Computers and Geosciences 111 (2018) 200–212

Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

3D geospatial visualizations: Animation and motion effects on spatial objects

Konstantinos Evangelidis [a,*], Theofilos Papadopoulos [a], Konstantinos Papatheodorou [a], Paris Mastorokostas [b], Constantinos Hilas [c]

[a] Technological Educational Institute of Central Macedonia, Faculty of Applied Technology, Department of Civil, Surveying and Geoinformatics Engineering, Terma Magnisias, 62124 Serres, Greece
[b] Department of Computer Engineering, Piraeus University of Applied Sciences, 250, Thivon & P. Ralli Str., 12244 Egaleo - Athens, Greece
[c] Technological Educational Institute of Central Macedonia, Faculty of Applied Technology, Engineering Informatics Department, Terma Magnisias, 62124 Serres, Greece

ARTICLE INFO

Keywords:
3D geospatial visualization
3D spatial objects motion
WebGL
Javascript

ABSTRACT

Digital Elevation Models (DEMs), in combination with high quality raster graphics provide realistic three-dimensional (3D) representations of the globe (virtual globe) and amazing navigation experience over the terrain through earth browsers. In addition, the adoption of interoperable geospatial mark-up languages (e.g. KML) and open programming libraries (Javascript) makes it also possible to create 3D spatial objects and convey on them the sensation of any type of texture by utilizing open 3D representation models (e.g. Collada). One step beyond, by employing WebGL frameworks (e.g. Cesium.js, three.js) animation and motion effects are attributed on 3D models. However, major GIS-based functionalities in combination with all the above mentioned visualization capabilities such as for example animation effects on selected areas of the terrain texture (e.g. sea waves) as well as motion effects on 3D objects moving in dynamically defined georeferenced terrain paths (e.g. the motion of an animal over a hill, or of a big fish in an ocean etc.) are not widely supported at least by open geospatial applications or development frameworks. Towards this we developed and made available to the research community, an open geospatial software application prototype that provides high level capabilities for dynamically creating user defined virtual geospatial worlds populated by selected animated and moving 3D models on user specified locations, paths and areas. At the same time, the generated code may enhance existing open visualization frameworks and programming libraries dealing with 3D simulations, with the geospatial aspect of a virtual world.

1. Introduction

We have come a long way since the appearance of the first web mapping projects at the end of the 20th century (Haklay et al., 2008). The technological progress that affected crucially this area included, first of all, the tremendous increase of internet speed connections (Eha, 2013), combined with efficient image compression and presentation techniques (e.g. tiled rendering) that allowed web clients to request rich raster images associated with large spatial datasets (Batty et al., 2010). As a result, today it is a routine job to perform a virtual tour, on an unknown or inaccessible area or surface, by navigating through a browser over a three-dimensional (3D) Digital Terrain Model (DTM) or a Digital Surface Model (DSM) or a set of joined panoramic photographs.[1] Secondly, the substantial progress occurred in Web Semantics and the related geospatial web services introduced by the Open Geospatial Consortium (OGC), with Web Map Service (WMS) being directly applicable and widely used, have enabled data sharing and interoperability among different fields of geosciences (Evangelidis et al., 2014). The aforementioned evolutions assisted also by the progress met in IT and hardware infrastructure extended the usage of location based services, also termed as geoservices, in people daily activities through the rapid penetration of smartphone and tablet devices. Recent studies showed that 35% of smartphone users in the five largest European economies access maps on their device (Oxera, 2013) while at the same time map usage via

* Corresponding author. Technological Educational Institute of Central Macedonia, Faculty of Applied Technology, Terma Magnisias, 62124 Serres, Greece.
E-mail addresses: kevan70@gmail.com (K. Evangelidis), priestont@gmail.com (T. Papadopoulos), conpap@teicm.gr (K. Papatheodorou), mast@teipir.gr (P. Mastorokostas), chilas@teicm.gr (C. Hilas).

[1] https://www.google.com/maps/streetview/explore/index.html.

https://doi.org/10.1016/j.cageo.2017.11.007
Received 22 November 2015; Received in revised form 21 December 2016; Accepted 1 November 2017
Available online 21 November 2017

smartphone is growing seven times faster than via the classic web.[2] In conclusion, today, technological capabilities from the one side and people daily needs from the other, justify efforts spent towards enhancing graphical representations of geospatial data over the web by providing 3D visualization and animation effects on spatial objects.

Computer graphics technologies can nowadays offer high 3D modeling visualization and animation capabilities combined with high satisfaction end-user interactivity, through a web browser. Such developments are supported by cross-platform frameworks like WebGL API,[3] which are interpreted and rendered directly by any web browser compatible with World Wide Web Consortium (W3C) proven standards, without the need of installing plug-ins. The above technologies are mainly applied in recreational applications (3D Games) or applications with 3D photorealistic effects representing the real world, without maintaining the geospatial dimension of the involved objects. In other words, the graphical representations in such frameworks are not associated with a georeferenced area or map, and the geospatial aspect of the involved animated and moving objects does not have any practical value.

In the geospatial community, Digital Elevation Models (DEMs) either DTMs or DSMs are widely used over the last 25 years providing views of the ground terrain or surface along with elevation values.[4] Today Geographic Information Systems (GIS) software can export DEMs into an interoperable XML-based format.[5] Such way DEMs may be potentially visualized through web browsers and furthermore, managed by applications developed under the prevailing on the World Wide Web (WWW) open software development technologies. By employing 3D models that represent inanimate or animated objects and also by utilizing open libraries providing animation and movement functionalities on these objects it is possible to achieve real world geo-referenced simulations like for example a herd of animated animals moving through a forest on top of a DTM served by a WMS. The above may be enhanced with major GIS functionalities, such as the creation of thematic layers for the various different categories of 3D models along with the interactivity of these spatial models with user specified geospatial locations, paths or areas. Thus, the coupling of geospatial functionalities on the one hand, with computer graphics web technologies on the other, may be considered as a strong applied research challenge and is reflecting the clear objective of the presented work.

To achieve this, we gathered potential desirable functionalities involving both animation and moving capabilities of 3D models (spatial objects), yet not currently offered, at least in a complete, structured and usable form, by existing open web-GIS environments or Javascript (JS) libraries. For example a user selected 3D model representing an animated object (e.g. a galloping horse), scaled to fit to a 3D world created by the user (e.g. a forest in a mountain area), and also moving on a user-specified path, is one of them. To populate a user-specified area with a 3D spatial object representing for example a plant, or with an animated texture in order to simulate for example water, are additional interesting functionalities. Ideally, such functionalities and many more should be applied on user specified dynamic geospatial worlds, designed in various independent thematic layers, possessing any geographic, or not, non angular coordinates. For this reason the presented work considers also typical GIS-based capabilities to fully provide the geospatial aspect of 3D modeling visualization, animation and moving effects.

Having made the proposed work available for the research community[6] it is possible to be directly exploited by acting as a supplement to existing JS libraries. For example it may extend the non geospatial libraries which focus on 3D animation (e.g. Three.js[7]) with geospatial functionalities. In addition it may enhance geospatial libraries that mainly focus on 3D globes and spatial data visualizations (e.g. Cesium.js[8]) with major GIS-based capabilities. The results of the above approach may act as a forerunner for enriching Earth Browsers with capabilities of representing 3D referenced inanimate or moving and animated spatial objects (e.g. flora and fauna) as well as phenomena simulations (weather phenomena, landslides, floods etc.)

2. Related work

Today JS has become the dominant programming language employed by the majority of websites and supported by all modern web browsers without plug-ins. The WebGL JS API, based on OpenGL,[9] utilizes hardware acceleration and brings 3D to the web, standing as the base for continuously emerging Javascript visualization frameworks and libraries developed on top of it. In this respect the presented work and the related works reviewed are exclusively based on the above free web standards. However, a reference to the following technologies should be cited:

- Java 3D, the 3D API for Java platform, running since 2012 on top of JOGL[10] binding to OpenGL specification, exhibits quite a lot sophisticated 3D featured projects (e.g. NASA World Wind[11]). To our best knowledge, research efforts rather focus on database connectivity issues (Hobona et al., 2006) or scientific analyses (Vance et al., 2005) than on animation or motion effects on geospatial objects.
- Google Earth, being one of the most popular APIs for building 3D mapping browser-based applications with JS has stopped being supported by dominant browsers for security reasons and Google announced its deprecation.[12]

Geospatial visualizations over the web are not a new topic (Kahkonen et al.,1999; Huang et al., 2001; Huang and Lin, 2002; Billen et al., 2008). However, the development of powerful and free JS frameworks and libraries on top of WebGL, compiled just in time by modern browsers, boosted up performance and allowed more complicated visualizations during last five years (Christen et al., 2012; Gesquière and Manin, 2012, Resch et al., 2014; Rumor et al., 2014; Krooks et al., 2014; Krämer and Gutbell, 2015).

Table 1 is an attempt to gather currently available WebGL visualization frameworks, libraries and projects and present their major visualization features and geospatial capabilities. Specifically, the 'animation' feature refers to the capability of loading animated 3D models and placing them on a 3D terrain, while the 'motion' feature refers to the capability of placing 3D models at different terrain positions in each rendering, simulating that way the movement of a 3D model. When, the motion of a model may be dynamically specified by the user, this is depicted in the table as 'interactive motion'.

Major GIS-based functionalities are evaluated through the following features:

- 'spatial reference' assigned to the 3D terrain through the transformation of the design dimensions in to a user specified coordinate system
- 'overlaying' of multiple thematic layers containing geospatial features or 3D models, with the aim to compose a 'Geospatial World'
- import of 'external feature data sources' with the aim to populate them with 3D models or animated textures, or to use them as specifiers of motion paths

[2] http://www.comscore.com/Insights/Press-Releases/2012/5/EU5-Map-Usage-via-Smartphone-Growing.
[3] https://www.khronos.org/webgl/.
[4] http://nationalmap.gov/standards/demstds.html.
[5] http://www.qgistutorials.com/en/docs/working_with_terrain.html.
[6] https://github.com/Prieston/3dav.
[7] http://threejs.org/.
[8] http://cesiumjs.org/.
[9] https://www.opengl.org/.
[10] http://jogamp.org/jogl/www/.
[11] http://worldwind.arc.nasa.gov/java/.
[12] http://googlegeodevelopers.blogspot.gr/2014/12/announcing-deprecation-of-google-earth.html.

Table 1
Features of WebGL based 3D geospatial visualization frameworks.

Name	Feature								
	type	visualization	Animation	motion	interactive motion	integrated geospatial functionalities			
						spatial reference	overlaying	external feature data sources	custom geospatial world
Three.js	Library	✓	✓	✓	✓				
Cesium	Library	✓	✓	✓		✓	✓		
Qgis2threejs	QGIS-plugin	✓				✓	✓		
X3DOM	Framework	✓	✓	✓	✓	✓	✓	✓	
OSG.js	API	✓	✓	✓	✓				
O3D	Project/ Library	✓	✓	✓	✓				
OpenWebGlobe	SDK	✓	✓			✓	✓	✓	
WebGL Globe	Platform	✓				✓			

- creation of a 'custom geospatial world' based on user-defined layers of features and 3D models

A thorough study of the selected software items resulted to the following conclusions strengthening the need for performing the presented work:

Three.js makes it possible to create complex 3D animations, geometries and impressive effects, while Cesium.js is ideal for creating globes and gathers plenty of advanced geospatial functionalities such as terrain visualization and imagery layers draw using geospatial web services and standards. Qgis2Threejs[13] is a python plugin of QGIS utilizing three.js library, which provides the capability of exporting terrain data, map canvas images and vector data to browsers supporting WebGL. Combining the above Javascript libraries it is possible to perform sophisticated 3D georeferenced visualizations with motion effects on 3D models. However interaction of spatial objects with external feature data sources towards creating custom geospatial worlds is not directly supported. O3D[14] is also a JS library implemented on top of WebGL, for creating rich, interactive 3D applications in the browser. OSGJS[15] is a WebGL API for creating 3D applications with Javascript, based on OpenSceneGraph[16] development philosophy. All the above mentioned components do not maintain the geospatial aspect of the involved entities.

X3DOM[17] is composed of X3D (Extensible 3D Graphics), which denotes a standard for declarative 3D graphics and DOM (Document Object Model), which describes hierarchy and interactivity issues associated with HTML content. Through this integration (Behr et al., 2009) it is possible to develop interactive 3D scenes, using a structured, textual representation. X3D Geospatial is a very recent development which promises a niche between GIS and 3D graphics and inherent support of spatial reference systems (Plesch and McCann, 2015). OpenWebGlobe[18] is a WebGL virtual globe solution for creating custom applications, large scale rendering and imagery data display and WebGL Globe[19] is a platform for visualizing spatially referenced data and sophisticated cartographies on the globe. Both of the above mentioned do not deal with motion effects on spatial objects.

The desired features of a visualization framework, with most important being the capability of creating custom 'living' geospatial worlds seem to be missing and therefore the vision of enriching existing findings remains unaltered. Though, many of the above mentioned software items provide significant functionalities the code homogenization and the integration of patchy clusters of code is a real bottleneck.

3. Method

The method adopted in the present work makes use of surveying engineering and geometry fundamentals to formulate the algorithmic base that document minimum functionalities for geospatial visualizations and motion effects on spatial objects.

3.1. Specifying the geometry

The Geometry should support representation of GIS digital terrain models which typically consist of a homogeneous grid of points in the 3D space. In this respect the Geometry, as shown in Fig. 1 is specified by:

- The extend of the grid as this is projected in XY plane provided by the minimum and maximum X, Y values (X_{min}, Y_{min}) and (X_{max}, Y_{max}),
- The dimensions of the grid as this is projected in XY plane provided by the number of its *columns* and *rows* and
- The elevation values of the projected in XY plane grid points beginning from the top left, crossing the grid by its rows and ending to the bottom right, provided by a one dimensional table *Z[]* maintaining coordinate values.

The Geometry employed is a *plane geometry* as documented in Three.js JS library.[20]

3.2. Locating a point over a motion path

Locating a point over a motion path is the most significant calculation to deal with when simulating the motion of a moving object between two given points of a 3D surface. This is because of the continuous changes that happen to the moving object's orientation as it passes through the triangles of the 3D surface. Similar calculation is required when, for example, fitting a polyline on a 3D terrain. At the present work, the 3D surface employed is regular, however, it should be stressed that such calculations are generalized in the case of irregular TIN (triangular irregular networks) surfaces.

In this paragraph what is requested is the location R as expressed by its coordinates R_x, R_y, R_z, of a point representing the position of an object in a time instance, moving on a specified path as this is expressed by its known starting $S(S_x, S_y, S_z)$ and ending $E(E_x, E_y, E_z)$ points. Point R always belongs to a triangle of the plane geometry, expressed by its vertices $A(A_x, A_y, A_z)$, $B(B_x, B_y, B_z)$ and $C(C_x, C_y, C_z)$ which is crucial to

[13] https://plugins.qgis.org/plugins/Qgis2threejs/.
[14] https://code.google.com/p/o3d/.
[15] http://osgjs.org/.
[16] http://www.openscenegraph.com/.
[17] http://www.x3dom.org/.
[18] http://www.openwebglobe.org/.
[19] https://www.chromeexperiments.com/globe.

[20] http://threejs.org/docs/#Reference/Extras.Geometries/PlaneGeometry.

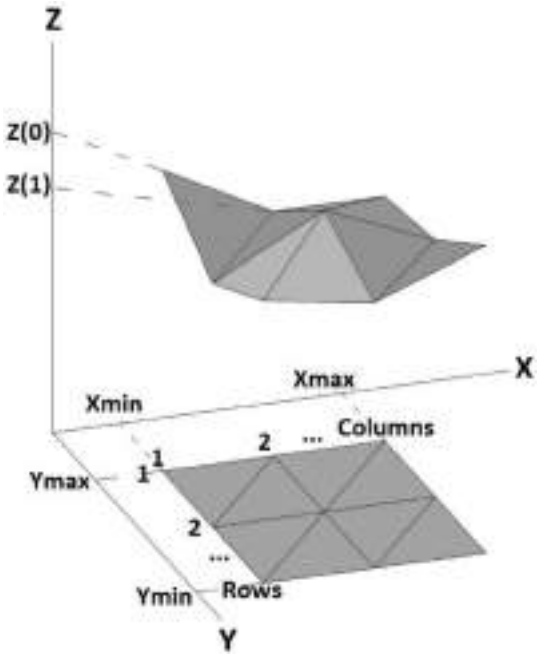


Fig. 1. Plane geometry.

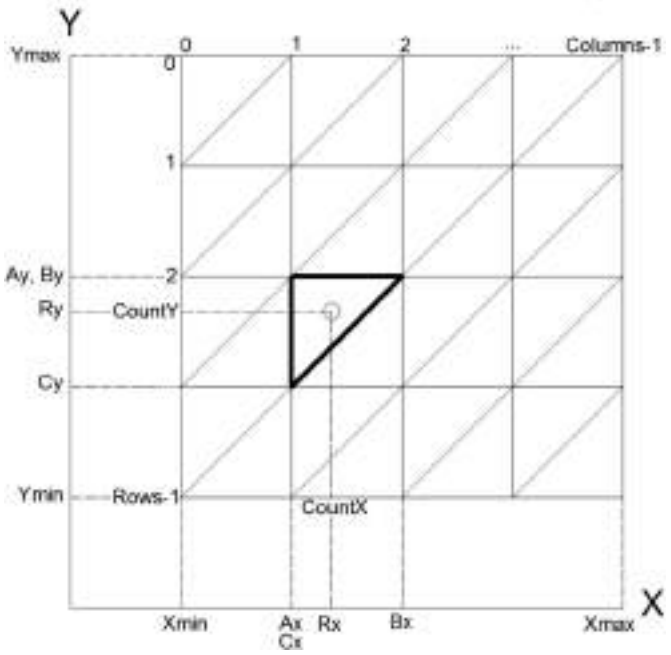


Fig. 2. Projection of plane geometry in XY plane.

be identified.

3.2.1. *Identifying the containing triangle*

The containing triangle ABC will be identified as a function of the known dimensions of the "Geospatial World" X_{min}, Y_{min}, X_{max}, Y_{max}, *columns*, *rows* and table *Z[]* which maintains elevation values (§3.1) and also the R_x and R_y components of the requested point R (§3.2.3).

Fig. 2 illustrates the world's projection on plane XY. Supposing that the requested point R is located inside triangle ABC it is possible to reach point R, by counting the intervening rectangles both horizontally (axis X) and vertically (axis Y) beginning from the top left one. Then, the number of the intervening rectangles - in real values – in axes X and Y are:

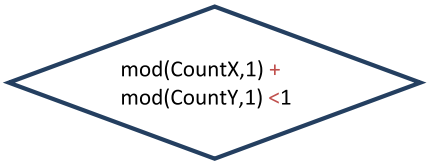
$$CountX = \frac{R_x - X_{\min}}{B_x - A_x} = \frac{R_x - X_{\min}}{Step(x)} = \frac{R_x - X_{\min}}{(X_{\max} - X_{\min})/columns}$$

$$CountY = \frac{Y_{\max} - R_y}{A_y - C_y} = \frac{Y_{\max} - R_y}{Step(y)} = \frac{Y_{\max} - R_y}{(Y_{\max} - Y_{\min})/rows}$$

where $(B_x - A_x)$ is the measure of the projection of triangle's line AB on plane XY and represents the grid step X_{step}, on axis X, $(A_y - C_y)$ is the measure of the projection of triangle's line AC on plane XY and represents the grid step Y_{step} on axis Y. These steps are equal to the ratio of the map extend (X_{max}-X_{min} and Y_{max}-Y_{min}) to the grid density (*columns* and *rows*) of the "Geospatial World".

The above *CountX* and *CountY* values reveal the rectangle on which the requested point R lies on. To find out on which of the two containing triangles is located (top left or bottom right) the following condition is applied:

'If the summary of the decimal parts of the above numbers is less than the unit then the requested point is located on the top left triangle ABC otherwise on the bottom right A'B'C', where B′ and C′ is identical to B and C respectively ':



$A_x = X_{min} + int(CountX)X_{step}$
$A_y = Y_{max} - int(CountY)Y_{step}$
$A_z = Z(int(CountY)Columns + int(CountX))$
$B_x = X_{min} + (int(CountX)+1)X_{step}$
$B_y = A_y$
$B_z = Z(int(CountY)Columns + int(CountX)+1)$
$C_x = A_x$
$C_y = Y_{max} - (int(CountY)+1)Y_{step}$
$C_z = Z((int(CountY)+1)Columns + int(CountX))$

$A'_x = X_{min} + (int(CountX)+1)X_{step}$
$A'_y = Y_{max} - (int(CountY)+1)Y_{step}$
$A'_z = Z((int(CountY)+1)Columns + int(CountX)+1)$
$B'_x = A'_x$
$B'_y = Y_{max} - int(CountY)Y_{step}$
$B'_z = Z(int(CountY)Columns + int(CountX)+1)$
$C'_x = X_{min} + int(CountX)X_{step}$
$C'_y = A'_y$
$C'_z = Z((int(CountY)+1)Columns + int(CountX))$

3.2.2. Calculating elevation R_z

The elevation R_z of the requested point will be calculated as a function of the vertices of the triangle ABC at which it is located (§3.2.1) and also its R_x and R_y components (§3.2.3). Two additional points P and Q, where initiated to assist this calculation and are defined as follows:

- P is the intersection of a line, which is parallel to triangle's hypotenuse and is crossing point R, with the triangle's side which is parallel to plane ZY
- Q is the intersection of a line, parallel to triangle's hypotenuse crossing point R, with the triangle's side which is parallel to plane ZX

Fig. 3 shows the requested point R along with points P and Q on triangle ABC as well as their projection on plane XY.

In the following documentation a segment is notated either geometrically, e.g. AB, or with its algebraic value, e.g. $d(A,B)$. Triangles $R_{xy}R'_{xy}Q_{xy}$ and $C_{xy}A_{xy}B_{xy}$ are similar; hence the ratios of the corresponding sides are equal:
$\frac{R'_{xy}Q_{xy}}{A_{xy}B_{xy}} = \frac{R'_{xy}R_{xy}}{A_{xy}C_{xy}}$ and the value of segment $R'_{xy}Q_{xy}$ is:

$$d\left(R'_{xy}, Q_{xy}\right) = \left|\left(R_y - A_y\right)\frac{\left(B_x - A_x\right)}{\left(C_y - A_y\right)}\right| \tag{1}$$

Therefore, the hypotenuse of the right triangle $R_{xy}R'_{xy}Q_{xy}$ (Fig. 4a) is:

$$d(R_{xy}, Q_{xy}) = \sqrt{\left(R_y - A_y\right)^2\left(1 + \left(\frac{\left(B_x - A_x\right)}{\left(C_y - A_y\right)}\right)^2\right)} \tag{2}$$

Similarly the value of segment $R''_{xy}Q_{xy}$ and the hypotenuse of the right triangle $R_{xy}R''_{xy}P_{xy}$ (Fig. 4a) is:

$$d\left(R''_{xy}, P_{xy}\right) = \left|\left(R_x - A_x\right)\frac{\left(B_y - A_y\right)}{\left(C_x - A_x\right)}\right| \tag{3}$$

$$d(R_{xy}, P_{xy}) = \sqrt{\left(R_x - A_x\right)^2\left(1 + \left(\frac{\left(C_y - A_y\right)}{\left(B_x - A_x\right)}\right)^2\right)} \tag{4}$$

Having equations (1) and (3) the values of Q_{xy}, P_{xy} can be calculated:

$$Q_x = R_x + d\left(R'_{xy}, Q_{xy}\right)$$
$$Q_y = Ay$$

$$P_x = A_x$$
$$P_y = R_y - d\left(R''_{xy}, P_{xy}\right)$$

Fig. 4 illustrates triangle projections on planes XZ and YZ.

Triangles $A_{xz}Q'_{xz}Q_{xz}$ and $A_{xz}B'_{xz}B_{xz}$ are similar; hence the ratios of the corresponding sides are equal:
$\frac{Q_{xz}Q'_{xz}}{A_{xz}Q'_{xz}} = \frac{B_{xz}B'_{xz}}{A_{xz}B'_{xz}}$ and the value of segment $Q_{xz}Q'_{xz}$ is:

$$d(Q_{xz}, Q'_{xz}) = \left|\left(Q_x - A_x\right)\frac{\left(B_z - A_z\right)}{\left(B_x - A_x\right)}\right|$$

Finally the Q_z is obtained by adding the above value at A_z, in case A_z is less than B_z, or at B_z in the opposite case:

$$Q_z = A_z + \left|\left(Q_x - A_x\right)\frac{\left(B_z - A_z\right)}{\left(B_x - A_x\right)}\right| \tag{5}$$

$$Q_z = B_z + \left|\left(Q_x - A_x\right)\frac{\left(B_z - A_z\right)}{\left(B_x - A_x\right)}\right| \tag{6}$$

Similarly P_z is calculated:

$$\frac{P_{yz}P'_{yz}}{C_{yz}P'_{yz}} = \frac{A_{yz}A'_{yz}}{C_{yz}A'_{yz}} \Rightarrow d\left(P_{yz}, P'_{yz}\right) = \left|\left(P_y - C_y\right)\frac{\left(A_z - C_z\right)}{\left(A_y - C_y\right)}\right|$$

P_z is calculated by adding the above value at C_z, in case is C_z is less than A_z, or at A_z in the opposite case:

$$P_z = C_z + \left|\left(P_y - C_y\right)\frac{\left(A_z - C_z\right)}{\left(A_y - C_y\right)}\right| \tag{7}$$

$$P_z = A_z + \left|\left(P_y - C_y\right)\frac{\left(A_z - C_z\right)}{\left(A_y - C_y\right)}\right| \tag{8}$$

Now, having calculated (equations (2) and (4)) the distances between the projected in plane XY points P, Q and R and also having the z components of points Q and P (equations (5)–(8)) it is possible to calculate the requested R_z component by equalizing the corresponding ratios of the projected line segments, as shown in Fig. 5:
$\frac{R_z - P_z}{d(R_{xy}, P_{xy})} = \frac{Q_z - P_z}{d(Q_{xy}, P_{xy})}$ and the requested elevation R_z is given by equation

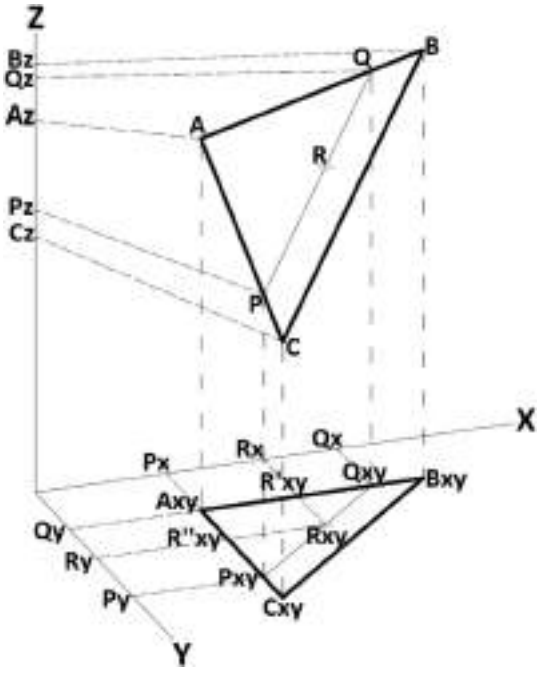


Fig. 3. Points R, P and Q over triangle ABC and their projections on plane XY.

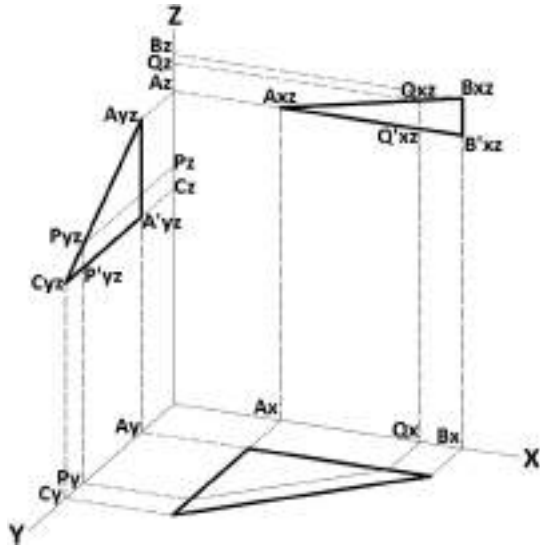


Fig. 4. Projections of triangle ABC.

(9) if P_z is less than Q_z, or equation (10) otherwise:

$$R_z = P_z + \left| \frac{d(R_{xy}, P_{xy})(Q_z - P_z)}{d(R_{xy}, Q_{xy}) + d(R_{xy}, P_{xy})} \right| \quad (9)$$

$$R_z = Q_z + \left| \frac{d(R_{xy}, P_{xy})(Q_z - P_z)}{d(R_{xy}, Q_{xy}) + d(R_{xy}, P_{xy})} \right| \quad (10)$$

To express elevation R_z as a function of the vertices of the triangle ABC at which it is located and also its R_x and R_y components the above equations are transformed by use of equations (2), (4) and (5)–(8) as follows:

$$R_x = S_x + d(S_{xy}, R_{xy}) \sin\left(a \tan\left(\frac{E_x - S_x}{E_y - S_y}\right)\right)$$

$$R_y = S_y + d(S_{xy}, R_{xy}) \cos\left(a \tan\left(\frac{E_x - S_x}{E_y - S_y}\right)\right)$$

To calculate $d(S_{xy}, R_{xy})$ the following assumption is considered: let the projected on plane XY distance traveled be equal to $d(S, R)$, as shown in Fig. 7.

Then, the elevation value R'_z may be calculated by employing the algorithm analyzed in paragraph 3.2.2. Subsequently, the slope of the object's orientation which is the angle formed by SE and plane XY, is equal to $a \tan\frac{R'_z - S_z}{d(S,R)}$ and finally the requested is:

$$R_z = \left(U_1 + \left|\left(R_y - \left|\left(R_x - A_x\right)\cdot\frac{(B_y - A_y)}{(C_x - A_x)}\right| - C_y\right)\cdot\left(\frac{A_z - C_z}{A_y - C_y}\right)\right|\right) + \left|\frac{\left[\sqrt{(R_x - A_x)^2\cdot\left(1 + \left(\frac{C_y - A_y}{B_x - A_x}\right)^2\right)}\right]\cdot\left[\left(U_2 + \left|\left(R_x + \left|\left(R_y - A_y\right)\cdot\frac{(B_x - A_x)}{(C_y - A_y)}\right| - A_x\right)\cdot\left(\frac{B_z - A_z}{B_x - A_x}\right)\right|\right) - \left(U_1 + \left|\left(R_y - \left|\left(R_x - A_x\right)\cdot\frac{(B_y - A_y)}{(C_x - A_x)}\right| - C_y\right)\cdot\left(\frac{A_z - C_z}{A_y - C_y}\right)\right|\right)\right]}{\left(\sqrt{(R_y - A_y)^2\cdot\left(1 + \left(\frac{(B_x - A_x)}{(C_y - A_y)}\right)^2\right)}\right) + \left(\sqrt{(R_x - A_x)^2\cdot\left(1 + \left(\frac{(C_y - A_y)}{(B_x - A_x)}\right)^2\right)}\right)}\right|$$

where $U_1 = C_z$ if ($C_z <= A_z$), A_z otherwise and $U_2 = A_z$ if ($A_z <= B_z$), B_z otherwise.

3.2.3. Calculating R_x and R_y components

Given the starting point S and the ending point E of an object's movement over a line segment SE and the measure $d(S, R)$ of the distance traveled, as derived from the speed of the object and the time lapse, the requested R_x and R_y components, are easily provided by simple geometric formulas, where $d(S_{xy}, R_{xy})$ is the projection of the slant distance $d(S, R)$ on plane XY, as shown in Fig. 6, below:

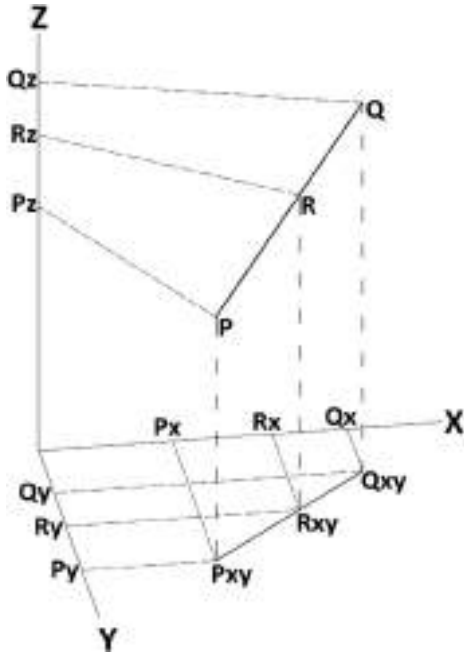


Fig. 5. P, Q and R projections on plane XY.

$$d(S_{xy}, R_{xy}) = d(S, R) \cos\left(a \tan\frac{R'_z - S_z}{d(S, R)}\right)$$

3.3. Transforming design coordinates to geographic

When defining the dimensions of the 'Geospatial World' the user is actually defines its spatial reference by setting its XYZ projected coordinate system. Whenever an external spatial dataset is imported, in order to be designed and overlaid with the "Geospatial World" a transformation of its spatial coordinates to design coordinates (pixels) takes place. On the other side whenever the user clicks on the "Geospatial World" in order to specify paths or insert 3D models, a reverse transformation happens. To transform the design coordinates in real world geographic coordinates the following equations were employed:

$$R_x = \left(R_{x(px)} + \frac{mapWidth_{(px)}}{2}\right) pixelSize + X_{\min}$$

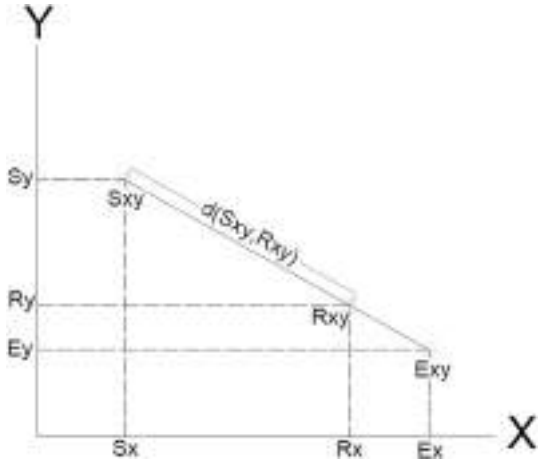


Fig. 6. Calculation of R_x and R_y components.

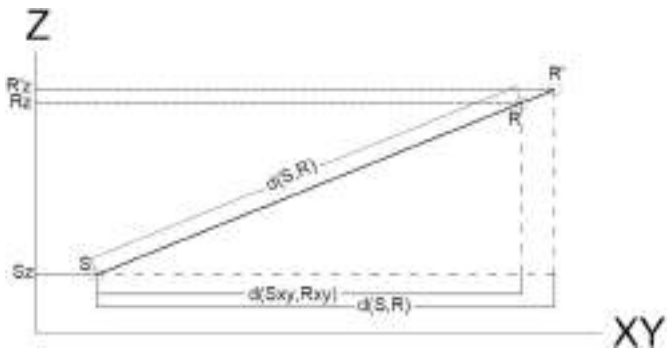


Fig. 7. Assumption considered for calculating $d(S_{xy}, R_{xy})$.

$$R_y = \left(R_{y(px)} + \frac{mapHeight_{(px)}}{2}\right) pixelSize + Y_{\min}$$

$$R_z = R_{z(px)} pixelSize$$

where *pixelSize* is easily determined by the "Geospatial World" dimensions:

$$pixelSize = \frac{X_{\max} - X_{\min}}{mapWidth_{(px)}} = \frac{Y_{\max} - Y_{\min}}{mapHeight_{(px)}}$$

The above are illustrated in Fig. 8.

3.4. Determining locations inside a polygon

To calculate coordinate values of point locations inside a polygon is useful in geospatial applications where an area has to be homogeneously populated with a set of objects (e.g. a cultivated land, a forest etc.). Given a polygon, firstly a surrounding rectangle is created by considering the minimum and the maximum values of the polygon vertices x_{min}, x_{max}, y_{min} and y_{max}. Then, the density placement of the point locations inside the polygon area is specified for both X and Y axis by values $step_x$ and $step_y$ respectively, as depicted in Fig. 9:

Then a number of line segments, depending on the desired density of the requested locations on Y axis, expressed by $step_y$, are designed parallel to X axis, and are defined by the following series of equations:

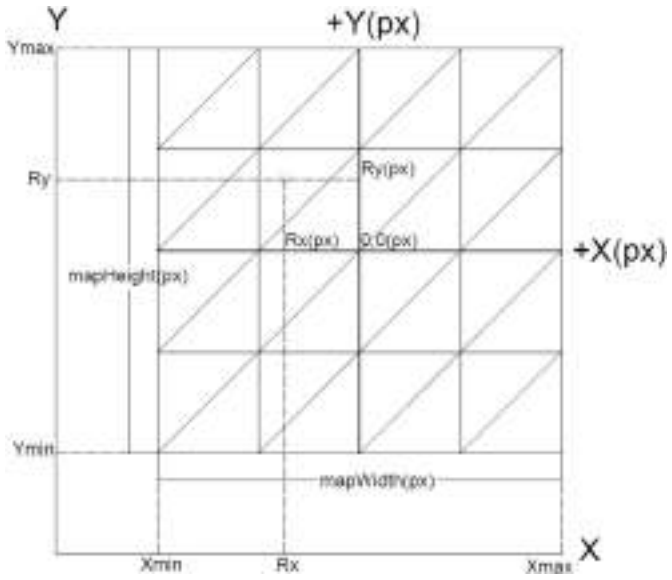


Fig. 8. Transformation between design and geographic coordinates.

$$y_i = b_i \text{ where } b_i = y_{\min} + istep_y$$

The intersection of the polygon segments ($y_j = a_j x_j + b_j$) with the aforementioned parallel lines, is expressed by the following group of equations:

$$y_i = y_j \Rightarrow b_i = a_j x_j + b_j \Rightarrow x_j = \frac{b_i - b_j}{a_j}$$

The solutions of the above group of equations provide a number of x_j values which may lead to the calculation of the corresponding y_j values. The results are filtered so that the intersection points belong to the specific polygon segments and not to the extensions of these segments along the lines to which they belong. To achieve this, a simple condition is applied, requiring that the coordinate values of the accepted intersection points, for each $step_y$, have to be between the coordinate values of the respective polygon segment endpoints. A table is created, populated with pairs of intersection points coordinates for each line parallel to X axis.

From the above mentioned table it is possible to retrieve the pair of intersection points forming each, parallel to axis X, segment: the *n-segment* is formed by the (*n*2-2*, *n*2-1*) pair of intersection points. The *pair of x values* for every, parallel to X axis, segment is sorted ascending and finally by considering the desired placement density on the X axis, expressed by $step_x$ value it is possible to calculate the final locations.

3.5. Calculating orientation changes on moving objects

A moving object is geometrically represented by a model and maintains a local coordinate system which is dynamically adjusted so that in every time instance (rendering), its X axis is always parallel to its orientation axis, as depicted in Fig. 10:

Orientation changes of a moving object are reflected to rotations around its axes and therefore occur in the following cases:

(1) Whenever the moving object is turning left or right, this is reflected with a rotation around the model's Z axis. In fact, such turnings occur every time the moving object reaches a vertex of its motion path. By knowing the coordinates of the vertices of the motion path (polar or Cartesian) it is easy to calculate the turning angle φ.
(2) Whenever the moving object is directed upwards or downwards, this is reflected with a rotation around the model's Y axis. In this case, the angle of slope φ, which is the angle formed by the model's

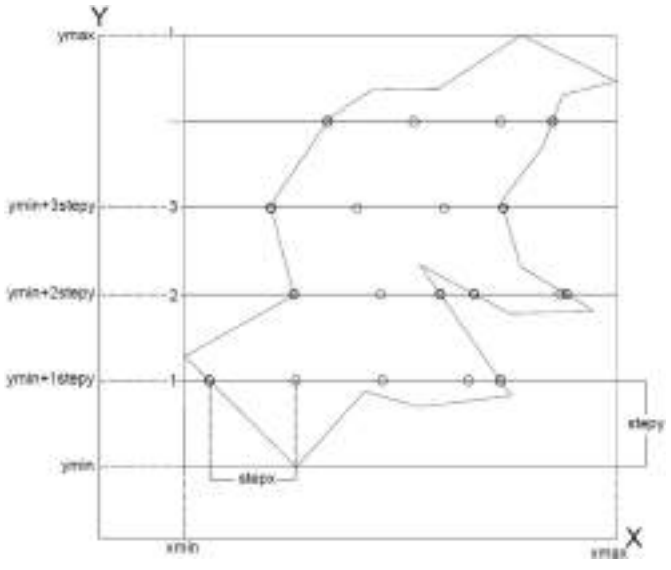


Fig. 9. Calculating locations inside a polygon.

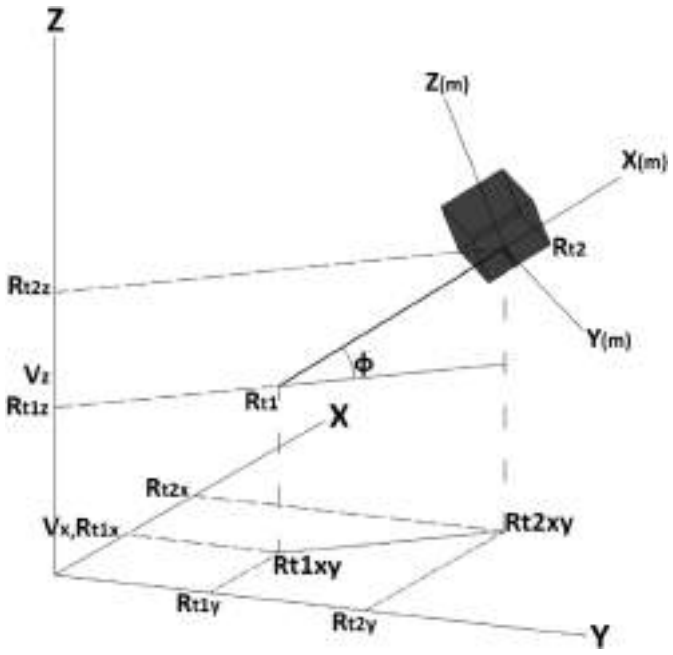


Fig. 10. Rotations around the axes of a model's local coordinate system.

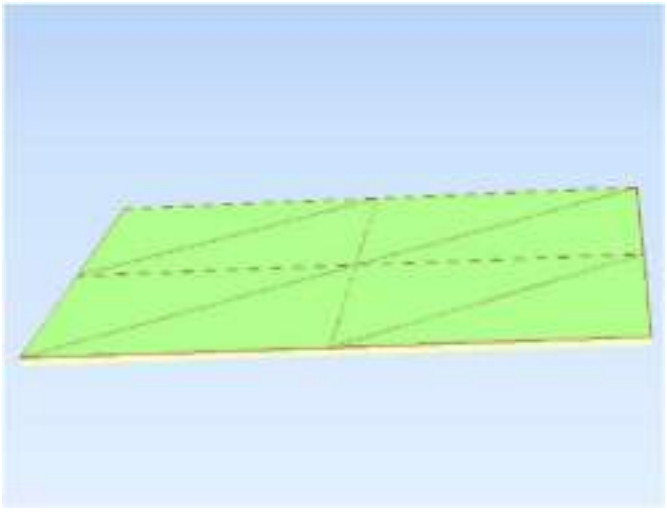


Fig. 11. A 2 × 2 plane geometry containing 9 grid points without elevation values.

orientation axis and axis Z of the global coordinate system, changes. In this case, the slope angle between two time instances t1 and t2, is identified by use of the fundamental surveying formula, $\tan\phi = \frac{R_{t2z} - R_{t1z}}{\sqrt{(R_{t2x} - R_{t1x})^2 + (R_{t2y} - R_{t1y})^2}}$ where the components of points R_{t1} and R_{t2} are calculated according to § 3.2. This formula is used to rotate only alive models (e.g. horse). For inanimate objects this kind of rotation is covered in the next case.

(3) Whenever the moving object is acting like an airplane that tilts when turning, this is reflected to a rotation around its model's X axis. Practically, this kind of rotation happens only in inanimate models (e.g. car), since the alive objects (e.g. man, horse etc.) are moving so that the horizontal component of their weight force tends to be zero. For the inanimate objects this rotation along with the rotation around their model's Y axis examined in the previous case, is satisfied through algorithms that provide the vertical vector of the surface they lie on. Such way these objects are always moving with their model's Z axis always vertical on the underlying triangle.

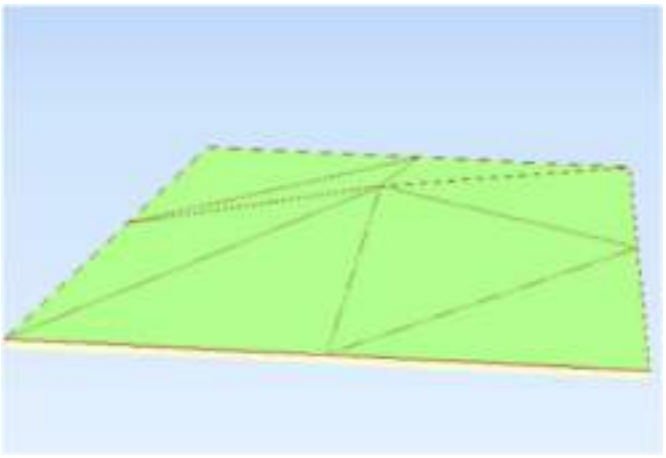


Fig. 12. Assigning elevation values.

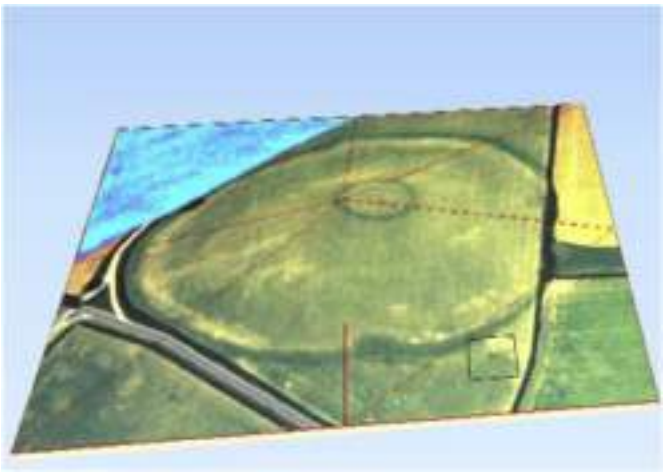


Fig. 13. Matching a texture simulating the geospatial world.

4. Application

The application of the method introduced in the previous paragraph includes the development of an experimental 'Geospatial World' in order to simulate on it, placement of stable living or inanimate spatial objects (e.g. plants, house) or motion of animated moving objects (e.g. animals).

4.1. Creating a 'geospatial world' for testing purposes

The creation of a 'Geospatial World' is the initial action required and practically includes the development of a 3D terrain upon which sophisticated visualizations and animation and motion effects on spatial objects, take place. Therefore, a testing 'Geospatial World' with an extend of width 100 and height 50 pixels and dimensions of 2 columns and 2 rows, contains 9 grid points as shown in Fig. 11

By assigning to the fifth point (central point) a random elevation value and to all other grid points, the value of zero results to the 3D terrain shown in Fig. 12.

Finally the texture of the geospatial world created is loaded via a raster image and arranged appropriately by matching its pixels with the corresponding grid nodes of the geometry (Fig. 13).

To make things simple, it should be noted that the real world dimensions of the 'Geospatial World' is 100 m width and 50 m height,

Fig. 14. Oversized model exceeding map limits is scaled to fit specified boundary box.

Fig. 15. Placing models at user-selected positions.

Fig. 16. Placing models at specified polylines.

Fig. 17. Populating an area with 3D models.

Fig. 18. Object moving over motion paths.

resulting to a pixel size of 1 m.

4.2. Scaling 3D models to fit world dimensions

Once the 'Geospatial World' is created, next step is to collect the desired spatial objects that are going to populate it. Plenty of 3D models representing inanimate or lively animated objects of real world may be found around [21,22]. As expected, the models have been deployed in arbitrary scale, resulting to a size which is normally disproportional to the geographic dimensions of the world to which they are about to be placed. For this reason each time a 3D model is inserted to the 'Geospatial World' it is resized appropriately in order to occupy a reasonable space according to its actual dimensions.

To achieve this, a bounding box with dimensions capable to fit the inserted model is created. For example in the case of a horse the box is 2.5 m wide which corresponds to an average horse length. Then the model under scaling is inserted so that it is clear with naked eyes if the size of the model fits within the box. As long as the inserted model exceeds the predefined dimensions, an automatic size reduction is performed, until it is completely contained within the bounding box as shown in Fig. 14.

Further techniques for simplifying the related mesh after resizing may be employed (Luebke, 2003), however such a process is out of the scope of the present work.

4.3. Placing models at selected locations

Selected locations are considered those resulting from user targeted or random clicks over the terrain or those imported by an external geospatial file containing point features. Selected locations are then populated with the desired 3D model (Fig. 15).

[21] http://blender-archi.tuxfamily.org/Greenhouse.
[22] http://www.turbosquid.com.

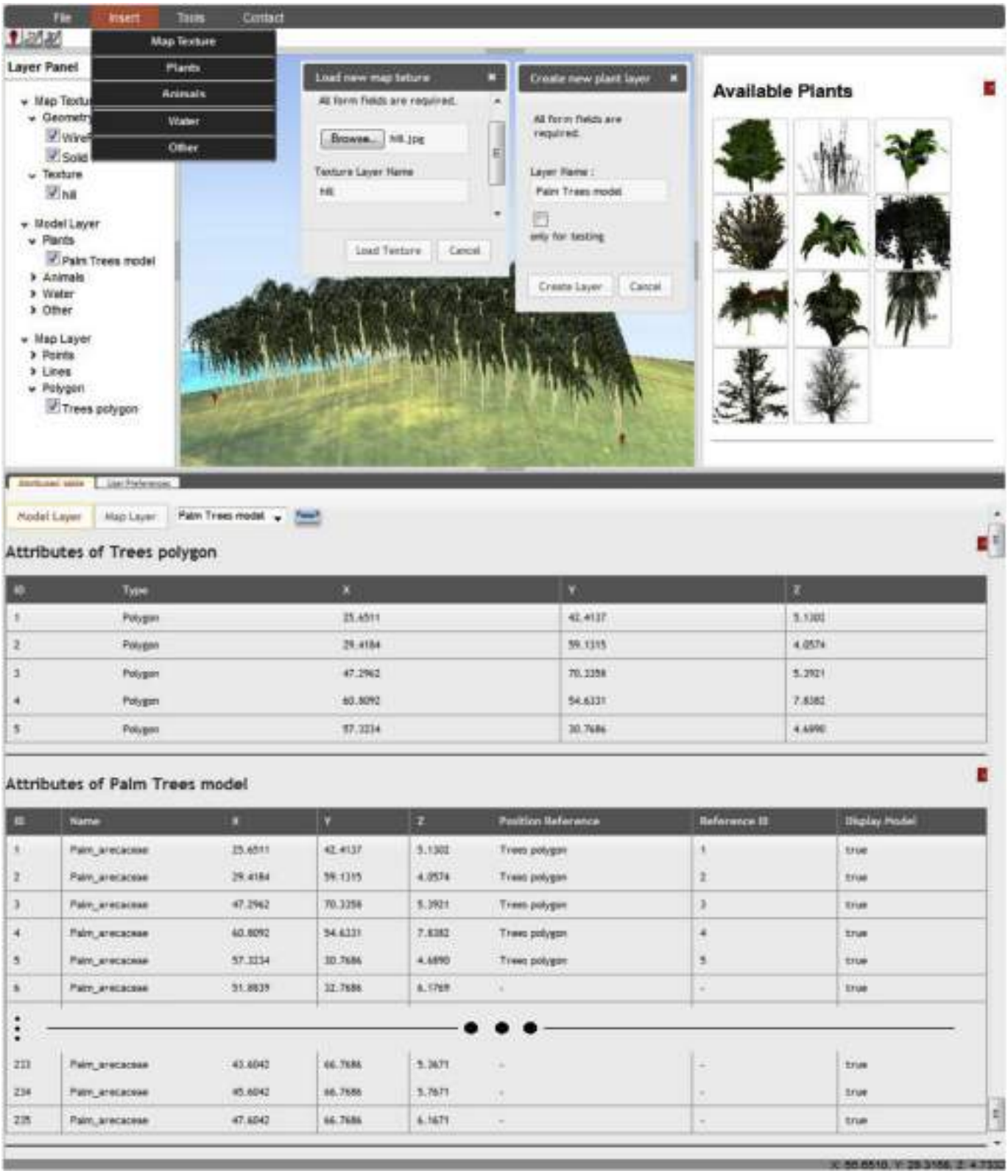


Fig. 19. Major GIS-based functionalities.

Such functionality is partially offered by Three.js library functions 'Raycaster'[23] and 'Loader'.[24] However, in order to store the exact

[23] http://threejs.org/docs/#Reference/Core/Raycaster.

[24] http://threejs.org/docs/#Reference/Loaders/JSONLoader.

Fig. 20. Some snapshots of the demonstrative 'Geospatial World'.

geographic coordinates of the located models, or in order to place a set of external geographic points in the appropriate pixels of the terrain, a transformation of the design coordinates to geographic coordinates and vice versa is required.

4.4. Placing models along polylines

Specified polylines are considered either those resulting by user clicks or those inserted via spatial datasets. Given the coordinates of the polyline vertices a number of intermediate points are identified and altogether are populated with the desired 3D model (Fig. 16).

4.5. Placing models inside polygons

With this functionality the user is capable of populating a polygon area, either manually by clicking in the desired polygon vertices or via a spatial dataset containing polygon features, with models of his choice. The method introduced in paragraph 3.4 was employed to implement this functionality (Fig. 17).

4.6. Implementing motion effects

The ultimate geospatial functionality with the seemingly most realistic simulation achievement is that of multiple animated 3D models (e.g. horses and cars) moving over user specified (or externally predefined) motion paths (Fig. 18). This functionality is implemented by use of the methodological approach introduced for locating a point over a motion path (§ 3.2) combined with that for calculating changes in the orientation of a moving object (§ 3.5). Especially in the case of inanimate objects (e.g. car) the composite rotation around their model's X and Y axis is

implemented by employing “look at” and “up” properties of a 3D object provided by three.js [25].

4.7. Incorporating GIS-based functionalities

The above deployed individual functionalities have to be offered in a way that the 'Geospatial World' creator can combine different 3D model types with various motion effects on dynamically defined paths. Major GIS functionalities are employed for this reason providing typical capabilities of importing feature layers, defining coordinate system, specifying points, paths and areas and overlaying the thematic layers. Furthermore, capabilities of manipulating attribute tables and managing 3D models are offered as shown in Fig. 19.

4.8. Combining all to make living 'geospatial worlds'

All functionalities so far implemented provide excellent capabilities for creating special 'Geospatial Worlds' with animated or fixed, stable or moving 3D spatial objects (Fig. 20). To boost up performance, object pooling pattern (Kircher and Jain, 2002) was adopted.

To assist demonstration of the presented work a video has been prepared and is available at:

https://www.youtube.com/watch?v=f5x-XRCxml8&feature=youtu.be.

5. Discussion and further developments

The present work employed the technologies related to 3D computer graphics and the existing visualization frameworks and libraries to achieve sophisticated geospatial world simulations associated with major GIS functionalities. There are of course several other interrelated issues beyond the scope and the central intentions of the whole venture that raise reasonable concerns. First of all, the algorithmic calculations were performed over a regular 3D terrain while the general case would be to be performed over a TIN. In addition there are many interesting level of detail (LOD) terrain algorithms, such as ROAM (Turner, 2000), for creating a land which is a major issue in 3D virtual worlds. This paper is not focused on providing the best algorithm for terrain creation. However, Three.js library supports almost all types of geometries and some of them can implement various algorithms including ROAM.[26] There are also various other interesting algorithms for searching the location of a point, calculating distances, getting the bounding box of a randomly positioned point, etc[27] and most of them have been written in C programming language for its memory handling capabilities. The presented is a purely surveying approach employing basic trigonometry for calculating distances, rotations etc., and aimed to make these algorithms available in the JS community.[28] The selection of JS was based due to its ability to perform directly over the web and be interpreted just in time by the web browser. Testing and comparing the presented algorithms with other existing and/or transforming the last mentioned in JS, is a real challenge and interesting subject of a future work, dealing with time processing issues. Additional similar issues, concerning processing capacity and visualization efficiency may be examined. For example, designing a complicated scene with plenty of moving objects after locating their position in each rendering is a costly process. Resizing techniques based on the locations of objects with regard to the gravity center, or bilinear interpolation would simplify things. Other simplifications processes may increase processing capacity such as for example mesh simplification after 3D models resizing.

A significant advantage of the above mentioned technologies coupling is the interaction between external feature layers (e.g. areas, paths, points in KML format) with 3D model layers. For example the homogeneous filling of spatial objects in a polygon area may not be actually met. In such real cases a further subdivision of the area in subareas with different density values for the requested 3D model, specified by an external feature layer may result to more realistic views of the geospatial world.

Finally, the extended geospatial user capabilities may be proved utilitarian for a wide range of multi-disciplinary activities which may be offhandedly classified into two general sections: a) navigation, and b) simulation. Regarding the navigational activities these may include visualizations of natural resources or historical events for educational purposes. For example, one could demonstrate troops movements between involved countries, during a significant World War II fight, by simply overlaying multiple layers of 3D military-related models moving over the globe in specified paths. Simulation activities may include modeling phenomena with significant research interest for geosciences such as those concerning floods, landslides, earthquakes etc. For example, the simulation of a strong rainfall and the progressive covering of the basin with water and its potential overflow, or the fall of a rock from a hill due to a landslide generate extreme geospatial visualization, animation and motion challenges. However to implement this, requires a significant enrichment of the existing JS libraries with special functionalities provided by physics engines, and this reveals an additional challenge of the presented work.

Acknowledgements

The authors wish to acknowledge financial support provided by the Research Committee of the Technological Educational Institute of Central Macedonia under grant SAT/GS/171214-263/23.

References

Batty, M., Hudson-Smith, A., Milton, R., Crooks, A., 2010. Map mashups, Web 2.0 and the GIS revolution. Ann. GIS 16 (1), 1–13.

Behr, J., Eschler, P., Jung, Y., Zöllner, M., 2009. X3DOM: a DOM-based HTML5/X3D integration model. In: Spencer, S.N. (Ed.), Proceedings of the 14th International Conference on 3D Web Technology. ACM, Darmstadt, Germany. New York, pp. 127–135, 16-17 June 2009.

Billen, M.I., Kreylos, O., Hamann, B., Jadamec, M.A., Kellogg, L.H., Staadt, O., Sumner, D.Y., 2008. A geoscience perspective on immersive 3D gridded data visualization. Comput. Geosci. 34 (9), 1056–1072.

Christen, M., Nebiker, S., Loesch, B., 2012. Web-based large-scale 3D-geovisualisation using WebGL: the OpenWebGlobe project. Int. J. 3 D Inf. Model. (IJ3DIM) 1 (3), 16–25.

Eha, B., 2013. An Accelerated History of Internet Speed (Infographic). Enterpreneur. Available from: http://www.entrepreneur.com/article/228489. (Accessed 7 November 2015).

Evangelidis, K., Ntouros, K., Makridis, S., Papatheodorou, C., 2014. Geospatial services in the cloud. Comput. Geosci. 63, 116–122.

Gesquière, G., Manin, A., 2012. 3D visualization of urban data based on CityGML with WebGL. Int. J. 3 D Inf. Model. (IJ3DIM) 1 (3), 1–15.

Haklay, M., Singleton, A., Parker, C., 2008. Web mapping 2.0: the neogeography of the GeoWeb. Geogr. Compass 2 (6), 2011–2039.

Hobona, G., James, P., Fairbairn, D., 2006. Web-based visualization of 3D geospatial data using Java3D. Comput. Graph. Appl. IEEE 26 (4), 28–33.

Huang, B., Jiang, B., Li, H., 2001. An integration of GIS, virtual reality and the internet for visualization, analysis and exploration of spatial data. Int. J. Geogr. Inf. Sci. 15 (5), 439–456.

Huang, B., Lin, H., 2002. A Java/CGI approach to developing a geographic virtual reality toolkit on the internet. Comput. Geosci. 28 (1), 13–19.

Kahkonen, J., Lehto, L., Kilpelainen, T., Sarjakoski, T., 1999. Interactive visualisation of geographical objects on the internet. Int. J. Geogr. Inf. Sci. 13 (4), 429–438.

Kircher, M., Jain, P., 2002. Pooling pattern. In: Proceedings of the 7th European Conference on Pattern Languages of Programms (EuroPLoP '2002), 3–7 July 2002. UVK - Universitaetsverlag Konstanz, Irsee, Germany, ISBN 978-3-87940-784-2, 2003.

Krämer, M., Gutbell, R., 2015. A case study on 3D geospatial applications in the web using state-of-the-art WebGL frameworks. In: Proceedings of the 20th International Conference on 3D Web Technology. ACM, Heraklion, Greece. New York, pp. 189–197, 18–21 June 2015.

Krooks, A., Kahkonen, J., Lehto, L., Latvala, P., Karjalainen, M., Honkavaara, E., 2014. WebGL visualisation of 3D environmental models based on Finnish open geospatial data sets. ISPRS Int. Arch. Photogrammetry Remote Sens. Spatial Inf. Sci. 1, 163–169.

Luebke, D.P., 2003. Mesh Simplification. Level of Detail for 3D Graphics. Morgan Kaufmann, pp. 38–44.

[25] http://threejs.org/docs/#Reference/Core/Object3D.
[26] https://threejs.org/docs/index.html#Reference/Core/BufferGeometry.
[27] https://www.geometrictools.com/Samples/Geometrics.html.
[28] https://github.com/Prieston/3dav.

Oxera, 2013. What is the economic impact of geoservices? Prepared for Google. Available from: http://www.oxera.com/Latest-Thinking/Publications/Reports/2013/What-is-the-economic-impact-of-Geo-services.aspx. (Accessed 7 November 2015).

Plesch, A., McCann, M., 2015, June. The X3D geospatial component: X3DOM implementation of GeoOrigin, GeoLocation, GeoViewpoint, and GeoPositionInterpolator nodes. In: Proceedings of the 20th International Conference on 3D Web Technology. ACM, pp. 31–37.

Resch, B., Wohlfahrt, R., Wosniok, C., 2014. Web-based 4D visualization of marine geo-data using WebGL. Cartogr. Geogr. Inf. Sci. 41 (3), 235–247.

Rumor, M., Roccatello, E., Scottà, A., 2014. A standard-based framework for real-time 3D large-scale geospatial data generation and visualisation over the web. In: Lee, D.J., Dias, E., Scholten, H.J. (Eds.), Geodesign by Integrating Design and Geospatial Sciences. Springer International Publishing, Switzerland, pp. 259–269.

Turner, B., 2000, April 3. Real time dynamic LOD terrain render with ROAM. Retrieved from http://www.gamasutra.com/view/feature/131596/realtime_dynamic_level_of_detail_.php. .

Vance, T.C., Merati, N., Moore, C., 2005. Integration of Java and GIS for visualization and analysis of marine data. Int. Arch. Photogrammetry Remote Sens. Spatial Inf. Sci. ISPRS 239–281.