TEACHING NUMERICAL METHODS WITH A VIDEO GAME TO MECHANICAL ENGINEERS

Brianno D. Coller

Northern Illinois University, DeKalb, Illinois 60115, Email: amy@ece.uoj.edu

1. INTRODUCTION

At Northern Illinois University, we are developing an advanced computing track within the undergraduate mechanical engineering curriculum. The purpose of the track is to introduce our students to concepts normally taught to computer scientists: object oriented programming, data structures, complexity analysis, and elements of software design among others. Rather than ship our students to the computer science department to learn such topics, we provide an authentic engineering context designed to engage our students.

In this paper, we present just one element of the advanced computing track: our undergraduate numerical methods course. The course differs from our previous, more traditional, numerical methods course in two important ways. First, the course is project-based. That is, all the course material and all the assignments are built around a central task. Students in the class write computer code that drives a virtual car around a virtual track as fast as possible. Essentially it is a video game. A screen shot is shown in Figure 1. Although it looks like only a game, it is a serious and sophisticated simulation of an automobile, capturing the correct physics of the engine, transmission, differential, suspension, steering mechanism, brakes, tire mechanics, and more.

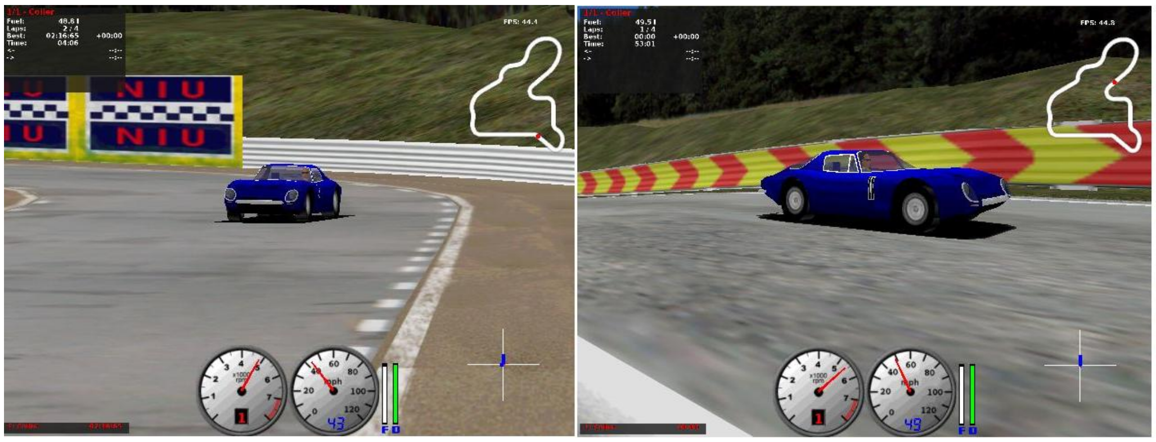


Figure 1: Screen shots of NIU-Torcs, the software which forms the basis of the numerical methods project.

Rather than simply generate numbers to fulfill the requirements of homework assignments, the project gives the students' computing a purpose. Their computing is intimately bound up with both the source of the problem and the use that is going to be made of the answers (Hamming, 1962). The project introduces new topics by making connections to driving, something for which most students have developed an intuitive understanding. Furthermore the project is fun, designed to engage students.

The second fundamental difference between this numerical methods course and the traditional one is that it continues to develop students' programming skills. Normally, students' formal training in computer programming ends after their first introductory class. They become proficient at writing only small self-contained programs, narrow in scope, limited in capability. In our new numerical methods course, students write hundreds of lines of computer code, interface with a larger body of software, begin learning object oriented programming and object oriented software design.

2. QUALIFICATIONS OF STUDENTS WHO TAKE THE COURSE

Students who participate in the course do so voluntarily. They have the option of taking the traditional numerical methods course which is offered in the fall semester and in summer school. They are told, up front, about the nature and scope of the class, and that it may require more work than the traditional numerical methods course.

Before the students are allowed to participate, the must have take all their calculus courses already, including differential equations. They have taken their elementary mechanics courses: both statics and dynamics. Finally, they have taken an introductory programming course from the compute science department. Their programming course is titled "Introduction to Programming in C++." Although the course has C++ in its title, the vast majority of time is spent on the procedural aspects of the language. The class construct is not introduced until the final week of lectures.

3. COMPUTING TASK

As previously stated, the students are given the task of writing computer code that instructs a computational model of a car to drive itself around a virtual race track. A layout of the race track as viewed from above is shown in Figure 2. It has straights in which the car can exceed 100 mph, and tight turns where the car must be going less than 30mph to maintain control. The turns are a combination of gentle and sharp, left and right. Not shown in the figure are elevation changes, banking, and anti-banking in turns.

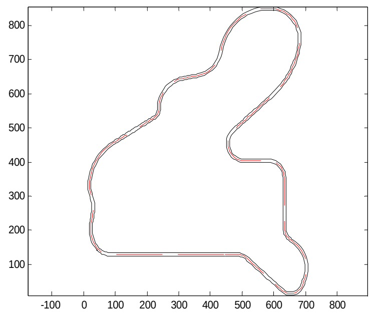


Figure 2. Standard race track.

In the beginning weeks of the semester, we give them a skeleton of a program that gets the car safely around the track. The driving style in this base program is reminiscent of the family station wagon, loaded with kids, on its way to grandma's house. By the end the semester, the goal is to have the car behaving as if Mario Andretti was behind the wheel, driving at the edge of controllability.

As an example of what the students must do in order to drive fast, consider the task of determining when to shift gears. The students do not have direct access to the torque vs. RPM curve for the engine of their virtual car. However, they can indirectly uncover all the necessary information by determining acceleration as a function of the car's speed in each of its gears. Students are able to calculate the car's speed by multiplying the rotation rate of the wheels by the wheel radii. To determine acceleration they must take a time derivative, numerically, of the velocity data. This is our mechanism for teaching finite differences. Here, their choice of forward difference, backward difference, or central difference scheme has practical significance, in sharp contrast to the problems one finds in textbooks. Furthermore, the order of the difference scheme has importance to the students who have a tangible interest in getting the most accurate calculation.

Upon computing the accelerations over a range of velocities in each of the four gears, one obtains a sequence of curves like that shown in Figure 3. The optimal shift points lie at the intersections of the curves. In order to determine these intersection points, students must write an interpolation routine for their discrete data, and then write an appropriate root-finding algorithm. Again, the nature of the problem has a meaningful impact on the methods chosen.

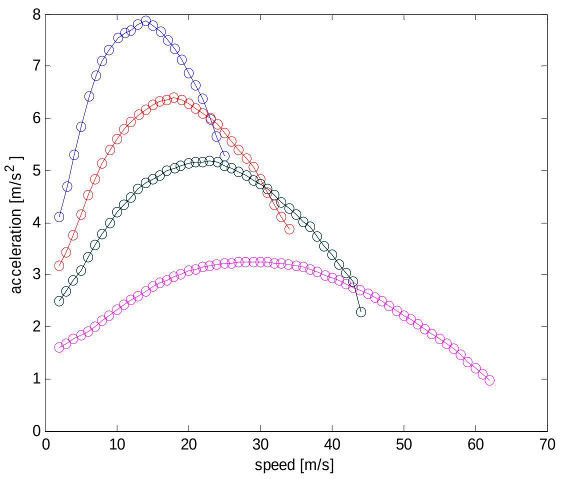


Figure 3. Acceleration data that students generate and then use to determine optimal shift points for their car.

In other exercises students employ numerical methods to calculate the maximum speeds at which their car can safely negotiate a turn. They calculate the latest possible instant to begin braking before entering a turn. They write algorithms for choosing the best “line” or path on the track which optimally cuts corners (Lopez, 2001), shedding precious seconds off of their lap time. In doing so, the learn how to perform polynomial interpolation, nonlinear regression, splines, solutions of linear algebraic equations, integration of functions, integration of differential equations, and more.

There is ample room for creativity. For example, students must devise strategies for passing other cars, and for recovering when their car slides off the asphalt onto slippery grass or sand. They have design choices to make when encoding the PID controllers for keeping the car centered on its desired trajectory, and for maintaining a desired speed through turns. Their driving algorithms are subject to many of the pitfalls common to real engineering practice: there are unmodeled dynamics, and the data they gather to feed their decision making algorithms are subject to noise. Therefore, the students have to decide on an appropriate degree of conservativeness or factor of safety when implementing their driving algorithms in the final race.

At the end of the semester students race in a friendly competition. First, they race their car for four laps around the track shown in Figure 2. Next, they race a mystery car for four laps around a mystery track. The mystery car has mass, suspension, and transmission parameters slightly different than those of the car that the students had been working with all semester. Students will not know these parameters before the race. Likewise, they will not know anything about the mystery track (its layout and surface friction) until just before the race. Therefore, students' code must be written general enough to adapt to new situations. This is one of the hallmarks of good programming.

Immediately before the mystery race, however, the students do have the opportunity to run their mystery car on two test tracks shown in Figure 4, which have the same friction properties as the mystery track. The oval track on the left of the figure has long straights that can be used for testing acceleration and braking. The fat circular test track is good for determining cornering abilities of the mystery car/track.

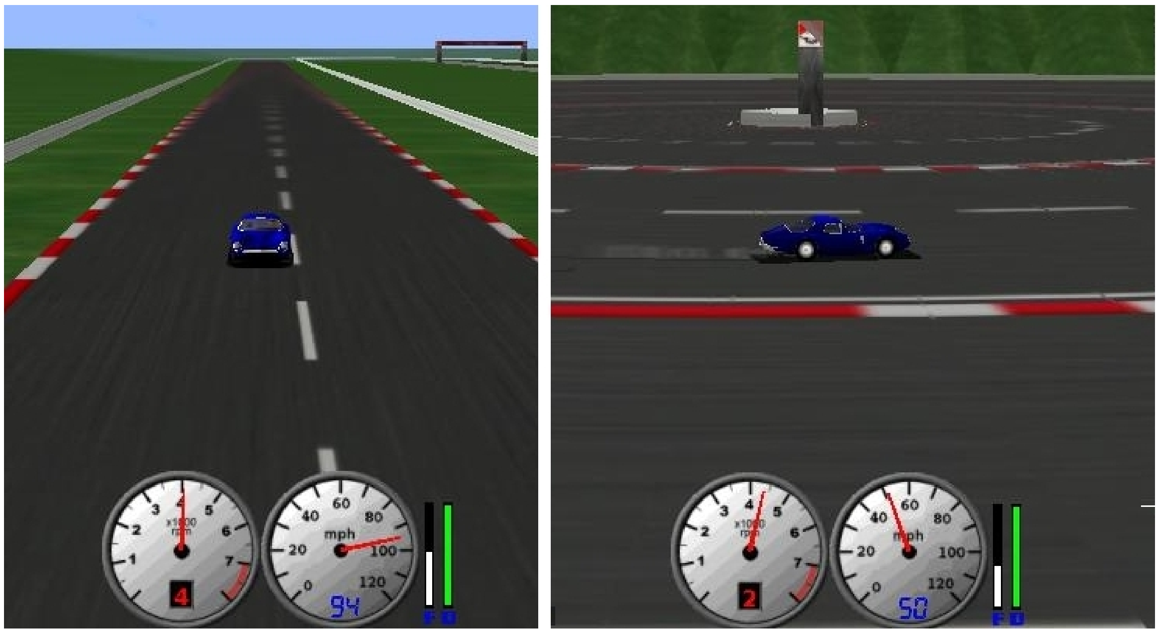


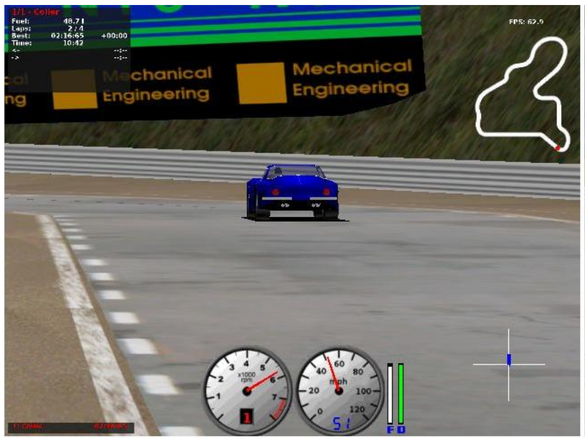
Figure 4. Test tracks used for developing driving algorithms.

In addition to the numerical methods, students gain considerable practice in writing computer code. Furthermore, they learn many of the fundamentals of object-based program design. They learn the importance of scalability, code reuse, and many lessons of computational efficiency.

4. CLOSING

As stated in the introduction, this updated numerical methods course is just one component in an advanced computing track within the undergraduate curriculum. The course that follows is a technical elective which builds upon the numerical methods course; we continue to use the race car simulation. This time, though, students write the code for the car itself. The final two courses in the curriculum track are capstone engineering design classes. Students will use the simulation tool that they have created to design and test a race car. This is precisely the type of design experience that industry is finding ever more valuable. Given the expense of building physical prototypes, companies would like to shift as much of the design process to the virtual domain as possible. We are giving our students such a design experience.

For more information on the curriculum track and on the software used in the courses, the reader is referred to Coller (2005). In the near future, we plan to make the software and course materials available to instructors who wish to adopt the course(s) at their institutions.



ACKNOWLEDGMENT

The project received support from the National Science Foundation under grant 0354557. Any opinions, findings, and conclusions are those of the author and do not necessarily reflect the views of the National Science Foundation. Other investigators on the project are M.J. Scott and U. Buy.

REFERENCES

Coller, B. D. (2005). Advanced Programming in the Mechanical Engineering Curriculum, *Proceedings of the ASEE Annual Conference*.

Hamming, R.W. (1962). *Numerical Methods for Scientists and Engineers*, Dover.

Lopez, C. (2001). *Going Faster!: Mastering the Art of Race Driving: the Skip Barber Racing School*, Bently Publishers.