Computing with Geometry as an Undergraduate Course: A Three-Year Experience*

John L. Lowther and Ching-Kuang Shene†
Department of Computer Science
Michigan Technological University
Houghton, MI 49931–1295
{john,shene}@mtu.edu

1 Motivation

Computing with geometry is a rapidly evolving interdisciplinary field involving computer science, engineering and mathematics. It has relationships to many other areas within computer science (*e.g.*, computational geometry, graphics, information/scientific visualization and computer vision) and serves as a vehicle for engineering students to approach product design and manufacturing processes. Moreover, this is a *geometric* world! Unfortunately, in a typical computer science curriculum, computing with geometry is virtually missing in spite of its impact on computer science and other fields, and its importance to increase students' employability. Furthermore, many educators still believe computing with geometry, especially curves, surfaces and solids, belongs to engineering and is not part of computer science curricula, despite the need in graphics and computer-aided design for software engineers and programmers.

A course about computing with geometry is important to computer science students and should have a place in computer science curricula. Due to the fact that computer science emphasizes discrete topics and that non-discrete topics have been gradually shifted to other disciplines, the Computer Science and Telecommunications Board and National Research Council suggested adding continuous mathematics back into computer science curricula [2]. Moreover, computing with geometry is the foundation of new manufacturing technologies which will help this country maintain competitiveness in the global economy [1, 4]. Computer science curricula do have some basic elements of geometric computing scattered throughout many courses. It would be better to organize these elements in a coherent way.

This paper suggests a possible remedy by designing a comprehensive, intermediate level interdisciplinary computing with geometry course for students in computer science, engineering and mathematics. Under the support of NSF, we have designed for this course an electronic book, a number of lab manuals, and software tools. The benefit of this course to engineering students seems obvious, since they are using computer-aided design systems daily. This course could provide a place for mathematics students to use their geometric knowledge and learn "algorithmic" and "computational" aspects that would enhance their understanding.

We will present course design merit in Section 2. Details can be found in [3]. Section 3 and Section 4 cover the course content and software tools. Course evaluation and dissemination are presented in Section 5 and Section 6. Finally, Section 7 has our conclusion.

2 Design Merit

Our course addresses the fundamentals of computing with geometry using an intuitive, elementary, and learning-by-doing approach by emphasizing the geometric nature of curves, surfaces and solids and leaving the complexity of algebraic derivations and computations to software tools, which are designed to help students visualize difficult concepts and algorithms. With the help of these fundamental concepts, students can easily learn the necessary derivations in later courses. Even though students may not take follow up courses, they will have acquired fundamental knowledge which will be useful when facing geometric problems.

*This work was partially supported by the National Science Foundation under grant DUE-9653244. The second author was also partially supported by the National Science Foundation under grant CCR-9696084 (formerly CCR-9410707) and by a grant from the Michigan Research Excellence Fund 1998–1999.

†Corresponding author

The central "theme" of this course is teaching students to do the following *conversion* correctly and efficiently:

$$\begin{aligned} \text{Geometry} &\Rightarrow \text{Representation} \Rightarrow \text{Algebra} \\ &\Rightarrow \text{Algorithm} \Rightarrow \text{Program} \end{aligned}$$

More precisely, a geometric object must first be converted to a representation (*e.g.*, polyhedron, parametric equations or implicit equations) from which an algebraic interpretation is obtained. Then, algorithms based on this representation are developed and corresponding programs are written. All course units are designed to follow this universal theme.

3 Course Content

Computing with Geometry is a 3-credit elective course for junior students in computer science, engineering and mathematics. The course prerequisites include calculus and linear algebra;[1] however, graphics is *not* required since non-CS students may not have this background. We have offered this elective course three times and are teaching the fourth time this year.[2] The course content is divided into eight units. The following sections briefly discuss the major topics that are covered and exercises.

3.1 Unit 1: Course Overview

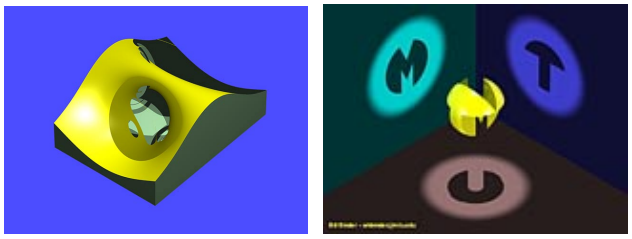
This course starts with an introduction to the field, followed by an elaboration of the "theme." This unit also touches upon the dimensional, geometrical, and combinatorial complexity of geometric problems, and the impact of finite precision on geometric computing. Figure 1(a) is the result of ray-tracing a cubic surface with an early version of a well-known system. Since this system used an "inaccurate" cubic equation solver, a smooth surface becomes scratched with holes.

3.2 Unit 2: Review of Geometric Concepts

This unit reviews important geometric concepts, including homogeneous coordinates and Euclidean, affine and projective transformations and their matrix representations. Computing with floating numbers will be further addressed with more examples of the effect of losing significant digits, error accumulation, and problems with associative and distributive laws. Interesting examples

[1]Calculus is required for the concepts of moving-triad, curvature, continuity issues, and surface characteristics such as the first and second fundamental forms and Dupin indicatrix, while linear algebra is needed for the discussion of transformations, the characterization of quadric surfaces, and other surface related topics.

[2]Since we are converting to a semester system from a quarter system, more materials will be covered.



(a) (b)

Figure 1: Loss of Significant Digits and Student Work

are used to show that, without careful programming practice, a theoretically sound stable iterative scheme can actually become chaotic. As an exercise, students are asked to construct examples that can demonstrate the failure of the associative and distributive laws.

3.3 Unit 3: Object Representations

Unit 3 covers representations of geometric objects. Topics include the wireframe model and its drawbacks (*e.g.*, ambiguity) and polyhedron models. A natural next step is the boundary representation in which flat facets and straight edges become surfaces and curves. The well-known winged-edge data structure is covered, followed by constructive solid geometry. Students are given all necessary tools and asked to construct an object from a cube or sphere using set union, intersection and difference so that the three side views would show the letters M, T and U. Student work is shown in Figure 1(b). A programming exercise asks the student to write a program that reads in a winged-edge data structure, displays the polyhedron, and performs some inquiries such as listing all edges/faces adjacent to a given vertex in clockwise order. Students receive a simple environment and implement their algorithms in C/C++ with a few simple OpenGL drawing primitives.

3.4 Unit 4: Parametric Curves and Surfaces

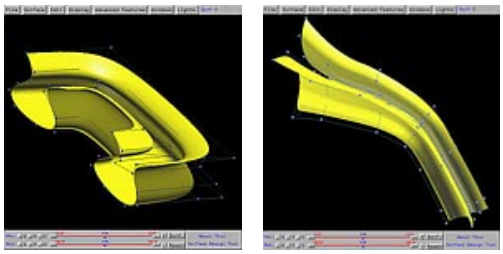
Fundamentals of parametric curves and surfaces and their related issues are discussed in Unit 4. Major topics concerning curves include (1) polynomial and rational curves, (2) tangent, normal, and binormal vectors (moving triad), (3) curvature and curvature sphere, (4) singular and inflection points, and (5) C^k- and G^k- (geometric) continuity. We use intuition based arguments to explain difficult-to-prove facts (*e.g.*, circles do not have polynomial parameterizations). The concept of points and lines at infinity provides students with an intuitive classification of conics. Parametric surfaces, surface patches, and isoparametric lines are also discussed. Exercises include the calculation of the moving triad and curvature at a given parameter and various types of continuity at the joint of two curves.

3.5 Unit 5: Bézier, B-splines and NURBS

Unit 5 presents important concepts of Bézier, B-spline and NURBS curves and surfaces, which provide a basis for our software tool DesignMentor [5, 6, 7]. We start with important properties of Bézier curves such as the convex hull property, partition of unity, affine invariance, and variation diminishing. Then, we move on to fundamental algorithms, including de Casteljau's algorithm, subdivision, degree elevation, and derivative computation. Applications such as font design are discussed. With this knowledge in hand, we proceed to Bézier surface patches, the 3D de Casteljau's algorithm, and joining patches together with C^1-continuity.

The discussion of B-spline curves is more involved. We start with a simple fact that a B-spline curve is merely several Bézier curves joined together. Then, we introduce knot vector, knot spans, clamping, and B-spline basis functions. Students will see and learn that only a few of these basis functions are non-zero on each knot span (*i.e.*, local control). Next, B-spline curves and surfaces are defined and their important properties are discussed and compared with those of the Bézier curves. Knot insertion, degree elevation, subdivision, de Boor's algorithm are also covered. The NURBS curves are introduced as the projections of 4-dimensional B-spline curves to the hyperplane $w = 1$. Since students have already learned about B-spline curves and surfaces, they can quickly pick up most of the important properties and algorithms of NURBS.

We also cover surface construction from curves using cross-sectional design techniques. These surfaces include ruled surfaces, surfaces of revolution, swung, swept and skinned surfaces (Figure 2).



(a) (b)

Figure 2: Swung and Skinned Surfaces

This unit has three programming assignments asking students to implement de Casteljau's algorithm, de Boor's algorithm with repeated knot insertion, and the 3D de Boor's algorithm for B-spline and NURBS surfaces. There are also hand calculation problems for knot insertions, degree elevations, and curve subdivision using de Casteljau's and de Boor's algorithms.

3.6 Unit 6: Implicit Curves and Surfaces

Implicit curves and surfaces and their applications are surveyed in this unit. We start with curve definitions, followed by the meaning and computation of tangent plane, normal vectors, first and second fundamental forms and curvature. Then, we proceed to important applications of implicit surfaces, including blending (*e.g.*, rolling ball, potential method, and Ricci method) and offset surfaces and their actual applications in computer-aided design and NC machining. Since curves and surfaces intersection requires deep mathematics, it is only briefly covered.

There are two assignments for this unit. The first asks students to design a model using *all* five non-degenerate quadric surfaces, and the second involves a simple surface tessellation. Figure 3 shows two samples.

(a) Quadrics (b) A Twisted Sphere

Figure 3: Student Work: Quadric and Tessellation

3.7 Unit 7: Computational Geometry Topics

This unit covers traditional computational geometry topics that are closely related to our need. These include convex hulls, Voronoi diagrams, Delauny triangulations, and polygon triangulations. We do not emphasize the theoretical nature such as proofs and complexity measures. Instead, we concentrate on the conceptual elements and tell students how and when to use a computational tool or technique. We also emphasize applications such as polyhedron editing and subdivision, surface reconstruction, and other topics that are currently very popular in visualization and medical imaging.

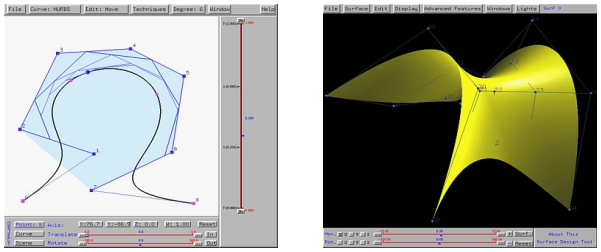
3.8 Unit 8: Robustness Issues

Problems involving inaccuracy and imprecise geometric input and their impact on geometric transformations are mentioned. Topics also include exact and interval arithmetics, robust algorithm design, and representative examples such as Dobkin's growing/shrinking pentagon and error accumulation of geometric transformations.

4 Software Tools

To help students learn more about the topics and gain deeper understanding, software tools were developed for the lab work. Some of these tools are interactive, allowing students to carry out experiments, while the others provide students with certain pre-defined animation sequences. In many cases, high quality public domain software such as POV-Ray are used.

The major component of our system, DesignMentor [5, 6, 7], is for visualizing properties of Bézier, B-spline and NURBS curves and surfaces. Students are provided with a drawing canvas with which they can design and modify one or more curves and surfaces. They can select, move and delete control points, show the control net and convex hull, and save and load curve and surface data. This system can display the inner working of de Casteljau's and de Boor's algorithms and their stepwise computation, and do knot insertion, degree elevation and subdivision. It also supports cross-sectional design as mentioned in Section 3.5. Figure 4 shows several features of a NURBS curve and surface.



(a) (b)

Figure 4: Our Curve and Surface Design Systems

Telling students that finite precision arithmetic may cause problems may not be strong enough. The best way to convince them is to show dramatic examples that can easily be recalled. We have a collection of many examples and images for showing the effect of losing of significance digits, error cumulation, and their impact to geometric computing. Examples also include quadratic equation solvers that use inaccurate methods and error accumulation in geometric transformations.

5 Course Evaluation

This course was taught in a computer-equipped classroom so that students can use a web-based electronic book and user guides, and practice the skills with DesignMentor. Most students were juniors.

The pre-test and post-test self assessments are used for course evaluation. Both have the same set of 17 questions, each of which asks the student to assess his or her level of understanding of a particular topic. The level of understanding ranges from 0 (no understanding) to 4 (excellent understanding). Table 1 shows that students assess their own understanding of the course topics as low ($17.8/17 \approx 1$) before the course and as very good ($49.37/17 \approx 3$) by the end of the course. A dependent t-test shows that the average gain score for the students who took both pre- and post-test is statistically significant with $p < 0.001$. Compared with quiz and exam scores, this was a fair assessment. Therefore, students did gain a good understanding of the subjects.

Table 1: Student Self Assessment Survey

	n	$\bar{x}$	σ	*Min*	*Max*	*Range*
Pre-test	20	17.80	7.83	7	37	30
Post-test	19	49.37	6.07	38	59	21
Gain	18	32.56	8.81	17	48	31

Reactions from students were also very positive. Here are some typical comments from an attitudinal survey: "the curve and surface programs were very nice and helpful", "POV-Ray assignments let us have a lot creativity", "the notes on the web were the best and most comprehensive of any class I've taken yet", and "[the exercises] helped understanding immensely". Almost all students prefer our intuitive and non-mathematical approach. There are, however, always exceptions. One student noted "I prefer a more rigorous approach, but generally balance was good."

6 Project Results and Dissemination

This project generated an electronic book, which is still evolving, three sets of user guides, and a software tool DesignMentor. The electronic book covers most topics presented in Section 3, while the user guides are for POV-Ray with a focus on modeling and DesignMentor. DesignMentor is written in C with OpenGL and GLUT, and supports SGI, SunOS, Solaris, Linux and Windows.

Our work was announced in early March 1999. At the end of Summer of 2000, our course information page, electronic book, and the curve and surface user guides had average daily hits of 9.3, 13.0, 5.2 and 3.1. Most of these hits were likely from off campus. There are about 1000 downloads. Table 2 is compiled from visitors' remote host addresses. *CS* covers all CS and related departments, including Math/CS, EECE, and EECS. When department information is not available, it is counted as `EDU` *Other* if the domain name is `EDU`. Those without domain names are in the *Other* category. The `EDU` domain total is 50.1%, which suggests that educators and students are interested in learning

computing with geometry and curve and surface design. Some CS educators' believe that computing with geometry, especially curve and surface design, has no place in CS curricula. Our data strongly suggest otherwise.

Table 2: Domain Distribution

CS	*Sci*	*Eng*	EDU *Other*	COM	*Other*
28.0	4.0	7.2	10.9	25.9	24.0

Table 3 shows a geographical distribution of the downloads. The top five places, in descending order, are US (39.3%), Germany (7.3%), France (6.9%), Sweden (3.0%) and UK (2.9%). The most wanted version is Windows 95/98/NT (68.0%). This is followed by SGI (16.0%), Linux (11.9%), and Sun Solaris/SunOS (4.1%).

Table 3: Geographical Distribution

N.Amer	*S.Amer*	*Europe*	*Far East*	*Other*
41.2	2.7	39.5	9.2	7.4

7 Conclusion

In this paper, we have presented the rationale and design merit of a computing with geometry course, and detailed its contents and software tools. We believe that computer science students who are equipped with the basic knowledge and skills of geometric computing will have a better chance in the job market, especially at the time when new manufacturing technology, computer-aided design and animation software development are entering a new era of importance. We found that educators and students are interested in computing with geometry. Our data also suggests that this project not only has a national impact, but also attracts visitors from over 50 countries. Moreover, we also have demonstrated that a carefully managed DUE project can blend research, curriculum design and software development together in an effective and productive way. We believe that this form of project development not only benefits our undergraduate and graduate students and faculty, but also sets a new model for curriculum development.

This project is only a first step in adding geometric computing to computer science curricula. Additional research work remains to be done, and our software development effort may not meet our long term goals. For example, we believe that stereo images are needed to help visualizing the geometry vividly, and that a virtual reality system may be used to step into the virtual world to touch and feel important geometric characteristics. Such extensions will benefit students and enable them to learn curve and surface design better. Moreover, other important and useful topics such as blossoming, and curve and surface interpolation and approximation might be included. We hope this ultimate goal can be realized with additional funding in the near future.

All materials, including a syllabus, student work, exercises and exams, an electronic book, an introduction to POV-Ray, the curve and surface user guides of DesignMentor, and various versions of DesignMentor, are available to the public at the following URL:

`http://www.cs.mtu.edu/~shene/NSF-2`

Acknowledgments

We thank our students Budirijanto Purnomo, Yuan Zhao and Yan Zhou for their excellent programming work, and many educators, students, engineers, and programmers who use our material and share with us their ideas and experience. Finally, we thank a number of reviewers in the past three years whose constructive comments have not only made this controversial paper better but also helped us further polish our work.

References

[1] A. Friedman, J. Glimm and J. Lavery, *The Mathematical and Computational Science in Emerging Manufacturing Technologies and Management Practices*, SIAM, Philadelphia, PA, 1992.

[2] J. Hartmanis and H. Lin (editors), *Computing in the Future: A Broader Agenda for Computer Science and Engineering*, National Academy Press, Washing, D.C., 1992.

[3] J. L. Lowther and C.-K. Shene, Geometric Computing in the Undergraduate Computer Science Curricula, *The Journal of Computing in Small Colleges*, Vol. 13 (1997), No. 2, pp. 50–61.

[4] C. Yap, *Report on NSF Workshop on Manufacturing and Computational Geometry*, Department of Computer Science, New York University, 1995.

[5] Y. Zhao, J. L. Lowther and C.-K. Shene, A Tool for Teaching Curve Design, *The Twenty-ninth SIGCSE Technical Symposium*, 1998, pp. 97–101.

[6] Y. Zhao, Y. Zhou, J. L. Lowther and C.-K. Shene, Cross-Sectional Design: A Tool for Computer Graphics and Computer-Aided Design Courses, *ASEE/IEEE Frontiers in Education*, Vol. II (1999), pp. (12b3-1)–(12b3-6).

[7] Y. Zhou, Y. Zhao, J. L. Lowther and C.-K. Shene, Teaching Surface Design Made Easy, *The Thirtieth SIGCSE Technical Symposium*, 1999, pp. 222–226.

A Three-Year Experience. Ã¢Âˆ—. John L. to computer science students and should have a place in computer ure 1(a) is the result of ray-tracing a cubic surface with an early Knot insertion, degree elevation, subdivision, de Boor's algorithm Computing with Geometry as an Undergraduate Course: A Three-Year Experienceâˆ— John L. Lowther and Ching-Kuang Sheneâ€ Department of Computer Science Michigan Technological University Houghton, MI 49931â€“1295 {john,shene}@mtu.edu. 1. Motivation. Computing with geometry is a rapidly evolving interdisciplinary ï¬eld involving computer science, en Course Structure. Your foundation year will give you an important grounding in computing, programming and essential mathematics before you move on to the rest of your degree. By the end of this year you will: Understand the fundamentals of computing.Â As a professional in this rapidly evolving sector, the ability to learn new skills is as important as what you know already. After successfully completing your degree you will have the knowledge to forge an exciting career, continually learning and extending yourself. You could go on to work as a software engineer, web or app developer, programmer, systems analyst, data scientist, artificial intelligence developer, academic/industrial researcher, entrepreneur, teacher or even researcher. Undergraduate Course Detail. AIST1110 Introduction to Computing using Python. This course aims to provide an intensive hands-on introduction to the Python programming language. Topics include Python programming language syntax, basic data types, operators for various data types, function definition and usage, file and operating system support, object-oriented programming, functional programming, module creation, visualization, multi-threaded programming, networking, cryptography, web/database access. The course will go through some important Python packages for artificial intelligence and mach