Assignment: Scene Graphs in Computer Graphics Courses

Dennis J. Bouvier
Saint Louis University
djb@acm.org

Abstract

A number of published papers recommend teaching scene graphs in the introductory computer graphics course [Bouvier 2002; Cunningham 1999; Hitchner and Sowizral 1999; Wolfe 1999]. However, little has been published concerning how to effectively use scene graphs in the introductory computer graphics course. This paper summarizes possible scene graphs exercises and teaching experience of the author.

1 Introduction

Scene graphs are data structures representing 3D geometry and other scene properties (e.g., lights and viewer position) in a hierarchical fashion. Scene graphs have been around for years in both academic and commercial settings [Brown 1995]. For several years, scene graphs were primarily used in Open Inventor [Ames et al 1997; Foley et al 1996; Wolfe 1999]; however, the recent popularity of the Java 3D API has attracted more attention to the use of scene graphs for both programmers [Baker 2002; OpenRM 2002; Sun 2002] and educators [Bouvier 2001; Bouvier 2002; Brown 1985; Cunningham 1999; Cunningham and Bailey 2001; Hitchner and Sowizral 1999; Wolfe 1999]. Most recently, open source projects have developed scene graph APIs as well [OpenRM 2002; Open Scene Graph 2002; OpenSG 2002].

Several educators advocate the use of scene graphs for a variety of pedagogic reasons [Brown 1985; Bresenham 1999; Cunningham 1999; Cunningham and Bailey 2001; Hitchner and Sowizral 1999; Wolfe 1999]. However, even within the publications advocating the use of scene graphs in the classroom, little advice has been given on how to use them in the classroom. This paper addresses this void.

2 Why Use Scene Graphs

Some may argue, 'one must know about scene graphs to get a job in graphics.' Others might argue that scene graphs are the way of the future for computer graphics. Neither of these reasons warrants the inclusion of scene graphs in graphics courses. However, there are good reasons as put forward by Cunningham [1999; 2000] and Wolfe [1999] as well as others [Cunningham and Bailey 2001; Hitchner and Sowizral 1999].

Scene graphs are useful in a computer graphics course as an abstraction of the 3D graphics rendering system. The educational power of the scene graph lies in the ease with which a complex scene can be represented and the corresponding ability to create and comprehend these complex scenes.

Scene graph concepts are useful in a course even when a scene graph API is not used in programming projects. Scene graphs are useful as design tools and documentation when the concepts are taught and used in the course.

3 What is a Scene Graph

A detailed technical explanation of scene graphs would go on for (possibly hundreds of) pages [Ames et al 1997; Cunningham and Bailey 2001; OpenRM 2002; Wernecke 1994]. Succinctly stated, a scene graph is a data structure and/or graphical representation of a hierarchical 3D scene. The data structure is typically a directed acyclic graph (DAG) in which the nodes of the DAG represent geometric transformations, geometry, lights, the viewer, and possibly other features of the 3D scene.

A particular scene graph implementation or product may be a file format (e.g., VRML [Ames et al 1997]), a programming API (e.g., Java 3D [Sowizral et al 1995]), or both (e.g., Open Inventor [Open Inventor 2002]). Scene graph files require a viewing or display program while an API is used to create an independent application. Section 4.3 presents more on this topic.

3.1 A Simple Scene Graph Representation

For the purposes of illustrating some of the assignments suggested in this paper, a simple scene graph representation is presented here. For a more complete explanation, see the cited references at the conclusion of this paper.

A scene graph is composed of at least three kinds of nodes: the root, transform, and visual object nodes. Figure 1 shows one possible set of graphical representations of the three basic scene graph nodes. The organization of these nodes in a DAG represents a scene graph.

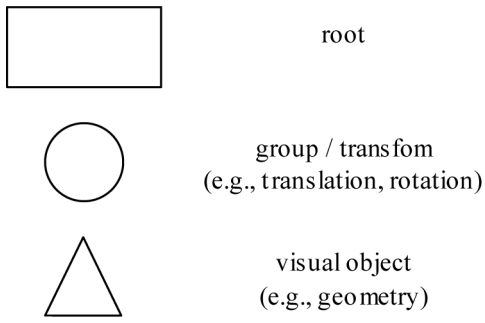

Figure 1 Simple Scene Graph Node Representations

The root node establishes the point of reference for the other nodes and can be considered the center of the 3D scene. There is only one root node in a scene graph and all other nodes in the scene graph are referred to directly, or indirectly, from the root node.

The group node can simply be a collection of references to child objects or transformations of its child object(s) (alternatively viewed as scaling, rotating or translating the reference space for the child objects).

The visual object node is geometry, lights, or other visual content. For the purposes of this discussion, the visual object will always be geometry. One detail being ignored is how material properties (i.e., reflectance coefficients and surface normal vectors) are specified in a scene graph.

A scene graph is typically organized into two major branches: the content branch and the view branch. Having the viewer information in a separate branch of the graph emanating from the

root of the scene graph allows the view position to move (or remain stationary) independently of the content.

Also missing from this discussion of scene graphs are dynamic elements (i.e., nodes for interactivity and animation) and other details, which vary by implementation. The missing details are not necessary to this discussion. The interested reader is referred to the appropriate documentation [Ames et al 1997; OpenRM 2002; OpenSG 2002; Sowizral et al 1995; Wernecke 1994].

3.2 A Simple Scene Graph Example

Consider a 3D scene consisting of a robot arm and a cube. The robot of this example consists of a base, a vertical shaft, a horizontal arm, and two 'fingers.' Figure 2 shows one such scene. The x-, y-, and z-axis orientation is shown for reference in this discussion, but is not part of the application.

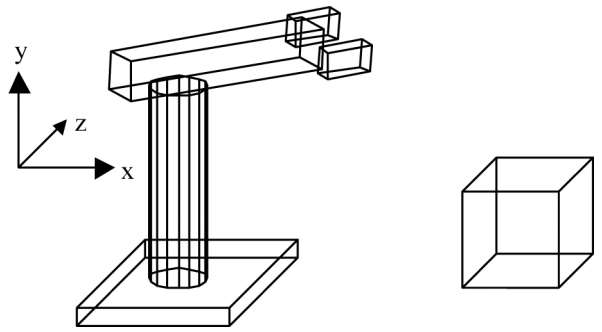

Figure 2: Rendering of Robot Arm with Cube

Figure 3 shows one possible scene graph for the scene of Figure 2. The scene graph is composed of a root node, eight group nodes, and six visual object nodes. The visual object nodes in this scene graph represent geometry of the scene.

As explained above, group nodes may transform their children. Consequently, the composition of the scene graph indicates which visual objects move independently and which move in a coordinated (i.e., hierarchical) fashion. For example, the cube may move independently of the robot (and vice-versa) since the cube visual object is a child of group 2 and none of the robot is. On the other hand, group 3 can move the entire robot, while group 5 can move just the upper parts of the robot.

Of course, other scene graph compositions are possible. However, not all scene graph compositions will allow the appropriate hierarchical behaviors. To design a proper scene graph composition requires having the knowledge (as specified, or through application understanding) of the hierarchal relationships of the scene's contents.

4 Scene Graphs in Graphics Courses

When adding new material to a course the instructor is faced with the issues of how to get the students to learn the new material. Adding a new topic to an existing course should be more than adding a lecture or some new reading material. Homework assignments, programming projects, and test questions that correspond to the new material is critical for learning.

Adding scene graphs to a computer graphics course is no different. The instructor considering adding scene graph material to a computer graphics course needs educational activities and testing. Obviously, a student can be required to use some scene graph API to create a graphical program, but what else is there?

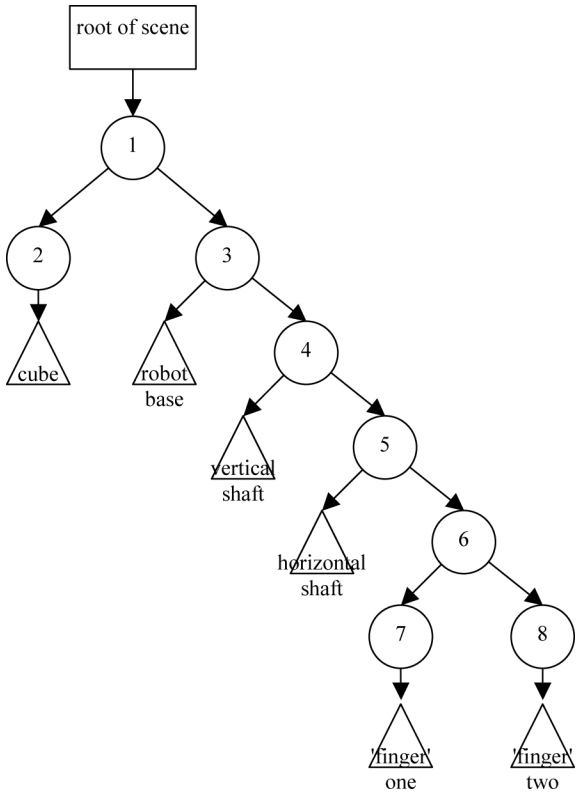

Figure 3: Scene Graph of Robot Arm with Cube

4.1 Possible Student Assignments

Below are four possible student assignments involving scene graph knowledge. A course could use all four, but typically assignments A, B, and C would be used for undergraduate courses while assignment D is probably most suited for graduate level instruction.

Assignment A: Create a Scene Graph

The first type of assignment is similar to the simple example given in Section 3.2 above. The student is given the description of a scene and asked to design a scene graph. Again, to design a proper scene graph composition requires having the knowledge of the hierarchal relationships of the scene's contents. Consequently, these details should not be absent from the scene description.

A student can undertake this type of assignment very quickly after having been exposed to the basic scene graph information. The student scene graphs can be hand drawn or involve computer tools.

This assignment serves to reinforce the basic concepts and issues involved in hierarchical compositions and (hopefully) increases understanding. More challenging assignments of this type include more advanced scene graph issues such as viewing, lighting, scoping, behaviors and interaction.

Assignment B: Scene Graph 'Understanding'

In this assignment a student produces a procedural representation for a given scene graph. This type of assignment can be undertaken only when the student knows about both scene graphs and some low-level implementation mechanism (e.g., OpenGL).

Table 1 A comparison of Scene Graph features

API/Language feature	Open Inventor	Java 3D	OpenSG	Open Scene Graph	Performer	VRML
API or Format	API	API	API	API	API	Format
API Host Language	C++	Java	C++	C++	C/C++	N/A
Documentation	Spec., tutorials, books	Spec., tutorials, books	Spec., tutorials	Spec.	Spec., programer's guide	Spec., tutorials, books
Version (as of 3/1/02)	2.1	1.3	1.0	0.8	2.5	VRML97
Source	Open	Closed	Open	Open	Closed/Open	N/A
Low-level	OpenGL	OpenGL or D3D	OpenGL	OpenGL	OpenGL	None
Organization	SGI	SUN	OpenSG	OpenSceneGraph	SGI	Web3D
OS Platforms	Unix	Windows & Unix	Windows & Unix	Unix	IRIX/Linux	Many

If a student knows about a low-level mechanism that features a transformation stack, then the translation is made from the scene graph DAG structure to the order of appropriate transformation operations with the appropriate stack operations. Completing this assignment gives students an appreciation of stack operations. Starting with the scene graph concept is an easy way to show students a meaningful use for the transformation stack.

In completing this type of assignment the student is forced to think about how the matrix stack implementation is useful in interactive or animated applications. Variations on such assignments can include optimization or other advanced issues.

Assignment C: Using a Scene Graph Language

A student produces a graphical application using a scene graph API or language. The level of difficulty ranges from fairly easy when implementing simple scenes using VRML to fairly complex when doing complex tasks using a scene graph API. The student may be given a specific scene to create or be left to design one (usually under some guidance or constraints).

The project should require the representation of some hierarchical object (e.g., the robot of the previous section) and animation of the hierarchical object. The power of the hierarchical representation may be best illustrated if the same, or similar programming project is also done without a scene graph.

Assignment D: Implement a Scene Graph System

The implementation of a scene graph system could be undertaken as a semester long project, or parts of it could be implemented as a single assignment. Open source scene graph APIs are of particular interest for this type of assignment (see section 4.3) as they allow implementation of advanced features without having to build an entire infrastructure.

A project of this type can require extensive knowledge of scene graph operations, a low-level graphics API (e.g., OpenGL), and significant programming experience. Consequently, this type of assignment would typically be reserved for graduate students.

4.2 Assessment

Assignments A and B lend themselves to in-class test questions. The assessment questions can be similar to the assignments taking care not to make them excessive in size. For example, give the description of a small application, such as the robot example, and ask the student for a scene graph representation of the scene. Or, give the student a graphical representation of a scene graph and ask the student how it can be implemented, or how elements of the scene graph related to each other.

Proficiency in the tasks of Assignment C can be tested in small programming related test questions including completing some code, modifying some code, questions regarding understanding of some code example, or API detail questions.

Due to the nature of Assignment D, it does not lend itself to assessment as Assignments A and B. However, interesting questions can be posed by asking how some advanced feature might be implemented or what obstacles exist to its implementation. An open question of this sort should be graded with respect to the time allotted for answering.

4.3 Selecting a Scene Graph Implementation

A course can utilize scene graph concepts without requiring the use of a scene graph API or language. However, if programming assignments are to be made (i.e., assignments C or possibly D), then a scene graph API or language will probably selected for class work.

Making a complete comparison of scene graph APIs and languages could involve an extensive exploration of features and performance. However, such a comparison is probably not necessary for making the selection of a scene graph for educational purposes.

There are several factors to consider in selecting a scene graph API that have little to do with the design of the API (e.g., platform, host language, and documentation). Table 1 summarizes the features most likely to be relevant in making the choice.

As previously mentioned, a scene graph implementation is a file format, a programming API, or both. This distinction determines if a particular scene graph definition is suitable for particular educational experiences. Creating a scene in VRML is not the same as creating an application in Java 3D.

The comparison attributes include those distinguishing the execution environments for each scene graph language may execute in which includes the operating system platform, low-level support requirements, and API host language.

The version number can be an indicator of stability or of the breadth of features present. Approach products with version numbers less than 1.0 with care.

Related to the maturity of a product is the level of available documentation. Older products tend to have more resources, such as tutorials and books, while newer products may have little more than the specification document to work from. The level of available documentation can make a difference in student success (or failures).

Whether or not the source code for a scene graph API is freely available can impact the possibilities of some programming assignments. As mentioned in Assignment D, being able to add advanced features (e.g., optimizations) to an existing scene graph API can be a reasonable assignment for upper level students.

Table 1 is not a comprehensive list of scene graph APIs and languages nor a complete comparison of scene graph languages listed. Table 1 is intended to show some scene graph options and decision criteria. Some scene graph languages missing from this list include OpenRM [OpenRM 2002], SSG [Baker 2002], and X3D [Web3D 2002].

5 Conclusion

An informal search of web resources found a number of universities making scene graph programming assignments for computer graphics students. The most common assignment required the student to create a graphics application using Java 3D. Unfortunately, the assignments found represent a small fraction of computer graphics courses. Further, simply creating a scene graph does not reveal the power of this representation.

The student assignments and other information, including the comparative list of scene graph choices, presented in this paper should enable additional instructors to use scene graph material in their courses.

The author has exercised assignments A, B, and C in undergraduate courses over the last two years with some success. Students utilizing scene graphs representations for 3D scenes are able to produce more complex animations with less work. The author is anxious to have graduate students so that assignment D can be tried.

References

AMES, A., NADEAU, D. AND MORELAND, J. 1997. *VRML 2.0 Sourcebook*, Wiley.

BAKER S. 2002. SSG: A Simple Scene Graph API for OpenGL, http://www.woodsoup.org/projs/plib/ssg/.

BOUVIER, D. 2001. From Pixels to Scene Graphs: The Evolution of Computer Graphics Courses, *SIGGRAPH 2001 Conference Abstracts and Applications*, August 2001

BOUVIER, D. 2002. From Pixels to Scene Graphs in Introductory Computer Graphics Courses, to appear *Computers and Graphics*.

BROWN, M. 1985. *Understanding PHIGS: The Hierarchical Computer Graphics Standard*, Template Software Division of Megatek Corporation.

BRESENHAM, J. 1999. Teaching the Graphics Processing Pipeline: Cosmetic and Geometric Attribute Implications, *Proceedings of Graphics and Visualization Education 1999*, July 1999.

CUNNINGHAM, S. 1999. Re-Inventing the Introductory Computer Graphics Course: Providing Tools for a Wider Audience, *Proceedings of Graphics and Visualization Education 1999*, July 1999.

CUNNINGHAM, S. 2000. Powers of 10: The Case for Changing the First Course in Computer Graphics, *The Proceedings of the Thirty-first Technical Symposium on Computer Science Education*, Austin, TX, March 2000, 46-49.

CUNNINGHAM, S. AND BAILEY, M. 2001. Lessons from Scene Graphs: Using Scene Graphs to Teach Hierarchical Modeling, *Computers & Graphics* 2001, number 4.

FOLEY, J. ET AL 1996. *Computer Graphics: Principles and Practice in C*, 2/e, Addison Wesley.

HITCHNER, L. AND SOWIZRAL, H. 1999. Adapting Computer Graphics Curricula to Changes in Graphics, *Proceedings of Graphics and Visualization Education 1999*, July 1999.

OPEN INVENTOR 2002. http://oss.sgi.com/projects/inventor/.

OPENRM 2002. OpenRM Project Overview http://openrm.sourceforge.net/overview.shtml.

OPEN SCENE GRAPH 2002. http://openscenegraph.org.

OPENSG 2002. http://opensg.org.

PERFORMER 2002. *SGI-OpenGL Performer Overview*, http://www.sgi.com/software/performer/.

SOWIZRAL, H., RUSHFORTH, K. AND DEERING, M. 1995. *The Java 3D™ 3D API Specification*, Addison-Wesley.

SUN 2002. *Java 3D API home page*, http://java.sun.com/products/java-media/3D.

WEB3D CONSORTIUM 2002. *X3D home page*, http://www.web3d.org/x3d/.

WERNECKE, J. 1994. *The Inventor Mentor*, Addison-Wesley, 1994

WOLFE, R. 1999. Bringing the Introductory Computer Graphics Course into the 21st Century, *Proceedings of Graphics and Visualization Education 1999*, July 1999.