Virtual Worlds Modeling for Web Deployment

George K. Moschovitis, Kostas Karpouzis and Stefanos Kollias

Abstract-- In this paper we propose a design architecture for the graphics engine of a virtual reality system to be deployed over the web. The graphics engine is decomposed in high and low level components. The high level component manages the hierarchical organization of the virtual objects, while the low level component acts as the interface to the graphics hardware and is mainly responsible for operations, such as vertex transforms and polygon rasterization. Several methods for geometry representation, compression and culling are compared and associated with different geometry types. The use of OpenGL as a general-purpose machine to implement mathematical operations is considered in the context of programmable shaders. Finally a prototype application for virtual worlds modeling is presented along with detailed discussion on technical and implementation issues.

***Index terms*—3D graphics, VRML, Java, modeling, multi-pass rendering, bezier patches, portals, shaders, tessellation, height fields.**

I. Introduction

The Web has recently transformed from a static text-and-image medium to a broadcasting gateway with full multimedia capabilities. Increased network bandwidth and home computer hardware advancement have contributed to this tremendous progress; this change also means that web content needs constant upgrading, in order to keep up with the ever-growing demands and possibilities.

The major evolution, with respect to the nature of the web-deployed content, is 3D virtual worlds rendering and navigation. Traditionally, creation and rendering of such content has required expensive workstations, running equally expensive software and exploiting dedicated exotic hardware [2]. Major developments in the semiconductor industry resulted in drastic reduction of hardware cost, while the advent of new rapid application development systems promise similar reductions of the cost associated with software development. As a result, standards such as VRML 1.0 and 2.0 [4] or Movingworlds emerged, which promised high quality user interaction in dynamic environments. However, these attempts failed to come up with refresh rates and photorealistic techniques that are required to provide realism in demanding applications.

Our application is a 3D viewing and authoring application implemented with Java. Some impressive features of the Java language are exploited, including rapid application development, binary portability to most platforms, object-oriented methodologies for the design and implementation of the software and robustness, as well as a dynamic object graph, similar to that of a VRML world or an MPEG-4 BIFS file [16]. Various subsystems and commands are encapsulated in autonomous objects that can be extended or replaced at runtime by the end user. The main graphics engine utilizes OpenGL as a low level interface for geometry transformation and rasterization hardware. As a result, the application can be used in a variety of computer systems, ranging from entry-level home computers to multi-processor workstations.

II. The Graphics Engine

In most modern implementations the graphics engine is decomposed in high and low level components. The high level module is responsible for the hierarchical organization of the objects in the virtual world while the low level module handles the geometry transformation and rendering.

A. High level graphics module

The high-level graphics module manages the geometry database that models the virtual world. This database, referred in the literature as a scene graph (see Figure 1), is hierarchical in nature. The hierarchical organization, apart from being intuitive and flexible, augments the auxiliary data structures used to optimize the database traversal. Despite the tremendous advances in graphics hardware performance, specialized processing of different geometry types is required to achieve interactive frame rates. The graphics pipeline is thus split in alternate paths.

Potential geometry types include architectural interiors, outdoor scenes, terrain, skinned creatures, organic objects and more.

The engine is tailored towards indoor scenes. The portal/cell paradigm is appropriate for such kind of

The authors are with the National Technical Univ. of Athens, Electrical and Computer Engineering Dept., CS Division, 9 Heroon Polytechniou, 157 73 Athens, Greece
Corresponding author email : gmosx@softlab.ece.ntua.gr

geometry. The scene is segmented in convex cells that enclose the empty space. Those cells are interweaved through portals, in effect windows to other cells. A BSP algorithm can "portalize" the virtual world and generate the cells. The inter-cell visibility is calculated during a preprocessing step and is used for aggressive first order culling; up to 90% of the scene graph can be pruned by this filter. The remaining cells are clipped against the viewing volume. The BSP hierarchical structure allows for efficient clipping of whole branches of cells if the parent node is found invisible. A pleasant attribute of the portal/cell method is that is compatible with the alternate paths requirement as we can associate different rendering subsystems to the cells. Moreover we can associate transformation matrices to the portals, to allow for special effects like mirrors and space warps.

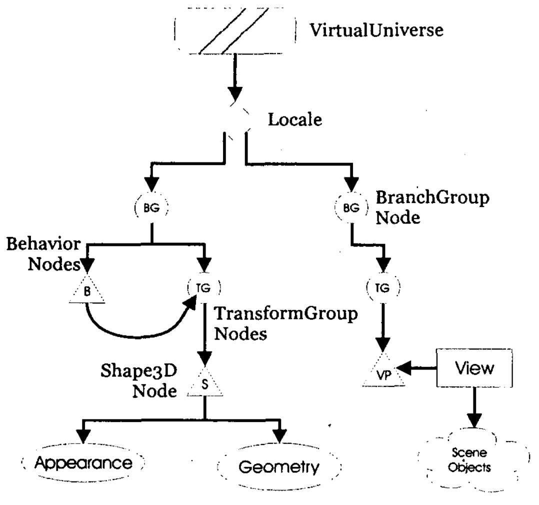

Figure 1: The Scene Graph describes the hierarchical nature of a virtual world.

Height fields are the de-facto standard solution for terrain rendering. More specifically we use the Real-time Optimally Adapting Meshes algorithm, which is based on a Binary Triangle Tree structure. Here each patch is a simple isosceles right triangle. Splitting the triangle from its apex to the middle of its hypotenuse produces two new isosceles right triangles. The splitting is recursive and can be repeated on the children until the desired level of detail is reached.

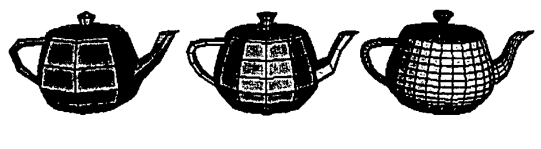

Figure 2: A teapot modeled using bezier patches shown with different subdivision steps

Three are the candidates for high fidelity rendering of organic objects. The surface subdivision method [6],[7] extends standard polygonal modeling techniques by recursively subdividing an existing mesh to increase complexity and smooth hard edges. The applied operator alters the surface of the object while leaving the original vertices intact. Non-Uniform Rational B-Spline surfaces [9] are polygonal approximations of an objects volume as defined by a set of analytically generated curves. Control vertices lying on or around the curve define these curves. Finally Bezier patches [10] are editable surfaces defined by four control points and the edges or curves between them (see Figure 2). The latter method was used in the prototype, because it is intuitive to model with, offers adequate control over the surface and generates polygons in optimized formats like tri-strips. However surface subdivision is promising and will be investigated in the future.

B. Low level graphics module

The low level module is responsible for transforming raw vertex coordinate data and rasterizing the polygons generated by the high level tessellator. The low level module is implemented on top of the OpenGL subsystem. OpenGL [3] is a general language for describing 3D graphics. Another view of OpenGL is as a software interface to graphics hardware.

Vertex transformation is relatively straightforward and is handled by OpenGL. Of interest is the skinning method for realistic animation of articulated objects. Traditional animation operates on objects, which consist of a hierarchical series of rigid bodies. This approach to animation produces artifacts, including interpenetration of object sections and gaps between object segments. Vertex blending enables "skinning" support that addresses these issues. Skinning is a technique that allows vertices to straddle more than one level of the object hierarchy. The triangles that include these shared vertices become stretched between hierarchy levels, thus eliminating gaps and hiding interpenetration. This allows artists a greater level of control over animation. In essence the vertex is transformed by the matrices of all "bones" that effect it ant he result is blended together. Modern graphics hardware accelerates this operation.

Texture mapping techniques [15] along with simplified light/material interaction models like gouraud or phong shading are typically used to produce relatively realistic scenes. Better results can be achieved using advanced mapping methods like bump mapping that model small irregularities on the surface of the objects.

Another use of texture mapping is to capture complex light interaction with the environment in the form of a lightmap. During a preprocessing step the light contribution on all surfaces is calculated using a radiosity solution [12]. This light map is sparsely sampled and compressed. Then the light map is applied as a separate step, when rendering the environment.

C. Programmable shading using modern Graphics Hardware

Programmable shader systems employ small programs to calculate the color for each pixel. Micro-programmable hardware is required to support a real-time programmable shader system. A better approach is to implement hardware programmability using the API, not by introducing a "backdoor" into programming the hardware directly, which is "limited at best, error prone at worst".

The OpenGL state machine is essentially a general-purpose machine to implement mathematical operations. In this view, the graphics hardware is just a subsystem applying calculations to a data stream - buffers are used as memory, internal paths become registers, logical and algebraic operations are implemented using alpha, stencil, and depth test, and texture and cube maps serve as lookup tables to look up values for scalar, 2D and 3D vectors (the latter even allowing for use of un-normalized vectors).

Operator	Method of evaluation
functionals e.g. y = f (x)	color tables and texture maps
conditionals e.g. if (A op B) then set stencil	alpha test and stencil test
arithmetic e.g. +, -, *	frame buffer / texture blending

Table 1: The use of OpenGL as general-purpose operator evaluator

One use of this scheme is the orthogonal illumination mapping technique [8], a texture-based multi-pass rendering method for performing hardware accelerated lighting calculations per-pixel rather than per-vertex. This technique has a number of advantages: it is simple, geometry-independent, and fast on today's commodity graphics cards.

III. Implementation

To demonstrate the applicability of the design and ensure the satisfaction of pragmatic constrains that restrict the current generation of hardware, a number of prototypes will be implemented. An early version of the Virtual Environment modeling tool was build to instigate the experimentation with the various concepts (see Figure 3).

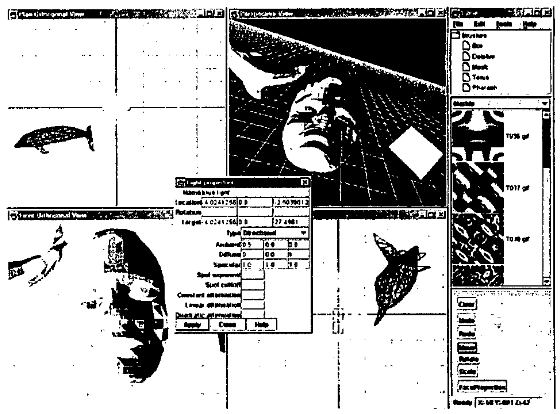

Figure 3: Positioning of virtual objects in the scene

Java was used as the implementation language for several reasons. The binary portability across various platforms may be attractive but the ability for rapid application development is certainly more important. Moreover, the object-oriented nature of java models successfully the hierarchical nature of the scene graph while allowing for code reuse thus minimizing the implementation effort.

The Java2 platform comes complete with Swing, a full-featured user interface API that was used to build the GUI of the application. Swing provides a rich set of lightweight and extendable components like dialogs, frames, tree controls, splitter bars, tables and more. We used those components to construct an impressive yet user-friendly interface the user can easily extend and customize. The extension mechanism is based on the Command pattern [1]. Every command or tool of the interface is an autonomous object that gets dynamically loaded by the java runtime. Advanced users can code their own tool-objects and place the binary classes in a special plug-in directory. The Application parses the directory at startup and seamlessly integrates all extensions to the toolset.

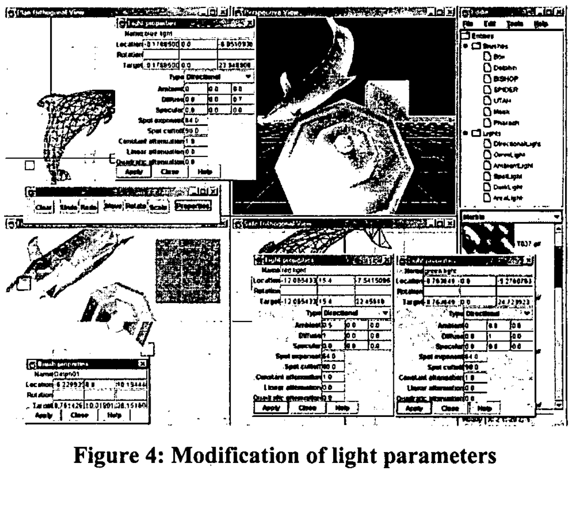

Figure 4: Modification of light parameters

The requirement for interactive frames and the general trend in the industry mandated the support of graphics acceleration hardware. The obvious choice was to use Java3D. This high level API is a collaboration effort between leading vendors that represents the cutting edge in graphics technology and also integrates other components of virtual reality like audio and stereoscopy.

However, we decided to implement the high level component of the engine ourselves in pure java. This white-box approach allows for greater possibilities of experimentation with various techniques and tuning of the pipeline according to the content. The low level component was implemented on top of OpenGL. In order to access the native GL subsystem the Java Native Interface was employed. More specifically the freely available Magician OpenGL library [5] was used.

The editor allows the user to create simple or complex objects manually using the built-in tools, or automatically using scripts. Loaders for commonly used formats including .plg .asc, .dxf and .3ds are provided for importing geometry from professional applications like 3D Studio MAX. Applying textures and changing material parameters like ambient, diffuse and specular components can modify the appearance of the objects (see Figure 5). The user can create virtual lights and position them to the scene. By altering parameters realistic light types of the physical world can be simulated. The user can also set atmospheric parameters like fog and mist. Simple animation functions can be applied to the objects. The final scene can be exported in VRML format [4], the de-facto standard for 3D graphics deployment in the Web. Common VRML browsers can be used to view the resulting .wrl file.

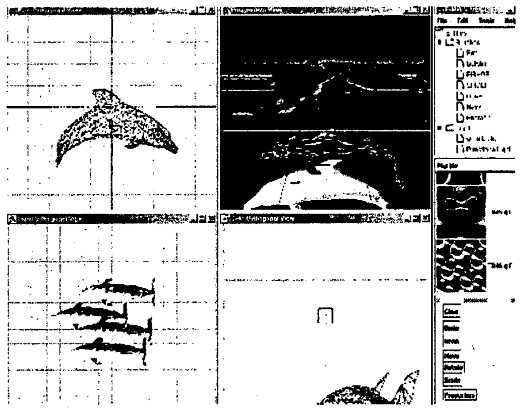

Figure 5: Different Materials applied to the same object

IV. Conclusions

This paper presents our vision for real-time graphics modeling and deployment for the Web. As discussed in the previous sections, specialized alternative paths in the graphics pipeline are required for efficient rendering of different types of scenery like terrain, indoor environments, live creatures etc. We also argued over the advantages of using OpenGL as a generalized signal-processing machine to support per pixel lighting and rendering of interesting shaders. Finally, a prototype modeling tool was presented and specific implementation details discussed.

In future work we intend to investigate the applicability of alternative rendering technologies like image-based or light field rendering [13],[14] as a stand-alone solution or as a means of optimizing the pipeline. We also consider the problem of geometry culling to be of paramount importance in the new era of huge virtual works so we will devote a considerable amount of our efforts in this area.

V. Acknowledgement

This work is funded by the Greek Secretariat of Research and Technology (Project PENED 99ED 478).

VI. References

[1] E. Gamma, R. Helm, R Johnson and J. Vlissides, "Design Patterns: Elements Of Reusable Object-Oriented Software", Addison–Wesley, 1995.

[2] A. Watt and M. Watt, "Advanced Animation And Rendering Techniques: Theory And Practice", Addison-Wesley, 1992.

[3] M. Segal and K. Akeley, "The OpenGL Graphics System, A Specification (Version 1.1)", Silicon Graphics.

[4] J. Hartman and J. Wernecke, "The VRML 2.0 Handbook", Addison-Wesley Developer Press, 1996

[5] A. Descartes, "Magician Programmers' Guide for Java Version 1.0", Arcane Technologies Ltd

[6] T. DeRose, M.Kass and T. Truong, "Subdivision Surfaces in Character Animation", SIGGRAPH 98, Orlando USA.

[7] D.N. Zorin, "Subdivision And Multiresolution Surface Representation", Phd Thesis, Caltech, Pasadena, California 1997.

[8] C. Everitt, "Orthogonal Illumination Maps", Ph.D. Thesis, http://www.opengl.org/News/Special/oim/Orth.html

[9] D. Terzopoulos and H. Qin, "Dynamic NURBS With Geometric Constraints For Interactive Sculpting," ACM Transactions on Graphics 13, 2 (Apr. 1994), pp. 103 – 136.

[10] J. Bruijns, "Quadratic Bezier Triangles As Drawing Primitives", Proceedings of the 1998 EUROGRAPHICS/SIGGRAPH workshop on Graphics hardware, 1998, p. 15.

[11] H. Hoppe, "Progressive Meshes", SIGGRAPH 96 Proceedings, 1996. pp. 99-108.

[12] S. E. Chen, "Incremental Radiosity: An Extension Of Progressive Radiosity To An Interactive Image Synthesis System", Conference proceedings on Computer graphics, 1990, pp. 135 – 144

[13] M. Levoy, P. Hanrahan, "Light Field Rendering", Proc. of ACM SIGGRAPH 96, pp. 31 – 42, NY, USA, 1996

[14] L. McMillan, G. Bishop, "Plenoptic Modeling : An Image-Based Rendering System", Proc. of ACM SIGGRAPH 95, pp.39 – 46, 1995

[15] P.S. Heckbert, "Survey Of Texture Mapping", IEEE CG&A, 6(11):56 – 67, 1986

[16] ISO/IEC JTC1/SC29/WG11 N3205, "MPEG-4 Multi-Users Technology (Requirements And Applications)", December 1999, Maui