Tracing Trees
A New Course in Data Structures

Robert Geist

July, 2008

1 τέχνη

As observed in [3], the τέχνη curriculum design relies heavily on a careful selection of large-scale problems from the visual domain. The design employs a *constructivist* view, but, as noted by Duffy and Cunningham [2], this term has come to encompass such a wide span of directions in education that further specification is essential. The view is Piagetian, sometimes termed *cognitive constructivism*, and draws directly from Piaget [8], Dewey [4], and Rousseau [9] in that fundamental tenets are these:

- Learning is an active process of constructing individual knowledge.
- Learning occurs when observations differ from expectations, and new mental models must be constructed to accommodate the differences.
- Teaching is the process of invoking and supporting these constructions.

It is not surprising then that problem-based instruction is a principal vehicle, but the design differs from others in the depth and scope of the problems addressed. Fundamental tenets of the implementation are that a visual problem domain will most quickly capture the attention and interest of students who have grown up in a society that is increasingly visually oriented, that a connection between scientific and artistic components will stimulate both deductive thought and creativity, and that toy problems would be of little value in effecting the principal desired accommodation, an ability to solve real problems. Thus the problems are large-scale, with one problem per semester, and they are selected from problems that have arisen naturally in the research investigations of the faculty.

2 Data Structures

The standard approach to Data Structures is breadth-first, wherein a wide range of structures are considered, hypothetical cases in which each might be useful are proposed, and some elementary algorithm analysis is offered. The breadth of topic, combined with the time constraints of the standard semester course, allows consideration of only elementary examples. The students are well aware that they are solving toy problems in series.

This course, in contrast, is depth-first. Investigations into data structures are driven by real needs for adequate execution time, as experienced by student teams in solving a real problem. Elementary algorithm analysis will be augmented by profiling code prototypes to obtain a priori runtime estimates of proposed solutions. This approach will sacrifice some breadth of coverage, but students will gather a first-hand understanding of the real use and real value of a variety of data structures. Seeking new structures as new problems arise should be a natural outcome for them.

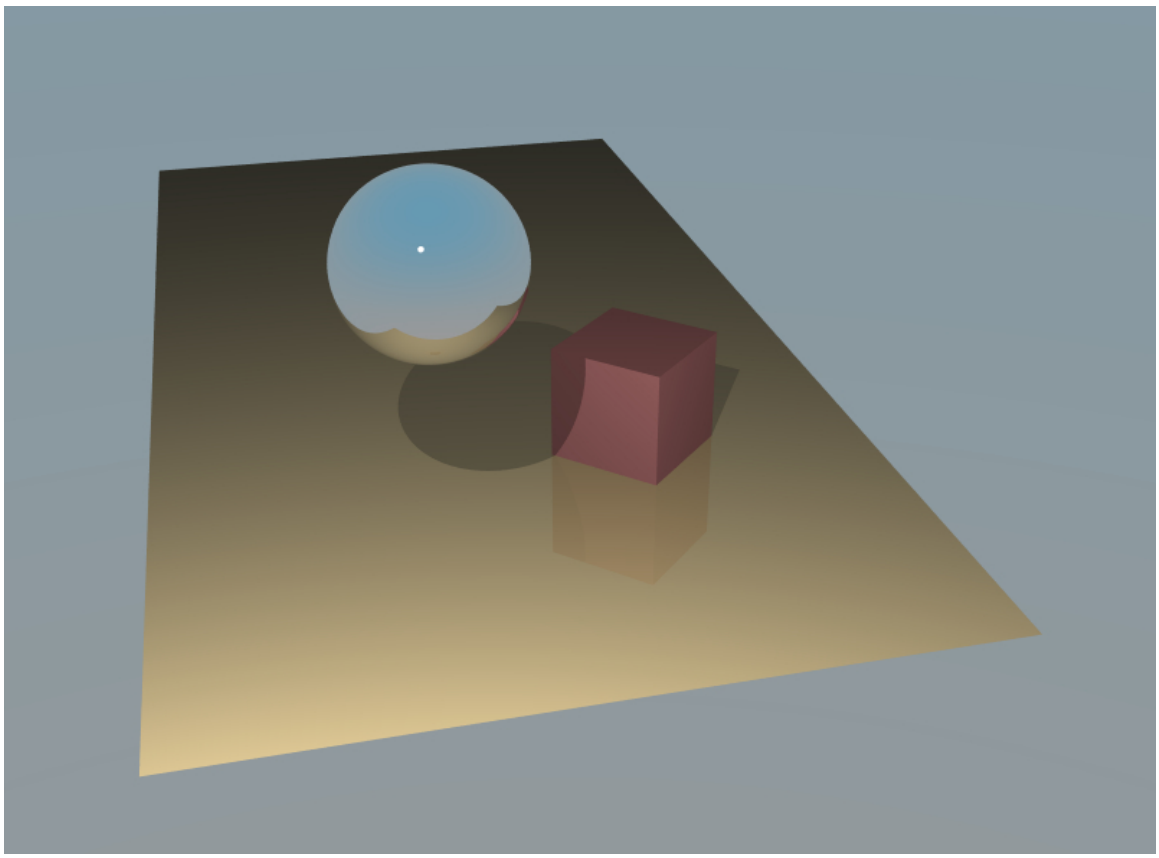


Figure 1: Example 102 scene.

To use a construction analogy, in a classical course students practice hammering nails, practice sawing 2×4s, learn that a nail gun is faster than a hammer, and learn that a SKIL saw is faster than a hand saw, but they haven't a clue as to how to build a house. In this course, the instructor will lead student teams in building houses. The students will learn of the tools and develop the skills to use them within a context of purpose.

3 Student Background

Data Structures (CPSC 212) is the third course for undergraduate majors in the School of Computing at Clemson. It is thus typically taken by first semester sophomores. Because the τέχνη curriculum design was in place last year, students who completed the first two courses have already experienced the difficulties of solving large, semester-long problems, have significant experience with C and some experience with C++.

The semester-long problem of the second course, CPSC 102, is writing a ray tracer to generate synthetic images. The scene of Figure 1 is typical of the output, although orientations that are not axis-aligned, such as the view and the cube placement, may not have been covered by all sections. Stronger students might have added additional light sources, textures (photos mapped to scene geometry), and more interesting, though still elementary, objects.

4 The New Problem

The new problem will build on the ray tracer from the second course. Some students will not have completed this course, and others who have completed it may no longer have copies of their final code. A generic ray tracer, written in C and capable of generating the scene of Figure 1, will be distributed to all students on the first day. The goal will be to modify this ray tracer to generate scenes such as those shown in Figure 2. These scenes, taken from [5], are also synthetic, but, obviously, they are much more complex. Rather than a handful of polygons or surfaces, these scenes each contain more than 250,000,000 triangles.

4.1 Phase 1

The course will begin with a code review of the distributed ray tracer. This will include discussion of C structures, arrays, linked lists, and function pointers. It will also include an orientation to three dimensional geometry, including a discussion of rotation and translation operators that are necessary to move objects (such as trees) into desired positions within an overall scene.

A forest is built from many trees, and each tree is built from many leaves. The first assignment will be an elementary increment: modify the distributed ray tracer to produce an image similar to that of Figure 3. The only significant problem here is handling the leaf geometry. It is implemented as a pair of texture-mapped triangles, arranged as a single rectangle. The texture (photo) contains more than just the color channels (r,g,b) for the leaf. It contains an alpha channel that determines per-pixel visibility within the photo. Thus, if a ray/triangle intersection test determines that a texture-mapped triangle was hit, the appropriate pixel in the photo must be examined to determine whether the intersection was within the leaf boundary. If so, the leaf color from the photo is used. If not, a miss is reported. Finding the appropriate pixel within the photo will require a discussion of interpolation, and handling the 4-channel photo will require a discussion of parsing externally-supplied files.

The transition to C++ will also begin with this phase. The ray tracing problem is a natural one for object-oriented solution. Operator overloading for vector operations on points or colors offers significant savings, and the different geometries (triangles, boxes, spheres, and texture-mapped versions of each) are conveniently handled as derived classes with polymorphic operators.

4.2 Phase 2

In the second phase, the students will be asked to ray trace a single tree, such as shown in Figure 4. Models for several tree species ([7]) will be supplied in the form of .obj files. This phase will occupy most of the semester. Parsing .obj files is more complicated than parsing simple, 4-component images. Once parsing is managed, the real fun begins. The image of Figure 4 contains only two textures, but it contains almost 500,000 triangles. From Phase 1, the students will know how to render linked lists of texture mapped triangles, and so the natural approach will be to build a linked list of triangles and ray trace as before. After this natural approach fails, some simple arithmetic will show the folly of the effort. Anti-aliasing will demand a minimum of about 5 rays per pixel, but most of these rays will generate another, to determine whether the object hit is in a shadow. The image of Figure 4, at 896×672 pixels, will thus require approximately 6,021,120 rays. To find the first visible triangle, each ray must be tested against each triangle, i.e., approxi-



(a) Target scene rendering



(b) Rendering with pine trees rather than beech trees

Figure 2: Sample renderings

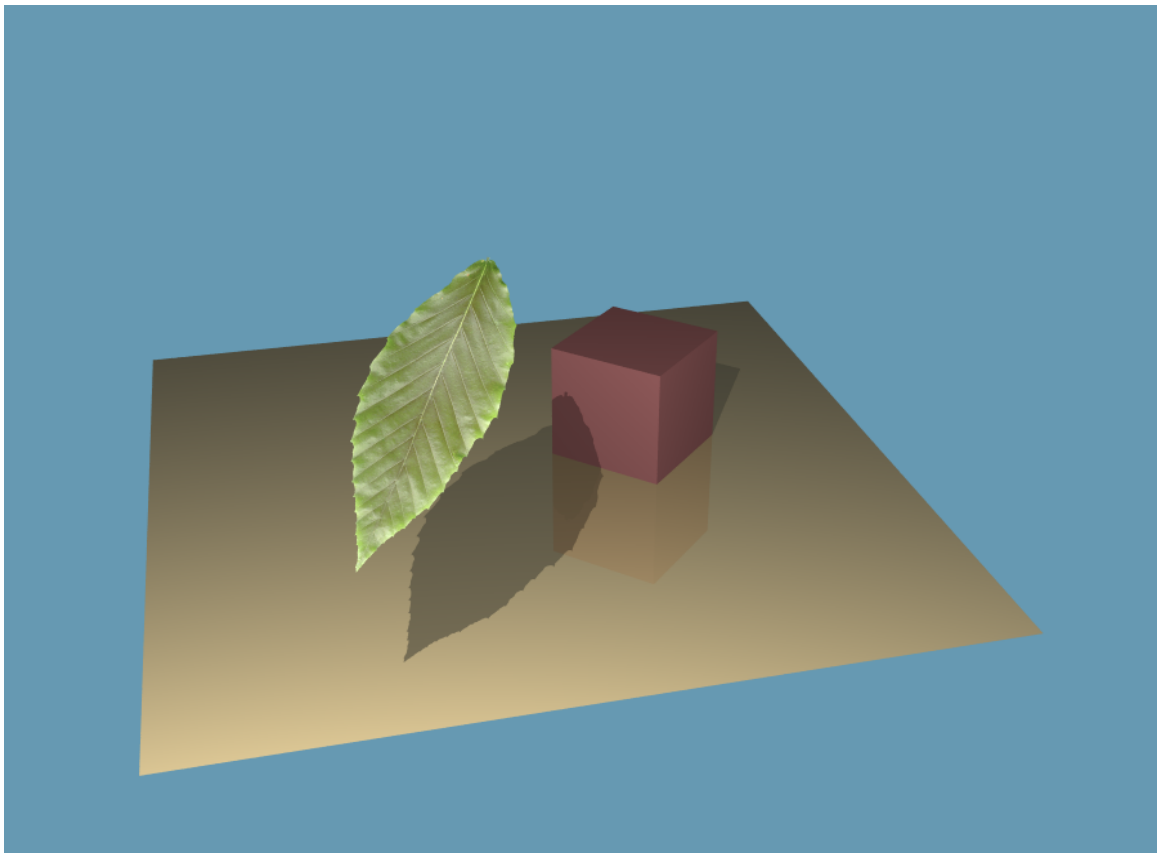


Figure 3: Assignment 1.

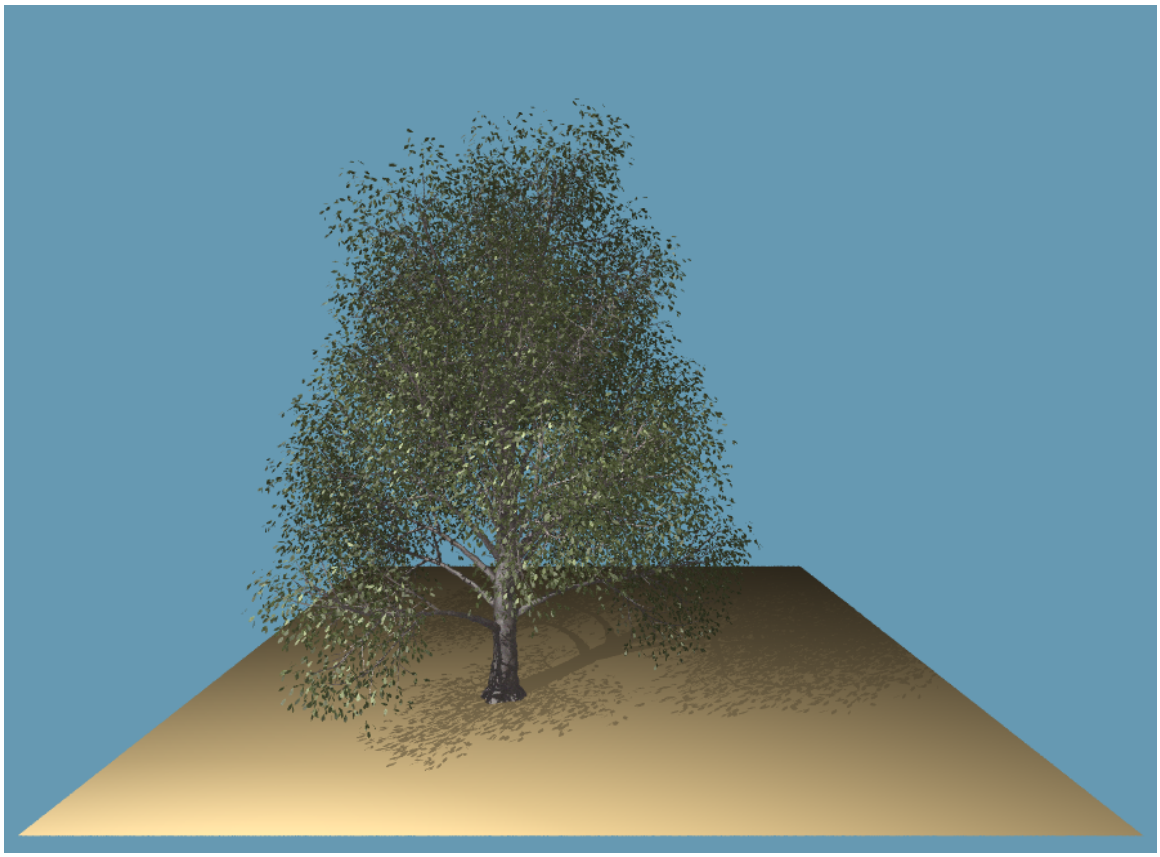


Figure 4: Assignment 2.

mately 3×10^{12} tests. The fastest, most efficient ray/triangle intersection test is the Möller/Trumbore test [6]. By placing CPU cycle counters into the calling code, one can determine an accurate measure of execution time for this test. On a fast, modern CPU, it takes approximately 85 ns. So, if the time for texture lookup and overhead is ignored, this will take at least $3 \times 10^{12} \times 85 \times 10^{-9}$ = 255,000 seconds, i.e., 3 days. Even if the students are willing to accept substandard images obtained by using only one primary ray per pixel, this will still take 14 hours.

At this point, it will be revealed that the images of Figure 2 were ray traced in 1 second. The short answer to the obvious question, "How?", will be, "clever data structures." The long answer will consume much of the semester.

The students will be given at least a full class period to come to the realization that each ray cannot be tested against each triangle, and so the scene geometry must be partitioned to avoid intersection tests. Thus the kd-tree, a balanced, binary tree of axis-aligned splitting planes, will be introduced. Construction of such trees alone will invoke interesting sub-problems, such as finding the median of a very long list of terms. The well-known, linear-time solution, which has the same basis as quicksort, will naturally lead to a discussion of sorting and why sorting is less efficient in this case. Building the tree will also require recursive traversals. Since the correctly constructed kd-tree is complete, a discussion of the advantages of heap representation will be conducted. Traversing the kd-tree to find a reduced collection of triangles to be ray-tested is a delicate operation that requires ray-plane intersection tests and an auxiliary stack to hold pointers to potential candidates, but the search and the need for the stack are easily motivated with a 2D analogy drawn on the classroom white/black board. Execution time that is $O(logN)$ versus that which is $O(N)$ will become truly meaningful.

4.3 Phase 3

Phrase 3 will require ray tracing a miniature forest consisting of at least three species of trees with multiple instances of at least one species. Only one copy of each species' geometry will be allowed. The entire scene will be represented as a kd-tree of bounding boxes, each leaf of which contains a list of one or more bounding boxes, each of which contains either a simple geometry list or another kd-tree representing a physical tree. Finally some discussion of mapping this highly parallel task to the new SIMD architectures (GPU/Cell) will be offered.

5 Assessment

Assessment of the principal, desired accommodation, an ability to solve real problems, is extremely difficult within the classroom setting. Results of assignments that extend over multiple class periods have questionable validity.

The approach taken here will use the 3-hour, final exam period for direct assessment. The first hour of this exam will be a standard, individual exam that is focused specifically on material and concepts covered in the course. At the end of this hour, students will form teams of size 3 and be offered extra credit for up to two hours spent tackling a list of problems taken from the numerous volumes of the ACM-ICPC International Programming Contest [1]. Since this extra credit portion will be optional, participation will speak directly

to student interest and motivation.

Top teams in the contest solve on the order of one problem per hour, so three problems will be the maximum credit available. This portion of the exam will be administered and graded by the External Evaluator, Professor R. Schalkoff.

An indirect assessment will also be carried out. Walker and Fraser [10] observed that numerous studies report a strong correlation between traditional student outcomes (e.g. grades, test scores) and perceptions of classroom environments. The latter can be measured with unobtrusive and time-saving survey instruments. Walker and Fraser used factor analysis on field tests to develop a survey instrument of 34 ratings on six scales, instructor support, student interaction and collaboration, personal relevance, authentic learning, active learning, and student autonomy. Their instrument was ostensibly targeted at distance education, but omitting the six items in the category of student interaction and collaboration, which is consistent with a cognitive constructivist view, yields a factorially valid instrument to measure classroom environments of any type. Thus the resulting 28-item survey will be administered during the final regular class session.

References

[1] ACM. Int. collegiate programming contest european div. http://www.acm.inf.ethz.ch/, 2008.

[2] D. Cunningham and T. Duffy. Constructivism: Implications for the design and delivery of instruction. In D. Jonassen, editor, *Handbook of Research for Educational Communications and Technology*, pages 170 – 198. Simon & Schuster/Macmillan, 1996.

[3] T. Davis, R. Geist, S. Matzko, and J. Westall. τέχνη: Trial phase for the new curriculum. *ACM SIGCSE Bulletin*, 39(1):415 – 419, 2007.

[4] J. Dewey. *Experience and education*. The Macmillan Company, New York, 1938.

[5] R. Geist and J. Steele. A lighting model for fast rendering of forest ecosystems. In *Proc. of the IEEE Symposium on Interactive Ray Tracing (RT08)*, page to appear, Los Angeles, California, August 2008.

[6] Tomas Mller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *journal of graphics tools*, 2(1):21–28, 1997.

[7] Greenworks Organic-Software. Xfrogplants v 2.0. http://www.xfrogdownloads.com/greenwebNew /products/productStart.htm, 2008.

[8] J. Piaget. *The development of thought: equilibration of cognitive structures (translated by A. Rosin)*. Viking Press, New York, 1977.

[9] J.-J. Rousseau. *Émile (translated by B. Foxley)*. Dutton, New York, 1955.

[10] S. Walker and B. Fraser. Development and validation of an instrument for assessing distance education learning environments in higher education. *Learning Environments Research*, 8:289–308, 2005.