Received: 19 July 2018 | Revised: 13 September 2018 | Accepted: 20 November 2018

DOI: 10.4218/etrij.2018-0352

ORIGINAL ARTICLE

ETRI Journal WILEY

Implementation of an open platform for 3D spatial information based on WebGL

Ahyun Lee | **Insung Jang**

Hyper-connected Communication Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea

Correspondence
Ahyun Lee, Hyper-connected Communication Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea.
Email: ahyun@etri.re.kr

Funding information
This work was supported by the Ministry of Land, Infrastructure and Transport (MOLIT), Korea, under the Urban Planning & Architecture (UPA) research support program supervised by the Korea Agency for Infrastructure Technology Advancement (KAIA) (grant 13 Urban Planning & Architecture 02). It was also supported by a grant (18DRMS–B147287–01) from the development of customized realistic three-dimensional (3D) geospatial information update and utilization technology based on consumer demand, funded by the Ministry of Land, Infrastructure and Transport of the Korean government.

VWorld is run by the Ministry of Land, Infrastructure, and Transport of South Korea and provides national spatial information, such as aerial images, digital elevation models, and 3D structural models. We propose herein an open platform for 3D spatial information based on WebGL using spatial information from VWorld. WebGL is a web-based graphics library and has the advantage of being compatible with various web browsers. Our open platform is also compatible with various web browsers. Accordingly, it is easily accessible via the VWorld site and uses the three-dimensional (3D) map program. In this study, we describe the proposed platform configuration, and the requests, management, and visualization approaches for VWorld spatial information data. Our aim is to establish an approach that will provide a stable rendering speed even on a low-end personal computer without a graphics processing unit based on a quadtree structure. We expect that users will be able to visualize 3D spatial information through the VWorld open platform, and that the proposed platform will become the basis for various applications.

KEYWORDS
3D map, geographic information system, spatial information, VWorld, WebGL

1 | INTRODUCTION

The VWorld Data Center is operated by the Ministry of Land, Infrastructure, and Transport (MOLIT) of South Korea and provides the latest, high-quality, three-dimensional (3D) national spatial information data [1,2]. It also provides a search function based on various metadata and information inquiry functions linked to two-dimensional (2D)/3D maps. The provided metadata include 3D structural models, aerial images, digital elevation models (DEMs), and administrative district information. This study describes an implementation approach of an open platform for 3D spatial information based on WebGL using VWorld data, as shown in Figure 1.

WebGL is a web-based graphical library that uses the JavaScript programming language [4–9]. WebGL uses the HTML5 canvas element based on OpenGL ES 2.0 and on a document object model interface, which is the official standard for W3C. W3C represents structured documents, which are independent of the platform or language [10]. Our WebGL-based platform can be executed on various web browsers, such as Google Chrome, Microsoft Internet Explorer 11, Edge, Mozilla Firefox, Safari, and Opera. Therefore, anyone can access the open platform's VWorld site [11] in various computing environments. In addition, the open platform is a basic platform for applications that use the national spatial information provided by MOLIT.

FIGURE 1 Open platform for three-dimensional (3D) spatial information based on VWorld data

OpenGlobus provides WebGL-based 3D maps through a web browser. However, if a request for terrain data is not completed, it is not properly rendered without texture images, as shown in Figure 2A [12]. A WebGL-based Cesium 3D map represents 2D terrain texture images on a spherical representation of the globe [13]. Therefore, elevation information of the ground is not illustrated, as shown in Figure 2B.

The most extensively used 3D map is Google Earth [14] (Figure 2C). Google Earth is based on a digital surface model (DSM), and it is thus faster than DEM-based approaches. A DSM approach uses a single-texture image that combines the terrain and structural information [15–18]. The image size is smaller than the sizes of the images used in DEM approaches. However, as shown in Figure 2D which is an enlarged image of Figure 2C, whereby the building objects do not appear clearer than their representations in the images rendered with DEM, whereby the terrain and the building images are used independently. Moreover, the building models cannot be treated as independent objects. It is thus difficult to select building objects or represent extra information associated with a building based on 3D models. By contrast, our open platform, which is based on DEM, can represent the Jeju Island with DEM, as shown in Figure 2E, unlike Figure 2C. Additionally, the selected building can be represented in a different manner, as shown in Figure 2F.

The platform of Seoul's 3D geographical information system (GIS) is similar to our platform in that it uses VWorld data, and it is a WebGL-based platform [19]. However, it deals only with the area of the city of Seoul, and outer city areas (other than inner-city areas) are highlighted in black color, as shown in Figure 2G. To accelerate the rendering of the 3D buildings, the texture images of all the buildings are merged and saved in a single file. This is advantageous because the number of requested files and the sizes of the files are small.

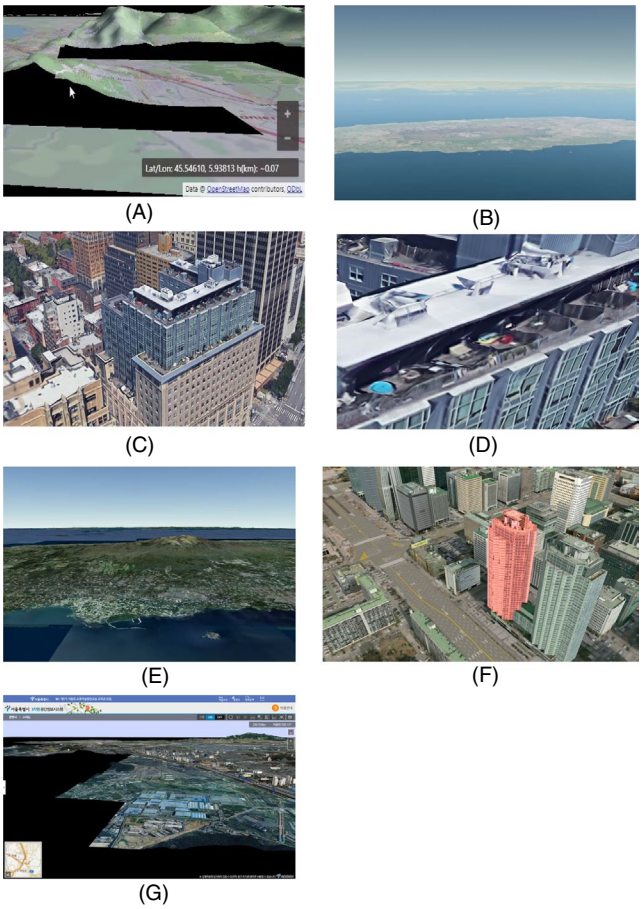


FIGURE 2 Examples of 3D map platforms: (A) OpenGlobus [12], (B) Jeju Island in South Korea illustrated using Cesium [13] without elevation data, (C) Google Earth view [14], (D) an enlarged image of (C) showing the uneven building surfaces, (E) Jeju Island illustrated with elevation data, (F) user-selected building displayed in red on our proposed platform [11], and the (G) Seoul 3D geographical information system platform [20]

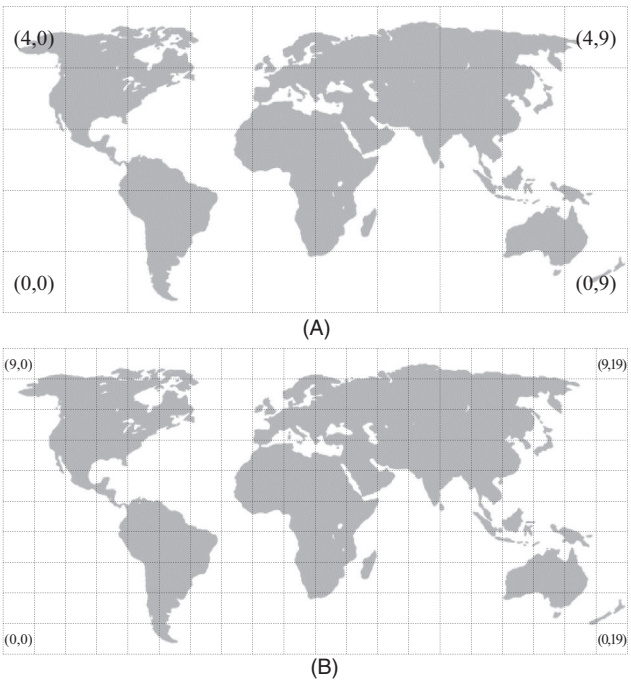


FIGURE 3 Examples of tile structures: (A) 50 tiles at level zero and (B) 200 tiles at level one

However, the resolutions and qualities are lower than those of the original files provided by the VWorld Data Center.

In this study, we describe the implementation details of the open platform for use as 3D spatial information in the VWorld Data Center. Our aim is to introduce a method that effectively renders VWorld spatial information (amounting to 30 TB or more) with a low-end personal computer (PC) and without the use of a graphics processing unit (GPU). We propose a method for managing, requesting, and rendering a large amount of VWorld spatial information data based on the quadtree structure. We hope that this study will be useful for applications that employ VWorld spatial information data based on our proposed open platform.

The rest of this study is structured as follows. In Section 2, we describe the spatial information data of VWorld. In Section 3, we describe the implementation details of the open platform for 3D spatial information. In Section 4, we describe the transformation and rendering process of the platform. Finally, in Section 5, we outline the concluding remarks.

2 | VWORLD SPATIAL INFORMATION DATA

The spatial information data provided by the VWorld Data Center [1] of the Ministry of Land, Infrastructure, and Transport of South Korea are composed of aerial images, DEMs, 3D building data, points-of-interest, hybrid images, and other elements with sizes that can exceed 30 TB in total. This section introduces the structure and features of the VWorld spatial information data. We also describe the unique 3D data format for the Oracle BI publisher resource tracking file (XDO) for 3D structural models.

2.1 | Tile

A tile is a unit that includes spatial surface image information related to the Earth. A tile consists of an aerial image (256 pixels × 256 pixels) and a DEM image (64 pixels × 64 pixels) with terrain elevations. A tile representing the spherical form of the Earth is composed of 50 tiles at level zero and is divided into 36° latitude and 36° longitude lines in the WGS84 coordinate system [20], as shown in Figure 3. The tiles at level zero are divided into four equal parts and comprise level one tiles, which are in turn divided by 18° latitude and 18° longitude lines. Therefore, the tile data consist of 50 × 4 tile levels/level for levels in the range of 0–15.

The tile ID definition function h(l, r, c) is defined as in (1) from the bottom-left part to the first horizontal line. In addition, l, r, and c denote the level, row, and column.

$$\mathrm{h}(l, r, c) = \begin{cases} 10\times 2^{l}\times r + c, & \text{if } l = 0 \\ 50\times 4^{l-1} + 10\times 2^{l}\times r + c, & \text{otherwise.} \end{cases} \tag{1}$$

2.2 | Transformation of coordinate system

The location of the VWorld 3D spatial information is defined by the latitude (lat) and longitude (lon) coordinates in a 3D coordinate system. The latitude and longitude (latlon) positions are based on the WGS84 coordinates, and the 3D positions are based on the ECEF left-handed coordinate system, as shown in Figure 4A. The x-axis passes through the latlon coordinates of (0, 0) and (0, 180), or (0, –180). Additionally, the y-axis passes through the latlon coordinates of (0, 90) and (0, –90), while the z-axis passes through the north and south poles. The WGS84 and ECEF transformation equations are defined in (2) and (3). The radius of the Earth R is 6,378,137 (m). Accordingly, the latlon coordinates comprise of radian angles, and x, y, and z, are the values of the ECEF vector.

$$\text{ECEF vector} = \begin{bmatrix} R * \cos lat * \cos lon \\ R * \cos lat * \sin lon \\ R * \sin lat \\ 1 \end{bmatrix}, \tag{2}$$

$$\text{WGS84 vector} = \begin{bmatrix} \sin^{-1}\left(z \Big/ \sqrt{x^2+y^2+z^2}\right) \\ -\text{atan2}\,(y,x) \\ 1 \end{bmatrix}. \tag{3}$$

The 3D web-based graphic library WebGL uses the right-handed coordinate system, as shown in Figure 4B. In Section 4, we describe how to transform the XDO of VWorld data into the WebGL coordinates.

$$\text{atan2}\,(y, x) = \begin{cases} \tan^{-1}\left(\frac{y}{x}\right), & \text{if } x > 0, \\ \tan^{-1}\left(\frac{y}{x}\right) + \pi, & \text{if } x > 0 \text{ and } y \geq 0, \\ \tan^{-1}\left(\frac{y}{x}\right) - \pi, & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2}, & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2}, & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined}, & \text{if } x = 0 \text{ and } y = 0. \end{cases} \tag{4}$$

2.3 | XDO

The VWorld 3D structural data are subdivided into the index data for XDO and XDO. A tile has index data. It has as many data objects as the value of *objectCount*, as shown in Table 1. It also contains as many XDO data objects as the value of *objectCount*. XDO is a new format for presenting the VWorld structural models. As shown in Table 2, an XDO contains data for a 3D structural model for an independent building. Accordingly, ***vertex*** is a 3D point, ***index*** represents the index of the vertex data array for the composition of the triangular mesh, and ***uv*** is the 2D position of the triangular mesh in the texture image.

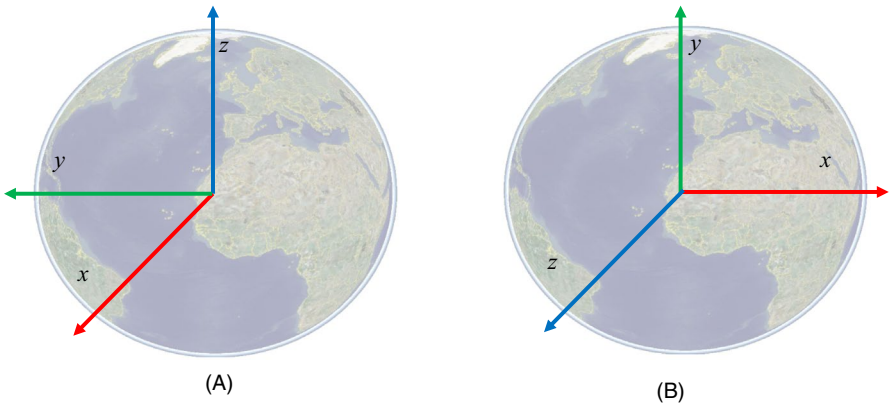


FIGURE 4 Coordinate systems: (A) Oracle BI publisher resource tracking file (XDO) and (B) WebGL

For example, Figure 5A shows a rendered building. Accordingly, a yellow mesh is represented by the three vertices v_0, v_1, and v_2, as shown in Figure 5B, which constitute the rendering result with the string of Figure 5A.

In this case, ***index*** is the position value (0, 1, 2) in the ***vertex*** data array. Additionally, ***uv*** has a 2D coordinate value at which the texture image resolution size is normalized to one, as shown in Figure 5C. Each of the values of the ***vertex*** and ***uv*** is four bytes long. Therefore, the size of the memory for the 3D coordinates for the ***vertex*** point is *vertexCnt* × 12 (bytes), and the size of the memory for the 2D coordinates for ***uv*** is *vertexCnt* × 8 (bytes). In the case of ***index***, the same ***vertex*** can be included in several **indices**. Hence, the number of **indices** is not proportional to the number of **vertices**.

VWorld XDO only provides 3D structural models for the major cities and facilities and not for the entire country. The currently available VWorld XDO data are only included on the tiles of levels 13–15. Level 14 contains bridges, and level 15 contains buildings. Level 13, in particular, contains terrain information for the Dokdo Island in South Korea, which is not a building.

3 | OPEN PLATFORM FOR 3D SPATIAL INFORMATION

This section describes the implementation details of the VWorld open platform. It describes the methods used to request and represent 3D data in accordance with the features of the spatial information of VWorld. In addition, given that our platform deals with a large amount of high-volume data, a method is also presented to effectively manage them. Figure 6 shows the task flow diagram of the proposed open platform. A series of processes is repeated at a rate of once per frame. The implemented platform consists of a basic unit referred to as a sector. The position of the camera is controlled using the user interface, and the draw request tiles (sectors) are searched by changing the camera position. The subsequent steps request and draw tiles (sectors) and execute WebGL-based rendering processes.

TABLE 1 VWorld index data for XDO

Name	Size	Contents
level	4	Level of tile
IDX	4	Longitude identity (ID)
IDY	4	Latitude ID
objectCount	4	Number of objects
	Object Data	
version	4	Version
type	1	Type
keyLen	1	Size of key
key	*keyLen*	Key
centerPos	16	Center coordinate
altitude	4	Altitude
box	48	Boundary
imgLevel	1	Image level
dataFileLen	1	Size of data file
dataFile	*dataFileLen*	Data file
imgFileNameLen	1	Size of image file name
imgFileName	*imgFileNameLen*	Image file name

TABLE 2 VWorld XDO data

Name	Size	Contents
vertexCnt	4	Size of vertices
vertex	*vertexCnt* × 12	Vertices
uv	*vertexCnt* × 8	uv
indexCnt	4	Size of indices
index	*indexCnt*	Indices
color	4	Color of object
box	48	Boundary

3.1 | Sector

A tile is a terrain image combined with an aerial image and a DEM. In our proposed platform, a structure that combines a tile with 3D structural models in the tile is called a sector. A sector includes an aerial image, a DEM, a 3D structural model (XDO), the size and location of the tile, and other information. Similar to a tile, a sector contains various levels (0–15), and the level of the displayed sector is determined based on the distance between the camera and the center of the sector. A sector's identity (ID) is determined in the same manner as a tile ID in (1).

A sector is the basic unit for managing data in our platform, as shown in Figure 7. The sector manager determines the data to be requested, stored, and displayed on a sector-by-sector basis. The open map application programming interface (API) provides map manipulation functionality, and the open data API provide data requests and management functions. The WebGL library is used to present the sector data in 3D.

3.2 | Quadtree structure

This section presents the draw–request sector search function shown in Figure 6. The sector search function is performed based on the quadtree structure [21,22]. Fifty sectors at level zero have four child sectors each, and each child sector contains another four child sectors. Therefore, the 15 levels are composed of $50\times4^{\text{level}}$ sectors. Our platform has a quadtree structure to effectively present the multilevel sectors in real time.

If a parent sector is not the rendering target in a quadtree structure, its child sectors are not rendering targets either. For example, if a level zero parent sector is not the rendering target, it is not necessary to perform the culling test (see Table 3) on all of its child sectors. That is, 1,073,741,824 sectors exist for the range of 1–15. Our main aim is to propose a method to render effectively VWorld spatial information (with a size amounting to 30 TB or more) with the use of a low-end PC, without a GPU, in real time.

Sectors of various levels are presented according to the position and direction of the camera, as shown in Figure 7. The level of the sector to be displayed is calculated as the distance between the camera pose $\boldsymbol{c}$ and the center pose $\boldsymbol{s}$ of the sector in the ECEF coordinate system.

$$\text{dist}=\sqrt{(\boldsymbol{c}_x-\boldsymbol{s}_x)^2+(\boldsymbol{c}_y-\boldsymbol{s}_y)^2+(\boldsymbol{c}_y-\boldsymbol{s}_y)^2}, \tag{5}$$

$$\text{level}=\lceil\frac{\log(R/dist)}{\log 2}\rceil. \tag{6}$$

If the levels of the displayed sectors are different, overlapping sectors are drawn. In addition, if there is a sector whose data request is incomplete, screen flickering may occur. For example, sectors for levels 10 and 8 do exist, but no sectors exist for level 9. When the presented level is changed from 8 to 10, the sector for level 9 cannot be displayed owing to the relevant web browser feature, which cannot guarantee that the request will be downloaded first.

Our quadtree-based platform has overcome the aforementioned limitations. First, it conducts a culling test on all 50 sectors at level zero, as shown in Table 3. In addition, the drawing level of the sectors that have passed the culling test is determined based on (6). If the level of a sector is smaller than the draw level, it is divided into four child sectors, and the culling test is conducted until the level of the sector becomes equal to the draw level.

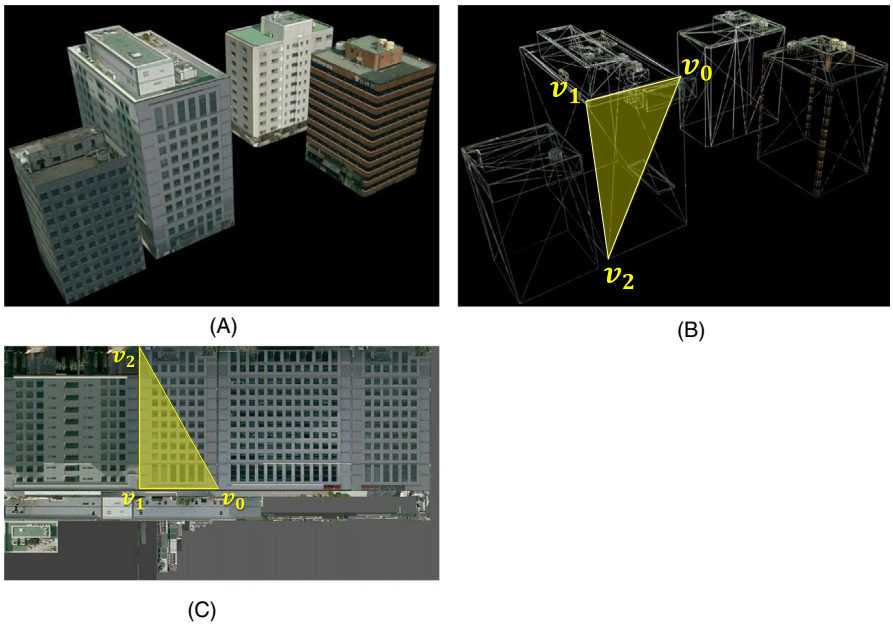


FIGURE 5 Examples of rendering of XDO 3D buildings: (A) rendered buildings, (B) a triangular mesh in a rendered building represented as a string, and (C) a triangular area in the texture image

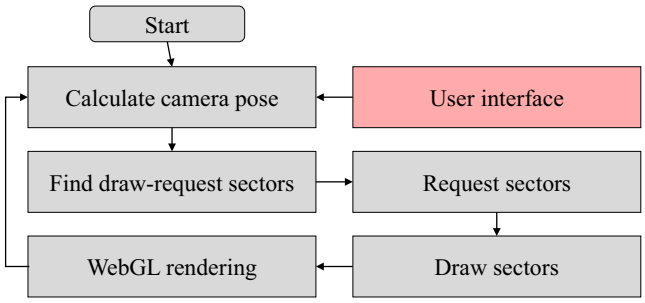


FIGURE 6 Task flow of the proposed platform: the red block indicates an event handler for the user interface, and the gray blocks are the render loop processes

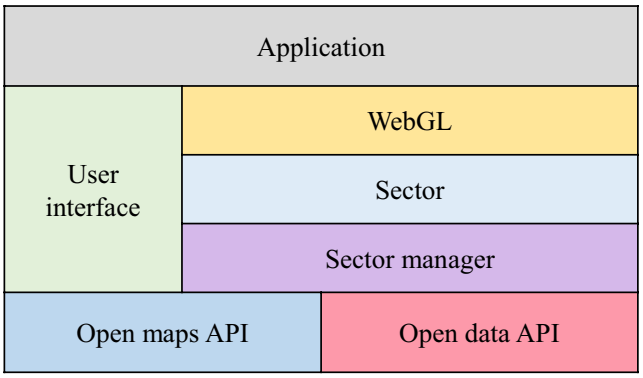


FIGURE 7 Structure of the VWorld open platform for spatial information

When the draw level and the level of the sector are the same, a check is performed to assess whether the data request has been completed. If the request of the child sector has been completed, it is placed in the draw array; otherwise, the parent sector is placed in the draw array, and the child sector is placed in the request array. A request for sector data in the request array is performed through the request sector function, as shown in Figure 6. Figure 8 shows an example of the rendered results according to the drawn levels of each sector.

3.2.1 | Culling test

The proposed platform performs a culling test according to three steps. Basic back-face and view-frustum culling tests are classic methods. Both culling tests were tested using the four sector corners. The corner $\mathbf{s}_i$ is a 3D vector based on the XDO coordinate system, as shown in Figure 9. In the case of back-face culling, if the angle between the corner point and the camera is less than 90°, the target is a rendering target and is in the front [23–25].

$$\text{if } (\mathbf{c}-\mathbf{s}_i)\cdot|\mathbf{s}_i|\geq 0, \quad \mathbf{s}_i \text{ is front.} \tag{7}$$

The view-frustum culling is performed using $\mathbf{s}_i$ only, which has been determined to be in the front. If one of the corners is located in the direction of the normal vectors of all six frustum planes, this sector must be inside the view-frustum and represents the rendering target. If the size of the red sector is too large, all the corners are located outside the view-frustum plane although the sector has to be rendered, as shown in Figure 10. When such a problem occurs on this platform, the target sector is the central sector. The central sector is the sector that contains the intersection of the principal line of the camera and the surface of the Earth. The numbers of rows and columns of the central sector at each level can be calculated as follows:

$$\begin{aligned}\text{center row} &= \left\lfloor (90+\text{lat}) * \frac{2^{lv}}{36} \right\rfloor, \\ \text{center column} &= \left\lfloor (180+\text{lon}) * \frac{2^{lv}}{36} \right\rfloor.\end{aligned} \tag{8}$$

The row, column, and level can be used to calculate the sector ID in (1) to determine whether the sector is a central sector or not. In the case of the central sector, the back-face and view-frustum culling tests are not applied. With the exception of the central sector, all the other sectors conduct the back-face and view-frustum culling tests.

TABLE 3 Pseudocode of the draw–request sector search function based on the quadtree structure

```
FUNCTION: FIND DRAW–REQUEST SECTORS()
  FOR: i = 0 to 49 DO
    IF: CULLING TEST(sector[i])
      drawLevel = Set the draw level (sector[i]).
      IF: drawLevel > sector[i].level
        FIND CHILD SECTORS (sector[i]).
      ELSE:
        IF: Check loaded data (sector[i]).
          Save in drawArray.
        ELSE:
          Save in requestArray.
END

FUNCTION: CULLING TEST(s)
  IF: Is center sector(s)
    RETURN true.
  IF: Back-face culling(s) AND View-frustum culling(s)
    RETURN true.
  ELSE:
    RETURN false.
END

FUNCTION: FIND CHILD SECTORS(s)
  s.child[0] = h (s.level + 1, s.row × 2, s.col × 2).
  s.child[1] = h (s.level + 1, s.row × 2, s.col × 2+1).
  s.child[2] = h (s.level + 1, s.row × 2 + 1, s.col × 2).
  s.child[3] = h (s.level + 1, s.row × 2 + 1, s.col × 2+1).
  FOR: i = 0 to 3 DO
    IF: CULLING TEST (s.child[i])
      drawLevel = Set the draw level (s.child[i]).
      IF: drawLevel > s.child[i].level
        FIND CHILD SECTORS(s.child[i])
      ELSE:
        IF: Check loaded data (s.child[i]).
          Save in drawArray.
        ELSE:
          Save s.child[i],parent in drawArray.
          Save s.child[i] in requestArray.
END
```

3.2.2 | Data request

Among the sectors that have passed the culling test, the sectors that are not requested are placed in a request array. When the quadtree structure-based search process is completed, it requests data by prioritizing the sectors in the request array, as shown in Table 4. First, the sectors in the request array are sorted in ascending order based on their levels. The distance between the camera and the center of the sector is then calculated at each array level, and the sectors are sorted in ascending order based on the distances.

The maximum number of simultaneous, default, and persistent connections per server/proxy is set to be in the range of 2–8 according to the web browsers used. Our open platform is optimized for Google Chrome and is limited to six connections. Therefore, only the previous two sector data are requested at each frame of the results when performing the two ascending sorting operations of Table 4. At this time, the requested data include an aerial image, a DEM, and index data for XDO. The aerial image is saved in matrix of 256 × 256 pixels at all levels, and the size per unit is approximately 40 kB to100 kB. The DEM is also less than approximately 10 kB in size, which is not large. The proposed platform renders approximately 10 to 70 sectors in a frame and requests an aerial image, a DEM, and index data for XDO, while it gradually divides the sectors. Therefore, 3D terrain data requests and management can be easily performed in a general network environment.

However, in the case of XDO data, as the size ranges from 100 kB to 10 MB, there is a constraint in requesting a large number of 3D structure models in real time. XDO does not immediately initiate the request but only places it in the request XDO array, as shown in Table 4. XDO data are requested in the frames where aerial images and DEMs are not requested. Terrain data with a smaller size than the size of large XDO datasets are preferentially represented. Thus, the platform can have a stable display environment for users in real time [3].

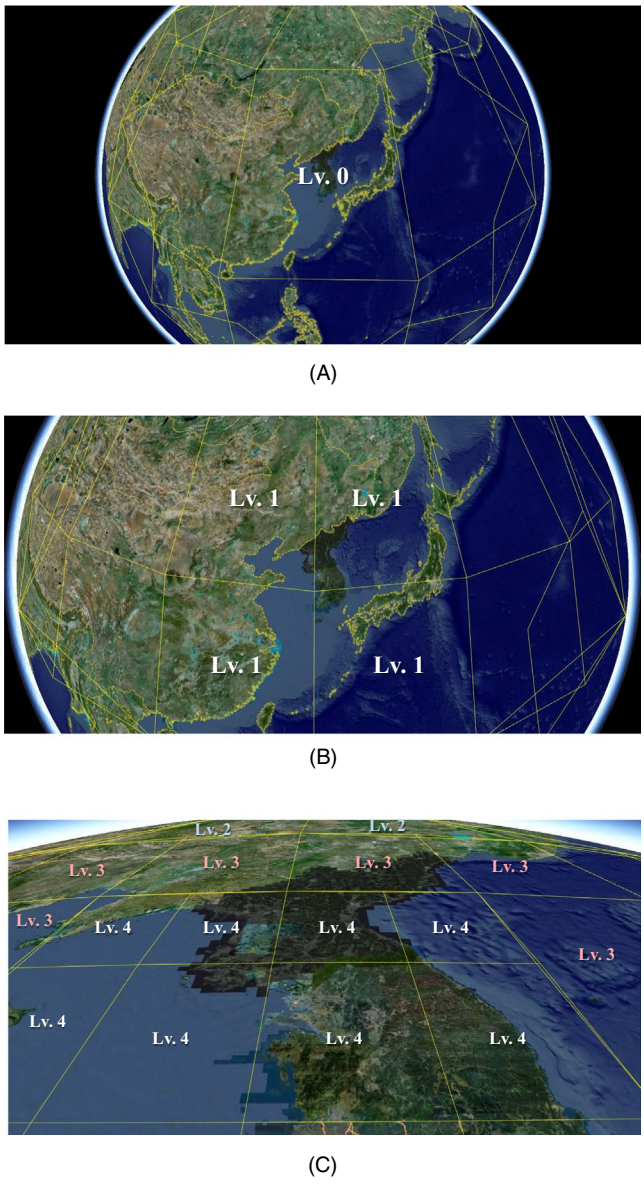


FIGURE 8 Example of rendering results: (A) level zero, (B) level zero sector division into four child sectors at level one when the camera approaches the sectors, and (C) a view of the sectors at multiple levels ("Lv." denotes the level of the sector)

3.2.3 | Quadtree structure

Only the sectors of levels 13 through 15 have XDO data for 3D structural models. Level 14 contains bridges, and level 15 contains buildings, as shown in Figures 11A and 11B. Specifically, level 13 contains terrain information on the Dokdo Island in South Korea and not building information, as shown in Figure 11C. Among the 3,358 islands in Korea, Dokdo—which is relatively small in size—has a prominent position in the national territorial sea. This area has been associated with a continuous diplomatic conflict with Japan although it has been used as a residential area for Koreans. Thus, Dokdo is illustrated in detail with the

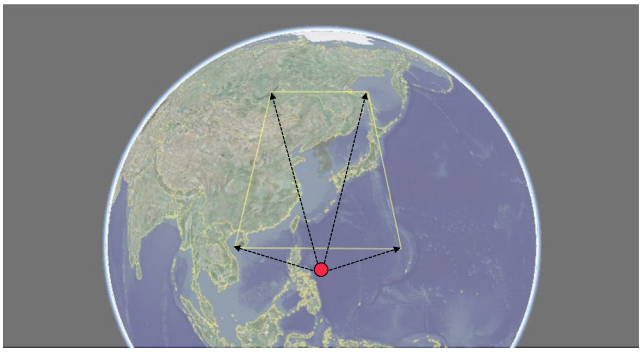


FIGURE 9 Corner vectors of a sector: the red dot is the center point of the Earth, and the four black arrows indicate the corner vectors of a sector

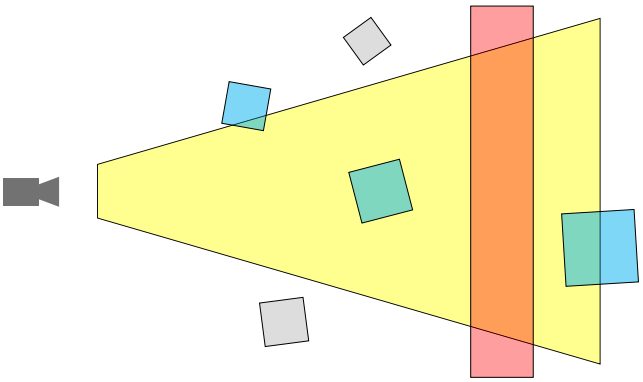


FIGURE 10 A sector (red) is bigger than the view-frustum range: blue blocks are inside (or partially inside) the view-frustum, and gray blocks are outside

TABLE 4 Pseudocode of the request sector function

```
FUNCTION: REQUEST SECTORS()
  lvArray = Sorting with level (requestArray).
  requestCount = 0.
  FOR: i = 0 to lvArray.eachLv.length DO
    distArray = Sorting with distance (lvArray.eachLv[i]).
    FOR: j = 0 to distArray.length DO
      Request sector data (distArray[j]).
      IF: distArray[j] has XDO
        Request index data for XDO.
        Put requested XDO data in requestXDOArray.
      requestCount ++.
      IF: requestCount ≥ maxRequestNumber
        RETURN
END
```

use of two XDO data objects rather than DEMs, unlike other terrains.

If a 3D structure model is represented only when a sector with an XDO is displayed, buildings are often not shown owing to changes in the camera position, as indicated in Figure 12A. To acquire a natural display when rendering the 3D structural models, the proposed platform draws the XDO, as shown in Tables 5 and 6. Even if a sector does not have XDO data, 3D models can be shown in the sectors of levels 12–15. The rendering results are shown in Figure 12B.

4 | TRANSFORMATION AND RENDERING

The translation and rotation of the virtual camera in the open platform are performed through the user interface. The represented sectors are determined according to the camera position, and the WebGL library is used for the 3D rendering. This section describes the procedure for a WebGL transformation [25,26]. The 3D objects in the XDO coordinate system are transformed into 2D objects in the screen coordinate system based on the WebGL transformation pipeline, as shown in Figure 13 [25].

The initial 3D object is included in the XDO coordinate system, as shown in Figure 4A. However, the WebGL coordinate system uses the right-hand coordinate system shown in Figure 4B. The transformation from XDO into the WebGL coordinate system is included in the model-view matrix **M**.

$$\mathbf{M}=\begin{pmatrix} -y\text{Axis}_{\text{xdo}} & z\text{Axis}_{\text{xdo}} & x\text{Axis}_{\text{xdo}} & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (9)$$

The model-view matrix **M** changes according to the user interface operation. The central point indicates the intersection of the principal line of the camera and the surface of the Earth.

The changes in the movement of the central point are the same as the movement of the satellite orbiting the Earth. The camera rotates around the Earth while pointing toward the center of the Earth. The changes in the latitude and longitude coordinates are rotated along the *z*- and *y*-axes, respectively, in the XDO coordinate system. Herein, **R** is the rotation matrix of each axis, and $\mathbf{M}'$ is a model-view matrix that excludes the tilt transformation and applies only the rotation of each axis.

$$\mathbf{M}'=\mathbf{M}\cdot\mathbf{R}(\alpha)_x\mathbf{R}(\beta)_y\mathbf{R}(\gamma)_z. \quad (10)$$

The tilt transformation imposes a translation by -*R* along the *x*-axis direction, whereby a rotation of the tilt angle θ along the *y*-axis is implemented first followed by a translation by *R* along the *x*-axis direction, as shown in Figure 14. When the tilt transformations are added, the camera will not aim toward the center of the Earth. Accordingly, the direction of the camera will be aimed toward the center of the Earth only when the tilt angle is equal to 0°.

$$\mathbf{L}=\mathbf{T}(R)_x\mathbf{R}(\theta)_z\mathbf{T}(-R)_x \quad (11)$$

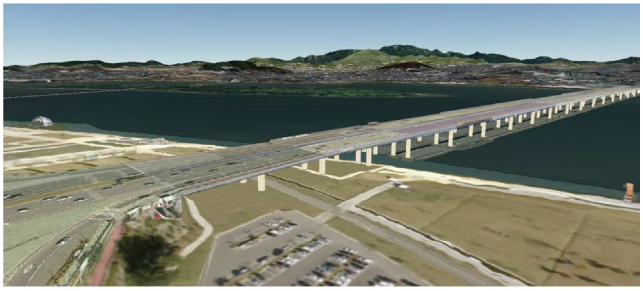
The transformation matrix between the XDO coordinates and the WebGL coordinate system is expressed by (12).

$$\mathbf{i}_{\text{webgl}}=\mathbf{M}'\cdot\mathbf{L}\cdot\mathbf{i}_{\text{xdo}} \quad (12)$$

The projection matrix defines the viewing volume of the frustum, that is, how the 3D object data are projected onto the screen. The projection matrix depends on the setting of the virtual camera. If the attributes of the virtual camera do not change, the original settings are used. The parameters are $=near$, $f=far, t=n*\frac{\text{fovy}}{2}, b=-n, r=t*\frac{\text{width}}{\text{height}}$, and $l=b*\frac{\text{width}}{\text{height}}$ in (13).

$$\mathbf{P}=\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \quad (13)$$

(A)



(B)

(C)

FIGURE 11 Examples of XDO model. Buildings in the (A) Yeouido area represented with 15 levels, (B) Mapo Bridge represented with 14 levels, and in the (C) Dokdo (13 levels)

The clip coordinates are transformed with the use of the projection matrix and are finally transformed to $\mathbf{i}_{screen}$ in the screen coordinate system based on the viewport transformation matrix $\mathbf{V}$.

$$\mathbf{V}=\begin{pmatrix} \frac{width}{2} & 0 & 0 & \frac{width}{2} \\ 0 & \frac{heigth}{2} & 0 & \frac{height}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{14}$$

$$\mathbf{i}_{screen}=\mathbf{V}\cdot\mathbf{P}\cdot\mathbf{i}_{webgl}=\mathbf{V}\cdot\mathbf{P}\cdot\mathbf{M}'\cdot\mathbf{L}\cdot\mathbf{i}_{xdo}. \tag{15}$$

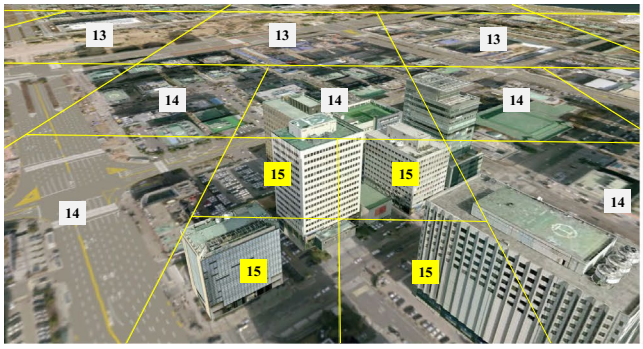
If the tilt transformation matrix $\mathbf{L}$ is not included, the inverse matrix of the summation matrix can be calculated. However, if the rotation and translation transformation matrices are overlapped, the inverse relationship is not established. To transform the WebGL coordinates into the XDO coordinates, the inverse matrix of the tilt transformation $\mathbf{L}^{-1}$ must be calculated first according to (16). In addition, the inverse transformation is calculated based on (17).

$$\mathbf{L}^{-1}=\mathbf{M}'^{-1}\mathbf{T}(-R)_x\mathbf{M}'^{-1}\mathbf{R}(\theta)_z\mathbf{M}'^{-1}\mathbf{T}(R)_x, \tag{16}$$

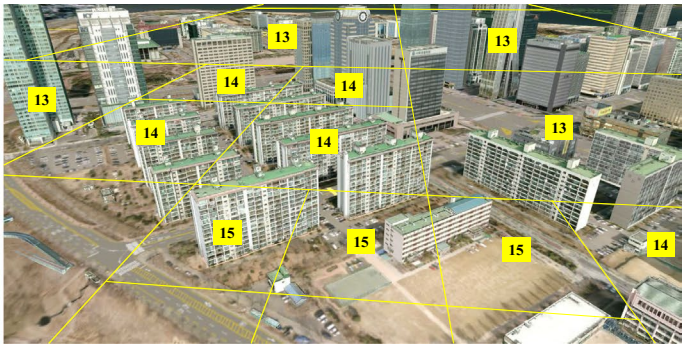
$$\mathbf{i}'_{xdo}=\mathbf{L}^{-1}\mathbf{M}'^{-1}\mathbf{i}_{webgl}. \tag{17}$$

5 | EXPERIMENTS

Experiments were performed using two types of computers. The first was a desktop computer with an Intel® 4.20 GHz Core™ i7 CPU, an NVIDIA GeForce GTX 1080Ti GPU, and a wired network. The other was a notebook with an Intel® 2.70 GHz Core™ i7 CPU, an Intel HD 620 graphics processing unit (GPU), and a wireless network. The desktop was equipped with an external GPU, whereas the notebook was not. The web browser used in the experiment was Google Chrome (version 67.0.3396.99, 64 bit).



(A)



(B)

FIGURE 12 Rendering examples of XDO with the proposed method: (A) buildings are presented only at level 15, and (B) buildings presented at levels 12–15

TABLE 5 Pseudocode used for the presentation of 3D structural models with levels in the range of 12–15

```
FUNCTION: DRAW BUILDING(s)
  IF: s.level = 15,
    Draw XDO data of s.
  IF: s.level = 14,
    FOR: i = 0 to 3 DO
      Draw XDO data of s.child[i].
  IF: s.level = 13
    FOR: i = 0 to 3 DO
      FOR: k = 0 to 3 DO
        Draw XDO data of s.child[i].child[k].
  IF: s.level = 12
    FOR: I = 0 to 3 DO
      FOR: k = 0 to 3 DO
        FOR: l = 0 to 3 DO
          Draw XDO data of s.child[i].child[k].child[l].
END
```

TABLE 6 Pseudocode used for the presentation of 3D bridges and Dokdo with levels in the range of 12–15

```
FUNCTION: DRAW BRIDGE(s)
  IF: s.level = 12
    FOR: I = 0 to 3 DO
      FOR: k = 0 to 3 DO
        Draw XDO data of s.child[i].child[k].
  IF: s.level = 13,
    FOR: I = 0 to 3,
      Draw XDO data of s.child[i].
  IF: s.level = 14,
    Draw XDO data of s.
  IF: s.level = 15
    Draw XDO data of s.parent.
END

FUNCTION: DRAW DOKDO(s)
  IF: s.level = 12,
    FOR: i = 0 to 3DO
      Draw XDO data of s.child[i].
  IF: s.level = 13,
    Draw XDO data of s.
  IF: s.level = 14
    Draw XDO data of s.parent.
  IF: s.level = 15
    Draw XDO data of s.parent.parent.
```

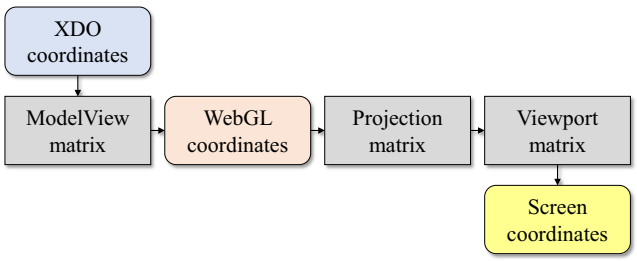


FIGURE 13 WebGL transformation pipeline [25]

Table 7 shows the processing times of each step in one of the frames at which the screen is updated. It effectively denotes the average value of the processing time measured when 2,000 frames are used when 106,496 polygons and 208 texture files are rendered. To render the same amount of data, the values of latitude, longitude, height, and the values of α, β, and γ, which are listed in (10), are set to the same values. Specifically, the request sector process is only requested in 146 frames in the case where a GPU is used, or in 713 frames in the case where a GPU is not used. Thus, an average number of 146 and 713 frames are requested, respectively. The "find-sectors" process is executed to identify the sectors to be presented. The rendering process includes a step for creating a 3D terrain model using an aerial image and DEM data and for inputting the 3D model data into a WebGL buffer. The number of frames per second (fps) is limited to a maximum of 60, depending on the web browser environment. Therefore, the maximum speed achieved with the use of GPU is 60 fps.

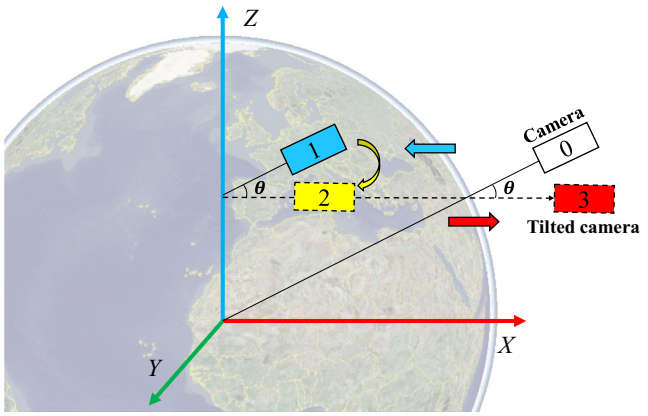


FIGURE 14 Tilt transformation in the XDO coordinate system

The experimental results showed that the proposed quadtree-based, open platform for 3D spatial information had a speed of at least 40 fps, even in the case where a GPU was not used. Therefore, our platform enhances the use of the 3D spatial information map only if a web browser is used that supports WebGL, irrespective of the computing environment.

We compared the processing time with that of the WebGL-based Seoul 3D GIS [20] using VWorld data. Unlike our platform, which deals with the global area, the Seoul 3D GIS deals only with the area of the city of Seoul, as shown in Figure 2G. To accelerate the rendering of the 3D buildings, the texture images of all the buildings in a single tile are merged and saved in a single file. Hence, the resolution and quality are lower than those of the original files provided by the VWorld Data Center.

To measure the processing time of the Seoul 3D GIS, we used the FPS meter in DevTools of Chrome. A speed of approximately 34.8 fps was recorded in an environment without a GPU. However, it is difficult to compare the absolute speed with that of our platform because we cannot measure

TABLE 7 Processing times of each step (expressed as the average values based on the use of 2,000 frames, while the maximum frame rate of web browser in experiment environment is 60 frames per second)

	Our platform		Seoul 3D GIS	
	GPU (ms)	Without GPU (ms)	GPU (ms)	Without GPU (ms)
Find sectors	5.114	18.194	60 fps	34.8 fps
Request sectors	0.342	0.344	60 fps	34.8 fps
Rendering	1.469	6.517	60 fps	34.8 fps
Total	6.925 (60 fps)	25.055 (39.9 fps)	60 fps	34.8 fps

the amount of 3D data, such as the polygons or texture files that are being rendered in a single frame in the case of the Seoul 3D GIS.

6 | CONCLUSIONS

In this study, we described the research and implementation method of an open platform for 3D spatial information based on WebGL with VWorld data built for the terrain of South Korea. We have also described how to render XDO data for 3D structural models using WebGL. With this platform, high-quality 3D spatial information with a total size of ≥ 30 TB can be effectively presented based on the quadtree-based sectors with a low-end PC and without a GPU. The experiments showed that a rendering speed of at least 40 fps could be maintained in a non-GPU computational environment. The performance of the proposed platform is similar to that of the state-of-the-art commercial platform in which it is difficult to disclose internal algorithms or methods. We expect that various applications related to 3D spatial information will be developed based on our open platform, which has been developed for public purposes.

In our future research endeavors, we will study an open platform for advanced 3D spatial information. For stability in various computing environments, we intend to propose an optimization method according to the computational environment. We also intend to provide OpenAPI for developers to support various applications using the VWorld spatial information platform. In addition, we intend to develop a VWorld platform based on Unity3D or a Unity3D asset for games, urban simulations, or MRs. We anticipate that when users purchase the Unity3D assets of the VWorld open platform, they will be able to simply create a VWorld 3D map.

ORCID

Ahyun Lee https://orcid.org/0000-0002-7788-1921

REFERENCES

1. VWorld Data Center, operated by the Ministry of Land, Transport and Maritime Affairs of, South Korea (2017), available at http://data.vworld.kr/data/v4dc_usrmain.do.
2. H. S. Jang et al., *Performance evaluation of CDN method in V-World web service as spatial information open platform service*, Spat. Inf. Res. **24** (2016), 355–364.
3. M. S. Kim and I. S. Jang, *Efficient in-memory processing for huge amounts of heterogeneous geo-sensor data*, Spat. Inf. Res. **24** (2016), 313–322.
4. C. Marrin, Webgl specification, Khronos WebGL Working Group, 2011.
5. E. Angel and D. Shreiner, *Interactive Computer Graphics: A Top-Down Approach with WebGL*, 7th ed, Addison-Wesley, Boston, 2015.
6. G. Lavoué et al., *Streaming compressed 3D data on the web using JavaScript and WebGL*, in Proc. 18th Int. Conf. on 3D Web Technol., San Sebastian, June 2013, pp. 19–27.
7. J. P. Suárez et al., *An open source virtual globe framework for iOS, Android and WebGL compliant browser*, in Proc. Int. Conf. Comput. Geospatial Research Applicat., Washington, D.C., USA, 2012, pp. 22:1–10.
8. B. Chen, and Z. Xu, *A framework for browser-based multiplayer online games using WebGL and WebSocket,* in Int, Conf. Multimed. Technol., Hangzhou, China, 2011, pp. 471–474.
9. J. Congote et al., *Interactive visualization of volumetric data with webgl in real-time,* in Proc. Int. Conf. 3D Web Technol, New York, USA, 2011, pp. 137–146.
10. L. Wood, *Programming the Web: The W3C DOM specification*, IEEE Internet Comput. **3** (1999), 48–54.
11. VWorld, 3D spatial information open platform base on WebGL, operated by the Ministry of Land, Transport and Maritime Affairs of, South Korea (2017), available at http://map.vworld.kr/map/wcmaps.do.
12. OpenGlobus: A. JavaScript library for interactive 3D maps, 2016, available at http://www.openglobus.org.
13. Cesium: An, open-source JavaScript library for world-class 3D globes and maps, Analytical Graphics, Inc. (AGI) and, Bentley Systems (2012), available at https://cesiumjs.org.
14. Google Earth, Google. 2017, available at https://earth.google.com/web.
15. K. Min et al., *A system framework for map air update navigation service*, ETRI J. **33** (2011), 476–486.

16. J. Tang and G. H. Gongm, *Modeling and real-time rendering technology of real large area terrain database*, J. Syst. Simul. **18** (2006), 453–456.
17. D. Koller et al., *Virtual GIS: A real-time 3D geographic information system,* in Proc. Conf. Visualization' 95, Atlanta, GA, USA, 1995, pp. 94–100.
18. M. Krivokuća et al., *Progressive compression of 3D mesh geometry using sparse approximations from redundant frame dictionaries*, ETRI J. **39** (2017), 1–12.
19. Seoul 3D, GIS (2018), available at http://3dgis.seoul.go.kr.
20. R. Stanaway, *GDA94, ITRF, WGS84: What's the difference?*, Working with dynamic datums, Australia, 2007.
21. H. Samet, *The quadtree and related hierarchical data structures*, ACM Computing Surveys (CSUR) **16** (1984), 184–260.
22. J. Kim and I. K. Jeong, *Single image-based 3D tree and growth models reconstruction*, ETRI J. **36** (2014), 450–459.
23. Broggi et al., *Terrain mapping for off-road autonomous ground vehicles using rational B-spline surfaces and stereo vision,* in IEEE Intell, Vehicles Symp., Gold Coast, Australia, 2013, pp. 648–653.
24. L. Zebedin et al., *Towards 3D map generation from digital aerial images*, ISPRS J. Photogramm. Remote Sens. **60** (2006), 413–427.
25. M. Woo et al., *OpenGL Programming Guide: The Official Guide to Learning OpenGL—Version 1.2, third ed*, Addison-Wesley, 1999.
26. J. F. Foley et al., *Introduction to Computer Graphics*, Addison-Wesley, Reading, MA, 1994.

AUTHOR BIOGRAPHIES

Ahyun Lee received his PhD degree in computer science from the University of Science & Technology, Daejeon, Rep. of Korea. From 2011 to 2012, he was an engineer at LG Electronics Inc. Pyeongtaek, Rep. of Korea. He has been a research scientist at Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea, since 2016. His research interests include computer vision, augmented reality, robotics, and spatial information.

Insung Jang received the BS and MS in computer engineering from Pusan National University, Busan, Rep. of Korea, in 1999 and 2001, respectively. Since 2001, he has been a senior member of research staff at Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea, and he has been working toward his PhD degree in computer engineering at the Pusan National University. His main research areas include platforms for geosensor, navigation, and location-based services.