Computers & Geosciences 72 (2014) 176–183

Contents lists available at ScienceDirect

Computers & Geosciences

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

Moving KML geometry elements within Google Earth

Liang-feng Zhu*, Xi-feng Wang, Xin Pan

Key Laboratory of GIS, East China Normal University, Shanghai 200241, China

ARTICLE INFO

Available online 1 August 2014

Keywords:
Virtual globe
Google Earth
KML
Moving transformation
Geometry element

ABSTRACT

During the process of modeling and visualizing geospatial information on the Google Earth virtual globe, there is an increasing demand to carry out such operations as moving geospatial objects defined by KML geometry elements horizontally or vertically. Due to the absence of the functionality and user interface for performing the moving transformation, it is either hard or impossible to interactively move multiple geospatial objects only using the existing Google Earth desktop application, especially when the data sets are in large volume. In this paper, we present a general framework and associated implementation methods for moving multiple KML geometry elements within Google Earth. In our proposed framework, we first load KML objects into the Google Earth plug-in, and then extract KML geometry elements from the imported KML objects. Subsequently, we interactively control the movement distance along a specified orientation by employing a custom user interface, calculate the transformed geographic location for each KML geometry element, and adjust geographic coordinates of the points in each KML objects. And finally, transformed KML geometry elements can be displayed in Google Earth for 3D visualization and spatial analysis. A key advantage of the proposed framework is that it provides a simple, uniform and efficient user interface for moving multiple KML geometry elements within Google Earth. More importantly, the proposed framework and associated implementations can be conveniently integrated into other customizable Google Earth applications to support interactively visualizing and analyzing geospatial objects defined by KML geometry elements.

1. Introduction

Nowadays a series of sophisticated and powerful online virtual globes, led by Google Earth, are becoming reliable platforms for communicating, analyzing and sharing geospatial information that is large-extent, multi-scaled, multi-source, massive and heterogeneous (Butler, 2006; Bailey and Chen, 2011; Goodchild et al., 2012; Yu and Gong, 2012). The Google Earth virtual globe has been widely embraced by earth scientists, educators, government officials and the general public as an important and everyday tool to conduct research, exchange ideas and share knowledge with a global perspective in a natural and intuitive way, mainly because it possesses the ability to support the OpenGIS KML Encoding Standard (OGC KML) (Wilson, 2008; Ballagh et al., 2011; De Paor et al., 2012; Lee and Guertin, 2012). As an open data standard for encoding representations of geospatial information visually, KML enables users of Google Earth or other geo-browsers to add custom geospatial data to virtual globes in a variety of formats, and allows users to create a variety of powerful user interface controls that interact with their own data (De Paor and Whitmeyer, 2011). Without having to develop a more sophisticated 3D/4D virtual environment from the low level, users of Google Earth only need to describe and save their own geospatial information in formats according to the OpenGIS KML Encoding Standard, and the Google Earth virtual globe can effectively load and vividly visualize that information over the Internet (Zhu et al., 2014).

In recent years, several earth scientists and industry developers have launched a series of explorations on how best to model and visualize geospatial information within Google Earth using KML (Whitmeyer et al., 2010; Bailey et al., 2012; Stewart and Baldwin, 2012; Martínez-Graña et al., 2013; Wang et al., 2013). For example, De Paor and Whitmeyer (2011) have described a variety of techniques and methods through which KML can be used to control the visualization of geological and geophysical data on Google Earth, and presented a method to create dynamic models that illustrate the internal structure of the Earth by using COLLADA and JavaScript. Postpischl et al. (2011) addressed the problem of the standardization and visualization of seismic tomographic models and earthquake focal mechanisms data sets using web technologies and KML. Blenkinsop (2012) has used a macro-enabled Excel workbook to convert field data into KML documents for the purpose of representing structural geology in Google Earth. Zhu et al. (2014) presented an automatic method for modeling and visualizing large volume of borehole information on Google Earth using KML. The above-mentioned advances have achieved successes to a greater or lesser extent in specific fields of use, which

* Corresponding author. Tel.: +86 13671721009.
E-mail addresses: lfzhu@geo.ecnu.edu.cn (L.-f. Zhu), wangxifeng0817@163.com (X.-f. Wang), xpan@admin.ecnu.edu.cn (X. Pan).

http://dx.doi.org/10.1016/j.cageo.2014.07.016
0098-3004/

also played significant roles in promoting the development and professional application of the Google Earth virtual globe.

After a thorough evaluation of the applications, we have detected several serious limitations when using the existing Google Earth desktop application and its user interface controls to model, visualize and analyze geospatial information. One of the current main shortcomings of Google Earth is its inability to represent the underground space of the Earth (De Paor and Whitmeyer, 2011; Navin and de Hoog, 2011). Technically speaking, Google Earth only provides a 2.5D digital globe that ideally suited for the modeling, visualization and analysis of geospatial information relevant to the Earth's surface and near-surface. Therefore, it is unable to directly display or analyze subsurface objects/phenomena/processes in the correct locations beneath the Earth's surface. In order to model, visualize and interpret subsurface features, some elegant tricks need to be designed and applied to bring subsurface information into view. Recently, two techniques have been developed to address the needs of visualizing the subsurface. One technique is to set an uplifted height value for the purpose of elevating the vertical position of subsurface features and to make them visible above the Earth's terrain surface (De Paor and Whitmeyer, 2011; Zhu et al., 2014). This is a static way of elevating subsurface models into view, and has the advantage of easy to implement. However, in this way, it is either hard or impossible to interactively move subsurface features because their altitudes would be fixed after lifted. A second approach is to pre-generate a set of KML objects containing KML ⟨TimeSpan⟩ tags with sequential increased altitudes, and to exploit the built-in Google Earth time slider control to raise objects up out of the surface (De Paor and Pinan-Llamas, 2006; De Paor and Williams, 2006; De Paor et al., 2008; De Paor and Whitmeyer, 2011; Dordevic, 2012). This approach provides high quality animation capability with visually appealing appearances, but suffers from the huge data redundancy. Moreover, the built-in Google Earth time slider is originally designed for controlling time intervals of displaying KML objects, thus it is not appropriate to elevate subsurface models. During the elevating process, the double-thumb feature and the display of values/units for elevation in the time slider often cause users confusion (Dordevic, 2012). Therefore, this approach is useful but should not be regarded as a perfect method for the modeling and visualization of subsurface features.

To overcome the above limitations on visualizing the subsurface, a more straightforward, intuitive and accessible solution can be obtained by moving subsurface features interactively within Google Earth to change their locations in 3D space. However, the current Google Earth desktop application, which falls short of advanced functions in 3D interaction for geospatial objects, is unable to offer existing tools or built-in functions to move geospatial objects freely and flexibly, especially when the objects are massive. Using the stand-alone Google Earth desktop application, users can choose and move a single KML geometry element, but they are unable to pick or move multiple geometry elements simultaneously. In addition, users cannot precisely control the movement distance through the user interface provided by the standard Google Earth desktop application. This shortcoming seriously limits the use of Google Earth in some specific Earth science subjects (especially geology and geophysics) and multidisciplinary research, and there is a pressing need for a more specialized tool to move geospatial objects within Google Earth.

Recently, some research teams have invested considerable effort into how to move geospatial objects within Google Earth, and several technologies have been proposed and applied by geologists and engineers. For example, Whitmeyer (2012) has written a Perl script to move points, lines and polygons in a KML file by a specified number of meters without distorting the shapes of geospatial objects. This script can be utilized to move continental components, crustal fragments and other geological structures for tectonic reconstructions. However, due to the limitation of the implementation program, this script seemed to lack the flexibility since it only supports the command-line (console) operation to process KML files. That is, users could neither move KML objects interactively nor control the movement distance through an efficient graphical user interface. By using the Google Earth web plug-in and its JavaScript application programming interface (API), Dordevic (2013) has designed a webpage (http://www.digitalplanet.org/API/SOS/index.html), which embeds interactive screen overlays as custom sliders, to control COLLADA models to emerge from the subsurface. This webpage has proven to be quite effective for handling 3D COLLADA models, but it could not be expanded to deal with other types of geospatial objects. In brief, up to now there are still no comprehensive methods or systematic applications for moving various types of geospatial objects within Google Earth. Therefore, there is a clear need for developing a universal method to handle all types of KML objects.

In this paper, we explore the transformation techniques and associated implementation methods for moving KML geometry elements within Google Earth. Our ultimate goal is to present a general framework for the moving transformation, which is suitable to deal with all types of KML geometry elements freely and flexibly. This paper first summarizes the classification and the description of KML geometry elements, as well as the method for defining their geospatial positions. Subsequently, the overall framework and key steps for performing the moving transformation are illustrated in great detail. The implementation program and its application are finally presented.

2. KML geometry elements

In digital globes, geospatial objects are generalized as points, lines, polygons and other types of geometry elements. As listed in Table 1, OGC KML 2.2 (Wilson, 2008) supports six types of geometry elements derived from the abstract ⟨kml:Geometry⟩ element, including five primitive geometry elements (⟨kml:Point⟩, ⟨kml:LineString⟩, ⟨kml:LinearRing⟩, ⟨kml:Polygon⟩ and ⟨kml:Model⟩) and one multiple geometry element (⟨kml:MultiGeometry⟩). For ⟨kml:Point⟩, ⟨kml:LineString⟩, ⟨kml:LinearRing⟩ and ⟨kml:Polygon⟩ elements, their positions on the Earth are defined by the ⟨kml:coordinates⟩ element (Wernecke, 2009). The initial placement of the ⟨kml:Model⟩ element is specified by the ⟨kml:Location⟩ element, and the ⟨kml:Location⟩ element contains the ⟨kml:longitude⟩, ⟨kml:latitude⟩ and ⟨kml:altitude⟩ child elements. The ⟨kml:MultiGeometry⟩ element is a container for zero or more geometric primitives associated with the same KML feature (Wilson, 2008; Wernecke, 2009). Therefore, it can be decomposed into a set of primitive geometry elements like ⟨kml:Point⟩, ⟨kml:LineString⟩, ⟨kml:LinearRing⟩, ⟨kml:Polygon⟩ or ⟨kml:Model⟩, and the position of each primitive geometry element is specified by the corresponding ⟨kml:coordinates⟩ or ⟨kml:Location⟩ element.

3. General framework and key steps

The existing Google Earth desktop application is insufficient to perform the moving transformation as it neither supports moving multiple objects simultaneously nor provides precise controls on movement distances. In order to move multiple KML geometry elements freely and flexibly, we need to transition from the Google Earth desktop application to the Google Earth web-browser plug-in, design and develop a custom component and corresponding operating interface by employing the Google Earth plug-in and its JavaScript API.

Table 1
KML geometry element types.

KML geometry element	Description	Define method for geospatial position
⟨kml:Point⟩	0-Dimensional geometric primitive, representing a spatial position.	A spatial location defined by a single geodetic longitude, geodetic latitude, and (optional) altitude coordinate tuple.
⟨kml:LineString⟩	Curve composed of a connected set of straight-line segments.	A list of two or more coordinate tuples. Each tuple contains the longitude, latitude, and (optional) altitude.
⟨kml:LinearRing⟩	Closed line string that should not cross itself, typically the outer or inner boundary of a polygon.	A list of four or more coordinate tuples where the first and last coordinate tuples must be the same (to form the closed loop). Each tuple contains the longitude, latitude, and (optional) altitude.
⟨kml:Polygon⟩	Planar surface defined by 1 exterior boundary and 0 or more interior boundaries.	One outer boundary ring and zero or more inner boundary rings. Each ring is defined by the ⟨kml:LinearRing⟩ element.
⟨kml:Model⟩	3D object described in a COLLADA file.	The exact coordinates of the initial placement of a ⟨kml:Model⟩ element is specified by the ⟨kml:Location⟩ element, comprising the ⟨kml:longitude⟩, ⟨kml:latitude⟩ and ⟨kml:altitude⟩ child elements.
⟨kml:MultiGeometry⟩	Container for zero or more geometric primitives associated with the same KML feature.	A collection of discrete geometric primitives listed above.

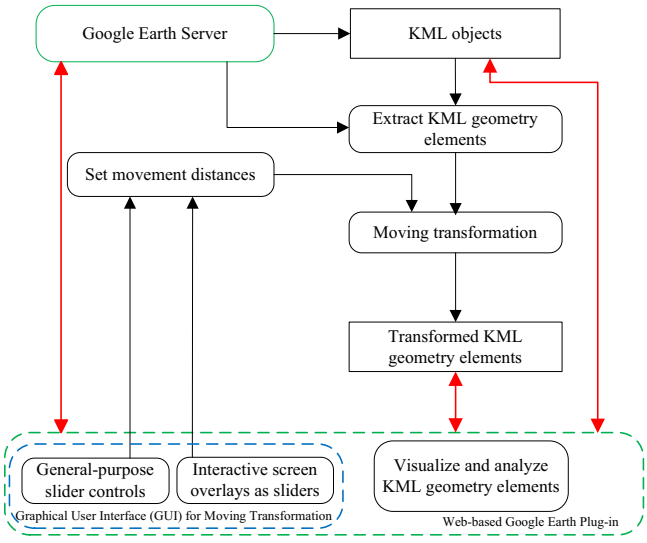

Fig. 1. Overall framework for the moving transformation of KML geometry elements within Google Earth. The sharp-cornered rectangles represent the KML objects; the round-cornered rectangles represent the program components that process the KML objects; the thin black arrow lines denote the data flows in the moving transformation; and the red double-headed arrows depict the graphical user interface controls and interaction between the program components and its user. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

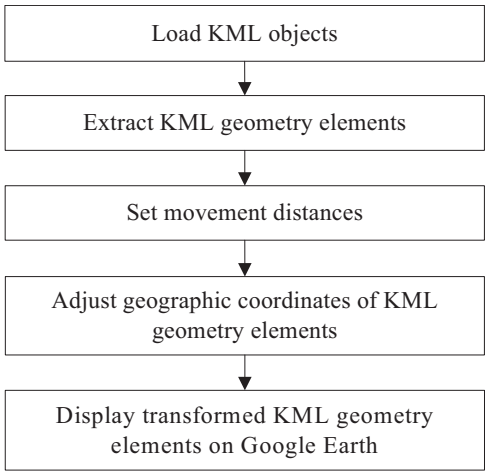

Fig. 2. Flow chart for implementing the moving transformation of KML geometry elements within Google Earth.

Fig. 1 shows a schematic representation of the overall framework for the moving transformation of KML geometry elements within Google Earth. As Fig. 2 shows, the implementation of the moving transformation can be decomposed into five steps, and the step-by-step execution is explained below.

3.1. Load KML objects

The first step is to load KML objects into the Google Earth plug-in, as initial input data for the moving transformation. KML objects are usually stored in a KML string or a hosted KML/KMZ file. In Google Earth API (Google, 2014), there are three methods of importing KML objects into the Google Earth plug-in: *KmlNetworkLink*, *fetchKml* and *parseKml* (Nurik, 2009). The *KmlNetworkLink* and *fetchKml* methods can be used to load a hosted KML/KMZ file from a specified URL, while the *parseKml* method is ideally suited for loading an arbitrary KML string that is not hosted publicly.

Here is an example of using the *KmlNetworkLink* approach to load a KML/KMZ file:

```
var link=ge.createLink('');
var kmlUrl ='http://www.sirrs.org/movekml/testpoint.kml';
//A KML file that is hosted at some public URL
link.setHref(kmlUrl);
var networkLink=ge.createNetworkLink('');
networkLink.set(link, true, true);
ge.getFeatures().appendChild(networkLink);
```

Here is the JavaScript code of using the *fetchKml* technique to load a KML/KMZ file:

```
var kmlUrl='http://www.sirrs.org/movekml/testpoint.kml';
//The hosted KML file
google.earth.fetchKml(ge, kmlUrl, function(kmlObject) {
if (kmlObject)
ge.getFeatures().appendChild(kmlObject);
});
```

The JavaScript code for loading and displaying a KML string with the *parseKml* method looks like this:

```
var kmlString="
+'<?xml version="1.0" encoding="UTF-8"?>'
+'<kml xmlns="http://www.opengis.net/kml/2.2">'
+'<Placemark>'
+'<name>Placemark from KML string</name>'
+'<Point>'
+'<coordinates>-122.1, 37.9, 0</coordinates>'
+'</Point>'
+'</Placemark>'
+'</kml>';
var kmlObject=ge.parseKml(kmlString);
ge.getFeatures().appendChild(kmlObject);
```

3.2. Extract KML geometry elements

The second step is to extract KML geometry elements from the imported KML objects in the Google Earth plug-in, used for the subsequent moving transformation. We can use the *getElementById* or *getElementByUrl* functions in the Google Earth API to get a specified KML geometry element by its ID or URL (Google, 2014):

```
var kmlElements=ge.getElementById(string ID); //Get a KML geometry element by its ID
var kmlElements=ge.getElementByUrl (string url); //Get a KML geometry element by its URL
```

In order to get a list of elements by a specified geometry type, we employ the *getElementsByType* function as follows (Google, 2014):

```
var points=ge.getElementsByType('KmlPoint');//Get all points
var linestrings=ge.getElementsByType('KmlLineString');//Get all line strings
var linearrings=ge.getElementsByType('KmlLinearRing');//Get all linear rings
var polygons=ge.getElementsByType('KmlPolygon');//Get all polygons
var models=ge.getElementsByType('KmlModel');//Get all models
var MultiGeometries=ge.getElementsByType('KmlMultiGeometry'); //Get all multi-geometries
```

3.3. Set movement distances

In the third step, we manually input the displacement along a specified orientation to specify how a transformation is performed on the selected objects. Within Google Earth, there are three cases of moving transformation to be taken into account: moving along the vertical direction, moving along the latitudinal (east–west) direction, and moving along the longitudinal (north–south) direction. Correspondingly, we have an obligation to set the movement distances respectively along each direction.

When setting movement distances within the Google Earth plug-in, there are two practical graphical user interfaces to control the displacement interactively:

The simplest approach is to incorporate existing general-purpose slider controls, such as the Tigra slider control (SoftComplex Inc., 2010) and the HTML-native slider, to control the movement distances interactively. The general-purpose slider control, which could be located on the webpage outside the Google Earth container (De Paor et al., 2012), usually consists of two elements: the draggable piece (thumb), and the track along which the thumb can travel freely. Once the thumb is dragged to a position on the track, the position of the thumb is translated into the value of the movement distance to change the displacement of the KML geometry elements.

A more compelling approach for controlling the displacement is to create custom slider controls within the Google Earth container by employing interactive screen overlays (Dordevic, 2012), and to drag those custom sliders in order to set movement distances. A screen overlay is an image that is fixed to a specified location on the screen (Wernecke, 2009). We can generate a custom slider through arranging two screen overlays with appropriate sizes into corresponding positions on the window of the Google Earth plug-in. One screen overlay is used as the stationary track, and the other screen overlay is used as the draggable thumb (Dordevic, 2012). Similar to the built-in Google Earth navigation controls, this custom slider also has the virtue of being semitransparent. That is, only when the screen defined by the slider gains focus can the slider becomes interactive and draggable. By using this screen-overlay-based slider, we can maintain application responsiveness and enhance interactivity when setting movement distances for KML geometry elements.

3.4. Adjust geographic coordinates of KML geometry elements

The main work of the fourth step contains calculating the transformed geographic location for each KML geometry element according to the movement distance, and adjusting geographic coordinates of the points in each KML geometry element. In this step, two essential problems need to be solved: one is how to calculate the transformed location for a KML geometry element, and the other is how to adjust geographic coordinates of multiple KML geometry elements in a fast and batched way. The following subsections are explanations of these problems.

3.4.1. Calculate geographic locations for moving transformation

Any moving transformation to a specified KML geometry element can be ultimately attributed to the coordinate transformation of the points composed that KML geometry element. In a relatively small range of the Earth's surface and near-surface, after moving transformation the geographic coordinate (longitude, latitude and altitude) of a specified point can be calculated as follows (Whitmeyer, 2012):

$$\lambda_1 = \lambda_0 + \frac{\Delta\lambda \times 180}{\pi \times (r+h_0) \times \cos\varphi_0}$$

$$\varphi_1 = \varphi_0 + \frac{\Delta\varphi \times 180}{2c + \pi h_0}$$

$$h_1 = h_0 + \Delta h$$

where λ_0, φ_0,and h_0 are initial 3D coordinates (longitude, latitude and altitude respectively) of the point (the longitude and latitude are defined in decimal degrees, and the altitude is defined in meters); λ_1, φ_1, and h_1 are 3D coordinates of the transformed point; $\Delta\lambda$ is the movement distance (in meters) along the longitudinal (north–south) direction; $\Delta\varphi$ is the movement distance (in meters) along the latitudinal (east–west) direction; Δh is the movement distance (in meters) along the vertical direction; r is the equatorial radius, r=6,378,137 m; c is the distance from the equator to a pole, c=10001965.729 m; and π is the circumference ratio (pi).

It should be pointed out that the above equations relate only to small scale objects and motions in a relatively small range of the Earth's surface and near-surface (from meters to tens of kilometers). In this situation, the curvature of the Earth is not an issue and should not be considered when moving transformation. For large objects such as continents and large movements over thousands of kilometers, a different approach using Euler poles

and quaternion interpolation (De Paor, 1996; Dordevic and Whitmeyer, 2014) is required.

3.4.2. Adjust geographic coordinates in a fast and batched way

On the Google Earth virtual globe, usually there are plenty of KML geometry elements that need to be moved almost simultaneously. Due to the vast amount and complicated structure of KML geometry elements, challenges arise when adjusting geographic coordinates of massive KML geometry elements at one time. In order to enhance the efficiency of moving a large volume of KML geometry elements within Google Earth, an automatic, fast and batched method to adjust geographic coordinates of KML geometry elements needs to be designed and implemented. In the fourth step, we first use Google Earth API to extract the initial 3D coordinates of each point in each KML geometry element, and then calculate the offsets of the 3D coordinates according to the movement distances, and finally adjust the location of the point in terms of the offsets.

The JavaScript code of moving multiple KML geometry elements almost at the same time works as follows:

```
function moveKmlGeometryElements(deltaLat, deltaLng, deltaAlt) {
// deltaLat – The movement distance (in meters) along the latitudinal (east–west) direction
// deltaLng – The movement distance (in meters) along the longitudinal (north–south) direction
// deltaAlt – The movement distance (in meters) along the vertical direction
var i=null;
var pi_value=Math.PI; //Pi
var radius_earth=6378137; //The equatorial radius
var c_distance=10001965.729; //The distance from the equator to a pole
var deltaLat1=0;
var deltaLng1=0;
var deltaAlt1=0;
//Move points
var points=ge.getElementsByType('KmlPoint');//Get all points
for ( i=0; i<points.getLength(); i++) {
var pointi=points.item(i);
//The initial 3D coordinates of a point
var lati=pointi.getLatitude();
var lngi=pointi.getLongitude();
var alti=pointi.getAltitude();
//Calculate the offsets of 3D coordinates according to the movement distances
deltaLat1=(deltaLat*180)/(2*c_distance+pi_value*alti);
deltaLng1=(deltaLng*180)/(pi_value*(radius_earth+alti)*
Math.cos(lati * pi_value/180.0 ));
deltaAlt1=deltaAlt;
var newlati=lati+deltaLat1;
var newlngi=lngi+deltaLng1;
if (newlati>90) { newlati=90; }
if (newlati<−90) { newlati=-90;}
if (newlngi>180) { newlngi=newlngi - 360; }
if (newlngi<−180) { newlngi=newlati+360; }
//Adjust the location of the point using the offsets
pointi.setLatLngAlt(newlati, newlngi, alti+deltaAlt1);
}
//Handle line strings, linear rings and polygons
var linestrings=ge.getElementsByType('KmlLineString');//Get all line strings, also including linear rings and polygons
for ( i=0; i<linestrings.getLength(); i++) {
var pointi=null;
var pointCoordArray=null;
pointi=linestrings.item(i);
pointCoordArray=pointi.getCoordinates ();
for (var j=0;j<pointCoordArray.getLength(); j++){
var pointCoord=pointCoordArray.get(j );
//The initial 3D coordinates of a point
var lati=pointCoord.getLatitude();
var lngi=pointCoord.getLongitude();
var alti=pointCoord.getAltitude();
//Calculate the offsets of 3D coordinates
deltaLat1=(deltaLat*180)/(2*c_distance+pi_value*alti);
deltaLng1=(deltaLng*180)/(pi_value*(radius_earth+alti)*
Math.cos(lati * pi_value/180.0));
deltaAlt1=deltaAlt;
var newlati=lati+deltaLat1;
var newlngi=lngi+deltaLng1;
if (newlati>90) {newlati=90;}
if (newlati<−90) { newlati=−90; }
if (newlngi>180) { newlngi=newlngi−360;}
if (newlngi<−180) {newlngi=newlati+360;}
//Adjust the location of the point using the offsets
pointCoordArray.setLatLngAlt(j, newlati, newlngi, alti+deltaAlt1);
}
}
//Move 3D models
var models=ge.getElementsByType('KmlModel'); //Get all 3D models
for (i=0; i<models.getLength(); i++) {
var modeli=models.item(i);
var modelLoc=null;
modelLoc=modeli.getLocation();
//The initial placement of a model
var lati=modelLoc.getLatitude();
var lngi=modelLoc.getLongitude();
var alti=modelLoc.getAltitude();
//Calculate the offsets of 3D coordinates
deltaLat1=(deltaLat*180)/(2*c_distance+pi_value*alti);
deltaLng1=(deltaLng*180)/(pi_value*(radius_earth+alti)*
Math.cos(lati * pi_value/180.0 ));
deltaAlt1=deltaAlt;
var newlati=lati+deltaLat1;
var newlngi=lngi+deltaLng1;
if (newlati>90) {newlati=90;}
if (newlati<−90) {newlati=-90; }
if (newlngi>180) { newlngi=newlngi – 360;}
if (newlngi<−180) {newlngi=newlati+360;}
//Adjust the location of the model using the offsets
modelLoc.setLatLngAlt(newlati, newlngi,alti+deltaAlt1);
}
}
```

In this example, we use a code fragment to handle line strings, linear rings and polygons simultaneously. As presented in Table 1, geospatial positions of the outer/inner boundaries in polygons are defined by the KML ⟨LinearRing⟩ elements. While in Google Earth API, the *KmlLinearRing* class is derived from the *KmlLineString* class. Therefore, any transformation to a polygon or linear ring can be traced back to the KML ⟨LineString⟩ element. Accordingly, the moving transformation of polygons and linear rings can be handled along with line strings.

It should be pointed out that we do not need any special code to deal with multi-geometries. This is because the ⟨MultiGeometry⟩ element is only a collection of multiple geometry elements associated with the same KML feature, and the transformation of any

multi-geometry can be traced back to corresponding geometric primitives like points, line strings, linear rings, polygons or models.

3.5. Display transformed KML geometry elements on Google Earth

Finally, transformed KML geometry elements are displayed in the Google Earth plug-in for 3D visualization and spatial analysis. Using graphical interactive devices (such as the mouse and the keyboard) in the 3D-rendering environment provided by Google Earth, we can freely observe the locations of the transformed KML geometry elements, and query the property information associated with those elements.

4. Implementation and application

To illustrate the effectiveness of the proposed framework for the moving transformation of KML geometry elements, a webpage (http://www.sirrs.org/MoveKml/en/MoveKml.html), termed Moving KML, is designed and developed using the Google Earth web browser plug-in and its JavaScript API. As shown in Fig. 3, the user interface of MovingKML is composed of three parts: (1) the input area, which is designed for importing the URL of a hosted KML/KMZ file, is located in the top of the webpage; (2) the Google Earth container, also incorporating three screen-overlay-based custom sliders, is located in the left side of the screen; and (3) the slider control area, containing three general-purpose slider controls, is located in the right side of the screen. In the slider control area, the Tigra slider control (SoftComplex Inc., 2010), a frequently-used DHTML component for adding vertical/horizontal sliders to HTML webpages, is instantiated as the interactive tool to control the movement distance. Any computer that already has the Google Earth web plug-in installed can freely visit this webpage on a decent broadband connection. After loading a hosted KML/KMZ file from a specified URL, users of this webpage can move multiple KML geometry elements arbitrarily.

When using MovingKML, we first input the URL of any hosted KML/KMZ file and click the “Load KML/KMZ” button, and then the KML geometry elements stored in the KML/KMZ file immediately appear in the Google Earth container. By employing the general-purpose slider controls, which are located on the webpage beside the Google Earth container, we can manually set the movement distance along a specified direction. Alternatively, the screen-overlay-based custom sliders, which are embedded in the Google Earth container, also can be used to control the movement distances. In the light of the predefined movement distance, MovingKML automatically adjusts the geographic coordinates of the KML geometry elements, and displays the transformed KML geometry elements on the Google Earth container quasi-instantly. By performing the above-mentioned operations with a flexible and intuitive graphical user interface (GUI), we are able to move a large volume of KML geometry elements in a fast and batched way.

An example of moving KML geometry elements is illustrated in Fig. 4. In this example, a KML file (http://www.sirrs.org/MoveKml/testkml/TestKmlFile.kml) is loaded into the Google Earth container. This file contains several subsurface objects, including a borehole model that incorporates a drilling location, several strata and corresponding contacts, and a COLLADA model that represents the subsurface geological structure. As shown in Fig. 4A, the subsurface objects originally hide behind the Earth's surface. Therefore, they couldn’t be directly displayed on the Google Earth

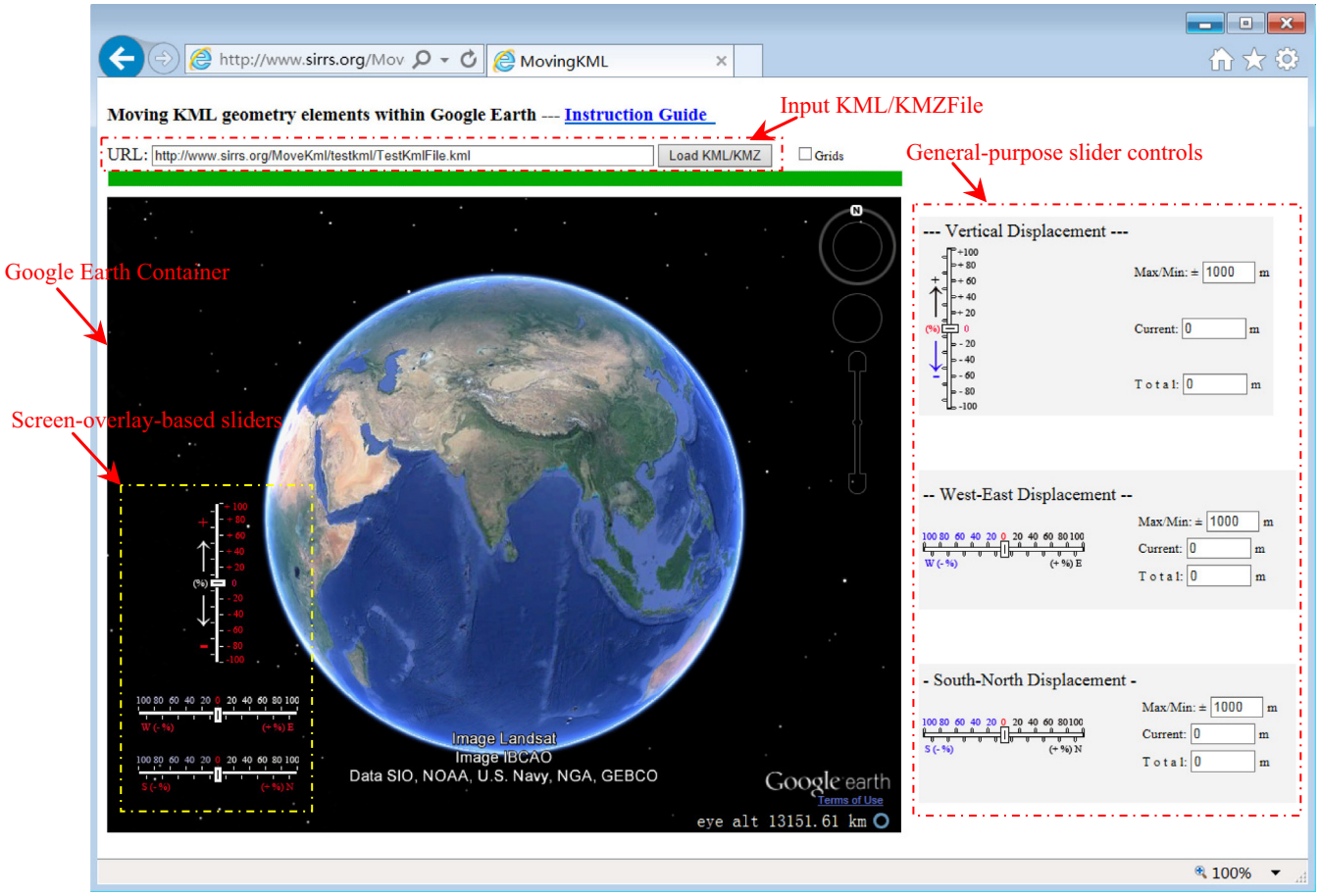

Fig. 3. User interface of MovingKML.

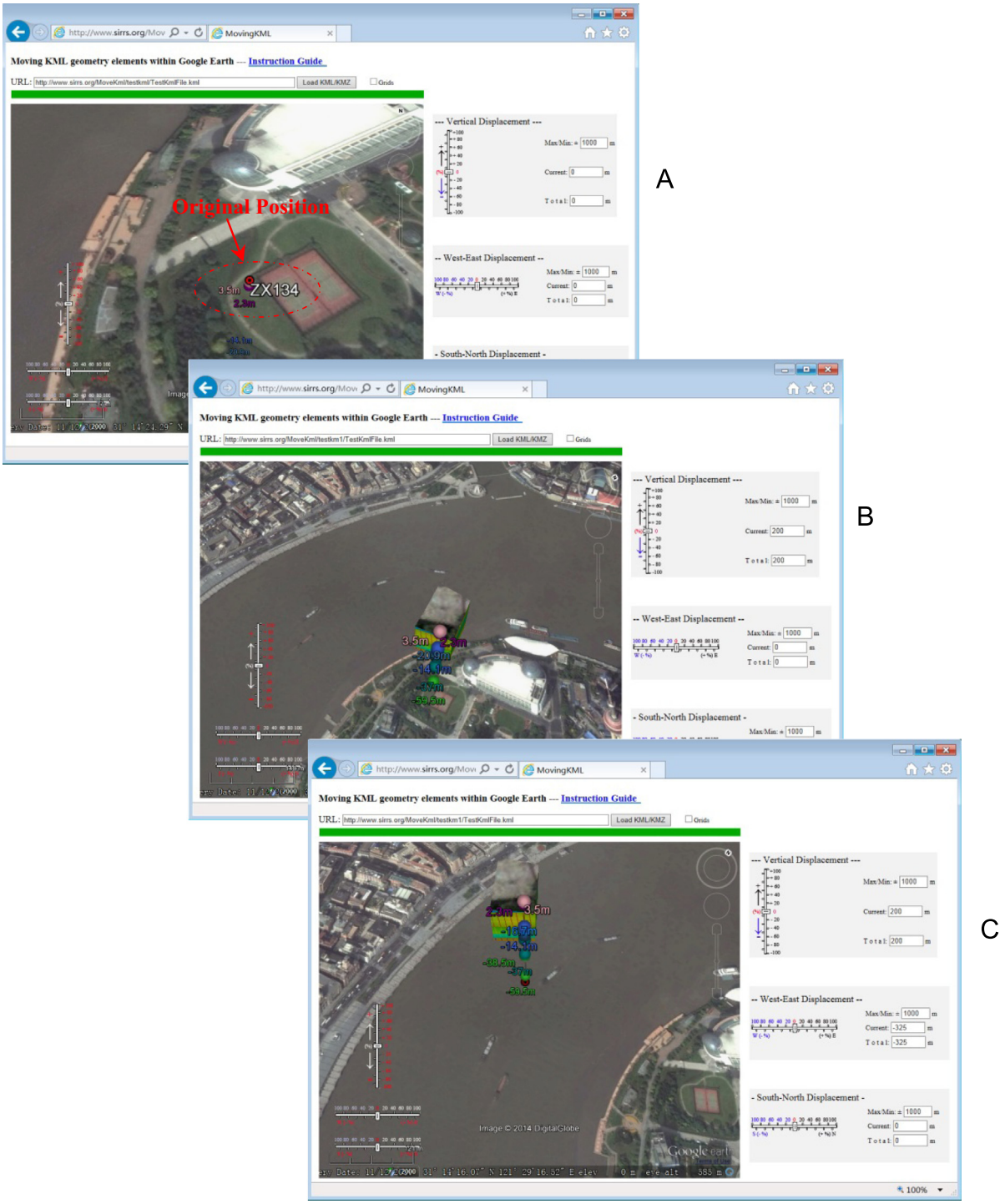

Fig. 4. Example of moving KML geometry elements using MovingKML: (A) The subsurface objects are originally located below the Earth's surface, and cannot be directly displayed on Google Earth. (B) The vertical positions of the subsurface objects are manually elevated by 200 m in order to make them visible above the Earth's terrain surface. (C) The horizontal positions of the subsurface objects are shifted by 325 m westwards from the previous positions.

virtual globe. In order to make them visible above the Earth's terrain surface, we employ vertical sliders to manually uplift the altitudes of the subsurface KML objects, and the subsurface objects immediately emerge from the subsurface (Fig. 4B). Similarly, using horizontal sliders provided by MovingKML, we also can horizontally move the KML geometry elements through setting the movement distance along the latitudinal (east–west) or longitudinal (north–south) direction (Fig.4C).

5. Conclusion

With the increasing need for geospatial information and the widespread use of the Google Earth virtual globe, KML is becoming an international standard that has been widely embraced by geoscientists as a means to represent, publish and exchange geospatial objects, and there are increasing amounts of KML-based information resources available that cater to different

disciplines and audiences. In the field of Digital Earth science and technology, it has become a topical research area to focus on interactively visualizing and analyzing geospatial information within virtual globe applications. Due to the current limitations of Google Earth, it is either hard or impossible to interactively move KML objects only using the existing Google Earth desktop application, especially when the data sets are in large volume. In this paper, we have designed and illustrated the general framework and associated implementation methods for moving multiple KML geometry elements within Google Earth. A key advantage of the proposed framework is that it provides a simple, uniform and efficient user interface for moving multiple KML geometry elements within the Google Earth virtual globe. More importantly, the proposed framework and associated implementations can be conveniently integrated into other customizable Google Earth applications to support interactively visualizing and analyzing geospatial objects defined by KML geometry elements, especially subsurface features.

Acknowledgments

This research was supported by the Social Science Foundation of Shanghai (Grant no. 2014BCK002), the National Natural Science Foundation of China (Grant no. 40902093), the National Science and Technology Program of China (Grant no. SinoProbe-08), the Development Foundation of Experimental Teaching Equipment in East China Normal University (Grant no. 64100010), and the Open Foundation of Key Laboratory for GIS (Grant no.KLGIS2014C02). We would like to thank Declan G. De Paor, Tom G. Blenkinsop and the Editor for their helpful and constructive suggestions for improving the paper.

Appendix A. Supporting information

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.cageo.2014.07.016.

References

Bailey, J.E., Chen, A., 2011. The role of virtual globes in geoscience. Comput. Geosci. 37 (1), 1–2.
Bailey, J.E., Whitmeyer, S.J., De Paor, D.G., 2012. Introduction: the application of Google Geo Tools to geoscience education and research. Geological Society of America Special Papers 492, pp. vii–xix.
Ballagh, L.M., Raup, B.H., Duerr, R.E., Khalsa, S.J.S., Helm, C., Fowler, D., Gupte, A., 2011. Representing scientific data sets in KML: methods and challenges. Comput. Geosci. 37 (1), 57–64.
Blenkinsop, T.G., 2012. Visualizing structural geology: from Excel to Google Earth. Comput. Geosci. 45 (1), 52–56.
Butler, D., 2006. Virtual globes: the web-wide world. Nature 439 (7078), 776–778.
De Paor, D.G., 1996. Computation of orientations for GIS – the 'Roll' of quaternions. Comput. Methods Geosci. 15, 447–456.
De Paor, D.G., Pinan-Llamas, A., 2006. Application of novel presentation techniques to a structural and metamorphic map of the Pampean Orogenic Belt, NW Argentina. Geol. Soc. Am. Abstr. Programs 38 (7), 326.
De Paor, D.G., Whitmeyer, S.J., 2011. Geological and geophysical modeling on virtual globes using KML, COLLADA, and Javascript. Comput. Geosci. 37 (1), 100–110.
De Paor, D.G., Whitmeyer, S.J., Gobert, J., 2008. Emergent models for teaching geology and geophysics using Google Earth. Eos Trans. Am. Geophys. Union 89 (53) (abstract ED31A-0599).
De Paor, D.G., Whitmeyer, S.J., Marks, M., Bailey, J.E., 2012. Geoscience applications of client/server scripts, google fusion tables, and dynamic KML. Geological Society of America Special Papers 492, pp. 77–104.
De Paor, D.G., Williams, N.R., 2006. Solid modeling of moment tensor solutions and temporal aftershock sequences for the Kiholo Bay earthquake using Google Earth with a surface bump-out. Eos Trans. Am. Geophys. Union, 87; (abstract S53E-05).
Dordevic, M.M., 2012. Designing interactive screen overlays to enhance effectiveness of Google Earth geoscience resources. Geo. Soc. Am. Special Pap. 492, 105–111.
Dordevic, M.M., 2013. Slope Faliure at Kilauea Volcano ⟨http://www.digitalplanet.org/API/SOS/index.html⟩ (accessed 28.03.14.).
Dordevic, M., Whitmeyer, S., 2014. Move and Rotate Google Earth Elements. ⟨http://geode.net/etc/poly⟩ (accessed 04.07.13.).
Goodchild, M.F., Guo, H., Annoni, A., Bian, L., de Bie, K., Campbell, F., Craglia, M., Ehlers, M., van Genderen, J., Jackson, D., Lewis, A.J., Pesaresi, M., Remetey-Fülöpp, G., Simpson, R., Skidmore, A., Wang, C., Woodgate, P., 2012. Next-generation digital Earth. Proc. Natl. Acad. Sci. USA 109 (28), 11088–11094.
Google, 2014. Google Earth API Reference. ⟨https://developers.google.com/earth/documentation/reference⟩ (accessed 20.03.14.).
Lee, T., Guertin, L., 2012. Building an education game with the Google Earth application programming interface to enhance geographic literacy. Geol. Soc. Am. Special Pap. 492, 395–401.
Martínez-Graña, A.M., Goy, J.L., Cimarra, C.A., 2013. A virtual tour of geological heritage: Valourising geodiversity using Google Earth and QR code. Comput. Geosci. 61 (1), 83–93.
Navin, J., de Hoog, M., 2011. Presenting geoscience using virtual globes. AusGeo News 104, 15–19.
Nurik, R., 2009. An overview of using KML in the Earth API. ⟨https://developers.google.com/earth/articles/earthapikml⟩ (accessed 20.03.14).
Postpischl, L., Danecek, P., Morelli, A., Pondrelli, S., 2011. Standardization of seismic tomographic models and earthquake focal mechanisms data sets based on web technologies, visualization with keyhole markup language. Comput. Geosci. 37 (1), 47–56.
SoftComplex Inc., 2010. Tigra Slider Control. ⟨http://www.softcomplex.com/products/tigra_slider_control⟩ (accessed 07.07.13.).
Stewart, M.E., Baldwin, K., 2012. Workshops, community outreach, and KML for visualization of marine resources in the Grenadine Islands. Geol. Soc. Am. Special Pap. 492, 63–76.
Wang, Y., Huynh, G., Williamson, C., 2013. Integration of Google Maps/Earth with microscale meteorology models and data visualization. Comput. Geosci. 61 (1), 23–31.
Wernecke, J., 2009. The KML Handbook: Geographic Visualization for the Web. Addison-Wesley, Upper Saddle River, USA p. 339.
Whitmeyer, S., 2012. Moving Polygons in Google Earth. ⟨http://www.digitalplanet.org/site/NewOct12.html⟩ (accessed 04.09.13.).
Whitmeyer, S.J., Nicoletti, J., De Paor, D.G., 2010. The digital revolution in geologic mapping. GSA Today 20 (4/5), 4–10.
Wilson, T. (Ed.), 2008. OGC KML. OGC07-147r2. Open Geospatial Consortium, Inc. ⟨http://portal.opengeospatial.org/files/?artifact_id=27810⟩ (251 pp. accessed 28.03.14).
Yu, L., Gong, P., 2012. Google Earth as a virtual globe tool for Earth science applications at the global scale: progress and perspectives. Int. J. Remote Sens. 33 (12), 3966–3986.
Zhu, L., Wang, X., Zhang, B., 2014. Modeling and visualizing borehole information on virtual globes using KML. Comput. Geosci. 62 (1), 62–70.