2to3D a User-Friendly Parametric CAD Program for Laser Cutting

CS701 Final Project Report

12/19/17

Leo McElroy
Middlebury College
Middlebury, VT
lmcelroy@middlebury.edu

Ben Brown
Middlebury College
Middlebury, VT
bwbrown@middlebury.edu

ABSTRACT

We created a user-friendly computer-aided design (CAD) program for laser cutting which integrates the desired features of vector drawing software and 3D parametric CAD programs. We implemented basic drawing, editing, and transformation tools. The program is capable of outputting SVGs (the Scalable Vector Graphics image format). Geometric constraints are solved using Cassowary.js - an implementation of the Simplex Algorithm. Linear constraints (coincidence, fixing of points, vertical, and horizontal) are efficient and robust. Nonlinear constraints (distance, line angles, parallel, and perpendicular) are not naturally accommodate by the Simplex Algorithm and required clever workarounds to function, consequently they are less robust. The program is accessible by virtue of being open-source and a web application which can run in any modern web-browser. We were successful in creating an accessible and functional parametric design program which simplifies the laser cutting workflow, however the project requires improvement before we foresee widespread adoption by laser cutting hobbyists. Future development of the project would include building out drawing features, adding circle and arc shape primitives, raster image import, utilizing a geometric constraint solver that accommodates nonlinear equations (such as gradient descent), and improving state handling of the program to support undo/redo tools.

Keywords

Laser Cutting; Digital Fabrication; Computer-Aided Design; Geometric Constraint Solver; CS701;

1. INTRODUCTION

In the the last ten years the popularization of 3D printers and makerspaces has caused interest in digital fabricators to blossom. Digital fabricators are machines which use computer controlled tools to produce 3D objects. The digital fabrication workflow begins in a computer-aided design (CAD) program where the user produces electronic schematics. These schematics are then brought into a computer-aided manufacturing (CAM) program where the user creates tool paths for a computer numerical control (CNC) machine, this process outputs instructions for the fabrication machine. Finally these instructions are uploaded to a digital fabricator and the schematic is recreated in the medium worked by the fabrication machine. Depending on the machine this medium can be plastic, paper, cardboard, metal, stone, wood, glass, or even organic material. The digital fabrication workflow is depicted in Figure 1.

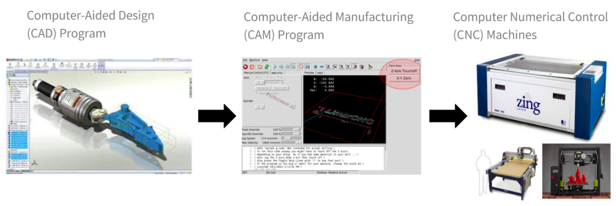
Computer-Aided Design (CAD) Program
Computer-Aided Manufacturing (CAM) Program
Computer Numerical Control (CNC) Machines

Figure 1: The digital fabrication workflow from CAD to CAM to digital fabricator.

Personal digital fabricators, or fabricators intended for use outside of an industrial manufacturing setting (think personal computer versus mainframe), include 3D printers, CNC mills, and laser cutters. Laser cutters are often considered the best balance of accessibility and usefulness of these digital fabricators. A laser cutter works by directing a high powered laser through a focusing head which is attached to a gantry. The gantry controls the position of the head and therefore the location of the laser on a workpiece that rests on a bed underneath the gantry. A top-view schematic of a laser cutter is depicted in Figure 2.

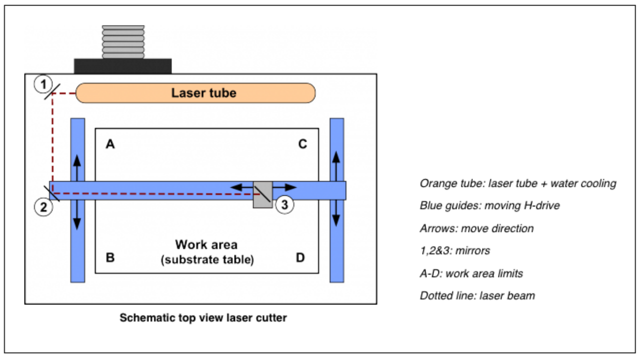
Laser tube
1
2
3
A
C
B
D
Work area (substrate table)
Schematic top view laser cutter
Orange tube: laser tube + water cooling
Blue guides: moving H-drive
Arrows: move direction
1,2&3: mirrors
A-D: work area limits
Dotted line: laser beam

Figure 2: Top-view schematic of a laser cutter adapted from Ref. [1].

Laser cutters are capable of cutting and engraving. Most commercial laser cutters will cut plastic, paper, cardboard, and wood up to a quarter inch thick. They can also engrave all these materials and most types of metal and glass.

There are two main paradigms for modeling in CAD programs - direct modeling and parametric modeling. In direct modeling the user interacts directly with the geometry. This is typically through transformations such as dragging, rotating, or scaling. In parametric modeling the design program utilizes a geometric constraint solver that allows the user to specify constraints and dimensions on geometry. The design is then automatically adjusted to satisfy these constraints when the user modifies geometry directly. Figure 3 depicts the common constraint capabilities of a parametric design program.

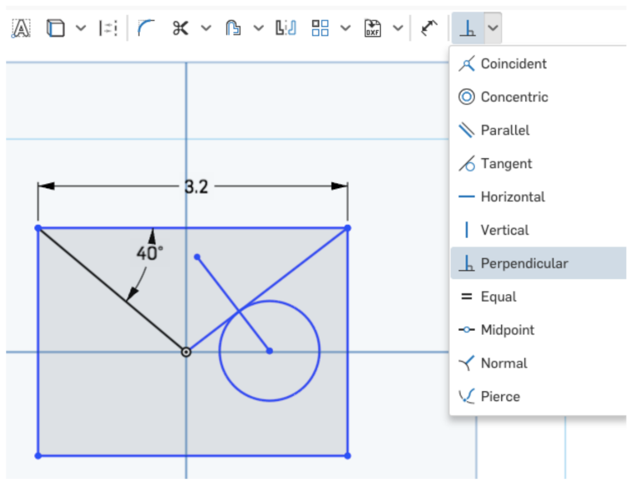

Figure 3: Screenshot of parametric CAD program (Onshape) which shows common geometric constraints.

Laser cutters are typically 2-dimensional fabricators which means there is no z-axis adjustability during fabrication. Consequently, prevalent drawing programs such as Adobe Illustrator and CorelDRAW were adopted for creating designs intended for laser cutting. Vector graphics, a scale-agnostic format where images are represented mathematically, are typically used for generating cutting paths. Rasters, a format where images are represented by pixels, are typically interpreted as engravings when laser cutting.

The issue that motivated our project is that the laser cutting workflow is often made unnecessarily cumbersome by users switching between various design programs. These include parametric CAD programs, with features intended for designing 3D real objects, and high-powered drawing programs, with features intended for creating intricate drawings. Our project aims to unify the essential features for laser cutting found in each of these design softwares into one program.

In Section 2 we present the current issue with the laser cutting workflow and design process. In Section 3 we briefly review programs with some of the features we were interested in incorporating into 2to3D. In Section 4 we describe the basic shape tools and how we handle geometric constraints using Cassowary.js. In Section 5 we cover the rest of the functionality and capabilities of the 2to3D program. And in Section 6 we discuss the success of the project, review the major design principal of 2to3D, and describe extensions to our work which will result in 2to3D becoming a viable substitute for existing programs used for laser cutting.

2. PROBLEM STATEMENT

The problem we attempted to solve is that existing CAD programs are not designed for laser cutting. Laser cutting requires 2D schematics so non-parametric drawing programs are appealing to use but laser cutters produce 3D objects so parametric design capabilities are highly desirable. Parametric design is based on setting constraints and dimensions (the way an engineer would design). This approach facilitates creating real objects. The need for parametric design and basic drawing capabilities results in users switching between a variety of programs which each offer some subset of desired features. Consequently users often find themselves switching back and forth between programs when creating designs. This issue with the current laser cutting workflow is depicted in Figure 4. A user creates the skeleton for a design in a parametric CAD program intended for modeling 3D objects (such as Onshape, Fusion360, or Solidworks). The user then imports this design into a some vector drawing software (Adobe Illustrator or CorelDRAW), then into a program capable of producing tool-paths (in this case CorelDRAW), which are finally sent to the laser cutter.

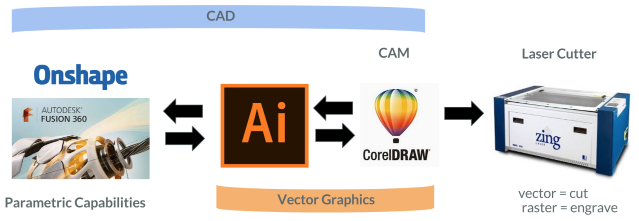

Figure 4: The current laser cutting workflow. Notice the "pong effect" between design programs in the CAD portion of the workflow.

Another issue with the current laser cutting workflow is the price and complexity of commercial CAD software. Be it parametric CAD - Onshape, Fusion360, Solidworks - or visual design software - the Adobe Suite, CorelDRAW - commercial design software often requires a user pay regular subscription fees or retains ownership over user produced data. This makes laser cutting inaccessible to new hobbyists who may be unwilling to pay the fees or concerned with privacy of files they create. Additionally, existing CAD software is over-engineered and inappropriate for a majority of laser cutting tasks. Most parametric CAD programs also incorporate sculpting and constructive solid geometry (CSG) tools which are completely irrelevant to a user interested in laser cutting. Likewise visual design programs involve complex layering systems and style modification tools with no relevance to physical designs. An ideal CAD program for laser cutting would incorporate features of both traditional drawing programs and parametric design programs. Our goal was to create a single-page web-based program specifically designed for laser cutting. A program that combined the best parts of 2D drawing programs and 3D CAD programs, while leaving the unnecessary parts behind. Specifically, we wanted our program to be SVG based so that drawings could be easily sent to a laser cutter, and we wanted to incorporate a geometric constraint solver so that we could easily create real-world objects. The laser cutting workflow we ultimately want to create would involve one program for producing all designs and tool paths. This is depicted in Figure 5.

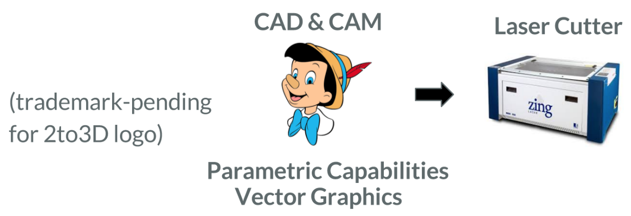

Figure 5: The laser cutting workflow we aim to create.

Semester time constraints meant we did not incorporate CAM or engraving features into 2to3D.

3. RELATED WORK

There are a number of CAD programs created by commercial enterprises and hobbyists that incorporate some of the features valuable for creating laser cutter designs. The goal of 2to3D is to integrate these features into one program. Below we highlight some programs that exemplify notable features or designs:

- **Easel from Inventables**: Easel is a web-based CAD program for CNC milling. It is non-parametric and integrated with CAM, this means it is capable of outputting G-Code. Easel can only be used with CNC milling machines. It is an excellent example of an easy to use CAD program with incorporated CAM.

- **Vectr**: Vectr is a simple vector drawing program capable of running in the browser. The drawing features are very similar to what we plan to implement. Vectr lacks any parametric capabilities.

- **Onshape**: Onshape is a commercial 3D parametric CAD program which is also web-based and integrated with CAM. The issue with Onshape is that users are required to pay to store private files. Additionally, Onshape lacks drawing features which are useful for creating engravings and a majority of the program is packed with features irrelevant to a 2D digital fabrication process.

- **SolveSpace**: A 2D/3D parametric CAD program created by Jonathan Westhues. Solvespace is an impressive recreational project. The most impressive feature is perhaps the robust and efficient geometric constraint solver. In SolveSpace constraints are represented as equations in a symbolic algebra system. Generally, these equations are solved numerically, by a modified Newton's method. If the sketch is not fully constrained, then the Jacobian is solved in a least squares method, meaning each equation is written in such a way as to minimize a useful penalty metric [2] [3]. The issue with SolveSpace is that it requires the user to download and compile the program before use. SolveSpace supports 3D-mesh design which means it incorporates many unnecessary features for a user interested solely in laser cutting.

- **jSketcher**: A parametric 2D and 3D modeler written in pure JavaScript, and developed mostly by Val Erastov. The goal of jSketcher is to produce parametric CAD software for the web. The issue with jSketcher is similar to SolveSpace, because it supports both 2D and 3D modeling it incorporates a myriad of features that obfuscate the essential workflow for laser cutting [4].

2to3D is differentiated from the projects above because it does not incorporate tools for creating or working with 3D meshes meaning it strictly supports 2D designs, it is free and open-source, it is web-based, it allows the user to retain complete control over data produced using the program, and it incorporates a geometric constraint solver to support parametric modeling. Ultimately, 2to3D will also incorporate tools for creating images/importing rasters which can be used for engraving. With this functionality 2to3D will support a truly unique set capabilities among the programs described.

4. METHODS

There were two main parts of 2to3D we needed to implement: the user interface and the geometric constraint solver. For each of these tasks, we employed a different library, with lots of JavaScript code gluing them together.

User Interface with React.js

React.js is an innovative JavaScript library that allows the programmer to specify different parts of the interface as *components*. Each component manages its own state and has its own render function. Whenever a component's state is changed, React automatically re-renders it, effectively abstracting the render loop from the programmer. In this way, a typical React App need only deal with event handling by manipulating state.

2to3D consists of one main DrawArea component, whose state contains all the primitive shape objects, along with the constraints on these objects, so that when the user draws a new shape or modifies an existing one, the interface is re-rendered.

Currently, we have the following shape primitives:

- **Freehand Curves**

- **Lines**

- **Bezier Curves**

Freehand curves are simply SVG paths, but within Line and Bezier objects are points whose values can be manipulated, both by the user and the constraint solver. We discuss this in more detail in the next section, but with these shape primitives we created the following drawing tools:

- **Freehand:** Simply draws a freehand curve following the cursor.

- **Polyline:** Creates lines where adjacent endpoints are constrained to be coincident, optionally allowing the user to close the polyline where it started, creating a polygon.

- **Bezier:** Creates bezier curves whose adjacent endpoints are constrained to be coincident and adjacent control points are constrained to be collinear with the endpoint between them.

- **Rectangle:** Creates four lines in a rectangle where the top and bottom lines are constrained to be horizontal and the left and right lines are constrained to be vertical.

Geometric Constraints with Cassowary.js

In order to design physical objects for laser cutting, we needed our program to implement geometric constraints. For example, we would like to be able to specify two lines as parallel, or a line have fixed length. All of the constraints we wanted to implement can be represented as systems of equations. For example, consider Figure 6 below. We constrain point a to always lie on the circle of radius 1, we fix the line with endpoints a and b to have length $\sqrt{2}$ using the distance equation, and we set the line with endpoints b and c to lie on the x-axis and have length 1.

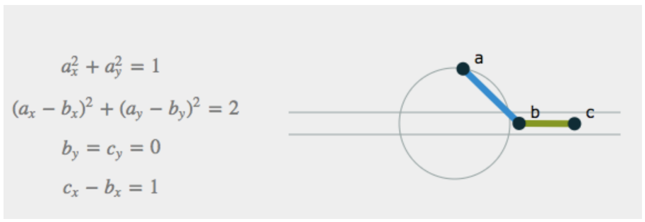

Figure 6: Parametric constraints represented as equations and inequalities; adapted from Ref. [5].

The interface allows the user to freely click-and-drag points around the canvas, which creates the problem of satisfying all the constraints in real-time. To tackle this problem, we found Cassowary.js [6], a JavaScript library that implements a version of the simplex algorithm, which was created to solve *linear programming* problems. A linear program consists of n variables on which we have a linear objective function and a set of linear constraints. For example, the following is a linear program on $n = 4$ variables.

$$\text{Minimize } C = x_1 + 3x_2 - 2x_4 \text{ subject to:}$$

$$x_1 + x_2 \geq 4,$$

$$x_3 - 4x_4 + x_2 = 42, \text{ and}$$

$$4x_1 + x_3 - x_2 \leq 2.$$

Therefore, we can formulate our geometric constraints as a linear program where each x and y value of every point in every shape is a variable and each geometric constraint is transformed into (possibly multiple) linear constraints. In addition to solving simple linear programs, Cassowary.js also allows us to specify non-required constraints of varying *strength*. For example, we can specify that we would like a particular point to stay where it is unless absolutely necessary. Because of this, we do not need to explicitly define an objective function for our linear program.

Some geometric constraints are easily transformed into linear (in)equalities, but others are not. For example, to constrain a line to be horizontal is easy, as we simply specify that the x coordinates of its endpoints must be equal, but constraining a line to be a fixed length is much more difficult. Our solution was to cleverly reform and approximate many geometric constraints using more than one linear equation or inequality.

Below are the geometric constraints we implemented, along with a description of how they were implemented.

- **Fixed:** Constrains a point to stay in the same location. Implemented by setting the x- and y-coordinates to be constants.

- **Coincident:** Constrains two points to lie in the same location. Implemented by setting the x-coordinates to be equal and the y-coordinates to be equal.

- **Horizontal:** Constrains a line to be horizontal. Implemented by setting the x-coordinates of the two endpoints to be equal.

- **Vertical:** Constrains a line to be vertical. Implemented by setting the y-coordinates of the two endpoints to be equal.

- **Parallel:** Constrains two lines to be parallel. This constraint is more difficult to implement, as there is no way to represent it with linear equations. However, we can implement a fixed-slope constraint using linear equations by fixing the ratio between the difference of the x- and y- coordinates. So, when this constraint is initially set, we constrain both lines to have a fixed slope, and whenever an endpoint is moved, we programmatically update the slope of the constraint.

- **Perpendicular:** Constrains two lines to be perpendicular. This constraint is implemented exactly the same as the parallel constraint, except for setting both lines to have the same slope, one is set to have the negative reciprocal slope of the other and is updated programmatically in a similar way.

- **Length:** Constrains a line to be a fixed length. This was the most difficult constraint to implement, as the distance equation is very nonlinear, so instead we must approximate it with many linear inequalities. The *Manhattan distance* is defined as the sum of the absolute differences between the x- and y-coordinates of two points. So, we can estimate a Euclidean distance constraint with multiple Manhattan distance constraints in the following way. If we constrain a point to be within a fixed Manhattan distance of another point, the it will lie within a diamond centered at that point. Then, using linear equations, we can transform our coordinate system by a fixed angle and set another Manhattan distance distance constraint. This constraint will have the diamond rotated by that fixed angle. Enough of these constraints will approximate a circle. However, this only constrains two points to be *within* a fixed distance, so we must create minimum Manhattan distance constraints in a similar way. All of these together allows us to constrain the two endpoints of a line to a fixed distance from each other. This process is illustrated in Figures 7 through 9.

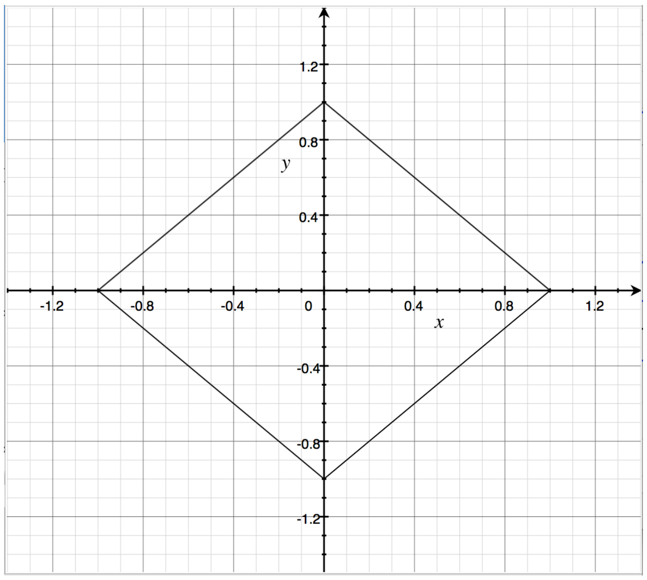

Figure 7: The region constrained by $|x| + |y| \leq 1$.

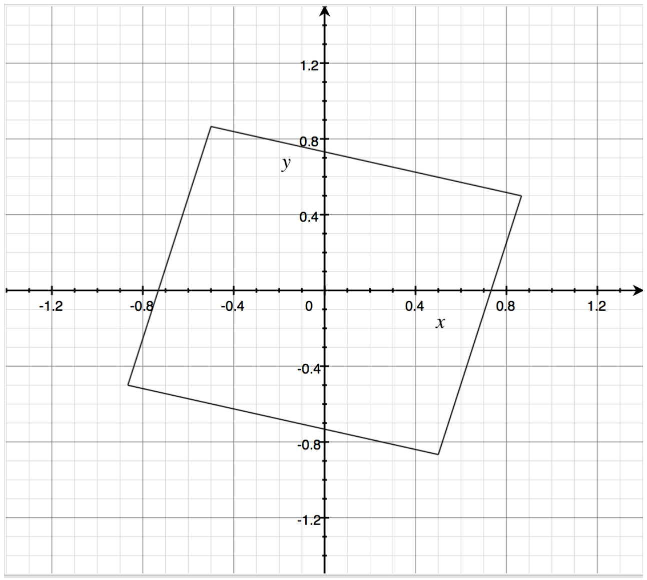

Figure 8: The region constrained by $|x'| + |y'| \leq 1$ where x' and y' are the coordinates of the point in the coordinate system rotated by $30°$.

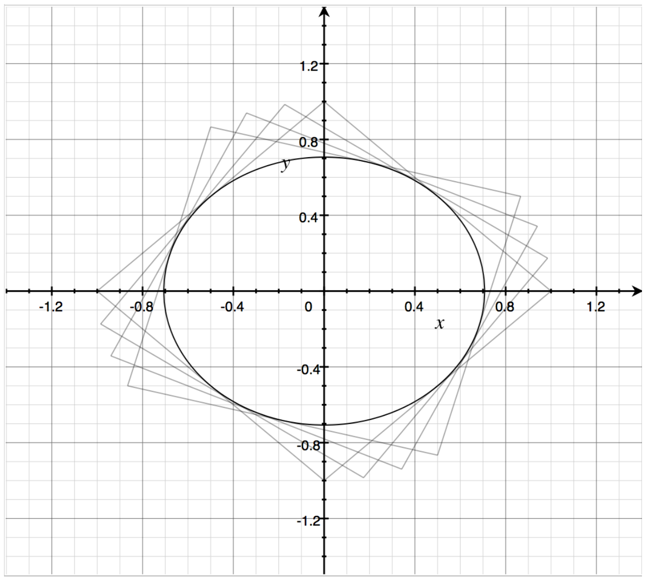

Figure 9: With enough of these regions, we approximate the circle of radius $\frac{\sqrt{2}}{2}$ about the origin.

Direct Transformations

2to3D implements three direct transformations: move, scale, and rotate. These tools allow users to specify geometry which will not be modified by the geometric constraint solver while being transformed. This is done by creating a new set of points at the respective locations of the points that compose the selected shapes. These points are then mapped to new locations through a $x-/y-$ translation function, a rotation function, or a scale function. Finally the original shapes are deleted leaving only the newly transformed shapes.

The transformation function translates all points by the displacement of the mouse from the initial selection point. The rotation function rotates all points around a pivot point (the center of mass) by the angle formed by the line which passes through the initial selection point and the pivot and the line which passes through the current cursor point and the pivot. The scale function works by forming a circle with radius equal to the distance between the initial selection point and the pivot. As the cursor is dragged farther from the pivot outside the circle the selected shapes are scaled up. As the cursor is dragged closer to the pivot inside the circle the selected shapes are scaled down.

Link to GitHub Repository

The full implementation can be found at:
https://github.com/leomcelroy/2to3D.

5. RESULTS

Other Capabilities

We produced a highly accessible CAD program suitable for laser cutting. The accessibility of 2to3D can be attributed to the fact that the program runs entirely in the browser (thus requiring no additional software beyond a modern browser), and that the program strips away all non-essential drawing features and all 3D modeling features. Currently 2to3D only supports the creation of drawings intended for translation into cutting paths. Figure 10 depicts a full digital fabrication workflow using 2to3D. The figure depicts the design of a press-fit box. The laser cutter available for use utilized CorelDRAW for CAM so the drawing was exported from 2to3D as a SVG and imported to CorelDRAW for printing.

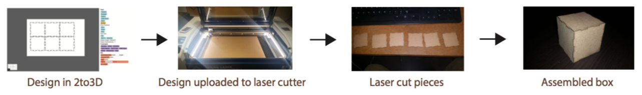

Figure 10: Full digital fabrication workflow using 2to3D. The laser cutter we had access to used CorelDraw for CAM so the drawing was created in 2to3D and imported to CorelDraw just for printing.

2to3D has a simple design that displays all tools to the user without being overwhelming. This design inspired by "lean-operating practices" also immediately and constantly conveys program state information by highlighting the current tool and changing the cursor style. Figures 11 and 12 are screenshots of the program.

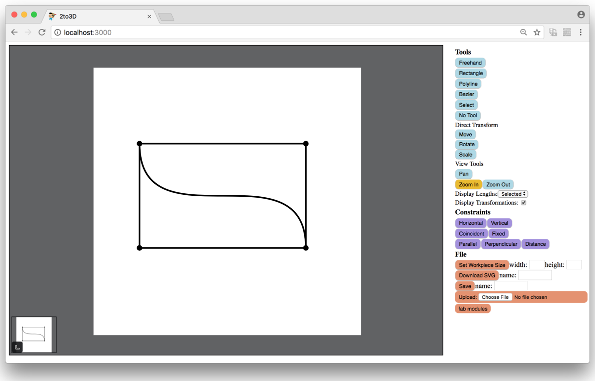

Figure 11: Screenshot of 2to3D program depicting example bezier and rectangle.

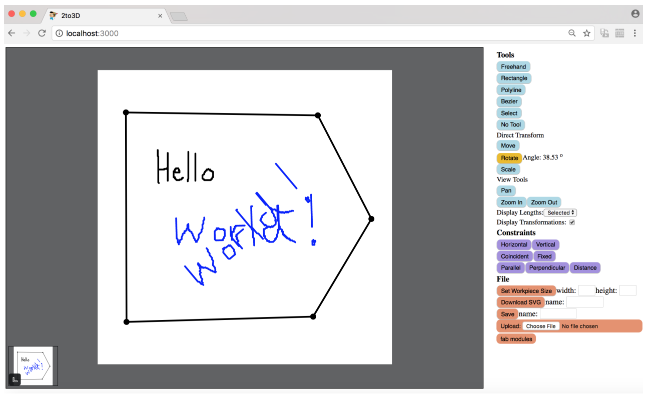

Figure 12: Screenshot of 2to3D program depicting rotation tool in action. Note the angle on display next to the highlighted tool in the right toolbox.

2to3D supports three shape primitives (line, bezier and freehand) which can be used to create more complex shapes. The program supports both linear and non-linear constraints but linear constraints are far more robust. The program also supports direct transformations - rotations, moves, and scales - that break constraints of transformed objects. This allows the user to make direct edits to drawing geometry that will exactly reflect user's intention. In addition to these three aspects which were detailed in Section 4, 2to3D also supports a myriad of other features. The drawing area allows zooming in/out and panning. The workpiece size is adjustable which facilitates creation of designs on any scale. SVG exports draw dimensions from workpiece size which simplifies the CAM process if the user decides to match drawing dimensions to laser cutter bed size. Hotkeys support swift and smooth workflows. Copy and paste capabilities allow users to quickly create redundant geometry. Geometry can be deleted to correct errors. And save and upload features allow users to continue work on unfinished projects or to share designs with others. Because the save feature relies on *JSON.stringify*() the drawing objects must be rehydrated on upload. This means that shape primitives must have their prototype objects reset so they maintain methods. A full list of features can be found in Table 1.

Table 1: Capabilities of 2to3D.

Drawing	**Navigation**
Freehand Rectangle Polyline Bezier	Pan Zoom in/out
Direct Transformations	**Display Options**
Move Rotate Scale	Lengths Transformation values
Constraints	**Other**
Horizontal (line) Vertical (line) Coincident (points) Fixed (points) Parallel (lines) Perpendicular (lines) Length (line)	Click and drag points Download SVG Save drawing Upload drawing Copy/Paste Delete Hotkeys Workpiece size adjustable

Link to Video Demo and Program

A demonstration of the program's features can be found at: https://www.youtube.com/watch?v=QDdJkAFzwLU.

The program itself can be accessed while on Middlebury College's network at:
basin.cs.middlebury.edu:3002.

6. DISCUSSION

We accomplished our main goal of creating a parametric web-based vector drawing program. Among the programs we studied 2to3D is perhaps the most accessible one with parametric capabilities. The program supports basic drawing tools and features including: polylines/polygons, freehand, beziers, rectangles, and direct transformations. Constraints were implemented using Cassowary.js which enabled robust and fast solving of linear constraints. Non-linear constraints had to be represented with abstractions which could be defined linearly. The major design principle guiding our work is that more is not necessarily better. The programs currently used for laser cutting are for the most part highly sophisticated, capable, and powerful. The issue is that they were not designed with laser cutting as an intended use case. 2to3D stands a chance to supplant multi-million dollar programs because it is not intended to be feature packed. Our own limited resources in implementing the program become our greatest advantage. 2to3D only contains features which are useful and relevant to laser cutting, or more generally relevant to 2D digital fabrication. The inclusion of only what are essential features reduces cognitive load on the user which hopefully in conjunction with the ease of accessing the program will make laser cutting a more approachable activity. As with all software engineering, the work is never done. Future work would include:

- Building out shape primitives to include circles and arcs, as well as associated constraints (such as tangent). Introducing these shape primitives would complete the set of drawing capabilities ordinarily found in 2D/3D parametric CAD programs.

- Improvement of the program's state-handling to enhance user experience. This could involve introducing atomic operations which would allow tracking of a file's timeline. Restructuring the program in such a manner would allow more comprehensive saving of files without the current issues of tracking constraints. It would also enable the creation of undo and redo features which greatly enhance usability.

- Perhaps the most important improvement to be made in the next iteration of 2to3D is the incorporation of a geometric constraint solver capable of handling nonlinear constraints without abstraction. Ideally this would entail creating a JavaScript library that can efficiently solve systems of nonlinear equations using gradient descent or automatic differentiation. Using one of these methods allows the solver to find solutions which are close to the user specified geometry, thereby enhancing the programs intuitiveness and usability.

- Users interested in fully encapsulating the CAD/CAM workflow within open-source web-based environments can use 2to3D in conjunction with MIT's Center for Bits and Atoms' "fab modules." We included a link to "fab modules" within 2to3D's toolbox. A user only must download their 2to3D design as an SVG and upload it into the "fab modules." This process could be improved by integrating the two programs into a single environment.

- Improve GUI and drawing features. This encompasses a variety of changes which include allowing the user to specify transformation values when conducting direct transformations, adding more drawing tools which can be interpreted as engravings, and/or incorporating a raster image import feature.

One other possible extension is the introduction of a backend server and database for storing files. This could enhance the program by allowing users to store projects in the cloud but also introduces challenges to scaling up to wide adoption of 2to3D. By its current design 2to3D only demands local computational resources. Consequently it requires minimal involvement from administrators and puts users in total control of their data.

Incorporating the bulleted extensions into 2to3D would result in a program that can be widely adopted by laser cutting hobbyists and enthusiasts.

7. ACKNOWLEDGMENTS

Thanks to Professor Grant for teaching our seminar course and thanks to the folks at Maker Works (in Ann Arbor, Michigan) for first teaching Leo how to laser cut (among so many other teachings).

8. REFERENCES

[1] "How does a laser cutter work," July 2012. [Online]. Available: redmine.laoslaser.org/projects/laos/wiki/How_does_a_laser_cutter_work

[2] J. Westhues, "solvespace," 2016. [Online]. Available: https://github.com/solvespace/solvespace

[3] A. Prokoudine, "Solvespace 2d/3d cad software released under terms of gpl," Aug 2013. [Online]. Available: libregraphicsworld.org/blog/entry/solvespace-released-under-gpl

[4] V. Erastov, "jsketcher," 2017, available at https://github.com/xibyte/jsketcher, GitHub.

[5] M. Keeter, "Constraint solver." [Online]. Available: https://www.mattkeeter.com/projects/constraints/

[6] G. J. Badros and P. J. Stuckey, "The cassowary linear arithmetic constraint solving algorithm," in *ACM Transactions on Computer-Human Interaction 8*, Dec 2001, pp. 267 – 306.