Pervasive Touristic Location Based Service Mobile App with a Social Perspective

Umberto Di Staso
Fondazione Graphitech
Trento, Italy
umberto.distaso@graphitech.it

Raffaele De Amicis
Fondazione Graphitech
Trento, Italy
raffaele.de.amicis@graphitech.it

Federico Devigili
Fondazione Graphitech
Trento, Italy
federico.devigili@graphitech.it

Gabrio Girardi
Fondazione Graphitech
Trento, Italy
gabrio.girardi@graphitech.it

System Architecture

One of the most important challenges to face has been considered the definition of a sound system architecture, robust and easily adaptable for future developments.

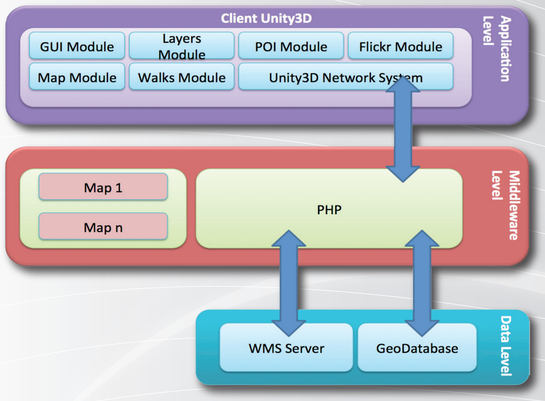

Figure 1. System Architecture

More in details, as mentioned in the abstract of this paper, the main focus has been on the design of a system architecture capable to efficiently deliver geo-information to a Game Engine in order to exploit its capabilities.

At the lowest level, the framework will feature a data level containing a number of different repositories.
The goal of this level is to provide harmonised access to data repositories through the use of two main databases.
The data level is made of two main components:

1) a GeoDatabase (PostGIS);
2) a WMS Server (NASA World Wind Server).

A GeoDatabase is an optimized database to store and query data related to objects in space, including points, lines and polygons. While typical databases can handle various numeric and character data types, additional functionality needs to be provided for databases to process spatial data types. These are typically called geometries.

Database systems use indexes to quickly look up values. The way most databases index data is not optimised for spatial queries. Instead, spatial databases use a spatial indexes to speed up their operations. In addition to typical SQL queries, such as SELECT statements, spatial databases can perform a variety of spatial operations. The following query types and many more are supported by the Open Geospatial Consortium:

- Spatial Measurements find the distance between points, polygon areas, etc.;
- Spatial Functions modify existing features to create new ones, for example by providing a buffer around them, intersecting features, etc.;
- Constructor Functions create new features with a SQL query specifying the vertices (points of nodes) which can make up lines. If the first and last vertexes of a line are identical the features can also be of the same type polygon;
- Observer Functions are queries that return specific information about a feature such as the location of the circle centre.

PostGIS is an open source software program that adds support for geographic objects to the PostgreSQL object-relational database. PostGIS follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC).

NASA World Wind Server is the server version of the famous NASA World Wind, the open source virtual globe developed by NASA and the open source community. WorldWind can receive geographical information from a server via the WMS - Web Map Service [8] protocol.
The Web Map Service is a standard protocol for serving geo-referenced map images over the Internet that are generated by a map server using data from a GIS database. WMS specifies a number of different request types, two of which are supported by any WMS server:

- GetCapabilities, wich returns parameters about the WMS server and the available layers;
- GetMap, which returns a map image or an elevation file accordingly with the provided parameters.

From a technological perspective, this level will rely on a number of software interfaces that essentially ensure access to the various datasets available. The set of software interfaces will be extremely wide in scope and will have to deal with scenarios ranging from:
- Access to interoperable datasets through web services or other web-based requests (e.g. through HTTP get). This requires the development of a software component capable to invoke services in the appropriate manner that would be transparent to the user.
- Access to interoperable geographical datasets.

The middleware level, has been engineered as an enterprise level infrastructure designed to provide all the functionalities required for interacting with a large set of geographical data.
From a technological point of view, the middleware will be based on an Apache web server. This server will act as a repository for all the static upper level components and as a base component for additional business logic software. The stored assets will consist on precompiled static meshes (graphical objects) ready to be used by the game engine for the end-user application and configuration files. The delivery of these objects has been achieved through simple HTTP requests over the Internet as depicted in Figure 1.

The core business logic built on top of the Apache webserver consists in a PHP module used to dynamically generate HTML responses to the application requests, like preference storage, list of available maps and other additional data not strictly related with the datasets.
More specifically , PHP is a general-purpose server-side scripting language originally designed for web development of dynamic Web pages. The language is interpreted by a web server with a PHP processor module and the output is usually a web page source document (HTML). It is one of the first developed server-side scripting languages to be embedded into an HTML source document, rather than calling an external file to process data.

The choice of a PHP-based solution lies in its easy deployment on most web servers and on almost all operating system and platforms, together with the PHP code supporting the integration with many relational database management systems (RDBMS). In the application, the game client, based on Unity3D, through the a utility class available through Unity3D, named WWW, calls a dedicated public PHP page stored on the web server in order to access specific information in the repository. The PHP script executes the request and queries the relevant database. The result of the request will be formatted as an HTML page and sent to the Unity3D WWW class.

Using this kind of strategy, the client application will communicate only with the middleware level, using normal HTTP requests natively supported by Unity3D.

The application level, visible in Figure 2, is made of the mobile client.

This has been developed as a set of modular components, whereby an advanced interface facilitates the interaction with functionalities implemented within the middleware layer. The choice of moving the business intelligence to the middleware layer allows greater flexibility and scalability. In fact, this way it becomes very simple to deploy clients for additional platforms to respond to different hardware and software specifications.

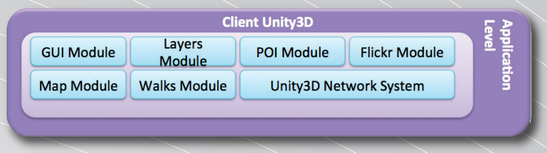

Figure 2. Application Level.

Part of the development at the client side has been focused on the human-computer interaction paradigms to be used on the two different hardware devices, i.e. desktop and tablet.

Furthermore, this architectural choice allows easy expansion of the system with new functionalities that can be developed at the server level in a centralised manner, and then easily made available to all the users.

The mobile client developed in Unity3D [1], the popular game engine, through a user-friendly graphical interface (GUI) that lets users interact with different features and services.
Unity3D is an integrated authoring tool for creating 3D video games or other interactive content such as architectural visualizations or real-time 3D animations. It consists of both an editor for developing/designing content and a game engine for executing the final product.
The development environment of Unity3D runs on Microsoft Windows and Mac OS X, and the client that can be deployed through it can be run on Windows, Mac, Xbox 360, PlayStation 3, Wii, iPad, iPhone, and Android.

Tile Model

The first problem to solve when designing applications that produce real-time three-dimensional terrain is closely related to the dynamic, and efficient generation of the elements forming the virtual world obtained as a final result.
The most efficient model for the generation of geometry-based geographic information is called Tile-model.

Tile-based model is centred on the recursive subdivision of the geometry belonging to a specific part of the area. In this way it is possible to increase or decrease the resolution of the created mesh according to the specific needs of the application.

Figure 3 shows the Tile model structure.

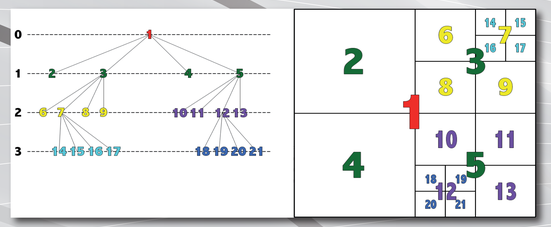

Figure 3. Tile-model structure [2].

Before explaining how the Tile-based algorithm works, it is mandatory that the reader masters some basic concepts:

Zero Level Tile Delta
One of the most important concepts concerning the Tile-based model is the Zero Level Tile Delta or ZLTD. The ZLTD expresses the number of degrees of the earth's surface covered by the level zero Orthophoto. The ZLTD is a fundamental parameter for the conversion of coordinates expressed in latitude and longitude to the virtual world coordinates.

The main conversions functions between geographical and world coordinates are the following:
- The formula responsible for Conversion conversion from geographic coordinates to Tile code.

$$\text{Tile (x,y): } (int)\left(\frac{longitude+180}{\frac{ZLTD}{2^{level}}}, \frac{latitude+90}{\frac{ZLTD}{2^{level}}}\right)$$

For example example when using a ZLTD to 1 degree, latitude and longitude equals to 11.11 and 46.00 respectively. and, finally, tAt the second level of detail, the final Tile code will be (764, 544). Assuming that the available Orthophoto is a pre-Tiled dataset in jpg format with the same ZLTD, the texture identifying the Tile will be identified by name 544_764.jpg [12].

- The function to determinine the position of a child Tile with respect to its parent Tile:

$$\text{Position (x, y): } \left(\frac{pTileX}{2^{level}} * ZLTD, \frac{pTileY}{2^{level}} * ZLTD\right)$$

- While to calculate the size of each Tile the following operator applies:

$$\text{Size: } \frac{ZLTD}{2^{level}}$$

Visibility
The visibility factor expresses whether a Tile is visible or not, depending on the user's point of view at a given instant moment in time.
For performance optimization, lit would be unacceptable to create three-dimensional models if they are were not visible.

The visibility factor is based on a very simple and mathematical calculus: knowing the size of each Tile and the six planes identifying the frustum of the camera in use, it is possible to estimate the distance between the bounding sphere of each geometry and each one of the planes that make the frustum, by establishing the visibility of the Tile on which the calculation was made [6].

Divisibility
The concept of divisibility is closely related to the distance between the centre of each Tile and the current point of view. Knowing the distance whichdistance that separates these two elements, it is crucial to decide when to subdivide each element in order to increase the resolution of an area or when to merge them if we the user are movinges away from our the reference object.

The Tile-Based Level Of Detail Algorithm
In this section we will explain how the algorithm to manage the level of detail works. The algorithm in question is recursive and the whole procedure is carried out entirely at each frame.

The main concept of this algorithm is the following: for each Zero level Tile (the number of these depends by the dataset in use), the first operation that is performed is a test of visibility, in this way is possible to isolate the non-visible Zero elements. After the first check, each remaining Tile is controlled if to assess if the distance from the point of view requires the an increase of the current resolution [3].

If the previous check returns true, each Zero level Tile is divided into four children, and for each of them the algorithm is executed in the same way as for the Zero level Tiles.

Figure 4 shows the general procedure.

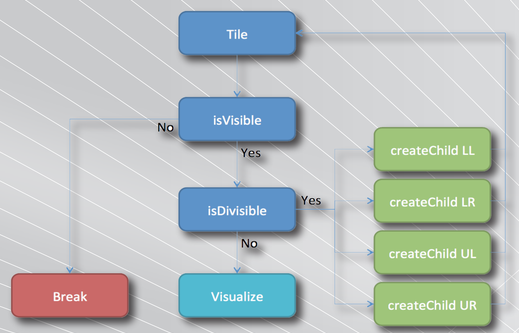

Figure 4. LOD alghorithm.

Applications

This section will provide some screenshots of the running software, showing the possible applications of the implemented geo-visualization system.

The architecture designed, described in section 3, is able to provide proper support to the required functionality. In this way, Unity3D can be used to display geographical information passing through an intermediate level, consisting of various PHP scripts, which allow the connection with the lower geographical data level.

With regard to figure 2, we will analyse the main modules developed using the Unity3D game engine.

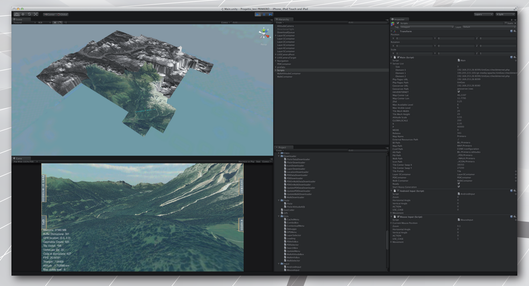

Figure 5. Unity 3D Game Engine

A. Maps Renderer
This is the basic functionality developed, which allows the display of any terrain in three dimensions, starting from Orthophoto and DEM data.

Figure 6. Maps Renderer Module running example.

B. Layers Module
A thematic layer is applied on a three-dimensional generated mesh; its purpose is to offer more information on the area generated through the system, so to apply multiple themes and manage its transparency. Each layer is provided using the WMS server.

Figure 7. Layers Module running example.

C. Point Of Interest
A point of interest is a three-dimensional marker located on the territory, which identifies an object belonging to a category placed exactly in those coordinates in the real world. Each POI has a range of information such as name, description, and photos, viewable via a popup activated by touching each of them. Each POI is streamed from a GeoDatabase.

Figure 8. POIs Module running example.

D. Paths
A path is constituted by a series of points connected one to another that can be used to represent a road, a path, a cycling track and so on. As happens for the POIs, several interaction options are available for each route through a contextual menu: information page with photos, description and altitude profile, three-dimensional reconstruction route, points of interest encountered crossing the track and, finally, an animation allowing a bird's eye view of the path in first person mode controller. The sequence of points which composes each path is provided by the GeoDatabase and simplified using the Douglas-Peucker algorithm [5] for points reduction.

Figure 9. Paths Module running example.

OPEN ISSUES
1) Float Precision
A problem that has been highlighted in the testing phase concerns the precision of the 32-bit float numbers that are used by Unity 3D to place objects in World Space. According to the model based on Tile, introduced in the previous sections, it is readily understood how to decrease the size of each created geometry (i.e. the Tile), when an increasing details is required. This has led to problems of numerical stability, due to round automatically carried out when the 23 bits of the mantissa of the float numbers 32 is no longer sufficient.

A partial solution to this problem, based on Thorne's papers [8], is to place the loaded map in the origin [7] of the world, where a one to one projection has been made between real coordinates and virtual world coordinate units (for example latitude 11.00 and 46.00 longitude coordinates are placed in unity (46,0,11)).

2) Collision System
Another open issue regards the collision detection system: although the application developed provides support in the majority of use cases required by the navigation system, some sequences of movements can cause the penetration of the camera within the terrain geometry objects.

Acknowledgment

The research leading to these results has received funding from the European Community's Competitiveness and Innovation framework Programme (CIP/2007-2013) under the Grant Agreement n. 270998. This publication reflects the authors view, and the European Commission is not responsible for any use which may be made of the information contained therein.

References

[1] S9. Unity 3D Game Engine, http://unity3d.com
[2] Patrick Cozzi, Kevin Ring, "3D Engine Design for Virtual Globes", CRC Press, pp.370–371
[3] Martin Reddy, Yvan Leclerc, Lee Iverson, and Nat Bletter, "TerraVision II, Visualizing Massive Terrain Databases in VRML", SRI International
[4] Williams, L. Pyramidal Parametrics. Proceedings of SIGGRAPH '83, pp.1-11.
[5] David Douglas, Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer, pp.112–122.
[6] Ulf Assarsson and Tomas Moller, "Optimized View Frustum Culling Algorithms", Chalmers University of Technology, pp.12-13.
[7] Chris Thorne, "Using a Floating Origin to Improve Fidelity and Performance of Large, Distributed Virtual Worlds", University of Western Australia.
[8] Chris Thorne, "Error Minimising Pipeline for Hi-Fidelity, Scalable Geospatial Simulation", University of Western Australia.