalities were added during the course of these projects:

- support for first experimental IFC schema additions for tunnels (Vilgertshofer et al., 2017);
- support for PCD and their analysis (Hecht, 2018);
- support for IFC Programming Language (IFC-PL) (Amann, 2018);
- support for linked data approaches (Beetz et al., 2019); and
- support for IFC4x1, IFC4x2, and IFC4x3 candidate versions of the IFC standard during their development (Jaud et al., 2020).

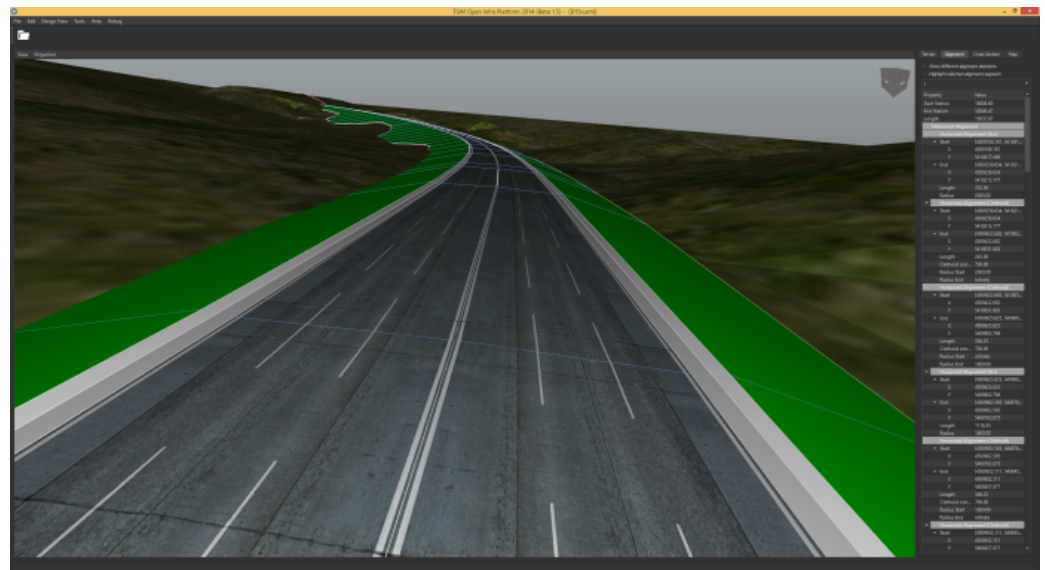


Figure 3: A perspective view of road geometry on terrain's surface with clearly marked fills.

Acknowledgements

The development of the software has been supported through many projects founded by:

- buildingSMART International Ltd.;
- Federal Ministry for Transport and Digital Infrastructure, Germany (Bundesministerium für Verkehr und digitale Infrastruktur);
- Federal Highway Research Institute, Germany (Bundesanstalt für Straßenwesen);
- German Research Foundation, Germany (Deutsche Forschungsgesellschaft);
- Leonhard Obermeyer Center, Germany; and
- Technical University of Munich, Germany.

We gratefully acknowledge their support.

References

- Amann, J. (2018). *An object-oriented language for embedding interpretation semantics in digital building models* [PhD Dissertation]. Technische Universität München.
- Amann, J., & Borrmann, A. (2015a). *Creating a 3D-BIM-compliant road design based on IFC alignment originating from an OKSTRA-accordant 2D road design using the TUM open infra platform and the OKSTRA class library* [Technical report]. Technische Universität München.
- Amann, J., & Borrmann, A. (2015b). Open BIM for Infrastructure – mit OKSTRA und IFC Alignment zur internationalen Standardisierung des Datenaustauschs. *Tagungsband Zum 6. OKSTRA-Symposium*.
- Amann, J., Flurl, M., Jubierre, J. R., & Borrmann, A. (2014, September). An alignment meta-model for the comparison of alignment product models. *Proceedings of the 10th European Conference on Product & Process Modelling*.
- Amann, J., Preidel, C., Tauscher, E., & Borrmann, A. (2018). BIM programming. In A. Borrmann, M. König, C. Koch, & J. Beetz (Eds.), *Building information modeling*. Springer.

- Amann, J., Singer, D., & Borrmann, A. (2015). Extension of the upcoming IFC alignment standard with cross sections for road design. *Proceedings of the ICCBEI 2015*.
- BASt. (2021). *OKSTRA: Objektkatalog für das Straßen- und Verkehrswesen*. Bundesanstalt für Straßenwesen. <https://www.okstra.de/>
- Beetz, J., Amann, J., & Borrmann, A. (2019). *Linked Data – Analyse von Einsatzmöglichkeiten von verbundenen Informationen (Linked Data) und Ontologien und damit befassten Technologien (Semantic Web) im Bereich des Straßenwesens* (pp. 82 pp.) [Technical report]. https://bast.opus.hbz-nrw.de/opus45-bast/frontdoor/deliver/index/docId/2181/file/LinkedData_Schlussbericht_01.0195.pdf
- BlenderBIM add-on. (2021). <https://blenderbim.org/>
- CloudCompare. (2016). <https://www.cloudcompare.org/doc>
- Hecht, H. (2018). *From Point Cloud Data to IfcAlignment* (pp. 26 pp.) [Technical report]. https://www.cms.bgu.tum.de/images/teaching/abim_seminar/Report_Hecht_From_Point_Cloud_to_IfcAlignment.pdf
- Hecht, H., & Jaud, Š. (2019). TUM OpenInfraPlatform: The open-source BIM visualisation software. *Proceedings of the 31st Forum Bauinformatik*. https://publications.cms.bgu.tum.de/2019_Hecht_Jaud_FBI.pdf
- IFC++. (2021). <https://ifcquery.com/>
- IFC.js. (2021). <https://agviegas.github.io/IFC.js/docs>
- IfcOpenShell. (2021). <https://ifcopenshell.github.io/docs/>
- ISO. (2016). *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure* (Vol. 2016) [Standard]. International Organization for Standardization. <https://www.iso.org/standard/63141.html>
- ISO. (2004). *Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual* (Vol. 2004) [Standard]. International Organization for Standardization. <https://www.iso.org/standard/38047.html>
- ISO. (2018). *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema* (Vol. 2018) [Standard]. International Organization for Standardization. <https://www.iso.org/standard/70303.html>
- Jaud, Š., Donaubaauer, A., & Borrmann, A. (2019). Georeferencing within IFC: A Novel Approach for Infrastructure Objects. In Y. K. Cho, F. Leite, A. Behzadan, & C. Wang (Eds.), *Computing in civil engineering 2019: Visualisation, information modelling, and simulation* (pp. 377–384). American Society of Civil Engineers. <https://doi.org/10.1061/9780784482421>
- Jaud, Š., Esser, S., Muhič, S., & Borrmann, A. (2020). DEVELOPMENT OF IFC SCHEMA FOR INFRASTRUCTURE. *Proceedings of 6th International Conference siBIM: Structured Data Is the New Gold*, 27–35. https://publications.cms.bgu.tum.de/2020_Jaud_siBIM.pdf
- Lockley, S., Benghi, C., & Černý, M. (2017). Xbim.essentials: A library for interoperable building information applications. *Journal of Open Source Software*, 2(20), 473. <https://doi.org/10.21105/joss.00473>
- OGC. (2016). *OGC Land and Infrastructure Conceptual Model Standard (LandInfra)* (Vol. 2016) [Standard]. Open Geospatial Consortium Inc. <https://www.ogc.org/standards/infragml>
- ParaView. (2021). <https://www.paraview.org/>

Point cloud library. (2021). <https://pointclouds.org/>

Singer, D., & Amann, J. (2014). Erweiterung von IFC Alignment um Straßenquerschnitte. *Proceedings of the 26th Forum Bauinformatik.*

Valero, E., Mohanty, D. D., & Bosche, F. (2020). Development of an open-source scan+BIM platform. In "Furuya. "Osumi Hisashi" (Ed.), *Proceedings of the 37th international symposium on automation and robotics in construction (ISARC)* (pp. 223–232). International Association for Automation; Robotics in Construction (IAARC). <https://doi.org/10.22260/ISARC2020/0033>

Vilgertshofer, S., Amann, J., Willenborg, B., Borrmann, A., & Kolbe, T. H. (2017). Linking BIM and GIS models in infrastructure by example of IFC and CityGML. In *Computing in civil engineering 2017* (pp. 133–140). <https://doi.org/10.1061/9780784480823.017>