# Research on Lightweighting and Rendering Optimization Techniques for Building Information Models

Wei Tong[1], Junfei Sun[1], Zezhong Tian[1]
Linlin Feng[1], Jia Mao[1], Shuhao Gu[1],and Xinghui Zhu[1]
[1]School of Computer Science, Xidian University, Xi'an, Shaanxi, 710162, China

**Traditional Building Information Modeling (BIM) services, using a client-server (C/S) architecture, struggle to adapt to the new demands and changes in the field of architectural informatization. The development of a comprehensive BIM visualization system, which is web-based, requires no installation or downloading, is cross-platform, and facilitates easy sharing, has become a new pathway for BIM advancement. However, when rendering the vast amount of data in BIM three-dimensional scenes on the web, several issues persist, including low transmission efficiency, slow loading speed, laggy screen display, and low rendering frame rates, particularly on mobile devices and terminals with limited hardware performance. Therefore, this paper proposes BIM lightweighting techniques based on geometric data and texture information, as well as a web-based rendering optimization method for three-dimensional models. In terms of BIM lightweighting, more efficient methods and algorithms are employed to simultaneously lighten the geometric data and texture information of the models. For rendering optimization, an efficient view frustum culling algorithm is introduced, along with a design for an adaptive rendering strategy to enhance rendering performance.Through testing the loading efficiency of different scale BIM models before and after optimization was tested on the web. Results show that, while maintaining display accuracy, the average loading time of the optimized models was reduced by $40\%$ compared to the unoptimized models. The average data compression rate reached $45\%$, and the average memory usage decreased by $32.55\%$. After stable rendering, the frame rate was close to 60fps.**

***Index Terms*—Building Information Modeling, Lightweight Processing, Rendering Optimization, Mesh Simplification, Texture Compression**

## I. Introduction

With the development of technologies such as the Internet of Things (IoT), Building Information Modeling (BIM), as the digital entity of a building in the virtual world, together with various sensors and IoT devices, constitutes a digital mapping of the real world and is widely used in areas such as smart cities, communities, campuses, and security [1]. In this new development trend, traditional desktop-level BIM systems that use a C/S architecture have high hardware requirements for clients, increased usage and learning costs, and are difficult to adapt to new needs and changes. Building a comprehensive BIM visualization system that is web-based, platform-independent, and easy to share, without the need for installation or downloading, has become a new path for the development of BIM.

However, when applying BIM models on the Web, although the browser can provide a unified and standardized interface that is easy to expand and develop functions in the later stage [2], its computational performance is inferior to that of the client, and its resource utilization is higher. Model loading time is also longer, especially on mobile and low-performance terminal devices, which exacerbates this issue. As the demand for building information models continues to increase, BIM models are becoming more complex and sophisticated. A sophisticated BIM model not only contains tens of thousands or even millions or tens of millions of components [3], but also various geometric, texture, and material information, making the model's volume easily reach several hundred MB, GB, tens of GB or even hundreds of GB, resulting in low transmission efficiency, slow loading speed, lagging screens, and low rendering frame rates during Web loading and rendering. This seriously affects the system's reliability, stability, and user experience. Existing research at home and abroad has focused mainly on model data simplification, model-level rendering, and model scene clipping to address the aforementioned issues.

Manuscript received June 12, 2023; revised August 30, 2023. Corresponding author: Xinghui Zhu (email: xhzhu@xidian.edu.cn).

## II. Related Work

The edge collapse simplification algorithm was first proposed by Hoppe et al. , which requires constructing a global energy function to measure the edge collapse error, making it inefficient [4]. To improve the efficiency of mesh simplification, Garland et al. proposed a simplification algorithm based on Quadratic Error Metrics (QEM) [5], but lacked consideration for simplification quality. Later, various new error functions were proposed based on the QEM algorithm to improve simplification quality. Asgharian et al. reduced the number of vertices through resampling based on the Nyquist theorem [6], while preserving the important features of the model surface. Li et al. introduced angle error control to simplify the model mesh based on the QEM algorithm for vertex curvature [7]. However, literature [6], [7] considered simplification of the model texture information, and lacked consideration for the significant changes in LOD models during switching.

Wu et al. proposed a texture compression algorithm that is based on model geometry information and has high compression ratio [8]. Buyukdemircioglu et al. simplified the texture information of 3D city models in the reconstruction and efficient visualization process of heterogeneous models, by using format conversion, duplicate item removal, and

merging into texture atlases [9]. Khan et al. obtained compact texture descriptions using information-theoretic compression techniques, and combined multiple local texture descriptors to improve pattern recognition performance [10]. However, neither literature [8], [9], [10] considered the simplification of model geometry data nor had an effective organization of multi-level textures.

Xu et al. proposed a BIM model conversion method from the Industry Foundation Classes (IFC) to the 3D Tiles data format [11]. During the conversion process, coordinate transformation, data mapping, spatial indexing, and detail level partitioning were performed, with the aim of improving the traditional model simplification approach and enhancing model loading speed and rendering performance. Zhu et al. presented a semantic-based entity model simplification method that maintains the consistency of the model semantics before and after simplification by studying the IFC semantic constraints [12]. However, literature [11], [12] did not preserve the hierarchical relationship of semantic information before and after merging, which brings certain difficulties to querying semantic information.

Che et al. proposed a feedback-based hierarchical combination optimization method for constructing more realistic 3D scenes [13]. Abualdenien et al. introduced a multi-level-of-detail (LOD) meta-model to explicitly describe the requirements of LOD and ensure the consistency between geometry and semantic information, as well as the topological consistency between different LODs [14]. Wang et al. presented a quasi-continuous LOD algorithm for lightweight rendering, and incorporated relevant constraints during the process of adjusting model details to address issues such as model abrupt changes and distortions [15]. However, literature [13], [14], [15] all lack consideration of data simplification for the models themselves.

Zhang et al. proposed a visualization method based on variable-depth embedding clustering fusion and Hilbert R-tree, aiming to address the issues of slow display and lagging when rendering massive geological data [16]. Ströter et al. introduced a new boundary volume hierarchy for GPU-accelerated direct volume rendering, as well as for volume network slicing and inside-outside intersection testing [17]. Hui et al. addressed the difficulty of model space collision detection by proposing an intelligent algorithm that integrates bounding boxes, which effectively improves the efficiency of collision detection in virtual scenes [18]. Although they improved the rendering efficiency of models from various perspectives, the literature [16], [17], [18] lacks consideration of algorithm complexity and the rendering effect is inadequate.

In summary, model lightweighting and rendering optimization are complementary, and currently, some mainstream BIM lightweighting methods still have some problems in practical applications, such as the inability to ensure model accuracy, poor rendering effects, high algorithm complexity, and neglect of the influence of texture data. Therefore, how to deeply simplify model data while ensuring model display accuracy, improve rendering frame rate during visualization, reduce model switching abruptness, and alleviate the rendering pressure of computers, is still a problem that needs to be solved urgently. This paper proposes a BIM lightweighting method based on geometric data and texture information, which aims to reduce the large amount of data in BIM models, enable them to be smoothly displayed on browsers and mobile devices, and lower the threshold for BIM services. Therefore, the web-based BIM model lightweighting technology studied in this paper has significant research significance and broad prospects. It has significant value in accelerating the application of BIM technology in practical engineering fields, expanding the application scope of BIM technology, and promoting the transformation of the construction industry towards digitalization and intelligence.

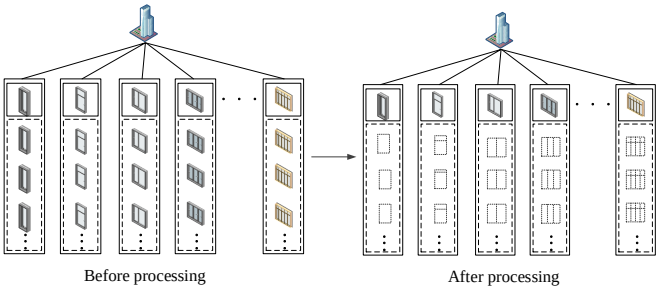



Fig. 1: BIM model before and after redundant component processing

## III. BIM Lightweight Processing Method

### *A. Based on Structural Matching: A Method for Retrieval and Removal of Redundant Components in BIM*

The building components in Building Information Modeling (BIM) are organized and managed in a tree-like structure based on categories, families, family types, and family instances [19]. Fig. 1 shows the comparison before and after removing redundant components.The section uses a BIM redundant component retrieval and removal method based on structural matching, with the matching and alignment information of components used as the basis for determining redundant components, thus removing a large number of redundant components [20]. This method has higher efficiency, accuracy, and redundancy removal rate compared to the traditional geometric instantiation method based on component reuse. The main idea is as follows:

1) first, by constructing an Item-based Structure Graph (ISG) based on components, the matching and alignment problem of building components is converted into an ISG structural matching problem.
2) Secondly, based on the Factorized Graph Matching algorithm and the Iterative Closest Point algorithm, the ISG matching model is obtained.
3) Then, ISG pre-matching, ISG full matching, and ISG partial matching are used to process it, achieving fast alignment of BIM components.
4) Finally, the component alignment results are used as the basis for component retrieval, thus removing redundant components.

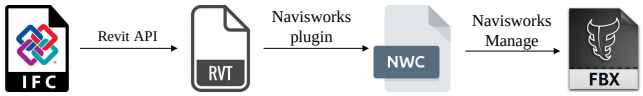



Fig. 2: BIM model before and after redundant component processing

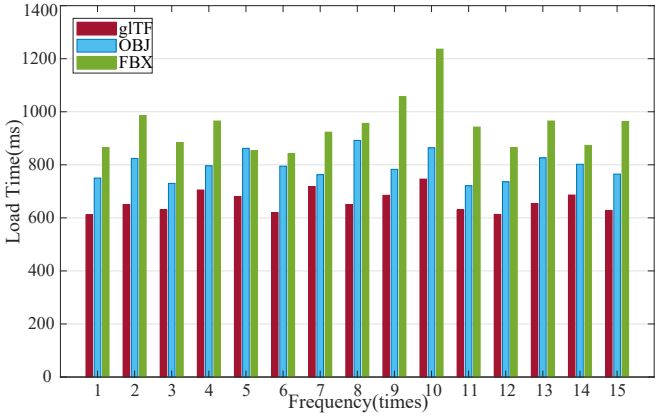


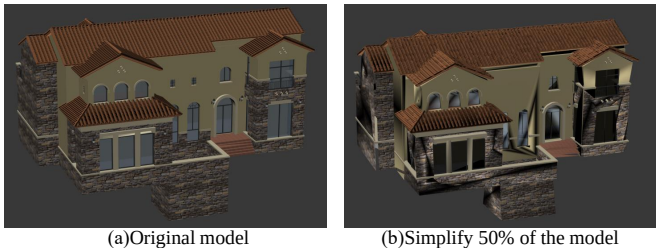

Fig. 3: BIM model before and after redundant component processing

### *B. Model Format Selection and Processing*

Although IFC files are one way to represent building information models, they cannot be rendered directly on the web and must be converted to a certain format [21]. The typical process for converting file formats is shown in Fig. 2. After a series of transformations and processing, the final result is an FBX model with materials and textures.To further lighten the BIM model, the model format is converted from the perspective of loading and rendering on the web. This is because different formats of models have different rendering performance on the web, as shown in Fig. 3. By conducting 15 loading tests on different formats of the same model on the web, the results show that the glTF format has the best effect, not only compressing the size of the file itself but also having a faster average loading speed than the other two. Therefore, considering both the lightweight of the model and the rendering optimization, this paper chooses to further convert the fbx format model to the glTF format model.

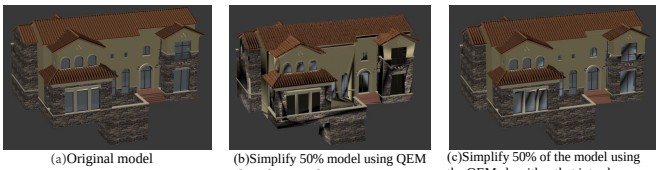
### *C. The Screen-Space Error and Angle Error Correlated Edge Collapse Mesh Simplification Algorithm*

After format conversion, the current BIM model's geometric information representation is essentially consistent with the classical 3D model data features. Therefore, mesh compression algorithms in computer graphics can be used to further reduce the weight of the current BIM model. This section proposes a screen-space and angle-error-related edge-collapse mesh simplification algorithm to further simplify the geometry data of the model. Firstly, based on the quadratic error measure algorithm, vertex curvature and angle error control are introduced to maintain better geometric features of the model. Then, screen-space error is introduced to reduce the number of inaccurate triangle meshes, thereby further reducing the weight



(a)Original model (b)Simplify 50% of the model

Fig. 4: Comparison of model simplification using the QEM algorithm with introduced vertex curvature



(a)Original model (b)Simplify 50% model using QEM algorithm introducing vertex curvature (c)Simplify 50% of the model using the QEM algorithm that introduces vertex curvature and angle errors

Fig. 5: Comparison of model simplification using the QEM algorithm with introduced vertex curvature

of the model and improving its display effect.The specific algorithm process is as follows:

(1) Description of the QEM Algorithm

The Quadratic Error Metric (QEM) algorithm is an improvement over the edge-collapse algorithm, and its error metric measures the distance between a vertex and the first-order neighborhood triangle as the evaluation indicator.

(2) The Improved Algorithm Based on Vertex Curvature and Angle Error Introducing Using QEM

The introduction of vertex curvature in the QEM algorithm aims to reduce the impact of changes in edge folding order and new vertex position on the simplification effect. When sorting the weights of the folded edges, the priority of edges with large vertex curvatures is reduced, which better preserves the geometric features of the model.

The QEM algorithm with introduced vertex curvature was used to simplify a certain building model, and the results are shown in Fig. 4. It can be observed that the simplified model has poor visual quality due to the changes in the position and shape of the newly generated triangles during the simplification process.

Therefore, based on this, we introduce angle error control to control the rotation direction of the newly generated triangles, thus preserving more geometric features and improving visual effects. The angle error here refers to the maximum angle change value of the normal vectors of the intersecting triangle faces after folding the edges of a certain triangle face.

The QEM algorithm with introduced vertex curvature and triangle error was tested, and through the comparison in Fig. 5, it can be observed that under the condition of equal numbers of triangles, the algorithm is clearly superior to the QEM algorithm with introduced vertex curvature. The geometric features of the model are better preserved, resulting in better visual effects. However, some geometric features are still lost, which leads to poor performance in multi-level rendering on the Web.

Fig. 6: Schematic diagram of LOD switching generated by the above algorithm

(3) Introduce Screen-Space Error

Although the above-mentioned algorithm can simplify the building model and achieve an acceptable visual effect, it still leads to the loss of some geometric features. When the model volume reaches hundreds of MB or even GB, this disadvantage will become more obvious. Therefore, to address the existing issues, screen-space error is introduced in the process of generating LOD (level of detail) using the above-mentioned algorithm, in order to control the errors generated during LOD generation and the amount of data entering the pipeline. The main idea is to use the distorted pixel values (the size of distorted triangle meshes mapped to pixels on the screen) as the error generated during the switching of multiple detail levels, in order to reduce the amount of data entering the rendering pipeline and ensure the visual effect of the model.

The length value of the triangle mesh edge mapped to the screen can be calculated using equation 1, where $L$ represents the edge length of the triangle mesh, $d$ represents the distance between the triangle mesh and the viewpoint, and $x$ represents the length of the plane in front of the viewing frustum.

$$\mathrm{SSE}(x) = \frac{Lx}{2d\tan\frac{\theta}{2}} \tag{1}$$

The maximum pixel value mapped to the screen for a mesh with edge length $L$ can be calculated using the value obtained in the previous step. This value can be used as the mesh error, and the calculation formula is given in equation 2:

$$\Delta(\text{mesh}) = \frac{\text{Weight} \times \text{Height}}{2xy} \times \mathrm{SSE}(x) \times \mathrm{SSE}(y) \tag{2}$$

When switching between LOD levels, the value of $\Delta(\text{mesh})$ is calculated, and the LOD level with the largest error within an acceptable range of model error is selected to balance the amount of data entering the pipeline with visual perception.

The algorithm was tested on a specific building model by setting different maximum folding edge lengths and viewing distances to generate LODs of different levels. Then, the corresponding $\Delta(\text{mesh})$ values were calculated to select a suitable value as the error criterion for LOD switching, within the acceptable range of model error. The experimental results are shown in Table I, and the LOD switching diagram is shown in Fig. 6. Based on the data in the table and the switching diagram, it can be concluded that the introduction of screen-space error further improved the QEM algorithm that introduced vertex curvature and angle errors.

TABLE I: LOD parameter settings and error results generated by a architectural model

| LOD level | Total number of patches | Maximum folding edge length | Range of visibility | $\Delta$(mesh) |
|---|---|---|---|---|
| 0 | 288889 | 0 | 1 | 0 |
| 1 | 226567 | 0.05 | 10 | 62 |
| 2 | 154746 | 0.07 | 20 | 34 |
| 3 | 8026 | 0.09 | 40 | 16 |

### *D. Texture Classification and Redundant Texture Elimination*

In order to achieve a realistic effect for a 3D building model, texture mapping is often applied, which is usually obtained from aerial photography. This results in a texture image where only $5\%$ of the area is effectively textured, and the remaining $95\%$ is invalid or redundant. According to the texture mapping mechanism of OpenGL, all image files are preloaded, and then the effective area is determined based on the texture UV coordinates and the corresponding 3D coordinates of the model, and finally texture mapping is performed. However, redundant textures being loaded in this process cause excessive memory usage, making it necessary to remove them. This can not only reduce the weight of the model, but also reduce memory consumption during rendering.

In most cases, texture coordinates within the $[0, 1]$ range are required for most texture mappings. However, due to the lack of a unified standard in the field and other factors, some texture coordinates may still exceed this range. Such textures are referred to as anomalous textures in this paper, and textures within anomalous textures that are heavily repeated are referred to as repetitive textures. Textures can also be classified based on the number of times they are bound. When a single texture image has a binding count of 1, it is referred to as a standalone texture. Therefore, textures can be classified into repeated textures, shared textures, and standalone textures based on texture coordinates and binding count.

According to the binding frequency of the texture, it can be divided into two types of textures. For individual textures, they can be updated directly, while for shared textures, an integer offset is applied first, followed by coordinate union calculations, and finally texture updates, in order to achieve the goal of redundant texture elimination.

### *E. Texture Compression Based on Wavelet Transform*

This section focuses on the compression of texture images for the facade information of building models, without considering the internal structure of the building model or the texture information corresponding to the facade. The wavelet method is very suitable for image compression, especially for building facades. This is because the distribution structure of texture on building facade models includes two structures: horizontal and vertical, which are very consistent with the two directions of wavelet transform. This makes it easy to reconstruct textures of different resolutions based on wavelet-based texture image compression [22].


Fig. 7: Comparison between the original image and the image compressed by Haar wavelet

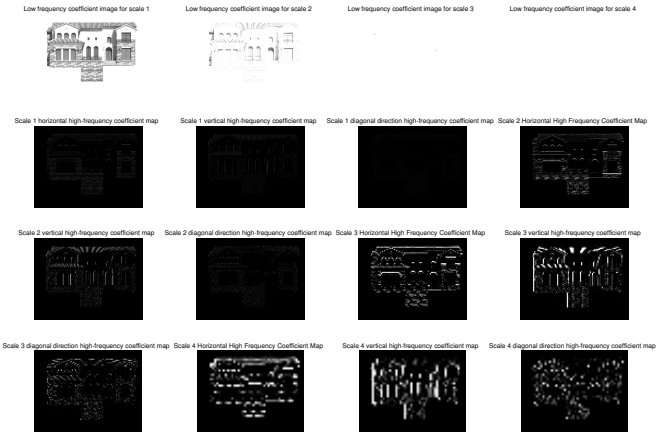



Fig. 8: Detailed diagram of Haar wavelet processing process

The Haar wavelet transform used in this paper consists of a row transform and a column transform. Let $\mathrm{A} = [a_{ij}]$ be a matrix, where $\mathrm{i} \in 1 \ldots \mathrm{n}, \mathrm{j} \in 1 \ldots \mathrm{m}$. The row transform is defined by equation 3, and the column transform is defined by equation 4:

$$H_{\text{row}}(A) = [t_{ij}] \tag{3}$$

$$H_{col}(A) = [t_{ij}] \tag{4}$$

where, when $j \leq m/2$, $t_{ij} = \left(a_{ij} + a_{i(j+m/2)}\right)/2$, and when $\mathrm{j} > \mathrm{m}/2$, $t_{ij} = \left(a_{ij} - a_{i(j-m/2)}\right)/2$. Similarly, when $\mathrm{j} \leq \mathrm{n}/2$, $t_{ij} = \left(a_{ij} + a_{i(j+n/2)}\right)/2$, and when $\mathrm{j} > \mathrm{n}/2$, $t_{ij} = \left(a_{ij} - a_{i(j-n/2)}\right)/2$.

To validate the effectiveness of the Haar wavelet transform on image compression, the front view of a certain 3D building model was used as an example. The model was subjected to four successive transformations by Haar wavelet transform in both the row and column directions. The resulting images after each compression are shown in Fig. 7. The resulting low-frequency coefficient images and the high-frequency coefficient images in the horizontal, vertical, and diagonal directions are shown in Fig. 8.

In Fig. 8, it can be seen that with each wavelet transform, the image is compressed, resulting in images with different resolutions. Moreover, with each successive compression, the volume of the image is also reduced, as shown in Table II with specific data.

TABLE II: The table of image volume data after Haar wavelet processing

| Image Category | Image Resolution | Image Size |
|---|---|---|
| Original image | $600 * 800$ | 3750 |
| Image after first compression | $300 * 400$ | 937.5 |
| Image after second compression | $150 * 200$ | 234.375 |
| Image after third compression | $75 * 100$ | 58.59375 |
| Image after fourth compression | $38 * 50$ | 14.84375 |


Fig. 9: Simplified LOD models matching geometry with textures

### *F. Constructing LOD Models That Match Geometry With Textures*

In UV mapping, each vertex has a corresponding coordinate in the UV coordinate system, which is used to represent the position of the vertex in the texture image. $U$ and $V$ respectively represent the horizontal and vertical coordinates of the texture.

When constructing a LOD model that matches geometry and texture, the texture resolution level corresponding to each LOD model should be determined first. Then, based on the resolution level corresponding to each LOD model, the texture sub-image to be used should be determined. Typically, lower resolution LOD models use lower resolution texture sub-images, while higher resolution LOD models use higher resolution texture sub-images.

By using the texture coordinate mapping method to perform level-by-level mapping, LOD models that match the geometry and texture can be constructed, as shown in Fig. 9. Models at different LOD levels are mapped to textures with different resolutions.

## IV. Optimization Method for 3D Model Rendering

### *A. A Depth Clipping Algorithm Based on Octree and Thread Separation*

Due to the substantial data volume in large-scale 3D architectural scenes, certain situations may lead to issues such as object clipping and fragmentation. Therefore, further optimization of the existing view frustum culling algorithm is necessary. The core idea is to enhance the effectiveness of polygon clipping during the view frustum culling process by performing depth detection and calculation on polygons. This approach reduces ineffective rendering and improves overall performance. The specific implementation steps are as follows:

1) In order to render objects effectively, it is necessary to transform them into the view frustum coordinate system. This process involves coordinate transformation, specifically converting the vertex coordinates of objects from the world coordinate system to the view frustum coordinate

system. In other words, it entails transforming the eight vertices of the Axis-Aligned Bounding Box (AABB) from the world coordinate system to the observation space.

2) The collision test between the AABB in the world coordinate system of the rendered model and the view frustum determines whether the model intersects with the view frustum. If all the vertices of a polygon are outside the view frustum, the polygon is completely clipped and no further steps are taken. If some vertices of the polygon are within the view frustum, a depth clipping process is required. By calculating the intersections of all edges with the view frustum, the polygon is clipped into two or more new polygons. One of these polygons lies inside the view frustum, while the remaining polygons are outside the view frustum.
3) The determination of whether the current vertex is depth clipped involves comparing the z-values of the projected point on the near clipping plane and the projected point on the far clipping plane. If the z-value of the vertex is less than the z-value on the near clipping plane or greater than the z-value on the far clipping plane, it is considered to be depth clipped.
4) After the polygon has been clipped, it undergoes another intersection check with the view frustum. If the clipped polygon is entirely outside the view frustum, it is completely discarded. However, if the clipped polygon partially resides within the view frustum, it is rendered accordingly.
5) After the rendering process is completed, it is necessary to transform the rendered results from the view frustum coordinate system back to the world coordinate system for display and subsequent processing purposes.

Although the aforementioned algorithms reduce the data volume entering the rendering pipeline, they also introduce additional computational overhead, resulting in suboptimal improvement in rendering frame rate. To optimize this process, we introduce the concept of thread separation based on the aforementioned improved algorithms. By leveraging Web Worker technology, the computational cost of the aforementioned algorithms is offloaded to threads outside the main thread. The JavaScript asynchronous mechanism is utilized for event loop processing, avoiding blocking the main thread and enabling it to focus solely on rendering operations. As a result, the computation and rendering processes are parallelized without impacting the main thread, thereby enhancing rendering performance and frame rate.

### B. Adaptive Rendering Strategy

The main idea of this adaptive strategy is to utilize the concept of the strategy pattern in design patterns, combined with existing frustum culling rendering algorithms, depth clipping rendering optimization algorithms based on octrees and thread separation, LOD rendering optimization, and IndexedDB cache optimization, to construct an adaptive rendering strategy for three-dimensional scenes based on frustum culling technology. These algorithms are encapsulated into different strategy classes, with each class responsible for implementing a specific rendering strategy. During the rendering process, the appropriate rendering strategy class is dynamically selected based on the complexity of the current scene, enabling better utilization of frustum culling technology and improving the rendering efficiency on the Web platform.

TABLE III: Table of configuration information for the main foundational hardware devices

| Hardware device name | Configuration details |
|---|---|
| System | $Windows10$ |
| CPU | Intel (R) Core (TM) Intel i5-10210U |
| RAM | $16GB$ |
| SSD | $512GB$ |
| HDD | $TOSHIBAMQ01ABD1001TB$ |
| GPU | $NVIDIAGeForceMX350$ |
| Pixel | $1920 * 1080$ |

## V. EXPERIMENTS RESULTS AND DISCUSSIONS

In order to validate the effectiveness and efficiency of the proposed method in this paper, a series of experiments and evaluations were conducted, this section selects four building information models of different scales for comprehensive experimental verification.

### A. Experimental Environment

The main hardware equipment and configuration information required for the operation and testing of the system are shown in Table III.

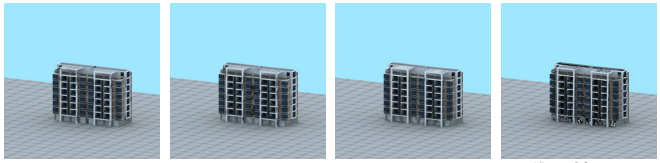
### B. Model Geometry Data Simplification Test

In this section, we tested the geometric data simplification of four models using the method proposed in this paper. Table IV shows the comparative data of the four models with different complexities before and after processing. For each model, compression was performed at $15\%$, $30\%$, $50\%$, and the table shows that the model size, number of triangles, and vertices all decreased, and the simplified models can ensure a certain display accuracy. At the same time, four different levels of models were generated, where the original model was taken as the LOD0 model, and the models simplified at $15\%$, $30\%$ and $50\%$ were respectively taken as the LOD1, LOD2, and LOD3 models in this paper.

Fig. 10 presents the comparison of the simplified model of a, and it can be observed that when simplified to $50\%$, the user's perception will be affected if viewed from a close distance. However, when displayed as an LOD model, this problem does not exist. The LOD display is shown in Fig. 11, which demonstrates that the visual effect of the model is well represented.

TABLE IV: Table comparing details of geometric data simplification before and after for the models

| Model name | Model size(MB) | Number of triangular faces in the model | Number of model vertices |
|---|---|---|---|
| Model a (Original) | 19 | 43960 | 28409 |
| Model a (simplified by 15%) | 17.8 | 36345 | 24374 |
| Model a (simplified by 30%) | 15.3 | 28657 | 20238 |
| Model a (simplified by 50%) | 13.5 | 18232 | 14813 |
| Model b (Original) | 36.8 | 199826 | 168654 |
| Model b (simplified by 15%) | 33.28 | 168852 | 143586 |
| Model b (simplified by 30%) | 28.76 | 137778 | 118458 |
| Model b (simplified by 50%) | 21.4 | 96213 | 84927 |
| Model c (Original) | 107.7 | 1613460 | 1040706 |
| Model c (simplified by 15%) | 96.55 | 1370441 | 884830 |
| Model c (simplified by 30%) | 79.39 | 1127322 | 728894 |
| Model c (simplified by 50%) | 59.85 | 803030 | 520953 |
| Model d (Original) | 201 | 8193785 | 6202213 |
| Model d (simplified by 15%) | 178.85 | 6963717 | 5272111 |
| Model d (simplified by 30%) | 148.7 | 5733549 | 4341949 |
| Model d (simplified by 50%) | 124.5 | 4093192 | 3101707 |



(a)Original model (b)Simplify 15% (c)Simplify 30% (d)Simplify 50%

Fig. 10: Comparison of simplified effects of model a

### C. Model Texture Data Simplification Test

In this section, the method proposed in this paper is used to simplify the texture data of four models. Table V shows the comparison of texture data before and after processing for models a, b, c, and d. Fig. 12 is a schematic of the multi-resolution texture generated by processing part of the texture data of model d.

### D. Model Loading Time Test

In this section, loading time tests will be conducted on four models, where the loading time refers to the time it takes for model data to be transmitted from the server to the client and successfully rendered and displayed on the browser interface. Considering that loading time may be affected by factors such as network, computing performance at a certain moment, and server status, this experiment uses the method to conduct 30 loading tests on each of the four models. Additionally, to highlight the changes in loading time, the loading time of rendering these four models directly without processing will also be recorded for 30 times. Finally, the average reduction ratio of model loading time will be used as a measure of the loading time test to further verify the performance advantages

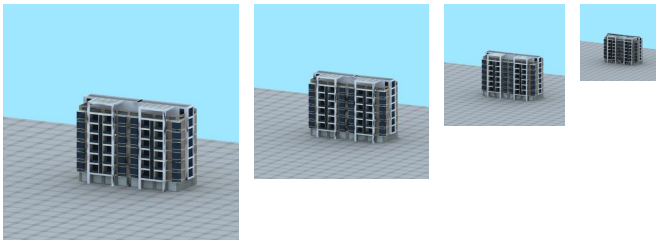


Fig. 11: Simplified LOD display of model a

TABLE V: Comparison Table of Details before and after Simplification of Model Texture Data

| Model name | Original Texture Size (MB) | first (MB) | second (MB) | third (MB) | fourth (MB) |
|---|---|---|---|---|---|
| Model a | 15.8 | 7.9 | 3.95 | 1.98 | 0.97 |
| Model b | 43.6 | 21.8 | 10.9 | 5.45 | 2.73 |
| Model c | 124.8 | 62.4 | 31.2 | 15.6 | 7.8 |
| Model d | 216.5 | 108.25 | 54.13 | 27.07 | 13.54 |

of this method. The results of the loading time reduction ratio test are shown in Fig. 13.

The results indicate that using this method for loading and rendering 3D models can reduce the average loading time to $40\%$ of the original loading time, which can effectively shorten the loading time.

### E. Model Compression Rate Test

This section will test the compression rate of the model. Firstly, lightweight tests will be conducted on each model for 30 times. Then, by comparing the data size before and after processing these models, the compression rate will be calculated, and the average compression rate will be used to measure the lightweight processing capability of this method. The specific compression results are shown in Fig. 14. As can be seen from the figure, the average value of data compression rate can reach $45\%$, which proves the lightweight processing capability of this method.

### F. Model Rendering Frame Rate Test

In this section, we will test the changes in model rendering frame rates, as frame rates usually reflect the smoothness of model loading and user experience. Therefore, we will compare and analyze the frame rate values of models directly rendered without using this method and the same models processed by this method, in order to verify the performance of this method in loading and rendering models. The frame rate values were recorded 30 times within $1.4s$ after the model was loaded. The test results are shown in Fig. 15, which displays the changes in frame rate values of the four models before and after processing.

Generally, the lowest acceptable frame rate for human eyes is above $30fps$. During usage, if the frame rate is consistently lower than $30fps$ and the frame rate fluctuates significantly, it will cause problems such as the screen not loading smoothly,

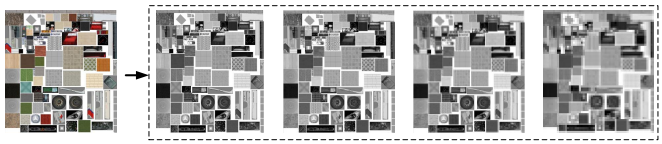


Fig. 12: Comparison of simplified effects of model a

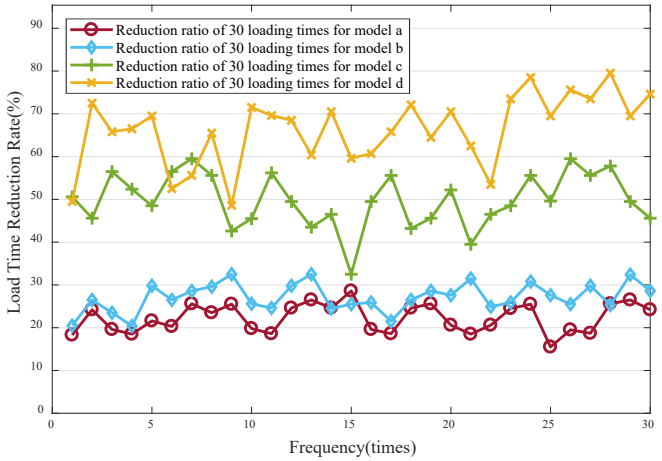



Fig. 13: Comparison of the proportion of reduced loading time for four models

seriously affecting the user's experience. As can be seen from Fig. 15, models a and b, due to their small size, can still be rendered normally even without the system's processing. However, the frame rate of both models fluctuates greatly, and after the frame rate stabilizes, the average frame rate is close to $50fps$, which will have a certain impact on the user's experience. Model c and d, due to their relatively large size, model c, which is not processed, can barely be rendered, but the frame rate is basically fluctuating around $30fps$ and is extremely unstable, resulting in screen stuttering and tearing. The unprocessed model d causes the browser to crash during rendering, seriously affecting the user's experience. After these four models were processed using this method, the rendering frame rate of models a, b, and c was improved, and the frame rate fluctuated less, and after the frame rate stabilized, the average frame rate was close to $60fps$. Compared with the other three models, the rendering frame rate of model d fluctuates greatly, but compared with the rendering without processing, its rendering frame rate is basically maintained above $30fps$, and the screen can be rendered smoothly.

### G. Model Memory Usage Test

High memory usage can lead to system crashes or slow performance on one hand, and data corruption or loss on the other, thereby impacting work efficiency and user experience. To further test the reliability and efficiency of the method, this section analyzes and compares the memory usage of the model before and after using the method. The memory usage data analyzed here is the average memory usage of the model during loading and rendering. To ensure the accuracy of the experiment, each model was tested 30 times, and the specific results are shown in Fig. 16.

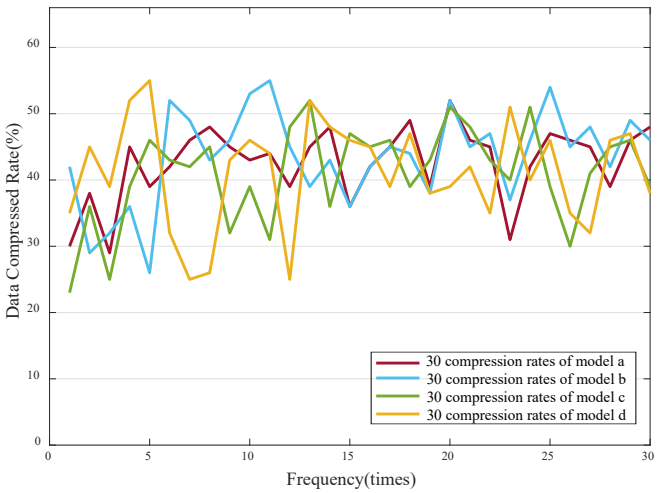



Fig. 14: Comparison of compression rates among four models

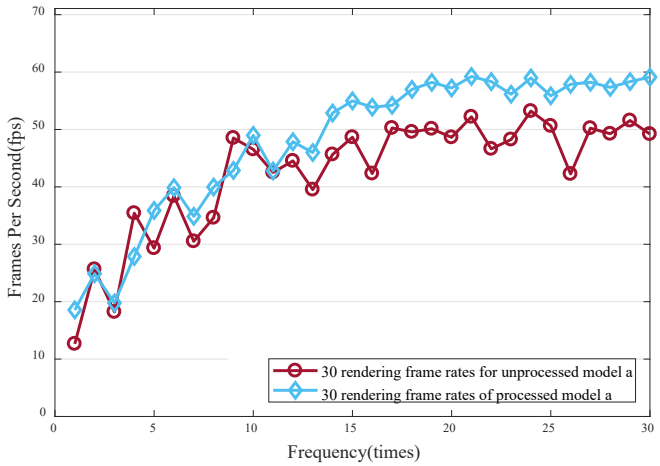
Based on the test results, the average memory usage of models a, b, c, and d before processing were $212.7MB$, $446.8MB$, $811.4MB$, and $1505.9MB$, respectively. After processing, their average memory usage reduced to $158.4MB$, $359.9MB$, $497.9MB$, and $802.7MB$, respectively, with memory usage reduction rates of $25.5\%$, $19.4\%$, $38.6\%$, and $46.7\%$. Therefore, by using this system to process the models, the memory usage can be optimized by an average of $32.55\%$.

## VI. CONCLUSION

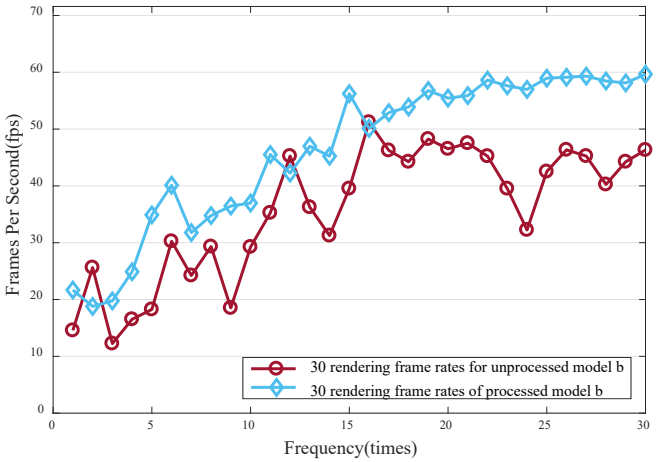
Through comprehensive experimental validation, the proposed lightweight processing method for BIM models based on geometric data and texture information, as well as the web-based 3D model rendering optimization method, have been proven effective. After applying these methods to BIM models while ensuring model display accuracy, the average data compression rate reaches $45\%$. Furthermore, extensive testing has been conducted on models of various scales, and the experimental results demonstrate that the processed models exhibit stable rendering frame rates close to $60fps$, a $40\%$ reduction in average loading time, and a $32.55\%$ decrease in average memory usage. These data validate the effectiveness and efficiency of the proposed methods, providing significant insights for research and application of web-based lightweight BIM and rendering optimization techniques, and contributing to the advancement of BIM technology.

## VII. ACKNOWLEDGMENT

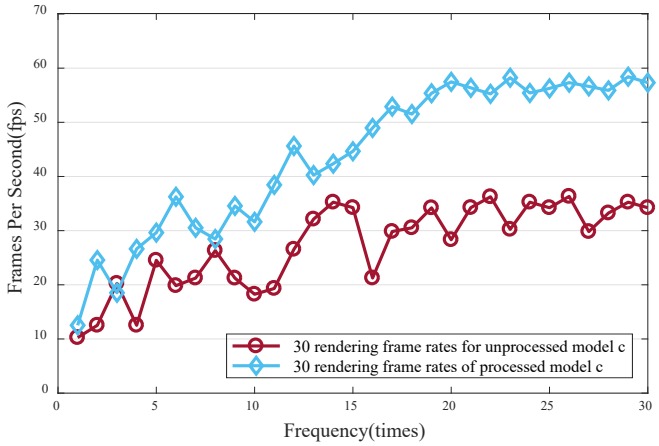
This work was supported in part by the National Natural Science Foundation of China under Grant No. 62220106004 and No. 61972308; in part by the Major Research Plan of the National Natural Science Foundation of China under Grant No. 92267204; in part by the Key Research and Development Program of Shaanxi under Grant No. 2022KXJ-093 and Grant No. 2021ZDLGY07-05, in part by the Innovation Capability Support Program of Shaanxi (Program No. 2023-CX-TD-02).



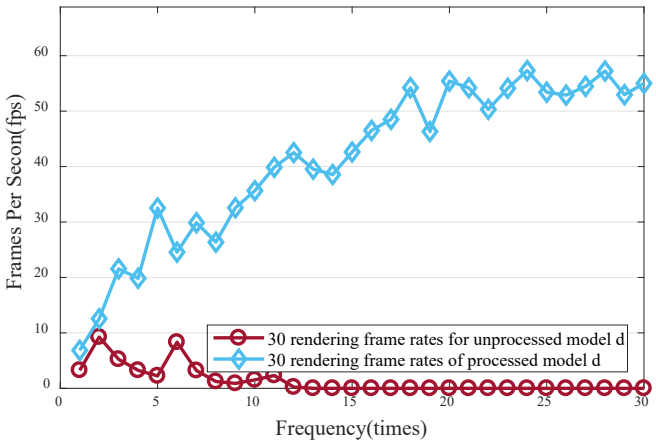

(a) 30 rendering frame rates for model a




(b) 30 rendering frame rates for model b




(c) 30 rendering frame rates for model c




(d) 30 rendering frame rates for model d

Fig. 15: Comparison of frame rate values before and after processing of four models

## REFERENCES


[1] J. Zhu and P. Wu, "Towards effective bim/gis data integration for smart city by integrating computer graphics technique," *Remote Sensing*, vol. 13, no. 10, p. 1889, 2021.

[2] J. Ma and B. Feng, "Integrated design of graduate education information system of universities in digital campus environment," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–12, 2021.

[3] Y. Q. Ang, Z. M. Berzolla, and C. F. Reinhart, "From concept to application: A review of use cases in urban building energy modeling," *Applied Energy*, vol. 279, p. 115738, 2020.

[4] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 1993, pp. 19–26.

[5] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 209–216.

[6] L. Asgharian and H. Ebrahimnezhad, "How many sample points are sufficient for 3d model surface representation and accurate mesh simplification?" *Multimedia Tools and Applications*, vol. 79, pp. 29 595–29 620, 2020.

[7] S. Li, L. Huo, T. Shen, J. Zhu, P. Li, and H. Liu, "A three-dimensional building model edge collapse simplification algorithm considering angle deviation," *Geomatics and Information Science of Wuhan University*, vol. 46, no. 8, pp. 1209–1215, 2021.

[8] X. Wu and G. Xu, "Texture image compression based on geometric information of 3d models," *Journal of Computer-Aided Design Computer Graphics*, no. 3, pp. 471–479, 2016.

[9] M. Buyukdemircioglu and S. Kocaman, "Reconstruction and efficient visualization of heterogeneous 3d city models," *Remote Sensing*, vol. 12, no. 13, p. 2128, 2020.

[10] F. S. Khan, R. M. Anwer, J. Van De Weijer, M. Felsberg, and J. Laaksonen, "Compact color–texture description for texture classification," *Pattern recognition letters*, vol. 51, pp. 16–22, 2015.

[11] 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, p. 102995, 2020.

[12] J. Zhu, P. Wu, and C. Anumba, "A semantics-based approach for simplifying ifc building models to facilitate the use of bim models in gis," *Remote Sensing*, vol. 13, no. 22, p. 4727, 2021.

[13] L. Che, F. Kang, and X. Hou, "Novel hierarchical level of detail combinatorial optimization method for large-scale 3d scene," *Journal of System Simulation*, vol. 29, no. 9, p. 2073, 2017.

[14] J. Abualdenien and A. Borrmann, "A meta-model approach for formal specification and consistent management of multi-lod building models," *Advanced Engineering Informatics*, vol. 40, pp. 135–153, 2019.

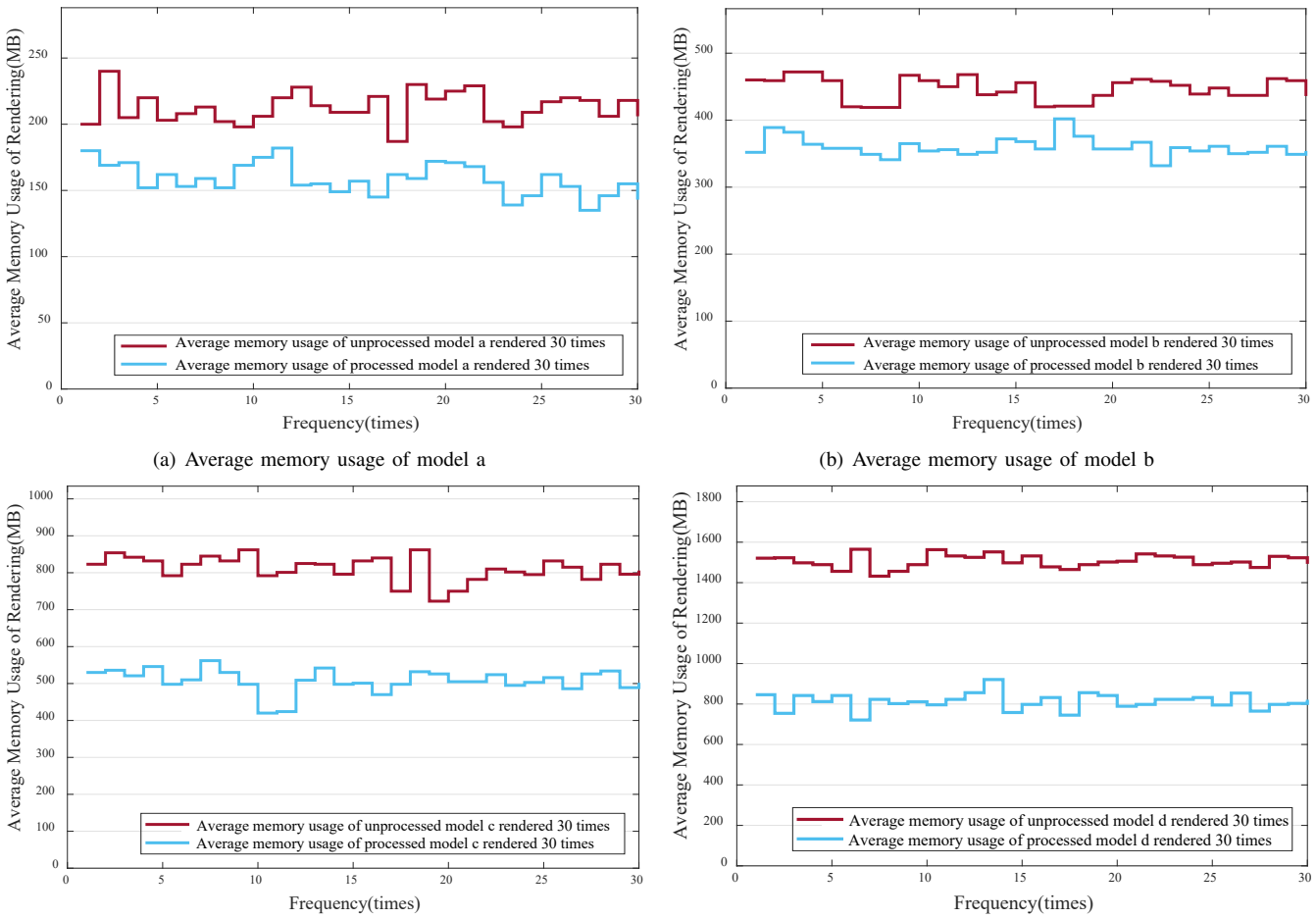
[15] J. Wang, X. Xia, Z. Zhang, Y. Zhang, Y. Shen, and B. Ren, "Class continuous lod algorithm for lightweight webgl rendering optimization," in *2022 International Conference on Networking and Network Applications (NaNA)*. IEEE, 2022, pp. 489–494.

[16] Y.-H. Zhang, C. Wen, M. Zhang, K. Xie, and J.-B. He, "Fast 3d visualization of massive geological data based on clustering index fusion," *IEEE Access*, vol. 10, pp. 28 821–28 831, 2022.

[17] D. Ströter, J. S. Mueller-Roemer, A. Stork, and D. W. Fellner, "Olbvh: octree linear bounding volume hierarchy for volumetric meshes," *The Visual Computer*, vol. 36, no. 10-12, pp. 2327–2340, 2020.

[18] X. Hui and X. Meng, "Research on collision detection of virtual scenes with intelligent algorithm of merged bounding box," *Computer Simulation*, vol. 38, no. 7, pp. 209–213, 2021.

[19] V. Croce, G. Caroti, L. De Luca, K. Jacquot, A. Piemonte, and P. Véron, "From the semantic point cloud to heritage-building information mod-




(a) Average memory usage of model a

(b) Average memory usage of model b

(c) Average memory usage of model c

(d) Average memory usage of model d

Fig. 16: Comparison of average memory usage before and after processing of four models

eling: a semiautomatic approach exploiting machine learning," *Remote Sensing*, vol. 13, no. 3, p. 461, 2021.

[20] X. Liu, C. He, C. Liu, and J. Jia, "Fast alignment of bim products based on structure matching," *Journal of System Simulation*, vol. 33, no. 7, p. 1626, 2021.

[21] J. F. Dols, J. Molina, F. J. Camacho-Torregrosa, D. Llopis-Castelló, and A. García, "Development of driving simulation scenarios based on building information modeling (bim) for road safety analysis," *Sustainability*, vol. 13, no. 4, p. 2039, 2021.

[22] J. Jam, C. Kendrick, K. Walker, V. Drouard, J. G.-S. Hsu, and M. H. Yap, "A comprehensive review of past and present image inpainting methods," *Computer vision and image understanding*, vol. 203, p. 103147, 2021.