Articled ID: 1009-0193(2002)04-0027-07

Object-oriented clusters
—A proposal for a new parallel processing paradigm

Philip Wong, Thomas Fischer
(School of Design The Hong Kong Polytechnic University, Hong kong, China)

Abstract: Clustering desktop computers in order to achieve cheap' supercomputing power' has become a popular approach for processing labor-intensive tasks for example in the academic and entertainment fields during the past few years. The distributed architecture and scalability of PC clusters provide a highly suitable hardware foundation in particular for the processing of large sets of data that are easily broken into pieces and distributed for parallel computation. However, required advanced programming skills, development time as well as data and algorithms that are not easily parallelized (as typically found in real-time shared virtual reality generation) are major obstacles on the pathway to flexible cluster application. In this paper, we propose a new cluster topology that takes advantage of the Java 3D data architecture to generate large virtual reality datasets. A discussion of our experimental investigation into this approach is given with special emphasis on our design perspective towards virtual reality production. The benefits of the proposed strategy are manifold and include easier and more economic development at comparatively gentle learning curves, significantly better runtime performance, and the reusability of generic software components. This can contribute to a wider range of real-time VR content to be produced and hence expand the application of virtual reality beyond the boundaries of the design field.

Key words: parallel processing; cluster topology; virtual reality; Java 3D
CLC number: TP311 **Document code**: A

0 Background

This paper proposes a new approach to cluster topology design as part of an ongoing project to develop a virtual gallery at the School of Design, Hong Kong Polytechnic University. From a designer's perspective, developing interactive 3D content for a narrative virtual environment continues being a long awaited dream. Despite various speculations, virtual reality is still not delivering what it is supposed to and widely unavailable to open applications such as Networked Education in Design.

It is one of the aims of this research project to develop strategies to achieve greater availability and flexibility of virtual reality by applying PC clustering technology. This type of technology as firstly been made available by the Beowulf Project in 1994. In this paper we use the term cluster for multi—PC setups that are optimized for speed rather than for reliability.

The nature and requirements of virtual reality production predestinate this field for parallel computation. Execution times last hours, days or even longer while no changes need to be applied to the underlying code base, the execution is CPU intensive and requires scalability and highly detailed and realistic resolution. One major hurdle on the pathway towards this ambition lies in the topological differences between virtual space, VR data struc-

Received date: 2002—06—15

tures and parallel hardware platforms required for processing and delivering 3D content. While VR geometry and behaviors are modeled after three-dimensional physical space and time, VR data (for example in VRML) is structured in tree hierarchies which are not easily accommodated by the linear (or on case of some particular designs n-dimensional) cluster structures.

Conventional cluster applications are developed on the parallel paradigm. Under this paradigm, a master node is responsible for breaking up tasks into smaller units and distributing these units to an array of slave nodes for computation. Once a slave node has finished computing its subtask, the results are send back to and reassembled on the master node, which keeps on assigning new tasks to the slave nodes. Current implementations employ either the Massage Passing Interface (MPI) standard or the Parallel Virtual Machine (PVM) library as a communication protocol within the cluster. However, these implementations of parallelism are heavily influenced by the von Neumann architecture, where a single central processor takes charge of all processes and has exclusive access to memory. The consequences are applications that execute sequentially. While robust and efficient, these parallel applications suffer from a number of weaknesses:

- Bottleneck created on the master node
- Limited portability
- Interfacing with graphic APIs not readily available

These limitations discourage VR developers from deploying the parallel paradigm to generate content. We believe however, that the recent maturing of Java as a fully developed object oriented programming language and the architecture of its 3D data structure can help changing this.

The essential concept of object-oriented programming is that components of an application are autonomous but at the same time communicates with each other. Each component performs a specified function that eventually contributes to accomplishment of a higher objective. Theoretically, components can exist on different computers to harness more computational power into the system. In fact, many applications achieve this by using technologies such as RMI or COBRA. However, dynamic generation of interactive 3D content with a cluster remains to be a threshold due to the fact that there was no object oriented graphics API that can make use of distributed processing, until the recent advent of Java 3D.

The first public specification of Java 3D, jointly developed by Sun Microsystems and SGI, was released in 1997. It is a high level, scene graph based API that uses either DirectX or the OpenGL low level API to take advantage of 3D hardware acceleration. Being fully object-oriented and platform independent, it offers some very interesting possibilities. For example, its Input Device class offers a convenient way to integrate peripherals like data gloves, motion trackers and head mounted displays into a system offering novel possibilities to experimental VR production. It is also possible to draw directly to the screen, bypassing the window system altogether. This feature alone offers new and interesting possibilities in the field interface design. Java 3D also allows the processing (compilation) of individual scene graph elements while allowing the later application of data changes and later assembly of pre-compiles elements into one complete scene graph. With Java, it is relatively easy and a common practice to program server-client systems that provide communication means among computers (e.g. within a cluster). Multi-threading is built into the core Java API which provides a way to efficiently use more than one CPU, possibly on different computers.

By combining the strength of the cluster concept with the object oriented Java 3D API, it is now conceivable that loose parallelism can be achieved on a cluster without using parallel programming. We propose achieving by dynamically mapping the structure of Java 3D scene graphs onto the hardware architecture topology of a cluster. As a possible application scenario, the following section proposes the design of a system, called the scene graph server, which generates dynamic interactive 3D content.

1 Design Principle

This section discusses a possible methodology for task distribution in a PC cluster which is uniquely offered by the scene graph structure of Java 3D. The following figure shows the structure of a Java 3D scene graph:

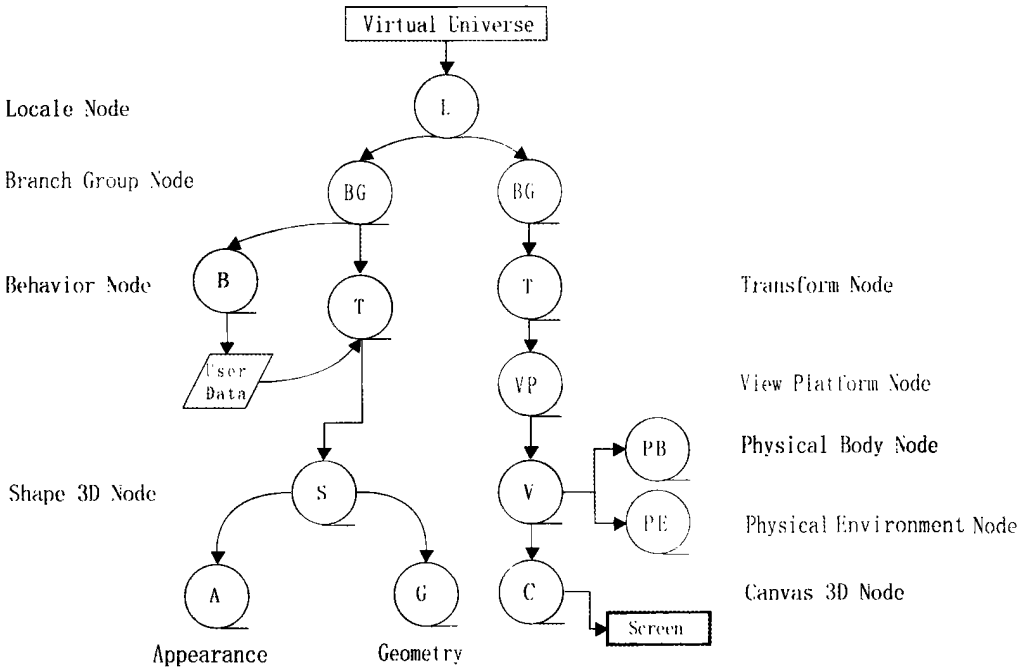

Figure 1 Java 3D scene graph structure

In Java 3D, the basic component is a node. A node essentially is an object that performs specific functions. The fundamental (or "root") node is called a "Virtual Universe" to which only a "Locale" object can be attached. "Branch Group" are objects that contain all the visible elements, as well as behaviors of those elements, present in the virtual environment and are attached to the Locale object. A branch group can be attached to other branch groups and their functions can be altered in runtime even when they are already compiled. All objects in Java 3D, together with their appearance and behaviors, are just hierarchical structures rooted at a branch group node.

The branch group object can be conceived as a 3D object that has a visual representation in the virtual universe. For example, a house, a car, a bird, a piece of cloud or a person are all branch groups in the virtual universe.

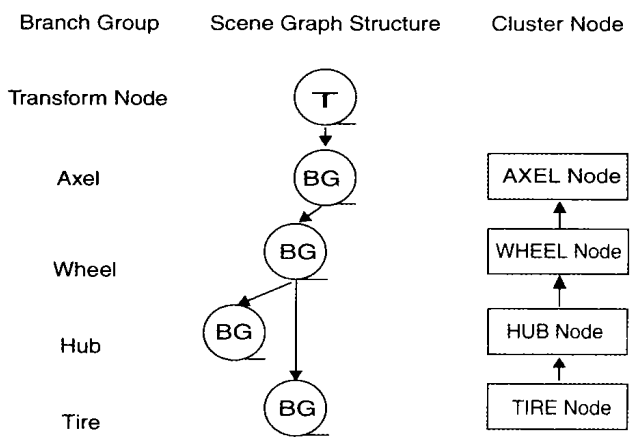

Figure 2 The Scene Graph of a Simple Wheel and the Corresponding Processing Nodes

The hub and tire are attached to the wheel that is attached to the axel. The axel is attached to a "Transform Group" that affects its behavior. The transform group causes the axel to rotate which in turn rotates the wheel, tire and hub. Conventionally, all these branch groups are compiled and executed on the same computer; but as the scene gets complicated, CPU load is quickly saturated. To solve this problem, individual nodes of a cluster can be used to process individual or groups of branch groups. This can be achieved by using Java 3D's possibility to pre—compile branch graphs. Each node is responsible for the following tasks (the blueprint server and final node are discussed in more detail in following sections):

- Maintaining the objects it is responsible for as assigned by the blueprint server
- Updating the state of the objects by querying the blueprint server
- Compilation of the branch group
- Serialization of the branch group
- Sending the branch group in the form of byte code to the final node
- Notifying the blueprint server of task accomplishment
- Notifying the blueprint server of local errors and perform fallback actions

Any 3D object in Java 3D, be it as simple as a ball or complicated as a flying bird, ultimately belongs to a branch group. A node of the cluster compiles the branch group into compiled format and transfers it over a network to another node where this branch group is attached to a branch group with a higher position in the scene graph hierarchy. A branch group usually makes references to other data objects. This is not a problem on a single computer. However, these references will break if the branch group has to be sent to another computer. The remedy is to make use of the Scene Graph IO class that comes with the J3Dfly Demo package from Sun. How 3D objects are assigned to which node can be decided on an application specific basis. A node can for example be assigned to a portion of the virtual environment spatially, processing all objects that are assigned to that area, or to individual users. At the end of the process, all branch groups are attached to a "root" branch group. This root branch group is attached to the virtual universe and is ready for rendering.

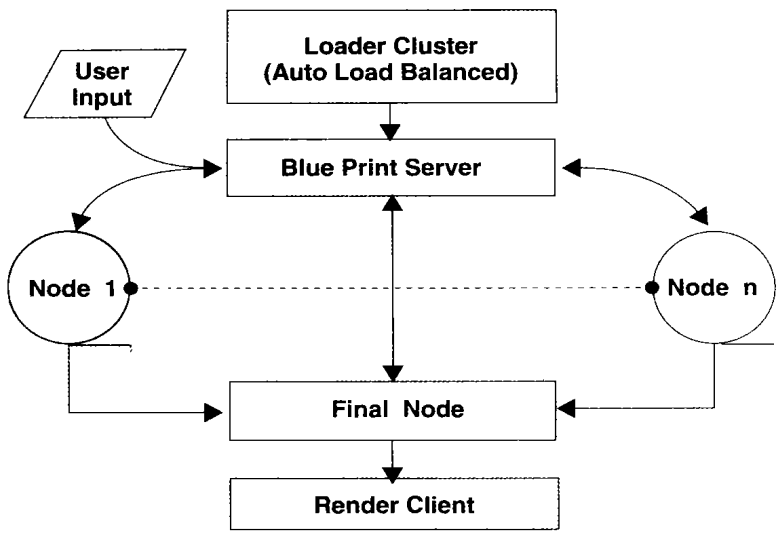

Figure 3 Structure Overview of the Scene Graph Server

The blueprint server acts as a centralized information center of the virtual environment. It keeps a "blueprint" of the virtual environment and state of the objects but not the objects themselves. It has the following major functions:

- Maintaining a database of the states of all objects.
- Distributing 3D objects to their designated nodes.

- Detecting changes in the virtual universe.
- Processing user input.
- Answering requests from nodes.

The blueprint is essentially a database that keeps track of attributes like spatial positions, conditions and ownerships of all 3D objects. It also supplies the querying node with instructions on how to reconfigure its tasks. For example, a node might need to send its output to a destination different from the one originally designated one due to changes in the scene graph structure.

Users are able to upload (or create) 3D objects to the virtual environment. In such cases, the blueprint server determines which node should the object's data be sent to and maintained. It only keeps track of the object's parameters and not a copy of the data itself. When changes are induced to a 3D object, either due to user input, interaction with other objects or by its own behavior, the blueprint server updates the parameters of that object. When queried by a node, these parameters are supplied. One advantage of using the blueprint approach is the ability to implement global changes to the virtual environment effectively, since all the nodes look at the same blueprint. Any changes and their effects are processed and reflected on the blueprint server before any node is aware of these changes. Depending on how data is distributed to the nodes, the database is structured accordingly.

The blueprint server is connected to another cluster that handles user—uploaded files. These files are 3D models created in other packages such as 3D Studio Max or Alias Studio. The uploaded files are processed by different loaders and converted into native Java 3D format. The user accesses a web interface that provides uploading services. The loading process is computation intensive so a load—balanced web server cluster is assigned to handle the workload. One last function of the blueprint server is to maintain constant communication with the final node. The function of the final node is to provide a "root" branch group for the other branch groups to attach to.

Output from all the cluster nodes are sent to a final node where the scene graphs is re—assembled. This final node maintains constant communication during runtime with the blueprint server to dynamically generate the root branch group. For example, if a user adds an object to the virtual environment, the final node creates a branch group node on the fly for this object to attach to.

Theoretically, the final node can also be the rendering client, provided this node is powerful enough. Another way for rendering is to employ a distributed rendering system such as VisAD developed by the Space Science and Engineering Center at the University of Wisconsin—Madison. However, rendering systems are out of the scope of this paper.

There are other methods of parallelizing an application with an object oriented programming language such as Java. For example, the DAMPP project employed applets to execute, on a remote computer, its tasks. Pseudo Remote Threads, created by Aashish N. Patil (a final—year undergraduate student at the Mumbai University, India) uses RMI to distribute Java threads over TCP/IP networks. These methods either require parallel programming or create a bottleneck in the system. In an environment that objects are constantly being created or deleted, these systems could become hard to manage due to their rigid design.

3 Conclusion

In a classic von Neumann architecture, bound by a CPU, only one process can be executed at any time and the time—sharing paradigm for multi—tasking as well as multi—threading cannot escape this boundary. With an object—oriented cluster architecture, the application achieves true multi—tasking with practically infinite scalability. By mapping the Java 3D scene graph data structure (tree—structured) to the linear topology cluster nodes, no parallel programming is required from the developer and a highly scalable structure can be provided.

Hence, parallel programming difficulties such as data synchronization, thread management and load balancing do not need to be addressed.

Since a virtual environment has potentially no spatial limits, this scalability is essential. Moreover, additional features like a terrain generation cluster, can easily be integrated into the system. It is important to note that there is no centralized control throughout the system. All cluster nodes merely performs their own tasks but collaborate with each other. Since the cluster nodes query the blueprint server once a task is accomplished, maximum throughput from that node can be guaranteed. At the moment, this architecture resembles other cluster architectures in that it is highly purpose centered and does not easily accommodate other applications. However, assuming that a development like the above proposed one would inspire other Java APIs to also make use of partial data structure compilation, object oriented clusters promise more flexibility classic clusters can offer.

Acknowledgements: The authors gratefully acknowledge the support from the academic and research staff at the School of Design, its Design Research and Technology Center, and the Interactive Systems Design stream at The Hong Kong Polytechnic University. This research is supported by The Hong Kong Polytechnic University's Large Equipment Fund.

References:

[1] Card S K, J D MacKinlay, B Sneiderman. Readings in Information Visualization. Using Vision to Think[M]. California: Morgan Kaufmann Publishers, 1999.

[2] Eckel B. Thinking in Java. The Definitive Introduction to Object—Oriented Programming in the Language of the World Wide Web(2nd ed)[M]. Englewood cliffs: Prentice Hall, 2000.

[3] Falk, L C Ceccato, C Hu, et al. Towards a Networked Education in Design[A]. Beng—Tiang Tan, et al. CAADRIA2000 Proceedings of the Fifth Conference on Computer Aided Architectural Design Research in Asia[C]. Singapore: 2000. 157—167

[4] Fischer T, C M Herr, C Ceccato. Towards Real Time Interaction Visualization in NED[A]. Clayton M J. ACADIA2000 Conference Proceedings[C]. Washington: Catholic University, 2000.

目标导向群—关于新并行处理范例的研究

菲利普翁, 托马斯西彻
(香港理工大学 设计学院, 中国 香港)

摘 要: 为取得"超计算功能", PC 电脑群已成为一种很受欢迎的方法, 这种方法能够处理诸如在过去几年里学术领域和娱乐领域的劳动密集型工作。而分布式的体系结构和可测量性的个人电脑为并行计算过程中易分散的大量固定数据提供了十分合适的硬件基础。然而, 这一过程需要先进的编程技巧, 大量的调试时间以及数据处理和计算规则并行的困难性(实时共享虚拟现实产生的典型)是灵活运用导向群的主要障碍。设计一个新的拓扑群, 这一拓扑群利用 Java 3D 数据结构以产生大量虚拟现实数据结构。但是专家们对实验研究中给予特别重视的关于虚拟现实作品的设计方法产生了争议。在相对平缓的识别图表曲线的过程中, 值得关注的是执行程序的运行时间以及软件类的组成部分的可利用性。这有助于建立一个广范围的实时的虚拟现实环境, 以促进设计方案的提出, 从而能越过设计领域, 扩大虚拟现实的应用范围。

关键词: 并行处理; 拓扑群; 虚拟现实; Java 3D
中图分类号: TP311 文献标识码: A

(上接 26 页)

概念设计中机器学习技术的运用

唐明晰
(香港理工大学 设计学院 技术设计研究中心, 中国 香港)

摘 要: 设计中开发计算理论的一个挑战是必须能支持计算机制的有效运用, 这一机制允许从设计专家那儿或设计样例中取得产生, 累加和转换的设计知识。而其中的一个方法是把机器学习机制综合成基于知识的支持系统, 以模拟设计过程初级阶段, 使设计成为一个增加和诱导学习的过程。模拟的需要产生于在不同的提取阶段获取, 提炼和转移设计知识的需求, 从而使得能轻而易举的熟练操作。在设计中, 现有的知识产生于过去的设计解决方案, 而过去的解决方案提供的反馈信息能更新和提高设计理论知识基础。但是, 没有学习接受能力, 设计系统不能反映设计家们在这一领域的成长经历, 也不能反映设计家们从以往设计案例中提取知识的能力。在此提出了方案设计和效力评价中的三种方法。

关键词: 概念设计; 机器学习; 知识
中图分类号: TP311 文献标识码: A