Cutting Trajectory Simulation

Paul Linca
linca_paul@yahoo.com

Abstract: Laser cutting machines have experienced a significant increase in popularity over the past years. Since the precision of these machines is paramount, the algorithms that have to be applied to compute the cutting trajectory must be perfectly implemented and tested. Thus, the purpose of this project is to implement a reliable algorithm to be applied and to accurately simulate the behavior of these intricate instruments. We aim to show a method of using graphics processing libraries and programming designs to serve as a solution to these kinds of problems. We will show how the SDL 2.0 library along with computer graphics algorithms dramatically simplify the process of rendering and showing the user a simulation of how a machine should work in real life.

Keywords: Cutting trajectory, graphics processing, SDL 2.0.

1. Introduction

Laser cutting is a technology that makes use of lasers in order to precisely cut materials in a predefined manner. Since it is predominantly used in all manufacturing processes, it has become essential that the laser cutting be performed accurately and efficiently so that the production costs and dangers are significantly reduced. In order to guarantee that the machines work properly a simulation of their behavior must be carried out beforehand.

In this paper, we discuss an implementation of computing the laser cutting trajectory of such industrial machines as well as provide a graphical simulation of it. There are many computer graphical processing libraries and frameworks that can be used to accurately render images that could be interpreted as simulations of the real world. Each library has its own compatible algorithms for computing coordinates in a 2D space. One such library is OpenGL. OpenGL is a cross-language and cross-platform programming interface for rendering 2D graphics that directly communicates with the graphical processing unit. It is a very powerful and well-known tool but since for the problem at hand, power is not a must, I decided to choose an alternative.

The solution I have chosen to this problem is the cross-platform software development library SDL 2.0. It provides a low-level access to keyboard, mouse and graphics hardware. That being said, I consider it a perfect fit for solving the problem at hand. Many respected programmers have aided the development of pixel manipulating algorithms that are compatible with SDL 2.0. As a programming language I have chosen C++ which is perfectly compatible with the graphics library and provides easy methods for arithmetic computations and reading data from streams such as text files.

2. Procedure and Algorithms

The problem of computing the laser trajectory such that it can be graphically rendered is a process that can be split into four simple steps.

Firstly, the input lines and arcs will be transmitted to our program through a text file as coordinates. We have designed our project such that the input text file should have the following design. Each line or arc/circle will have it's coordinates on a separate line of the text file. Furthermore, each line will have a number which will represent a flag for the program to correctly read the data. The flag can take values of '1' or '2', depending if the line contains a line or an arc/circle respectively.

Secondly, a canvas onto which the trajectory will be rendered must be created. This is a problem to which SDL 2.0 provides an incredibly simple solution. The graphics library can create a separate window with custom dimensions, position and flags. The function in question is SDL_CreateWindow(). The desired result looks like *Figure 1*. This will be our blank canvas on which lines and arcs/circles will be rendered.

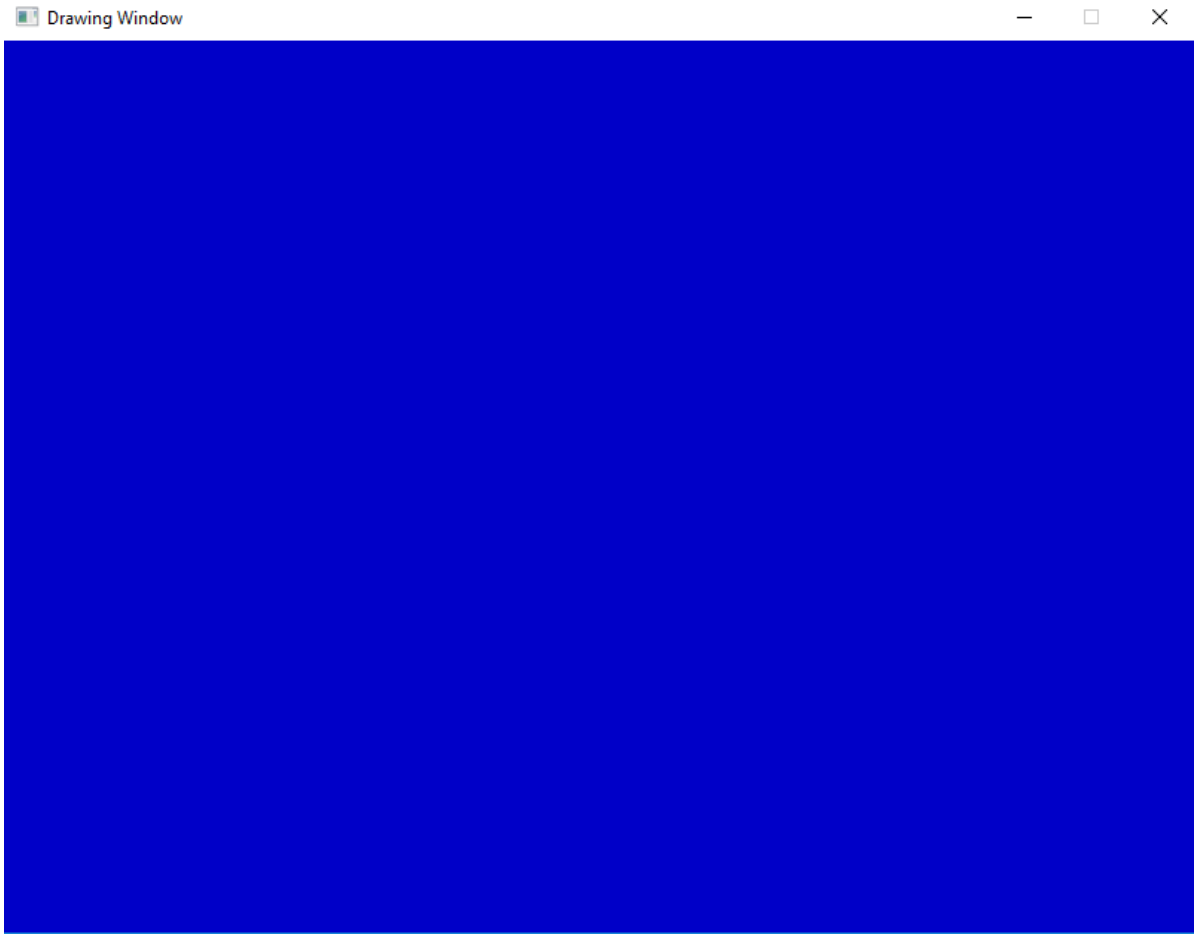

Figure 1, SDL Window

2.1 Bresenham's algorithm

The third step is the line drawing. One of the most well-known algorithms that solves this problem is Bresenham's line algorithm. This algorithm determines the points that should be selected in order to build an approximated straight line between two given points. The way Bresenham implemented his solution is only using integer addition, subtraction and bit shifting which is the factor that makes his algorithm very efficient. The algorithm computes each pixel needed from the start pixel to reach the end pixel. It does so by calculating the slope of the resulting line and gradually incrementing or decrementing the current coordinates until they correspond to the final pixel. After this, each pixel will be colored and rendered on the previously created window using the function SDL_RenderDrawPoint(). One of Bresenham's algorithm's strongest characteristics is that it avoids floating point multiplication and addition.

Consider the endpoints of the line as being (x_0, y_0) and (x_1, y_1). The algorithm chooses the next integer coordinates that are closest to the ideal coordinates on the screen. Thus, on successive rows x can remain the same or increase by 1 and on successive columns y can remain the same or increase by 1. The general equations of the line through endpoints is given by the following figure.

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}$$

Figure 2, line equation.

That being said, we can easily calculate, for example, the next y coordinate for a known column by rounding the quantity in *Figure 2* to the nearest integer like in *Figure 3.*

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$$

Figure 3, next y coordinate.

2.2 Midpoint algorithm

The last step that needs to be addressed is computing and rendering circle arcs. The Midpoint algorithm is the perfect solution to this. By only providing the center point and radius of the resulting circle, the Midpoint algorithm calculates, similarly to Bresenham's, the perimeter points of the circle in the first octant and mirrors them to the other octants. For the desired simulation we had to introduce a small change to the traditional algorithm. Since Midpoint renders all octants at the same time, it would provide an unrealistic laser cutting simulation. Thus, our implementation renders the octants in order.

Considering the first octant, for any given pixel (x, y) the next pixel to be rendered will either be (x, y + 1) or (x – 1, y + 1). The decision between the two can be boiled down to finding the mid-point of the possible pixels, i.e. (x – 0.5, y + 1) and, depending on its position relative to the circle perimeter we choose the first one if it lies inside or the second one if it lies outside its boundary. After computing the all the pixels of an octant, we need only to mirror them to their corresponding pixels on the other seven octants. Of course, these values need to be sorted and calculated accordingly.

2.3 Additional functionalities

To the base functionalities of the project, we decided to implement additional functionalities / improvements. After the rendering of the preset lines and arcs, whose coordinates have been read from a file, the user can furthermore add lines or arcs using the mouse as input. Thus, the user can left click on a point that he would want a line to start and release it at the point he would like the line to end. Upon performing this action, the user can observe the line being gradually rendered on the canvas. Similarly, the user can right-click on a point that will be considered as the mid-point of the circle and release it where the imaginary line formed would represent the radius of the circle.

3. Results

We believe that the methods and algorithms chosen provided a satisfactory solution to the problem at hand. As we have seen in *Figure 1*, the canvas created for the simulation is intuitive and easily configurable, which provides a means of simulating different situations. Furthermore, we observed that the gradual render of the lines and arcs look and feel like the real machine, as seen in *Figure 4* and *Figure 5* below.

The simulation of gradual cutting was implemented by using a custom timer which makes the main thread of the program wait between the rendering of each individual pixel. This delay can be programmatically adjusted to more closely represent how different machines would work.

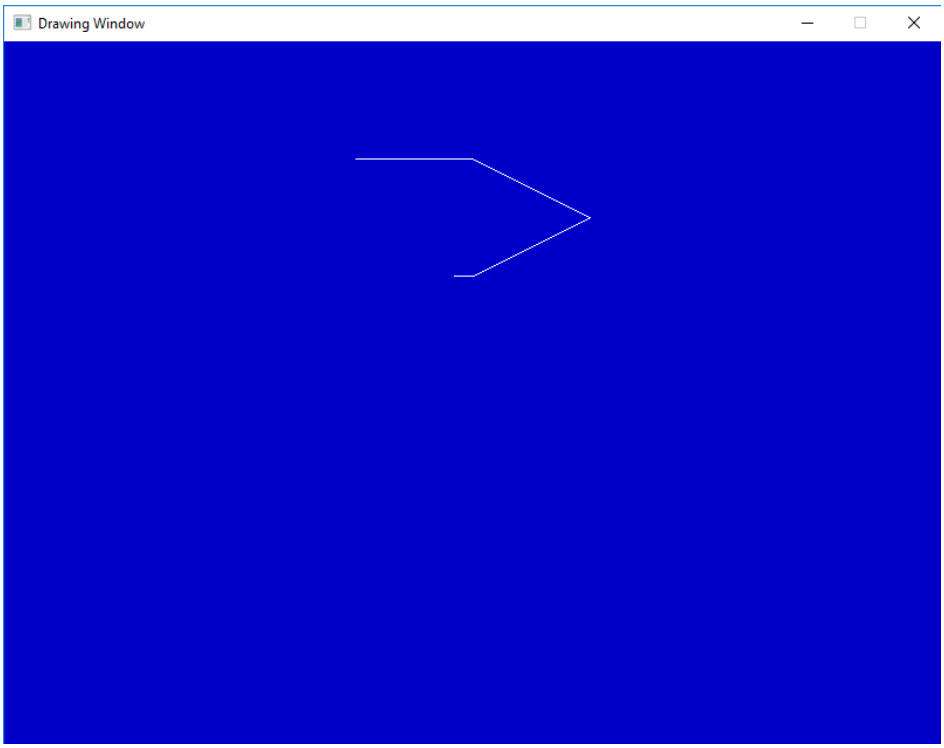

Figure 4, line drawing.

The programs first computes and renders the lines corresponding to the coordinates in the input file and then computes and renders the arcs/circles.

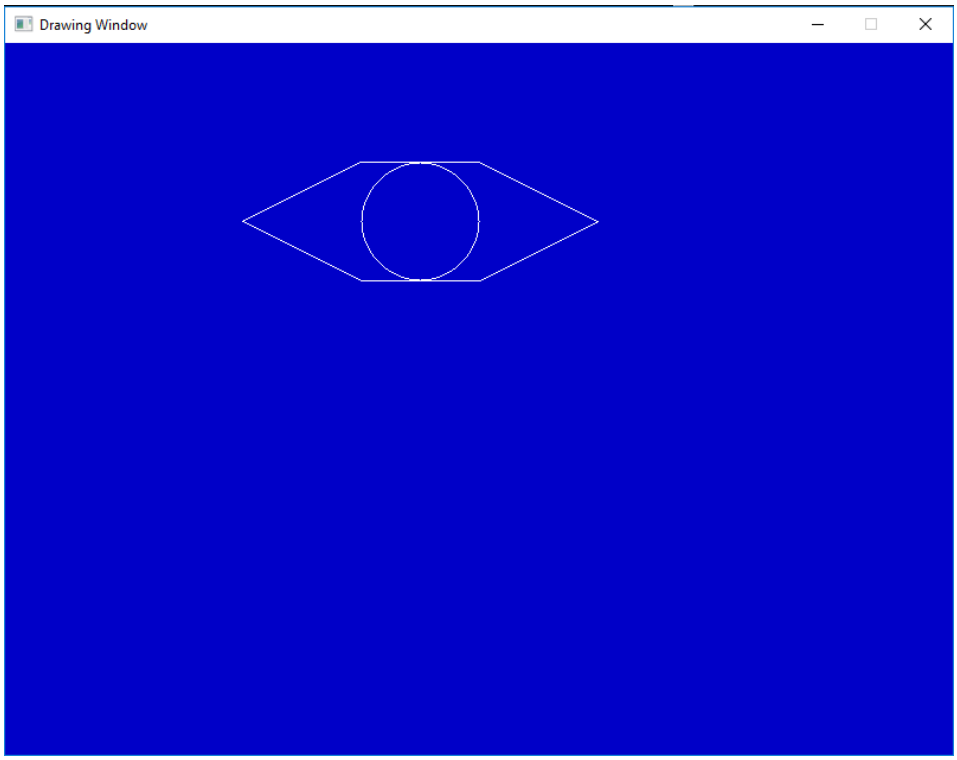

Figure 5, arc drawing.

4. Conclusions

We believe that laser cutting is a staple of society's technological advancement and having laser cutting machines guarantee safety and reliability is a must. Our chosen approach to solving this problem has proved to be ideal and efficient. That being said, a number of improvements can be added.

5. References

Fonseca i Casas, Pau and Pi, Xavier and Casanovas, Josep and Jové, Jordi (2013). "Definition of Virtual Reality Simulation Models Using Specification and Description Language Diagrams". SDL 2013: Model-Driven Dependability Engineering. Lecture Notes in Computer Science. 7916. Springer Berlin Heidelberg. pp. 258–274.

Van Aken, J.R., "An Efficient Ellipse Drawing Algorithm", CG&A, 4(9), September 1984, pp 24-35

Abrash, Michael (1997). Michael Abrash's graphics programming black book. Albany, NY: Coriolis. pp. 654–678.

Donald Hearn; M. Pauline Baker. Computer graphics. Prentice-Hall. ISBN 978-0-13-161530-4.

Perrottet, D et al.,"Heat damage-free Laser-Microjet cutting achieves highest die fracture strength", Photon Processing in Microelectronics and Photonics IV, edited by J. Fieret, et al., Proc. SPIE Vol. 5713 (SPIE, Bellingham, WA, 2005).