τέχνη Trees: A New Course in Data Structures

Andrew T. Duchowski
School of Computing
Clemson University
Clemson, SC 29634-0974
duchowski@clemson.edu

Robert Geist
School of Computing
Clemson University
Clemson, SC 29634-0974
geist@clemson.edu

Robert Schalkoff
Department of Electrical &
Computer Engineering
Clemson University
Clemson, SC 29634-0915
rjschal@clemson.edu

James Westall
School of Computing
Clemson University
Clemson, SC 29634-0974
westall@clemson.edu

ABSTRACT

The τέχνη method is an approach to undergraduate computer science education that is based on cognitive constructivisim, in the sense of Piaget, and which invokes several course design directives that include re-combining art and science, problem-based learning, problem selection from the visual problem domain, and cognitive apprenticeship. The paper describes a new τέχνη course in *data structures*. It includes a full comparative assessment of the realized improvement in student problem solving capability and, for the first time, cognitive authenticity in problem selection, in that the course problem is a variation on a very recent research result.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Curriculum*

General Terms

Design, Experimentation, Human Factors

Keywords

computer graphics, curriculum design, problem-based instruction, ray-tracing, τέχνη

1. INTRODUCTION

The τέχνη project provides an unusual, perhaps radical, alternative to the standard design of the computing curriculum for the bachelor's degree in computer science. As described by Davis et al. [4, 5], it is built on a foundation of *cognitive constructivism* and draws directly from Piaget [13], Dewey [6], and Rousseau [15] in its basic tenets:

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

Nevertheless, this foundation alone is an insufficient specification of method. The τέχνη designers specify pillars, upon which courses should be designed and taught:

- *re-combining art and science* An original motivation for the τέχνη design was (and is) effecting a reconnection of divergent branches in the U.S. educational system, art and science. The word, τέχνη, is the Greek word for *art*. It shares its root with τεχνολογία, the Greek word for *technology*. The lateralization of brain functions need not be enforced by curriculum. Addressing single problems that require linear reasoning, spatial manipulation, and visual processing can offer bi-lateral engagement, with its attendant creativity and originality.
- *problem-based learning* Problem-based learning is well-known [7], and its use is widespread. Carefully designed problems demand that learners acquire self-directed learning strategies, critical knowledge, and the problem-solving proficiency needed for effective progress in deriving solutions. The τέχνη method differs from other problem-based learning methods in the size and scope of the problems addressed. It calls for one, large-scale problem per semester.
- *visual domain* Problems are chosen from the fields of computer graphics and visualization. Human society is increasingly visually-oriented, and visual problems quickly capture the attention and interest of young students. Further, in his discussions on visual communication, Cunningham [3] observed that the tools of thought for effective problem solving, as well the attendant communication skills for group efforts, were extremely well-supported by computer graphics. Cunningham's work was a principal motivating factor in the original τέχνη design.
- *cognitive apprenticeship* Resnick [14] observed that the time-honored master-apprentice relationship could be transferred from the arena of physical skills to that of cognitive skills,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE'11, March 9–12, 2011, Dallas, Texas USA
Copyright 2011 ACM 978-1-4503-0500-6/11/03 ...$10.00.

given an appropriately designed learning environment. Process is the key, rather than any artificial balance in the roles of master and apprentice. There is great value in observation of a master at work. Original solutions to challenging problems, including missteps and erasures, carry a vitality that is missing in solutions that have been cleaned and polished for presentation in more formal settings. Of course, such vitality will only appear if the cognitive demands on the master are authentic. Thus the ideal problems for τέχνη courses are those that challenge both student and instructor.

The entire philosophical basis of the τέχνη method, as well as the results of instructional experiments with introductory courses, have been presented in several papers, e.g. [4, 5, 10], and in the dissertation by Matzko [9].

We present here the results of teaching a new, τέχνη-based course in *data structures* at Clemson University during the Fall semester of 2009. In addition to the new course design, we offer two important extensions of previous efforts. First, we include the results of a full, comparative assessment of the realized improvement in student problem-solving capability. The assessment instruments were designed prior to the beginning of the course(s), and included pre-test and post-test evaluations with linear regression models. Second, for the first time we have invoked full cognitive authenticity in problem selection. The semester-long problem is a variation on graphics research results that were published only one year earlier.

2. DATA STRUCTURES

A Data Structures course is a standard component of most undergraduate curricula in computing. The topic remains a specific requirement for CAC/ABET accreditation in the 2009-2010 cycle [2]. The conventional approach to Data Structures, as typified by the popular book of Weiss [17], 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.

The τέχνη 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 is augmented by profiling code prototypes to obtain a priori runtime estimates of proposed solutions. This approach sacrifices some breadth of coverage, but students 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 become a natural process for them.

To use an analogy from the construction industry, 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 at the conclusion of the course they have not a clue as to how to build a house. In the τέχνη course, the instructor leads student teams in building houses. The students learn of the tools available 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 University. It is a 4-hour course, with 3 lecture/discussion sessions and a single 2-hour lab session each week. It is typically taken by first semester sophomores. At the beginning of the semester, students were randomly assigned to one of two sections. The *conventional* section was taught by author Duchowski using a conventional approach which followed the topics and order of the Weiss [17] textbook. The τέχνη section was taught by author Geist.

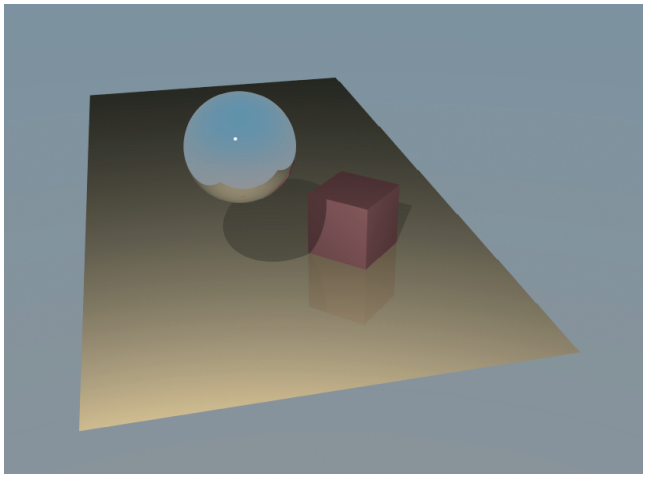


Figure 1: Example 102 scene.

Because the τέχνη curriculum design was in place during the 2008-2009 academic year, students who had completed the first two courses had already experienced (the joy and) the difficulty of solving large, semester-long problems, had significant experience with C, and had some experience with C++.

The semester-long, τέχνη 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 for τέχνη Data Structures builds on the ray tracer from the second course. Of course, some students had not completed this course, and others who had completed it no longer had copies of their final code. A generic ray tracer, written in C and capable of generating the scene of Figure 1, was distributed to all students on the first day. The course goal, also presented on first day, was then to build a C++ ray tracer to generate scenes such as those shown in Figure 2. These scenes, taken from [8], are also synthetic, but, obviously, they are much more complex. Rather than a handful of polygons or surfaces, each of these scenes contains more than 250,000,000 triangles.

The apparent impossibility of the task at hand is then easily described. Simple arithmetic shows 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 by the ray is in a shadow. The images of Figure 2, at 896×672 pixels, will thus each require approximately 6,021,120 rays. To find the first visible triangle, each ray must be tested against each triangle, i.e., approximately 3×10^{12} tests. The fastest, most efficient ray/triangle intersection test is given by the Möller/Trumbore algorithm [11]. By placing CPU cycle counters into the calling code, one can determine an accurate measure of execution time for this

(a) Target scene rendering

(b) Rendering with pine trees rather than beech trees

Figure 2: Sample renderings

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.

It was then revealed to the students that each of the images of Figure 2 was actually ray-traced in less than 1 second. "How is this possible?" was the universal response. The short answer is "Data Structures," and the long answer is the course content.

4.1 Phase 1

The course begins with a code review of the distributed ray tracer, which is available online at http://www.cs.clemson.edu/~geist/212/rt.v0zup.c. This includes discussion of C structures, arrays, linked lists, and function pointers. It also includes an orientation to 3D geometry, including a discussion of dot and cross products and their geometric interpretation, and translation and rotation operators (expressed as triples of dot products) 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 is an elementary increment: modify the distributed C ray tracer to produce a C++ ray tracer that will generate an image similar to that of Figure 3. The distributed ray tracer, though strictly C code, will compile under g++, but assignment requirements included replacement of all function pointers, structs, and unions. As noted earlier, students work in self-selected teams, with a limit of 4 students per team. The only significant problem in this phase 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 requires a discussion of interpolation, and handling the 4-channel photo requires a discussion of parsing large, externally-supplied files of relatively simple format. The portable pixmap format (.ppm) was used for all textures/images. A simple extension to the P6 format allowed 4 bytes per pixel.

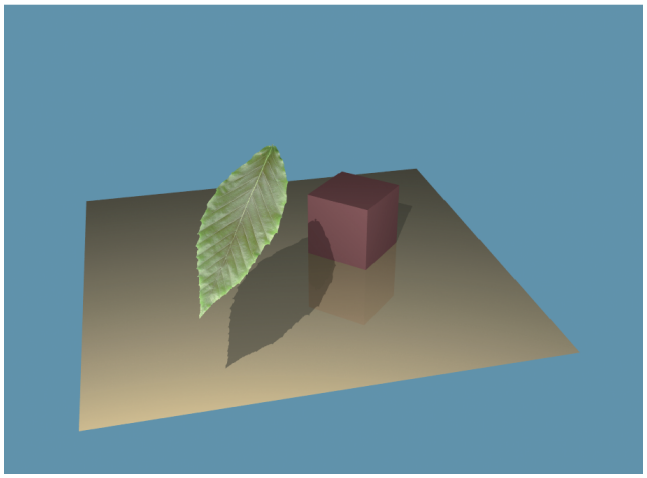


Figure 3: Assignment 1.

The transition to C++, begun with this phase, was generally seen as a relief, rather than a burden. The ray tracing problem is a natural one for object-oriented solution. Operator overloading for vector operations on points and colors offers significant savings, and the different geometries (triangles, boxes, spheres, and texture-mapped versions of each) are conveniently handled as derived classes with with virtual methods and polymorphic operators.

4.2 Phase 2

In the second phase, the students were asked to ray trace a single tree, such as that shown in Figure 4. Models for several tree species ([12]) were supplied in the form of .obj files. This phase occupied most of the semester. Parsing .obj files is far more complicated than parsing simple, 4-component .ppm images. Once parsing, with attendant focus on fast I/O, is mastered, 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 know how to render linked lists of texture mapped triangles, and so the natural inclination is to build a linked list of triangles and ray trace as before.

The students were given 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 [1], a balanced, binary tree of axis-aligned splitting planes, is introduced. Construction of such trees alone invokes interesting sub-problems such as the often repeated task of finding the median of a very long list of items (triangle vertices). The natural approach proposed by students is to sort the list and select the middle item. This leads to a discussion of sorting techniques and their comparative performance on very large tasks. During the development of quicksort, it is observed that the same basic compare-and-swap operation used in quicksort could be used to build a an

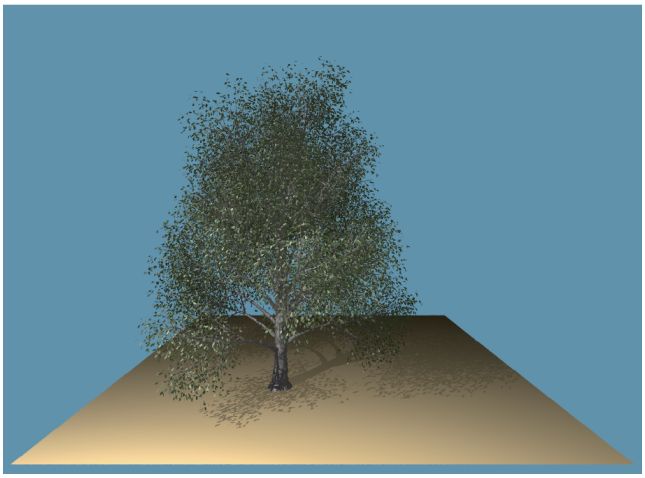


Figure 4: Assignment 2.

$O(N)$ algorithm to find the median! This linear algorithm is then integrated into the kd-tree construction. Although sorting per se is ultimately abandoned for the ray tracing task, the students become quite knowledgeable on the topic.

Building the kd-tree also requires recursive traversals. Since the correctly constructed kd-tree is complete, a discussion of the advantages of heap representation is 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. Pseudo-code for the traversal is show in Figure 5. Execution time

```
(t_min,t_max)=intersect(ray,bounding_box)
stack.push(root,t_min,t_max)
while(not stack.empty):
    (node,t_min,t_max) = stack.pop
    while(not node.is_leaf):
        a = node.splitaxis
        t_split = (node.splitvalue - ray.origin[a])/(ray.dir[a])
        (one,two)=order(ray.dir[a],node.left,node.right)
        if(t_split≥t_max)
            node=one
        else if(t_split≤t_min or t_split<0)
            node=two
        else
            stack.push(two,t_split,t_max)
            node=one
            t_max=t_split
    for triangle in node.triangles:
        t_hit = min(t_hit,intersect(ray,triangle))
    if t_hit < t_max
        break
return t_hit
```

Figure 5: Stack-based kd-tree traversal for ray tracing

that is $O(logN)$ versus that which is $O(N)$ becomes truly meaningful when N reaches the levels found in this task.

4.3 Phase 3

Phrase 3 requires ray tracing a miniature "forest" consisting of multiple species of trees with multiple instances of at least one species. Only one copy of each species' geometry was allowed. The other instances contain only rotation and translation information. The entire scene is 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. Correct handling of normals to surfaces was a delicate matter. Each surface, e.g. a physical tree leaf, had a normal with respect to its defined geometry at the point of ray-surface intersection. This normal had to be combined with the normal to the enclosing bounding box, set during placement of the tree, in order to yield the correct world-coordinate normal.

Figure 6: Example (with permission) from student C. J. Corsi

The students were encouraged to be creative in their designs and reminded that there was no "correct" or "incorrect" final image. A sample student image is shown in Figure 6.

This somewhat singular focus on the structures needed for ray tracing forests did not completely preclude other, more traditional topics from the discussions. Brief forays into AVL trees, red-black trees, hashing, and dynamic programming were also made, after front-loading the structures and methods necessary for the ray tracing task. The lab sessions were used primarily to reinforce understanding of these auxiliary topics and components of the C++ language.

Finally some discussion of mapping highly parallel tasks such as ray tracing to the relatively new SIMD architectures (GPUs) was 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. Instead, we developed ABET-like embedded instruments which were incorporated into the τέχνη course (treatment) and the *conventional* course in both pre-test and final exam. Two analyses were carried out. First, a t-test was used on the per-student change in score from the pre-test to the final exam. The null hypothesis for this test was:

H_0: There is no benefit to the τέχνη method.

Second, a linear regression analysis was conducted using 3 regressors, the pre-test score, the standard constant for such tests (1), and

Table 1: Change from pre-test to final exam

section	mean change	standard dev.	population
τέχνη	0.02	0.26	24
conventional	-0.13	0.28	17

T = 1.7269, α=0.05, reject H_0

a *class* value, which was set to 1 for the τέχνη class and 0 for the *conventional* class. The estimated coefficient of *class*, if significant, is then an indicator of the value of the τέχνη method. Thus, if F denotes the score on embedded final exam questions, P the score on pre-test questions, C the class, and E the error, the linear model was

$$F = \beta_P P + \beta_C C + \beta_K 1 + E$$

All pre-test questions and all final exam embedded questions were multiple choice, to avoid any bias in grading and issues of partial credit. Pre-test questions were largely focused on C code-reading. An example is shown in Figure 7. Embedded final exam

If the following code:

```
#include <stdio.h>

void myfunc(int a)
{
if(a >= 0){
        printf("%d ",a);
        a–;
        myfunc(a);
        }
return;
}

main()
{
int b = 5;
myfunc(b);
printf("\n");
}
```

were compiled and executed, which of these would appear on stdout?

1. no output - infinite recursion
2. compile error
3. 5 4 3 2 1 0
4. 5 4 3 2 1
5. 1 2 3 4 5
6. 0 1 2 3 4 5
7. none of these

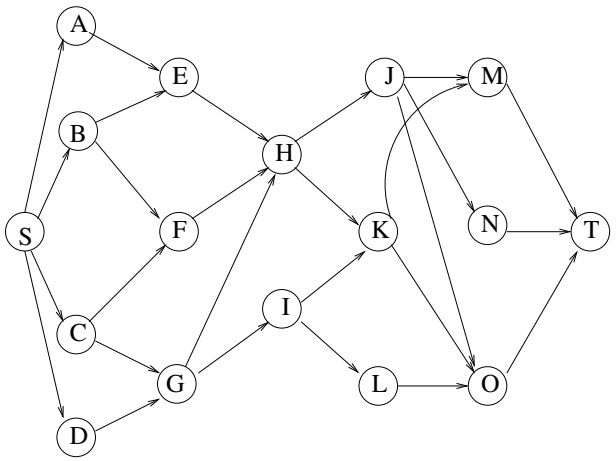
Figure 7: Sample Pre-test question

questions avoided those areas specific to the τέχνη section. In particular, there was no mention of kd-trees, ray-object intersection tests, or parsing .ppm or .obj files. Rather, questions focused on C++ code-reading, algorithm execution time order, and solving small problems. An example question is shown in Figure 8.

In Table 1 we show the results of the t-test on the change from pre-test to final exam. The embedded questions on the final exam(s) were, of course, decidedly more difficult than those on the pre-test, and so the magnitude of the mean change per section is, by itself, not significant. Nevertheless, the difference between sections is significant. It should also be noted that the sections suffered different attrition rates, and so the final populations were no longer the same size.

In Table 2 we show the results of the regression analysis. The coefficient of determination (R^2) was 0.196928. The most important

What is the length of the shortest path from S to T in this network?



1. 10
2. 11
3. 12
4. 13
5. 14
6. 15
7. 16
8. 17
9. 18
10. 19

Figure 8: Sample Post-test question

Table 2: Regression analysis on embedded final exam scores

regressor	estimate	standard dev.	t-ratio	p-value
constant	0.499446	0.106333	4.697	0.000034
pre-test	0.062996	0.143390	0.439	0.6629
class	0.176040	0.059004	2.984	0.0050

coefficient estimate is that for the class regressor, in that it shows the benefit accrued from the τέχνη treatment, which was significant at the 1% level (p=0.0050).

An indirect assessment was also carried out. Walker and Fraser [16] 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 targeted at distance education.

We constructed a survey instrument with 19 questions that were a composite of those suggested at http://oerl.sri.com/. The topics included satisfaction, effectiveness, opportunity to learn, assignments, and engagement. Sample questions and mean scores thereon are shown in Figure 9.

A positive correlation between attitude and post-test (embedded final exam question) results, favoring the τέχνη treatment, was found on all questions except one. Those in the τέχνη class more strongly favored the future use of a different set of software examples. Apparently, the singular focus held inadequate variation.

· I feel my classroom experience in this course generated enthusiasm for the subject.
1) Very little 2) A little 3) Somewhat 4) A lot 5) A great deal
τέχνη: 3.90 *conventional*: 3.00

· I feel my software development experience in this course generated enthusiasm for the subject.
1) Very little 2) A little 3) Somewhat 4) A lot 5) A great deal
τέχνη: 3.14 *conventional*: 2.69

· I feel the software development experience in this class used real-world examples.
1) Very little 2) A little 3) Somewhat 4) A lot 5) A great deal
τέχνη: 3.52 *conventional*: 2.38

Figure 9: Sample survey questions and results

6. CONCLUSIONS

We have presented the design of a new course in *data structures*, a design that is consistent with the goals and foundations of the τέχνη method [4, 5, 9]. The single, semester-long, driving project is construction of a C++ ray tracer that is capable of rendering, in less than a minute, scenes that are built from more than a million triangles. The driving project forces deep, performance-related investigations into many of the areas that are covered by traditional data structures courses, but often covered at a much more superficial level.

In addition to the course design, there are two important contributions of this effort. First, a formal, comparative evaluation between the τέχνη method and the traditional method of teaching data structures was conducted. At the beginning of the Fall semester of 2009, data structures students were each randomly assigned to one of two classes, which were then taught that semester, one with each method. Both pre-tests and post-tests were administered. The post-tests took the form of selected multiple choice questions that were embedded in the final exams of both classes. Analysis included a t-test on the score change from pre-test to post-test and linear regression where the regressors included the pre-test score and a class value (0 = *conventional*, 1 = τέχνη), whose coefficient then estimated the benefit of the τέχνη treatment. Both the t-test and the regression analysis yielded statistically significant results showing that the τέχνη method provided better learning outcomes. An attitudinal survey was also carried out, and it showed, in general, more positive attitudes associated with the τέχνη treatment, although a notable exception was the interest of the τέχνη students in seeing a more varied collection of examples.

The second important contribution was the first achievement of cognitive authenticity in τέχνη problem selection. The chosen task was only one year removed from research results of the instructor, and thus it represented an area of both high interest and significant challenge.

The overall evidence in support of the τέχνη method is compelling. Nevertheless, caveats are in order. We believe that we have provided a fair comparison, but it is difficult, perhaps impossible, to control for instructor bias, motivation, interest, and enthusiasm, even if the same instructor were to teach both classes, and, of course, we used different instructors. Moreover, we must acknowledge that teaching with this method requires substantially more effort than a traditional, textbook-directed approach. If that same additional effort were devoted to any other alternative method, would the result not also be quite positive? Finally, we have observed push-back from members of our own faculty whose research specialties are outside visual computing. They are both uncomfortable with venturing into visual computing and unconvinced that appropriate rewards will be attached to the substantial additional effort that must be expended.

Still, we remain hopeful that the additional benefits to student learning outcomes, such as observed here, will ultimately carry the day.

7. ACKNOWLEDGMENTS

This work was supported in part by the U.S. National Science Foundation under Award 0722313.

8. REFERENCES

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[2] Computing Accreditation Commission. 2009-2010 criteria for accrediting computing programs. http://www.abet.org/Linked Documents-UPDATE/Criteria and PP/C001 09-10 CAC Criteria 12-01-08.pdf, 2008.

[3] S. Cunningham. Graphical problem solving and visual communication in the beginning computer graphics course. *ACM SIGCSE Bulletin*, 34(1):181 – 185, 2002.

[4] T. Davis, R. Geist, S. Matzko, and J. Westall. τέχνη: A first step. *ACM SIGCSE Bulletin*, 36(1):125 – 129, 2004.

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

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

[7] B. Duch, S. Gron, and D. Allen. *The Power of Problem-Based Learning*. Stylus Publishing, LLC, Sterling, VA, 2001.

[8] 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)*, pages 99–106, and back cover, Los Angeles, California, August 2008.

[9] S. Matzko. τέχνη *and Quest-Oriented Learning*. PhD thesis, Clemson University, 2008.

[10] S. Matzko and T. Davis. Using graphics research to teach freshman computer science. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Educators program*, page 9, New York, NY, USA, 2006. ACM.

[11] T. Möller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

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

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

[14] L. B. Resnick. Learning in school and out. *Educational Researcher*, 16(9):13–20, 1987.

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

[16] 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.

[17] M. A. Weiss. *Data Structures and Algorithm Analysis in C++*. Pearson/Addison-Wesley, Boston, MA, USA, 3rd edition, 2006.