December 2003

Three Dimensional Representations of USGS Digital Elevation Model Data Using JAVA-3D: Implementation and Performance Issues

Mark Pendergast
Florida Gulf Coast University

THREE DIMENSIONAL REPRESENTATIONS OF USGS DIGITAL ELEVATION MODEL DATA USING JAVA–3D: IMPLEMENTATION AND PERFORMANCE ISSUES

Mark O. Pendergast
Florida Gulf Coast University
mpenderg@fgcu.edu

Abstract

This paper investigates performance issues that arise when displaying large 3 dimensional geographic data sets in an OpenGL/Java3D environment. Five possible schemes for the rendering of Digital Elevation Models from the United States Geological Survey are tested for memory usage and rendering processor efficiency. These implementations exercise and compare Java3d geometry alternatives including, interleaved data sets, indexed geometry, geometry strips, and primitive versus object based data formats. Of the schemes tested, the use of interleaved, primitive data formats in TriangleStrips was found to use the least memory and have the fastest rendering times.

Keywords: Java3d, GIS, elevation data, three dimensional computer graphics, performance

Introduction

Three-dimensional modeling of geographic information systems is becoming a reality due to the low cost of relatively powerful workstations, improved performance and memory capabilities of graphics hardware, and the introduction of cross-platform standards such as OpenGL and Java3d. This coupled with networked environments and parallel processing in desktop machines has allowed the development of "enormous virtual environments" (Fukatsu 1998). For example, (Gerstner 2002) presents an application that allows the visualization in three dimensions of rainfall mapped onto terrain data. This visualization and its ability to interactively rotate and zoom through data helps meteorologists to understand the interplay of different atmospheric processes and could improve rainfall estimates based on the measured radar data. Due to the large size of the data sets the application uses multiple resolutions to permit real-time visualizations. (Coors 1998) uses "load on demand" protocols in order to enhance performance for rendering distributed 3D models of cities in urban planning applications. These models can literally consist of tens of millions of data points requiring billions of calculations to display. In order to provide smooth animation while moving through a scene it necessary to recalculate and redisplay a scene 30 times a second. Thus despite the recent advances in memory capacities and processor speeds, programmers must still create efficient algorithms in order to meet the expectations of users.

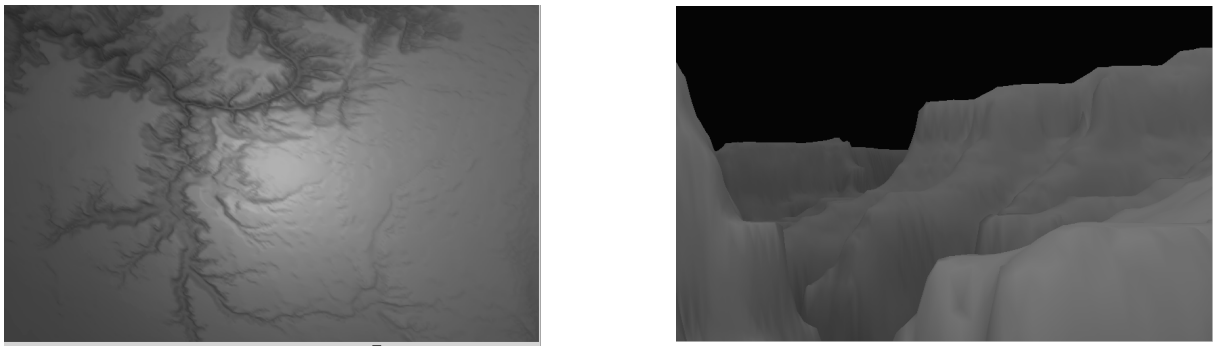
Application specific data handling techniques like the ones described in (Gerstner 2002) and (Coors 1998) are one avenue to enhance performance. A second, more generic approach is to use the most efficient rendering mechanisms that are built into the graphical development environment. Java3D, like most programming APIs, allows the programmer to perform a task in several different ways. This paper looks specifically at the task of rendering large terrain elevation data sets as provided by the United State Geological Survey (USGS) in a JAVA 3D environment with a goal of identifying the optimal Java3D programming constructs to use. Five different schemes for the rendering of Digital Elevation Models (DEM) are tested for memory usage and processor efficiency. The paper concludes with specific recommendations for JAVA3d developers.

Background

USGS Digital Elevation Model (DEM) Data

A Digital Elevation Model (DEM), consists of a sampled array of ground elevations (in meters) for positions at regularly spaced intervals. The basic elevation model is produced for the Defense Mapping Agency (DMA), and is distributed by the USGS and EROS Data Center in the DEM data record format (USGS 1998). The 1-Degree DEM (3 by 3 arc-second data spacing) provides coverage in 1 by 1degree blocks for all of the contiguous United States, Hawaii, and limited portions of Alaska. The 3 by 3 arc-second spacing results in a 1201 by 1201 array of elevations for each 1 degree block, or roughly one elevation measurement every 92 meters or 100 yards. This is approximately equivalent to that which can be obtained from contour information represented on 1:250,000 scale maps. A newer STDS format is also available (USGS 2002).

The DEM files have been used in the generation of isometric projections displaying slope, direction of slope (aspect), and terrain profiles between designated points. They can be combined with other data types such as stream locations and weather data to assist in forest fire control and with remote sensing data to classify vegetation. Other applications that have been developed include: determining the volume of proposed reservoirs, calculating the amount of cut and fill materials, and programs to assist in determining landslide probability (USGS 1998).



a b

Figure 1. Computer Images Created from DEM Data of the Grand Canyon Area

JAVA 3D

Java 3D is a high level, cross-platform interface for writing programs to display and interact with three-dimensional graphics. Java 3D is an extension to the Java 2 and is implemented as a layer on top of low level rendering APIs such as OpenGL and DirectX, see figure 2. The Java3D library provides a collection of high-level constructs, in the form of standard objects, for creating and manipulating 3D geometry, and structures for rendering that geometry. Java 3D provides the functions for the creation of imagery, visualizations, animations, and interactive 3D graphics application programs and applets. Java 3D primitives resemble OpenGL functions in many ways but are not dependent on having OpenGL as the low-level API (Walsh 2002, Bouvier 2000).

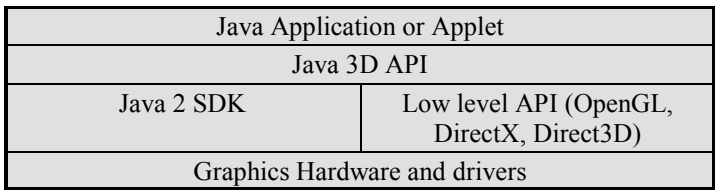


Figure 2. Java 3D Implementation Hierarchy

Java3D employs a scene-graph-programming model for describing graphical scenes. Scene graphs are hierarchies of objects used to render, store, and organize scene information. Objects exist to represent the "virtual universe", environment (lighting, fog), groups of objects, transformations (rotation, scaling), appearances (material and texture), and geometry (vertices, polygons). Figure 3 presents a simple scene graph. The manner is which a programmer selects and organizes Java3D objects can have a profound effect on the memory and processor requirements of an application. The next section presents and compares five possible variations for representing DEM data.

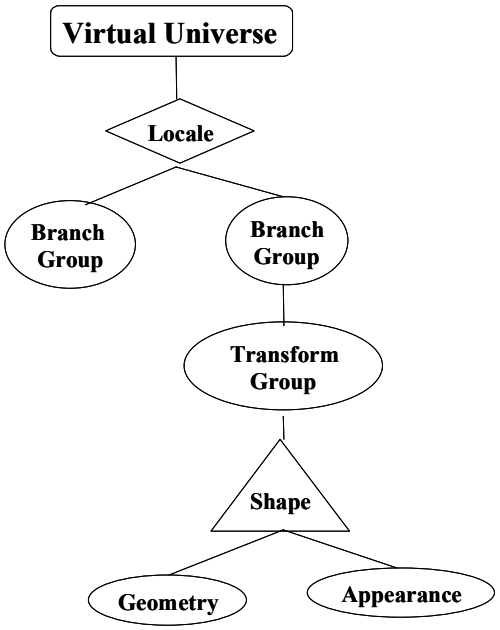


Figure 3. Generic Java3D Scene Graph

Representation Options

Available DEM data covering the continental United States covers an area of 25 degrees of latitude by 60 degrees of longitude and each degree is represented by a 1201x1201 matrix of elevations. If one wanted to model the entire United States at the 3 arc-second level of detail it would be necessary to create a model containing 25x60x1201x1201 or ~2.2 billion vertices. Limiting a model to data contained in just one DEM file still requires 1201x1201 or ~1.45 million vertices. If solid rendering is to be done the likely choice of geometrical primitives to use is a triangle (to insure that all points lie in a plane) figure 4. Modeling the 1201x1201 matrix using triangles requires 2.88 million triangles, 2*[rows-1]*[columns-1]. If the model is to be implemented on a Wintel or Linux platform with modest performance capabilities (1.5GHz processor and 256MB ram), and real-time fly-through is to be supported, then it is critical to construct a scene graph that optimizes both processor load and memory usage.

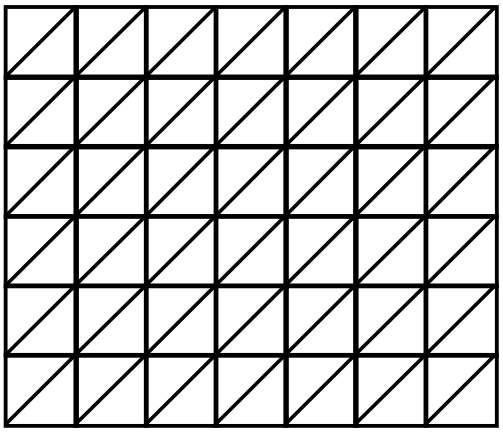


Figure 4. Pattern of Triangles Needed to Render a DEM Matrix

Java3D provides three geometry objects for rendering series of triangles, namely the TriangleArray, TriangleStripArray, and IndexedTriangleArray. Each of these objects in turn support data in several formats, as arrays of floats or doubles, as interleaved arrays of floats or doubles, and using Point3D objects and Vector3D objects. The data can be passed between the application and geometry objects either by value or by reference. For each vertex in a data set it is necessary to store the coordinates (X, Y, Z) either as 3 floats, 3 doubles or a Point3d/f object, and a normal vector (NX, NY, NZ), stored either as 3 floats, 3 doubles, or a Vector3d/f object. If lighting is not used, then a color (R,G,B) 3-tuple can be stored instead of a normal vector. Normals are vectors attached to each vertex indicating the surface's orientation for lighting and shading purposes. A computer's graphics adapter hardware uses the color the surface, its normal vector, and the location and direction of a light source in order to determine the actual color displayed on a screen (Foley, et al, 1997, p 723). Normal vectors are fairly computationally expensive to calculate, so its best to design algorithms to calculate them just once. When the data is stored in interleaved format, then the coordinate, color, and normal data is stored in one array. For example, the data array would contain the color for vertex 0, normal for vertex 0, coordinate for vertex 0, then the color for vertex 1, normal for vertex 1, coordinate for vertex 1, and so forth. When using a non interleaved format, then a separate array exists for color information, another for coordinates, and a third for normal vectors. The TriangleArray holds an array of triangles, each triangle requires data for 3 vertices. For the matrix of DEM data, this means each data point is stored in 6 different vertices. The TriangleStripArray consists of strips of vertices where the vertices 0,1,2 describe the first triangle, vertices 1,2,3 describe the next triangle and so forth. In this scheme, each data point is stored in two vertices. The IndexedTriangleArray has one vertex data set and a list of indexes into that set that describe the triangles. Thus each data point is represented by only one vertex. However, Java3d converts indexed geometry into non- indexed form since most graphics hardware does not support indexed rendering (Walsh 2002). This conversion entails significant memory usage, more than negating any advantage indexing provides over a TriangleStripArray. Tests showed using IndexTriangleStrips required **4 times** as much memory per data point over TriangleStripArrays.

To better understand memory usage and rendering times for the various settings, a DEM data viewer was implemented using five different configurations. The resources required for each configuration was measured for data sets with (1201x1201), (400x400), (240x240), (150x150), (120x120) elevation data points. The data viewer implemented a single directional light source and a single material.

Configurations

- Option A – TriangleStripArray, Interleaved Normals and Coordinates, pass by reference, data stored as an array of floats.
- Option B – TriangleArray, Interleaved Normals and Coordinates, pass by reference, data stored as an array of floats.
- Option C – TriangleArray, non- interleaved Normals and Coordinates, pass by reference, data stored as two arrays of floats (one for the coordinates, one for the normals).
- Option D – TriangleArray, non-interleaved Normals and Coordinates, pass by reference, data stored as two arrays, of Point3f objects for the coordinates, Vector3f objects for the normals.
- Option E – Indexed Triangle Strips (using the GeometryInfo object), data stored an array, of Point3f objects for the coordinates, normals were generated using the NormalGenerator object.

All tests were run on a Dell P-3 running Windows 2000 Professional, 1 GHz processor with 256 megabytes of RAM, and an ATI Radeon AGP graphics board with 32 megabytes of memory configured to support OpenGL. Memory usage was measured in terms of both peak memory requirements (memory used after the object was built but before garbage collection was run) and steady state requirements (memory used after garbage collection was run). Rendering time was measured in milliseconds starting just prior to the Canvas3D object beginning the rendering operation (preRender) to just after the image buffers were swapped (postSwap). The viewer was compiled using Sun Microsystems Java2 SDK version 1.4.0_01 (j2sdk1.4.0_01).

Results

The following tables and figures present the results of the test runs. Blank cells indicate that the system could not cope with the demands of the test. In those cases the workstation either hung or a memory allocation exception occurred.

Table 1. Rendering Times in Milliseconds

	1200x1200	400x400	240x240	150x150	120x120
Option A	1686	194	75	30	20
Option B		552	202	81	52
Option C		552	206	83	53
Option D		559	202	81	55
Option E			175	70	45

Table 2. Peak Memory Usage in Bytes

	1200x1200	400x400	240x240	150x150	120x120
Option A	69,965,904	8,567,064	2,811,720	1,796,904	989,544
Option B	207,554,992	23,234,992	8,489,392	3,450,880	2,284,480
Option C	207,555,016	23,235,016	8,489,416	3,450,904	2,284,504
Option D		76,973,680	27,653,456	10,975,576	7,087,576
Option E		242,000,872	59,534,864	32,393,624	21,329,912

Table 3. Steady State Memory Usage in Bytes

	1200x1200	400x400	240x240	150x150	120x120
Option A	69,197,584	7,709,584	2,784,784	1,093,512	703,032
Option B	207,365,400	23,045,400	8,299,800	3,245,064	2,078,664
Option C	207,365,424	23,045,424	8,299,824	3,245,088	2,078,688
Option D		76,803,560	27,651,224	10,805,456	6,917,456
Option E		27,863,624	10,857,416	3,554,752	3,251,584

Table 4. Steady State Memory per Data Point in Bytes

	1200x1200	400x400	240x240	150x150	120x120
Option A	48	48	48	48	48
Option B	144	144	144	144	144
Option C	144	144	144	144	144
Option D		480	480	480	480
Option E		174	188	158	226

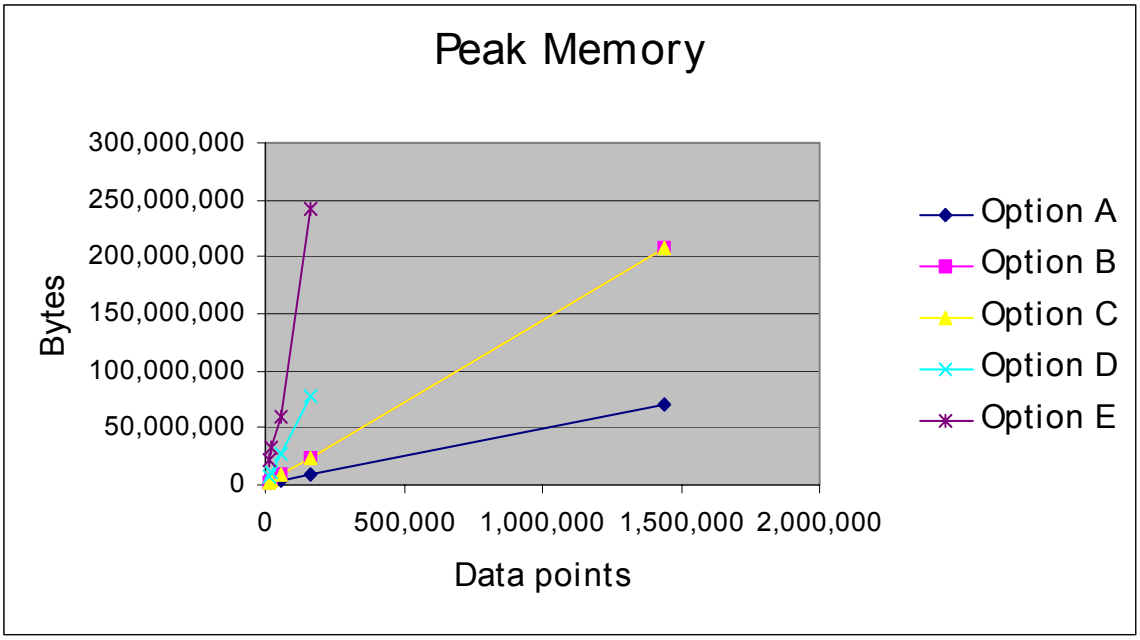


Figure 5. Peak Memory Versus Data Points

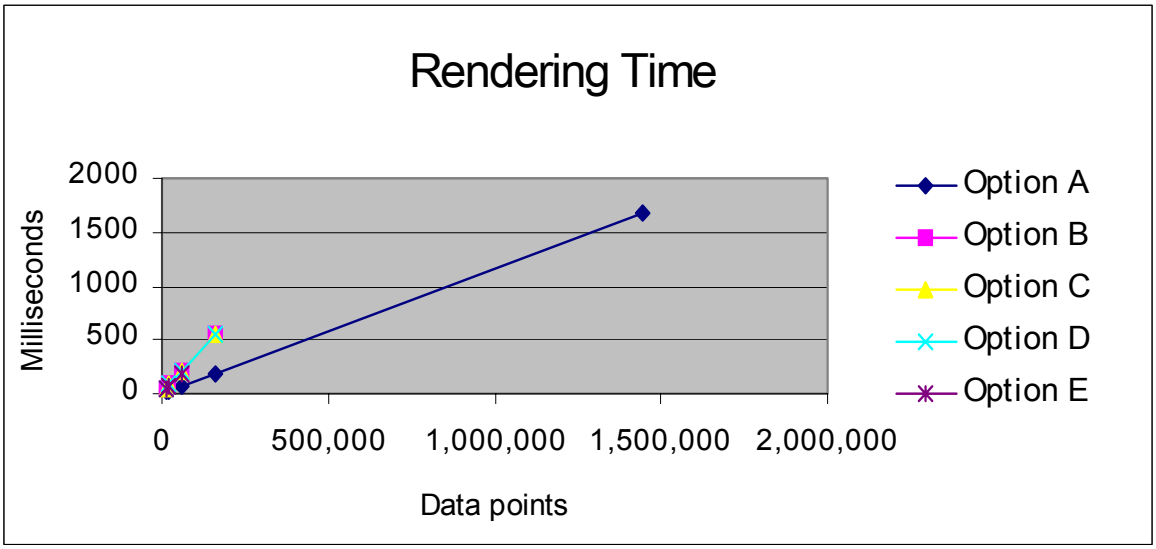


Figure 6. Rendering Time Versus Data Points

Implications for Application Development

The results clearly indicate that the use of TriangleStripArray object with by-reference-interleaved float arrays (Option A) is superior to the other options tested. The rendering time was 2.5 times faster than that of the other options and memory requirements are 1/3 of the other options. Option A was the only one that was able to successfully render an entire 1-degree by 1-degree DEM data set. When examining the differences between option B and C, it can be noted that interleaving data does not effect memory usage or rendering performance. Option D demonstrates that the use of JAVA3D objects, namely Point3f and Vector3f to hold geometric data increases memory usage by a factor of three and has no discernable effect on rendering performance. Object-oriented purists could make a case for the use of standard objects over the use of arrays of primitive data types on ideological grounds, however, this must be done with the realization that performance will be compromised.

Some specific recommendations for developers:

- Use the Strip object forms (TriangleStripArray, LineStripeArray, etc.) whenever possible. This is born out by the data and by programming tips in (Sun Microsystems 2002).
- Since Java3D internally converts geometry data to floats, the use of doubles is not warranted. The exception to this would be to avoid loss of precision during calculations, e.g. when calculating normal vectors.
- Unless you need a particular method provided by Point3f or Vector3f, it is better to store coordinate and normal information as arrays of floats or other native data formats (Sun Microsystems 2002).
- If lighting is not used, then the overhead of calculating and storing normals is not required. However, you will need to devise a scheme to assign colors to vertices in order for different surfaces to be discernable. In the case of DEM data, it is possible to assign darker colors to vertices with lower elevations and lighter colors to higher elevations, thereby mimicking the effects of overhead lighting on a scene.
- Even though Java3d internally operates on an interleaved array of floats for the data, interleaving the data does not seem to enhance performance or memory usage. Non interleaved data, i.e. separate arrays for colors, normals, and coordinates is much easier to manipulate and program.
- The use of indexed geometry is of questionable value. While it would on it face appear to save memory by not having to store vertex information more than once, the complexity of creating the index arrays (a separate one is required for coordinates,

colors, and normals), and the fact that Java3d converts it to a non indexed format anyway more than erases any advantage that it hold. It is possible that future graphics hardware will support indexed geometry on a cross platform basis.

- The test-bed viewer created for the purposes of this study placed all the geometry in a single Shape3D node. A better scheme would be to segment the DEM data into multiple zones, each in its own Shape3D node. This would enhance culling operations and improve fly-through performance.

The work presented in this paper is part of a larger ongoing project who's goal is to create a large virtual world based on USGS mapping data that allows real-time display and fly through capabilities. This virtual world can then be used as a basis for GIS applications in areas such as meteorology, flood and erosion control, wilderness fire fighting, environmental and growth management, among others.

References

Bouvier, D., *Getting Started with the JAVA 3D API*, Sun Microsystems Press, 2000.

Coors V., Flick S., "Integrating levels of detail in a Web-based 3D-GIS," in *Proceedings of the Sixth ACM International Symposium on Advances in Geographic Information Systems*, November 1998.

Foley, J.D., Van Dam, A., Feiner, S.K., Huges, J.F., *Computer Graphics Principles and Practice* (2nd ed.), Reading, MA: Addison-Wesley, 1997.

Fukatsu S., Kitamura Y., Masaki T., Kishino F., "Intuitive Control of 'Bird's Eye' Overview Images for Navigation in an Enormous Virtual Environment," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology 1998*, November 1998.

Gerstner T., Meetschen D., Crewell S., Griebel M, Simmer C., "A Case Study on Multi-Resolution Visualization of Local Rainfall from Weather Radar Measurements," in *Proceedings of the Conference on Visualization '02,* October 2002.

Sun MicroSystems, "JAVA3D API Performance Guide," Sun Microsystems, **http://java.sun.com/products/java-media/3D/collateral/1.2.1.perfguide.html,** 2002.

USGS, S*tandards for Digital Elevation Models*, U. S. Department of the Interior, U.S. Geological Survey, National Mapping Division, 1998. USGS EROS Data Center, DEM Data, **http://edc.usgs.gov/geodata/**

USGS, Spatial Data Transfer Standard (STDS), U. S. Department of the Interior, U.S. Geological Survey, National Mapping Division, 2002.

Walsh, A.E., Gehringer, D., *JAVA 3D API Jumpstart*, Upper Saddle River, NJ: Prentice Hall PTR, 2002.

Woo, M., Neider, J., Davis, T., Schreiner, D., *OpenGL Programming Guide* (3rd ed.), Reading, MA: Addison Wesley, 2000.