AN INTEGRATED GLOBAL VISUAL SIMULATION SYSTEM BASED ON MAP DATA

Chen Gang, Wan Gang and Wu Zhiqiang

Department of Cartography, Institute of Surveying and Mapping

66 Longhai Road, Zhengzhou 450052, P.R.China

E-mail:grant@public2.zz.ha.cn

Abstract

This paper reports on significant progress in our efforts to design and construct a real-time 3D terrain environment visual simulation system, named VGTVS (Virtual Global Terrain Visualization System). VGTVS supports the accurate depiction of terrain elevation and imagery, in addition to features such as ground cover and trees, moving vehicles, buildings and other static objects, roads, and atmospheric effects. Most of above models come from multi-scale map data. Thus an entire environment composed of heterogeneous parts must be simulated and integrated at rendering time. The system must be set up to efficiently manage this integration and, ultimately, to manage dynamically the complexity of each part with respect to the others in order to both conform to a strict time budget and to present the most telling details. In this paper we describe a visual simulation system that provides a hierarchical spatial data structure supporting all the parts described above. We discuss in detail our implementations for some of these parts, concentrating especially on global terrain visualization.

1 Introduction

In the past Geographic Information Systems (GIS) were 2D, map-based systems with decidedly non-interactive rates for high resolution display. More recently one might have a 3D GIS but still would type in a coordinate or make a database query and then wait (sometimes several seconds or more) for display of the results. Now with the developing of computer 3D graphics hardware and some efficient algorithms for terrain visualization, we have techniques for organizing geographical information to make integrated GIS visualization systems. We describe such a system here, " Virtual Global Terrain Visualization System (VGTVS)", where we have added some new techniques for fast handling of large amounts of visual detail (most of the scene data come directly from vector map data). In VGTVS, users can move through the scene in real-time by means of a standard input device such as a mouse, and to interact with the GIS. For example, you can build up such a terrain visual simulation — Flying to the Earth. When the view-point, which is in the outer space at beginning, is coming forward to the Earth gradually, the scenes in this simulation is changing as following: from the globe to a continent, then the countries, the states of the country and so on. Finally, there are static houses, running cars and something else in the scene. Of cause, the detail of terrain is increasing accordingly.

A number of software tools is available that support real-time walk-through in three-dimensional scenes. Usually, current visualization software tools for this purpose limit their walk-through capabilities to a scene that is loaded once and stored in main memory. As a whole, the scene can be replaced by another one. While this is good enough for several applications, it is far from satisfactory for geo-exploration, where gigabytes (or even terabytes) of data, for example the data of global terrain, need to be explored That is, while the walk-through progresses, data must be loaded dynamically from disk. To reduce the graphics load, VGTVS displays exact data only where they are useful. In particular, a user should see all the available details of a scene when she can see them in reality, that is, in the close neighborhood of the viewpoint. The farther away we look, the fewer details we can see in reality and need to see in a virtual world. This is in accordance with the regular pattern of vision — “the closer，the clearer”.

2 System architecture

In recent VR applications, it is still an efficiency problem the interaction with large worlds, including their visualization, whenever the memory requirements exceed available main memory. So we combine in VGTVS the two loosely coupled components. One is for visualization and the other is data handling. Figure 1 shows the overall system architecture of VGTVS in which these two components communicate over a (local area or wide area) network. Multiple clients are allowed to connect to a database and explore the same data set. Each client can independently request data corresponding to its needs, especially to adjust the visual part of its local scene to the user’s movements. Different databases could serve queries for different regions, for example one database-server per city.

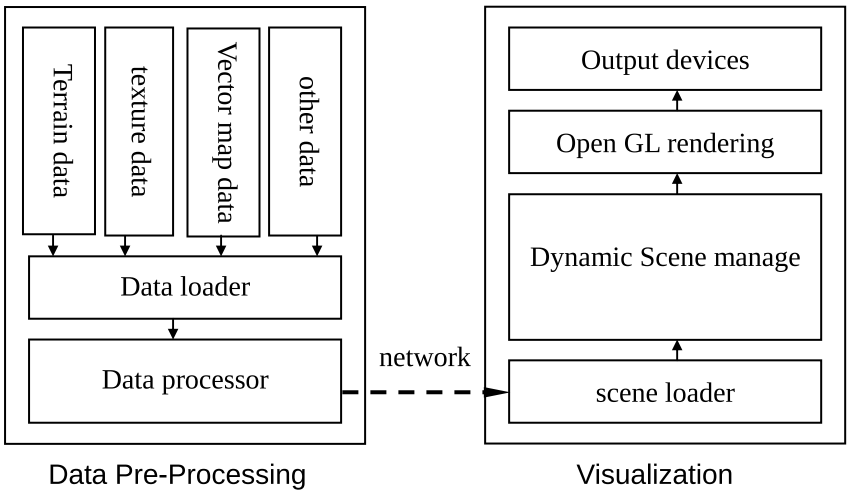

FIGURE 1. System Architecture

The main data types which are incorporated in the data handing component are the grid digital elevation model (DEM) which can be generated directly from contour data of maps, and the texture data which can be satellite images, aerial photographs or topographic raster maps. Other data types are thematic information connected to a point location on the terrain and more types can be added as needed. However, in this version of VGTVS, the main source of a area visual data is the corresponding vector map data, as we can’t get the other types.

In the visualization component, most of the work is performed by the dynamic scene manager, which takes care of updating the actual scene dynamically according to the user’s movements. The actual scene is only the currently visible part, a fragment of a much larger virtual world that is stored in the database. Furthermore, the scene manager also takes care of the correct LOD (level of detail) handling of the rendered scene. An update of the scene can be caused by user movements or changed display parameters from the user-interface. Such an update can involve loading new data from the database, or it can also require re-triangulation of parts of the scene at other LOD. In the following, we will discuss some important processions and new techniques in VGTVS.

3 Data Pre-Processing

There are three forms of data processing in the system of VGTVS: one is off-line pre-processing, which is done once per dataset; another is intermediate on-line processing, which is performed during the data paging stage; and the last one is real-time on-line processing, which is typically done once per frame.

However, the pre-processing is playing an important role in VGTVS. The main purpose of the pre-processor is to gather the different types of data and transform them into a format that is readable by and quickly accessible to the run-time system. These tasks are inherently compute intensive and cannot be performed by the run-time system at high enough rates. Some of the basic data types within this domain include terrain geometry (in current implement, we choose the terrain model type of RSG (Regular Square Grid)), imagery (e.g. photo texture, pre-shaded relief maps, contour maps, etc.), and surface elements (e.g. vector data of roads, rivers and region signs in maps). The source data may come in a variety of file formats, which the pre-processor must translate to a single common format. Furthermore, to get higher frame rates, all the computing works that could be done before scene rendering, for example, calculating for the normal of points in DEM and the color values, should be finished during pre-processing. So it just read the results when needed during scene rendering and this does save more time.

Because of the data paging algorithms, VGTVS utilizes a quad-tree structure for organizing multi-resolution terrain data in a hierarchical manner. Each node in a quad-tree identifies a fixed, square area at a given discrete resolution. Additional constraints force the dimensions of the raster tiles associated with the nodes to be powers of two because of our LOD algorithm for terrain [1]. So both the height field and imagery are represented as regular grids, and we do not require additional compute time to triangulate the height field—the different discrete levels of detail of the geometry are rather represented by the pyramidal regular grid structure. In fact, we take full advantage of the maps in different scales and the data in geography database when we build the models of multi-resolution terrain. Because the information on maps and the data in geography database are the results what people have learned about their environments. So they are very valuable for the environment simulation. Furthermore, the different map scales data can be used to build the terrain pyramid structure directly. With such pyramidal structure, when eye-point is moving to Earth's surface, the scene data of different scales should change in turn as following: 1:2000 TT (ten thousand), 1:1000 TT, 1:100 TT, 1:50 TT, 1:25 TT, 1:10 TT, 1:5 TT and so on. Hence data cutting is also an important handing during data pre-processing. Figure 2 shows different scenes based on map data of different scales.

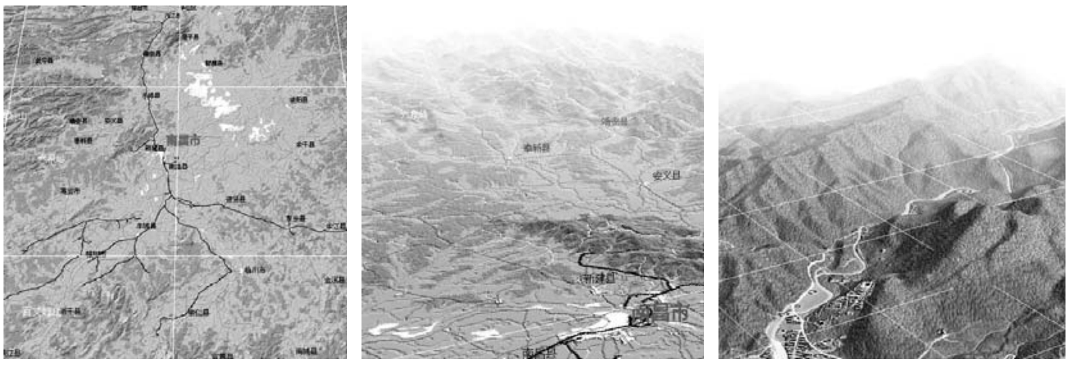
(a) Scene of 1:50 TT scale (b) Scene of 1:25 TT scale (c) Scene of 1:5 TT scale

FIGURE 2. Scenes Based on Map Data of Different Scales

In addition, because the data in different scales are based on different coordinate system, even the data in same scale may have their own coordinate origin, so it is very necessary to set up a uniform coordinate system for the data. Geography coordinate system is a good choice. The pre-processing of the data is done entirely in geography coordinate system (WGS-84). This ensures good registration between different source datasets and avoids the discontinuity problems often associated with flat-projected data. However, if a single, geocentric coordinate system were used, and assuming 32-bit single precision floating point is used to describe geometrical objects, the highest attainable accuracy on the surface of the Earth is half a meter. Clearly, this is not sufficient to distinguish features with details as small as a few centimeters. This lack in precision results in "wobbling" as the vertices of the geometry are snapped to discrete positions. To overcome this problem, we define a number of local coordinate systems over the globe, which have their origins displaced to the (oblate) spheroid surface that defines the Earth sea level and this is also used in other global systems [2].

4 Real-time Rendering System

Many factors determine the quality of a real-time 3D visual simulation. Two of these are the fidelity of the scenes rendered and the frame rate of the simulation, which are at odds with each other. High fidelity scenes provide a greater degree of realism, enhancing the effectiveness of the simulator as a training device or visualization tool, but slowing frame rate. High frame rates, in excess of 15 frames per second, provide smooth motion and improve interaction between the user and simulator, but allow less time to render individual frames. The challenge is to add as many features that enhance realism as are required for the intended use of the simulator while keeping the frame rate above the threshold at which the human eye is able to detect individual frames.

The purpose of the real-time rendering system is to support highly interactive frame rates during scene simulation while keep high detail at the same time. We have taken several advantages of new techniques to get high frame rates in VGTVS. These include hierarchical and tiled data structure, LOD algorithms for terrain, objects management and so on. Some important parts of these tasks are discussed in the following sections.

4.1 Hierarchical and Tiled Data Structure

VGTVS applies level of detail techniques to simplify both geometry and texture detail. This is necessary to maintain interactive frame rates as the global views that contain millions, or even billions, of surface polygons, with gigabytes worth of imagery. The dynamic scene manager is the center of importance of the visualization component. For the purpose of the real-time rendering, the scene manager must be responsible for updating the scene dynamically according to the user's movements, it has to take care of the level of detail (LOD) handling, and must perform the triangulation of the

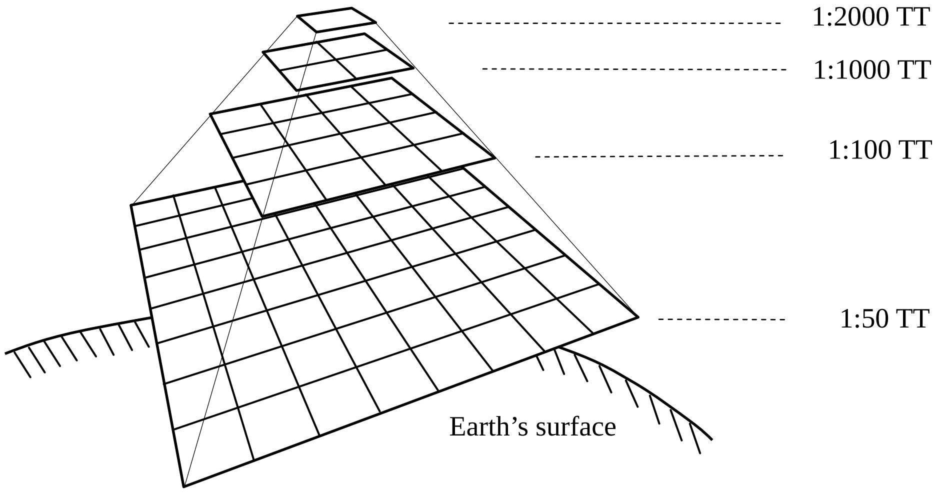

FIGURE 3. Pyramid Data Structure Based on Map data of different scales

surface. The actually rendered scene is only the currently visible part of a much larger virtual world, stored in the terrain database. The dynamic scene manager maintains the points of each terrain patch in a hierarchical data structure (actually quad-tree [3]). So the scene is subdivided into a set of different levels and also a set of rectangular patches, called the scene tiles, in each level. To accommodate data paging, level of detail management, and view culling, a quad-tree data structure is used to spatially subdivide and organize the terrain raster data. As mentioned above, VGTVS utilizes a quad-tree structure for organizing multi-resolution terrain data in a hierarchical manner in practice (figure 3). The globe is subdivided into a small number of pre-determined areas, each corresponding to a separate quad-tree.

Given a definite LOD of terrain, the dynamic scene manager proceeds in two stages. First is a coarse-grained simplification in which quad-nodes are selected at the appropriate resolution. This is effectual for a quad-tree can efficiently be transmitted in a depth-first order traversal.

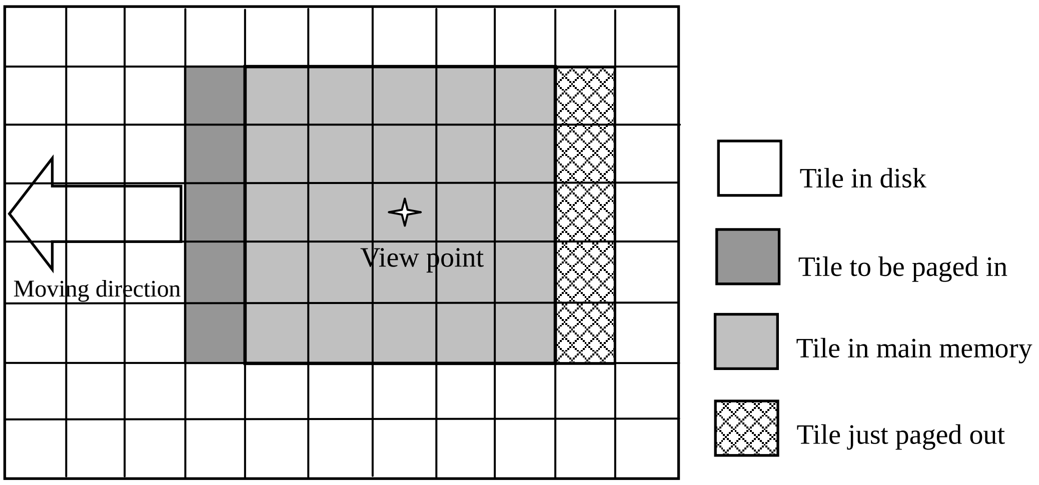

FIGURE 4. Paging Algorithm of Terrain Based on Data Tiles

When a curtain level is selected, followed by a fine-grained simplification in which individual vertices within each node are decimated. Whenever the observer's viewpoint moves across the boundary of the central patch of the scene tiles, new rows or columns of patches are loaded from the database; outdated or invisible ones are discarded as figure 4 shows.

4.2 Terrain Level of Detail

The scene manager also takes care of the correct LOD assignment for every patch. The appropriate use of different LOD for different parts of the visible scene can significantly reduce the number of geometric primitives that have to be rendered. Simply put a larger scene with fewer details can be displayed at the same frame-rate as a smaller one with more details. Furthermore, the use of different LOD can also lead to an enhanced realistic impression of the scene, since the farther the objects are, the lower we choose their LOD. This corresponds nicely with the human viewing system that cannot distinguish small details of objects that are far away.

At any time, the selected vertices in that quad-tree, which are currently marked to be displayed, form a restricted quad-tree (RQT) [4]. Whenever an update is scheduled, a RQT reflecting the new situation is extracted and provided for rendering the patch at the requested LOD. The restricted quad-tree, or an error-range quad-tree for incremental mesh refinement is the basic transfer and query unit. This unit is used for reloading or updating a terrain patch from the spatial database.

Moreover, the LOD strategy can differ between applications. In VGTVS There are four different parameters which can be modified interactively: the error threshold in the center of the scene map, the error growth with the distance to the viewpoint (radial growth), the error growth with the angular difference from the view direction (axial

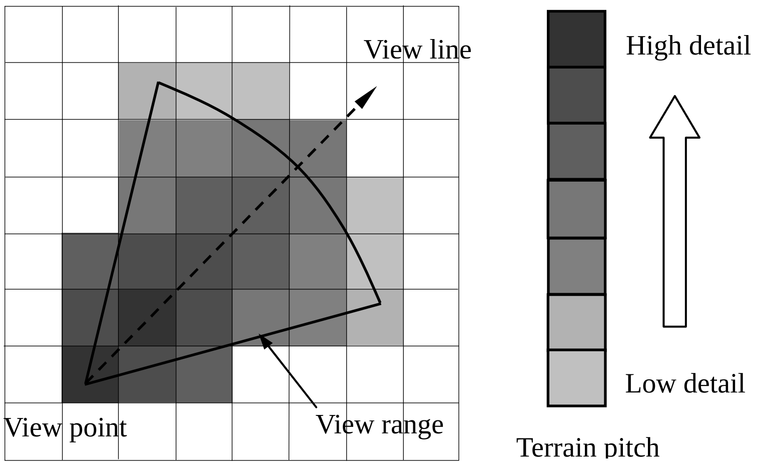

FIGURE 5. The Principle of Terrain Pitch for Setting View-dependent LOD

growth), and the axial growth weight. Figure 5 shows an example of such an error distribution. The patch below the viewpoint has the highest LOD of the terrain surface approximation. The high LOD runs along the view direction, and the accuracy gradually decreases with the distance to the viewpoint and view direction. For efficiency reasons, we associate one LOD value with each patch of the scene. However, this discretization of LOD has no adverse effects on the visual impression because the restricted quad-tree triangulation guarantees smooth LOD transitions between different terrain patches [1].

4.3 Object Management

Not only terrain surface data can be attached to every patch, but also other possible data types include texture information, land use coverage or other geographically referenced objects, and points or polygons with additional non-geometrical attributes. For every patch, the scene manager maintains the various data types in different data structures. Each data structure is appropriately chosen for the different

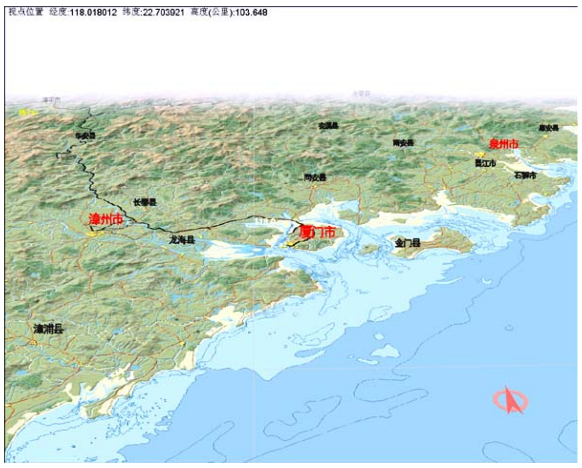
视点位置 经度:118.018012 纬度:22.703921 高度(公里):103.648

FIGURE 6. Rendering Scene with Vector Map Data as Surface Objects

data types, and the respective query and update mechanisms.

In current VGTVS, when eye point is far away from Earth's surface, we mainly take the vector data of different scale maps as the surface's objects. For instance, vector data of roads, rivers and settlement place (figure 6). We also use the data of geographical name lettering to label area. Of cause, we give the third dimension coordinate for each name in the three dimensions virtual space. As we all known, in a curtain level of scale map, cartographic feature data are also separated into different layers according to their importance and this can be used directly to define level of detail for themselves.

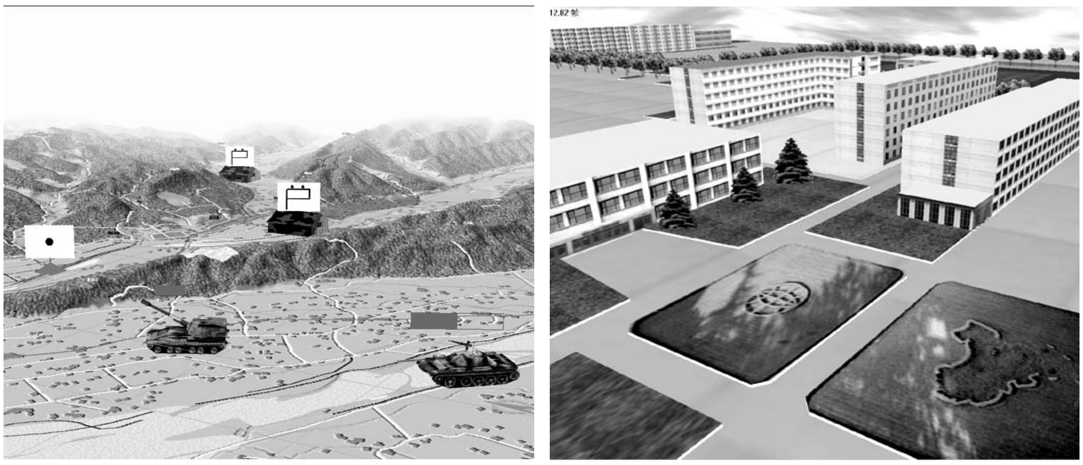
FIGURE 7. Add Three-dimensional Objects into Scene of Large Scale Data

Nevertheless, when eye point is closer to surface, it is not enough that just using the vector map data to rendering. Some objects with three-dimensional coordinates, for example, buildings, trees and cars, should be added in order to get high similitude (figure 7). At this time, the object manager will handle the display of static and moving protrusive objects. For the latter it also handles animation. Since hundreds to thousands of objects may appear in certain scenarios, the object manager must accept and handle multiple levels of detail for each model [5]. Individual objects will have levels of detail, and there will also be grouping strategies based on data patches of terrain. In general moving and static objects should be managed differently since they will have different grouping strategies and different ways to handle object detail (e.g. building textures). In current VGTVS, we define a bounding box for each object and build up hierarchical bounding volumes for each group [6]. It is efficient for visibility testing that culling to the frustum, which is bounded by the near and far clip planes and the four sides of the viewing pyramid. So the rendering system doesn't waste time processing objects that couldn't possibly appear in the final image.

5 Conclusions and Future Work

Up to the present, we have designed and implemented a real-time 3D visualization system, VGTVS, which integrates visual simulation techniques for interactive rendering and management of global terrain databases with query capabilities for spatial geographic data. The system allows visualization of terrain and other data types over the entire surface of the Earth, and manages very high resolution datasets at real-time rates, by taking advantage of hierarchical and tiled data structure. Our method has a small memory requirement and the visualization system can implement on PC of NT 4.0/Windows 2000 operating system. The system has met its original requirements for high interactivity and the ability to handle huge databases, and has proved useful for many visualization tasks.

Though we can handle huge graphical databases for real-time rendering, there is still some obvious “Popping” phenomenon when scenes change between different scales of map data. So our next step will focus on the algorithms to overcoming this shortage and then try to build up an integrated global visualization system based on Digital Earth.

References

[1] Chen Gang. A Research on Multi-resolution Surface Description and Real-time Rendering for Virtual Terrain Environment. Ph.D. EE. *Institute of Surveying and Mapping China,* October 2000.

[2] KOLLER,D.,LINDSTROM,P.,RIBARSKY,W.,HODGES,L.F.,FAUST,N., and TURNER, G. Virtual GIS: A Real-Time 3D Geographic Information Sys-tem. In *Proceedings of Visualization' 9*5, October 1995, pp. 94–100.

[3] Samet, H. The Quadtree and Related Hierarchical Data Structures. *ACM Com-putting Surveys* 16(2), June 1984, pp. 187–260.

[4] Renato Pajarola, “Acess to Large Scale Terrain and Image Databases in GIS”, Ph.D. EE. *Swiss Federal Institute of Technology Zurich*,1998,.

[5] Xia, J., and Varshney, A. Dynamic view-dependent simplification for polygonal models. In *Visualization ’96 Proceedings* (1996), IEEE, pp. 327–334.

[6] Hoppe, H. View-dependent refinement of progressive meshes. *Computer Graphics (SIGGRAPH ’97 Proceedings)* (1997), 189–198.