International Journal of Geo-Information

MDPI

Article

A Precise Urban Component Management Method Based on the GeoSOT Grid Code and BIM

Huangchuang Zhang, Chengqi Cheng and Shuangxi Miao *

College of Engineering, Peking University, Beijing 100871, China; zhanghuangchuang@pku.edu.cn (H.Z.); ccq@pku.edu.cn (C.C.)

* Correspondence: miaosx@pku.edu.cn; Tel.: +86-184-8212-7707

Received: 4 March 2019; Accepted: 24 March 2019; Published: 26 March 2019

Abstract: Currently, the rapid development of cities and the rapid increase in urban populations have led to a sharp increase in urban components, making precise urban component management, query efficiency, and operational visualization urgent problems to be solved. In this paper, an in-depth study is carried out, pointing out that the current two-dimensional map or component management method based on a real-life three-dimensional city has defects, including query difficulty, fuzzy management, and inefficiency, and it is impossible to accurately and efficiently manage urban components. Then, this paper uses a combination of GIS technology and BIM technology as the starting point. On one hand, this combined technology is based on the high efficiency of the underlying data organization of the GeoSOT grid code and the accuracy of real geographic location expression; on the other hand, based on the integrity of the building information representation and the accuracy of the relative position of internal components of BIM, a precise urban component management method based on GeoSOT grid code and BIM is proposed. Finally, based on this method, a real-time 3D Earth visualization platform is established by using the Cesium platform. Taking the fire hydrant component management of the commercial Guanlan Street in Baiyin City, Gansu Province, China as an example, the precise management of the components in this area is realized, which proves that the method can achieve precise urban component management.

Keywords: urban component management; GeoSOT; GIS-BIM; 3D WebGIS; Cesium

1. Introduction

Urban components are various facilities within the urban management public area, including public facilities, road traffic, city appearance, landscaping, housing, and other municipal engineering facilities and municipal utilities. These components are an important object in urban management and are at the core of digital city management. Therefore, urban component management is the top priority of urban management. Today's cities are growing rapidly, and the size of the urban population is increasing dramatically. According to statistics from the UN, by 2030, nearly 60% of the world's population, which is nearly 5 billion people, is expected to live in urban areas [1]. In addition, worldwide, the growth rate of urban land is at least twice that of the urban population, and in some places, the growth rate is three or four times faster [2,3]. A recent study showed that more than half of city land on Earth will be built in the first three decades of the 21st century [4]. The rapid development of cities has led to a sharp increase in urban components, making the precise urban component management, query efficiency, and operational visualization urgent problems to be solved. This requires urban component management to change from fuzzy management to refined and efficient management, from qualitative static management to quantitative dynamic management, from lagging management to real-time management, from two-dimensional to three-dimensional scenes, and so on.

However, the current domestic part is based on the component management method on the two-dimensional map. Because the real scene cannot be realized, the visualization effect is very poor, and the position of the component cannot be accurately located, which is not suitable for the effective management of management personnel. Part of the real-time 3D city-based component management uses a comprehensive real-life 3D image to build a 3D model, but this 3D model has no structure, so it is difficult for managers to query and cannot be accurately and efficiently managed [5,6]. The accuracy and efficiency of the urban management system are steadily decreasing, which seriously affects the use of the urban management system. The reason for this decrease is not only the defects in the current urban component management methods, but also involves three main aspects. First, problems such as inaccurate and omitted early component censuses have been highlighted. Second, urban road changes, urban expansion, and many components have emerged, which result in urban components in some areas not being included in system management. Third, many cities use old topographical maps that need updating [7].

Through the analysis of the above research and applications, we find that there is an urgent need for a new method to solve the problems of management precision, query efficiency, and operation visualization in urban component management. Currently, the combination of GIS and BIM technologies has become a trend that has great demand. This combination is widely regarded as a major research direction to break through the bottlenecks in current research areas, such as urban management, environmental planning, and emergency management [8–13]. This paper also starts from this basis and proposes a new urban management method to meet these needs. Through the combination of the GeoSOT grid code and BIM technology, a real-time 3D visualization earth platform was built by using the Cesium platform to achieve refined and efficient management of urban components. Within this combined technology, GeoSOT globally divided the three-dimensional grid technology from the bottom of the data to achieve data gridding processing and packaging [14–16]; BIM technology, from the perspective of three-dimensional urban building models, achieved architectural visualization and provided a complete and actual situation construction engineering information database.

The essential difference between the method of this paper and the previous methods is that the proposed method combines the GeoSOT grid with real geographic information from the building information model. On one hand, the proposed method realizes the association between a certain part of the building model and a certain spatial area of the earth's surface; on the other hand, it realizes gridding processing of building information, thereby realizing the precise and efficient management of urban components. However, previous methods did not combine the building information model with real geographic information or gridded processing data but only with the two-dimensional platform or the three-dimensional platform for the inherent three-dimensional building model and for parsing and establishing the corresponding parts database, which resulted in poor accuracy and the inefficient management of urban components [5–7]. The innovative aspect of this proposed method is that it integrates the advantages of the multilevel spatiotemporal transformation of the GeoSOT grid and the rich geometric and semantic information of the BIM model. Multilevel meshing is used to manage a variety of urban components with different granularities, and multilevel precise management of the model data is realized from the data management level.

2. Materials and Methods

This section mainly describes, in detail, the method proposed in this paper. The basis of the precise management implementation is based on the GeoSOT global meshing technology to analyze the BIM model to obtain the coordinate information and attributes of each component and to perform gridded data processing and grid encapsulation, respectively. The GeoSOT grid code is the unique ID of each GeoSOT grid, and each GeoSOT grid is a data container that contains all aspects of all parts of the grid area. Through this method of gridding data organization, as well as the calculation of the GeoSOT global meshing grid, the floating point number calculation of the traditional global meshing grid is directly realized by an integer multiple of 2, and the grid coding is realized in degrees

and seconds. This method not only has high interchangeability and aggregation with the traditional latitude and longitude recording method but also greatly improves the efficiency of spatial relationship and position index and is beneficial for the efficient storage of geospatial data.

Figure 1 shows the general idea of the method. First, a 3D Earth platform is constructed with a 3D Earth platform that is built using the Cesium platform. The Cesium platform, which was created by AGI (Analytical Graphics, Inc., Exton, PA, USA), is a cross-platform virtual globe for dynamic spatial data visualization. It is an open-source JavaScript library for building a world-class 3D Earth platform and for providing the best performance for visualization [17–19]. Second, according to the GeoSOT global split technology, the earth space is decomposed into a multiscale nested space stereo grid system that assigns a unique split code to each stereo. Third, a three-dimensional model of urban architecture and its components through BIM technology was built, and then its file format was converted to a Cesium loadable 3D entity file format; that is, from FBX format to GLTF format. Fourth, the file is parsed to obtain the coordinates, texture, and various attribute information of each component inside the model by converting the coordinates to the geographic coordinates of the WGS84 on the corresponding three-dimensional sphere. Finally, the converted BIM 3D model file is loaded on the 3D Earth platform, and each part of the model is coded into the database through GeoSOT global splitting technology to realize fine management, efficient management, and real-time dynamic management of the components.

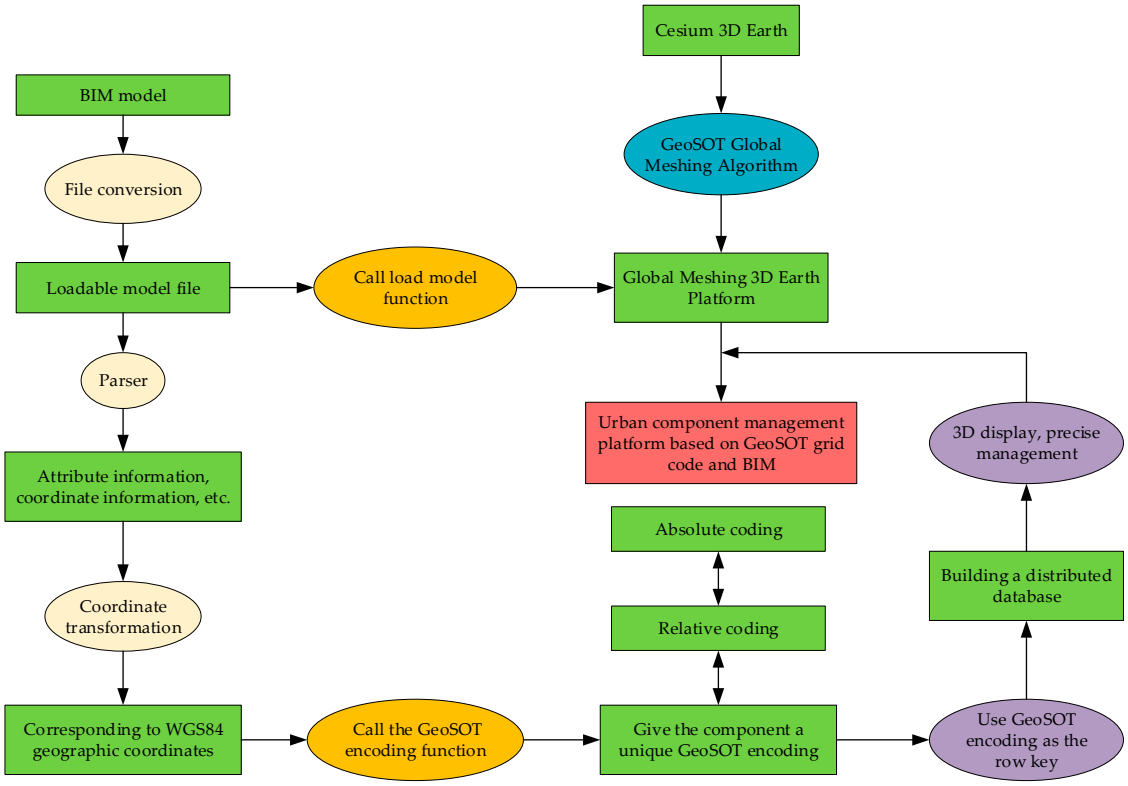


Figure 1. General idea.

In short, the method proposed in this paper has the following advantages over other existing methods, with high precision, efficient query, intuitive and easy operation, and a good visualization effect. Compared with the method based on a two-dimensional map, the proposed method realizes real-time visualization, which is convenient for nonprofessional operation and management. Compared with the real-life three-dimensional city-based method, the BIM model used in this method can realize the complete and identical actual situation construction engineering information database and solve the defect that the three-dimensional model has no structure and is difficult to query. Moreover, the GeoSOT splitting technique is used to decompose the surface of the earth into a

multiscale nested patch system, in which each patch is given a unique split code, and then a unique split code is assigned to each component to achieve precise component management.

2.1. Global Meshing

2.1.1. GeoSOT Global Grid

GeoSOT (Geographic Coordinate Subdividing Grid with One Dimension Integral Coding on 2^n Tree) was proposed by the Cheng Chengqi research team of Peking University [15]. The core idea is to expand three times, as shown in Figure 2. First, the Earth is expanded to 512° × 512°, then 1° is expanded to 64′ and, finally, 1′ is expanded to 64″, and the quadrilateral tree of completeness and division is realized to form an up to the earth (level 0), down to the cm-level bin (level 32) multiscale quad tree grid.

The GeoSOT grid consists of 32 levels. The 0-level grid is defined as the 512° square of the Earth's expansion. The corresponding area is global, and the grid is coded as G, meaning Globe. The level 1 grid is defined as an average of 4 copies on a 0-level grid, each level 1 grid size is 256°, and the grid code is Gd, where d is 0, 1, 2, or 3, and so on, to obtain a 2- to 9-level grid. Each 9-level grid is 1° in size and is encoded as Gddddddddd. Level 9 and above is the GeoSOT degree grid. The 10th to 15th levels are hierarchical grids; the 16th to 21st levels are second-order grids. The size of the 32-level grid is (1/2048), and its grid code is Gddddddddd-mmmmmm-ssssss-uuuuuuuuuu.

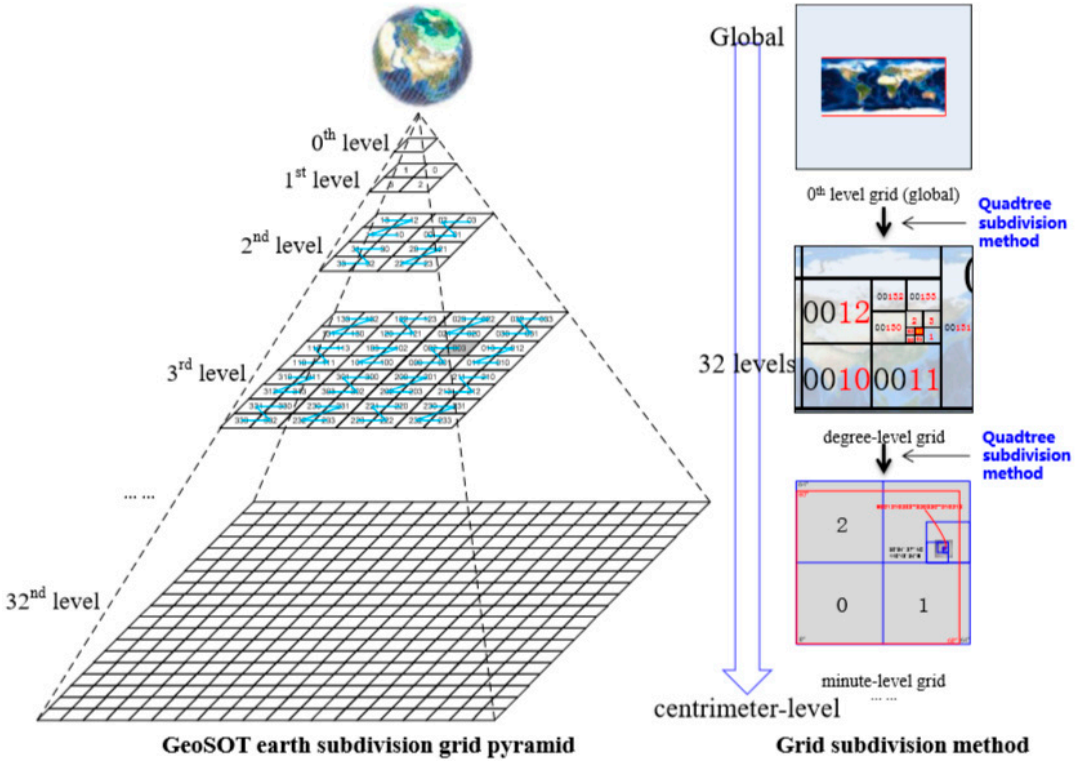


Figure 2. The binary one-dimensional encoding rule of GeoSOT [14].

2.1.2. Algorithm Implementation

GeoSOT global grid split algorithm implementation, including the encoding function, decoding function, and drawing grid function in three parts, draws the grid function and includes single grid drawing, local grid drawing, and global grid drawing. The GeoSOT global grid split algorithm flow chart is shown in Figure 3.

The encoding function is used mainly to realize the conversion of longitude, latitude, and elevation to GeoSOT encoding. Its input is four parameters of level, longitude, latitude, and elevation, and

the output corresponds to the GeoSOT code. The decoding function is used mainly to realize the conversion of GeoSOT encoding into longitude, latitude, elevation, and level. Its input is GeoSOT encoding, and the output is the corresponding level, longitude range, latitude range, and elevation range. The grid drawing function is used mainly to realize the drawing of the solid mesh. To draw a single solid mesh, just call the DrawSingleGrid function and enter the GeoSOT code. To draw a local area mesh, just call the DrawRangeGrid function, input level, longitude range, latitude range, and elevation range. When the longitude and latitude ranges are taken as the minimum and maximum, respectively, the global solid mesh is drawn. The specific code of the above function is shown in Appendix A.

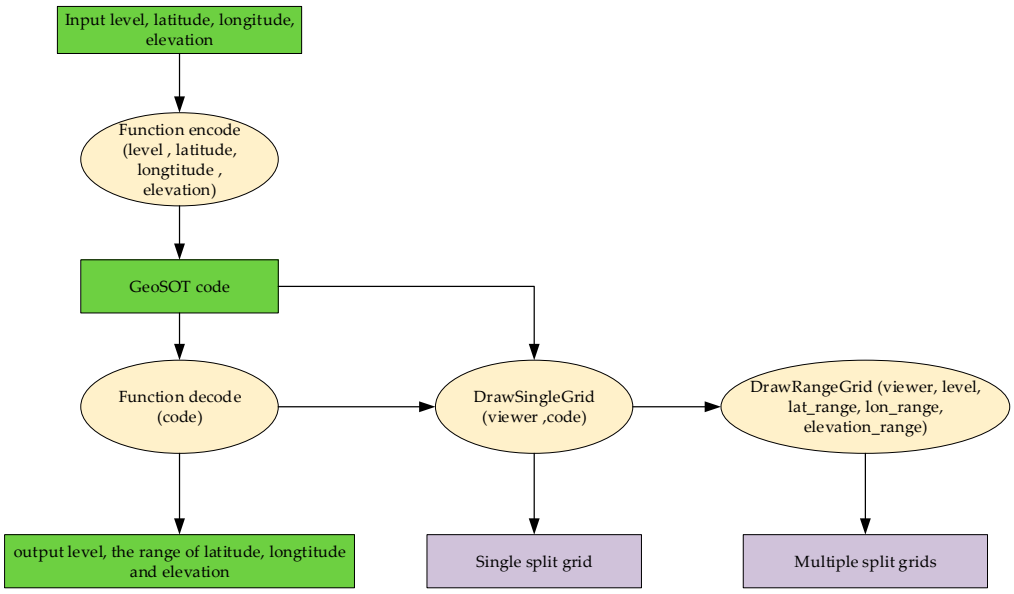


Figure 3. GeoSOT global grid split algorithm flow chart.

2.2. Building Information Modeling

Building Information Modeling (BIM) technology was first introduced by Autodesk in 2002 and is now widely recognized in the industry globally. This technology can help to achieve the integration of building information; from the design, construction, and operation of the building to the end of the life cycle of the building, all kinds of information is always integrated into a three-dimensional model information database [20–25]. The core is to provide a complete and consistent construction engineering information library for the model by establishing a virtual three-dimensional model of architectural engineering and using digital technology. The repository contains not only the geometric information, professional attributes, and status information that describe the building components but also the status information of the noncomponent objects. The biggest feature of the BIM 3D model is its visualization, because BIM visualization is a kind of visualization that can form interaction and feedback between components, including information other than the size, position, and color of the components, and can reflect the internal structure of the object [26–29]. The BIM model that is used in this paper is shown in Figure 4.

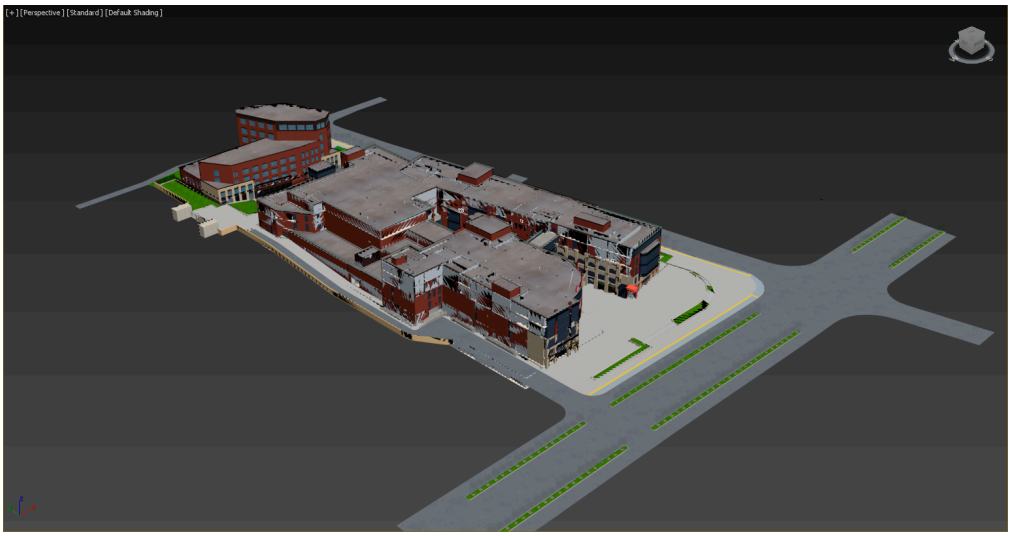


Figure 4. View the Building Information Modeling (BIM) model of the commercial Guanlan Street in Baiyin City in 3D MAX.

2.2.1. BIM Model Loading

The method proposed in this paper is a general method that is applicable to any urban building BIM model. Because we did not obtain data from Beijing, Shanghai, and Shenzhen, this article uses the BIM model of the commercial Guanlan Street in Baiyin City. The file type of this model is FBX. The FBX file is difficult to parse, and the Cesium 3D Earth platform does not support loading. Therefore, the file must be type converted and then converted into a GLTF file. The conversion is divided into three steps, the flow chart for which is shown in Figure 5.

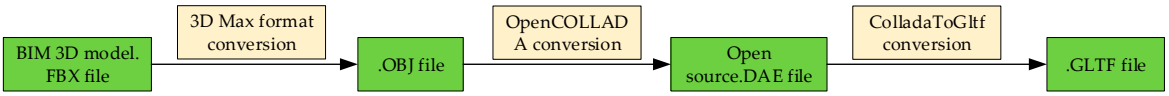


Figure 5. BIM model file conversion flow chart.

The FBX file is converted into an OBJ file by the 3D MAX software and the parameters in the OBJ export option are selected as shown in Figure 6.

1. Install the plugin OpenCOLLADA in 3D MAX software to convert OBJ files into DAE files in open-source format
2. To convert DAE files into GLTF files that are easy to parse and load, we need to use the open-source pipeline tool developed by Khronos Group, which can be directly converted into GLTF files. The tool can be downloaded directly from GitHub at https://github.com/KhronosGroup/COLLADA2GLTF. Enter the command mode under Windows and enter the file where COLLADA2GLTF-bin.exe is located, then enter the following command to convert, -f DAE model path -e, you can get the GLTF file in the target folder.

After successfully converting the GLTF file, it can be loaded on the Cesium platform. There are two ways to load a 3D model in Cesium. One is to add the entity method through entity addition, and the other is to add the primitive method through the prototype. The results obtained by these two methods are basically the same. The method chosen in this paper is to add the model through the prototype. Figure 7 shows the flow chart of the BIM model data loading process.

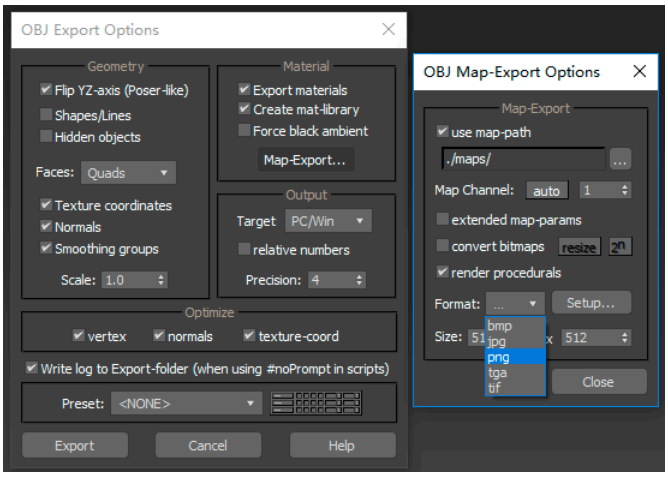


Figure 6. The parameters in the OBJ export option.

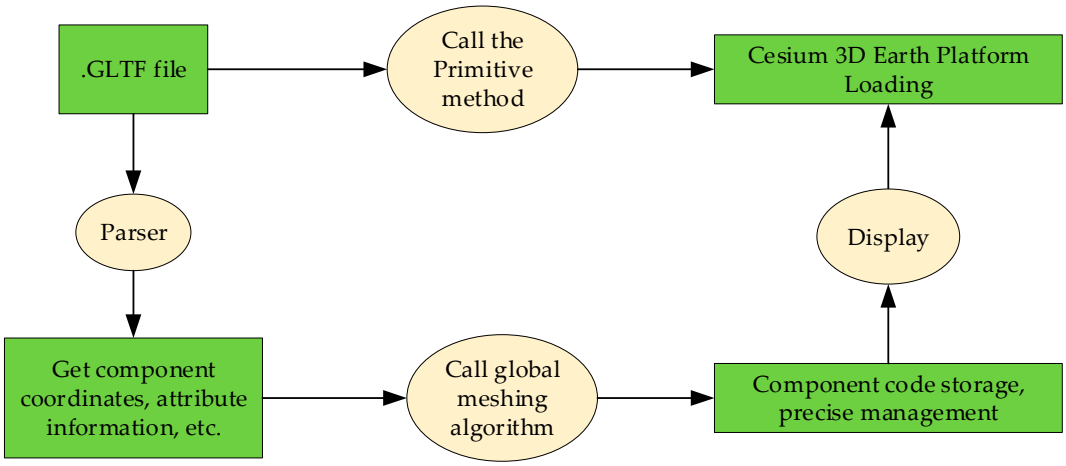


Figure 7. BIM model data loading and processing flow chart.

2.2.2. BIM Model Parsing

Model parsing is used to parse the converted GLTF file. The GLTF file is mainly composed of four parts [30,31], as shown in Figure 8. The first part is a JSON file, which mainly stores the node hierarchy, material, camera, lighting, etc. of the model. The JSON file is the core of the GLTF model. It has two main functions: First, it is equivalent to the catalog of the entire GLTF model. By looking up a specific name in the JSON file, the application can read its corresponding content and then obtain the corresponding data in the other three parts of the file according to the information that is provided by the content. Second, the JSON file also has environment and scene settings. Second, the JSON file also has environment and scene settings. The JSON file contains parameter information such as the camera position, lighting, and material that are used for WebGL graphics drawing. The application program sets the environment and scene parameters of the model by reading the information. The second part is a binary file, which is used to store the graphic data of the model. The data content, such as vertex coordinates, texture coordinates, index, and animation, was mainly used to establish the data buffer of WebGL graphics rendering. The third part is some image files such as PNG, JPG, and other formats that were used to texture map the model. The fourth part is the shader file, which is mainly the vertex shader and fragment shader that are required for graphics rendering.

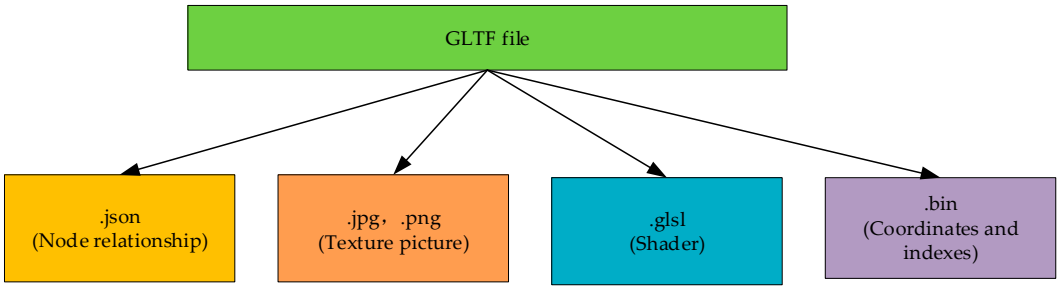


Figure 8. GLTF file composition.

Parsing the GLTF file, that is, parsing the JSON file, which contains the main information of the file, as shown in Figure 9. We can clearly see the information that is contained in the JSON file, such as accessors, images, nodes, and so on.

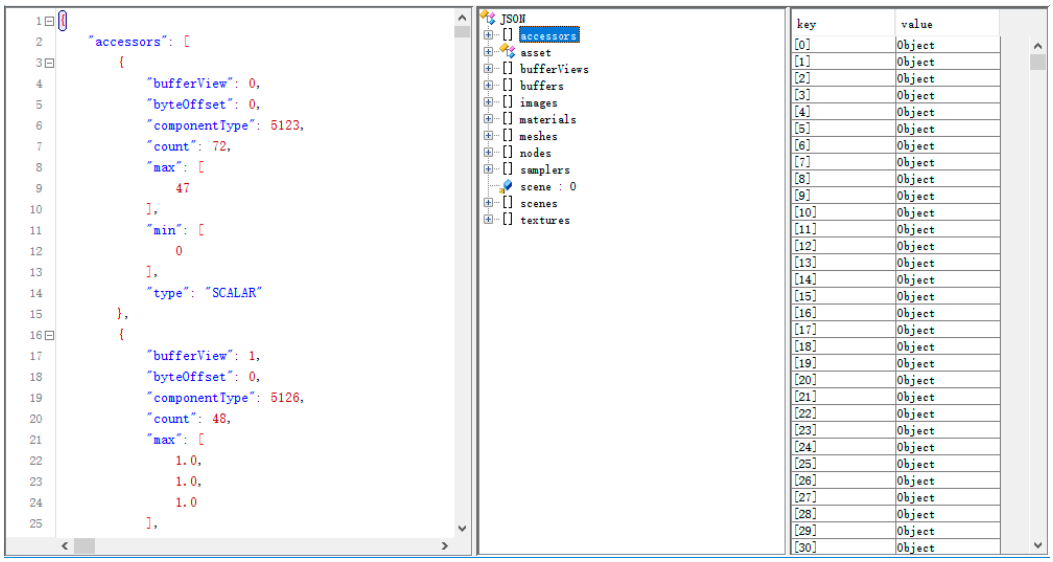


Figure 9. JSON file information.

JSON is the native format of JS. No special APIs or toolkits are required to process JSON data in JS. glTF parsing uses JS's native function JSON.parser to convert a JSON-formatted string file into a JSON object. Then, from the JSON object, the vertex coordinates, texture coordinates, index and other graphic information, camera, animation and other scene information, and the corresponding shader and texture image information of the model, and so on, are read.

After the above function is obtained, the create function first reads the corresponding vertex coordinates, texture coordinates, indexes, and other specific data from the binary file and creates a corresponding buffer, for which the coordinate analysis information is shown in Figure 10. Then, vertex shaders and fragment shaders are created, and WebGL functions are used to set various environment and material properties. Finally, the generated WebGL rendering information is transmitted to the visualization module. The analysis flow chart is shown in Figure 11.

Component	Center coordinates
XFXJK151	[-63.4733829498291,118.94406127929688,13.776638507843018]
XFXJK150	[-63.481422424316406,119.00621795654297,13.930339336395264]
XFXJK171	[-111.93613815307617,97.53602981567383,13.77664566040039]
XFXJK170	[-111.99944305419922,97.53601837158203,13.930352687835693]
XFXJK173	[-111.93613815307617,103.05786132812 5,13.77664566040039]
XFXJK172	[-111.99944305419922,103.0578498840332,13.930352687835693]
XFXJK175	[-98.94279861450195,112.77212905883789,13.930352687835693]
XFXJK174	[-99.00611877441406,112.77212905883789,13.77664566040039]
XFXJK169	[-71.25835418701172,97.84099578857422,13.776640892028809]
XFXJK168	[-71.31718063354492,97.82063674926758,13.93034553527832]
XFXJK177	[-98.94279861450195,120.96413803100586,13.930352687835693]
XFXJK176	[-99.00611877441406,120.9641227722168,13.77664566040039]
XFXJK167	[-87.87470626831055,71.05411148071289,13.77664566040039]
XFXJK166	[-87.87470245361328,70.99078750610352,13.930352687835693]
XFXJK179	[-98.94279861450195,129.41211700439453,13.930352687835693]
XFXJK178	[-99.00611877441406,129.4121246337891,13.77664566040039]
XFXJK165	[-117.73811340332031,72.20303344726562,13.77664566040039]
XFXJK164	[-117.67478942871094,72.20303344726562,13.930352687835693]
XFXJK163	[-107.58899688720703,37.25212478637695,13.77664566040039]

Figure 10. Parsed component coordinate information.

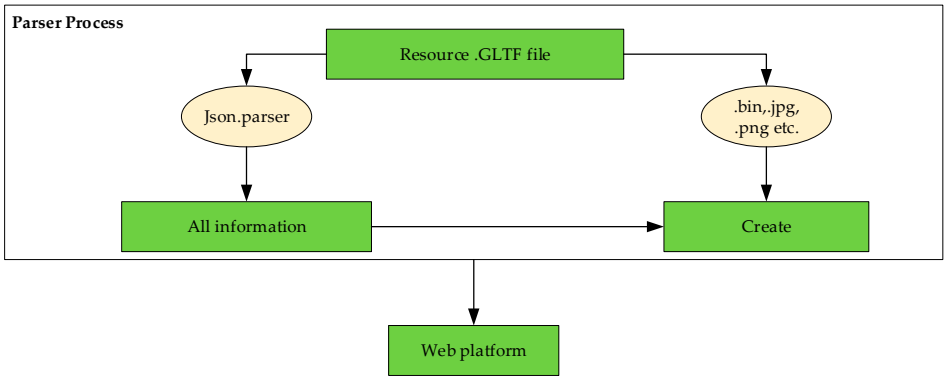


Figure 11. GLTF file parsing process.

2.2.3. Coordinate Transformation

In Cesium, there are two commonly used coordinate systems: the WGS84 geographic coordinate system and the Cartesian space Cartesian coordinate system. Through the model analysis, the various attribute information of the building is obtained, including the coordinates, attributes, etc. of each component. However, the coordinates of the resolved components are only the offset from the center point coordinates of the model and are derived from the spatial Cartesian coordinate system inside the model. The model is loaded on the 3D sphere by using the WGS84 geographic coordinate system, which requires coordinate transformation. The steps are as follows.

1. Since the coordinates of the center point are set by us, it is assumed that the coordinates of the center point coordinates on the three-dimensional sphere are in the WGS84 geographic coordinate system longitude, latitude, and elevation are lng, lat, and alt, respectively.
2. Find the (x, y, z) corresponding to the Cartesian space Cartesian coordinate system on the three-dimensional sphere.
3. According to the parsed coordinate data, these coordinates are the offsets from the coordinates of the center point, assuming (X, Y, Z), so (x + X, y + Y, z + Z) is the Cartesian space right angle coordinates in the coordinate system on the three-dimensional sphere.
4. Convert all calculated (x + X, y + Y, z + Z) to the corresponding (lng, lat, alt) in the WGS84 geographic coordinate system, which is the target coordinate that is needed.

In the Cesium platform, cesium.js provides these coordinate transformations. There are two ways to convert the WGS84 geographic coordinate system into a Cartesian space Cartesian coordinate system.

The first one is indirect conversion. Since there is no specific latitude and longitude object in Cesuim, to obtain the latitude and longitude, you first need to calculate it as radians and then convert it. The code is as follows.

var ellipsoid = viewer.scene.globe.ellipsoid (1)

var coord_wgs84 = Cesium.Cartographic.fromDegrees(lng, lat, alt) (2)

var coord_xyz = ellipsoid.cartographicToCartesian(coord_wgs84) (3)

The second way is direct conversion, cesium.js provides the corresponding conversion function Cesium.Cartesian3.fromDegrees (longitude, latitude, height, ellipsoid, and result), you can directly call the function.

For the Cartesian space, input the Cartesian coordinate system into the WGS84 geographic coordinate system, that is, the inverse operation of the WGS84 geographic coordinate system into the Cartesian space Cartesian coordinate system. The code is as follows.

var ellipsoid = viewer.scene.globe.ellipsoid (4)

var cartesian3 = new Cesium.cartesian3(x, y, z) (5)

var wgs84 = ellipsoid.cartesianToCartographic(cartesian3) (6)

var lat = Cesium.Math.toDegrees(wgs84.latitude) (7)

var lng = Cesium.Math.toDegrees(wgs84.longitude) (8)

var alt = wgs84.height (9)

3. Results

The experimental results of the precise urban component management method based on the GeoSOT grid code and BIM include the following aspects.

First, GeoSOT global split stereo mesh implementation, including global solid mesh rendering implementation, arbitrary hierarchical latitude and longitude elevation single solid mesh rendering implementation, and local area solid mesh rendering, as shown in Figure 12.

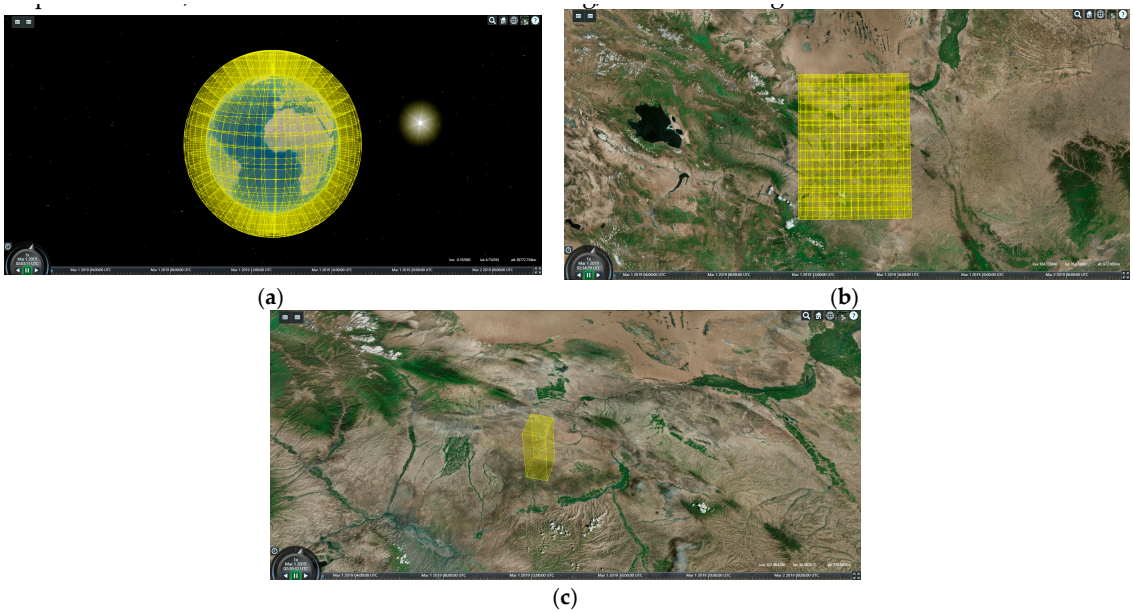
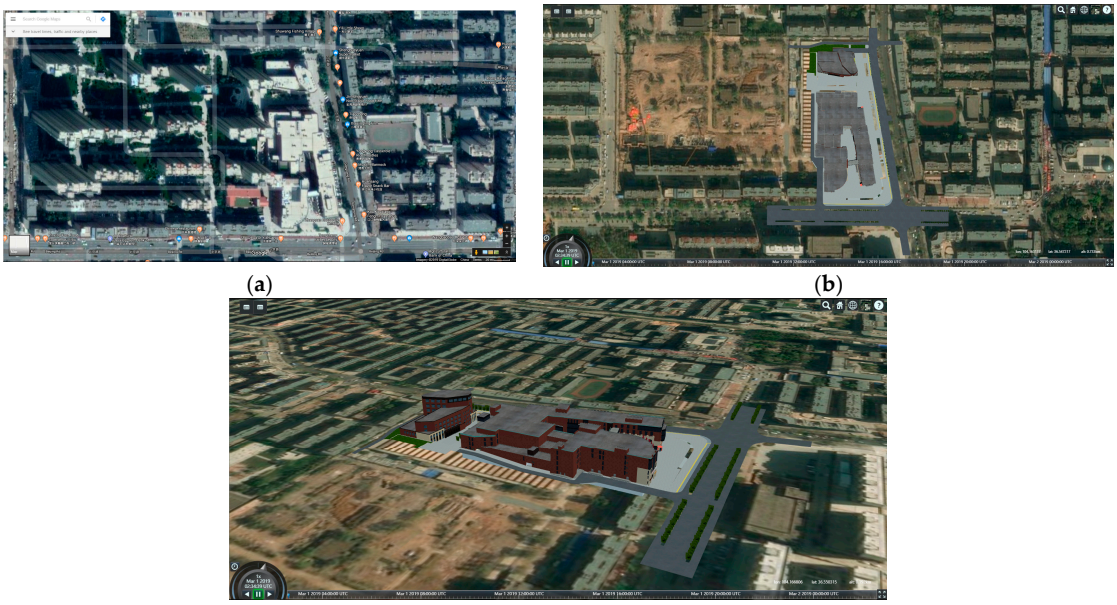


Figure 12. This is the GeoSOT global split stereo grid implementation. (**a**) 6-level GeoSOT global split stereo grid. (**b**) Local area GeoSOT globally split stereo grid. The level, latitude, and longitude parameters are (12, [35.5555, 37.5860], [103.1827, 105.2057], [0,50]). (**c**) Any level, latitude, longitude, and elevation, a single GeoSOT global split stereo grid. The level, latitude, longitude and elevation parameters are (12, 36.546219, 104.164491, 20).

Second, the BIM model is loaded and coded. The model file is loaded, and the BIM model is loaded according to the physical 1:1 scale and is in the real geographical position of the building (Baiyu City Commercial Street), as shown in Figure 13.



(**a**) (**b**) (**c**)

Figure 13. This is the BIM model loading display of the commercial Guanlan Street in Baiyin City. (**a**) Google map search real location. (**b**) Viewing the model from a vertical perspective. (**c**) View of the model from a side perspective.

As already mentioned, through the BIM model analysis, the coordinate information of each component of the building is obtained, and then the coordinates of WGS84 of each component on the three-dimensional sphere are obtained by Cesium coordinate transformation. At this time, we directly call the GeoSOT grid drawing function, determine the level, and input the latitude, longitude, and elevation to achieve the coding of the model parts. To facilitate the coding of components, we divided the model externally and internally, as shown in Figure 14a. According to the size of the original proportion of the fire hydrant, the 25-level three-dimensional grid size is suitable for fire hydrant parts. The hydrant parts are coded as shown in Figure 14b.

(**a**) (**b**)

Figure 14. (**a**) Segmented model. The position of the internal hydrant parts can be clearly seen. (**b**) Fire hydrant parts coding. Three-dimensional grids of different colors represent the hydrant parts on different floors.

Excluding the coding of the model fire hydrant parts, we also partitioned the model as a whole and designed a set of semantics from absolute coding (GeoSOT grid code) to relative coding (significant reduction of coding length) to the position information. The coding system of the code can greatly

assist nonprofessionals, or firefighting component installers can quickly find the target fire hydrant on the ground, as shown in Figure 15.

Figure 15. Model partition.

Third, a database was established to achieve precise management. After the coding of each part of the model was completed, each grid code was used as a key to establish a database to realize the two-way query and to dynamically add and delete components, to realize the refined management, efficient management, real-time dynamic management, etc. of urban components.

For example, in a precise two-way query, such as the fire protection component and the business system data interconnection, the fire protection component in the model in the platform is clicked, which can query and pop up the information on the fire protection component in the business system. In the same way, searching for a firefighting component in the business system can display the firefighting component directly in the platform, as shown in Figure 16.

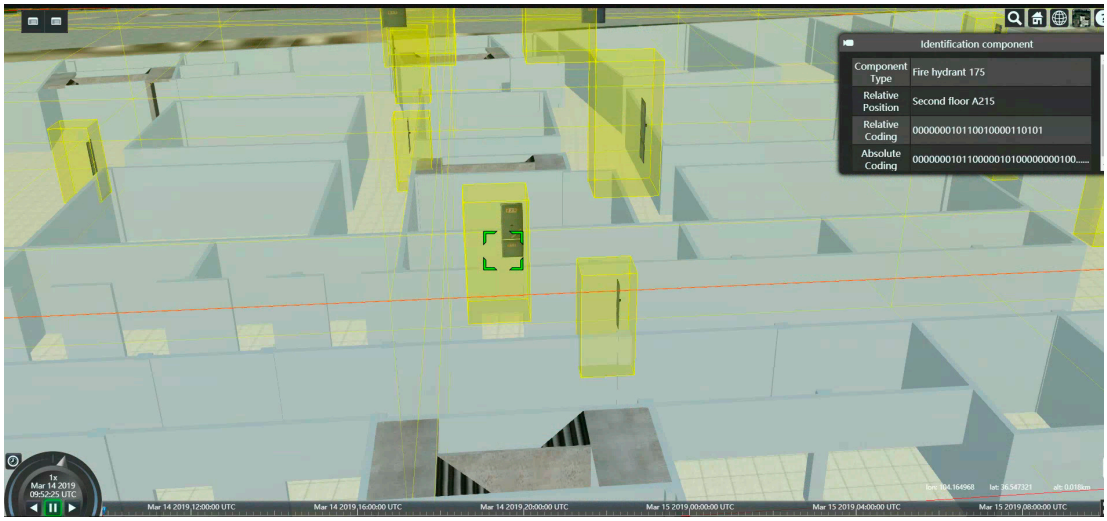


Figure 16. Fire component inquiry business system data display.

For example, to accurately add fire hydrant components, click the "Add component" button to pop up the dialog box. In the interface, as the mouse moves, the system automatically fills in the longitude, latitude, and elevation. We only need to select the parts to be added, such as the fire hydrant, then click on the location you want to add in the platform model, then click the confirm button and

the system will automatically fill in the relative code (RCode); the GeoSOT code is also the absolute code (GCode). Finally, click the Confirm Add button is clicked to complete the addition, and the corresponding information will be stored in the database and can be queried, as shown in Figure 17, in which the red grid contains the successfully added hydrant components.

Figure 17. Accurately add fire hydrant components and inquire.

4. Discussion

Through the above experimental results, we can analyze the following conclusions:

(1) From the experimental results that are shown in Figures 12–14, it can be concluded that the method does achieve the fusion of the GeoSOT grid with real geographic information and the building information model, which is not the previous method. In addition, combined with the GeoSOT grid, BIM can truly improve urban component management.

Global meshing is achieved through the GeoSOT grid, and the GeoSOT grid can be as small as a centimeter. The entire urban architecture, roads, etc. can be modeled through BIM technology, and the city is presented in its original proportion. The two are just complementary. The BIM model lacks real geolocation attributes, and the GeoSOT grid just makes up for this; the GeoSOT grid is just a way of splitting the Earth with only its geographic location and lack of city information, and the BIM model just makes up for this. The combination of the two is perfect for improving the problem of urban component management.

(2) From the experimental results shown in Figure 16 for the precise query of the fire hydrant and the precision-added fire hydrant components shown in Figure 17, it can be seen that the method can indeed achieve the precise management of urban components. Moreover, the BIM model data visualization can be realized on the 3D Earth platform, and it is more intuitive and easier to operate for urban component management, such as querying components, adding components, and so on.

Accuracy is mainly reflected in two aspects: the accuracy of the GeoSOT grid for real-world geographic location and the accuracy of the BIM model for the relative position of urban components within the building. In addition, because it is directly operated by meshing and visualizing the three-dimensional earth, by realizing the virtual reality and putting the whole building in front of us, turning urban component management, such as moving, deleting, adding component, and so on, into stacked wood can make those nonprofessionals easy to get started, thus making management more efficient.

(3) The method is highly scalable because it operates directly on 3D Earth. Through gridding, down to a city and up to a country, you can achieve unified management.

This is the natural advantage of this method. Based on the Cesium three-dimensional earth, its starting point is to start from the whole earth, from large to small. In contrast, it is easier to move from

small to large. The GeoSOT grid remains unchanged. We only need to model the scope. There are one to two cities. Similar to the fire hydrant components in this article, we can achieve unified management of fire hydrant components across the country.

5. Conclusions and Directions for Future Research

Currently, cities are developing more rapidly, and the management of 3D urban components is gaining more attention. Urban components are important objects of urban management. Managing the 3D parts of the city is the key way to construct 3D digital city management. The combination of GIS and BIM will become an important trend in the development of 3D digital urban management. This paper proposes a combination of GIS and BIM technologies that uses a precise urban component management method based on the GeoSOT grid code and BIM, which effectively solves the shortcomings of the current urban component management methods that cannot achieve refined, efficient and dynamic management of urban components. In addition, the method proposed in this paper with combined GIS and BIM technologies plays a large role in the future of smart fire protection.

Future research should be as follows. The grid management system relies on the GeoSOT global dissection theory as the theoretical basis and completes the accurate identification and efficient query and retrieval of fire parts, facilities, and personnel by constructing a large distributed index table and relies on modern high-precision sensors to obtain real-time information on firefighting parts, facilities, and firefighters in firefighting scenes. Through the modeling of fire facilities operation, fire brigade and material control, risk hazard investigation, social rescue situation monitoring, and operational abnormalities, a fire disaster classification warning model should be established to realize the fire warning. Through grid-based data management and the precise location of the disaster site, the pathless navigation method should be used to establish scientific rescue of the disaster site. Through the internet of things data at the time of the disaster, combined with the gridded proximity analysis and the shortest path analysis algorithm, the grid analysis of fire protection should be completed, thereby completing the fire lifecycle management from predisaster early warning and disaster relief and disaster analysis to achieve intelligent fire protection.

Author Contributions: H.Z. conceived, designed, and performed the experiments, and wrote the manuscript; C.C. and S.M. supervised the study; S.M. offered helpful suggestions and reviewed the manuscript. All authors have read and approved the submitted manuscript, have agreed to be listed, and have accepted this version for publication.

Funding: This research was supported by the National Key Research and Development Program of China (Grant No. 2018YFB0505300, Grant No. 2017YFB0503703) and the National Natural Science Foundation of China (Grant No. 41801301).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

1. encode function specific code:

```
function encode(level, lon, lat, height) {
    let lonlatResolution = LON_LAT_TABLE[level];
    let heightResolution = HEIGHT_TABLE[level];
    let lonInt = Math.floor(Math.abs(lon)/lonlatResolution);
    let latInt = Math.floor(Math.abs(lat)/lonlatResolution);
    let heightInt = Math.floor(Math.abs(height) / heightResolution);
    console.assert(lonInt <= Math.pow(2, level- 1));
    console.assert(latInt <= Math.pow(2, level-1));
    console.assert(heightInt <= Math.pow(2, level-1));
    if (heightInt > Math.pow(2, level - 1))
        console.log(height);
    let lonBinS = lonInt.toString(2);
```

```
        let latBinS = latInt.toString(2);
        let heightBinS = heightInt.toString(2);
        console.assert(lonBinS.length <= level - 1);
        console.assert(latBinS.length <= level - 1);
        console.assert(heightBinS.length <= level - 1);
        let lonBin = Array(level-lonBinS.length).join('0') + lonBinS;
        let latBin = Array(level - latBinS.length).join('0') + latBinS;
        let heightBin = Array(level - heightBinS.length).join('0') + heightBinS;
        lonBin = lon >= 0¿1' + lonBin:'0' + lonBin;
        latBin = lat >= 0¿1' + latBin:'0' + latBin;
        heightBin = height >= 0 ? '1' + heightBin : '0' + heightBin;
        let codeArray = Array();
        console.assert(lonBin.length === latBin.length && latBin.length === heightBin.length);
        for (var i = 0; i < lonBin.length; i++) {
            codeArray.push(lonBin[i]);
            codeArray.push(latBin[i]);
            codeArray.push(heightBin[i]);
        }
        return codeArray.join('');
    }
```

2. decode function specific code:

```
    function decode(code) {
        console.assert(code.length % 3 === 0 && code.length > 3);
        let level = code.length / 3;
        let lonArray = Array();
        let latArray = Array();
        let heightArray = Array();
        for (var i = 3; i < code.length; i++) {
            if (i % 3 === 0) {
                lonArray.push(code[i]);
            }
            else if (i % 3 === 1) {
                latArray.push(code[i]);
            }
            else {
                heightArray.push(code[i]);
            }
        }
        let lonString = lonArray.join('');
        let latString = latArray.join('');
        let heightString = heightArray.join('');
        let lonInt = parseInt(lonString, 2);
        let latInt = parseInt(latString, 2);
        let heightInt = parseInt(heightString, 2);
        let lonlatResolution = LON_LAT_TABLE[level];
        let heightResolution = HEIGHT_TABLE[level];
        let lonMin = lonInt * lonlatResolution;
        let latMin = latInt * lonlatResolution;
        let heightMin = heightInt * heightResolution;
        lonMin = code[0] === '1' ? lonMin : -lonMin;
```

```
        latMin = code[1] === '1' ? latMin : -latMin;
        heightMin = code[2] === '1' ? heightMin : -heightMin;
        return {
            level: level,
            min_coord: [lonMin, latMin, heightMin],
            max_coord: [lonMin + lonlatResolution, latMin + lonlatResolution, heightMin
+ heightResolution]
        }
    }
```

3. DrawSingleGrid function specific code:

```
    function DrawSingleGrid (viewer, code, style) {
        let codeRange = decode(code);
        let level = codeRange.level;
        console.log(codeRange);
        if (codeRange.min_coord[0] >= 180 || codeRange.max_coord[0] <= -180 ||
            codeRange.min_coord[1] >= 90 || codeRange.max_coord[1] <= -90)
            return;
        let lon_min = codeRange.min_coord[0] < -180?-179.999 : codeRange.min_coord[0];
        let lat_min = codeRange.min_coord[1] < -90?-89.999 : codeRange.min_coord[1];
        let lon_max = codeRange.max_coord[0] > 180 ? 179.999 : codeRange.max_coord[0];
        let lat_max = codeRange.max_coord[1] > 90 ? 89.999 : codeRange.max_coord[1];
        console.log(lon_min, lat_min, lon_max, lat_max);
        viewer.entities.add({
            name:
            code,
            rectangle: {
                coordinates: Cesium.Rectangle.fromDegrees(lon_min, lat_min, lon_max, lat_max),
                material: Cesium.Color.WHITE.withAlpha(0),
                extrudedHeight: HEIGHT_TABLE[level],
                height: codeRange.min_coord[2],
                outline: true,
                outlineColor: Cesium.Color.YELLOW
            }
        });
    }
```

4. DrawRangeGrid function specific code:

```
    function DrawRangeGrid (viewer, level, lon_range, lat_range, height_range) {
        let lonlatResolution = LON_LAT_TABLE[level];
        let heightResolution = HEIGHT_TABLE[level];
        let lon_min = lon_range ? lon_range[0] : -180;
        let lon_max = lon_range ? lon_range[1] : 180;
        let lat_min = lat_range ? lat_range[0] : -90;
        let lat_max = lat_range ? lat_range[1] : 90;
        let height_min = height_range ? height_range[0] : 0;
        let height_max = height_range ? height_range[1] : Math.pow(2, 16);
        console.log("long lat resolution", lonlatResolution);
        console.log("height resolution", heightResolution);
        for (var lon = lon_min; lon <= lon_max - lonlatResolution; lon += lonlatResolution) {
            for (var lat = lat_min; lat <= lat_max - lonlatResolution; lat += lonlatResolution) {
                for (var height = height_min; height <= height_max; height += heightResolution) {
```

```
                    viewer.entities.add({
                        name:
                            encode(level,
                                lon + lonlatResolution / 2,
                                lat + lonlatResolution / 2,
                                height + heightResolution / 2
                            ),
                        rectangle: {
                            coordinates: Cesium.Rectangle.fromDegrees(lon, lat, lon + lonlatResolution, lat + lonlatResolution),
                            material: Cesium.Color.WHITE.withAlpha(0.01),
                            extrudedHeight: heightResolution,
                            height: height,
                            outline: true,
                            outlineColor: Cesium.Color.YELLOW
                        }
                    });
                }
            }
        }
    }
```

References

1. UNDESA. World Urbanization Prospects, the 2011 Revision: Highlights. 2012. Available online: http://www.un.org/en/development/desa/publications/world-urbanization-prospects-the-2011-revision.html (accessed on 26 March 2019).
2. Angel, S.; Parent, J.; Civco, D.L.; Blei, A. The dimensions of global urban expansion: Estimates and projections for all countries, 2000–2050. *Prog. Plan.* **2011**, *75*, 53–107. [CrossRef]
3. Seto, K.C.; Fragkias, M.; Güneralp, B. A meta-analysis of global urban land expansion. *PLoS ONE* **2011**, *6*, e23777. [CrossRef]
4. Seto, K.C.; Güneralp, B.; Hutyra, L.R. Global forecasts of urban expansion to 2030 and direct impacts on biodiversity and carbon pools. *Proc. Natl. Acad. Sci. USA* **2012**, *109*, 16083–16088. [CrossRef]
5. Li, W. Research on Smart City Management in Jinjiang City under the Background of Big Data. Master's Thesis, Huaqiao University, Quanzhou, China, 2018.
6. Wu, Y.; Yang, Y.C.; Gao, J. Design and implementation of digital urban component information management system. *J. Xi'an Univ. Sci. Technol.* **2013**, *33*, 319–324.
7. Yang, D.F.; Han, J.P.; Xue, T.Y. Design and Implementation of 3D Digital Urban Management Component Database. *Surv. Tech. Equip.* **2016**, *18*, 5–10.
8. Amirebrahimi, S.; Rajabifard, A.; Mendis, P.; Ngo, T. A BIM-GIS integration method in support of the assessment and 3D visualisation of flood damage to a building. *J. Spat. Sci.* **2016**, *61*, 317–350. [CrossRef]
9. Isikdag, U.; Zlatanova, S. Towards defining a framework for automatic generation of buildings in CityGML using Building Information Models. In *3D Geo-Information Sciences: Lecture Notes in Geoinformation and Cartography*; Lee, J., Zlatanova, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 79–96.
10. El-Mekawy, M.; Ostman, A. Semantic mapping: An ontology engineering method for integrating building models in IFC and CityGML. In Proceedings of the 3rd ISDE Digital Earth Summit, Nessebar, Bulgaria, 12–14 June 2010.
11. Karimi, H.A.; Akinci, B. *CAD and GIS Integration*, 1st ed.; Karimi, H.A., Akinci, B., Eds.; CRC Press: Boca Raton, FL, USA, 2010.
12. ElMeouche, R.; Rezoug, M.; Hijazi, I. Integrating and managing BIM in GIS, software review. In Proceedings of the ISPRS 8th 3DGeoInfo Conference & WG II/2 Workshop, Istanbul, Turkey, 27–29 November 2013.

13. Arroyo Ohori, K.; Biljecki, F.; Diakite, A.; Krijnen, T.; Ledoux, H.; Stoter, J. Towards an integration of gis and bim data: What are the geometric and topological issues? In Proceedings of the 12th 3D Geoinfo Conference on ISPRS Annals of the Photogrammetry Remote Sensing and Spatial Information Sciences, Melbourne, Australia, 26–27 October 2017; Volume IV-4/W5.
14. Cheng, C.Q.; Tong, X.C.; Chen, B.; Zhai, W.X. A Subdivision Method to Unify the Existing Latitude and Longitude Grids. *ISPRS Int. J. Geo-Inf.* **2016**, *5*, 161. [CrossRef]
15. Cheng, C.Q. *An Introduce to Spatial Information Subdivision Organization*; Science Press: Beijing, China, 2012.
16. Li, S.; Cheng, C.; Chen, B.; Meng, L. Integration and management of massive remote-sensing data based on GeoSOT subdivision model. *J. Appl. Remote Sens.* **2016**, *10*, 034003. [CrossRef]
17. Cesium. Available online: https://cesiumjs.org/about/ (accessed on 6 November 2018).
18. glTF. Available online: https://www.khronos.org/gltf/ (accessed on 6 November 2018).
19. Murshed, S.M.; Al-Hyari, A.M.; Wendel, J.; Ansart, L. Design and Implementation of a 4D Web Application for Analytical Visualization of Smart City Applications. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 276. [CrossRef]
20. Penttila, H. Describing the Changes in Architectural Information Technology to Understand Design Complexity and Free-form Architectural Expression. *J. Inf. Technol. Constr. (ITcon)* **2006**, *11*, 395–408.
21. Dossick, C.S.; Neff, G. Organizational divisions in BIM-enabled commercial construction. *J. Constr. Eng. Manag.* **2009**, *136*, 459–467. [CrossRef]
22. Wang, X.; Love, P.E. BIM+ AR: Onsite information sharing and communication via advanced visualization. In Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Wuhan, China, 23–25 May 2012; pp. 850–855.
23. Wang, X.; Truijens, M.; Hou, L.; Wang, Y.; Zhou, Y. Integrating augmented reality with building information modeling: Onsite construction process controlling for liquefied natural gas industry. *Autom. Constr.* **2014**, *40*, 96–105. [CrossRef]
24. Taylor, J.E.; Bernstein, P.G. Paradigm trajectories of building information modeling practice in project networks. *J. Manag. Eng.* **2009**, *25*, 69–76. [CrossRef]
25. Aouad, G.; Lee, A.; Wu, S. *Constructing the Future: Nd Modelling*; Routledge: London, UK, 2006.
26. Peters, E. BIM and geospatial information systems. In *Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies*; IGI Global: Hershey, PA, USA, 2010; pp. 483–500.
27. Mignard, C.; Nicolle, C. Merging BIM and GIS using ontologies application to urban facility management in active3D. *Comput. Ind.* **2014**, *65*, 1276–1290. [CrossRef]
28. Deng, Y.; Cheng, J.C.; Anumba, C. Mapping between BIM and 3D GIS in different levels of detail using schema mediation and instance comparison. *Autom. Constr.* **2016**, *67*, 1–21. [CrossRef]
29. Borrmann, A.; Kolbe, T.; Donaubauer, A.; Steuer, H.; Jubierre, J.; Flurl, M. Multi-scale geometric-semantic modeling of shield tunnels for GIS and BIM applications. *Comput.-Aid. Civ. Infrastruct. Eng.* **2015**, *30*, 263–281. [CrossRef]
30. GLTF—Runtime 3D Asset Delivery. Available online: https://github.com/KhronosGroup/glTF (accessed on 5 December 2018).
31. GLTF/README. Available online: https://github.com/pasu/glTF/blob/master/specification/1.0/README.md (accessed on 5 December 2018).

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).