Teaching Algorithms and Data Structures through Graphics

Andrew T. Duchowski and Timothy A. Davis

School of Computing, Clemson University, Clemson, SC, USA

Abstract

This paper presents experiences from a first-time implementation of a data structures and algorithms course based on a specific computer graphics problem, namely surface reconstruction from unorganized points, as the teaching medium. The course required sophomore students to implement the algorithm found in Hoppe et al.'s SIGGRAPH '92 paper of the same title. This problem was chosen since the solution lends itself well to an exploration of data structures and code modularization into distinct project phases and milestones, both of which are traditionally taught in early CS courses. While the original course goals were accomplished, our experiences suggest potentials for greater streamlining of these concepts, which are detailed herein.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Graphics Data Structures and Data Types

1. Introduction

In recent years, enrollments in undergraduate computer science programs have experienced decline. As a result, educators in the field have been seeking new ways to attract new students, as well as retain those currently enrolled. One method that has been used in various programs is problem-based learning, which seeks to engage students in learning targeted material through the development of solutions to relevant and interesting problems. We must ask, however, what kinds of problems are relevant and interesting to students?

Our answer is computer graphics. To understand the popularity and proliferation of visual media in our culture, one need only observe the explosive growth of ventures such as YouTube. College students appear to be extremely interested in the visual entertainment industry, which includes computer gaming, film, and television. We have capitalized on this interest at [univ] through a new approach to undergraduate computer science instruction, termed τέχνη (or TEXNH), that uses problem-based learning with graphics.

Problems in computer graphics often lend themselves well to teaching general concepts in computer science for several reasons. First, graphics problems are complex enough to provide a platform for teaching sophisticated topics, while the solutions to these problems can be quickly evaluated for correctness through visual feedback. We can also provide students with a level of artistic freedom, manifested through image content, which is rarely available in such courses otherwise.

The focus of this paper is to describe a new second-year course in the τέχνη curriculum: Algorithms and Data Structures (CS3). In Section 2, we describe the τέχνη project and experiences to date. Section 3 covers related work in problem-based learning with graphics, while Sections 4 and 5 provide details for the organization of the course. Section 6 discusses specific suggestions for future offerings before we conclude in Section 7.

2. τέχνη project overview

The name τέχνη is the Greek word for art and shares its root with τεχνολογία, the Greek word for technology. As such, the term reveals the close academic relationship the two fields have held historically. A major goal of the τέχνη project is to reunite these areas for more effective means of teaching.

The inspiration for τέχνη originated from our experiences in the establishment of a new cross-disciplinary digital production arts (DPA) program. This master's level degree combines elements of computer science, art, theater, and psychology, among others. Graduates who have completed the program pursue careers in the special effects industry for

© The Eurographics Association 2007.

film, television, and gaming. Studios that have hired our students include Rhythm & Hues, Industrial Light & Magic, Pixar, Blue Sky, Electronic Arts, and Sony Imageworks.

The primary goal of the τέχνη project is to incorporate graphics projects and research from DPA and computer science in the undergraduate computer science curriculum. This material takes the form of semester-long projects in required courses leading to a B.A. in computer science. We believe this approach to be an effective pedagogical method in teaching general computer science concepts since it naturally encompasses several education-theoretical techniques, including: visual feedback, problem-based learning [DGA01], intentional learning [Mar97], constructivism [BA98], and problem-based learning [Cun02]. Through this approach, our goal is to improve understanding of key computer science concepts, while engaging students through projects focused on a field of current interest. This approach provides opportunities for students to explore new topics as they naturally arise in large-scale graphics projects. Accordingly, the instruction is problem-based, with projects ranging from traditional graphics problems, such as ray tracing, to cutting-edge implementations from current research, as discussed later in this paper.

The first course in the τέχνη curriculum is CS1 (first-year course introducing computer science), in which we use a semester-long project in image processing as the motivating problem to teach required basics. The course culminates in a final project involving the implementation of an image recoloring algorithm described by Matzko and Davis [MD06]. All of these experiences provide a the basis for the problem studied in the second course in the curriculum, CS2.

For CS2, students are required to implement a ray tracer, a task previously reserved for our graduate-level advanced graphics course. The ray tracer project provides an ideal pedagogical platform for problem-based learning for several reasons: it naturally covers a broad range of computer science concepts; it provides visual feedback at all stages, allowing program correctness to be determined immediately; and it naturally leads to discussion and implementation of an object-oriented paradigm. The course has been offered in various forms several times with excellent results, in terms of student engagement and images produced. Additional details are given by Davis et al. [DGMW04].

The course discussed in this paper is the third course in the curriculum.

3. Related work

By their second year in college, students typically have been exposed to computer graphics in numerous formats. Many recent educational approaches in this area have attempted to teach various subject topics using image processing. Several efforts have focused on allowing students to explore image processing topics, due primarily to the interesting nature of such projects and the computer science concepts they reinforce. Such projects span a wide range of students, from elementary school [MV05] to college [WN05, AR98, Bur03, FP97, Hun03] and prove to be engaging as students enjoy seeing visual results and solving real-world problems.

As tools for teaching introductory computer science concepts, image processing and rendering projects lend themselves well to encouraging students to learn two-dimensional arrays and dynamic memory allocation [Bur03]. Additionally, since a single image may contain hundreds of thousands of pixels, students cannot rely on hard-coded solutions and are thus forced to write generalized algorithms [MV05]. Additionally, problem-based learning further emphasizes the necessity of complex programming techniques, and using computer graphics provides an effective problem-based for teaching general computer science concepts [Cun02].

Previous work using techniques in rendering or image processing has been performed, but on a limited scale only. Past projects include those with ready-made GUI environments [AR98], code for function definitions [WN05], and pre-written functions [AR98, Bur03, FP97, Hun03] for various image manipulations.

Our approach for CS3 under τέχνη is unique in at least two ways. First, the problem is semester-long, and designed to enforce all topics in the course. Second, the level of difficulty for this project far exceeds any of the problems we have seen in the literature.

4. Course Content and Project Description

The course discussed in this paper is entitled CPSC 212, Algorithms and Data Structures, otherwise known as CS3. It focuses on abstract data types, measures of program running time and time complexity, and algorithm analysis and design techniques. Due to the nature of the course, it also introduces object-oriented design and implementation (using C++).

The semester-long project in the course involved surface reconstruction from unorganized points, as proposed by Hoppe et al. [HDD*92]. This algorithm was chosen as a motivating problem for the course due to its robust use of data structures and algorithms. Moreover, the algorithms employed showcase the need for their efficient design and implementation; otherwise, processing the large data set would require significantly long periods of computation.

In its simplest form, the goal of the project is to find a surface that spans a given set of points, such as images produced by 3D scanners (e.g., laser range finders) [Hop94]. The problem can be expressed formally as follows: given a list of 3D points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^3$, generate a list of triangles $\{\triangle_1, \ldots, \triangle_t\}$ representing the simplical surface approximating the point-sampled object.

The problem is conquered by dividing the approach into four successive stages:

© The Eurographics Association 2007.

1. Phase I: Tangent Plane Estimation
   For each data point $\mathbf{x}_i$, obtain a tangent plane $Tp(\mathbf{x}_i)$ represented by the plane's origin $\mathbf{o}_i$ and unit normal $\hat{\mathbf{n}}_i$. These are calculated via computation of the spatial mean and Principal Components Analysis of a set of k points of X nearest to $\mathbf{x}_i$, denoted as the point's k-neighborhood $Nbhd(\mathbf{x}_i)$.
2. Phase II: Consistent Tangent Plane Orientation
   Given the set of tangent planes $\{Tp(\mathbf{x}_1),\ldots,Tp(\mathbf{x}_n)\}$ from above, ensure that tangent planes "sufficiently close" to each other are "consistently oriented." For a pair of sufficiently close tangent planes, represented by $Tp(\mathbf{x}_i) = (\mathbf{o}_i,\hat{\mathbf{n}}_i)$ and $Tp(\mathbf{x}_j) = (\mathbf{o}_j,\hat{\mathbf{n}}_j)$, they are consistently oriented if $\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j \approx \pm 1$, otherwise either $\hat{\mathbf{n}}_i$ or $\hat{\mathbf{n}}_j$ is flipped.
3. Phase III: Signed Distance Function
   Once the tangent planes are consistently oriented, a signed distance function $f(\mathbf{p})$ is computed at each point $\mathbf{p}$ situated at the vertex of a cube in a 3D lattice. The signed distance function is defined as the distance between point $\mathbf{p}$ and its projection $\mathbf{z}$ onto $Tp(\mathbf{x}_i)$, i.e., $f(\mathbf{p}) = (\mathbf{p}-\mathbf{o}_i)\cdot\hat{\mathbf{n}}_i$.
4. Phase IV: Contour Tracing
   The last stage of the algorithm maps each cube's permutation of the signed distance function at the cube's 8 vertices onto a particular triangle configuration, i.e., the Marching Cubes algorithm [LC87]. The result is a list of triangles approximating the object's (simplical) surface.

5. Implementation: Data Structures and Algorithms

Each of the four phases of the algorithm relies on the clever application of traditional data structures and algorithms. Considering the above algorithm as the logical "interface" to the problem, the algorithm's phases are again described below in terms of the solution's "implementation."

5.1. Phase I: Tangent Plane Estimation

Tangent plane estimation relies on efficient organization of the input data set, or point "cloud" to facilitate collection of the $\mathbf{x}_i$ point's k-neighborhood $Nbhd(\mathbf{x}_i)$ (we arbitrarily set $k = 5$). To do so, a kd-tree is constructed to allow efficient nearest-neighbor (nn) and kth-nearest neighbor (k-nn) queries. To construct the 3D kd-tree, we first introduced the C++ `class` as a mechanism for representation of an Abstract Data Type (ADT; the `Point` object in this case). Initially, only 2D points were discussed.

Although the kd-tree is considered to be an advanced data structure used for spatial partitioning, it serves as a suitable platform for discussion of more rudimentary algorithms on which it is based, namely sorting. Sorting, in turn, motivates the general comparison of asymptotic performance and hence algorithm analysis, which are topics covered in the traditional version of the course.

Given a point's k-neighborhood $Nbhd(\mathbf{x}_i)$, calculation of the tangent plane origin (centroid) and normal requires yet another fairly simple algorithm and quintessential ADT, Principal Components Analysis (PCA) and the representation of a matrix. It is worth noting that at this point in their experience, students are likely to have at least heard of eigenvalues and eigenvectors, but may not be quite comfortable with their practical applications.

Results of the first phase are shown in Figure 1. Two data sets were used: *conics* and *mechpart* containing 15887 and 4102 points, respectively. The *conics* object served as a good example of a fairly voluminous data set, one demanding algorithmic efficiency. The *mechpart* object was one originally used by Hoppe et al. in their paper and provided good comparison to the project's progress.

5.2. Phase II: Consistent Tangent Plane Orientation

The second phase of the algorithm serves to introduce graph algorithms, most notably the (Euclidean) Minimum Spanning Tree (MST) and its traversal. Here, the featured ADT is the binary (or Fibonacci) heap, acting as a priority queue (or minheap). Since Prim's MST algorithm relies on efficient implementation of the minheap, algorithm analysis (and NP-completeness) is discussed again at this stage.

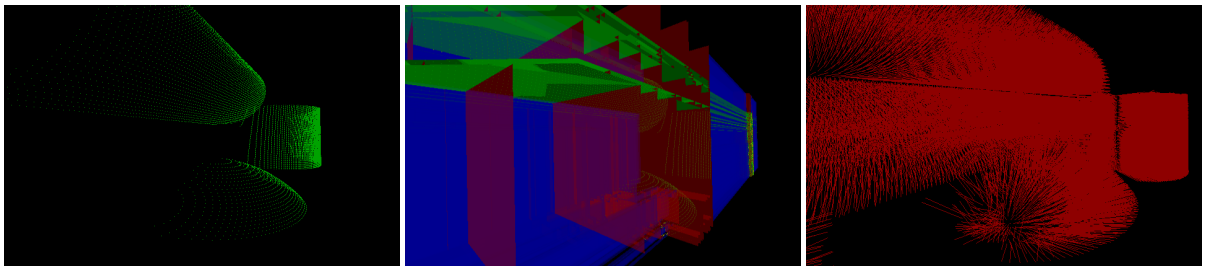
Results from phase II are shown in Figure 2. Note that this section of the course explored several options during implementation: two forms of heap implementations (binary and Fibonacci), as well as differing construction of the MST: Euclidean MST or Reimannian graph. Furthermore, the *conics* object posed an unexpected problem, namely it consisted of several connected components (see Figure 2b), which the original SIGGRAPH '92 never considered (justly so since it operated under the assumption of manifold surfaces).

5.3. Phase III: Signed Distance Function

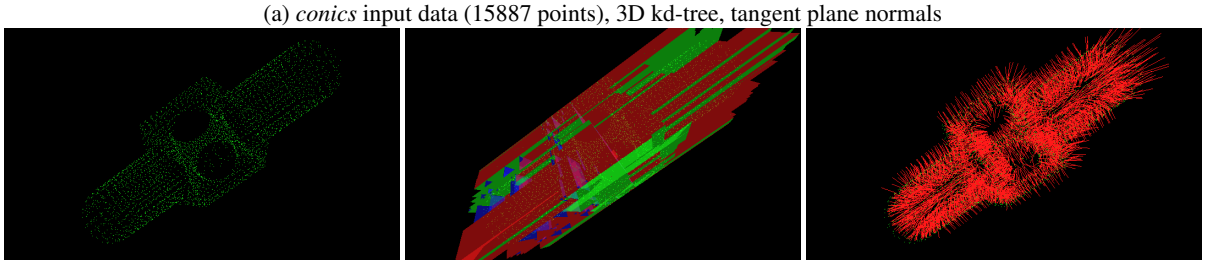
Calculation of the signed distance function once again relies on the collection of the nearest neighbor, although this time the neighbors of the tangent plane $Tp(\mathbf{x}_i)$ are sought (the same point-based kd-tree as before can be used if each point maintains backpointers to the tangent plane originally defined on it). Although this phase of the algorithm lacks a particularly foundational aspect, it serves as a way to introduce some of C++'s perhaps more esoteric members of the Standard Template Library (STL). In particular, `bitset<3>` is introduced to represent the cube's 8 vertex indices, e.g., $\mathbf{p}_i[k] = \mathtt{b}[k]\ ?\ \mathbf{c}_i + \Delta/2\ :\ \mathbf{c}_i - \Delta/2$ where $\mathbf{c}_i$ is the cube centroid and $\mathtt{b}[k]$ is the bitset defined for $k = 0,1,2$, the x, y, and z coordinates of the cube's vertex ($\mathbf{p}_i[k]$) with Δ denoting the cube width.

Another `bitset<8>` is used to represent the bit code for the signed distance function evaluated at each cube vertex, i.e., $\mathtt{fb}[k] = f(\mathbf{p}_i) \geq 0\ ?\ 1\ :\ 0$. A cube is considered valid if there are more than 0 but less than 8 bits set in the $\mathtt{fb}[k]$ bitset, i.e., if `(0 < fb.count()) && (fb.count() < 8)`.

© The Eurographics Association 2007.

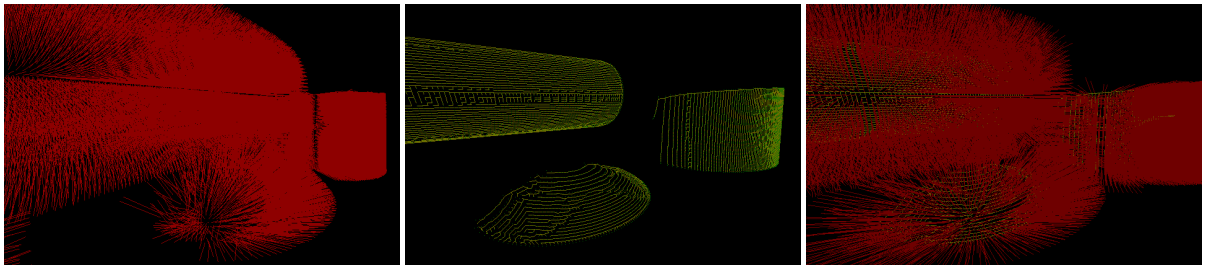


(a) *conics* input data (15887 points), 3D kd-tree, tangent plane normals

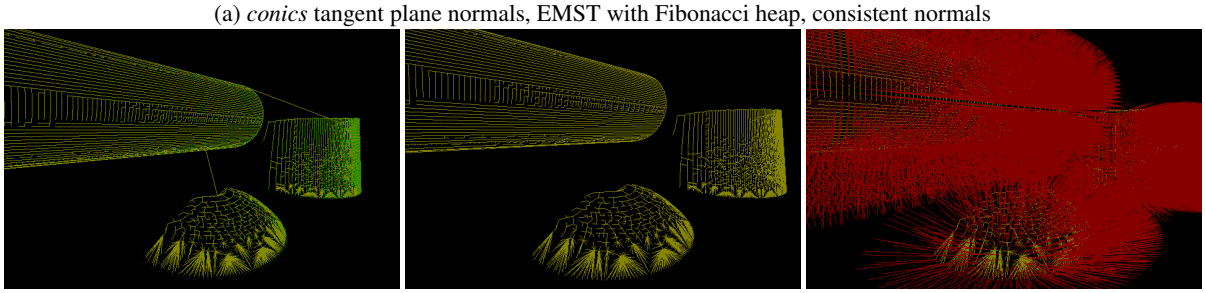


(b) *mechpart* input data (4102 points), 3D kd-tree, tangent plane normals

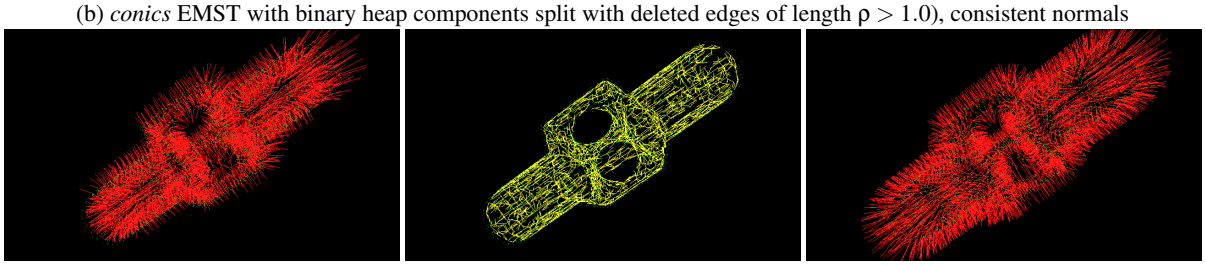
Figure 1: *Phase I: tangent plane estimation.*



(a) *conics* tangent plane normals, EMST with Fibonacci heap, consistent normals



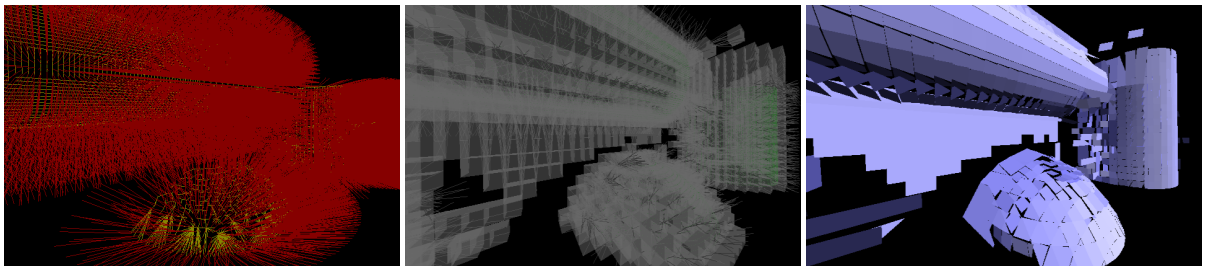
(b) *conics* EMST with binary heap components split with deleted edges of length $\rho > 1.0$), consistent normals



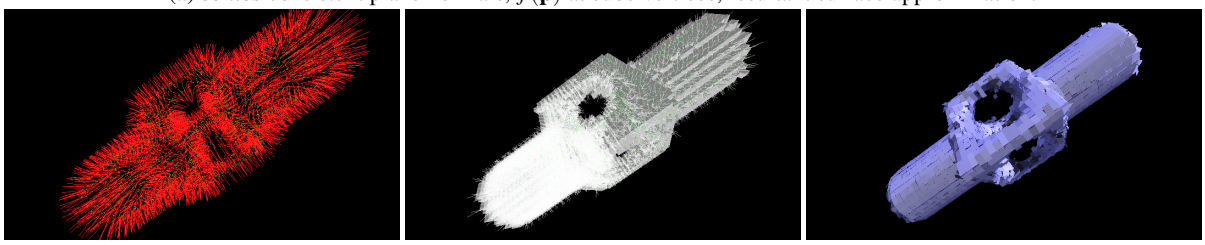
(c) *mechpart* tangent plane normals, Reimannian graph, consistent normals

Figure 2: *Phase II: consistent tangent plane orientation.*

© The Eurographics Association 2007.



(a) *conics* consistent plane normals, $f(\mathbf{p})$ at cube vertices, resultant surface approximation.



(a) *mechpart* consistent plane normals, $f(\mathbf{p})$ at cube vertices, resultant surface approximation.

Figure 3: *Phase III & IV: signed distance function & contour tracing.*

5.4. Phase IV: Contour Tracing

In the final phase of the algorithm, the isosurface is extracted with the help of a lookup table. The STL `pair<int,int>` is used to define an edge 3-tuple where each of the three pairs denotes two cube vertex indices (or edges). For each of the three edges so defined, the point of intersection along each edge is calculated parametrically. The normal for each resultant triangle is obtained before the triangle is added to the list of triangles defining the simplical surface approximation. At this point, the triangulated surface for the point cloud has been created.

6. Formative Impressions

The surface reconstruction problem provides a fairly logical and robust progression through a typical data structures and algorithms textbook (we used Weiss' 3rd edition [Wei06]), and in this sense, the course was successful in teaching the required material. At the same time, the amount of material that is relevant to the problem's solution can be overwhelming. Indeed, this was probably the greatest source of frustration – which algorithms and data structures to select and which to leave out. In any case, the problem must always be a means for learning the target concepts, and *not* the end goal itself.

Although no quantitative analysis was performed during this first-run course, we offer qualitative "lessons learned" from experiences gathered from its inauguration. Following the structure of the problem in the previous sections, notes and suggestions are offered for a more streamlined approach to future instantiations of the class.

6.1. Phase I: Tangent Plane Estimation

Phase I was the most involved and time-consuming segment of the course, consuming 8 of the 16 weeks for students to complete. Consequently, the remaining three phases were inappropriately compressed. In future instantiations, a more balanced approach would be beneficial.

Reasons for the extended duration of the first phase are numerous. This phase was split into four milestones:

1. 2D point C++ class (with centroid calculation)
2. 2D matrix C++ class (with PCA implementation)
3. 2D kd-tree C++ class (with nn and k-nn queries)
4. 3D extensions

Emphasis was placed on 2D implementations for ease of visualization and discussion, e.g., it is much easier to manually draw concepts of point cloud centroid, eigenvectors, and kd-tree. It was not clear, however, that students appreciated these explanations of the underlying mathematical concepts. Possibilities for future offerings of the course are either simply omitting the 2D representations in the interest of time compression, or reformulating the entire problem in 2D.

Beyond the 2D/3D distinction, a substantial amount of time was spent on several coding "niceties" and conveniences, which may only be convenient to experienced programmers. Two examples stand out: computation of the "running mean" and implementation of the 2D matrix class using templates. Both could have been replaced with simpler solutions that avoid unnecessary complexity.

Because the first phase of the semester-long project involved sorting (a requirement for kd-tree implementation), several lectures were devoted to discussion of algorithm

© The Eurographics Association 2007.

analysis and sorting. While these lectures are definitely integral to the material presented in the course, it may be more conducive to learning if this discussion is left to a time period following completion of the first phase, when students can be more focused on theory.

Additional time could be saved if teaching of the Principal Components Analysis were avoided. Students were eventually given code to compute and sort the eigenvalues (Jacobi matrix rotation from Numerical Recipes [PTVF92]). Actually, this portion of the project provides an opportunity for gaining experience in importing "foreign" code into one's own project.

6.2. Phase II: Consistent Tangent Plane Orientation

Challenges in phase II were rooted in the selection of appropriate data structures for generating a minimum spanning tree. Prim's algorithm was chosen over Kruskal's, requiring a choice of appropriate data structure for a heap to provide functionality of a priority queue (instead of one supporting the union/find algorithm required by Kruskal's). Implementation of the heap was problematic due to several potential choices (a "list" heap based on C++'s `list` container, the Fibonacci heap, and the binary heap). A better approach would be to concentrate on one choice (i.e., the binary heap) without exploring alternatives. Actual implementation of the binary heap was further complicated by additional requirements.

Additionally, construction of the graph for MST creation could be performed in one of two ways, by either implementing the complete graph (for Euclidean MST) or the Reimannian graph. The *mechpart* object, for example, appeared to be better processed with the use of the Reimannian graph (where edges are added to the graph only if they are in each other's nearest neighborhoods, i.e., they are "sufficiently close"—interpreted as edge distance being smaller than the radius of the k-nn neighborhood). Implementation of the EMST, on the other hand, is considerably simpler due to the creation of a complete graph (all edges added without a need to test for their inclusion).

In summary, problems encountered during this phase of the project stemmed from an excessive number of potential choices for implementation. Specifically, choices were available for MST algorithm, heap implementation, and graph creation. Interestingly, the two data sets chosen appeared to require two differing graph constructions (complete graph for *conics*, Reimannian graph for *mechpart*). While this may provide additional class discussion possibilities, it prevented "rapid prototyping" of code on a smaller data set.

6.3. Phase III: Signed Distance Function

Evaluation of the signed distance function required traversal along a uniform 3D grid within the point cloud's bounding box. At each grid cell (voxel), eight cube vertices were calculated as mentioned above, and the signed distance function $f(\mathbf{p})$ is evaluated for the tangent plane closest to the cube vertex.

There is nothing particularly difficult about this phase of the project beyond the three nested `for` loops required to cover the volume of space occupied by the point cloud. At this stage of implementation, one can simply cover interesting or useful code structures such as the C++ STL's `bitset`$< k >$ as discussed.

In practice, however, at this point in the semester (about the tenth week, with another six remaining) students were already inundated with material. Consequently, instruction of the class migrated to code dissection, as well as movement into the lab where the form of pedagogy transformed from lecture to hands-on help.

6.4. Phase IV: Contour Tracing

The final phase of the project required execution of the Marching Cubes algorithm with assembly of triangles from plane/cube intersections as suggested by the signed distance function. At this stage of the course, however, many students had fallen behind, disallowing the deeper concepts expressed by the algorithm to be discussed. Some of the more persistent students were able to accomplish this final step, with additional help.

7. Conclusion and Recommendations

This first attempt at teaching data structures and algorithms from a graphics point of view had its moments of success, but also several problems; however, the fault lies in the execution of the approach and not its intent. The given problem is suitable for problem-based learning, particularly because it touches upon such a large number of rudimentary techniques. A major point to remember is to stay focused on a set of approaches that teach the target material through the solution of the problem at hand.

Below are point-form recommendations for restructuring the class to streamline the related topics.

1. Phase I: Tangent Plane Estimation
   - C++ intro: a `Point` class to represent the point cloud
   - C++ `matrix` class (no templates)
   - Principal Components Analysis: code integration (provide code to calculate the PCA)
   - kd-tree construction and nn and k-nn queries
2. Theoretical Interlude
   - Algorithm analysis
   - Sorting comparisons
3. Phase II: Consistent Tangent Plane Orientation
   - Binary heap

© The Eurographics Association 2007.

- Prim's algorithm
- Graph ADT
- EMST construction and traversal

4. Phase III: Signed Distance Function
   - C++ bitset
5. Phase IV: Contour Tracing
   - Triangle ADT

The above outline for the course is still "front-heavy" meaning that the bulk of time is likely to be spent in the first two phases. However, deciding on specific approaches (e.g., binary heap, Prim's, complete graph) without digressing to consider alternative (e.g., Fibonacci heap, Kruskal's, Reimannian graph) should streamline the course and limit the potential for "information overload."

A key issue that needs to be determined is whether any time should be spent on 2D variants of the implementation (e.g., 2D kd-tree). Instruction is easier in two dimensions but it is somewhat time-consuming and may divert the student's attention from the given, inherently three-dimensional, problem at hand. A potentially interesting alternative may be to re-work the original problem of surface reconstruction into its two-dimensional equivalent (curve reconstruction). This approach may lead to a more interactive pedagogical style if students could experiment with a real-time application wherein they could click to create their own 2D data sets and run their programs to generate the curves.

In informal discussion with the students, one remarked that even though the project may have been too demanding, she said that everyone in the class knows data structures and C++ programming extremely well. Further, students suggested that they would be more engaged by performing laser scans on objects they would like to work with, such as a character's face (or even their own!).

References

[AR98] ASTRACHAN O., RODGER S. H.: Animation, visualization, and interaction in CS 1 assignments. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 1998), ACM Press, pp. 317–321.

[BA98] BEN-ARI M.: Constructivism in computer science education. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 1998), ACM Press, pp. 257–261.

[Bur03] BURGER K. R.: Teaching two-dimensional array concepts in Java with image processing examples. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 2003), ACM Press, pp. 205–209.

[Cun02] CUNNINGHAM S.: Graphical problem solving and visual communication in the beginning computer graphics course. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 2002), ACM Press, pp. 181–185.

[DGA01] DUCH B., GRON S., ALLEN D.: *The power of problem-based learning*. Stylus Publishing, LLC, Sterling, VA, 2001.

[DGMW04] DAVIS T., GEIST R., MATZKO S., WESTALL J.: τέχνη: a first step. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 2004), ACM Press, pp. 125–129.

[FP97] FELL H. J., PROULX V. K.: Exploring Martian planetary images: C++ exercises for CS1. In *SIGCSE '97: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 1997), ACM Press, pp. 30–34.

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surfrace Reconstruction from Unorganized Points. In *Computer Graphics (SIGGRAPH '92)* (New York, NY, 1992), ACM, pp. 71–78.

[Hop94] HOPPE H.: *Surface Reconstruction from Unorganized Points*. PhD thesis, University of Washington, Seattle, wA, 1994.

[Hun03] HUNT K.: Using image processing to teach CS1 and CS2. *SIGCSE Bull. 35*, 4 (2003), 86–89.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 163–169.

[Mar97] MARTINEZ M.: Designing intentional learning environments. In *SIGDOC '97: Proceedings of the 15th annual international conference on Computer Documentation* (New York, NY, USA, 1997), ACM Press, pp. 173–180.

[MD06] MATZKO S., DAVIS T.: Using graphics research to teach freshman computer science. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Educators program* (New York, NY, USA, 2006), ACM Press, p. 9.

[MV05] MCANDREW A., VENABLES A.: A "secondary" look at digital image processing. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 2005), ACM Press, pp. 337–341.

[PTVF92] PRESS W. H., TEUKOLSKY S. A., VETTERLING W. T., FLANNERY B. P.: *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, Cambridge, 1992.

[Wei06] WEISS M. A.: *Data Structures and Algorithm*

© The Eurographics Association 2007.

Analysis in C++, 3rd ed. Pearson Education (Addison-Wesley), Boston, MA, 2006.

[WN05] WICENTOWSKI R., NEWHALL T.: Using image processing projects to teach cs1 topics. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer Science education* (New York, NY, USA, 2005), ACM Press, pp. 287–291.

© The Eurographics Association 2007.