Vonthron et al. Visualization in Engineering (2018) 6:2
DOI 10.1186/s40327-018-0061-x

Visualization in Engineering

RESEARCH Open Access

# Removing duplicated geometries in IFC models using rigid body transformation estimation and flyweight design pattern

Andre Vonthron[1*], Christian Koch[2] and Markus König[1]

## Abstract

**Background:** The digital process of Building Information Modeling (BIM) involves the creation and modification of CAD-based building models. The complexity of such models has been increasing steadily within the last few years. BIM Models are usually being exchanged using open and standardized data formats. In this context, the Industry Foundation Classes (IFC) are widely used. Therefore, software vendors provide interfaces for dealing with the IFC format. To obtain a high level of data integrity, however, IFC elements are often managed as completely distinct entities, which can result in the creation of multiple copies of identical pieces of information. Since the trend to provide web-based solutions for BIM applications is also becoming increasingly important, especially the conflict between available resource consumption and suitable response times must be considered. Although existing optimization algorithms can reduce the size of an IFC file by analyzing its structure syntactically, there is still the gap to detect identical pieces of geometries that are syntactically distinct. Also, when subsequently merging such geometries, the available sharing concepts must be questioned.

**Methods:** The contribution of this paper is twofold. On the one hand, we propose an algorithm to retrospectively detect identical geometries by estimating the rigid body transformation. On the other hand, we outline and evaluate the available possibilities for sharing geometries within the IFC data model. The so-called flyweight pattern is applied to provide and maintain the appropriate reuse of identical information.

**Results:** The methodologies are exemplary demonstrated by modeling and optimizing a typical tunnel lining structure, which contains many repetitive elements. As a result, a noticeable reduction of storage and processing time can be measured.

**Conclusions:** Establishing BIM in large building projects, where complexity not only depends on variation and geometric detail, but also depends on enormous repetition of these elements, a significant benefit is expected.

**Keywords:** Building information modeling, IFC optimization, Rigid body transformation, Flyweight design pattern

## Introduction

Building Information Modeling (BIM) has been established in construction projects worldwide and requires the creation of digital building models. These so-called BIM models do not only include a CAD-based representation of geometries, but also define semantical layers. Such a model represents a central and consistent entity during the entire lifecycle of a building project. Also, many stakeholders (for example, planners, designers, and engineers) take part in a construction project, where they attempt to modify and add data to corresponding BIM models. Because they often use different software, an open and standardized data exchange is fundamentally important. Therefore, the Industry Foundation Classes (IFC), developed and maintained by buildingSMART organization (buildingSmart 2015), are commonly used.

With the increasing scale of project size, geometric detail and linked information, the complexity of such models has also been steadily increasing. This particularly implies that the size of the resulting IFC file increases. When passing multiple import and export

* Correspondence: andre.vonthron@ruhr-uni-bochum.de
[1]Ruhr-Universität Bochum, Bochum, Germany
Full list of author information is available at the end of the article

stages, several reports (Backas 2001; Fischer and Calvin 2002; Bazjanac 2002; Pazlar and Turk 2006) have observed that it can lead to an unmanageable amount of data. Amongst other reasons, including adding unused instances, Sun et al. (2015) demonstrated that the bloat is mainly driven by producing redundant geometries for identical types of building elements, such as windows, doors, or columns. Because in BIM models data integrity is of crucial importance, it is supposed that many IFC software neglect the demand of model efficiency. That issue often plays a major role accepted as a data format in larger projects, such as BIM applications in infrastructure projects, which contain an even larger number of repetitive elements. Also, considering the demand of web-based BIM solutions the conflict between file size and limited bandwidth plays a significant role.

In fact, there is the demand of eliminating redundancy from bloat IFC files. Existing optimizers can already eliminate redundancy using syntactical analysis, but are not capable of identifying equivalence within absolute defined geometries. Since optimized IFC files will subsequently pass import and export stages again, there is also the demand to discuss sharing concepts and how they can be preserved. Therefore, a twofold methodology will be proposed. First, we introduce an algorithm to detect identical product geometries based on estimating rigid body transformation. Secondly, we discuss the possibilities of sharing identical geometries to reproduce an efficient model. Also, the second contribution includes the specification on how to deal with and preserve shared instances when performing import or export using the flyweight design pattern. For the proof of concept, a case study implements and evaluates the methodologies considering a tunnel lining model from a project in the field of mechanized tunneling.

## Background

### Industry foundation classes

Since the paper deals with the Industry Foundation Classes, a simplified and short overview of the class structure is presented. Figure 1a outlines a class diagram including the relevant inheritance graph. Three major superclasses are considered, namely *IfcProduct*, *IfcPropertyDefinition* and *IfcRelationship*. Products hold a spatial or physical context and they are either of subclass *IfcSpatialStructureElement* or *IfcElement*. Spatial structure elements maintain spaces (like building stories or rooms) or they decompose hierarchical structures. A geometric context can be assigned, but usually the semantic aspect is used exclusively. Instead, instances of *IfcElement* represent physically existing elements and should always contain a geometric context.

For the association of objects the class *IfcRelationship* realizes the principle of objectified relations, where in this context, Fig. 1b outlines aggregations, product containments and property defines. When creating a product tree of a building, the relationship class *IfcRelAggregates* decomposes different spatial structures, for example, stories within a building. In contrast, the relationships of class *IfcRelContainedInSpatialStructure* assign products to a specific spatial level, usually representing physical elements.

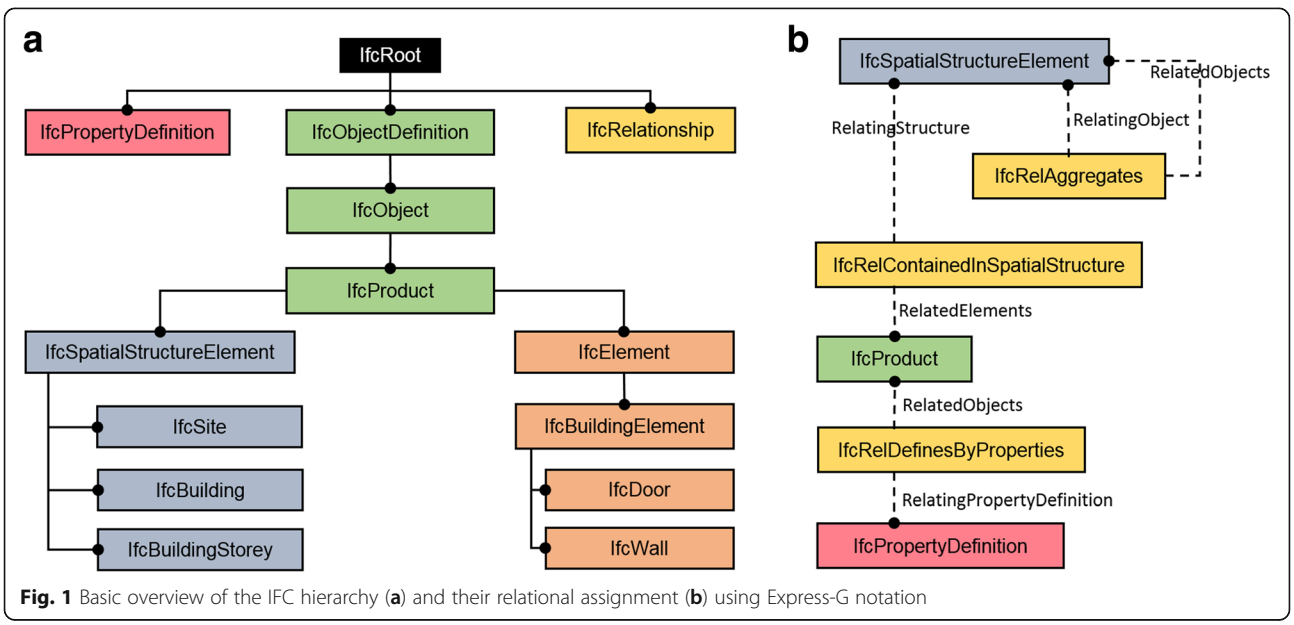



**Fig. 1** Basic overview of the IFC hierarchy (**a**) and their relational assignment (**b**) using Express-G notation

### Rigid body transformation estimation

Computing the alignment of two points sets, for which a pairwise correspondence is known, is a common application, for example in computer vision or bioinformatics. Several mathematical problem definitions exist depending on different goals. The "Orthogonal Procrustes Problem" (Hurley and Cattell 1962; Schönemann 1966) defines the search for rotational component, as also does the "Wahba's Problem" (Wahba 1965) including pairwise optional weight factors. The "absolute orientation problem" (Jones 1980) defines the search for rotation and scale components. The most recent term is "estimating 3d–rigid body transformation" (Eggert et al. 1997), which focuses on finding rotation and translation. The linear and affine transformation, which maps $N$ points of a source system, denoted as $p_i$, to corresponding points of a target system, denoted as $p_{i'}$, can generally be written as

$$p_{i'} = \alpha R p_i + t + e_i, \tag{1}$$

where $R$ is a $3 \times 3$ rotation matrix, $\alpha$ represents a uniform scale factor and $t$ represents the translational component. Also, the equation considers a pairwise error $e_i$, which is zero for congruent point sets, but can be exploited for some solutions. While scale and translational components can easily calculated (see Eq. 3-5 and Eq. 7-10) and removed from the problem definition, solving the original Orthogonal Procrustes Problem represents the most challenging part. It can be solved by using iterative methods or closed form solutions. In our case the latter are considered, because they always guarantee convergence. A typical approach involves the root mean square deviation (RMSD), which must be minimized. Using least square optimization the objective function is

$$(R, \alpha, t) = \underset{R,\alpha,t}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| p_i' - \alpha R p_i - t \right\|^2. \tag{2}$$

Four major algorithms exist to solve Eq. 2. The first and the second involve matrix representations, either using singular value decomposition (SVD) (Kabsch 1976; Kabsch 1978; Arun et al. 1987), also known as *Kabsch algorithm*, or using eigensystems of orthonomal matrices (Horn et al., 1988). The third and fourth involve quaternion representations (Horn, 1987; Walker et al., 1991). A comparative overview is given by Eggert et al. (1997).

### The flyweight design pattern

In object-oriented software design the application of design patterns are fundamentally important to create well-structured software and to maintain sophisticated concepts. Gamma et al. (1995) present an essential set of such patterns. These are subdivided into three categories, namely creational patterns, structural patterns, and behavioral patterns.

The so-called Flyweight Design Pattern belongs to the structural pattern collection. Motivated by the reuse of glyph objects (for example, letters in document editors) as well as the construction of graphical user-interfaces, it provides a methodology to share identical information efficiently. Therefore, an object is split into its extrinsic and intrinsic state. The extrinsic state depends on the identity part of the object and, thus, cannot be shared and must be created in any case. However, the intrinsic state represents a completely independent part of the object. It can be shared between multiple instances and consequently avoids the creation of redundant information. Such a shareable part is called a flyweight. Figure 2 outlines the corresponding class diagram in Unified Modeling Language (UML), giving the basic architecture for implementing the flyweight behavior. A *Client* represents an object which holds a specific extrinsic state. An intrinsic state is modeled as a flyweight object which can be shared among multiple clients (*ConcreteFlyweight*) or which can exist separately (*UnsharedConcreteFlyweight*). Either way, a mechanism must be implemented which ensures the unique existence of such objects. This can be managed by using a factory implementation which handles the creation of objects. In this case, a Flyweight Factory decides whether to allocate a new object (flyweight does not yet exist) or to provide an existing reference (flyweight is already present). A client must also pass a key, for example a specific type, to identify the requested object. The factory stores instantiated objects in an internal storage map and, therefore, functions as container (the so-called flyweight pool) for existing objects.

## Related work

In literature, the topic of IFC Optimization deals with reduction of file size. While this usually implies the optimization of existing IFC files in a postprocessed manner, this paper also covers optimization methods in the design stage to directly produce optimal files.

Considering the stage of postprocessed optimization several algorithms have been proposed to remove duplicated instances. The Solibri Ifc Optimizer (Solibri Inc. 2015) is a proprietary software which can reduce the file size. While the exact method is corporate secret, Pazlar & Turk (2007) have empirically determined, that the Solibri Ifc Optimizer merges equivalent data and updates corresponding references. Instead, Sun et al. (2015) published an open-source solution called IFCCompressor, which produces similar results and which works independently of a certain IFC schema. For both solutions, it can be noticed that a kind of flyweight pattern is reproduced from duplicated instances. Because they are just working schema- or text-based, they cannot detect

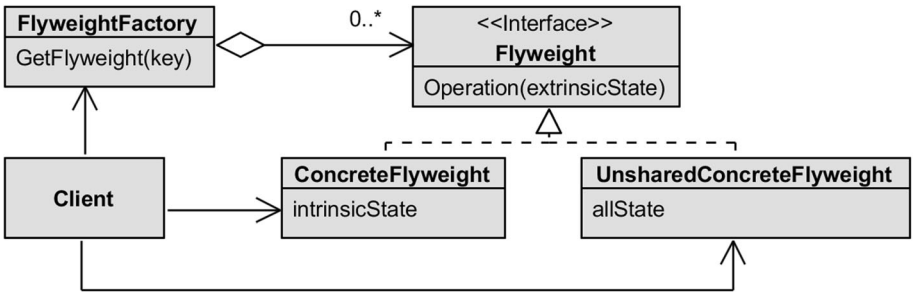



**Fig. 2** Basic UML-architecture of the flyweight design pattern (Gamma et al., 1995, p. 198)

geometric equivalence with respect to an unknown transformation. An algorithm which also contributes to such geometrical duplicates is presented by Liu et al. (2016). Their algorithm uses the Iterative Closed Point (ICP) method to detect repetitive objects and then merges duplicates by using IfcMappedItem (cf. Fig. 3a). Furthermore, it preliminarily filters the products to only compare when entity names match. However, for all results of the related algorithms, it remains unclear which requirements must be fulfilled to preserve data integrity, for example when shared objects are modified in subsequent processes.

Considering the stage of design, there exist some explicit sharing concepts defined in the IFC schema. For the reuse of geometric items the entity *IfcMappedItem* (first defined in IFC2x, c.f. Figure 3a) can be utilized. For sharing a representation, first a representation map is created to identify a representation for mapping and to define a placement origin. Then this map can be referenced as a mapping source by many mapped items to implement reuse. Furthermore, a Cartesian transformation operator can be attached to such a mapped item. Not only is the sharing of representations among multiple client product representations itself, but also the explicit knowledge about being shared a great advantage in preserving integrity on schema level.

For the reuse of properties, an attachment to the relating type object is possible (c.f. Fig. 3b). This works properly if it represents a predefined set of properties for a specific type. Sharing properties for only a subset of a client objects is forbidden by the schema which, consequently, would imply the creation of a distinct property set for each client individually. For instance, a manufactured date can be the same value for multiple clients, but is not generally a static value property of a type.

## Methods

### Basic algorithm

For the deduplication of product geometries, a three-phased algorithm (cf. Fig. 4) is proposed. The first phase addresses a semantical grouping. Grouping by Entity avoids useless comparisons, for example, by comparing geometries of *IfcWindow* and *IfcDoor*. However, when using instances of *IfcBuildingProxyElement*, this filter becomes ineffective. Therefore, a subsequent grouping by meta information, using attributes, properties or entity types can be additionally performed. The second phase compares included representations topologically as well as geometrically. In general, topological comparison is not trivial, because equivalent topologies could be defined in different manners, for example, using different decompositions or different sequences. Therefore, when

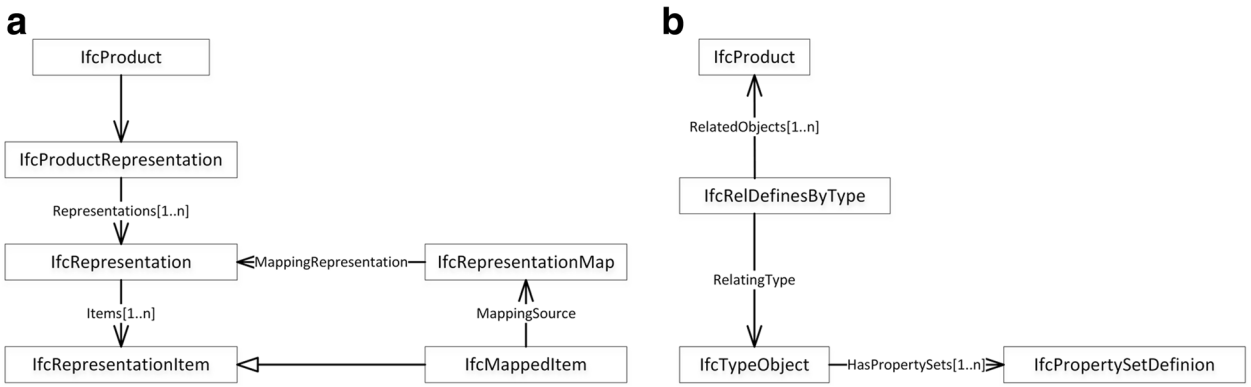



**Fig. 3** Express-G Diagrams showing the reuse concepts for geometry (**a**) and type properties (**b**)

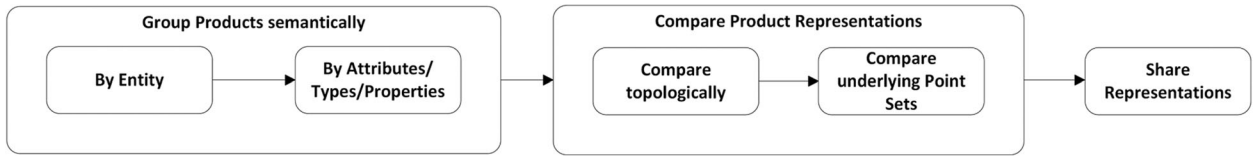



**Fig. 4** Flowchart diagram showing the three-phased algorithm sequence

having equivalent geometries, we assume that simple duplication would have produced them in the same manner. So, equivalence can be determined by traversing the instance graph of their definitions by comparing entity definitions as well as comparing the number of aggregated instances for each equivalent pair. When topological representation matches, the underlying point sets are compared. This involves the estimation of a rigid body transformation to detect a geometric correspondence. While the estimate cannot assure correctness by concept, a subsequent pairwise transformation with the estimate can verify the result. Finally, the third phase reassigns representational matchings using the flyweight design pattern to construct a model with nonredundant geometric elements.

### Comparing point sets by estimating the rigid-body transformation

When topological equivalence is given, the underlying point sets are compared. To estimate the rigid transformation, we decided to apply the Kabsch algorithm (see Section 2.2). It does not only simplify the rotational representation according to the use of matrices, but also exists in several implementations. Before application, the objective function must be reduced by eliminating the scale component. This is realized by taking the ratio of sums of the vector distances

$$\alpha = \frac{\sum_{i=1}^{N} |p_i'|}{\sum_{i=1}^{N} |p_i|} \tag{3}$$

from which then the systems can be adapted to

$$q_i' = {}^{p}i'/_{\alpha} \tag{4}$$

and

$$q_i = p_i. \tag{5}$$

The new objective function is

$$(R, t) = \underset{R,\alpha,t}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| q_i' - Rq_i - t \right\|^2. \tag{6}$$

where the Kabsch algorithm can now be directly applied. First, by calculating and subtracting the centroids of both systems, the translational component can be removed. The corresponding centroids are the mean vectors of each system, denoted as

$$\overline{q} = \frac{1}{N} \sum_{i=1}^{N} q_i \tag{7}$$

and

$$\overline{q'} = \frac{1}{N} \sum_{i=1}^{N} q_i' \tag{8}$$

Then, performing subtraction of these individual centroids, the new centroids of both systems are projected to world's origin, denoted as

$$q_{c,i} = q_i - \overline{q} \tag{9}$$

and

$$q_{c,i}' = q_i' - \overline{q'}. \tag{10}$$

Using such scale and translation aligned systems, the objective function from Eq. 6 can be further simplified to

$$R = \underset{R}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| q_{c,i}' - Rq_{c,i} \right\|^2. \tag{11}$$

Considering the new objective function, its inner expression can be expanded and simplified to

$$\begin{aligned} \left\| q_{c,i}' - Rq_{c,i} \right\|^2 &= \left( q_{c,i}' - Rq_{c,i} \right)^T \left( q_{c,i}' - Rq_{c,i} \right) \\ &= \left( q_{c,i}'^T - q_{c,i}^T R^T \right) \left( q_{c,i}' - Rq_{c,i} \right) \\ &= q_{c,i}'^T q_{c,i}' - q_{c,i}'^T Rq_{c,i} - q_{c,i}^T R^T q_{c,i}' + q_{c,i}^T R^T Rq_{c,i} \\ &= q_{c,i}'^T q_{c,i}' - q_{c,i}'^T Rq_{c,i} - q_{c,i}^T R^T q_{c,i}' + q_{c,i}^T q_{c,i} \\ &= q_{c,i}'^T q_{c,i}' - 2q_{c,i}'^T Rq_{c,i} - q_{c,i}^T q_{c,i}. \end{aligned} \tag{12}$$

Because each rotation matrix $R$ satisfies the orthogonal condition $R^T R = I$, it can be eliminated. Furthermore, the terms which include $R$ can be merged since for any scalar $a$ it holds that $a = a^T$. Because the corresponding terms result to a scalar value, they can be exchanged so that

$$q_{c,i}^T R^T q_{c,i}' = \left( q_{c,i}^T R^T q_{c,i}' \right)^T = q_{c,i}'^T Rq_{c,i}. \tag{13}$$

Now, substituting the expression of Eq. 11 with the result from Eq. 12, this leads to:

$$
\begin{aligned}
R &= \underset{R}{\operatorname{argmin}} \sum_{i=1}^{N} \left( {q'_{c,i}}^{T} q'_{c,i} - 2 {q'_{c,i}}^{T} R q_{c,i} + {q_{c,i}}^{T} q_{c,i} \right) \\
&= \underset{R}{\operatorname{argmin}} \left( \sum_{i=1}^{N} \left( {q'_{c,i}}^{T} q'_{c,i} \right) - 2 \sum_{i=1}^{N} \left( {q'_{c,i}}^{T} R q_{c,i} \right) + \sum_{i=1}^{N} \left( {q_{c,i}}^{T} q_{c,i} \right) \right) \\
&= \underset{R}{\operatorname{argmin}} \left( -2 \sum_{i=1}^{N} {q'_{c,i}}^{T} R q_{c,i} \right) \\
&= \underset{R}{\operatorname{argmax}} \left( \sum_{i=1}^{N} {q'_{c,i}}^{T} R q_{c,i} \right).
\end{aligned} \tag{14}
$$

Terms not depending on $R$ and leading scalar multipliers can be excluded because they won't affect the minimization function. Also, by flipping the sign, the minimization problem can be reformulated to a maximization problem. Before finding the maximum, we note the following expression:

$$
\begin{aligned}
Q'\mathrm{R}\mathrm{Q} &= \begin{bmatrix} {q'_{c,1}}^{T} \\ {q'_{c,2}}^{T} \\ \vdots \\ {q'_{c,N}}^{T} \end{bmatrix} R \begin{bmatrix} q_{c,1} & q_{c,2} & \cdots & q_{c,N} \end{bmatrix} \\
&= \begin{bmatrix} {q'_{c,1}}^{T} R q_{c,1} & & & \\ & {q'_{c,2}}^{T} R q_{c,2} & & \\ & & \ddots & \\ & & & {q'_{c,N}}^{T} R q_{c,N} \end{bmatrix},
\end{aligned} \tag{15}
$$

where $Q'$ represents the matrix composed of ${q'_{c,i}}^{T}$ line-by-line, and $Q$ represents the matrix composed of $q_{c,\,i}$ column-by-column. Using matrix trace, where $\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii}$ with the commutative property $\mathrm{tr}(AB) = \mathrm{tr}(BA)$, we can express the maximization as

$$
\begin{aligned}
\underset{R}{\operatorname{argmax}} \left( \sum_{i=1}^{N} {q'_{c,i}}^{T} R q_{c,i} \right) &= \underset{R}{\operatorname{argmax}} (\mathrm{tr}(Q'RQ)) \\
&= \underset{R}{\operatorname{argmax}} (\mathrm{tr}((Q')(RQ))) \\
&= \underset{R}{\operatorname{argmax}} (\mathrm{tr}(RQQ')).
\end{aligned} \tag{16}
$$

Subsequently, the correlation matrix is denoted as $H = QQ'$. Therefore, a solution which maximizes $\mathrm{tr}(RH)$ can be found by factorizing $H$ using singular value decomposition. The factorization is given by $H = U\Lambda V^T$, where $U$ and $V$ represent unitary matrices and $\Lambda$ is a diagonal matrix with elements $\sigma_i \geq 0$. Substituting the decomposition into the trace, and using its commutative property, again, it can be expressed as

$$
\mathrm{tr}(RH) = \mathrm{tr}\left(RU\Lambda V^T\right) = \mathrm{tr}\left(\Lambda V^T R U\right). \tag{17}
$$

Because $V$, $R$ and $U$ are orthogonal, the product $S = V^T R U$ is orthogonal as well. Consequently, each column vector $s_j$ of $S$ is orthonormal and $s_j^T s_j = 1$. For each entry $s_{ij}$ of $S$ it holds that $|s_{ij}| \leq 1$, because

$$
\begin{aligned}
& {s_j}^T s_j = 1 = \sqrt{\sum_{i=1}^{3} {s_{ij}}^2} \qquad\qquad j = 1\ ,..,3 \\
& \Rightarrow \sum_{i=1}^{3} {s_{ij}}^2 = 1 \Rightarrow {s_{ij}}^2 \leq 1 \Rightarrow \left| s_{ij} \right| \leq 1.
\end{aligned} \tag{18}
$$

Considering the trace, it easily follows that

$$
\mathrm{tr}(\Lambda S) = \begin{pmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3 \end{pmatrix} \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix} = \sum_{i=1}^{3} \sigma_i s_{ii} \leq \sum_{i=1}^{3} \sigma_i
$$

and, consequently, it is maximized when each $s_{ii} = 1$. This is true if and only if $S$ equals the identity matrix $I$. In this case

$$
\begin{aligned}
& I = S \\
& \Leftrightarrow I = V^T R U \\
& \Leftrightarrow \left(V^T\right)^{-1} = RU \\
& \Leftrightarrow R = \left(V^T\right)^{-1} U^{-1} = \left(V^T\right)^T U^T = VU^T.
\end{aligned} \tag{19}
$$

The derived result gives the rotation matrix for any rotation, except in case of reflection. A reflection is expressed by reversing a row vector of a rotation matrix. Because of a potentially odd number of changed signs, $\det(VU^T)$ can become −1. In such a case, when $S$ does not equal the identity matrix exactly, the last element $\sigma_3 = -1$. Therefore, $V$ would be multiplied by exactly this matrix to create a valid rotation. Thus, in general, S can be rewritten as

$$
S = \begin{pmatrix} 1 & & \\ & 1 & \\ & & \det(VU^T) \end{pmatrix}. \tag{20}
$$

Considering rotations as well as reflection, the optimal rotation matrix can then be expressed as

$$
R = VSU^T. \tag{21}
$$

After calculating the required parameters, a transformation matrix $M$, which fulfills the homogenous transformation equation (cf. Eq. 22) can be constructed by concatenating the sub-transformations in a homogenous manner (cf. Eq. 23),

$$
\begin{pmatrix} p'_i \\ 1 \end{pmatrix} = M \begin{pmatrix} p_i \\ 1 \end{pmatrix}, \tag{22}
$$

$$M = T'RAT^{-1}, \tag{23}$$

where

$$T' = \begin{pmatrix} 1 & 0 & 0 & \overline{p}'_x \\ 0 & 1 & 0 & \overline{p}'_y \\ 0 & 0 & 1 & \overline{p}'_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, A = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T = \begin{pmatrix} 1 & 0 & 0 & \overline{p}_x \\ 0 & 1 & 0 & \overline{p}_y \\ 0 & 0 & 1 & \overline{p}_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Finally, when a transformation matrix $M$ aligning two point sets has been calculated, Eq. 22 must be checked against all pairs of points. The point sets are equivalent if and only if each pair fulfills that equation. Lastly, if two product representations are detected to be equal, the representation of the second system can be substituted by the representation of the first system. Consequently, the matrix corresponding to the actual placement of the object in the second system must be pre-multiplied by the transformation matrix $M$.

Iteration over large points sets as well as performing matrix operations is very time-consuming. Therefore, the time complexity of the algorithm is determined and classified using Big O Notation. For the identification of the relevant quantities of an IFC Model, the number of all instances is denoted as $N_i$ and the number of products is denoted as $N_P$. The number of points within a product is still denoted as $N$, but must be considered as an average amount of underlying points for products in a model. Based on these quantities, Table 1 presents the time complexity of the algorithm's subroutines. All instances can be managed by using a binary search tree. Beginning with (1) semantical grouping, the types of the instances can be accessed in $O(1)$ and properties in $O(N_i + \log N_i)$. For all instances, step (1) concludes to $O(N_p(N_i + \log N_i))$. Determining the time complexity of the geometric comparison (2), first topological comparison is considered. The comparison of the underlying tree has the complexity $O(N \log N)$. The comparison of the underlying point sets is of complexity $O(N^2)$. Considering a worst-case scenario of potential comparisons, an instance is compared with $(N_p - 1)$ other instances and then the remaining with $(N_p - 2)$ instances and so on, which is the series $(N_p-1)\cdot(N_p-2)\cdot\ldots\cdot 1 = \frac{N_p(N_p-1)}{2}$ and in Big O Notation concludes to $O\left(N_p^2 N^2\right)$.

**Table 1** Time complexity using O-Notation

| Subroutine | Complexity | |
|---|---|---|
| 1) Semantical grouping | | $O(N_p(N_i + \log N_i)$ |
| By type | $O(1)$ | |
| By property | $O(N_i + \log N_i)$ | |
| 2) Geometric comparison | | $O\left(N_p^2 N^2\right)$ |
| Topological comparison | $O(N \log N)$ | |
| Compare underlying point sets | $O(N^2)$ | |
| Removing scale and translation | $O(N)$ | |
| Create H = Q*Q' | $O(N^2)$ | |
| Singular value decomposition | $O(1)$ | |
| Point comparison p' = M*p | $O(N)$ or $O(N^2)$ | |
| 3) Share identified duplicates | | $O(N_p N^2)$ |
| Assign representation | $O(1)$ | |
| Pre-multiply transformation | $O(N^2)$ | |
| $\sum$ | | $O(N_p(N_i \log N_i + N_p N^2))$ |

Finally, the overall complexity concludes to

$$\begin{aligned} & O\left(N_p(N_i \log N_i) + O\left(N_p^2 N^2\right) + O\left(N_p N^2\right)\right) \\ & = O(N_p\left(N_i \log N_i + \left(N_p + 1\right)N^2\right) \\ & = O\left(N_p\left(N_i \log N_i + N_p N^2\right)\right). \end{aligned} \tag{24}$$

This means that the complexity mainly depends on the number of products $(N_p)$ in an IFC model and the number of the point sets $(N)$. Furthermore, when utilizing a grouping based on related properties, this also adds a specific factor $(N_i \log N_i)$. The overall complexity is polynomial, though it includes potential for parallel optimization, which promises reduction to the outer factors $N_p$ and $N_i$.

### Sharing data using implicit flyweights

Implicit sharing of identical parts can be applied throughout most entities within the IFC. Practically, any instance could be referenced by a multitude of other instances. However, for a shared instance it is not necessary to know whether it is to be a shared or non-shared component. According to the IFC Schema, there are some restrictions necessary to preserve model quality. On the one hand, when an entity includes an inverse attribute to a client entity with cardinality 1, an instance of the actual entity must not be shared amongst a multitude of these client objects. If it was shared, the inverse attribute could not be resolved consistently, and thus would violate model integrity. A typical example is the entity *IfcProductDefinitionShape*, considering the inverse attribute 'OfProductDefintionShape'. In IFC2x3 it refers to a single instance of *IfcProduct*. Since IFC4, it has been relaxed to include multiple products. On the other hand, sharing geometric content does not only include pure instances, but also involves a distinction between local definitions and how these are placed into a world coordinate system applying multiple local transformations. Because the IFC payload normally includes a high ratio of such definitions, there is the need for specific investigation.

Supporting internal program models to decompose the geometric model representation into appropriate intrinsic and extrinsic states, they need to adopt the

concept of scene graphs. Basically, a scene graph is a tree structure which decomposes a 3D scene into spatial structures, where each structure defines its specific local coordinate system. Besides a transformational component, each structure can define further local properties, states or levels of detail. Subsequently, the decomposition of geometric representation into intrinsic definition and placement can be derived. Geometric components, like shapes, can be implemented as an intrinsic part and, consequently, be shared among different leaf nodes. Instead, the parts that differ among client objects and that are responsible for their identification or behavior must be implemented extrinsically.

To guarantee an internal scene graph structure, geometric models should be mapped to the IFC as described. Groups of a scene graph are mapped to entities of *IfcSpatialStructureElement*. The corresponding composition type should be chosen as "partial" for sub nodes and as "element" for the root node. The leaf nodes, which include geometric representations, are mapped to *IfcElement* entities. To share geometric representations implicitly within the IFC, different levels can be considered. For the definition of the visual context, *IfcProductRepresentation* is the most upper entity related to a product. A product representation includes a list of entities *IfcRepresentation*, which can represent shapes (*IfcShapeModel*) or styles (*IfcStyleModel*). Furthermore, every representation can include 1-to-many occurrences of *IfcRepresentationItem* to provide more detailed subdivisions. Considering these three levels, sharing information at the most upper level is mandatory to achieve minimum resource consumption. For example, sharing the representation of identical products, the level of product representation is most adequate (Fig. 5b). When geometry and related styles must be divided, the representation level fulfills the task most properly (Fig. 5c), and lastly, when also parts of shapes should be shared amongst a multitude of representations, the item level should be considered (Fig. 5d).

In case of creating flyweights on product representation level, the local placement of *IfcElement* serves as local coordinate system. Hence, using a product representation amongst multiple products, the individual linear transformation (or local placement) can be applied by using the *IfcAxis2Placement3D* entity. A corresponding $4 \times 4$ transformation matrix is converted as follows. Initially, the last row can be discarded, because it is used only for technical reason calculating homogenous transformation products. The translation component (4th column) is stored as position attribute, the $z$-Axis (3rd column) is stored as axis attribute and the $x$-Axis (1st column) is stored as 'RefDirection'. The $y$-Axis is not stored explicitly since it can be derived from the cross product of $\vec{x} \times \vec{z}$. In case of creating flyweights on the level of *IfcRepresentation,* implicit sharing is more restricted. While shape models or style models can be shared separately, a shape model still depends on its initial transformation and the corresponding coordinate system of *IfcElement.* To apply further local transformation, utilizing *IfcMappedItem* would be necessary. Lastly, constructing flyweights on the level of representation items enables the finest detail of sharing information. However, applying further transformations can only be applied using a mapped item again.

Implementing flyweight structures using implicit mappings not only avoids redundancy of geometric components, but also enables implicit sharing for non-geometric instances. However, for importing and editing IFC files, the behavior in case of modifying shared objects must be considered. When a shared object is being modified, it would affect each client object and therefore could easily violate the integrity. Consequently, some software vendors interpret each object reference as a new copy. In contrast, implementing a Copy-On-Write (COW) pattern would be more efficient. A new copy representing an Unshared Concrete Flyweight (cf. Section Rigid Body Transformation Estimation) is created only when a modification of an object is attempted. This pattern does not require the knowledge about inverse relations and, thus, can be implemented easily. However, when the amount of modification increases, it is likely that duplicates might be produced again. Otherwise, a full resolution and tracking of all references can provide a complete solution. Such a dependency tree can still be managed quite efficiently, for instance, using data structures like binary balanced trees.

## Results and discussion

For the proof of concept, a case study implements the presented methodologies. It deals with an IFC-based tunnel model as being a representative model of a large BIM infrastructure project. Initially, the case study presents different ways the model could be created, including duplication affected versions, but also presents models using sharing strategies. Subsequently, the deduplication algorithm (cf. Section Rigid Body Transformation Estimation) analyses and improves the affected models. Simultaneously, the affected models are also processed using the Solibri IFC Optimizer (Solibri Inc. 2015) as well as the IFCCompressor (Sun et al. 2015). The processed models are compared against the original model, against the models processed by the existing optimizers, and against a prototype shared model. Measuring efficiency in storage consumption, we use metrics based on the file size and derived metrics, such as optimization ratio (Eq. 25) to express the relative portion of a processed file

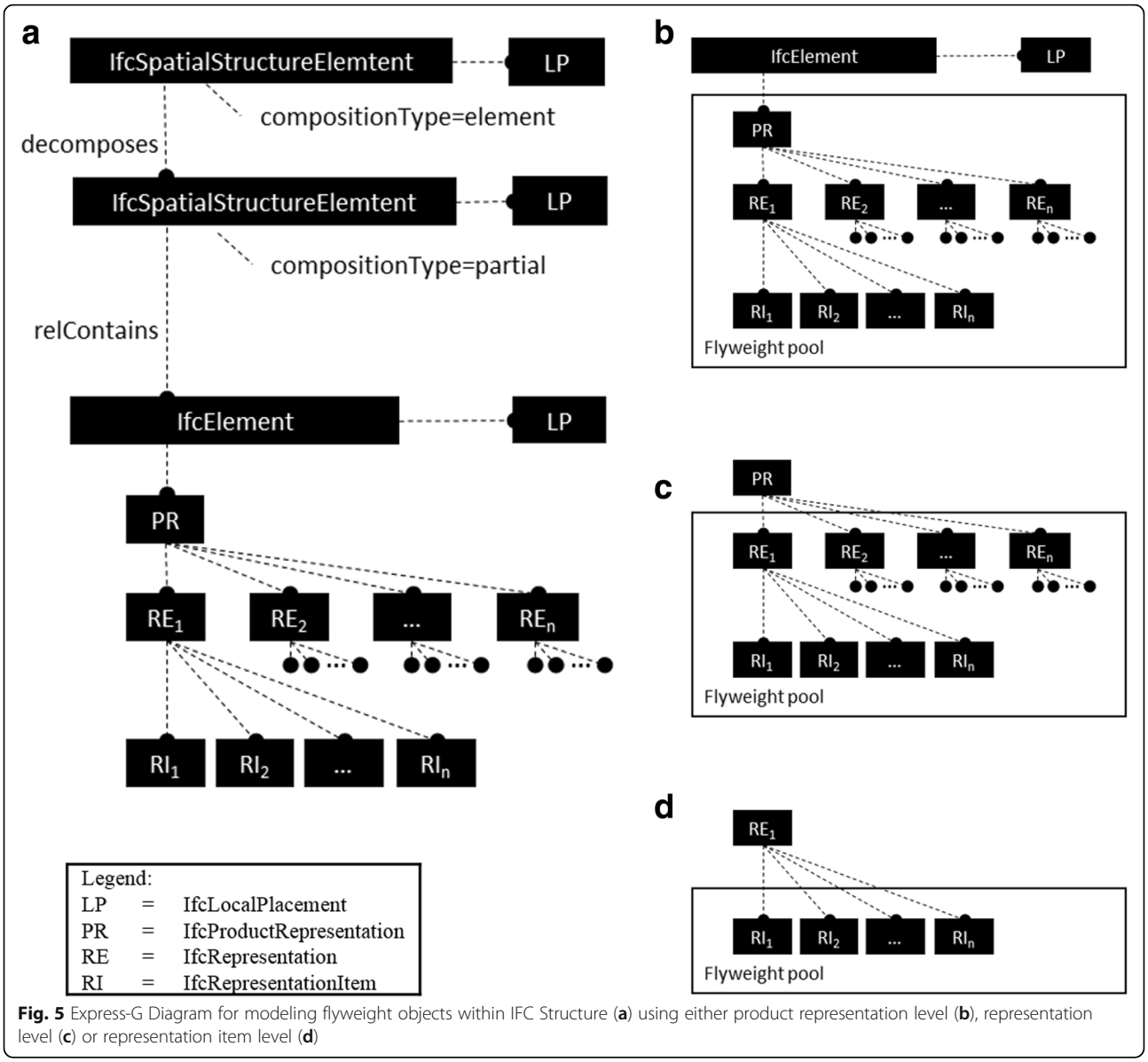



**Fig. 5** Express-G Diagram for modeling flyweight objects within IFC Structure (**a**) using either product representation level (**b**), representation level (**c**) or representation item level (**d**)

to the original input file, and an optimization rate (Eq. 26) to express the percentage of which an original model has been reduced,

$$optimzation\ ratio = \frac{optimized\ file\ size}{original\ file\ size}, \tag{25}$$

$$optimzation\ rate = 1 - optimization\ ratio. \tag{26}$$

Furthermore, the models are investigated in terms of completeness by visual verification and by comparing the number of relevant instances to relocated in processed models. Besides the investigation of model qualities, also the efficiency of the algorithm itself by means of processing time is analyzed.

Starting with a detailed description of the cases study, it deals with data from the metro line tunneling project "Wehrhahnline" in Düsseldorf, Germany. The corresponding tunnel model represents the tube of the eastern branch of the project. It consists of 530 tunnel rings arranged in succession. These rings are entirely equivalent in structure and geometry, whereas positioning along the tunnel axis represents distinct information. Each ring itself is decomposed of eight prefabricated ring segments.

Based on that given cross-sectional parameters as well as based on a given alignment, a 3D−model can be parametrically derived. The geometrical model has been generated using the Rhinoceros software (McNeel 2017) and the Grasshopper Plugin (Davidson 2017). Adapting

an IFC layer to the generated geometry model, the IFC Engine DLL (RDF Ltd. 2017) has been linked to Grasshopper as C# library. Because IFC4 standard does not include specific classes for tunneling, we use *IfcBuilding* class for spatial decomposition as well as *IfcBuildingProxyElement* class for representing physical elements. Therefore, Fig. 6 exemplary shows the instancing of one segment within the spatial structure tree.

The spatial structure of the entire tunnel is represented by an *IfcBuilding* instance using the composition type "element". It aggregates a list of *IfcBuilding* instances itself, which are of composition type "partial" representing spatial structures for the tunnel rings. Each of these spatial structures contains 8 instances of *IfcBuildingElementProxy*, which represent the segments semantically. Furthermore, individual properties are attached by using property sets. Here, for example, a property called 'ringNumber' has been linked for each spatial structure of a ring to characterize the built sequence. The geometric layer is implemented using representation items of the entity *IfcFaceBasedSurfaceModel*. Figure 7 depicts the geometric extent of the entire tunnel model.

Based on these descriptions, models could have been created in various manners of including duplications or flyweight arrangements. Considering models which are affected by duplication, we separate between geometries using model coordinates and geometries using world coordinates. When using model coordinates, for each tunnel ring a template geometry would be placed into a local coordinate system, which assigns a specific transformation. Duplicating the ring geometry leads to entirely equivalent geometric definitions. In contrast, when using world coordinates, the coordinates of such geometries would already have been transformed into world coordinates. Such finalized geometries lead to completely distinct information representing the same geometry. For models using sharing strategies, only model coordinates are considered, because the strategies require these in principle. Therefore, we subdivide into explicit and implicit shared contents. Both categories have been described and thus we refer to Related work section for explicit sharing using *IfcMappedItem* as well as we refer to Methods section for implicit sharing using the levels of *IfcProductRepresentation*, *IfcRepresentation* and *IfcRepresentationItem*.

Producing the duplication scenarios by modeling and exporting the original model, using model coordinates produces approx. 120 Megabytes and using world coordinates produces approx. 123 Megabytes. There is no important difference to observe yet. Conversions from numbers to ASCII characters cause the actual gap in file size. In contrast, the described sharing strategies produce models which result in enormously reduced file sizes, outlined by Fig. 8. The corresponding sizes are in range between 3.50 Megabytes and 5.02 Megabytes. In this case, using shared geometries consumes up to 97.1% less storage than non-shared versions. To put these initial results into context, it can easily be

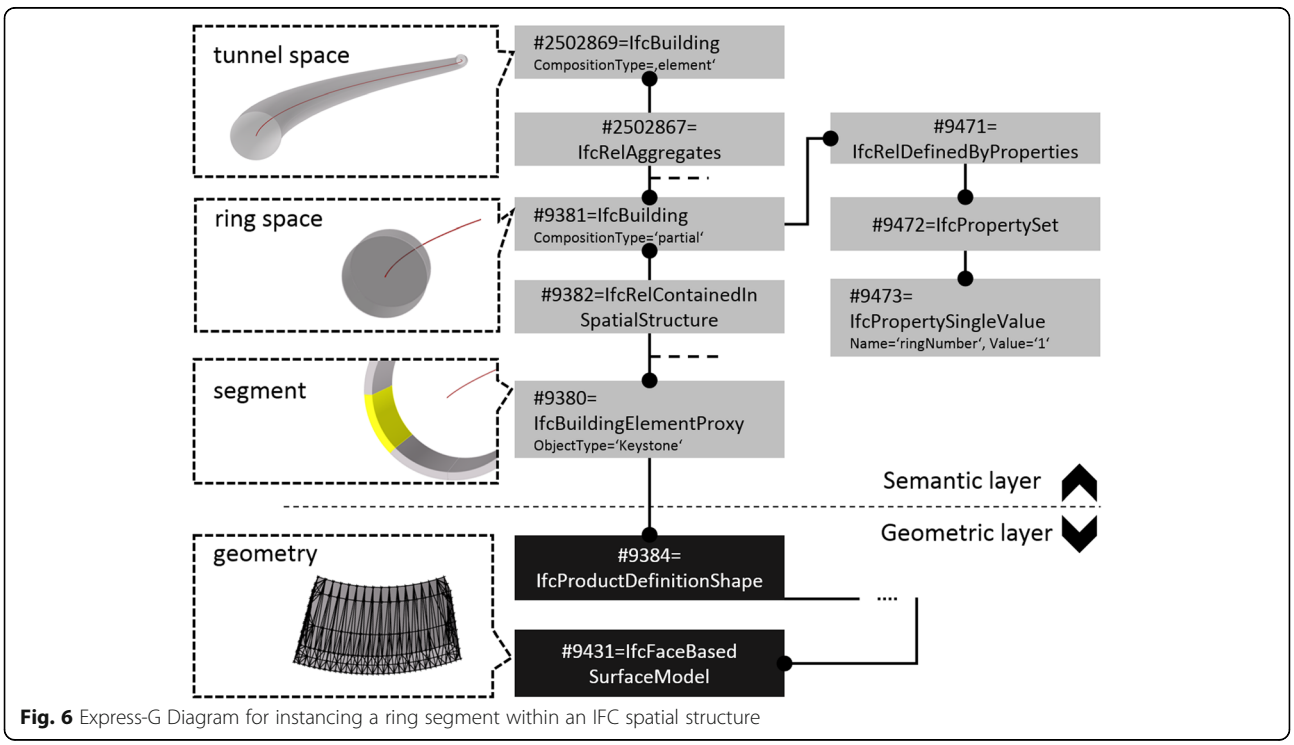



**Fig. 6** Express-G Diagram for instancing a ring segment within an IFC spatial structure

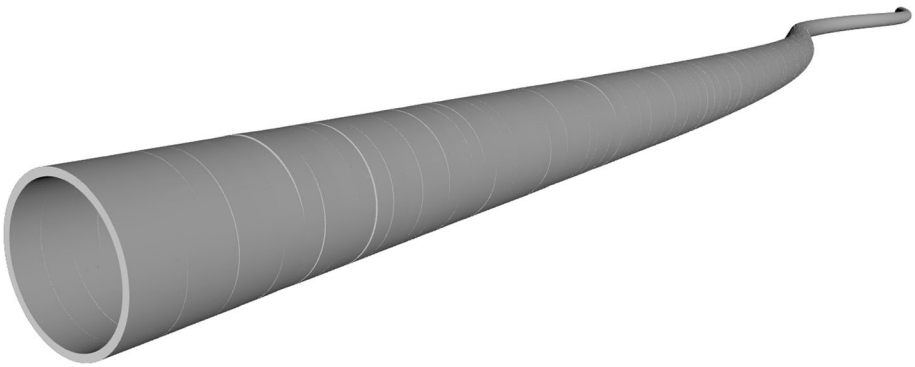


**Fig. 7** Geometric extent of the parametrically derived tunnel model

noticed that such methods can add significant benefit to the design stage, for example, when sharing such models over the Internet.

Furthermore, also providing benefit for subsequent interactions, optimization algorithms must be applied. Existing optimizers can provide enormous reduction rates when performing the version of the tunnel model which uses model coordinates. Figure 9 outlines the corresponding results. In the case of model coordinates version, the Solibri Optimizer reduces the file size about 95%, which consumes 5.96 Megabytes of storage space, as well as the IFCCompressor reduces the file size about 93%, which consumes 7.79 Megabytes of storage space. However, when processing the model version which is based on world coordinates, no significant change can be observed. In contrast, the algorithm proposed by the authors would also provide a successful reduction, even in case of using world coordinates.

For the described tunnel model using world coordinates, the presented algorithm is utilized with specific input parameters and produces intermediate result as follows. Performing phase one, the grouping of products by entity type results into 3 groups, distinctly containing either instances of *IfcSite*, *IfcBuilding* or *IfcBuildingElementProxy*. The site, of course, appears only once. The instances of *IfcBuilding* number in total 531, representing one spatial structure of the tunnel and 530 included spatial structures of the rings. The group of proxy instances includes 4240 elements. When performing successive grouping by the value of attribute 'ObjectType', the proxy collection is subdivided into eight groups of 530 elements each. Then, phase 2 concentrates only on elements which include geometry. Performing topological comparison, first, it is recognized that each element includes exactly one instance of *IfcProductRepresention*, which exactly references to one instance of *IfcShapeModel*, which again references exactly to one instance of *IfcFaceBasedSurfaceModel*. Secondly, performing topological comparison also on these representation items, the same structures and the same number of instances are detected. In this subroutine, the underlying points of instance *IfcCartestianPoint* are also collected. When topological comparison terminates, in this case, the groups remain as previously constructed. Subsequently, these elements are compared geometrically. For the specific implementation of estimating the rigid body transformation, the software library Eigen (Jacob and Guennebaud, 2017) has been utilized to easily handle matrices in source code as well as computing fast singular value decomposition using Jacobian SVD method. After geometric comparison, the groups cannot be further subdivided, because all products within these groups has been

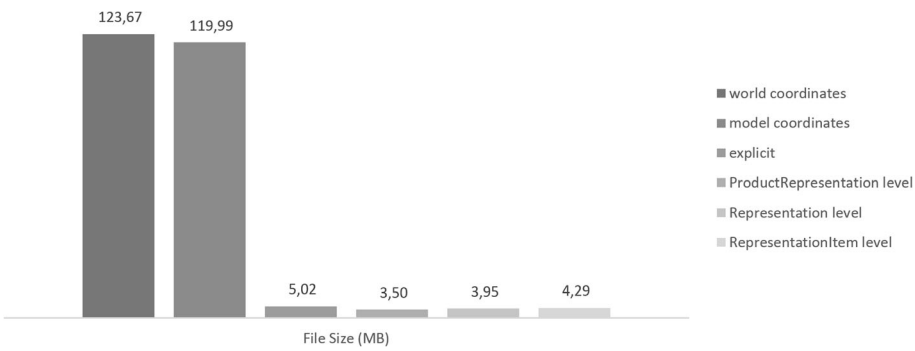



**Fig. 8** Storage requirements of the IFC Tunnel Model using different modeling strategies

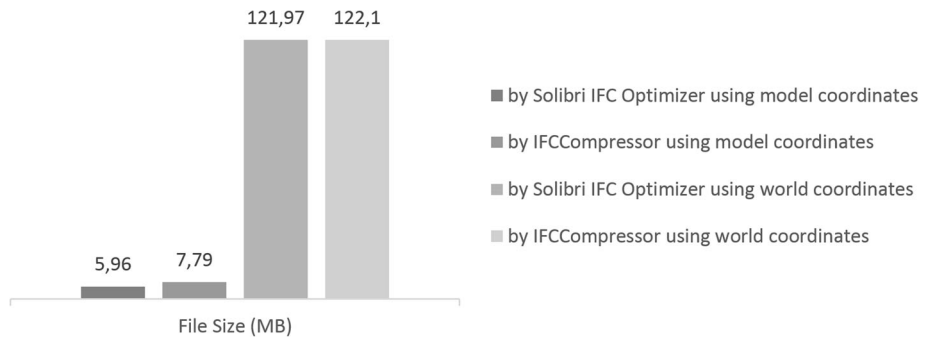



**Fig. 9** Optimization results of existing optimizers using model coordinates versus world coordinates

found to be equivalent. Though, there are additional results by means of reconstructed transformations relative to the first appearance of a geometric construct. Last, for each group of products their representation is set to a single flyweight instance and the transformation results are pre-multiplied to the corresponding placement. Therefore, the utilized sharing concept depends on the initial configuration when starting the algorithm. However, we show results for any strategy. Because utilizing the same API's for setting a 3D tunnel model into an IFC context as well as for performing the algorithm on the IFC instances, it turns out that the same results have been created, which has already been outlined by Fig. 8.

The algorithm has been compiled and run on a OSX 10.12.4 system using a 2.4Ghz Inter Core i5 processor, 16GB 1333Mhz RAM and Inter HD Graphics 3000 512 MB. Performing the tunnel model using world coordinates the overall processing time is in average 20.875 s. Considering partial time periods, the grouping procedure only took 5 ms, which in this case is caused by the linear time complexity $O(N_p)$. Instead, most of the processing time is consumed by comparing products within the eight groups, which are in range of approx. 2500 to 2700 ms (cf. Table 2).

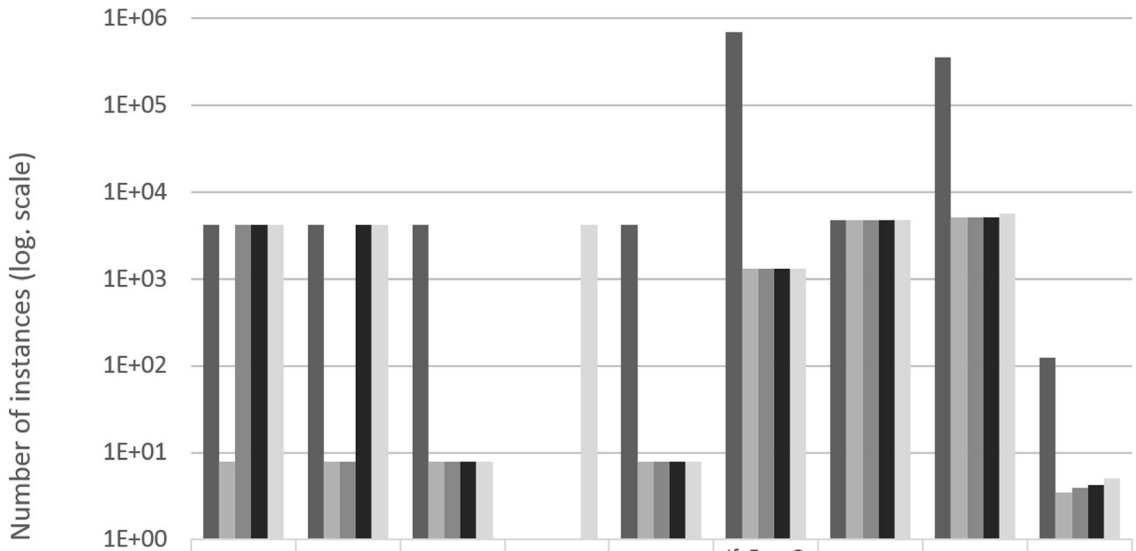
Fortunately, in this case, the subroutine terminates after ($N_p$-1) comparisons, because products have been found group-wise equivalent. However, the specific complexity $O(N_p\ N^2)$ remains of polynomial scale, which explains the large processing times. Lastly, sharing representations takes 47 ms, which also comes from the $O(N_p)$ complexity, but is somewhat higher than the grouping routine and caused by the factor needed for multiplying transformation matrices.

For the verification of the results, also visual output as well as the number of specific product and product-related instances are analyzed. Visual output is verified by visual perception, where no differences can be observed compared to the geometric model which already has been depicted by Fig. 7. Semantical integrity is verified by analyzing the number of instances of specific classes. Comparing the number of instances, each of the models includes 531 occurrences of *IfcBuilding*, representing 1 spatial structure for the overall tunnel and, of course, 530 included spatial structures for representing rings. But with respect to modeling strategy, Fig. 10 outlines significant differences.

Considering the world coordinates based model, for each segment it utilizes completely new unshared instances starting from *IfcProductDefinitionShape* down to *IfcCartesianPoint*. Until *IfcConnectedFaceSet* it remains single duplication, but when defining faces and corresponding coordinates, it becomes clear why the file size bloats up. Instead, using a sharing concept, it dramatically reduces the number of instances in the latter case. Because of the eight different segments type of the tunnel rings, eight flyweight geometries exist. For these elements, of course, the underlying instance path must be created anyway. In comparison, most numbers reduce by the scale of 530, which equals the number of rings. Placements remain the same, because they represent distinct information. Considering the use of representation level and item level, they lose a little efficiency by using distinct instances on either product representation or representation level. Lastly, the explicit mapping method consumes similar amounts compared to the item level strategy. It creates explicit geometric representations for

**Table 2** Time consumption for product comparisons within the generated groups of IFC instances

| Entity | Object type | Time consumption [ms] |
|---|---|---|
| IfcBuildingElementProxy | "Keystone" | 2534 |
| IfcBuildingElementProxy | "Counter Segment Left" | 2550 |
| IfcBuildingElementProxy | "Counter Segment Right" | 2518 |
| IfcBuildingElementProxy | "A" | 2528 |
| IfcBuildingElementProxy | "B" | 2618 |
| IfcBuildingElementProxy | "C" | 2709 |
| IfcBuildingElementProxy | "D" | 2693 |
| IfcBuildingElementProxy | "E" | 2673 |




| | IfcProductDefinitionShape | IfcShapeModel | IfcFaceBasedSurfaceModel | IfcMappedItem | IfcConnectedFaceSet | IfcFaceOuterBound, IfcPolyLoop, IfcFace | IfcAxis2Placement3D | IfcCartesianPoint | File Size (MB) |
|---|---|---|---|---|---|---|---|---|---|
| world coordinates | 4240 | 4240 | 4240 | 0 | 4240 | 695360 | 4770 | 360930 | 123,67 |
| Product Representation level | 8 | 8 | 8 | 0 | 8 | 1312 | 4770 | 5106 | 3,5 |
| Representation level | 4240 | 8 | 8 | 0 | 8 | 1312 | 4770 | 5106 | 3,95 |
| Representation Item level | 4240 | 4240 | 8 | 0 | 8 | 1312 | 4770 | 5106 | 4,29 |
| explicit mapping | 4240 | 4248 | 8 | 4240 | 8 | 1312 | 4778 | 5640 | 5,02 |

**Fig. 10** Comparison of the numbers of geometric instances and corresponding file sizes for the discussed modeling strategies

the eight segment types, which is why there exist eight additional instances of IfcShapeModel. Then it utilizes an IfcMappedItem for each segment, which for the whole tunnel results in 4240 occurrences. Furthermore, the number of placements and corresponding points increases, which is caused by the individual transformation of mapped items. Concluding the instance comparison, all numbers are reasonable with respect to the selected strategy.

## Conclusion

Driven by the burden of large file sizes when dealing with the IFC file format, the paper investigates methods to reduce the size of IFC models. In literature, the duplication of elements is pointed out as the main reason. Especially, geometric duplicates can lead to unmanageable file sizes. In many cases, existing optimizers (Solibri IFC Optimizer and IFCCompressor) already can significantly reduce file size. However, when considering geometries, which have been placed directly into the world coordinates system, the optimizers do not eliminate such redundancy. Such models require the solution to the absolute orientation problem to identify duplicates.

The proposed methodology is twofold. On the one hand, it introduces an algorithm to detect world coordinates based geometries. The first phase of algorithm performs grouping based on semantical knowledge to ensure that only elements are compared, which are likely to include same content. Then, the second phase analyses geometry by comparing topological structures and, subsequently, determines a potential rigid body transformation. Finally, when duplicates have been detected, the algorithm assign a sharing technique. On the other hand, sharing techniques are discussed. These techniques are covered separately, because they do not only present modeling strategies when optimizing IFC files. In fact, they also are relevant when creating such models in the design stage aiming to prevent duplication beforehand.

For proof of concept, a case study on a tunnel lining model has been conducted. In both cases, (1) applying the algorithm on existing IFC files as well as (2) modeling in the design stage and exporting to IFC files, enormous reduction rates are observed. It has also been verified that such a reduction process does not influence visual output and semantical completeness. According to the specific sharing concept, the case study only

shows negligible differences. Using an implicit strategy, sharing on various levels can be realized, but also requires the operating software to preserve integrity by either tracking instances or using the copy-on-write pattern. Explicit sharing, instead, provides integrity by schema and therefore, does not require such mechanism. However, sharing is limited to the representation item level.

Considering future BIM projects, the impact of such techniques should not be underestimated. On the one hand, projects are becoming steadily larger and, more importantly, will increasingly be applied to non-traditional sectors. This can be, for example, infrastructure projects like the construction of tunnels or bridges. Such projects introduce a very large project dimension per se, and highly likely include repetitive elements. On the other hand, when considering the software supply, requirements such as collaboration and mobile availability are increasingly influencing the development of web-based solutions. Therefore, resource consumption must be considered to utilize software over limited bandwidth channels and with limited resources on remote devices.

While this paper contributes to IFC data exchange by (1) providing improvement to the elimination of duplicated instances and by (2) investigating modeling techniques for sharing such duplicates, further topics like Model View Definitions (MVD) and Level of Geometry (LoG) are relevant as well. Often tasks in a BIM project require only a subset of information. By applying a specific MVD the IFC model can be filtered, leading to a reduced file size. Also, many models are over-detailed to be used in a specific application. By choosing an adequate Level of Geometry the file size can also be reduced. Finally, not only is an efficient exchange of data important, but also the efficient real-time exploration and interaction is of great significance. Therefore, an efficient visualization pipeline often plays a decisive role. Concepts like flyweight modeling and the usage of different LoGs can be adapted, for example, to improve the visualization in web-based environments.

#### Acknowledgements
Financial support is provided by the German Research Foundation (DFG) in the framework of the project D1 of the Collaborative Research Centre SFB 837. These supports are gratefully acknowledged. Furthermore, the authors would like to express their gratitude to the city of Düsseldorf for providing fundamental data in the frame of SFB 837 to complete this research.

#### Availability of data and materials
Not applicable.

#### Funding
Not applicable.

#### Authors' contributions
All authors contributed extensively to the work presented in this paper. AV reviewed and analyzed the literature, developed the methodology and applied the case study, and drafted the manuscript. CK and MK supervised the entire processes of this study. All authors read and approved the final manuscript.

#### Competing interests
The authors declare that they have no competing interests..

## Publisher's Note


#### Author details
[1]Ruhr-Universität Bochum, Bochum, Germany. [2]Bauhaus-Universität Weimar, Weimar, Germany.

Received: 3 August 2017 Accepted: 5 January 2018
Published online: 09 February 2018

#### References
Arun, KS, Huang, TS, Blostein, SD. (1987). Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-9*(5), 698–700. https://doi.org/10.1109/TPAMI.1987.4767965.

Backas, Susan (2001): SPADAX Final Report. Available online at http://cic.vtt.fi/vera/documents/SPADEX_final_report.pdf, checked on 11/17/2015.

Bazjanac, Vladimir (Ed.) (2002): Early Lessons from Deployment of IFC Compatible Software. 4th European Conference on Product and Process Modelling (ECPPM 2002). Portoroz, Slovenia, 2002.

buildingSmart (2015): IFC Overview Summary. Available online at http://www.buildingsmart-tech.org/specifications/ifc-overview, checked on 12/22/2015.

Davidson, Scott (2017): Grasshopper. Available online at http://www.grasshopper3d.com/, checked on 3/22/2017.

Eggert, DW, Lorusso, A, Fisher, RB. (1997). Estimating 3-D rigid body transformations: A comparison of four major algorithms. *Machine Vision and Applications*, *9*(5–6), 272–290.

Fischer, M, & Calvin, K (2002). *PM4D final report*. CIFE at Standford University. Available online at http://web.stanford.edu/group/4D/download/PM4D_Final_Report.pdf.

Gamma, Erich; Helm, Richard; Johnson, Ralf; Vlissides, John (1995): Design Patterns. Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley.

Horn, BKP. (1987). Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, *4*(4), 629–642. https://doi.org/10.1364/JOSAA.4.000629.

Horn, BKP, Hilden, HM, Negahdaripour, S. (1988). Closed-form solution of absolute orientation using orthonormal matrices. *J. Opt. Soc. Am. A*, *5*(7), 1127–1135. https://doi.org/10.1364/JOSAA.5.001127.

Hurley, John R.; Cattell, Raymond B. (1962): The Procrustes Program: Producing Direct Rotation to Test a Hypothesized Factor Structure. In *Behavioral science* 7 (2), p. 258.

Jacob, Benoît; Guennebaud, Gaël (2017): Eigen. Version 3.1: Tuxfamily. Available online at http://eigen.tuxfamily.org, checked on 3/20/2017.

Jones, AD. (1980): Manual of photogrammetry. With assistance of C.C. Slama, C. Theurer and S.W. Hendrikson. Fourth edition. Falls church, Va (12). Available online at https://doi.org/10.1080/00690805.1982.10438226.

Kabsch, W. (1976). A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, *32*(5), 922–923.

Kabsch, W. (1978). A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, *34*(5), 827–828.

Liu, X, Xie, N, Tang, K, Jia, J. (2016). Lightweighting for Web3D visualization of large-scale BIM scenes in real-time. *Graphical Models*, *88*, 40–56.

McNeel, Robert (2017): Rhinoceros 5. Version 5: Robert McNeel & Associates. Available online at https://www.rhino3d.com, checked on 3/22/2017.

Pazlar, Tomaz; Turk, Ziga (Eds.) (2006): Analysis of the geometric data exchange using the IFC. 6th European Conference on Product and Process Modelling (ECPPM 2006). Valencia, Spain.

Pazlar, Tomaz; Turk, Ziga (Eds.) (2007): Evaluation of IFC Optimization. Proceedings of CIB W78 conference on Bringing ITC knowledge to Work.

RDF Ltd. (2017): IFC engine DLL. Available online at http://www.ifcbrowser.com, checked on 3/22/2017.

Schönemann, PH. (1966). A generalized solution of the orthogonal Procrustes problem. *Psychometrika*, *31*(1), 1–10.

Solibri Inc. (2015): Solibri IFC optimizer. Available online at http://www.solibri.com/products/solibri-ifc-optimizer/, checked on 1/7/2016.

Sun, J, Liu, Y-S, Gao, G, Han, X-G. (2015). IFCCompressor: A content-based compression algorithm for optimizing industry foundation classes files. *Automation in Construction*, *50*, 1–15.
Wahba, G. (1965): A least squares estimate of spacecraft attitude. In SIAM Review 7 (3), p. 409.
Walker, MW, Shao, L, Volz, RA. (1991). Estimating 3-D location parameters using dual number quaternions. *CVGIP: image understanding*, *54*(3), 358–367.