Scene Graph Creation And Management For Undergraduates

Matthew Jones and Amit Shesh[†][1]

[1]Illinois State University, USA

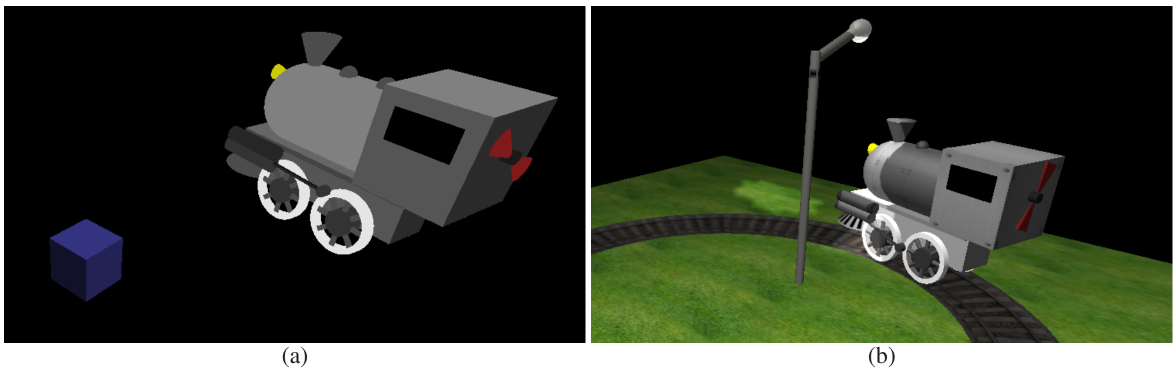

Figure 1: *Results produced by a student. (a) Hierarchical model created using implicit objects using XML input. Each object is given a separate decal color as lighting and texturing were not yet supported. (b) Scene using previous model with lighting and texturing. The model was given a headlight and was animated to move on a circular track (please see accompanying video). This was accomplished by the student (first author) over* 3 *assignments spanning* $6-7$ *weeks.*

Abstract

This paper describes the context and results of a student project related to scene graphs spanning several assignments in an undergraduate computer graphics course. The project progressively built an application that created a list of objects from an XML specification, modified it into a scene graph, implemented part-by-part animation and added point lights and textures. Students were encouraged to build creative models using implicit shapes. It was completed individually by 21 *undergraduate students in three stages spanning* $6-7$ *weeks. This project was further extended in the last two assignments by incorporating a ray tracer within it.*

Categories and Subject Descriptors (according to ACM CCS): K.3.2 [Computer and Information Science Education]: Computer Science Education—

1. Background and Context

The scene graph project is part of the only computer graphics course offered for computer science undergraduates at our university. Students taking this course have prior programming experience in C++, data structures and algorithms, but have no experience in OpenGL programming. Our course does not require taking a prior course in linear algebra. It covers traditional topics recommended in the ACM curriculum [ACM08] such as modeling and transformations, lighting and texture mapping using the OpenGL rendering pipeline in C++, culminating in a ray tracer.

Lectures include a revision of linear algebra (vectors, matrices, coordinate geometry). Transformations are taught using both their mathematical foundations as well as code examples developed in class. Hierarchical transformations and scene graphs are discussed in class, but no code for scene graphs is shown or provided to students. The OpenGL shading model is discussed in detail and illustrated in code examples, as is texture mapping.

2. Scene graph Project: Instructor's perspective

Much work on CG education recommends that scene graphs be a part of traditional CG courses [CB01, Wol00, Bou02].

[†] {mdjones | ashesh}@ilstu.edu

DOI: 10.2312/conf/EG2013/education/011-012

We adopt this recommendation by emphasizing on scene graphs for pipeline-based and raytracing graphics.

This is not motivated as a "project" per-se to students, but can be regarded as one because students progressively build a single application over the course of 3 assignments. To help struggling students keep up, solutions to earlier assignments are provided so that students may use them to start the next assignment. However many students continue using their own code after correcting earlier problems.

The first of these assignments provided students with an XML parser that parses a scene made of implicit objects (planes, spheres, cylinders and cones) and represents them as a list. Students are expected to create two new implicit objects (cone and box) and render a scene containing them. They are also expected to create the model of a locomotive using these implicit objects that must have (1) at least 4 wheels with at least 4 spokes each (2) a body and (3) at least one propeller with at least 2 fan blades.

The second assignment asks students to implement a scene graph (construction and rendering). They modify the XML parser so that instead of compiling a list of objects, it compiles a scene graph. Finally they write "animator" functions that transform specific nodes of the scene graph over time to produce a desired animation. Specifically they write functions to move the above locomotive at a constant speed along a circular track on the X-Z plane.

The third assignment asks students to attach lights and textures (including texture transformations) to scene graph nodes and manage them correctly during rendering. They use an existing class that imports images as OpenGL textures. The XML parser is again suitably modified to support lighting and textures. Using this modified infrastructure, they are asked to enhance their earlier locomotive animation by adding (1) a circular road-textured track (geometry) (2) at least one headlight to their locomotive that moves correctly with it (3) at least 2 stationary light sources in the scene (4) textures on at least one part of the locomotive.

Student creativity is fostered by giving them freedom to decide the exact appearance of the locomotive. In many cases students exceed the minimum requirements to create more complex models. Most students who create complex models in the first assignment manage to correctly animate it in the next assignment without compromising on its structure. They are also encouraged to find and use appropriate images for textures from the web, and cite their source in their code. As the examples in this paper show, the student went above and beyond the requirements of almost all the assignments and created a well-textured animated scene. Please see the supplementary material for the model file created by the student.

In Fall 2012, all 21 students completed the above assignments. The median scores (out of 100) on the assignments were 95, 94 and 81 respectively.

3. Scene graph Project: Student's perspective

The scene graph assignments introduced an interesting application of the tree data structure. Though the first assignment did not use a scene graph, it definitely showed us its value. The concept was simple: given an XML file, read in and draw each of the objects. Although implementing this part was simple, arranging the objects to complete the locomotive was a cumbersome process because every object had to be specifically and uniquely transformed.

The second assignment introduced the scene graph. Though the programming was more challenging, arranging the objects was far simpler. For example, instead of having four wheels each having completely different sets of transformations for its constituent parts, you could make a single wheel group and transform it.

Finally, we added textures and lighting. For lighting, the most complicated part was correctly calculating the normals at each vertex for each object. This, like some of the other parts of the assignment, was easier than it seemed. Calculations for the base objects were simple math, but stretched and other interestingly transformed objects seemed like they needed recalculation. It turned out that each normal was being transformed with its associated vertex automatically, but needed to be re-normalized. Obtaining the texture coordinates for objects created by us was the most challenging part of texture mapping. Thinking of texture mapping as wrapping the object in a blanket (the texture) made the understanding a bit easier.

All of these changes required changes to the XML reader. The main challenge here was to conceptually connect the structure of the parser with the incremental construction of the scene graph. We handled nested groups with a stack. These updates were the cause of a fair number of programming errors, especially with certain operations specific to only leaf nodes.

The completion of this series of assignments taught us the fundamentals of 3D computer graphics. It was particularly effective to have us first create models without the scene graph to truly show how beneficial they are.

References

[ACM08] ACM CS curriculum, 2008. http://www.acm.org/education/curricula/ComputerScience2008.pdf. 1

[Bou02] Bouvier D. J.: From pixels to scene graphs in introductory computer graphics courses. *Computers & Graphics 26*, 4 (2002), 603–608. 1

[CB01] Cunningham S., Bailey M. J.: Lessons from scene graphs: using scene graphs to teach hierarchical modeling. *Computers & Graphics 25*, 4 (2001), 703–711. 1

[Wol00] Wolfe R.: Bringing the introductory computer graphics course into the 21st century. *Computers & Graphics 24*, 1 (2000), 151–155. 1