04-SA

IEEE Sixth International Symposium on

Multimedia Software Engineering

13-15 December 2004 Miami, Florida, USA

IEEE COMPUTER SOCIETY

IEEE

With support of Florida International University, Florida, USA
IEEE/CS Technical Committee on Computational Intelligence
IEEE/CS Technical Committee on Multimedia Computing
IEEE/CS Technical Committee for Services Computing

Proceedings

IEEE Sixth International Symposium on Multimedia Software Engineering

Miami, Florida

December 13 – 15, 2004

Supported by

IEEE Computer Society Technical Committee on Computational Intelligence
IEEE Computer Society Technical Committee on Multimedia Computing
IEEE Computer Society Technical Community for Services Computing

IEEE COMPUTER SOCIETY
http://computer.org
Los Alamitos, California
Washington • Brussels • Tokyo

System Architecture for 3D TerraFly Online GIS

Naphtali Rishe, Yanli Sun, Maxim Chekmasov, Andriy Selivonenko, Scott Graham
Florida International University
{rishen, maximc, suny, selivona, grahams}@cs.fiu.edu

Abstract

We present the architecture of 3D TerraFly, a three-dimensional extension of TerraFly, which is a Web-enabled geographic information system designed for visualization, analysis and composition of remotely-sensed imagery. 3D TerraFly has a client/server architecture. The client-side application is a web-embedded component called the 3D TerraFly Viewer that performs 3D terrain rendering and interactive navigation. The server side applications include Texture Imagery Servers and Mesh Servers that, respectively, handle texture data requests and mesh data requests.

1. Introduction

Increasing demands for 3D GIS (three dimensional geographic information systems) have emerged in many areas, such as urban planning, environmental monitoring, telecommunication, real-estate marketing, weather simulation and military training. Intensive research of 3D GIS has covered various aspects for representing and analyzing real world phenomena. Many 3D GIS applications have been developed and some commercial 3D GIS systems are on the market. Most of them are capable of integrating complex semantic and geometric GIS data into 3D visualization, and of providing comprehensive exploration and analysis functionalities. We refer to [1] for a detailed review of the history, current status, and future trends of 3D GIS.

When designing a 3D GIS that is expected to serve users over the Internet (online), researchers meet certain challenges. First, a bottleneck is the limited computation power that client users can access. Internet users are usually equipped with low-end PCs which are not capable of performing real-time computation-intensive tasks. Unlike professional graphics workstations, their processors and memory capacities are not suitable for real-time handling of data-intensive 3D visualization tasks.

Limited Internet bandwidth is another challenge for online 3D GIS. The bandwidths of the Internet connections vary over a large range, from 56 Kbps dial-up to fast Internet connections of 100 Mbps. In most cases, the users' network connections cannot transfer the volumetric data required to support real-time 3D terrain presentation without compression and other transmission optimization strategies.

In the current paper, we present the architecture of 3D TerraFly, a 3D extension of TerraFly, which is an Internet-enabled GIS designed for 2D remotely-sensed imagery visualization, analysis and composition [2, 3]. The key system requirements for 3D TerraFly, which we have addressed in the present architecture, are: to provide a realistic "bird's-eye view" 3D terrain visualization to reflect real world geographic phenomena, as well as smooth 3D navigation; to support various datasets and to be able to integrate rich semantic geographic information into the presentation; to have high performance and availability; and to be user friendly.

2. Architecture overview

The system is designed as a client/server architecture. The client side application is a web-embedded component called 3D TerraFly Viewer that performs 3D terrain rendering and interactive navigation. The server side applications include Texture Imagery Servers and Mesh Servers that handle texture data requests and mesh data requests correspondingly. Texture and mesh servers are designed as multiple layered distributed systems with a web front end and server clusters that support data replication and dynamic load balancing; see Figure 1.

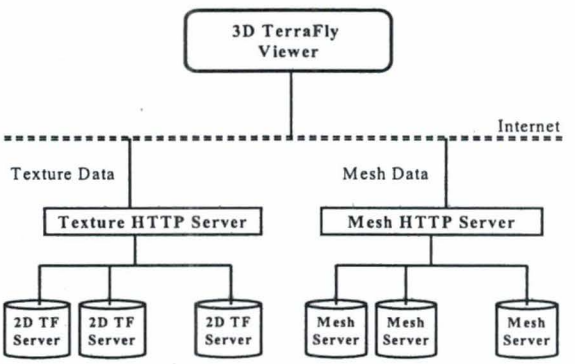

Figure 1. System architecture

0-7695-2217-3/04 $20.00 © 2004 IEEE

3. 3D TerraFly viewer

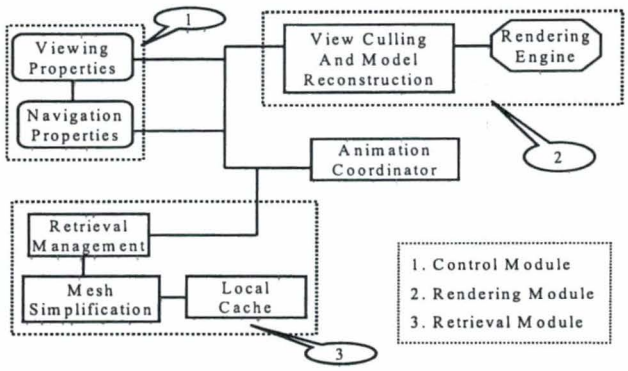

Figure 2. 3D TerraFly viewer architecture

3D TerraFly Viewer is the client side application responsible for retrieving data, modeling, rendering terrain, and controlling animation. Functionally, it can be divided into four modules: the Control Module, the Rendering Module, the Animation Coordinator, and the Data Retrieval Module, as shown in Figure 2.

The Control Module provides the navigation interface to the users and presents related viewing and navigation properties. Viewing properties are used to define static viewing behavior for each frame, including viewpoint coordinate, viewing volume configuration and three viewing transformation angles (Heading, Pitch and Roll). The viewpoint coordinate is the geographic reference point used in terrain navigation. The view frustum's Near side is defined as close to the viewpoint as possible and the Far side is virtually infinite. The actual terrain data covered within the viewing volume is determined by the view culling process. Navigation properties describe animation in terms of flight direction, speed and frame-rate. Navigation allows the viewer to move in three dimensions. The flight direction and speed are decomposed into horizontal and vertical vectors. The horizontal direction vector is represented as an angle in the range 0-360 degrees clockwise. Terrain animation is performed by the viewpoint coordinate update for each frame and re-rendering the scene accordingly.

The Rendering Module is responsible for reconstruction and rendering of the terrain model. Based on dynamic presentation status, the scene for the frame is produced via the following steps: (i) Calculate areas inside the view frustum according to the viewing parameters (viewpoint coordinate, rotation angles and view frustum shape) for the current frame; (ii) Build the region based on multiple resolution models; (iii) Perform region tessellation to obtain the texture and mesh tiles required for each sub-region in the terrain model and contact the Data Retrieval Module to retrieve them if needed; (iv) Bring all rendering primitives to the rendering engine to generate terrain display. The entire processing flow in the Rendering Module is presented in Figure 3. OpenGL [4] is used as the rendering engine in our system. In the rendering process, a double-buffered rendering scheme is used to provide smooth animation. Each frame is first rendered on a background buffer and swapped to the front when rendering is complete. This scheme helps to reduce screen flash.

The Data Retrieval Module manages the Internet data retrieval, local caching and online mesh simplification. Texture and mesh data is retrieved by incremental tile streams. This strategy can significantly reduce the inter-frame data redundancy during terrain animation. Mesh data is stored and retrieved in grid format. Grid meshes may contain redundant geometry vertices. The simplification process is used to eliminate redundant geometry elements and to further smooth small height variations that are not important for rendering quality. To reduce the data access and processing overhead, texture/mesh data and mesh simplification results are kept locally in a LRU (least recently used) cache.

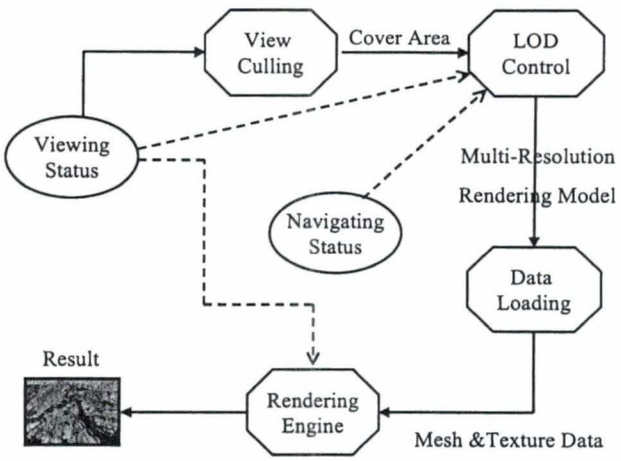

Figure 3. Processing stream in rendering module

Texture and mesh tiles are retrieved asynchronously by multiple threading. In real-time cases, frequent thread allocation and de-allocation can impose a significant overhead on system performance. A thread pool technique is used to tackle this problem. A thread pool contains a certain number of pre-allocated threads and the size of the thread pool can grow on demand. The threads in a thread pool are initially allocated in Sleep (or Suspended) state to avoid unnecessary CPU time consumption. There are two thread-pools in our system: one for texture image retrieving and decoding, and one for mesh tile loading and decimating. In each thread pool, the threads are designated for specific uses. This allows a thread to maintain task related resource allocations permanently to avoid repeated resource allocation/de-allocation overhead.

On the client side, a certain amount of the most recently used texture and mesh data is kept in memory-based caches. The cached data includes currently used and

pre-retrieved data, as well as online simplification results. Texture and mesh data is kept in separate caches.

The Animation coordinator is used to control the terrain animation, predictive data pre-fetching and pre-rendering to best utilize the system's idle time and network bandwidth. If the rendering of the current frame is finished before a given time, the animation coordinator starts preparing the next frame in advance. With the current navigation speed, direction and frame rate, the data model for the next frame is constructed in the background and the needed data is retrieved. When data pre-fetching is finished, the next frame is rendered in the background and is switched to the foreground when the frame is ready to be displayed.

4. Mesh server subsystem

Mesh servers store mesh datasets in grid formats. The stored mesh data include USGS DEM (digital elevation model) meshes and meshes derived from the original DEM meshes. The USGS DEM dataset provides coverage of North America in multiple resolutions (10 meters, 30 meters, 1 arc second, 3 arc seconds and 30 arc seconds). These are used as base mesh data sources in our system. The resolutions of DEM meshes are sparse and are provided in different measurement units. To support multiple resolution rendering and to fill the gaps of resolutions in DEM mesh, mesh data of 8×8, 16×16, 32×32, 64×64, 128×128 and 256×256 meters are derived from DEM meshes and stored on the servers. The derived elevation data is also stored in grid format.

The mesh server subsystem is implemented as a multiple layered distributed system. Functionally it includes three major layers, namely the HTTP Front End Layer, the Mesh Directory Layer and the Mesh Data Server Layer; see Figure 4.

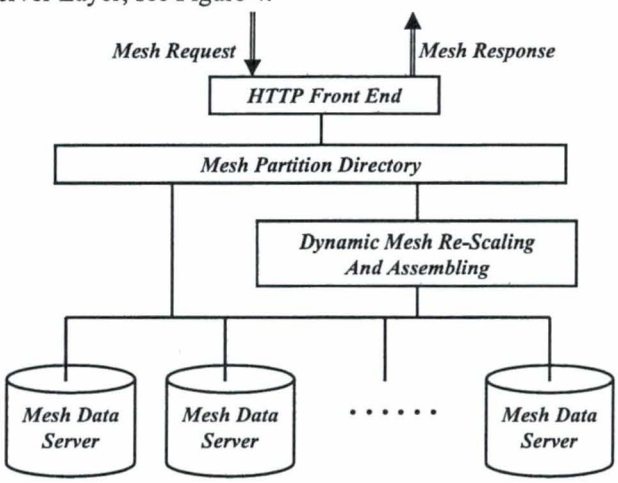

Figure 4. Architecture of mesh server subsystem

The HTTP Front End Layer is an HTTP web server that handles mesh requests. The large volume of mesh data managed on the server side makes centralized storage impractical. To effectively store the volumetric mesh datasets and to efficiently handle mesh requests from multiple users, mesh data is portioned and stored in a distributed environment. Mesh datasets are split along UTM zones. Inside each UTM zone, mesh data may be further split depending on the mesh unit size. A mesh unit is identified by its attributes, namely resolution, UTM zone number and covered geographic area identified by the coordinates of the top-left and bottom-right corners. The Mesh Directory Layer is used to manage distributed mesh data storage. Each mesh unit is represented by one partition entry containing mesh unit attributes (UTM zone number, resolution and covered area) and the address of the mesh file server.

The Mesh Data Server Layer allows data to be retrieved at arbitrary resolution. When a mesh tile request is received, the covered area of the requested tile is obtained from the request parameters with the tile number, resolution, tile size and the UTM zone among them. If the correct resolution of the requested mesh tile can be found in the stored mesh datasets, the mesh directory locates the mesh unit(s) (a mesh tile can either be contained inside one mesh unit or in two or four mesh units) by range query and the mesh tile can be directly obtained from the mesh unit(s) without rescaling. Otherwise, the requested mesh tile is generated from the mesh units with the closest resolution to the requested tile by a mesh scaling process.

The mesh datasets are replicated on multiple file servers and the mesh server subsystem is deployed as a clustered server infrastructure. This scheme helps to improve performance of data access, system availability, and degree of fault tolerance.

5. Texture server subsystem

The texture server subsystem provides terrain imagery for visualization. It is capable of giving unified access to imagery from various GIS datasets, such as satellite images, aerial photography, and topographic maps. Different datasets have different characteristics, so management and access to them may vary in the following aspects: (i) In terms of storage, some datasets are stored locally, while others are provided by third-party sources and should be accessed remotely; (ii) The datasets may have to be accessed via different protocols; (iii) The datasets may support different image resolutions and image granularity (size); (iv) Images may have various formats and have to be handled by different image decoding/encoding algorithms; (v) The datasets may use various geographic coordinate systems. The texture server subsystem is designed as a multi-layered architecture, which hides complex data and processing details and gives a clear abstraction and unified interface to the various datasets. Functionally, there are three layers: the HTTP

Front Layer, the Image Processing Layer, and the Data Source Layer; see Figure 5.

The HTTP Front Layer is the top layer. It serves as the system interface to handle imagery tile requests carried via HTTP protocol. There are two types of requests: metadata query and image data query. Metadata query is used to obtain available resource information such as available data sources and their underlying resolutions. Imagery data can be queried either by individual tiles or by a range query.

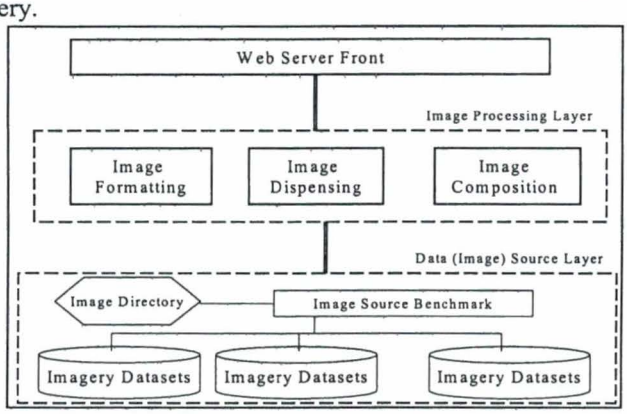

Figure 5. Texture server subsystem

The Image Processing Layer is the middle layer. It is designed to perform image processing tasks. Logically it is comprised of three modules: image formatting, image dispensing and image composition. The image formatting module is used for image encoding/decoding and format conversion according to format specifications contained in image requests. The user specified image formatting information includes encoding format (e.g. JPEG), color channels (black & white or full color) and compression quality or color table size. The image dispensing module is used to handle large scale GIS image downloads. The requested images can cover an area of arbitrary size and are usually generated from multiple tiles. This feature was designed in 2D TerraFly to process images for download and is not used in the 3D system. The image composition model supports online image compositions—mainly image overlaying operations. The composition process visualizes non-imagery GIS information and combines the visualization result with the regular GIS imagery. The image composition is managed under a component-based framework. Under the framework, the information resources and their underlying visualization procedures are managed as self-contained components. Image composition tasks can be of arbitrary complexity and are described by a script language.

The Image Source Layer is the bottom layer. It integrates GIS imagery datasets and provides a unified imagery access interface to the upper layers. A component-based strategy is used in dataset management to flexibly and cost-effectively integrate various data sources. Each dataset is managed as a component defined by the ImageSource data structure, which describes image properties and the access protocols of the underlying datasets. The dataset specific access protocols include dataset address, query methods (HTTP, TCP/IP, file, or database), query parameters and optional authentication information (for remote access only). Image dataset properties include dataset name, underlying imagery format (JPEG, GIF, and PPM), color channels (full-color or black-white), tile size and resolutions (measured in meters/pixel).

6. Conclusion

We have discussed the system architecture of the 3D TerraFly GIS. The architecture has to support efficient management and distributed storage of large volumes of imagery data. The system employs hierarchically clustered servers to support resource replication and parallel processing. Image processing tasks such as image formatting and composition are computationally intensive in nature. Large capacity proxy cache servers have to be attached to the servers that are engaged in time-consuming tasks to cache image composition results and third-party imagery data to improve data processing performance.

7. Acknowledgements

This material is based on work supported by the National Science Foundation under Grants No. HRD-0317692, EIA-0320956, and EIA-0220562

8. References

[1] S. Zlatanova, A.A. Rahman, and M. Pilouk, "3D GIS: current status and perspectives", in *Proceedings. of the Joint Conference on Geo-spatial theory, Processing and Applications*, Ottawa, 8-12 July 2002, 6 p., CDROM.

[2] N. Rishe, S.-C. Chen, N. Prabhakar, M.A. Weiss, W. Sun, A. Selivonenko, and D. Davis-Chu, "Terrafly: A High-Performance Web-Based Digital Library System for Spatial Data Access", *ICDE 2001: International Conference on Data Engineering*, Heidelberg, Germany, April 2-6, 2001, pp. 17-19.

[3] Y. Sun, TerraFly: Internet Geographical Information System Based on Scalable Heterogeneous Framework, *Master Thesis*, School of Computer Science, Florida International University, 2001.

[4] M. Woo, J. Neider, T. Davis, and D. Shreiner, "OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2 (3rd Edition)", Addison-Wesley, 1999.