Hindawi
Advances in Civil Engineering
Volume 2020, Article ID 8865107, 22 pages
https://doi.org/10.1155/2020/8865107

Research Article

A Novel HDF-Based Data Compression and Integration Approach to Support BIM-GIS Practical Applications

Zeyu Pan, Jianyong Shi, and Liu Jiang

Department of Civil Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Correspondence should be addressed to Jianyong Shi; shijy@sjtu.edu.cn

Received 5 August 2020; Revised 23 November 2020; Accepted 4 December 2020; Published 22 December 2020

Academic Editor: Qian Wang

Constructing a fully mapped virtual world is the premise of establishing the digital twin system; thus, the integration of BIM (Building Information Modeling) and GIS (Geographic Information System) constitutes a fundamental part of the mature digital twin. Although quite a few theoretical unified data models of BIM and GIS integration have been presented in precedent research, the practical web-based applications and software support for real-world city information modeling are still difficult to achieve. One of the challenges is storage inefficiency. It thence leads to burdens in exchange and analysis, which hugely impede the realization of virtual-real fusion. To address that issue, in this research, we contribute to exploring an HDF- (Hierarchical Data Format-) based, innovative, and synthesized scheme with three significant technical processes. First, data reorganization trims the original data with efficient redundancy elimination and reordering for IFC-SPF (IFC STEP Physical File) and XML-encoded CityGML (City Geography Markup Language). Next, bidirectional transformation methods for BIM and GIS modeling data, images, and analytical data into HDF are proposed. They retain the entities and relationships in modeling data and can further improve the compactness and accessibility by ultimately 60%–80% which saves 497,612 KB from 565,152 KB in the test of ZhongChun transformation substation. Finally, data aggregation enhances the bond of the integrated model and heterogeneous data resources from the transformed HDF files. The case studies show that the approach proposed in this paper could reach high efficiency for practicability of the BIM + GIS integration model. This light-weight integration method can further improve the front-end service responsiveness in digital twin applications.

1. Introduction

There is a trend that the researchers and the governments are putting increasing emphasis on the multisource associated event-driven city planning and management, from regular facility maintenance to evacuation and rescue operations for city-level disasters such as flood, wildfire, and earthquake. In our perspective, the premise of making such an event-based decision is to create the digital twin, literally a thorough, dynamic digital reflection of the real urban environment which should be capable of (1) containing as many city features as possible, (2) synchronizing with its host dynamically through multiple urban data resources, and (3) forming a real-time and integrated response to the events. Functionally, two renowned kernel technologies are integrated as the core foundation of the digital twin to support the city features, known as BIM (Building Information Modeling) at the microlevel and GIS (Geographic Information System) at the macrolevel [1]. Previous attempts for BIM and GIS integration often focus on the semantic mapping of both information models in static cases [2–5], which is inadequate for practical digital twin applications. Generally speaking, for the web services which generate gigabytes of data per second like urban online services, the data to be processed at the edge can retain shorter response time, more efficient processing, and smaller network pressure [6]. It spawns a demand to explore an efficient, cooperative, and heterogeneity-friendly approach for data exchange, query, and integration in practical situations to relieve computing and storage burden. Thence, we

conduct this research to deal with the merging of heterogeneous data formats, compact data storage and exchange, and efficient response to urban events.

BIM excels at dealing with data storage and exchange for 3D models, especially the specific interior components of structures, including buildings, tunnels, and bridges. Meanwhile, GIS masters at managing the potential topological relationships between the entities and events in exterior geographic spaces, while the interior information is usually quite limited. The concept of merging BIM and GIS is naturally reasonable. Borrmann et al. state that integrating BIM and GIS tools is essential for planning extensive infrastructure facilities [2]. Ninić et al. [7] also pointed out that it is indispensable to integrate BIM and GIS when solving multiple scalar issues of planning, designing, and analyzing infrastructure projects [7]. Based on comprehensive BIM-GIS integration, in the future, the urban stakeholders, including governments or single residents, will be able to create a sustainable living environment [1] through more intelligent and approachable management patterns for energy management or lifecycle management. To realize this schema-level forge, plenty of researchers [3–5, 8–13] have been endeavoring for years on analyzing and refactoring those two notable vendor-neutral data standards, IFC and CityGML. IFC (Industry Foundation Classes) is designed to describe the indoor building environment, and complementarily, CityGML (City Geography Markup Language) conveys the outdoor urban environment as more powerful. They have great potential in storage, manipulation, organization, and exchange of urban data. For the sake of serving as stable, open, and consensus standard data models, both of them are vendor-neutral.

However, there are two main challenges during the attempt at integrating those two technologies. The first one is about conceptual data models. The incompatibilities in IFC and CityGML schema, especially the geometric representation [8] and the entity ambiguity, are so aggravating for BIM-GIS integrated applications. The other one is about the scale of integrated data. When the project scale grows, the data complexity, heterogeneity, and volume will explosively increase. Even for a simple city application, gigabyte-level data storage, transmission, and analysis are quite ubiquitous.

At present, pertinent researchers have spent much attention enhancing or extending the concepts of data models, which is undoubtedly essential. However, in comparison, the necessity of developing an efficient storage and exchange format for integrated information has been long overlooked. To make up for the conceptual incompatibilities in both data structures, the researchers have generally followed two respective measures: embedding BIM to GIS or merging them into a uniform, customized data structure. The first attempts include geometric representation transforming [8, 9, 13], georeferencing [10], property mapping [11], multiscale geometric semantic modeling [2], and geometric translation through an indirect path [12]. At the same time, the second attempt involves developing an integrated data model for underground utility management [3] and presenting an ontology-based general data model for integration [4, 5].

When transforming the geometries and semantics from IFC into the target CityGML, no matter which schemes to use, it is inevitable to encounter the inherent defects of CityGML listed by Yang and Eastman [14]. The defects inheriting from XML (Extensible Markup Language), GML (Geographic Markup Language), and from CityGML itself include high parsing cost, too many possible variations of geometric representation, verbiage, and hence massive storage burden. Deng et al. [15] succeeded in mapping partial geometrical representations, Brep (Boundary Representation), Sweptsolid, and CSG (Constructive Solid Geometry) of IFC into Brep of CityGML for LoD1~LoD4 (LoD: Level of Detail). However, the storage size of LoD4 in CityGML will grow to 250%~400% compared with the original IFC files [15]. Additionally, even a low level-of-detail (LoD2) of a single city might be easily over 2 GB (without compression). These phenomena will cause disastrous effects on those high storage-dependent applications in BIM-GIS integration, such as urban-level visualization and big data analysis. As a result, they will reduce the reusability and the capacity of processing the data.

This paper aims to find a practical solution to relieve the storage burden, which can improve data mining capacity in urban data and solve the insufficiency of existent data formats of BIM and GIS for digital twin development. We conduct this research to settle this issue from pretransform, transform, and posttransform processes. More specifically, it is divided into first the refinement of source data such as IFC-SPF, XML-encoded CityGML, pictures, and so on (without format change), then the transformation into a compact, unified data format (Hierarchical Data Format, HDF [16]) (with format change), and finally the enhancement of the bond between each data resource. Thus, three crucial procedures are proposed: (1) data reorganization of IFC-SPF and XML-encoded CityGML files; (2) data transformation for IFC-SPF, CityJSON, photos, time history functions, and so on; and (3) data aggregation of the transformed heterogeneous data.

To facilitate understanding, the rest of this paper is organized as follows. In Section 2, the fundamental features and defects of current BIM-GIS integrated data management will be analyzed. It will then introduce the candidate data format HDF and two experimental, inspiring light-weight data formats (IFC-HDF and CityJSON). In Section 3, the following stuff elaborates on this research's principal methodologies, including data refinement, data transformation, and data aggregation. Next, in Section 4, we will present result evaluation and case validation in integrating BIM and GIS through HDF. In Section 5, the main contributions in current research are summarized. Finally, in Section 6, a few directions in the further study are presented.

2. Background

2.1. Basic Features of BIM-GIS Projects. It is genuinely common to have multiple sources of data files in one BIM-GIS application, such as emergency response [17], leading to two unignorable challenges: storage burden and data

heterogeneity issues. On the one hand, in a typical AEC (Architecture, Engineering, and Construction) project, the relevant information resources might involve not only building information models but also 2D drawings, documentation, photographs, point cloud data, FEM (Finite Element Method) data, etc. On the other hand, there are also multiple geographic data resources, such as maps, texture, oblique photogrammetry data, and so forth. Although BIM and GIS technologies have been endeavoring to merge their relevant data into their respective open consensus data models, it is impossible to cover and transform all other data models with massive preliminary preparation.

2.2. Defects of Current Official Data Formats of IFC and CityGML in Data Sharing. Data serialization is a term in computer science that represents the process of converting structured data to a format that allows the sharing of the data in a form that can recover its original structure [18]. The most prevalent IFC format is IFC-SPF (STEP Physical File, STEP is STandard for the Exchange of Product Model Data, with the extension of ".ifc"). Concurrently, there are also several variations such as IFC-XML (Extensible Markup Language) for cross-domain information exchange, IFC-JSON (JavaScript Object Notation) [19] for web-based applications, and IFC-OWL (Web Ontology Language) [20] for linked data services. In contrast, CityGML owns one specific official data format referred to as XML-encoded CityGML. Besides, there are some experimental alternatives for both data standards, namely, IFC-HDF and CityJSON. A detailed discussion of both formats will be presented later in this article.

Here are some issues in BIM-GIS transformation. The data formats mentioned above might have one or more following disadvantages.

(1) Text-based: it is a tricky feature of a data format, as it can be profitable for increasing human readability but needs massive labor work when being parsed. The data formats in BIM are text-based. Their entity instances tend to be laid out in files since their implicit relationships are established by references such as #45 = IFCAXIS2PLACEMENT3D (#42, #43, and #44) in IFC-SPF. It is difficult for highly efficient random information access. Therefore, it induces poor performance in cases with considerable building information. The parsing of CityGML, as mentioned before, has great difficulty in dealing with malformed XML. In contrast, in the binary format, HDF organizes the collected data with a predefined hierarchical structure. It allows the users to access the information randomly and isochronally by tracking the path reference link. Although HDF format is binary serialized, it is also self-described and highly readable after decoding. It enables the users without prior knowledge of HDF format to smoothly explore the file and retrieve data required because all information, such as data hierarchy and data types, is encoded in itself [21].

(2) Verbose (IFC-XML, IFC-JSON, IFC-OWL, and XML-encoded CityGML): those IFC formats explicitly translate the metadata hidden in IFC-SPF from relative schema specifications and repeat these attribute names in each instance declaration, as shown in Figure 1. It improves readability to some extent but will generally burden 13%, 48%, and even 1272% on storage for each format than the original IFC-SPF [22]. The same dilemma occurs in CityGML since the root format is XML, where related developers have long criticized the verbosity. On the other hand, a feature in HDF, the so-called compound data type, fixes both readability and verbosity issues. To illustrate, Figure 2 suggests that the attribute names will be declared only once as data types for each type instantiation of IFC, while the whole file remains easy to understand.

(3) Redundant: during the whole lifetime of each BIM or GIS project, no matter which schema version (IFC Schema Specification-buildingSMART Technical [23]; CityGML | OGC [24]) the model file complies with, data redundancy involving instance repetition and geometry complexity [25] is unintentionally generated, as depicted in Figure 3. Sometimes, it may even accumulate up to 60–70% in relatively large models [26]. To achieve further performance on BIM-GIS integrated applications, data redundancy eliminating pretreatments based on the original SPF file are inevitable.

In this paper, the pretreatment of IFC-SPF, which includes duplicate removal and instance reordering, will be introduced. It is noteworthy that this pretreatment process is also feasible during the translation to other data formats of IFC.

(4) Storage challenge: all the prevalent IFC formats have this issue, even for IFC-SPF. Meanwhile, using XML-encoded CityGML as a 3D urban data storage usually tends to cost massive file size [14], especially in high level-of-detail cases. In Krijnen's fundamental work on IFC-HDF, he has mentioned some vital conclusions that IFC-HDF is far more compact than the other formats and could have an alluring performance that saves around 50–70% size in comparison with IFC-SPF on large building models [27]. Unlike many different data formats, HDF allows the exploiters to randomly access information by indexing with decompressing the chunked datasets of interest rather than the whole file. The compact storage and random access capability of data archives will profoundly enhance data transmission and demonstration performance.

(5) Incomplete: IFC schema is designed extensible, which allows the professional developers to attach their findings or integrate information from other domains onto IFC schema as extensions, for instance, structural health monitoring [28], point cloud [29], design change management [30], and so

IfcJSON fragment

```
{
    "axis": {
        "instanceId": 43,
        "directionRatios": [0, 0, 1],
        "type": "IfcDirection"
    },
    "instanceId": 45,
    "location": {
        "instanceId": 42,
        "coordinates": [0, 0, 0],
        "type": "IfcCartesianPoint"
    },
    "refDirection": {
        "instanceId": 44,
        "directionRatios": [1, 0, 0],
        "type": "IfcDirection"
    },
    "type": "IfcAxis2Placement3D"
}
```

IFC-SPF fragment

```
#42=IFCCARTESIANPOINT((0.,0.,0.));
#43=IFCDIRECTION((0.,0.,1.));
#44=IFCDIRECTION((1.,0.,0.));
#45=IFCAXIS2PLACEMENT3D(#42,#43,#44);
```

Figure 1: Expression comparison between IfcJSON and IFC-SPF format.

	id	type	GlobalId	OwnerHistory	Name	Description	ObjectType	ObjectPlacemer
0	800342	IfcBuildingElementPart	1UVe6Ptx2FL1b1jcB2oTKc	EI_id_#22_If...	Gevel betimmering, hor...	None	None	EI_id_#799975_IfcL
1	798896	IfcBuildingElementPart	1spECmAd1bUcY8PhxZPL$o	EI_id_#22_If...	Tengels	None	None	EI_id_#797823_IfcL
2	851951	IfcBuildingElementPart	1A7qmCfNeypHA_a5Motcmj	EI_id_#22_If...	400 Isolatie wand	None	None	EI_id_#851886_IfcL
3	849108	IfcBuildingElementPart	3pdah9ZwyL6w_tkLtyfvj9	EI_id_#22_If...	400 Isolatie wand	None	None	EI_id_#849042_IfcL
4	503	IfcBuildingElementPart	0CMdCMGf21$eLdXN0Q7yX6	EI_id_#22_If...	Steen - Baksteen	None	None	EI_id_#432_IfcLoca
5	409	IfcBuildingElementPart	0FMv1xIafTJKC69h0p1Vxj	EI_id_#22_If...	410 Luchtspouw buiten	None	None	EI_id_#338_IfcLoca

Figure 2: An exemplary dataset for type IfcBuildingElementPart of IFC in HDFViewer 3.1.0 as a demonstration of data simplicity in HDF.

IFC-SPF segment

```
#31 = IFCAXIS2PLACEMENT3D(#32, #33, #34);
#32 = IFCCARTESIANPOINT((0., 0., 0.));
#33 = IFCDIRECTION((0., 0., 1.));
#34 = IFCDIRECTION((1., 0., 0.));
#35 = IFCBUILDINGSTOREY('36_p3GiZXEHxQ3gWc
#36 = IFCLOCALPLACEMENT(#30, #37);
#37 = IFCAXIS2PLACEMENT3D(#38, #39, #40);
#38 = IFCCARTESIANPOINT((0., 0., 0.));
#39 = IFCDIRECTION((0., 0., 1.));
#40 = IFCDIRECTION((1., 0., 0.));
```

Figure 3: Simple data redundancy in IFC-SPF.

on. Likewise, CityGML also allows these kinds of Application Domain Extension (ADE) to increase the schema's extensibility. However, such extension development work might cost lots of human labor, and it usually relies on a particular version of the schema. Besides, to some extent, those schema extensions enrich the valuable information that the original schema can participate in specific applications, but, due to its domain-related features, the data it can store and present are still quite limited. Contrarily, HDF, the domain-neutral binary data format, is compatible with various kinds of data formats. Hence, it is essential for the efficient organization of heterogenetic data in complex projects.

Owing to its capacity in efficiently treating those heterogeneous data, the specific data format, Hierarchical Data Format (HDF), is chosen as the candidate format for relieving the storage burden and dealing with data heterogeneity issues in this research. The following paragraphs will make a brief introduction to HDF.

2.3. HDF's Basic Concepts

(1) Definition and function: vividly speaking, HDF is analogous to a powerful, portable synthesis of databases and organized file systems. As defined by its official organization, it is designed as a scientific binary data format capable of supporting an unlimited variety of data types and is portable and extensible for high volume and complex data [16, 31]. It is widely supported for creation, edition, and parsing by popular programming languages such as C, Fortran, C++, Java, Python, Matlab, IDL, etc. Besides, it also excels at hierarchical compartmentalization and self-descriptive compound data type [32].

(2) Characteristics: one of HDF's most attractive features for complicated projects is that it can be treated as a versatile file system and can hold various heterogeneous datasets, including images, tables, graphs, and even documents, including PDF or Excel besides the primary datasets.

(3) Specialty: HDF adopts a specific chunked compression storage method, through which the whole binary file can be decomposed intuitively into a customized or predefined number of data chunks with an inherent identification index. Due to the chunked compression feature, HDF is adept at storing considerable amounts of data with an astonishing rapid random information retrieval, which is truly precious for city-level data management.

In comparison to relational databases, the option of using HDF as a target repository to help incorporate the storage and manipulation of metadata and data in specific

applications can remarkably improve the portability and flexibility [33]. HDF meets the requirements of both light-weight data transmission and real-time interoperation with low delay in BIM-GIS integration applications.

2.4. Two Enlightening Data Formats. Referring to the latest studies [14, 27, 29, 32, 34], to develop a proper exchange data format for light-weight applications in BIM and GIS, respectively, two promising alternatives have been proposed, respectively, namely, CityJSON and IFC-HDF. Both have splendid features innately when serving as a potential compact variant. Hence, it is quite natural to come up with a scheme that assimilates them as two base data formats and make proper adjustments to develop an appropriate container for BIM-GIS integration. Before that, some essential characteristics of CityJSON and IFC-HDF will be discussed first.

2.4.1. CityJSON. CityJSON is developed to offer a compact and developer-friendly alternative data format, namely, JSON (JavaScript Object Notation), for storing 3D city models rather than the original inefficient CityGML, according to its website [35]. Although it remains experimental and underexamined by practical experiences, the characteristic it possesses is quite impressive [14]. In comparison with XML-encoded CityGML, CityJSON is much more compact (about one-sixth of equivalent XML-encoded CityGML), which is essential for serving as an acceptable exchange format. It is easier to deal with by developers since JSON is a long liable data format and more human-readable.

When XML-encoded CityGML is being parsed into CityJSON, some significant changes have been made. First, it removes the direct hierarchical relations (for instance, the component trunks contained in a particular city object) of CityGML. It rebuilds them into an implicit form: parent/children relation to realizing the aim of a relatively flattened data structure. This treatment is controversial because it will undoubtedly improve the random accessibility of minor components. Still, in another view, it might decrease the human readability and proficiency when searching for the composition relationships. Secondly, to enlarge the flexibility of the primary feature of numerous vertices' point storage due to Brep shape representation, the developers of CityJSON figured out a solution where the coordinates of vertices are segregated from the former slots in those primitive GML leaves and then rearranged with duplicate removal. The superior entities must reference the point through the index. Furthermore, in CityJSON, most object IDs embody 3D primitive geometric form element IDs except for city objects and semantic surfaces [14].

As a GIS-specific exchange data format, CityJSON will no doubt be powerful, but there are still some adjustments required to make for this research's purpose. LoD4 (the interior environment) of city objects in CityGML, including building, bridge, and tunnel modules, was not designed to be involved in CityJSON. Even if it were supported, it would affect data management (due to the index mapping) and data storage [22] itself to some extent. This omission might decrease the analysis dimension of this format for specific applications that need indoor data. For completeness, the interior data are compensated with IFC by converting both indoor and outdoor information into a particular compact exchange data format, namely, HDF, here, with a method to seamlessly transform the geometric and semantic information for both data structures. Besides, in the XML or JSON data format, the texture materials tend to be attached beyond the main file. It might cause some inefficiency when carrying on data transmission. The pretreatment of appearances is considered and will then be converted into the target HDF to improve the data completeness further.

2.4.2. IFC-HDF. The scientific binary data format, HDF, was first introduced to IFC in 2015 by Krijnen et al. [34]. This series of work about IFC-HDF [27, 29, 32, 34] revealed the possibility to develop a new variation of IFC for exchange, query, and storage with HDF's remarkable storage capacity, high compression, and random information retrieval ability of large volume data cases and its aptitude to be compatible with a wide range of heterogeneous data types.

Before being adopted in the AEC industry, HDF as an efficient exchange data format has already had great fame in different applications of all kinds of fields for many years, especially in those sensitive with data storage and query capacity such as web-based visualization and high-performance, parallel data analysis [36]. It is common sense that with the development of data acquisition methods, the data scale and data complexity for each field, involving BIM and GIS, have become an inspiring treasure and a severe issue simultaneously. As data scale and heterogeneity grow, plain text-based data formats will cause problems when processing substantial N-dimensional datasets. Then, the systems' scalability and flexibility will be challenged [37]. HDF, as a self-described, neutral data format, has some essential characteristics for web-based services, including efficient parallel I/O (Input/Output) for fast read/write, compact data storage, and storage and management of metadata and data provenance information to facilitate complex analysis workflows [36]. Hence, researchers or professionals tend to select HDF as the fundamental data archive format for high-performance real-time analysis and visualization systems owing to its splendid storage and access efficiency. For example, HDF has been exploited by nuclear physics [38], analytical chemistry [21], spaceborne altimetry [39], social science [40], meteorology [36], and so forth.

Krijnen 2019, stated that there are more inspiring facts about the effects of using HDF as the data format of the 3D building information [32]. Two critical conclusions can be derived: (1) for data storage, a well-padded HDF with its inherent compression will reach one-fifth (maximum 80% compression ratio in particular cases) of the original IFC-SPF and nearly the same or even better than IFC-SPF (gzip); (2) for processing and access time, IFC-HDF will be far more efficient than IFC-SPF in all cases and the query time surprisingly remains almost same for different scales of model thanks to its chunked compression algorithm. IFC-

ZIP also has great compression capacity, but when used for information extraction, the whole clumsy file needs to be decompressed first, while HDF need not be due to its chunked compression mechanism.

Analogous to CityJSON, IFC-SPF has been designed to present the content with flattening data mode. At the same time, unlike CityJSON, IFC-SPF spreads out almost all elements as entity instances in its model file (ISO 10303-21 : 2002 [41]).

For the sake of rebuilding the hierarchical system of the model through the implicit references, the whole text-based file must be parsed first, and the relationships must be retrieved. For large model files, this process based on IFC-SPF might be challenging. HDF provides a flexible alternative to search for interest without parsing the whole hierarchical system due to random access capability.

3. Methodology

The succeeding subsections are divided into three functional parts: data reorganization, data transformation, and data aggregation. We proposed a methodology that links up the above three procedures to support the storage, exchange, and retrieval of large-scale project data in practical digital twin applications. The processes of each part are summarized in Figure 4. They are set to be extensible and modular, which means components in this approach are flexible to embrace data processing efficiency.

Here is a brief explanation of the technology roadmap depicted in Figure 4. The first phase is the data reorganization. It processes the raw data from source files with respective pretreatment methods for each data resource, including IFC-SPF, CityJSON, images, and analytical data. We contribute to detecting and eliminating the duplicate redundancies and reference redundancies in IFC-SPF with high performance during the data reorganization. The next phase is the data transformation. It takes charge of transforming the pretreated data into HDF. In this phase, we try to maintain each source's data structure to realize the bidirectional transformation, which means it is feasible to regenerate the source file back from the transformed HDF. Besides, our contribution is to present a recursion solution to change the information embedded in CityGML/CityJSON to the target format HDF, and the reversal conversion is also realized. The final phase is data aggregation. It is responsible for establishing the semantic links between the objects in the transformed heterogeneous resources under the supervision of project ontologies derived from practical situations and experiences to enhance the bond between the transformed data.

Utilizing the comprehensive approach with data reorganization, data transformation, and data aggregation based on HDF proposed in this research, the BIM-GIS related data can be integrated at the storage and semantic level in a more practical way. Furthermore, there is another point to be mentioned. In each phase, the procedures' outcomes can be exported separately as the resource for other situations, which gives this scheme more flexibility to adapt to new applications.

3.1. Data Reorganization. To further promote the efficiency and performance of data exchange, storage, and analysis of BIM-GIS integration, some pretreatments for different primitive data resources are necessary. Compared to the modeling data studied in this paper, the reorganization of images and analytical data is relatively simple; this phase of those data is moved into the data transformation section. In this subsection, we emphasize the methods applied to IFC-SPF and XML-encoded CityGML.

3.1.1. Procedures for IFC-SPF. This procedure for IFC-SPF is composed of two functional parts: eliminating redundant information and reordering instances. Both are based on pure text processing and hence version-neutral with different IFC schema, which guarantees no information loss.

(1) Elimination of Redundant Information. Redundancy always occurs in each IFC file, no matter what the file size is. Additionally, there is a positive correlation between file size and data redundancy. A more intuitive demonstration of this phenomenon will be exhibited later with example files in Section 4.1.

Four procedures for the elimination of redundant information are proposed (Algorithm 1): Text separation*, Repetition detection*, Reference replacement*, and Reassembly*. There are three necessary containers to maintain the process which are CD (Content Dictionary) for storing instance content with its identifiers where the key is instance id (the value is the wrapped content), RR (Reference Relationship Dictionary) for storing reference relationships such as {'#30':['#24','#31']} in #30 = IFCLOCALPLACEMENT (#24,#31) case, RP (Repetition Pair Dictionary) for storing detected duplicates, and their replacements such as {'#40':'#34' or wrapped #34} in #40 = IFCDIRECTION (1.,0.,0.) and #34 = IFCDIRECTION (1.,0.,0.) case.

Main data flow (Figure 5): the BodySection retrieved through TextSplit will be transported to RepetionDetect to traverse each instance. During the traversal, the instance line will be split as ID-Content pair with a content repetition check to make sure whether it is redundant or not. After that, reference replacement will be executed to replace the redundant instances' ID with its initial sibling. Finally, all sections will be reassembled as the target output.

Note: it is generally impossible to get complete trimming for a redundant IFC-SPF with just one epoch, owing to its tree-like reference structure, which will generate recursive redundancy. An illustration is presented in Figure 6 to elaborate on this phenomenon.

Iteration of instance line generation is taken to refrain from insufficient trimming. The iteration would be terminated once the count of repetition pairs reaches zero. After several trials (Figure 7), patterns emerge that the duplicate removal process's epochs will come close to 11 or more, implicitly determined by the deepest branch of IFC structure (might be slightly different depending on the IFC schema version, here checked IFC2x3).

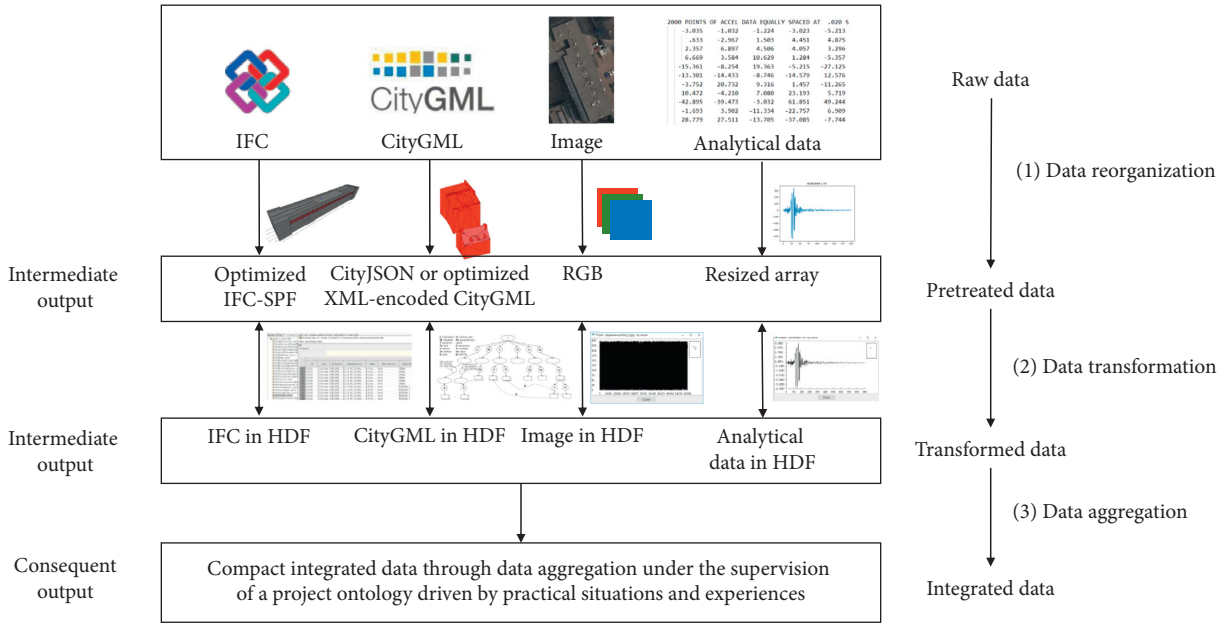


Figure 4: Methodology of this research.

Input: *File*: Original IFC Step Physical File
Output: *File* // Final trimmed file
(1) **function** DUPLICATEREMOVE(*File*)
(2) *Lines* ← *File.readlines*()
(3) // do Step 1
(4) *TextSpilt*(*Lines*) ⟶ *HeadSection*, *BodySection*, *FootSection*
(5) // do Step 2
(6) *RepetitionDetect*(*BodySection*) ⟶ *CD*, *RR*, *RP*
(7) // loop Step 3 until length[RP] = 0, where RP can be renewed by Step 2
(8) *ReferenceReplace*(*CD*, *RR*, *RP*) ⟶ $BodySection_{renew}$
(9) // do Step 4
(10) *Reassembly*(*Headsection*, $BodySection_{renew}$, *FootSection*) ⟶ $File_{new}$
(11) **return** $File_{new}$
(12) **end function**

Algorithm 1: Duplicate removal: remove the repetition occurred in IFC-SPF.

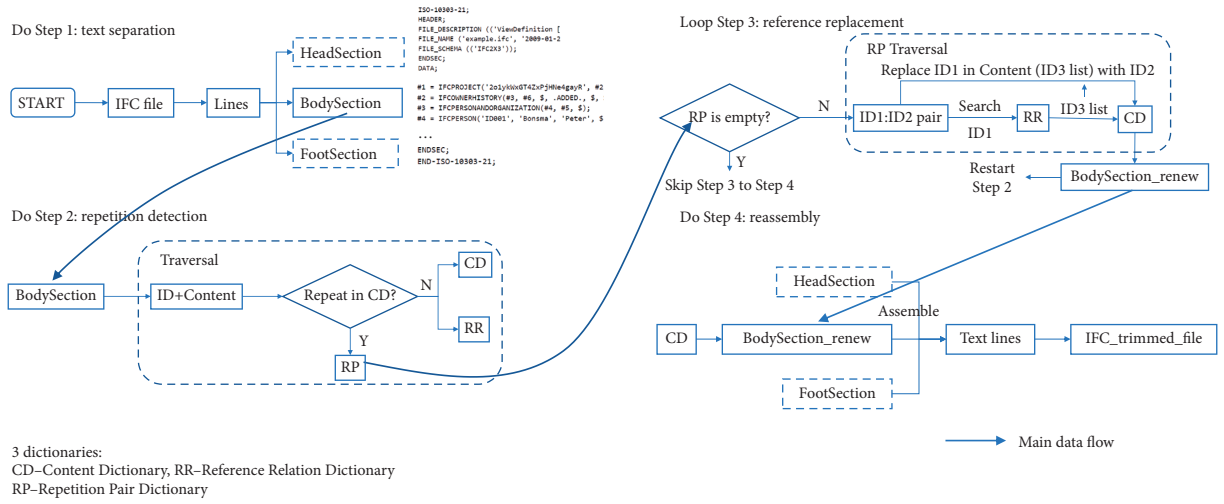


Figure 5: Data flowchart for duplicate removal of IFC-SPF.

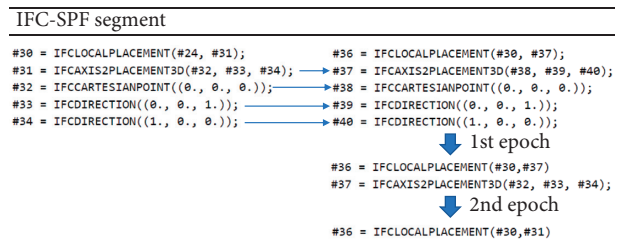


Figure 6: Recursive redundancy in IFC.

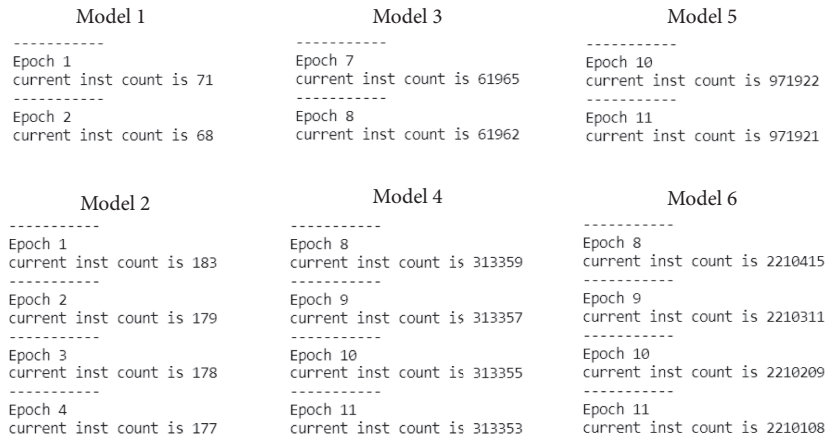


Figure 7: A demonstration of the epoch cost of each test model in this research (detailed information of each model will be discussed in Section 4).

Input: BodySection: List of Instances
Output: $File_{new}$
(1) **function** INSTANCEREORDER(*BodySection*)
(2) // do Step 1
(3) *ReferenceCount*(*BodySection*) $\longrightarrow$ *RC*, *CD*
(4) // do Step 2
(5) *IDSubstitute*(*RC*, *CD*) $\longrightarrow$ $BodySection_{renew}$
(6) // do Step 3
(7) *Reassembly*(*HeadSection*, $BodySection_{renew}$, *FootSection*) $\longrightarrow$ $File_{new}$
(8) return $File_{new}$
(9) **end function**

Algorithm 2: Instance reordering: reorder the instances based on the particular principle that the more frequently an instance is referenced, the smaller the referred identifier is assigned.

(2) Reordering of the Instances. This treatment aims to reset the instance order with one principle: the more frequently an instance is referenced, the smaller the referred identifier is assigned. Assigning frequently referenced objects with short identifiers can help decrease the byte cost of serialization. The compactibility can be further improved since the bytes for storage the references take decrease. Analogous to duplicate removal, the critical point is to find the appropriate substitutes for the target reference in "instance content." However, unlike duplicate removal, reordering does not need iterations. If this process begins right behind duplicate removal, then it can be simplified.

Instance reordering (Algorithm 2): Reference count*, ID substitution*, and Reassembly*. There are two necessary containers to maintain the process, which are RC (Reference Count Dictionary) and CD (Content Dictionary). CD here has almost the same capability as that in the Duplicate Removal section, while RC stores the frequency of each instance being referenced.

Main data flow (Figure 8): the retrieved BodySection will be split into ID-Content pairs. Meanwhile, the reference count will be recorded as the sorting principle. After that, reassign each instance's ID and further substitute the reference IDs in the instance contents. Finally, all sections will be reassembled as the target output.

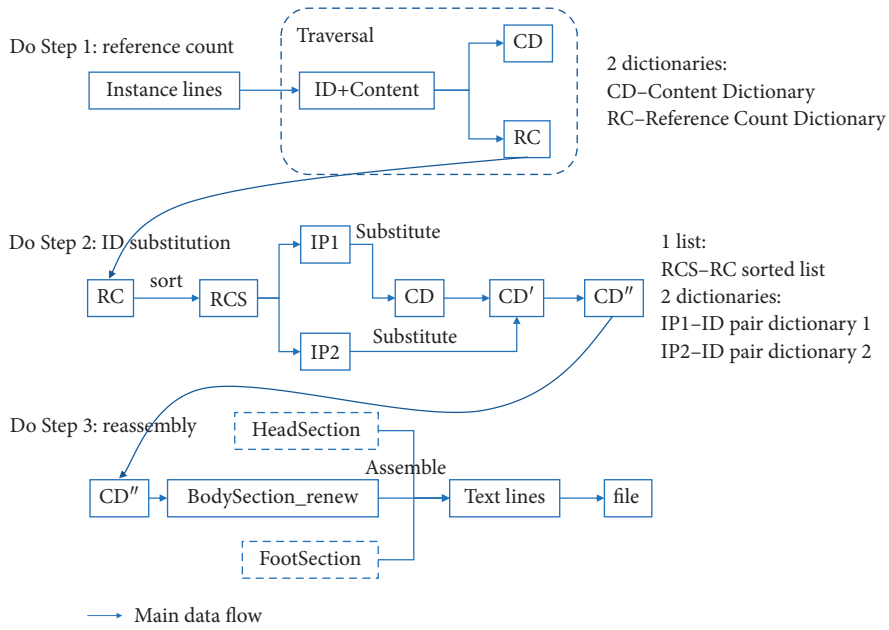


Figure 8: Data flowchart for instance reordering of IFC-SPF.

In Figure 9, part of the reference count statistics of each test model is demonstrated. It will be convenient for us to make the ID substitution process by extracting the original files' reference frequency.

3.1.2. Procedures for XML-Encoded CityGML

(1) Removal of the Vertex Redundancy. In XML-encoded CityGML, to guarantee its explicit hierarchical structure of city object instantiation, the geometric representations such as surfaces and vertices tend to be embedded deep and separately in related XML trees. That feature will generate geometric redundancy. Although the reference solutions based on x-link are proposed to relieve this dilemma, due to GML's versatile representation patterns, this problem can still not be radically solved. Contrarily, after converting the representations into CityJSON, all vertices are extracted, and the surfaces can remain connected to relevant vertices by a particular indexing mechanism inherited from Wavefront OBJ [14, 42]. With this isolated vertex container, the duplicates can be removed, and the reference can be refreshed completely. Besides, by now, CityJSON has supported the bidirectional conversion with XML-encoded CityGML. Hence, it is worthwhile to transform XML-encoded CityGML into CityJSON, which can be processed by an open-source toolkit citygml-tools [43] before merging data into HDF.

(2) Pretreatment for Irregular Datasets. Based on its process difficulty, the original data containers in CityJSON can be divided into three levels: regular, semiregular, and irregular. In contrast, because of its EXPRESS specialty, the IFC data can be all regarded as a regular level.

(a) Regular: in the regular cases, the original data are well formed with the fixed dimensional size of data, including vertices (x, *y*, and z table stored, as shown in Figure 10) and vertices-texture. Those kinds of data can be parsed directly into a highly compressed data form.

(b) Semiregular: for example, in the case of appearance in CityJSON, the textures are listed with verbose but regular map elements, which can be parsed as a compound 2D data table, as depicted in Figure 11.

(c) Irregular: the irregular data are an aggravating issue when making the transformation process. For instance, in CityJSON, inherited from GML, the boundaries can be composed of different kinds of patches, leading to dimension confusion and difficulty to be parsed as a compact data table in HDF. Hence, padding is added to the original container and the voids are filled with "−1" to solve this dilemma. This pretreatment contains three steps: detect irregular arrays, calculate the maximum scale for each dimension, and finally, resize the container with paddings filled with −1.

The result of padded irregular data in HDF is shown as follows (Figure 12). For convenience in presentation, the second and third dimensions are switched, but the data stored are still in dimension size of $10 \times 1 \times 6$.

3.2. Data Transformation. As being narrated above, due to its remarkable adaptability of supporting heterogeneous data sources and other strengths, HDF is chosen as the candidate storage and exchange data format in this research to support the integration of BIM and GIS. The following subsections put forward the bidirectional transformation for four kinds of data in BIM-GIS integration, including IFC, CityJSON, images, and analytical data.

Model 1

```
Reorder start
Five most referenced id and reference count:
ID      Count
#2      10
#22     7
#25     5
#20     4
#56     4
```

Model 3

```
Reorder start
Five most referenced id and reference count:
ID      Count
#22695  1961
#29     874
#52     171
#56     135
#27868  72
```

Model 5

```
Reorder start
Five most referenced id and reference count:
ID      Count
#41     21234
#511476 6966
#134747 4871
#131435 4734
#19     4483
```

Model 2

```
Reorder start
Five most referenced id and reference count:
ID      Count
#2      26
#22     14
#20     9
#87     7
#47     6
```

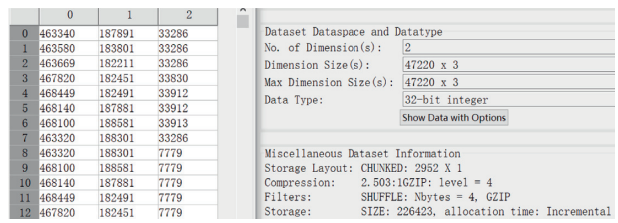
Model 4

```
Reorder start
Five most referenced id and reference count:
ID      Count
#22     12447
#49     1681
#519    1541
#45     1507
#174    1088
```

Model 6

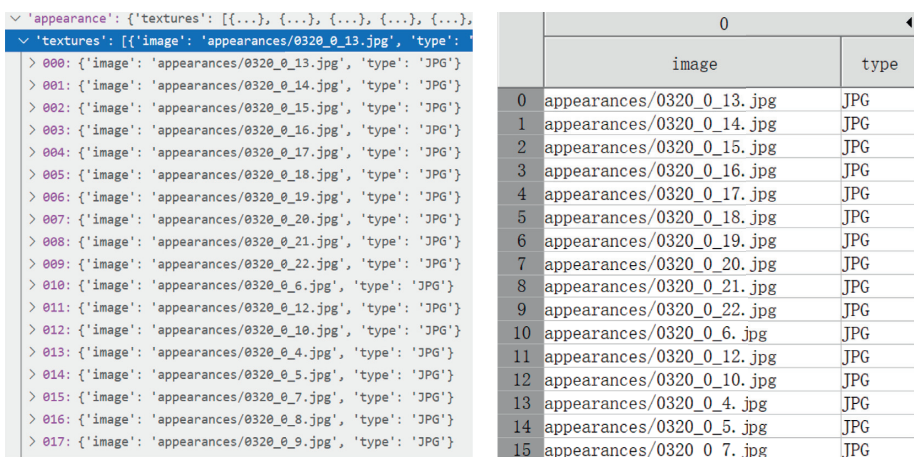
```
Reorder start
Five most referenced id and reference count:
ID      Count
#41     34442
#19     6923
#38     3624
#31     2599
#10144  1909
```

Figure 9: A demonstration of reference count statistics extracted from the original files.



	0	1	2
0	463340	187891	33286
1	463580	183801	33286
2	463669	182211	33286
3	467820	182451	33830
4	468449	182491	33912
5	468140	187881	33912
6	468100	188581	33913
7	463320	188301	33286
8	463320	188301	7779
9	468100	188581	7779
10	468140	187881	7779
11	468449	182491	7779
12	467820	182451	7779

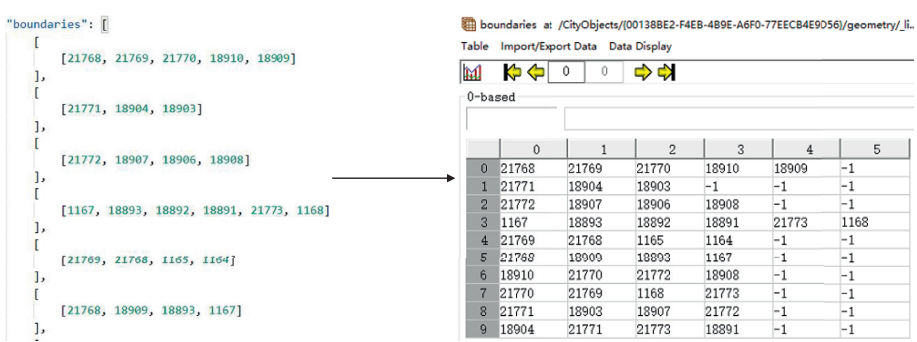
Figure 10: Regular vertex storage in HDF, with 2.503 compression ratio and 47220 vertices.



```
'appearance': {'textures': [{...}, {...}, {...}, {...}, {...},
'textures': [{'image': 'appearances/0320_0_13.jpg', 'type': '
  000: {'image': 'appearances/0320_0_13.jpg', 'type': 'JPG'}
  001: {'image': 'appearances/0320_0_14.jpg', 'type': 'JPG'}
  002: {'image': 'appearances/0320_0_15.jpg', 'type': 'JPG'}
  003: {'image': 'appearances/0320_0_16.jpg', 'type': 'JPG'}
  004: {'image': 'appearances/0320_0_17.jpg', 'type': 'JPG'}
  005: {'image': 'appearances/0320_0_18.jpg', 'type': 'JPG'}
  006: {'image': 'appearances/0320_0_19.jpg', 'type': 'JPG'}
  007: {'image': 'appearances/0320_0_20.jpg', 'type': 'JPG'}
  008: {'image': 'appearances/0320_0_21.jpg', 'type': 'JPG'}
  009: {'image': 'appearances/0320_0_22.jpg', 'type': 'JPG'}
  010: {'image': 'appearances/0320_0_6.jpg', 'type': 'JPG'}
  011: {'image': 'appearances/0320_0_12.jpg', 'type': 'JPG'}
  012: {'image': 'appearances/0320_0_10.jpg', 'type': 'JPG'}
  013: {'image': 'appearances/0320_0_4.jpg', 'type': 'JPG'}
  014: {'image': 'appearances/0320_0_5.jpg', 'type': 'JPG'}
  015: {'image': 'appearances/0320_0_7.jpg', 'type': 'JPG'}
  016: {'image': 'appearances/0320_0_8.jpg', 'type': 'JPG'}
  017: {'image': 'appearances/0320_0_9.jpg', 'type': 'JPG'}
```

	0	
	image	type
0	appearances/0320_0_13.jpg	JPG
1	appearances/0320_0_14.jpg	JPG
2	appearances/0320_0_15.jpg	JPG
3	appearances/0320_0_16.jpg	JPG
4	appearances/0320_0_17.jpg	JPG
5	appearances/0320_0_18.jpg	JPG
6	appearances/0320_0_19.jpg	JPG
7	appearances/0320_0_20.jpg	JPG
8	appearances/0320_0_21.jpg	JPG
9	appearances/0320_0_22.jpg	JPG
10	appearances/0320_0_6.jpg	JPG
11	appearances/0320_0_12.jpg	JPG
12	appearances/0320_0_10.jpg	JPG
13	appearances/0320_0_4.jpg	JPG
14	appearances/0320_0_5.jpg	JPG
15	appearances/0320_0_7.jpg	JPG

Figure 11: Original form in JSON (left) and resultant form in HDF (right), with 33.388 : 1 compression ratio in texture case; the source file is 3-20-DELFSHAVEN from [35].



```
"boundaries": [
  [
    [21768, 21769, 21770, 18910, 18909]
  ],
  [
    [21771, 18904, 18903]
  ],
  [
    [21772, 18907, 18906, 18908]
  ],
  [
    [1167, 18893, 18892, 18891, 21773, 1168]
  ],
  [
    [21769, 21768, 1165, 1164]
  ],
  [
    [21768, 18909, 18893, 1167]
  ],
```


	0	1	2	3	4	5
0	21768	21769	21770	18910	18909	-1
1	21771	18904	18903	-1	-1	-1
2	21772	18907	18906	18908	-1	-1
3	1167	18893	18892	18891	21773	1168
4	21769	21768	1165	1164	-1	-1
5	21768	18909	18893	1167	-1	-1
6	18910	21770	21772	18908	-1	-1
7	21770	21769	1168	21773	-1	-1
8	21771	18903	18907	21772	-1	-1
9	18904	21771	21773	18891	-1	-1

Figure 12: Original form in JSON (left) and resultant form in HDF (right) in boundary case.

Two fundamental data units in HDF are group and dataset. Dataset is analogous to the table in relational databases, and group is similar to the directory in a file system. Besides, both data units can have attributes to store extra information, which is helpful when converting a complex data structure such as CityJSON.

3.2.1. IFC-SPF to HDF. This transformation work references an officially published ISO standard, ISO /TS 10303-26 : 2011 binary representation of EXPRESS-driven data [44]. Since IFC is a variant of EXPRESS, it is feasible to transplant EXPRESS-HDF into IFC-HDF. For interoperability, a promotion to the parsing process has been made so that the reverse transformation, namely, from HDF to IFC-SPF, can also be accomplished. Besides, through this algorithm, it will not be necessary to rely on a particular IFC-relevant toolkit in parsing or regenerating IFC-SPF. The critical point is to extract information such as explicit attributes directly from the original EXPRESS-based schema. It also allows us to parse the IFC-SPFs regardless of the version of the IFC schema. Furthermore, to get prepared for the data aggregation phase, an index map between IFC-SPF-based identifier and HDF-based identifier is deliberately appended to the main contents as can be seen in Figure 13. In Section 3.3, more details about the function of the index map will be explained.

As is represented in Algorithm 3, it is feasible to gain bidirectional transformation between IFC-SPF and HDF through those two functions. The metadata of the original file will be extracted and parsed as the HDF's attributes. It is critical to parse the raw instance text lines into the form of "inst_id = ifctype (paramList)" to wrap the instances with their relevant explicit attributes. The reverse transformation depends on unwrapping and reassembly.

Besides, after several tests, it is proved that the bidirectional transformation here is seamless, which means it will be much compacter and, more importantly, safe to store the source data.

3.2.2. CityJSON to HDF. IFC is a derivation of EXPRESS, making it possible to represent complicated data models with an object-oriented approach concisely [45]. Hence, the model information involved in IFC-SPF, due to its EXPRESS-like configuration, can be easily parsed as field-value datasets and then dumped into HDF. However, on the contrary, CityJSON tends to elaborate its contents with explicit hierarchical data structure with numerous, disperse, small lists, which is challenging to be parsed directly into HDF. So far, we have managed to realize the bidirectional transformation by tracking the whole hierarchical structure in CityJSON through recursion. The inefficient explicit hierarchy causes the major storage challenge here. In future work, we consider developing an object-oriented variant of CityGML/CityJSON to flatten the data structure further. The below graph (Figure 14) is an HDF diagram that represents a typical CityJSON file. It contains the transformed data resources and their relationships. The dashed line refers to reference redirecting in HDF.

Algorithms 4 and 5 show how to constitute the bidirectional conversion between CityJSON and HDF. Recursion is the key to the conversion processes here. The recursion process helps us to maintain the whole original hierarchy from CityJSON without grasping the complicated structure in advance.

In the positive conversion from CityJSON to HDF, a particular function named DictParse is designed to experience recursion. The inputs are (a) "hdfFile," the target HDF for writing, (b) "aDict," the recursive data units, and (c) "prev_path," the path of the parent group stored in the target HDF. It is essential to check the input dictionary's values, whether they are instances of a dictionary, list, or others, to further recurse or parse them into HDF datasets and attributes. Additionally, in the particular case that needs converting a list of objects into HDF data units, some subgroups are inserted in the form of "_list_item_#num" to remind the back-conversion process to translate the items into a list.

On the other hand, in the reversal conversion from HDF to CityJSON, similarly, another function designed to experience recursion is GroupWalk. The inputs contain (a) "group," the reversive data unit stored in HDF, (b) "grandParentNode," the grandparent node related to current data defined in target dictionary, (c) "parentNode," the parent node related to current data defined in target dictionary, and (d) "rootDict," the target output dictionary. This process is to assemble the target output dictionary with the data extracted from source HDF through recursion. Groups in HDF are iterated to find the attributes and datasets to parse them into the target dictionary. When dealing with datasets, if the datasets are detected as padded, it will be cleaned first before further transformation. Besides, a list is allocated here to prepare to hold the objects in the "list_item_#num" group and, finally, attach the list directly to the grandparent node if the list's length is not zero.

3.2.3. Image to HDF. Image is an indispensable part of practical digital twin projects. In this section, we will focus on the reorganization and transformation of images. With the help of OpenCV (an open toolkit for image processing) [46], the large source TIFF and BMP images, or processed PNG and JPG, will be feasible to be compressed with different qualities (0–100) into HDF and be regenerated to meet the need of specific applications. The images should be further encoded through OpenCV as streaming data in storage to increase storage efficiency; meanwhile, they can be decoded as real image files for formal utility. The algorithm of the processes above is summarized in Algorithm 6. Besides, the procedure of image processing might be followed by the flowchart in Figure 15. In the last process of Figure 15, it shows the result of decoding the stored image data in HDF and restitute to regular images.

3.2.4. Analytical Data to HDF. Most analytical data have a fine structure and primary data types, which can be easily parsed into HDF. Here is an example of time history functions. The input time history functions grouped with 1,

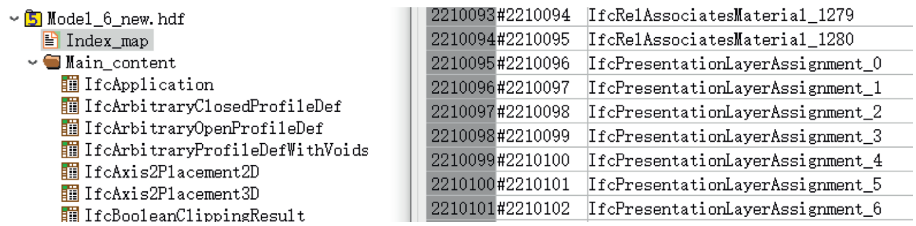


Figure 13: The demonstration of the index map (on the left side is the primary structure of IFC in HDF; on the right side is the index map dataset).

Input: *IFC-SPF*: The serialized IFC file with the extension of '.ife'
Output: *HDF*: The resultant serialized IFC file with the extension of '.hdf', *Back_IFC-SPF*: Reversal transformed IFC-SPF
(1) **function** IFC2HDF(*SPF*)
(2) // 0. Premise: Prepare explicit attributes of the corresponding version of EXPRESS Schema before start
(3) //1. Parse SPF into metadata and instance list with RegularExpression or others.
(4) *Lines* ← *SPF.readlines*()
(5) *Head_Section*, *Raw_inst_lines* ← *Extract*(*Lines*)
(6) *File_Description*, *File_Name*, *File_Schema* ← *MetaData*(*Head_Section*)
(7) *Parsed_insts* ← *IterAndParse*(*Raw_inst_lines*)
(8) // 2. Apply semantics to the instances based on relevant explicit attributes and classify them with IFC Entity, such as "IfcCartesianPoint"
(9) *Semantic_insts* ← *WrapAndClassify*(*Parsed_insts*)
(10) //3. Transform the semantic insts into array and further datasets in HDF by defining datatype
(11) *Datasets_in_hdf* ← *MainTransform*(*Semantic_insts*)
(12) // 4. Write HDF
(13) *HDF* ← *WriteToDisk*(*Datasets_in_hdf*)
(14) **end function**
(15) function HDF2IFC(*HDF*)
(16) // 1. Extract MetaData
(17) *File_Description, File_Name, File_Schema* ← *Extract*(*hdfFile.attrs*)
(18) *MetaData* ← *Repr*(*File_Descriptiion, File_Name, File_Schema*)
(19) //2. Parse Entity instances from datasets
(20) *Raw_inst_list* ← *Iter AndUnwrap*(*hdfFile*['*MainContent*'])
(21) //*Representwith*"*a* = *b*(*c*);" *for each instance where a is inst_id, b is i fcType, c is parameter_list*
(22) *Inst_text_list* ← *Repr*(*Raw_inst_list*)
(23) // 3. Reassemble and Write IFC-SPF
(24) *Back_IDC-SPF* ← *ReassembleAndWrite*(*MetaData*, *Inst_text_list*)
(25) **end function**

Algorithm 3: Bidirectional transformation between IFC-SPF and HDF.

2, and Z directions will be resized and then serialized into HDF, as shown in Figure 16.

3.3. Data Aggregation. As is narrated at the beginning of the Methodology section, this phase, data aggregation, is critical to integrating the transformed heterogeneous data related to the digital twin by establishing semantic links between them under a practical project ontology's supervision. The ontology enables us to extract the partial, pertinent data from HDF in a particular logic. We consider the comprehensive of the transformed HDF files as a compact version of the BIM-GIS project repository, which can hold the necessary data for multiple urban simulations or other applications. This repository will include typical modeling data and relevant analytical data. The required data for a specific integrated application can be queried [27], extracted, or even linked (to save storage) directly from the repository.

There are two main reasons why we choose to import the integrated data model in the posttransformation phase rather than directly extract data from the original or, maybe, post-reorganized data. The first reason is reusability. The city objects are usually taken as static data in digital twin applications since these models' main contents do not change so frequently. It is efficient to manage the data in the same compact format which will be handy for retrieval. Bidirectional conversions have been developed between these data with HDF, which means it is safe to store the data in the compact format, HDF, for extracting and renewing the analytical data back to the original formats. The second reason is accessibility. Krijnen et al. [32] mentioned that due to the special chunking and compression mechanism when

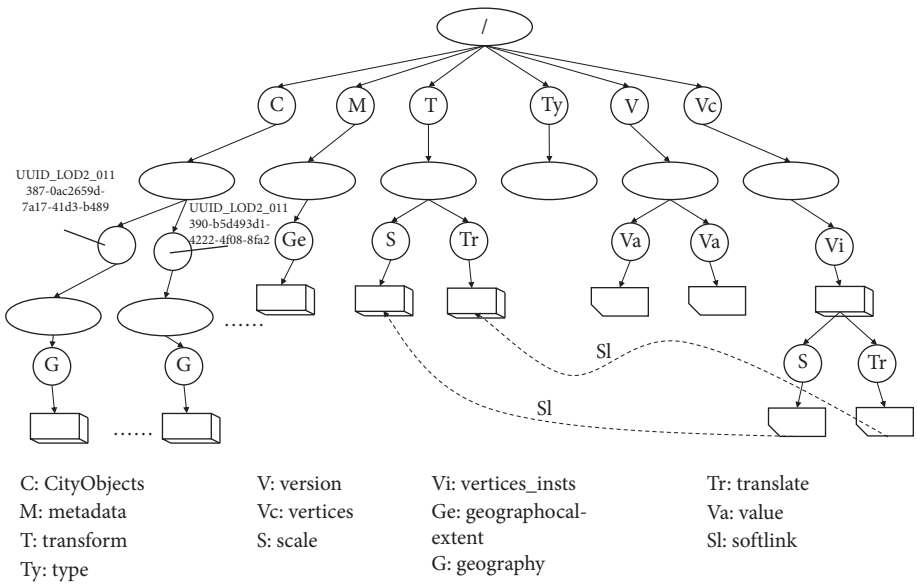


Figure 14: HDF diagram for an example CityJSON file.

Input: *CityJSON*: The serialized CityGML file with the extension of '.json'
Output: *HDF*: The resultant serialized CityJSON file with the extension of '.hdf'

```
(1) function DICTPARSE(hdfFile, aDirect, prew_path)
(2)    prev_prev_path ← prev_path
(3)    for key, valucinaDict.items() do
(4)       prev_path ← " {0}/{1}".format(prev_prev_path, key)
(5)       If type(value) ≡ dict then
(6)          DictParse(hdfFile, value, prev-path)
(7)       else
(8)          if type(value) ≡ list then
(9)             root_path ← prev_path
(10)            for i = 0 ⟶ len(value) do
(11)               item ← value[i]
(12)         if type(item) ≡ dict then
(13)            prev_path ← "{0}/_list_item_{1}".format(root_path, i)
(14)            DictParse(hdfFile, item, prev_path)
(15)         else
(16)            //Create datasets with pretreated array from regular/semiregular/irregular data
(17)         end if
(18)         end for
(19)      else
(20)         if type(value) ∈[str, int, float] then
(21)            // Convert these kinds of data into attributes
(22)         end if
(23)      end if
(24)   end if
(25)   end for
(26) end function
(27) function CITYJSON2HDF (CityJSON)
(28)   //1. Prepare source json and target hdf.
(29)   jDict ← CityJSON
(30)   hdfFile ← Open(name, mode = 'ω')
(31)   //2. Recurse DietParse function
(32)   DictParse(hdfFile, jDict,".")
(33)   //3. Write HDF
(34)   HDF ← WriteToDisk(Datasets_in_hdf)
(35) end function
```

Algorithm 4:Transformation from CityJSON to HDF.

```
Input: HDF: The resultant serialized CityJSON file with the extension of '.hdf'
Output: CityJSON: The serialized CityGML file with the extension of '.json'
(1) function GROUPWALK(group, grandParentNode, parentNode, rootDict)
(2)   for attr ∈ group.attrs do
(3)     // Attributes are parsed, respectively, according to their data type, such as string, int, and float
(4)   end for
(5)   specialList ← List()//Prepare to hold the objects in "_list_item_#num"
(6)  for elem ∈ group do
(7)     if IsStarWith(elem.name, "_list") then
(8)       newNode ← GroupWalk(elem, parentNode, {}, {})
(9)       specialList.append(newNode)
(10)    else
(11)      if type(elem) ≡ Dataset then
(12)        rawList ← Listify(elem)
(13)       // Remove the Paddings (-1) if the datasets are padded
(14)        treatedList ← checkAndParse(rawList)
(15)        parentNode[elem.Name] ← treatedList
(16)    else
(17)      if type(elem) ≡ Group then
(18)        grandParentNode ← parentNode
(19)        childNode ← parentNode[elem.Name] ← {}
(20)        parentNode ← childNode
(21)        rootDict ← GroupWalk(elem, grandParentNode, parentNode, rootDict)
(22)      end if
(23)    end if
(24)    end if
(25)  end for
(26)  if len(specialList) ≠ 0 then
(27)    grandParentNode[group.Name] ← specialList
(28)  end if
      return rootDict
(29)  end function
(30)  function HDF2CITYJSON(HDF)
(31)    //1. Prepare source hdf.
(32)    hdfFile ← Open(name, mode='r')
(33)    // 2. Recurse GroupWalk function
(34)    rootDict ← GroupWalk(hdfFile, {}, {}, {})
(35)    //3. Write json
(36)    json.dump(rootDict, CityJSON)
(37) end function
```

Algorithm 5: Transformation from HDF to CityJSON.

storing data in HDF, it costs much less both time and data size of reading. Since there are many requests for real-time analysis and response in BIM-GIS web-based applications, it is natural to choose the extraction steps with HDF.

The technique route to aggregate the related transformed city data can be followed as above (Figure 17). The content which is enclosed by the dashed block is still under experiment. The urban ontology for BIM-GIS integration is designed as shown in Figure 18. For simplicity, the relationships of the entities have been omitted in this figure. This ontology is composed of 7 subontologies, including Analysis, Facility, Feature, Geometry, PhysicalElement, SpatialElement, and Resource. Each subontology will maintain the essential elements for the BIM-GIS integrated applications. The ontologies' function is to organize the relevant, valuable data in a particular logic so that the relationship between the different kinds of pretreated source data can be revealed. It is efficient to retrieve the information from these HDFs and instantiate them through the ontology. The HDFPath entity is set in case some resource data are hard to instantiate. It records the link of the target file and the path to access the data. Moreover, in 3.2.1 IFC-SPF to HDF transformation, the index map appending to the main content was mentioned. It is beneficial for data retrieval. Since we have reordered the entity instances in the original IFC file, the identifiers have become consistent, which means it is able to get the target table and index through "Index_map [instance_id-1] [1]" in constant-time query with no relevance to the count of the instances in original IFC-SPF.

Section summary: the research's core purpose is to explore a practical solution to relieve the storage burden and improve the capacity of data mining of heterogeneous data in BIM-GIS cooperative domains. We propose the methodology based on developing a comprehensive of three

Input: *InputPath*: Folder path stores appearances, *OutputPath*: resultant hdfs' folder path
Output: *Files*: HDF Files

```
(1) function IMAGE2HDF(InputPath, OutputPath)
(2)     img_list ← List()
(3)     Create hdf File ⟶ f
(4)     f.create_group("appearances")
(5)     Get image file names ⟶ f
(6)     for i = 0 ⟶ len(name_list) do
(7)         image_name ← name_list[i]
(8)         image_dir ← path.join(path, image_name)
(9)         // openev.imread can read multiple image formats
(10)        img ← opencv.imread(dir_image)
(11)        // opencv.cvtColor can change the color matrix into another assigned form
(12)        img ←opencv.cvtColor(img, opencv.COLOR_BGR2YCR_CB)
(13)        // opencv.imencode can encode image matrix to a streaming array
(14)        //img_fps is the target image quality, range: (0, 100)
(15)        img_param ← [int(opencv.IMW RITE_JPEG_QUALITY), img_fps]
(16)        img_encode ← opencv.imencode(".JPEG," img, img, img-param) [1]
(17)        img_list.append(img_encode)
(18)        f["appearances"].create_dataset(name = img_name, dtype = unit8,
(19)        data = img_encode, compression = "gzip," chunks = true, shuffle = true)
(20)    end for
(21) end function
```

Algorithm 6: Image processing: save image to hdf and regenerate image from hdf (the image format can be JPG, PNG, BMP, and TIFF).

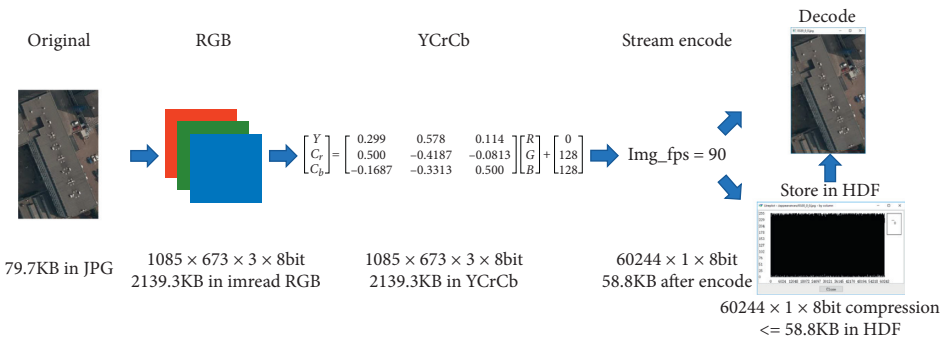


Figure 15: Flowchart of image processing with an exemplary image (image source from [35]).

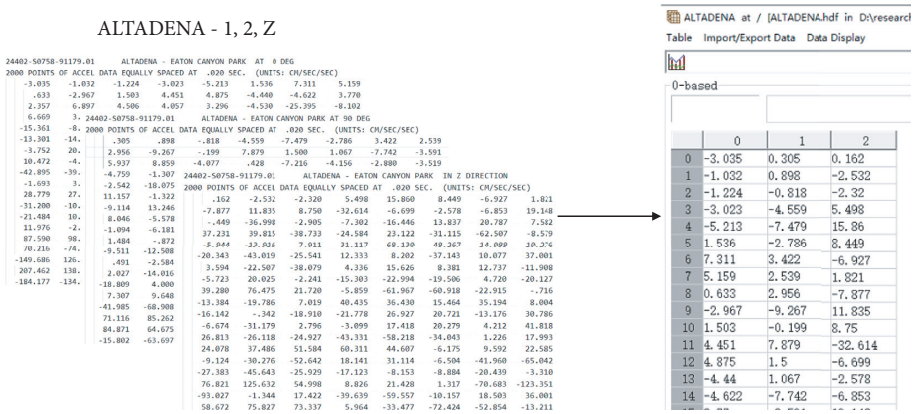


Figure 16: An example of transformed analytical data in HDF.

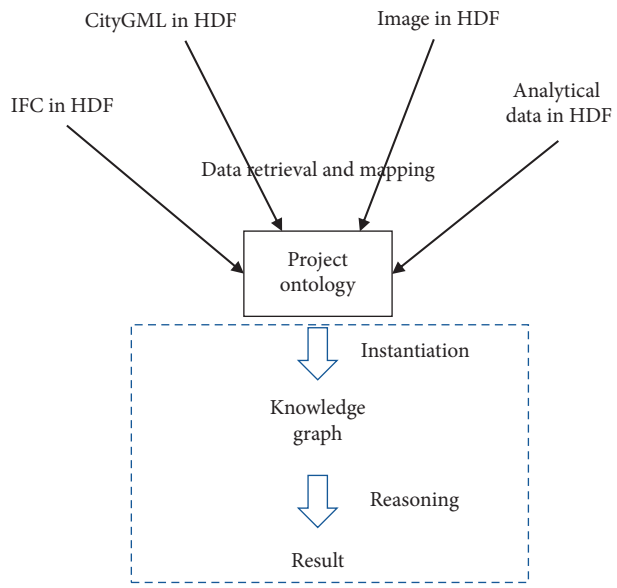


Figure 17: Technique route of data aggregation phase.

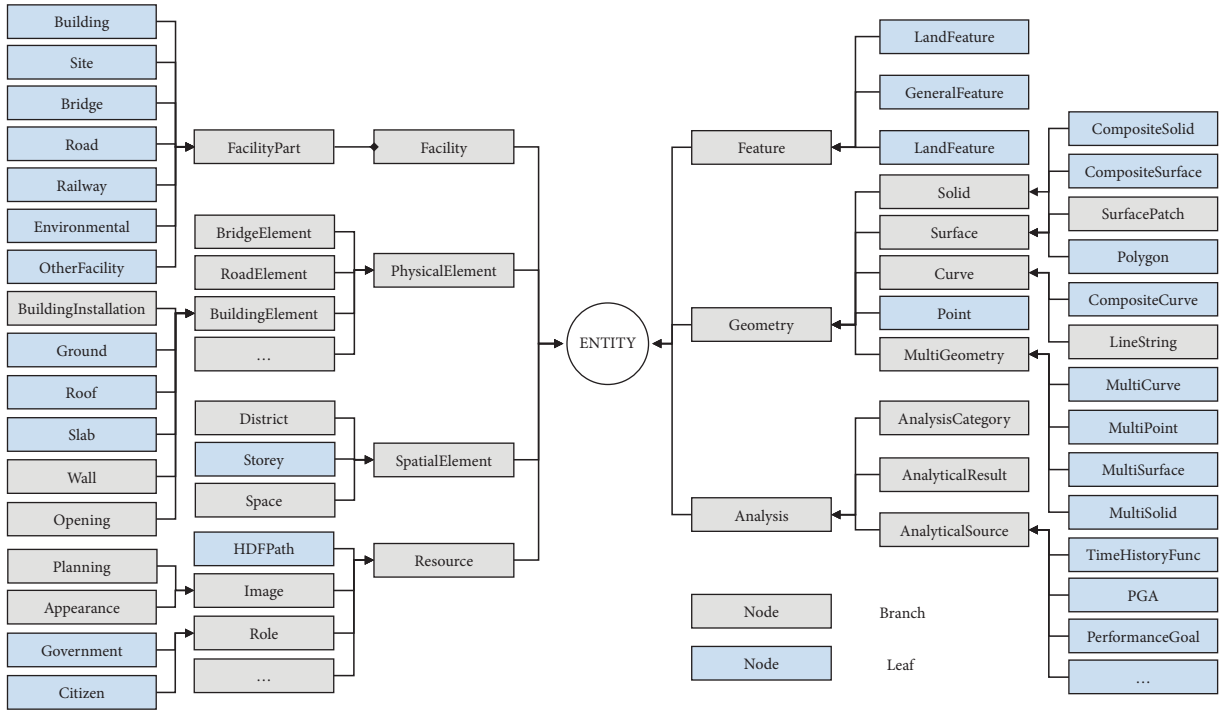


Figure 18: The experimental urban ontology designed to aggregate information from the transformed data.

essential procedures in this section. Through data reorganization and data transformation against data resources, the valuable data are reserved and eventually transformed into the candidate compact and highly efficient data format, so-called HDF, for further data mining processes. During data aggregation, a project ontology can be imported, and further retrieval and mapping from the transformed HDFs should meet the requirements of this ontology. Some adjustments have been made in previous phases like reordering and index maps to extract the desired information more efficiently.

4. Result Evaluation

4.1. For IFC-SPF. Seven test models were chosen to verify the feasibility and effectiveness of the proposed procedures. These models can be divided into three groups according to their volume as (1) group 1-tiny model: Model 1 and Model 2; (2) group 2-medium model: Model 3 and Model 4; (3) group 3-big model: Model 5, Model 6, and Model 7. The detailed information about the component count and file size of these models is listed in Figure 19. Model 6 and Model

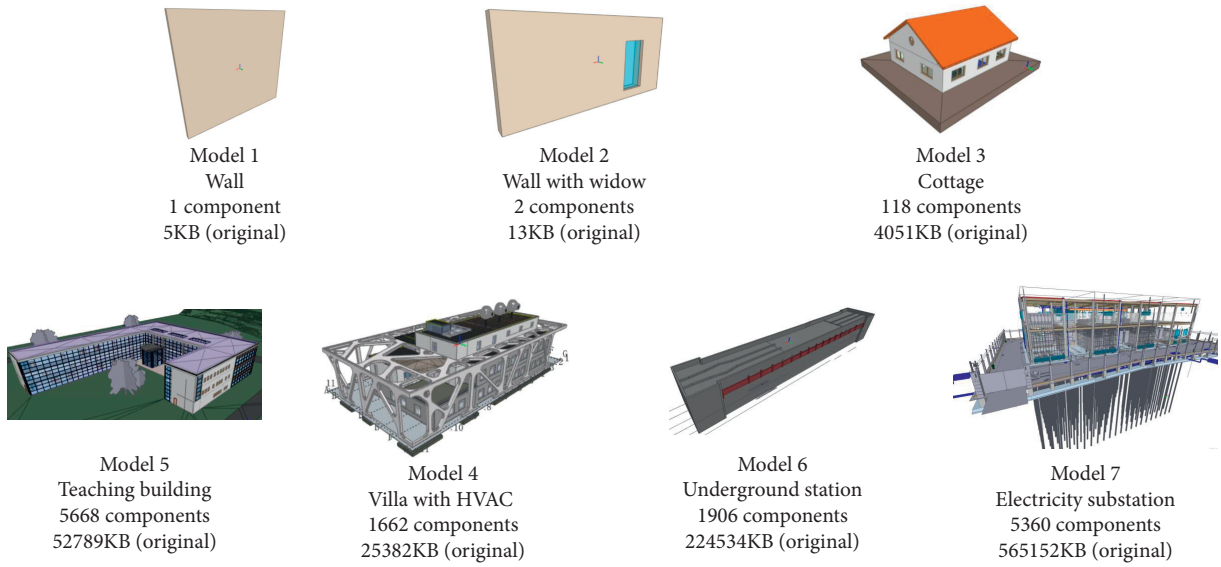


Figure 19: Detailed information of 6 test model cases in IFC-SPF.

7 are real-world infrastructure models that represent HongCaoLu underground station and ZhongChun electricity substation in Shanghai. The two models have involved the architecture and construction sections and the MEP (Mechanical, Electrical, and Plumbing) modeling data; herein, the corresponding IFC-SPF volumes reach 224534 KB and 565152 KB, respectively.

Primary data checks can be run in multiple IFC-SPF editors, including BIMVision (Figure 20). It can be checked whether the component count remains the same, whether components are well positioned or not, and whether the property sets are identical in the property section. It concludes that the redundant data will be eliminated without data loss after executing the proposed procedures.

The duplicate count table of the 3 topmost redundant types for 7 IFC-SPF cases (Table 1) was generated through the comparison before and after the pretreatment process. From this table, it can be found that, in most cases, IfcCartesianPoint is the most redundant type (in Model_7, its rank remains 4^{th} with 1145153 duplicates).

Figure 21 illustrates the result of the two procedures on IFC-SPF. It indicates that the process for transforming the original IFC-SPF into the targeted data format with reorganization pretreatment performs remarkably on giant models such as Models 5, 6, and 7. However, there might be some abnormal expansion in tiny models such as Models 1 and 2, which is probably caused by the intrinsic reserved space for each HDF. Fortunately, the number of those tiny models involved in urban-level data management is usually relatively low, so we focus on the performance of giant model data compression and further transmission.

Figure 21 also shows that it is possible to save even more transformation time and storage space due to duplicate removal and instance reordering compared to the previous study. The largest test model reaches 565,152 KB in size with 270,152 KB of the intermediate optimized model and 67,544 KB of the final output HDF. The optimization ratio comes to 88% of the original file and 75% of the optimized file, saving almost 300 MB after data reorganization and almost 500 MB after transformation. Furthermore, since the reverse transformation (from HDF to IFC) has been realized in this research, the resultant HDF can now be treated as the compact variant of IFC for efficient query and other data retrieval processes.

4.2. For CityJSON. The following table depicts the sizes of the regular datasets such as "vertices" and "transform" objects depending on JSON (compact version) and HDF, respectively, to testify HDF's efficiency in storing GIS information vividly (the source CityJSON files here originated from [35], and the resultant HDF cost savings can be detected in Table 2).

In this research, the bidirectional conversion of CityJSON and HDF has been accomplished. We verified the new resultant CityJSON files with cjio (a Python CLI to manipulate and validate CityJSON [47]) and got validated with the reversal transformation.

An example of the transformed HDF from CityJSON is demonstrated in Figure 22. This figure expresses almost all details when transforming CityJSON, including forming the basic hierarchical structure, dealing with simple datasets by using attributes, padding the irregular arrays, and deconstructing the object list into "_list_item_#num" subgroups.

So far, since we chose to migrate the whole explicit hierarchy into HDF, although it reserves the original structure, it might cause unpleasant inflation in storage. The redundancy of the original structure can be seen in, for instance, the identical expression for the same kind of city objects, as shown in Figure 22. This kind of hierarchical redundancy will significantly affect the performance of HDF storage. In future work, we consider learning experience from EXPRESS and introduce an object-oriented approach to CityGML/CityJSON to further simplify the expressions of

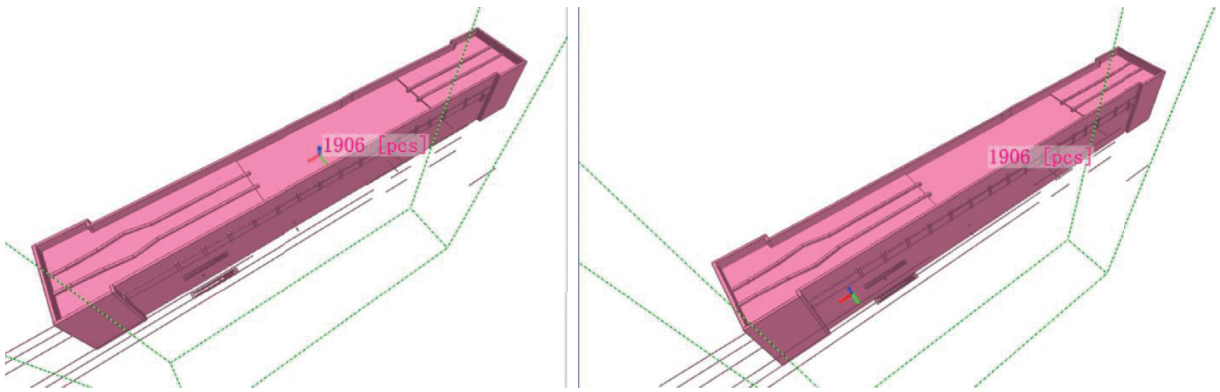


Figure 20: Comparison between original (left) and optimized (right) checked in BIMVision (a free viewer for IFC) (pcs means the count of components here).

Table 1: Partial duplicate count for 7 test IFC-SPFs.

Model	Original size (KB)	1st redundant type	Count	2nd redundant type	Count	3rd redundant type	Count
Model_1	5	IfcCartesianPoint	6	IfcDirection	6	IfcAxis2Placement3D	3
Model_2	13	IfcCartesianPoint	15	IfcDirection	11	IfcPropertySingleValue	5
Model_3	4051	IfcCartesianPoint	10160	IfcPresentationStyleAssignment	2612	IfcSurfaceStyle	2605
Model_4	25382	IfcCartesianPoint	59628	IfcPolyLoop	28444	IfcFaceOuterBound	28376
Model_5	52789	IfcCartesianPoint	9879	IfcAxis2Placement3D	7191	IfcShapeRepresentation	4992
Model_6	224534	IfcCartesianPoint	1211164	IfcAxis2Placement4D	99665	IfcShapeRepresentation	99665
Model_7	565152	IfcPolyLoop	1289515	IfcFaceOuterBound	1286657	IfcFace	1286632

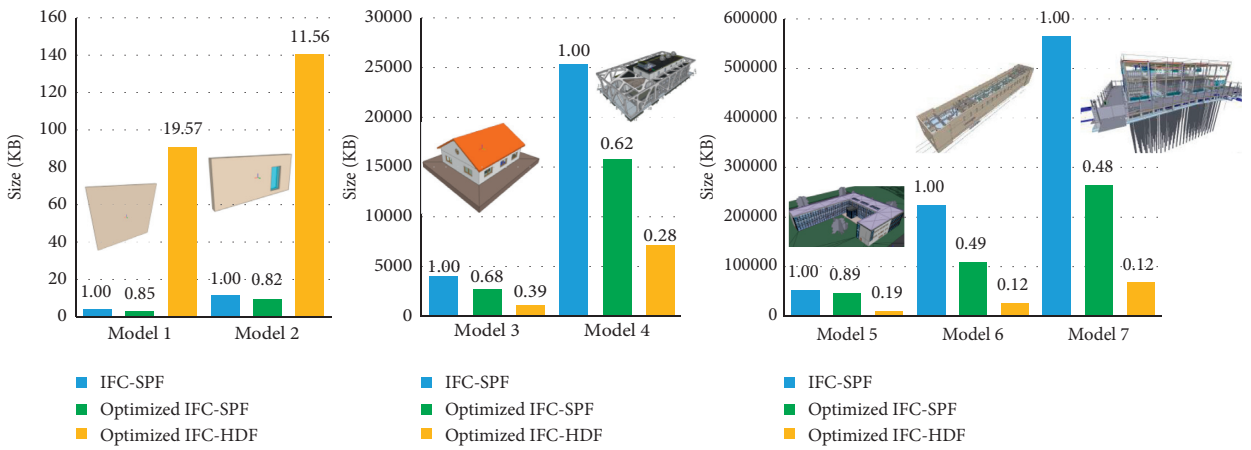


Figure 21: Comparison of the file size of IFC-SPF, optimized IFC-SPF, and IFC-HDF (optimized IFC-SPF means the intermediate IFC-SPF after data reorganization).

particular objects and make full use of HDF compound datasets.

5. Discussion

In our perspective, three main capacities of digital twin should be achieved: (1) containing as many city features as possible, (2) synchronizing with its host dynamically through multiple urban data resources, and (3) forming a real-time integrated response to the events. BIM and GIS, as two fundamental technologies of digital twin, aim at exploiting urban data from the micro and macro level. Previous studies have shown that further merging the micro and macro level urban data can bridge the cross-domain data analysis gap. A considerable number of studies have been conducted to integrate BIM and GIS technologies. Much effort was put on extending one original data structure to help embody the other or developing some novel integrated data structures. However, relatively little research focuses on the integrated storage and exchange format itself.

Table 2: Storage cost for vertices and transform objects in JSON and HDF.

Vertices and transform object*	JSON (KB)	HDF (KB)	Compression ratio
Two buildings	6	12	100%
3-20-DELFSHAVEN	506	158	−69%
DenHaag_01	541	164	−70%
LoD3_Railway	1378	297	−78%
VM05_2009	1512	368	−76%
Vienna_102081	2066	485	−77%
DA13_3D_Buildings_Merged	29125	7460	−74%
Zurich_Building_LoD2_V10	95239	22534	−76%

*Here, we compared the performance of the regular data containers which are mentioned in Section 3.1.2.

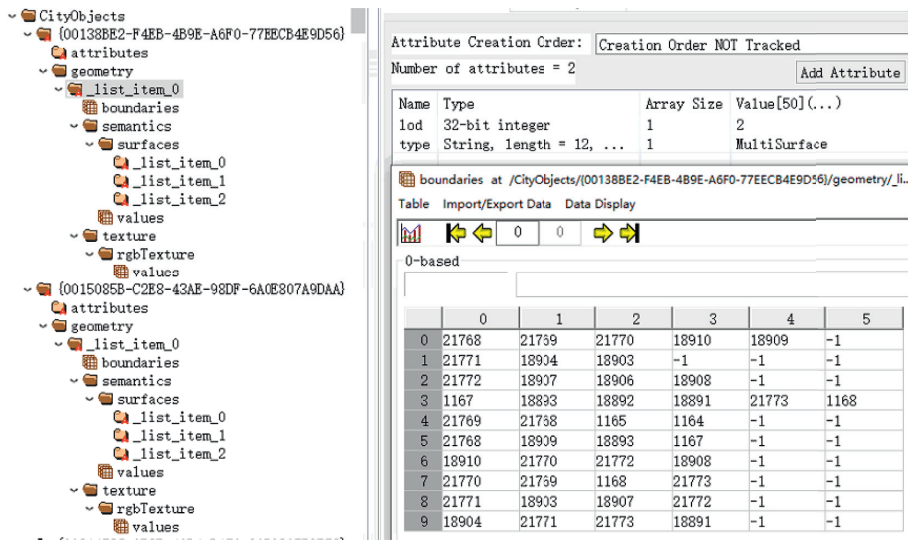


Figure 22: A demonstration of almost all details of the transformed CityJSON in HDF.

To help collect the valuable information from each field for further integral applications, it is also necessary to propose an appropriate scheme which should obtain high efficiency of integration under the premise of little information loss.

Both BIM and GIS have been ameliorating their own open official data standard. This paper analyzed the defects of currently prevalent official data formats when chosen as an alternative media for integration, including text-based, verbose, redundant, storage-challenging, and incomplete. To solve this dilemma, we propose an HDF-based comprehensive scheme of compact, vendor-neutral, and heterogeneity-friendly approach to support the integration of BIM-GIS. The scheme is constituted with three subprocedures: data reorganization, data transformation, and data aggregation.

First, in the data reorganization part, we contribute to proposing the duplicate removal and instance reordering with pure text processing no matter which programming language is chosen (we have succeeded in Python and C#) and which version of IFC schema is followed. It is fruitful to have the original file get slim (51% reduction in Model 6 and 52% reduction in Model 7) and relieve the burden of storage and exchange of detailed structural models in digital twin. Additionally, instance reordering improves the query efficiency through the index map since it is already consistent in identifiers. Moreover, we treat the data in CityJSON as regular, semiregular, and irregular and pretreat them separately to make them prepared for further transformation.

Next, in the data transformation phase, the bidirectional conversion to HDF of all four kinds of data is realized in city information modeling. For IFC-SPF, we add the index map and file metadata of the original IFC in HDF and then back-convert the data from HDF to IFC. Besides, the version-neutral transformation process can be fulfilled without importing a special IFC processor. The transformation efficiency is very satisfactory for IFC since it is EXPRESS-based and object-oriented. The test results show that the largest test model can be first optimized from 565,152 KB into 270,152 KB by data reorganization and finally transformed into 67,544 KB of HDF. For CityJSON, we also propose the bidirectional conversion algorithms with recursion to improve interoperability. All the hierarchical structures of the CityJSON are retained, and the redundancy of hierarchy is, to some extent, harmful for HDF storage since the datasets become small but numerous. For images, we apply the encode and decode process of a standard image. The bidirectional transformation is especially vital to the interoperability, representing the ability to convert the modified HDF (if it happens in further analysis) back into the original file format to make it entirely consistent.

Finally, in the data aggregation phase, a technique route is proposed to establish the semantic relationships between the objects in transformed heterogeneous resources under a project ontology's supervision. In most application scenarios, there is no need for all the related models' information to get analyzed. Populating all the instances from BIM and GIS of an integrated project into a unified urban ontology will induce storage waste. Thus, we consider an HDF-based project repository as the critical base waiting for data extraction to further instantiate only the required data with the target application ontology. By retrieving the relevant data from transformed HDF files, it is possible to further save query and storage costs. In the future, more practical experiments will be conducted to evaluate this phase.

6. Conclusion and Future Work

Being processed under the proposed comprehensive data reorganization, data transformation, and data aggregation, the potentially valuable information will be fully restored. The data storage will be saved about 60%–80% for IFC-SPF and 60%–70% for regular CityJSON contents, more for original CityGML files. The effect on semiregular and irregular data in CityGML deserves further research. We believe this light-weight processing and transformation for both data formats might be surprisingly profitable in further integrated BIM-GIS applications.

Until now, although the intrinsic information of both BIM and GIS has been extracted compactly and transformed together in storage and exchange level integration and partially semantic level integration, there are still two issues remained to be solved: (1) the organization of this proposed integrated data structure is still relatively loose and dispersed and (2) the semantic ambiguity of interfields might cause novel redundancy. To help guide the organization of the integrated data structure of BIM-GIS in HDF, in future research, more further efforts will be made to improve our project ontology. Besides, the next step to do is to examine the proposed ontology in more practical scenarios.

Additionally, through this research, it has been found out that the data transformation of CityGML usually takes more effort than that of IFC, which is caused by data regularity. IFC standard is derived from EXPRESS, and the physical file can be easily divided into defined entities. Each instance of a defined entity has the same attribute matrix, making it easier to be transformed into HDF datasets. In future work, we consider absorbing both the original JSON structure and object-oriented approach to simplify the expressions of specific objects in CityGML/CityJSON to achieve more satisfying results when parsing into HDF.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Additional Points

Highlights. (i) Proposing an HDF based synthesized scheme towards the practical application of BIM-GIS integration with data reorganization, transformation, and aggregation processes. (ii) Executing efficient data reorganization with redundancy elimination and reordering for original IFC-SPF and CityGML. (iii) Presenting bidirectional and vendor-neutral data transformation methods to embody heterogeneous data formats including model data, images, and analytical datasets in urban projects. (iv) Delivering a compact representation for BIM-GIS integrated information based on HDF.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The research project presented in this paper was supported by Shanghai Science and Technology Innovation Plan (15DZ1203403) provided by Science and Technology Commission of Shanghai Municipality 15DZ1203403 and the Research Plan of Multidimensional Information Model System in Power Transmission and Transformation Engineering provided by State Grid Corporation of China.

References

[1] H. Wang, Y. Pan, and X. Luo, "Integration of BIM and GIS in sustainable built environment: a review and bibliometric analysis," *Automation in Construction*, vol. 103, pp. 41–52, 2019.

[2] A. Borrmann, T. H. Kolbe, A. Donaubauer, H. Steuer, J. R. Jubierre, and M. Flurl, "Multi-scale geometric-semantic modeling of shield tunnels for GIS and BIM applications," *Computer-Aided Civil and Infrastructure Engineering*, vol. 30, no. 4, pp. 263–281, 2015.

[3] M. Wang, Y. Deng, J. Won, and J. C. P. Cheng, "An integrated underground utility management and decision support based on BIM and GIS," *Automation in Construction*, vol. 107, Article ID 102931, 2019.

[4] E. P. Karan, J. Irizarry, and J. Haymaker, "BIM and GIS integration and interoperability based on semantic web technology," *Journal of Computing in Civil Engineering*, vol. 30, no. 3, Article ID 04015043, 2015.

[5] C. Kuster, J.-L. Hippolyte, and Y. Rezgui, "The UDSA ontology: an ontology to support real time urban sustainability assessment," *Advances in Engineering Software*, vol. 140, Article ID 102731, 2020.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] J. Ninić, C. Koch, and J. Stascheit, "An integrated platform for design and numerical analysis of shield tunnelling processes on different levels of detail," *Advances in Engineering Software*, vol. 112, pp. 165–179, 2017.

[8] J. Zhu, X. Wang, P. Wang, Z. Wu, and M. J. Kim, "Integration of BIM and GIS: geometry from IFC to shapefile using open-

source technology," *Automation in Construction*, vol. 102, pp. 105–119, 2019.

[9] J. Zhu, X. Wang, M. Chen, P. Wu, and M. J. Kim, "Integration of BIM and GIS: IFC geometry transformation to shapefile using enhanced open-source approach," *Automation in Construction*, vol. 106, Article ID 102859, 2019.

[10] A. A. Diakite and S. Zlatanova, "Automatic geo-referencing of BIM in GIS environments using building footprints," *Computers, Environment and Urban Systems*, vol. 80, Article ID 101453, 2020.

[11] T. W. Kang and C. H. Hong, "A study on software architecture for effective BIM/GIS-based facility management data integration," *Automation in Construction*, vol. 54, pp. 25–38, 2015.

[12] Z. Xu, L. Zhang, H. Li, Y.-H. Lin, and S. Yin, "Combining IFC and 3D tiles to create 3D visualization for building information modeling," *Automation in Construction*, vol. 109, Article ID 102995, 2020.

[13] A. Rafiee, E. Dias, S. Fruijtier, and H. Scholten, "From BIM to geo-analysis: view coverage and shadow analysis by BIM/GIS integration," *Procedia Environmental Sciences*, vol. 22, pp. 397–402, 2014.

[14] H. Ledoux, K. A. Ohori, K. Kumar, B. Dukai, A. Labetski, and S. Vitalis, "CityJSON: a compact and easy-to-use encoding of the CityGML data model," *Open Geospatial Data, Software and Standards*, vol. 4, no. 1, p. 4, 2019.

[15] Y. Deng, J. C. P. Cheng, and C. Anumba, "Mapping between BIM and 3D GIS in different levels of detail using schema mediation and instance comparison," *Automation in Construction*, vol. 67, pp. 1–21, 2016.

[16] "OGC Hierarchical Data Format Version 5 (HDF5®) Standard | OGC," 2020, https://www.opengeospatial.org/standards/HDF5.

[17] V. Zverovich, L. Mahdjoubi, P. Boguslawski, F. Fadli, and H. Barki, "Emergency response in complex buildings: automated selection of safest and balanced routes," *Computer-Aided Civil and Infrastructure Engineering*, vol. 31, no. 8, pp. 617–632, 2016.

[18] "Data Serialization – the Hitchhiker's Guide to Python," 2020, https://docs.python-guide.org/scenarios/serialization.

[19] K. Afsari, C. M. Eastman, and D. Castro-Lacouture, "JavaScript Object Notation (JSON) data serialization for IFC schema in web-based BIM data exchange," *Automation in Construction*, vol. 77, pp. 24–51, 2017.

[20] P. Pauwels and W. Terkaj, "EXPRESS to OWL for construction industry: towards a recommendable and usable ifcOWL ontology," *Automation in Construction*, vol. 63, pp. 100–133, 2016.

[21] O. Rübel, A. Greiner, S. Cholia et al., "OpenMSI: a high-performance web-based platform for mass spectrometry imaging," *Analytical Chemistry*, vol. 85, no. 21, pp. 10354–10361, 2013.

[22] "IFC formats – buildingSMART technical," 2020, https://technical.buildingsmart.org/%20standards/ifc/ifc-formats/.

[23] "IFC schema specification – buildingSMART technical," 2020, https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/.

[24] "CityGML | OGC," 2020, https://www.opengeospatial.org/standards/citygml.

[25] Z.-Z. Hu, S. Yuan, C. Benghi et al., "Geometric optimization of building information models in MEP projects: algorithms and techniques for improving storage, transmission and display," *Automation in Construction*, vol. 107, Article ID 102941, 2019.

[26] G. Lee, C. M. Eastman, and R. Sacks, "Eliciting information for product modeling using process modeling," *Data & Knowledge Engineering*, vol. 62, no. 2, pp. 292–307, 2007.

[27] T. Krijnen and J. Beetz, "A SPARQL query engine for binary-formatted IFC building models," *Automation in Construction*, vol. 95, pp. 46–63, 2018.

[28] M. Theiler and K. Smarsly, "IFC Monitor - an IFC schema extension for modeling structural health monitoring systems," *Advanced Engineering Informatics*, vol. 37, pp. 54–65, 2018.

[29] T. Krijnen and J. Beetz, "An IFC schema extension and binary serialization format to efficiently integrate point cloud data into building models," *Advanced Engineering Informatics*, vol. 33, pp. 473–490, 2017.

[30] A. Jaly-Zada, C. Koch, and W. Tizani, "IFC extension for design change management," in *Proceedings of the 32nd CIB W78 Conference*, Eindhoven, The Netherlands, October 2015.

[31] "HDF5 documentation," 2020, https://portal.hdfgroup.org/display/HDF5/HDF5.

[32] T. F. Krijnen, *Efficient Storage and Retrieval of Detailed Building Models: Multi-Disciplinary and Long-Term Use of Geometric and Semantic Construction Information Eindhoven*, Technische Universiteit Eindhoven, Eindhoven, Netherlands, 2019.

[33] W. Duane, D. Livingstone, and D. Kidd, "Integrating environmental models with GIS: an object-oriented approach utilising a hierarchical data format (HDF) data repository," *Transactions in GIS*, vol. 4, no. 3, pp. 263–280, 2000.

[34] T. F. Krijnen, J. Beetz, S. Ochmann, R. Vock, and R. Wessel, "Towards extending IFC with point cloud data," in *Proceedings of the 22nd EG-ICE Workshop*, pp. 13–15, Eindhoven, The Netherlands, June 2015.

[35] H. – CityJSON, https://www.cityjson.org/, 2020.

[36] G.-K. Mbogo, S. V. Rakitin, and A. Visheratin, "High-performance meteorological data processing framework for real-time analysis and visualization," *Procedia Computer Science*, vol. 119, pp. 334–340, 2017.

[37] C. Vitolo, Y. Elkhatib, D. Reusser, C. J. A. Macleod, and W. Buytaert, "Web technologies for environmental big data," *Environmental Modelling & Software*, vol. 63, pp. 185–198, 2015.

[38] Y. Chen, F. Wang, S. Li, B. J. Xiao, and F. Yang, "The implementation of a data acquisition and service system based on HDF5," *Fusion Engineering and Design*, vol. 112, pp. 975–978, 2016.

[39] V. Nandigam, K. Lin, M. Phan, A. Borsa, S. J. S. Khalsa, and C. Crosby, "Rapid access and visualization of Spaceborne Altimetry data from ICESat and ICESat-2," in *Proceedings of the IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 5270–5273, Valencia, Spain, July 2018.

[40] K. Ackermann and S. D. Angus, "A resource efficient big data analysis method for the social sciences: the case of global IP activity," *Procedia Computer Science*, vol. 29, pp. 2360–2369, 2014.

[41] ISO 10303-21, *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 21: Implementation Methods: Clear Text Encoding of the Exchange Structure*, International Organization for Standardization, Geneva, Switzerland, 2002, https://www.iso.org/standard/33713.html.

[42] O. B. J. Wavefront, "Summary from the encyclopedia of graphics file formats," 2020, http://www.fileformat.info/format/wavefrontobj/egff.htm.

[43] "citygml4j/citygml-tools: collection of tools for processing citygml files," 2020, https://github.com/citygml4j/citygml-tools.
[44] ISO/TS10303-26:2011 Industrial automation systems-Product data representation and exchange-Part 26: Implementation methods: Binary representation of EXPRESS-driven data, https://www.iso.org/standard/50029.html.
[45] D. Yang and C. M. Eastman, "A rule-based subset generation method for product data models," *Computer-Aided Civil and Infrastructure Engineering*, vol. 22, no. 2, pp. 133–148, 2007.
[46] OpenCV, 2020, https://opencv.org/.
[47] CityJSON/io, "Python CLI to process and manipulate cityjson files," 2020, https://github.com/cityjson/cjio.