Preface

In the past few years, I have been involved in the recruitment of both undergraduate and graduate students for the Intergraph Process, Power, & Marine (PP&M) division – the leading global provider of engineering software for the design, construction, and operation of plants, ships, and offshore facilities. In order to attract talented individuals from a diverse range of colleges, our job openings were posted not only in Alabama universities, but also in top ranked national universities such as Vanderbilt, Carnegie Mellon, Georgia Tech, Texas A&M, and Purdue. To identify the qualified candidates from amongst several hundred applicants, I sent the following seven pre-interview questions to the candidates and asked them to reply within two weeks with their answers to as many questions as possible. Many developers at Intergraph PP&M work with geometric creation and manipulation in one way or another. Therefore, the seven questions range from basic geometric analysis to numerical programming.

1. Area tolerance

No two components can be manufactured exactly the same. In practice, dimensional or geometric tolerances (e.g., distance and angular tolerances) are introduced in Computer Aided Design systems and manufacturing processes. The smaller the tolerance required, the more expensive the component will be to machine. Therefore, it is often desirable to specify the largest possible tolerance while maintaining proper functionality.

Denoting the distance tolerance by ε, then two circles are considered to be the same if

$$|R_1 - R_2| < \varepsilon,$$

where R_1 and R_2 are the radii of circles 1 and 2 respectively. Assume that we know the areas of the two circles (denoted by A_1 and A_2) and the radius of circle 1 (i.e., R_1). Please derive the "area tolerance" Δ in terms of the given distance tolerance ε such that, when $|A_1 - A_2| < \Delta$, we know two circles are equal with respect to the distance tolerance. One should avoid computing radii via $R_i = \sqrt{A_i/\pi}, i = 2, 3, \cdots, n$, as the square root computation is an expensive arithmetic operation. Although a non-calculus based approach is acceptable, a calculus based approach is preferable since calculus is the study of change.

2. Angular tolerance

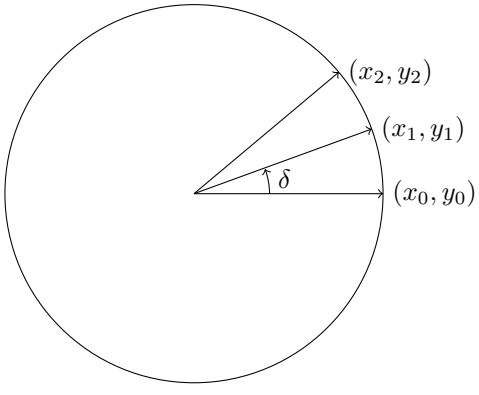
Again, we denote the distance tolerance by ε and assume that the model space (the longest model we would create) is 10^4 meters. Based on ε and the limit of model space, I would like you to derive the angular tolerance Δ such that it can be used to determine whether two lines confined in the model space are collinear.

Hint: Two lines are considered to be collinear if the maximum deviation between these two lines is less than the distance tolerance ε. From this criterion, you may derive the angular tolerance such that you know two lines are collinear with respect to ε if the angle between these two lines is less than the angular tolerance Δ. Drawing two lines on a piece of paper may help you analyze the problem.

3. Improve performance of circle stroking

Most graphics programming tools provide only a line drawing capability. To display a circle on computer screen, we first need to break the circle into evenly spaced tiny pieces such that each piece can be well-approximated by a line. We then draw these lines on the screen. Because these lines are so small, a circle looks smooth on the screen.

Breaking a circle into many tiny pieces is equivalent to computing many evenly spaced points on the circle, a process known as *circle stroking*. If we want to stroke $n+1$ points on circle and start the first point at $\theta_0 = 0$, then the angle increment δ would be $2\pi/n$ as shown below.



Accordingly, all other points may be computed as follows:

$$x_1 = r\cos(\theta_1) = r\cos(\theta_0 + \delta)$$
$$y_1 = r\sin(\theta_1) = r\sin(\theta_0 + \delta)$$

$$
\begin{aligned}
x_2 &= r\cos(\theta_2) = r\cos(\theta_1+\delta)\\
y_2 &= r\sin(\theta_2) = r\sin(\theta_1+\delta)\\
&\cdots\\
x_n &= r\cos(\theta_n) = r\cos(\theta_{n-1}+\delta)\\
y_n &= r\sin(\theta_n) = r\sin(\theta_{n-1}+\delta)
\end{aligned}
$$

The above formula indicates that we need to call trigonometric functions (i.e., sin and cos) for $2(n+1)$ times. Since trigonometric functions are relatively more expensive to compute than multiplication and division, the above approach is not optimized in terms of CPU usage. Please provide a suggestion on how you can speed up the computation.

4. Approximation by circular arc

A circle in quadratic form is

$$x^2 + y^2 + 2Ax + 2By + C = 0. \tag{0.0.1}$$

Its center and radius are given by $(-A,-B)$ and $\sqrt{A^2+B^2-C}$ respectively. Given n points (x_i, y_i) $(i = 1, 2, \cdots, n)$, we want to find a circle that interpolates (x_1, y_1) and (x_n, y_n) and approximates the remaining points in a least squares sense. Substituting (x_1, y_1) and (x_n, y_n) in the above equation gives

$$x_1^2 + y_1^2 + 2x_1A + 2y_1B + C = 0$$

$$x_n^2 + y_n^2 + 2x_nA + 2y_nB + C = 0$$

Subtracting the second equation from the first yields

$$2(x_1 - x_n)A + 2(y_1 - y_n)B + x_1^2 + y_1^2 - x_n^2 - y_n^2 = 0.$$

Assume that $|x_1 - x_n| \neq 0$ so we may express A in terms of B, i.e.,

$$A = \frac{y_1 - y_n}{x_n - x_1}B + \frac{x_1^2 + y_1^2 - x_n^2 - y_n^2}{2(x_n - x_1)} = \alpha B + \beta \tag{0.0.2}$$

Replacing A in equation (0.0.1) gives

$$2(x\alpha + y)B + C + x^2 + y^2 + 2x\beta = 0.$$

If an arbitrary point $\boldsymbol{p}_i = (x_i, y_i)$ is not on the circle, then

$$e_i = \left|2(x_i\alpha + y_i)B + C + x_i^2 + y_i^2 + 2x_i\right| > 0$$

is the approximation error. The sum of all squared errors is

$$\mathcal{E} = \sum_{i=2}^{n-1} \left(2(x_i\alpha + y_i)B + C + x_i^2 + y_i^2 + 2x_i\beta\right)^2$$

From calculus, $\mathcal{E}$ is minimized if

$$\frac{\partial \mathcal{E}}{\partial B} = 0 \quad \text{and} \quad \frac{\partial \mathcal{E}}{\partial C} = 0.$$

Optimized B and C are obtained by solving the above two linear equations and A is computed via equation (0.0.2).

Implementing the above approach will not result in a circle that interpolates (x_1, y_1) and (x_n, y_n). Do you see a reasoning problem in this approach?

5. Evaluation of e^x

Exponential function e^x and trigonometric functions such as $\sin x$ and $\cos x$ are often evaluated via the Maclaurin series (a special form of Taylor series). Expressing e^x by the Maclaurin series gives

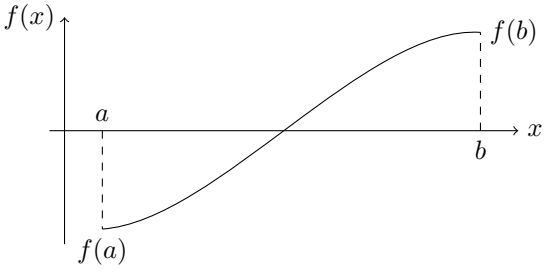
$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}.$$

This Maclaurin series converges to e^x absolutely and uniformly for $x \in (-\infty, +\infty)$. Please write a program to evaluate e^x via Maclaurin series at $x =$1 and $\pm$25 and compare your results with the ones obtained by using either a pocket calculator or the C/C++ internal math function `exp(x)`. An explanation of your implementation and findings is expected.

Hint: Let $P_n = \sum_{i=0}^{n} \frac{x^i}{i!}$. Then, P_n is said to be converged to e^x with respect to the given tolerance ε if $|P_n - P_{n-1}| = \left|\frac{x^n}{n!}\right| < \varepsilon$. Let's assume that $\varepsilon = 10^{-15}$.

6. Implementation of bisection method

Assume a continuous function f is monotonic on the interval $[a, b]$ and $f(a) \times f(b) < 0$. Then, there exists a solution x in $[a, b]$ such that $f(x) = 0$ as illustrated below.



To find the solution numerically, we can use the bisecting method. As the name suggests, we start by selecting x at the mid of the interval $[a, b]$ and compute $f(x)$. If $|f(x)| < \varepsilon$, x is the root of function f. Otherwise, we reduce the interval by half using the following criterion:

```
if (f(a) * f(x) < 0.0)
{
   b = x;
}
else
{
   a = x;
}
```

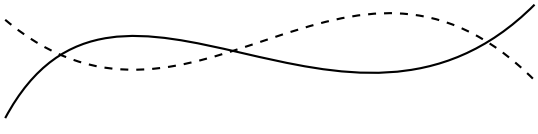
Based on the reduced interval, we compute again $x = 0.5(a+b)$ and check if $|f(x)| < \varepsilon$. Repeating this process should find the root of $f = 0$.

If you are not familiar with object oriented programming, you may simply implement a workable recursive function to compute the root of $f(x) = x^3 + 2x - 2$ in $[0, 1]$. Otherwise, I would like you to demonstrate how you can implement, say, `Bisection` class to compute the root of the above function. Furthermore, you may want to show me how you can compute the root of another function $f(x) = \sin(x)$ in $[2, 4]$ based on `Bisection` class with minimum additional implementation.

7. Curve/Curve intersection

Simple curves such as lines and arcs are widely used in Computer Aided Design and Manufacturing (CAD/CAM) systems. In addition to a line and arc, a piecewise polynomial curve (either a composite Bézier curve or a B-spline curve) is also used to model complex geometric shapes.

Assume we already have an API to solve a line/line intersection problem. I would like you to derive a method to compute the intersection points of two generic curves as shown below by calling the line/line intersection API.



Two curves intersect each other at multiple places

> I am not interested in C/C++ code implementation. Instead, I would like to see how you analyze the problem (math) and organize your data (computer science). If you want, you can write the pseudo code to clarify your approach.

Among those candidates who sent back their answers, very few could give satisfying analyses. I was quite surprised to find that candidates who were working toward their master's degrees in computer science were unable to demonstrate the desired programming skill, though they all indicated in their resumes that they are proficient in C/C++ and objected oriented programming.

It appears to me that there is a gap between academic training and real world computing. For example, mammals (or people) are often used in many text books to teach *inheritance* and *polymorphism* (the topics in *object oriented programming*). Engineering students are taught how mammals form a base class and how dogs and cats are derived from mammals and thus form the derived classes. The need for inheritance and polymorphism sounds very artificial and may fail to draw students' attention to the topic and may even cause their resistance to the idea. Therefore, when asked to use inheritance with polymorphism to design a bisection method that can be used to solve any function, candidates had no clue where to start. For this reason, I was motivated to write a numerical programming book to answer the above interview questions and, more importantly, to help serious science and engineering major students as well as entry level programmers learn the programming skills and become familiar with the software development process.

Readers may ask why I wanted to write another numerical programming book when there are already an abundance available. One simple reason is that those books emphasize more on numerical methods than on the software development process. Implementation of numerical methods is only one important part of developing computational software. Programming productivity, extensible and reusable code, quality control, and automated testing are also equally important parts of the software development process, which is the main theme of this book.

All examples in this book were carefully selected to demonstrate the strong need for mathematics and computer science in the software industry. To make this book accessible to wide spectrum of readers, the implementation of many algorithms emphasizes on understanding the underlying basics of techniques, not on the refinements that may, in practice, be needed to achieve optimal performance and reliability. Although most examples are mathematically oriented, they require no advanced mathematics beyond calculus and should be understood by engineering and computer science students and professional developers. Through these hands-on exercises, readers should be able to minimize the learning curve and become competent in designing and developing their own applications.

C/C++ and Microsoft Visual Studio are chosen in this book because they are both widely-used in software industry. If you do not have the professional version of Microsoft Visual Studio, you can download the Microsoft Visual Studio Express (a freeware) from

the Microsoft web site or simply use whatever development platform you already have. It should be pointed out that all algorithms are implemented such that they are readily understood by beginners. Excessive use of C/C++ pointer operations, STL iterator operations, and C/C++ specific syntaxes are purposely avoided so that readers can convert them to other programming languages with minor efforts.

There is a Chinese saying,

抛砖引玉

which may be translated as "*By showing you how to make a brick, we hope that you will refine the technique and skill to figure out how to make a piece of jade jewelry.*" Such is the intention of this book.

Organization of this book:

This book is written for those who want to pursue a career in developing computational software for engineering and scientific applications. Unlike traditional numerical programming books that focus on the analysis and implementation of numerical methods, this book emphasizes on the development of a reliable and reusable software package. Readers will not only learn implementation of numerical methods but also the software development process that includes creating and using a dynamic-link library, designing flexible test drivers, writing scripting tools for productivity, performing and validating an automated test suite. Based on the computational library developed in this book, readers will also learn how to develop a windows-based application for data visualization and manipulation. Multi-core processors bring parallel computing to mainstream customers. The shift to parallel computing leads to fundamental changes in the design of software. As a result, computer science students and programmers now need to learn parallel computing techniques that allow software to take advantage of the shift toward parallelism. For this reason, this book discusses also how classical numerical programs can be parallelized via Open Multi-Processing (OpenMP). In particular, readers will learn how to create multi-threaded applications with minimal and manageable changes in their serial code.

Numerical methods discussed in this book include evaluation of polynomial and series, root-finding, linear and nonlinear systems, inverse of a matrix, eigenvalues and eigenvector, integration, and least squares approximation. These methods are grouped and taught based on their implementation styles rather than their relevance. By completing these hands-on programming exercises, readers will gain an assured confidence and the sound programming skills needed to design and implement their own computational software that include creation of sharable library, test drivers, test suites, automation scripts, data comparison and validation tools.

Chapter 1 is a fast-paced brief introduction to C/C++ programming under Microsoft Visual Studio to familiarize readers with basic C/C++ syntax and commonly used debugging tools.

Chapter 2 discusses floating-point notation, comparison, and arithmetic. One important thing about computing over a set of floating-point numbers is that it is not arithmetically closed. If we take two floating-point numbers x, y and choose an arithmetic operator $(+, -, \times, \div)$ then, in general, the result will not be exactly representable in the floating-point system no matter how high precision is used. Failure to understand such limitation is often the source of problems in numerical programming. Therefore, rudimentary understanding of floating-point is a pre-requisite for programmers.

Chapter 3 continues the study of advanced C/C++ programming such as default arguments, data structure and class, double pointers, dynamic memory allocations, and STL containers. Algorithm efficiency analysis and big O notation will also be discussed. This chapter is designed to help readers to gain the required C/C++ proficiency in implementing numerical methods as well as to understand the importance of good analysis and design of numerical methods. Examples in this chapter are specially selected to illustrate how well-designed numerical methods can improve not only performance but also numerical stability.

Chapter 4 is devoted to give readers an insight on how a computational software library may actually be developed in a software house. Readers will learn how to create and use a dynamic-link library, how to design flexible test drivers, and how to write scripts to improve productivity, to execute test suites automatically, and to compare the test results with the predicted outcomes.

Chapter 5 deals with recursive algorithm. Because of its problem-solving power and simplicity in implementation, the recursive algorithm is a preferred powerful problem-solving tool. Recursion in numerical methods will be discussed in this chapter with emphasizes on performance and memory usage.

Chapter 6 discusses linear systems. Topics include solution to system of linear equations, matrix manipulation, inverse of a matrix, eigenvalue and eigenvector.

Chapter 7 and 8 cover an important topic in software engineering: design extensible and reusable code. Many numerical programming books focus more on understanding the underlying basics of techniques than on how to design a reusable code. These two chapters explore how to use function pointers, "generic" data pointer, and inheritance with polymorphism to design extensible and reusable code.

Chapter 9 discusses the least square approximation method whose applications can be found in many fields such as computer aided design, metrology, image processing, and so on. Starting with the best fit line, circle, arc, and plane, we expand the discussion to the generic normal equations of least squares approximation. Examples are specially selected to draw readers' attention to how math plays a vital role in software development.

Chapter 10 aims to develop a simple windows-based application for data visualization and manipulation. CAD/CAM systems have revolutionized much of the engineering design and manufacturing processes. A CAD system is essentially a collection of sophisticated tools that are exposed to users via graphical user interface (GUI) for

creating, visualizing, and manipulating data. Through this miniature application, readers will get a glimpse of how sophisticated CAD/CAM systems are developed.

Chapter 11 discusses how classical numerical methods can be parallelized to take the advantage of multi-thread programming. As multi-core processors bring parallel computing to mainstream customers, the key challenge in computing today is to transition the software industry to parallel programming. Programmers at all levels will soon find that they will have to deal with parallel programming whether they are ready or not. Common problems associated with parallel computing such as data race conditions, workload balance, synchronization, and parallel slowdown are discussed in detail.

Appendix A is a brief introduction to Perl programming.

Appendix B contains answers to all seven pre-interview questions presented in the preface.

Reference resources:

Wikipedia, Microsoft Developer Network (MSDN), and Intel web sites have been the primary resources of reference for terminologies and historical events such as advancements of parallel programming and Microsoft products.

Acknowledgement:

I would like to take this opportunity to thank my colleagues at the Intergraph Math department. It is my honor and privilege to work with and learn from those talented individuals. In particular, I want to thank Dr. Lujun Wang who, as a new developer, used the manuscript of this book as training material and provided me with valuable feedback. My special thank also goes to my friend and former colleague Dr. Jean-Jacques Malosse, who was my mentor and set a high bar for me when I joined Intergraph nineteen years ago.

About the author:

Dr. Yong-Ming Li is a senior technical manager at Intergraph Corporation with 19 years of professional software development experience. Dr. Li specializes in curve, surface and solid modeling. He has developed numerous geometric modeling algorithms to support SolidEdge®, SmartPlant®, and SmartMarine®. He received his bachelor's degree in mechanical engineering from Chengdu University of Science and Technology in China. While studying for his master's degree at Northwestern Polytechnical University in Xi'an, China, Yong-Ming Li received the *Sino-British Friendship Scholarship* and went to England in 1988 to continue his postgraduate study. At the University of Birmingham, he earned an M.Sc. in production engineering and Ph.D. in mechanical engineering with an emphasis on geometric modeling and computer-aided design.