

# Computational Geometry and Image Processing Applications for an Undergraduate Algorithms Course

Daniel E. Stevenson  
Computer Science  
University of Wisconsin – Eau Claire  
Eau Claire, WI 54702  
stevende@uwec.edu

## Abstract

Image related computations are becoming mainstream in today's computing environment. However, Computer Science departments are still offering image related courses as electives or at the graduate level. Since imaging is becoming a core topic, it needs to be covered in the core courses. Most departments cannot afford to add another course into their curriculum at this level, so they only remaining choice is to integrate image related applications into the existing core courses. This paper addresses this issue as it relates to an Algorithms course. The specific prerequisites and goals of our Algorithms course are described. The image related applications that have been adopted are presented and their impact on the course is discussed.

## 1. Introduction

Traditionally, computational geometry and image processing concepts are taught either at the graduate level or as senior level electives at the undergraduate level. This is an unfortunate situation for a couple of reasons.

First, image manipulation is becoming increasingly common in everyday computing. Thus, it no longer belongs as a graduate or elective topic. These ideas now need to be introduced as soon as possible in the curriculum to give students the exposure they need for today's computing environment.

Second, both computational geometry and image processing provide a wealth of practical, real-world examples. Traditional "toy" examples, such as "Towers of Hanoi" are used in most introductory classes. These examples have the benefit that they are easy for students to understand and, thus, invaluable in learning basic concepts. However, to most students they lack connection with the real world. Practical applications that connect what the students are learning to the real world are necessary to stimulate the minds of today's students, especially with the increasing majority of undergraduates deciding not to pursue graduate work. In my experience, they are always looking for how the concepts they are learning will benefit them when they get a job. Showing them how the theory they just learned can be directly applied to more interesting applications,

such as those that image processing can provide, really seems to capture the minds of my students. For example, I had several students last year ask me how they can take their programming projects and get them up onto the web so they can have their friends, who are not Computer Science majors, look at the type of work they are doing.

This paper discusses some of the projects and classroom examples I have used in our Algorithms class here at the University of Wisconsin – Eau Claire. There are several other courses that could also benefit from such integration. Data Structures is prime candidate [7]. However, there are also potential applications in Software Engineering as well as Programming Language Theory.

The placement of the Algorithms class in our curriculum as well as its prerequisites and goals is discussed in Section 2. Section 3 covers the specific computational geometry and image processing examples and projects used in the class. And finally, Section 4 provides a discussion of these projects.

## **2. Algorithm Course Background**

Before the integration of image computation into an Algorithm class is discussed, I need to describe the basic format and placement of the Algorithms class here at Wisconsin – Eau Claire.

### **2.1 Prerequisites**

Algorithms is taken in the Fall of Sophomore year. It is the 3<sup>rd</sup> course in the major. The two previous courses are a survey course and a programming language course.

#### **2.1.1 Survey Course**

The survey course covers a variety of computer science topics from database access using SQL to number system representation. It is designed so that we don't need to keep reteaching these topics as they appear in subsequent courses. The Algorithms course doesn't gain as much from this course as other classes, such as Databases and Architecture. Thus, a course like this is not a necessary prerequisite to adapt the Algorithm projects outlined below.

#### **2.1.2 Java Programming**

The 2<sup>nd</sup> course in the major is a Java course. This course not only introduces programming, but also has a heavy focus on proper OO design methods. Students are always anxious to get to GUI programming in this Java course. However we have found that GUI programming is best understood if it is presented at the tail end of this class. GUI programming in Java involves too intimate a knowledge of how interfaces and inheritance work to effectively teach it much earlier. Thus, coming into the Algorithms class, students have a fairly good knowledge of basic Java Object-Oriented programming skills in addition to some basic GUI skills.

### **2.1.3 Math**

In addition to the two Computer Sciences classes listed above, Calculus I is also a required prerequisite for the Algorithms course. In my opinion this is not enough math to do either complex algorithm analysis or advanced image processing. However, the students are also taking a Discrete Math course at the same time as Algorithms. Thus, I try to leverage off their partial knowledge of Discrete Math as much as possible. In addition, I also end up showing them a lot of the relevant math on the board as we cover each topic. This is especially true when we cover topics that require geometry or linear algebra.

### **2.1.4 Data Structures**

Unlike traditional Algorithm courses, our course comes before the Data Structures class (the 4<sup>th</sup> course in our curriculum). We find that this order works quite well given the built-in data structure libraries that come with most modern languages environments. For example, we make extensive use of the Java Generic Library (a port of the C++ Standard Template Library) as well as the Java Collections Framework. The idea is to let students use the built-in data structures to code their algorithms. Then, in the next course, they build and analyze their own data structures. This approach has minimal impact on the Algorithms course since we can use the built-in libraries. However, it gives the students the ability to code algorithms to test new data structures once they reach the Data Structures course.

## **2.2 Goals**

Obviously, knowledge of algorithms is one of the major goals of this course. However, there are several additional goals we, as a department, have mapped out for this particular course. Namely, a continued focus on OO design skills including the introduction of Frameworks, the design of more advanced GUI front-ends, and the concept of multiple threads of execution.

### **2.2.1 Frameworks**

The use of OO Frameworks is a major departure from the traditional Algorithms course. The traditional courses tend to focus on application areas and present different problem solving techniques that can be used with each area. For example, a number of different algorithms can be shown that will solve various aspects of graph problems [1]. Although this arms the student with the ability to solve graph problems, it doesn't generalize well to solving other types of problems.

The Frameworks approach is the opposite. The focus is on the problem solving techniques (i.e. Divide-and-Conquer, Dynamic Programming, Greedy, Backtracking, etc) [2, 3]. For each technique, it is shown that it can apply to any number of different application areas. Not only does this make it easier for students to apply their knowledge

to new application areas they haven't seen before, but in many cases we can engineer the code so that it is reusable as well.

To make reusable code, we build generic Frameworks for each technique. Then to solve a particular problem, we build problem specific code that implements a particular interface. We then plug this implementation into our Framework, which only knows about the interface and not the specific problem it is solving.

This allows us to reuse code for a particular algorithm, say Backtracking code, on many different problems that can be solved using that approach. In addition, it also allows us to easily compare different problem solving techniques for the same problem without having to recode the problem again and again. That is, we just plug in the problem specific code into multiple problem solving frameworks and directly compare the results.

### **2.2.2 GUI Programming**

As was mentioned in the prerequisite section above, the students will only have a basic knowledge of building GUI front-end for their applications coming into this course. Thus, one of the goals is to introduce more advanced GUI programming concepts. This implies that every programming assignment should have a GUI front-end. These front-ends should cover everything from the display of simple components, such as buttons, to 2D drawing and display of images. 3D GUIs are not covered in this course.

### **2.2.3 Threads**

The most recent goal added to this course is the coverage of multiple threads of execution. We don't want to discuss all the aspects of synchronization and deadlock, as these are topics for an Operating Systems class. However, we want early coverage of the basic ideas of Java Threads in our curriculum since we feel distributed computation is an important part of the future of computing.

## **3. Integration of Image Related Algorithm**

Given the above prerequisites and goals of our Algorithms course, I set out to find examples that could be used for programming projects as well as on the board during lecture. What I have done in the past is try to provide a mix of various problems from different domains so that the students see that these problem-solving techniques are useful for real-world problems and not just for the traditional "toy" problems. Thus, I don't use image related applications for all of my examples, but the more interesting ones currently in the course are described below.

### **3.1 Convex Hulls**

Convex Hulls provide a wealth of possibilities for an Algorithms class. The following sub-sections show how they are used in our course.

### 3.1.1 Complexity and Sorting

The first topic covered is a high-level description of worst-case time complexity. This is something that needs to be done before any of the problem solving techniques are discussed. It will give us a measure to compare the different techniques to each other. Traditionally, the concept of time complexity can be explained using searching or sorting algorithms. For example, there are a number of different methods that perform sorting and each method has a particular time complexity. These complexities can be computed analytically and also measured experimentally. Sorting is an easy concept for students to understand, so this is the first example shown to them in our course.

However, sorting by itself is a bit dry and it is hard for many students to get motivated about finding better ways to sort large lists of numbers. This is when convex hulls are first introduced [5,6]. Actually, they are introduced through an application that can benefit greatly from convex hulls such as robotic path planning or raytracing. The interesting thing about convex hulls is that there are some direct parallels to sorting algorithms [6]. There is a brute force algorithm just like there is in sorting. Graham's Scan uses sorting as its first step. The Jarvis March is similar to Selection Sort. The MergeHull and QuickHull algorithms are similar to MergeSort and QuickSort, respectively. The same complexity arguments can be made for these convex hull algorithms as were made for the sorting algorithms, including the relative balancing of the two divide-and-conquer algorithm. Thus, the introduction of convex hulls has led to two benefits for the course. First, it is easy to produce examples that have a huge number of sample points. This drives home the point that problems do in fact get big and one actually does need to worry about algorithm complexity. Second, it is an interesting exercise to have the students try to draw parallels between the various convex hull algorithm and the sorting algorithms (Of course, you have to change the name of MergeHull and QuickHull!). Taking complex problems and reducing them to simpler ones is a good technique for students to learn.

### 3.1.2 QuickHull

Since time is spent describing the details of convex hulls when describing complexity, it seems only natural to make the first programming project involve convex hulls. Last semester the students were asked to design a divide-and-conquer framework. The concepts of divide-and-conquer were originally described with some easy to understand traditional examples such as sorting and searching. This enabled the students to get a firm grasp on the method with examples that they could write down on paper. However, divide-and-conquer is a powerful method and it would be a shame to only show it in action with a problem as simple as sorting. Thus, they were asked to implement a convex hull specific problem to plug into their divide-and-conquer framework. The particular algorithm they had to implement was the QuickHull algorithm.

In the process of implementing QuickHull, some additional point and line computations need to be performed. Specifically, one needs to know how to compute the distance from a point to a line as well as perform a test to determine which side of a line a particular

point lies on. The Java2D API actually provides methods that perform both of these sub-tasks. However, before I told the students this fact, I posed the problem and made them come up with solutions in lecture. For the actual assignment, they were allowed to use the Java2D API methods. However, the discussions in lecture were quite interesting as I tried to guide the students into using their long-forgotten geometric math skills. Honing these geometric skills I believe to be essential to being able to understand image-processing algorithms.

In addition to the benefits of this assignment listed above, there was also the benefit of building a GUI front-end. The students were required to provide a GUI for their QuickHull algorithm that was able to show the starting set of pointing and the final convex hull. Extra credit was given to those students who could graphically show the algorithm's progress as it ran. The students found the GUI to be a valuable part of the assignment, not only from the aspect of learning how to build GUIs in Java but also in terms debugging their programs. Convex Hull algorithms are hard to debug if all you have is text output to work with.

### **3.2.3 Approximation Algorithms**

Towards the end of the course, Approximation algorithms are discussed. This topic occurs after seeing both Backtracking and Branch-and-Bound. Both these techniques can lead to large time-complexities for given problems. Approximation algorithms are introduced for the situations where it is more important to have an approximate answer quickly than it is to have the correct answer after a long computation. There are many situations in image processing where approximate answers are good enough. One example that I use in class to illustrate this idea is the approximation algorithm for computing convex hulls given in [6]. At this point in the semester convex hulls make a good example since the students have dealt with them extensively. Additionally, the graphical nature of the hull computation makes on-line demos quite effective.

## **3.2 Triangulation**

One of the techniques discussed in the course is the Greedy method. Last semester I decided to stick with the traditional shortest path problem as the assignment that went along with the discussion of the Greedy method. However, in lecture I presented a Greedy method to perform a triangulation. Note that the triangulation problem could be given as the programming assignment instead of the shortest path problem just as easily.

The Greedy triangulation is performed by always adding the shortest edge that doesn't intersect any existing edges in the triangulation. This continues until the proper number of edges has been reached. Not only does this graphically illustrate the simple principle of the Greedy method, but the implementation also involves learning some additional geometric concepts. For example, we need to be able to determine if two line segments intersect. This is made more difficult by the fact that they we want to discount those intersections that occur at the endpoints, given possibly noisy data. In addition, we also

need to know when to stop adding edges. This can be calculated using facts about the geometry of triangulations.

### 3.3 Convolution

The last topic we cover in the course is an introduction to parallel algorithms. The main purpose is to introduce students to the concepts of Threads and Synchronization [4]. In a later Operating Systems class these concepts are refined, but an early introduction is a good idea with the current movement toward distributed computing.

The parallel programming project used last fall was an implementation of a convolution filter. The concept of convolution was first discussed without parallel issues. Most students had used programs like Photoshop before and had an understanding of what image filters could do. However, they were excited to learn what actually happened when these filters were applied. The discussion of the brute-force method convolution uses to perform filtering went fairly quickly. However, the more interesting aspect, which we spent several days discussing, was the construction of the filter masks. Students were amazed that a simple set of numbers could do such a variety of interesting things to images. The general discussion of filters in lecture turned into a detailed description of the nature of masks and how they are created. This naturally led to discussions about Gaussians, cosine waves, discrete 1<sup>st</sup> derivatives, etc. The idea of doing discrete derivatives was a new concept to most people in the class. Calculus I was a prerequisite, but the idea of doing it discretely was a new and interesting concept to them. I can't emphasize how much I was impressed with student initiative on this topic. For example, I didn't assign this as homework, but several students went out and found programs like Photoshop that can show the values they use for their various masks. They would bring these values to lab and we would have long discussions on how the particular mask was created and what it was actually doing to the image.

The actual programming assignment was to implement a convolution filter that took an image and a mask as input, convolved the image with the mask, and produced a new image. This required that they learn how Java handles pixel level information in images, which is harder than it first appears. Additionally, they had to simulate the process of convolution as a parallel algorithm by breaking up the image based on a given number of processors. They had to fork off Threads to process each sub-image. Synchronization was involved in having to wait until the entire convolution was complete before continuing. In addition, a GUI interface was built that could show the image before it was processed and again after it was processed. Some of the students took it upon themselves to also show the process of convolution graphically on the image as it was performed. This addition to the GUI will be put into the assignment next time I use it because it clearly showed how the sub-images were being processed independently.

## 4. Discussion

The integration of these computational geometry and image processing algorithms into our Algorithms course addresses a number of issues.

First are the current problems in core Computer Science courses. One such problem is the exclusive use of “toy” examples without ever getting to more interesting real-world applications. The convex hull and triangulation algorithms can be easily framed as applications whose first step requires the respective method. Students can directly see the use of the convolution filter from using programs like Photoshop. From my experience, the students find these applications more stimulating than the traditional examples. Another problem with the core courses is that they need to cover more imaging topics since image manipulation should be more than only an elective topic in today’s computing environment. Again, these applications help solve this problem.

The other issues that addressed with the image related applications were the goals our department had set down for the Algorithms course. The use of Frameworks turns out to be extremely valuable in integrating image-processing applications. I didn’t need to introduce an additional topic into the class called “image computation methods” to go along with the traditional “graph methods” or “matrix methods”. Instead, I was able to keep the original topics of the course intact (i.e. Divide-and-Conquer, Dynamic Programming, Greedy, Backtracking, Branch-and-Bound, Approximation Algorithms, and Parallel Processing) and just provide image related applications for these problem-solving techniques. Additionally, I found it to be valuable to introduce each concept with a simple traditional example before I discussed the image related example. This not only gave the students a firm grasp on the concept before they saw the more complex real-world application, but also encouraged them to look for relationships between real-world problems and the basic theories they learn in all their Computer Science courses. Additionally, GUI building and Thread concepts needed to be covered in this particular course. Image related computations almost always benefit from a GUI front-end and many can be performed using parallel processing.

Overall, I feel that image related applications are a natural fit for an Algorithms course. They can be demonstrated graphically and have the property that they usually contain a large enough number of points/lines/etc that one has to worry about complexity issues for even the average problem. For next year’s version of the course, I am planning even more integration of image related applications. Specifically, I am currently looking for good image based examples to use for the Dynamic Programming and Backtracking portions of the course. I hope that others consider such integration into their Algorithms course as well.

## References

1. Cormen, Leiserson and Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
2. Horowitz, Sahni and Rajsekar, *Computer Algorithms / C++*, Computer Science Press, New York, NY. 1997.
3. Neapolitan and Naimipour, *Foundations of Algorithms using C++ pseudocode*, Jones and Bartlett, Toronto, ON, 1998.
4. Oaks and Wong, *Java Threads*, O’Reilly, Sebastopol, CA. 1997.

5. O'Rourke, *Computational Geometry in C*. Cambridge University Press, New York, NY. 1993.
6. Preparata and Shamos, *Computational Geometry an Introduction*. Springer-Verlag, New York, NY 1985.
7. Sarkar and Goldgof, "Integrating image computation in undergraduate level data-structure education", v. 12, n. 8, pp. 1071-1080, 1998.
8. Weiss, *Data Structures & Problem Solving Using Java*, Addison-Wesley, Reading, MA. 1998.