RWTH AACHEN UNIVERSITY

VisiCut: An Application Genre for Lasercutting in Personal Fabrication

Bachelor's Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Thomas Oster

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Univ.-Prof. Dipl.-Ing. M. Arch Peter Russell

Registration date: Sep 2nd, 2011
Submission date: Sep 30th, 2011

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, September 2011
Thomas Oster

Contents

List of Figures

List of Tables

Abstract

Laser-cutters are central devices in personal fabrication and widely used in fab labs. Since those are open for everyone, an easy to use software solution is important. There are many vendor supplied solutions and some open source approaches, but none of them can provide full platform independence and good usability.
We created a tool named *VisiCut*, which allows using laser-cutters from nearly any operating system and even prepare the laser-jobs at home to minimize the time needed in the lab. This tool has some advantages over the existing solutions, which include positioning directly on a live camera picture of the material, detailed preview rendering and saving of complete jobs for easy distribution and portability.
As base for VisiCut, we created a new library dedicated for laser-cutting, which is called *LibLaserCut*. This library provides easy interfaces for implementing laser-cutter drivers. The first driver, we implemented for the library, controls an Epilog ZING laser-cutter.
Both are written in pure Java, which make them platform independent. They are licensed as free software in order to be available for everyone and to allow continuous development and community support.
For VisiCut, we analyzed existing interactive systems used in fabrication environments and conducted a survey to to determine the habits of people using the laser-cutter in our fab lab. We also created a few UI prototypes and did user tests to improve them.
For the LibLaserCut, we analyzed the possibilities of the Epilog ZING laser-cutter among with available software solutions. We designed an interface for laser-cutter drivers and implemented the driver for the Epilog ZING, which is based on the open source driver *CUPS-epilog*.
Further, we evaluated VisiCut through a user test and tested the whole system on different platforms. We also created a list of improvements and enhancements, which should be addressed in future developement.

Überblick

Lasercutter sind zentrale Geräte in der Personal Fabrication und sehr verbreitet in FabLabs. Da diese offen für jedermann sind, ist einfach zu bedinende Software sehr wichtig. Es gibt viele vom Hersteller bereitgestellte Lösungen und einige Open Source Ansätze, jedoch bietet keine davon volle Plattform Unabhängigkeit und eine gute Bedienbarkeit.
Wir haben ein Tool namens VisiCut erstellt, welches ermöglicht, Lasercutter von fast jedem Betriebssystem zu benutzen und sogar Projekte zu Hause vorzubereiten um die benötigte Zeit im Lab zu minimieren. Dieses Tool hat einige Vorzüge gegenüber den existierenden Lösungen, wie zum Beispiel das Positionieren direkt auf einer Kamera Vorschau des Materials, detailiertes Vorschau Rendering und das Speichern von kompletten Projekten für einfache Verteilung und Mobilität.
Als Grundlage für VisiCut haben wir eine neue Bibliothek namens *LibLaserCut* erstellt, welche auf Lasercutting spezialisiert ist. Diese Bibliothek stellt einfache Schnittstellen bereit um Treiber für Lasercutter zu implementieren. Der erste Treiber, den wir implementiert haben, steuert einen Epilog ZING Lasercutter.
Sowohl VisiCut, als auch LibLaserCut sind in reinem Java geschrieben, was sie plattform unabhängig macht. Sie sind als freie Software lizensiert um für jedermann verfügbar zu sein und kontinuierliche Entwicklung und Unterstützung durch eine Gemeinschaft zu ermöglichen.
Für VisiCut haben wir bestehende interaktive Systeme, die in Fabrication Umgebungen genutzt werden analysiert und eine Umfrage durchgeführt um die Gewohnheiten von Menschen, die den Lasercutter in unserem FabLab benutzen festzustellen. Des weiteren haben wir einige UI Prototypen erstellt und Tests durchgeführt um diese zu verbessern.
Für die LibLaserCut haben wir die Möglichkeiten unseres Epilog ZING Lasercutters zusammen mit verfügbaren Software Lösungen analysiert. Wir haben eine Schnittstelle für Lasercutter entworfen und einen Treiber für den Epilog ZING implementiert, der auf dem Open Source Treiber *CUPS-epilog* basiert.
Außerdem haben wir VisiCut mit Hilfe eines Benutzertests evaluiert und das gesamte System auf verschiedenen Plattformen getestet. Des Weiteren haben wir eine Liste mit Erweiterungs- und Verbesserungsmöglichkeiten erstellt, welche in zukünftigen Entwicklungen berücksichtigt werden sollte.

Acknowledgements

First of all, I want to thank Prof. Jan Borchers, Univ.-Prof. Dipl.-Ing. M. Arch Peter Russell, René Bohne and the whole chair i10 for giving me the opportunity to write this thesis and experiment with all the expensive equipment.
Special thanks to Clio Kakoulli, Birgit Stolte, Mariana Bocoi and all the others who constantly helped me by testing my prototypes, mocking me for bad UI Design and seeing things from a completely different point of view.
Also, I want to thank all members of the "Thesis Research Camp" for all the suggestions and one more night of work instead of sleep.
Finally I want to thank my girlfriend Manuela for being patient with me in the last weeks and for giving me the hint to the idea with the camera.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in American English.

Download links are set off in coloured boxes.

File: myFile[a]

[a] file_number.file

Chapter 1

Introduction

"The real technical problems came because people working on the project didn't really follow my proposal at all, but set out to do other things instead of making a laser."

—Gordon Gould

Fab labs are places, where everyone can use expensive digital fabrication machines like 3D-printers, CNC-milling machines and laser-cutters in order to create "almost anything" they can imagine. In his book, Neil Gershenfeld defines the term *fab lab*, as a "lab for fabrication", which is basically "a collection of commercially available machines and parts linked by software and processes we developed for making things" [Gershenfeld, 2007].

In this chapter, we will explain, what a laser-cutter is and what it is capable of. We will describe our motivation to develop new software for it and finally, we will give a short overview, what structure the rest of this thesis is.

1.1 What is a Laser-Cutter and What is it Good for?

A laser-cutter allows fast and high quality cutting and engraving of a wide range of materials. Basically, it is a machine, which directs a high-power laser beam on material to be cut or engraved. "The material then either melts, burns, vaporizes away, or is blown away by a jet of gas" [Oberg et al., 2004, p. 1447], leaving an edge with a high-quality surface finish.

In the fab lab Aachen[1] , we have an Epilog ZING laser-cutter, which directs the output of its CO_2-laser tube though a construction of mirrors (see figure 1.1). There are

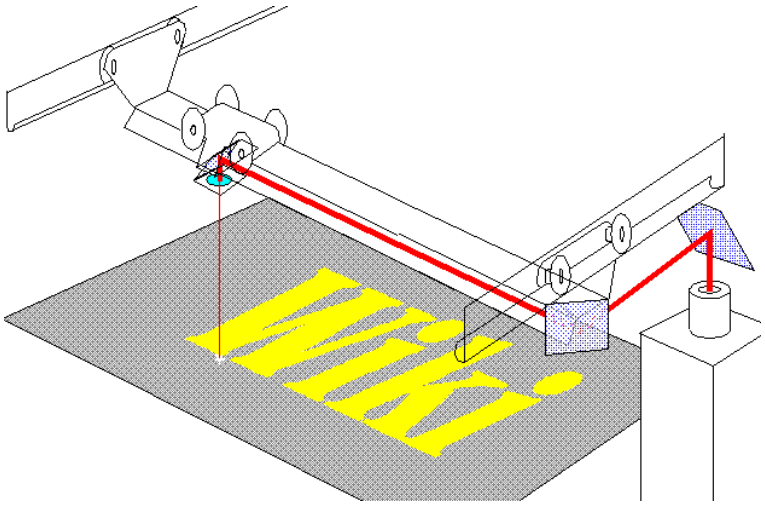

Figure 1.1: A laser-cutter, which directs its beam through mirrors
source: http://en.wikipedia.org/wiki/Laser_engraving

other types of laser-cutters (e.g. $Yb : Fiber$, $Nd : YVO4$ or $Nd : YAG$ lasers), which can used for different materials, but in this thesis we will restrict our research to the CO_2 laser-cutter, more precisely the Epilog ZING.

[1] http://fablab-aachen.de

1.1.1 Modes of Operation

Our laser-cutter has three different modes of operation[2], which we will explain in this section.

Vector Mode

In vector mode, the laser moves along a path with constant speed, allowing to cut material or engrave lines, depending on the power and the movement speed of the laser beam (see figure 1.2).

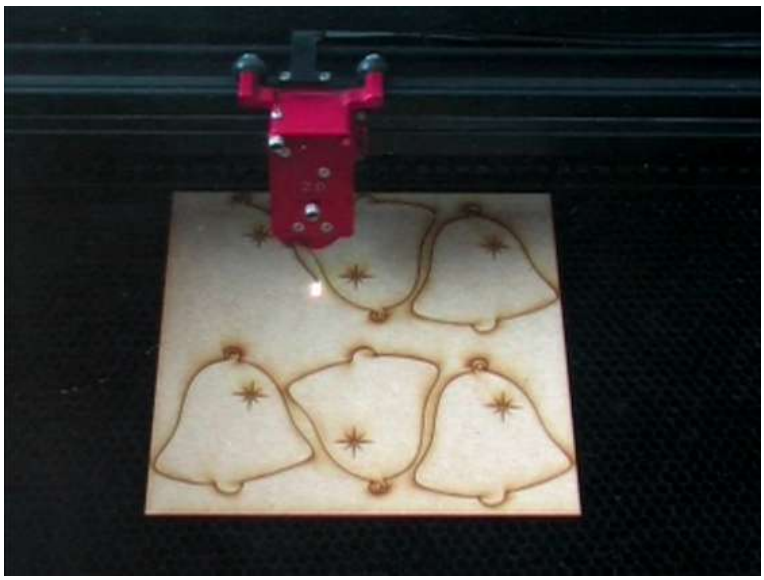

Figure 1.2: A laser-cutter cutting out bells in vector mode
source: http://www.instructables.com/image/FXXQZ80FABDYR2R/Laser-cutting.jpg

[2]There is also the *Stamp Mode*, but since this is not really different from raster modes and not widely used, we will not regard it in this thesis

Raster Mode

In raster mode, the laser moves line-wise along a raster and engraves a representation of an image by switching the laser on and of according to the raster pattern (see figure 1.3).

Figure 1.3: A letter engraved in wood with raster mode
source: http://www.imajeenyus.com

3D-raster Mode

3D-raster mode is different from raster mode in the way, that the laser is not only switched on and of, but set to different power levels, resulting in different engraving depths. This allows three dimensional effects as shown in figure 1.4.

Figure 1.4: 3D effects using 3D raster mode
(with friendly permission from
http://www.thunderlaser.com)

1.1.2 The Positioning Problem

A common problem when working with the laser-cutter is, that due to the corner- or center-based positioning available in the current drivers, it is sometimes hard, to engrave on real-world objects (which already have certain shapes or engravings), because the target position of the engraving is not always corner or center based and the resulting size is not easy to imagine. This can be very annoying or even expensive (see figure 1.5).

Figure 1.5: An expensive mistake: The engraving is not where it should be

1.2 Motivation

Gershenfeld writes, that in order to get closer to a real "Personal Fabricator", "the intention over time is, to replace parts of the fab lab with parts made in the fab lab " [Gershenfeld, 2007, p. 215].

There are several approaches to do this for common fab lab tools like 3D-Printers (e.g. the MakerBot[3]), Milling machines (e.g. the Mantis CNC Mill[4]) and even laser-cutters (e.g. the boot strappable open laser cutter[5]).

Since the hardware is "linked by software" and the available software is often limited or expensive (see 2—"Related work"), the aim of this thesis is to replace *software* in the fab lab by *software* made in the fab lab.

[3] http://www.makerbot.com/
[4] http://makeyourbot.org/mantis9-1
[5] http://builders.reprap.org/2011/05/boot-strappable-open-laser-cutter.html

1.2.1 VisiCut: A Tool to Simplify Laser-Cutting

Different users prefer different software to create their models for laser-cutting (see A.5), which often makes it necessary to import files created in another graphic software. This step can be time consuming, because most graphic software is not completely compatible to each other and suitable export and import settings have to be determined. Sometimes even corrections have to be made. Performing this task on the target computer takes valuable lab time.

Importing into other graphic software takes valuable time in the lab

Thus we created the tool VisiCut, which allows the user to import and prepare a laser-job at home, taking without time limitations, and then saving it in the VisiCut Portable Laser Format (see 3.4.4—"The Portable Laser Format"). The saved file can be opened the VisiCut application on another computer, e.g. one in the lab, and be sent to a laser-cutter.

Additionally, VisiCut provides a good visual preview and solves 1.1.2—"The Positioning Problem" by providing camera supported visual positioning. Also the Portable Laser Format allows easy publishing of prepared laser-jobs e.g. via thingiverse[6] .

1.2.2 An Open Source Laser-Cutter Library

For enabling VisiCut to control a laser-cutter, it needs some kind of driver. Since most vendors supply only drivers for a specific operating system (OS), which are sometimes optimized to work with certain graphic software, there are often licensing issues and compatibility problems. There are some open source projects, but as we will outline in section 2.2—"Existing Software for the Epilog ZING", they all have some drawbacks. Since there is no special API for laser-cutting in any of the major OS, most drivers are written as printer drivers. This has certain limitations (see section 2.2.4—"Limitations of the Printer Driver Approach") because a laser-cutter is in many ways different from a printer.

Major OS lack a laser-cutter specific API

[6] http://www.thingiverse.com

There is a need for an open source library, dedicated to laser-cutting

There is, therefore, a need for a library, which provides an API dedicated for laser-cutting and allows easy implementation of new drivers, which can be used with any application using that library. In order to be able to support a wide range of different devices and to become a widely used library, it should be open source, because "by making the users of a product into codevelopers, you speed debugging, improve quality and gain specialized features, that may eventually turn out to be important to a wider audience" [O'Reilly, 1999].

1.3 Overview

In the following, we give an overview of the chapters in this thesis and their contents.

- The chapter 2—"Related work" introduces some interactive systems, with similar concepts to the ones in VisiCut. After that, the different existing drivers for the Epilog ZING laser-cutter are listed compared to each other.
- The chapter 3—"Own work" starts with an initial survey (see A—"User Survey to Determine Habits in Laser-Job Creation") to determine the habits of users, who use the laser-cutter. A declaration of requirements for our system follows, which we derived from the previous chapter and the user survey. The next section contains the overall 3.3—"Architecture" of the system. The rest of the chapter is split into two sections:
  - First, the section 3.4—"VisiCut" presents the developement process of the *VisiCut* tool by showing the ceveral prototypes and some implementation details
  - The following section 3.5—"LibLaserCut" we will describe the design and implementation of the *LibLaserCut* library and the implementation of the Epilog driver (see section **??**—"**??**"epilogdriver.

- In 4—"Evaluation" we will discuss if the requirements from 3.2—"Requirements" are met and we will measure the overall usability with the 4.2—"System Usability Scale".
- 5—"Summary and future work" contains a summary of our work and some ideas how to improve it.

Chapter 2

Related work

"Your work is to discover your world and then with all your heart give yourself to it."

—Hindu Prince Gautama Siddharta, the founder of Buddhism, 563-483 B.C.

The related work can be grouped in two categories. First, we will introduce some interactive systems, which contain concepts important to the development of VisiCut. In the second part, we will give an overview of the existing software for controlling an Epilog ZING laser-cutter.

2.1 Interactive Systems with Real-World Input

2.1.1 CopyCAD

The CopyCAD System, introduced by [Follmer et al., 2010] allows to extract shapes of physical objects by scanning a picture, interactive editing of the shapes directly on the object and reproducing an object with the manipulated shapes on a milling machine (see figure 2.1).

This approach enables users to use real world objects as a

base for their own ideas. Since the manipulating is done directly on the milling machine, no computer (except for controlling the CopyCad System) is necessary. On the other hand, drawing with a pen is not very accurate and on a computer one could additionally use shapes from digital files.

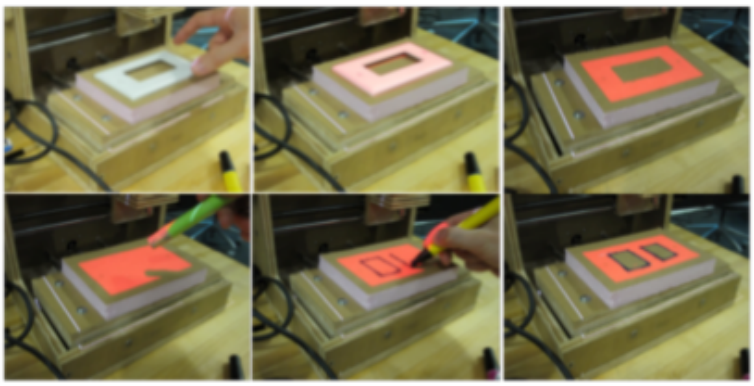

Figure 2.1: CopyCAD system interaction for modifying a lightswitch cover. Clockwise starting in top left: 1. Place object, 2. copy shape, 3. shape is projected, 4. delete interior shape, 5. draw new geometry, 6. copy new geometry. source:[Follmer et al., 2010]

In our work, we use a camera to get a preview of the material (3.4.3—"The WYSIWYG Part"), but instead of editing directly on the material, we use a computer for editing, which also removes the need of an expensive projector.

2.1.2 Pictionaire

Pictionaire is an interactive tabletop collaboration system (see figure 2.2) introduced by [Hartmann et al., 2010]. It "offers capture, retrieval, annotation, and collection of visual material". For the setup, it uses a tabletop system with touch and device input, a projector to project an image onto the tabletop and a high-resolution camera to capture the whole device-surface (see figure 2.3). This enables the users to annotate directly on captured images of real-world objects (see 2.4).

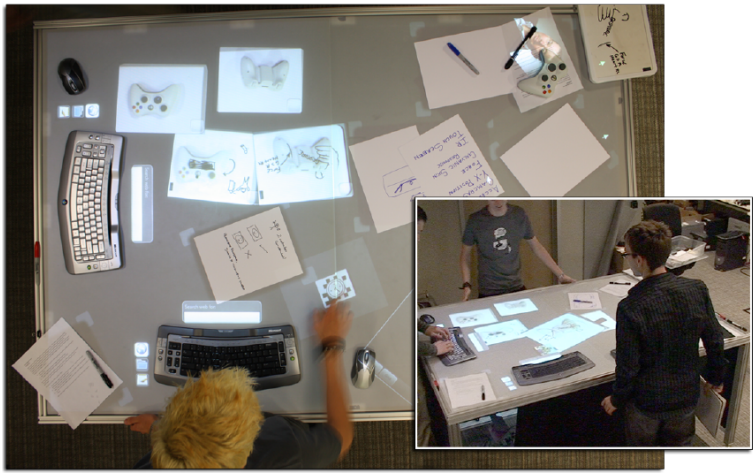

Figure 2.2: Pictionaire supports design collaboration across physical and digital artifacts.
source: [Hartmann et al., 2010]

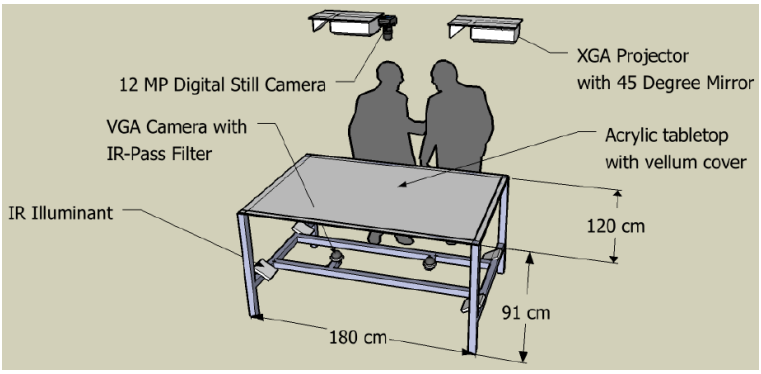

Figure 2.3: The Pictionaire table offers touch and device input, top projection, and high-resolution image capture
source: [Hartmann et al., 2010]

In VisiCut, we will use a similar approach of capturing photos of existing materials and overlay them with virtual graphic content, but in contrast to Pictionaire, we use a computer for editing, instead of a tabletop system.

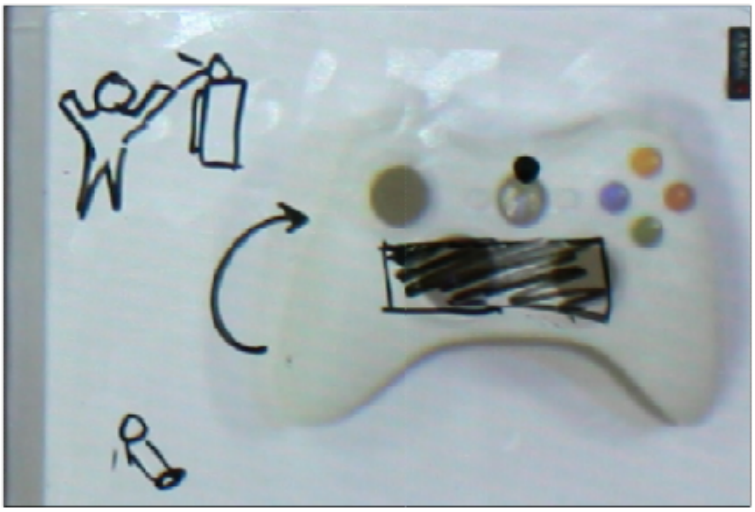

Figure 2.4: Annotations on a photo of a game controller. source: [Hartmann et al., 2010]

2.1.3 Summary

The two presented systems show, that a camera can be used to create an environment which mixes objects from the physical world and the virtual world. However in our environment, most users create their models on a computer before working with the software, it is not necessary to be able to modify the model on the material using multi touch or similar input devices. Instead, we import the captured image into our software an let the user virtually edit it. This has the advantage, that no projector is needed, which makes the design cheaper and the setup less complex.

2.2 Existing Software for the Epilog ZING

In this section, we will compare the different existing software capable of controlling an Epilog ZING laser-cutter to each other. After that, we will outline some drawbacks of the printer driver approach. At the end of this section, we will summarize the differences in order derive requirements for our system in section 3.2—"Requirements".

2.2.1 Epilogs Dashboard

Epilog provides the Laser Dashboard™Print Driver[1] , a printer driver for Microsoft Windows, which allows laser-cutting from generally every windows application, that can use a printer. It can use the laser-cutter either as a local USB printer or as a network printer through the Ethernet interface. The settings specific to the laser-cutter are entered in the dashboard window (figure 2.5), which opens as a printer configuration dialog.

The Epilog Dashboard works as printer driver for Windows

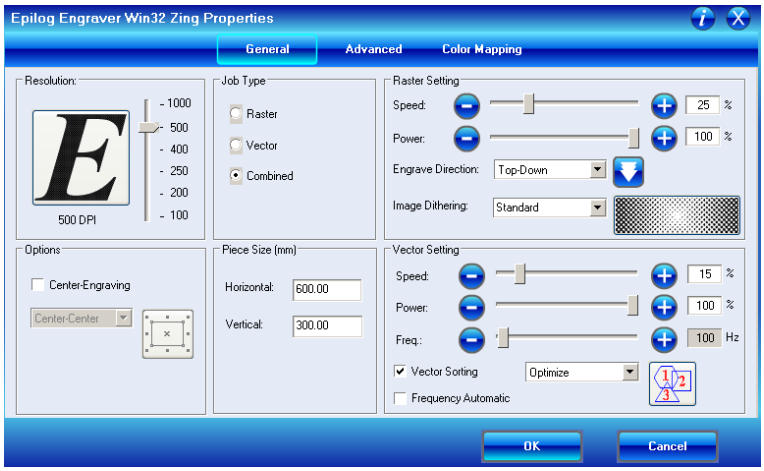

Figure 2.5: Settings dialog of the Epilog Dashboard™

The driver can set parameters like power, speed and frequency per color or for the whole job, but only lines thinner than a dpi-dependent threshold are recognized to be cut. Everything else will be dithered by the selected algorithm (Standard, Brighten, LowRes, Floyd Steinberg, Jarvis and Stucki) and added to the engraving part. The settings can also be saved to a file, which allows the creation of a database for often used materials.

Parameters for certain materials can be saved to a file

The driver also supports a 3D-engraving mode, where the grey value of every pixel is translated into laser power, allowing different engraving depths and 3D effects.

[1] http://www.epiloglaser.com/downloads_zing.htm

Positioning

The user can select between absolute positioning, meaning the upper left corner of the graphic is mapped to the upper left corner of the laser-bed, or select from predefined centered starting points (see figure2.6). It is possible to manually move the laser-head and with it the reference point, before starting a job, which allows accurate positioning on existing objects. However, this has some limitations which are outlined in 1.1.2—"The Positioning Problem".

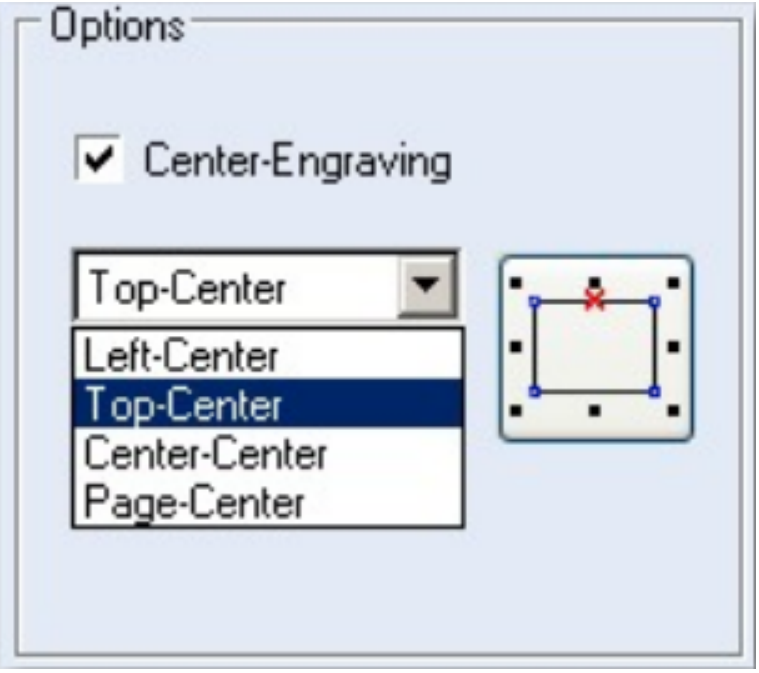

Figure 2.6: Positioning options in the Epilog Dashboard

Preview

No laser-cutter specific preview is provided

The dashboard window does not provide any preview. The only preview provided is just the default printer preview of the operating system's printer dialog, containing the printed part as rastered image. Neither a difference between cutting and engraving is visible, nor any preview how the graphics look after dithering.

2.2.2 CUPS-Epilog

The CUPS-Epilog[2] driver is a back-end for the open source printing system CUPS.

CUPS:
The Common Unix Printing System[3] is developed by Apple Inc.[4] for Mac OS X and other UNIX-like OS. For more information, see [Miller et al., 2009]

Definition: *CUPS*

The CUPS-epilog driver is an open source driver, maintained by the AS220 Labs since 2008. It allows using the Epilog Legend laser-cutter as printer in CUPS compatible environments, like many Linux and Mac OS versions. It consists of a single C file, which can be compiled to a binary which can be registered as CUPS back-end. The driver is capable of controlling the laser-cutter via Ethernet.

CUPS-epilog is a Linux and Mac OS compatible driver which can use Ethernet

Since the driver comes without a PPD file, the laser specific settings cannot be set in a configuration dialog, but they have to be specified on the printer queue.

POSTSCRIPT PRINTER DESCRIPTION:
A PostScript Printer Description file contains a description of the capabilities of a printer. This information allows the operating system to provide a configuration dialog, which contains settings specific to the printer described by a PPD file.

Definition: *postscript printer description*

This means, for every configuration, the user has to create a new printer queue or alter the existing one. This results in most users defining one printer queue per material.

[2] http://www.as220.org/git/cgit.cgi/attic/cups-epilog.git/
[3] http://www.cups.org/
[4] http://www.apple.com/

How the Driver works

Since we used this driver as a base for implementing our own driver (see 3.5.2—"Implementation: The Epilog Driver"), we analyzed the way its different routines work together.

When printing to a queue using this back-end, the executable is called by CUPS and provided with the print job in PDF or PostScript, depending on the CUPS version and preferences. Once started, the driver checks whether the

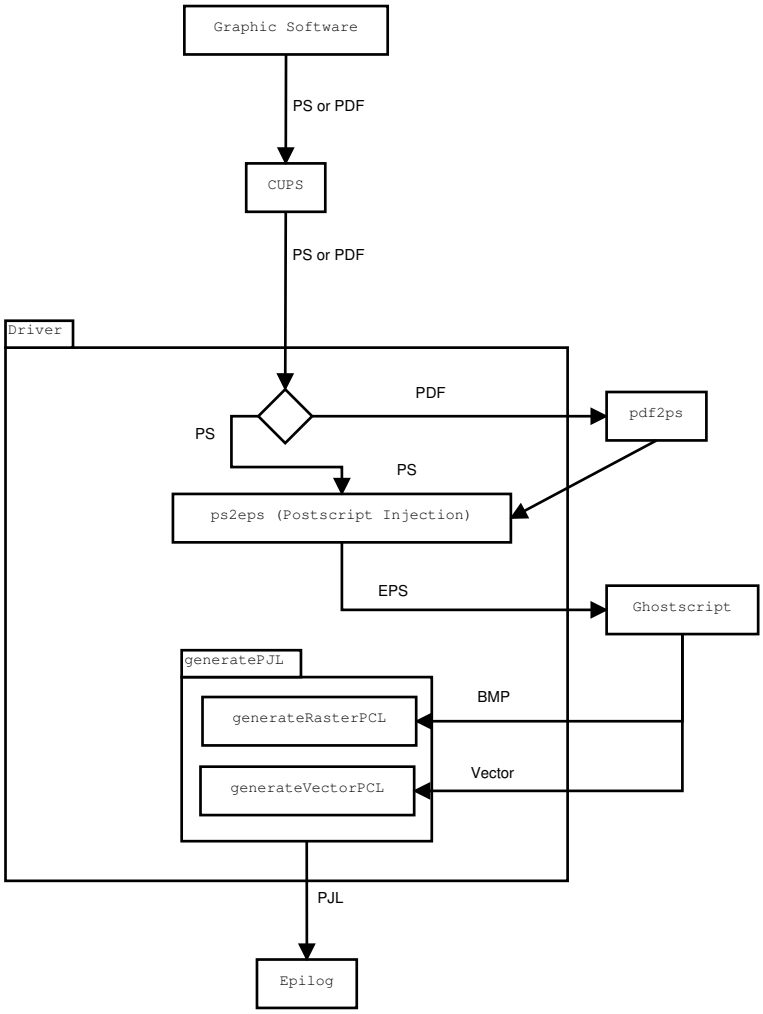

Figure 2.7: Data flow in the cups-epilog driver

given file is PDF or PostScript. If it is a PDF-file, the external tool "pdf2ps" is called to convert the file to PostScript. The important part is the routine "ps2eps" which copies all the Postscript code and adds a codeblock, which overloads Postscript's line-function. So every time the postscript-interpreter shall draw a line, the overloaded-line function is called which checks if the linecolor is red. If this is the case, it writes the line parameters to stdout, which get captured later into the .vector-file. Otherwise it calls Postscript's native line-function (see figure 2.7).

cups-epilog injects a PostScript function, which makes the Interpreter "Ghostscript" yield the VectorData for every red line, while rasterizing the print job

This modified PostScript code is handed over to ghostscript, the postscript-interpreter. Ghostscript interprets the code and creates a rasterized bitmap file. The overloaded line function dumps all red lines to stdout, which is saved in a .vector file. After ghostscript is finished, cups-epilog generates two PCL Blocks, one for the raster part from the bitmap file, the other one for the vector part, from the vector file.

PCL,PJL:
The Printer Control Language (PCL) was originally developed by HP to control laser-printers. It is often encapsulated in the Printer Job Language (PJL) which allows job management and multiple control languages in print jobs. For more information see History of Printer Command Language PCL[5] or [Smith, 1998]

Definition: *PCL,PJL*

The two PCL Blocks get embedded in a PJL block, which is sent via LPD-Protocol as specified by [McLaughlin, 1990] to the Epilog laser-cutter over Ethernet. The driver does not support the USB interface at this time.

2.2.3 Ctrl-Cut

Ctrl-Cut is similar to CUPS-epilog, but provides a printer control panel

Ctrl-Cut[6] , like the cups-epilog, is another CUPS back-end. The code, however, is written in C++ and has a modular architecture instead of a monolithic code block. It also provides a printer control panel 2.8 which makes the config-

[5] http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?objectID=bpl04568
[6] https://github.com/Metalab/ctrl-cut

uration of parameters easy and allows different settings to be specified per print job.

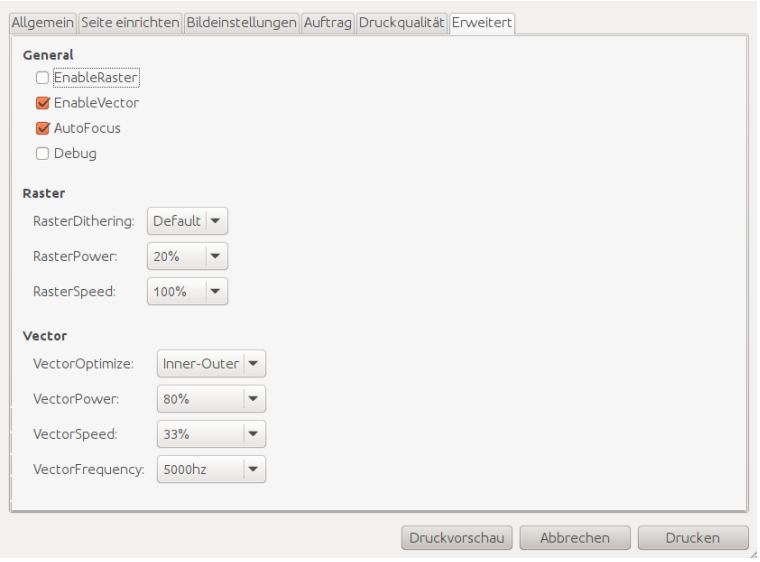

Figure 2.8: The printer configuration dialog of Ctrl-Cut

Currently, it only supports the *Epilog Legend 36EXT*, but future support for other laser-cutters is planned (see README.md[7]). Like the CUPS-epilog driver, it only supports Ethernet connections to the laser-cutter. Also special packaging for Mac OS X and Ubuntu Linux is provided.

2.2.4 Limitations of the Printer Driver Approach

Implementing a laser-cutter driver as a printer driver has the advantage of the existing infrastructure for printers. Nearly any graphic related software is capable of using a printer through the operating system's interfaces.

On the other hand, installing such a driver requires administrative privileges on most operating systems and is a bigger security threat than a user space program. Also most printing systems (e.g. CUPS, which is used in Mac OS X and many Linux versions) only provide the printer driver with file formats like PostScript or PDF, which lack some

[7] https://github.com/Metalab/ctrl-cut/blob/master/README.md

advanced features of other graphic file formats, like layers and groups. Since all files sent to a printer driver get converted into such format, laser-cutter drivers implemented as printer drivers have no access to this information anymore, meaning it would be very hard or even impossible, to create a printer driver for a laser-cutter, which automatically cuts everything in layers named "cut" and engraves everything else.

Printer drivers need administrative privileges, can not take advantage of high-level concepts like layers and are often platform dependant

A third disadvantage of printer drivers is the platform dependence. Although the CUPS system is used on Mac OS X and Linux and can also be installed in Windows, it is not available by default everywhere. There are some operating systems (e.g. Android), which do not even provide support for printers.

2.2.5 Summary of the different drivers

Name	Laser Dashboard™	CUPS-epilog	Ctrl-Cut
OS	Windows	UNIX, Mac OS	UNIX, Mac OS
Supported Modes	vector, raster, 3D raster	vector, 3D raster	vector, raster
Differentiation	line width	line color+width	line color+width
Settings	printer configuration, predefined settings	print queue	printer configuration
Connection	USB,Ethernet	Ethernet	Ethernet

Table 2.1: Summary of Different existing Drivers

Table 2.1—"Summary of Different existing Drivers" shows, that none of the drivers works on all three major OS. Also only the vendor provided version supports all three modes of operation (see 1.1.1—"Modes of Operation"). Since the drivers differ in the way they decide which part of a graphic belongs to vector mode and which does not, graphics created for one of the drivers have to be altered in order to use them with another. The two ways of configuring the laser settings in the open source drivers both have some drawbacks: if there is a certain set of frequently used materials, configuring the settings each time before send a job is not optimal. Also if one has to specify the settings in the

None of the drivers works on all OS and support all three modes of operation

printer queue, it is possible to have one queue per material, but changing a value just a little always requires configuring a printer queue.

2.3 Summary of Related Work

From the introduced interactive systems, we got some concepts for the visual part of VisiCut. We will use these results together with the results of the driver comparism to derive the requirements for our software in the next chapter.

Chapter 3

Own work

"All things are difficult before they are easy."

—Thomas Fuller

In this chapter, we first present the results of our user survey (A—"User Survey to Determine Habits in Laser-Job Creation"). From this results and the results of the previous chapter, we will derive the requirements for our software. This is followed by a description of the architecture, which defines the boundaries between VisiCut and the LibLaserCut.

Finally, we will present the design and implementation of VisiCut and LibLaserCut separately.

3.1 Survey

To get an overview of the habits of users when creating jobs for laser-cutting, we made a user survey (see A—"User Survey to Determine Habits in Laser-Job Creation"), which revealed the following results:

- Windows, Linux and Mac OS X are often used, as well as a wide range of different graphic software, so our

software should not be restricted to one of them (see figure A.11 and figure A.5).

- Importing, positioning and configuring the laser-cutter and material values takes approximately 66% of the overall time (see figure A.4). Since time in fab labs is valuable, we aim to minimize the time needed for these steps.

- Most of the files contain parts for cutting and engraving (figure A.8) and the users way to separate them varies (see figure A.7). In order to minimize the learning curve and the influence on the creation process, the software should be able to deal with several ways of separation.

3.2 Requirements

From the results of chapter 2—"Related work" and the above results, we derive the following requirements R1-R6:

- **R1 Platform independence**: the software should be portable to all major OS without big effort. This removes the need for fab labs to use specific OS and also enables the users to use the software on their home computer, no matter what OS they use.

- **R2 Provide preview**: In order to move the design process from the labs to the home environment, the program should provide a preview of the graphic and how it approximately looks like, when laser-cut on certain materials.

- **R3 Reusable API**: Since there is no laser-cutter API available yet, the library should provide a general and reusable API so it can be a base for a wide range of laser-cutter specific software.

- **R4 Easy sharing and publishing of work**: Since the user shall be able to prepare jobs at home, there has to be an option to save the work for transporting it into the lab where the job is executed.

- **R5 Store material specific settings**: Since there are some materials, which are frequently used, the software should be able to store the settings in a material database, so the user just has to select the material instead of dealing with technical terms like power and speed.

- **R6 Support different ways of separation**: The user should be able to model his idea in a way he is familiar with. Thus, we want to minimize the requirements a graphic file has to fulfill to be used with our software. Supporting different ways of encoding the laser-profile also allows using graphic files created for different drivers without modification.

3.3 Architecture

We want to separate the front-end (VisiCut) from the back-end (LibLaserCut) to provide a reusable API, which is independent from the UI tool we create (figure 3.1). This allows the creation of other applications like VisiCut and other laser-cutter related software, without rewriting the driver.

Separate the driver part from the UI part

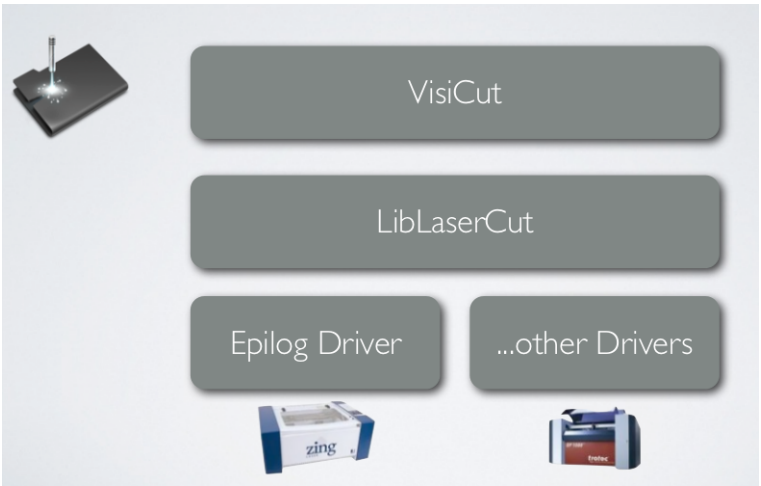

Figure 3.1: The Software Architecture to provide reusable API and laser-cutter drivers

3.3.1 Front-End Architecture

The UI tool VisiCut never uses a driver directly, but uses always the interfaces provided by the library. This enables adding drivers without changing the VisiCut tool, even without recompiling, since we can add classes to the Java classpath after compilation.

3.3.2 Back-End Architecture

We support all three modes of operation (see 1.1.1—"Modes of Operation"), each with a low- and a high-level API. The high-level API provides support for advanced data structures as they are used by common graphic applications and file formats, whereas the low-level API uses data structures which are similar to the ones used in drivers, giving full control over the laser process.

Divide API into low- and high-level

A driver for this library just has to understand the low-level structures, because the library will automatically convert all calls to the high-level API to calls to the low-level API (see table 3.1—"Low- and high-level data structures for the three modes of operation").

Drivers only use low-level

We do not provide a mechanism to connect to a laser-cutter, which allows each driver to implement the communication completely independent. So the driver can decide, if it supports USB or Ethernet or whatever it is capable of.

The driver is responsible for the connection

mode	**high-level**	**low-level**
vector	shape, path	polygon
raster	grey scale image	black/white image
3D raster	color image	grey scale image

Table 3.1: Low- and high-level data structures for the three modes of operation

3.4 VisiCut

In this section we will give an overview of the development of VisiCut. First, we will define some terms we use for describing certain elements and operations in VisiCut. Then we will sketch the design process by presenting our different prototypes and the integration of the concepts of the interactive systems introduced in section 2.1—"Interactive Systems with Real-World Input". Finally we will go into some implementation details.

3.4.1 Terms and Definitions

In order to understand the following concepts of VisiCut, we define some terms:

GRAPHIC FILE:
A graphic file is a file created by the user with a tool like Adobe Illustrator, which contains a model of the user's laser-project

Definition:
graphic file

GRAPHIC OBJECT:
A graphic object is a visible part of a graphic file, meaning one of the following categories: path (rectangle, circle, bezier curves), text or image (embedded raster image).

Definition:
graphic object

LASER-PROFILE:
A laser-profile corresponds to a way to handle graphic objects on a laser-cutter. An example of a laser-profile could be "cut" which moves the laser-head along each shape of path objects in order to cut the material.

Definition:
laser-profile

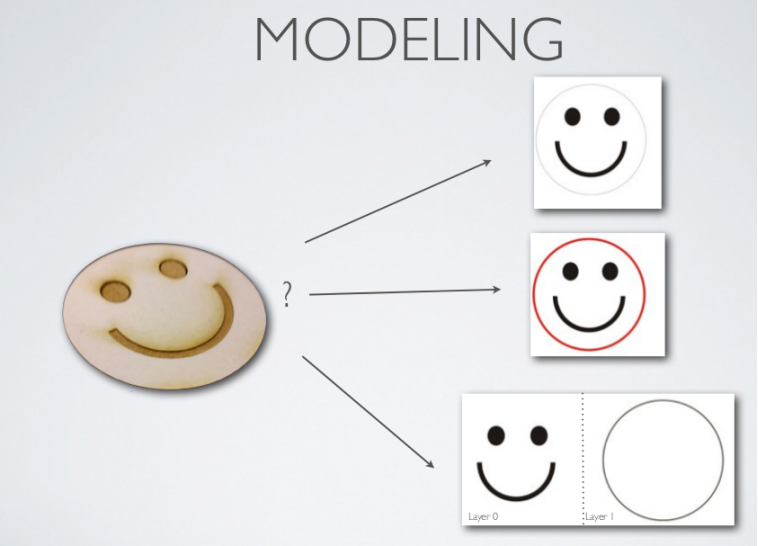

Figure 3.2: Modeling the same object in three different ways

3.4.2 The Modeling Problem and the Mapping Concept

User tests have shown (figure A.7), that there are different approaches of modeling a laser-job into a graphic file (see figure 3.2 for examples).

In order to deal with the modeling problem, we define the term mapping:

Definition: *mapping*

MAPPING:
A mapping is a function, which maps each object in a graphic file to a laser-profile

3.4.3 Design of VisiCut

VisiCut is was created using an iterative and incremental software developement technique, according to the definitions in [Cockburn, 2008]. We created several UI-prototypes and made user tests, which revealed weak-

nesses of UI design and missing features. We also took some steps to refactor ([Fowler and Beck, 1999]) parts of the project, because the new features did not always fit into the existing code structure.

We will now sketch the iterative design of VisiCut by presenting three prototypes. After that, we will explain the advantages of the camera preview and finally show some implementation details, including a description of VisiCut's Portable Laser Format

Prototype 1: Mapping by Selecting Objects

In the first prototype, each visible element in the graphic could be selected via a click and then could be mapped to a laser-profile by a context-menu. User tests revealed, that this is very intuitive, but many users have files containing a big number of graphic elements, so this method is only feasible for small graphic files.

The objects are selectable via mouse clicks

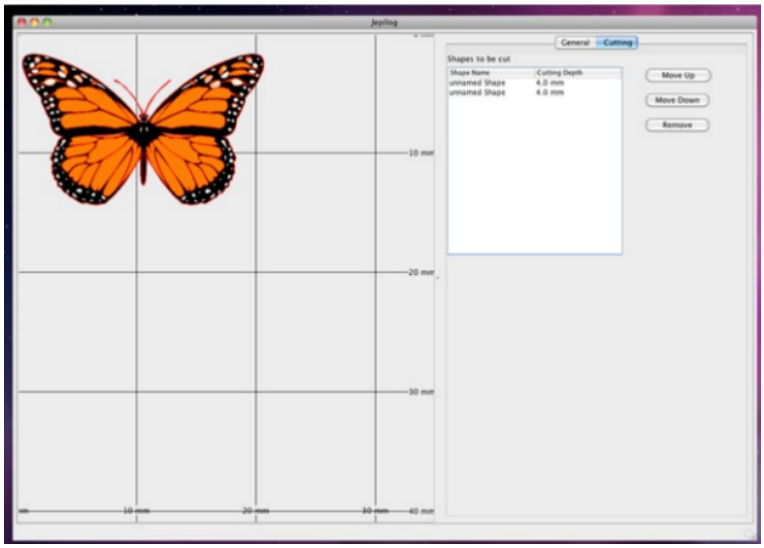

Figure 3.3: In the first prototype, everything is selectable by clicking

Prototype 2: Mapping by Filter Rules

Objects are categorized by defining powerful filter rules

Since many users (e.g. architecture students whit models of complete buildings) have big input files, we concluded that there is a need for rule-based mapping. The second prototype allows creating a mapping, based on a set of filters, which specify a set of graphic elements by their attributes, each mapped to a laser-profile. Additionally it was possible to make a filter calculate the outline of a set of shapes instead of using the shapes itself.

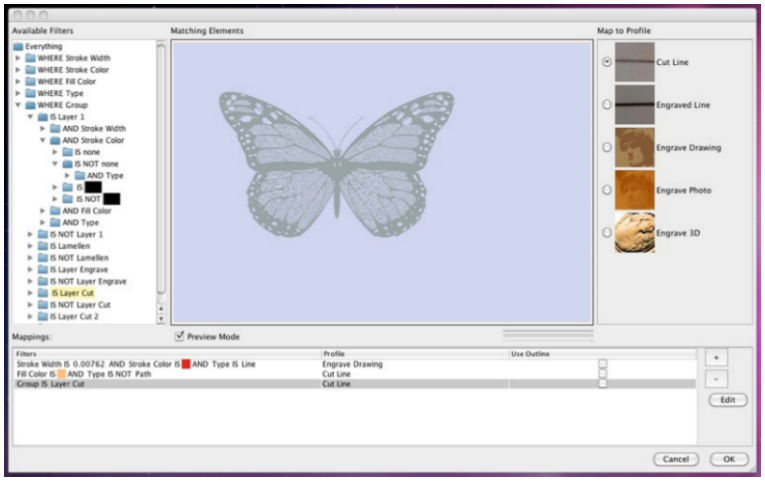

Figure 3.4: Prototype 2 allows nearly any mapping through complex filter rules

Most users are confused and do not need so much power

User tests revealed, that the dialog is very hard to understand and most users have problems understanding the mapping concept at all. Furthermore, a survey (see A—"User Survey to Determine Habits in Laser-Job Creation") showed, that all participants use at most one attribute, like line color, line thickness or layers to differentiate between laser-profiles.

Prototype 3: Mapping by Only One Attribute

Provide default mappings and single attribute mappings

The results of the survey see A—"User Survey to Determine Habits in Laser-Job Creation" lead us to the final prototype, which contains a set of default mappings and a sim-

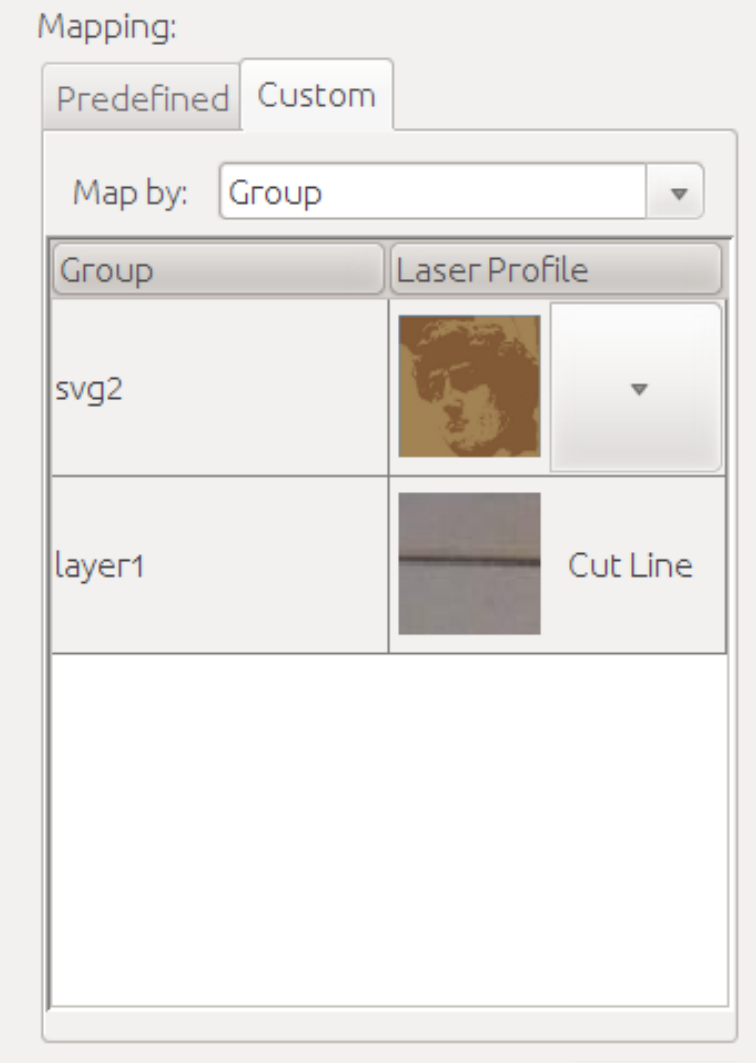

Figure 3.5: Prototype 3 allows mapping by only one attribute

plified version of the mapping dialog in prototype 2. The simplified dialog just allows selection of one attribute and differentiation between all the existing values of this attribute (see figure 3.5).

The available attributes depend on the file format and the file itself. The `GraphicObject` interface (see section 3.4.4—"The GraphicObject Interface") allows any `Importer` class, which loads a file of a specific file for-

Only attributes appearing in the file are displayed

mat into the program, to specify an arbitrary number of attributes and assign values to the elements in the file. This means it is possible, that if you load a DXF file, there is an attribute *layer*, which does not exist in SVG files.

VisiCut also analyzes the different values of all attributes in the input file, so if e.g. everything is in the same layer, it makes no sense to separate by layer, so it will not be shown in the dialog.

The current attributes supported by the different file formats are shown in table 3.2—"The available attributes for the supported file formats".

format	**attributes**	**examples**
All	type	rectangle, image, text
	stroke-width	0.2pt 1mm
	stroke-color	red
	fill-color	green, rgb=22,123,123
SVG	group	"group 123", "surface"
DXF	layer[1]	"layer 3"

Table 3.2: The available attributes for the supported file formats

The WYSIWYG Part

a camera provides preview on a photo of the object and allows interactive visual positioning

To solve the positioning problem, VisiCut takes an approach similar to the CopyCAD project [Follmer et al., 2010], but instead of mounting a projector and moving the complete editing onto the real world object, we just add a camera on top of the laser-cutter to capture a picture of the material. Once calibrated, the position of the camera is known, so there is a defined transformation, to map image coordinates to laser-cutter coordinates. This allows VisiCut to render the preview directly on the camera image and enables the user to position and resize his graphic directly on the preview (see figure 3.6). Since this "allows the user to

[1] In the current implementation of the DXF importer, layers are mapped to groups, but this may change in the future

see the real world, with virtual objects superimposed upon or composited with the real world" [Azuma et al., 1997], VisiCut can be seen as Augmented Reality system.

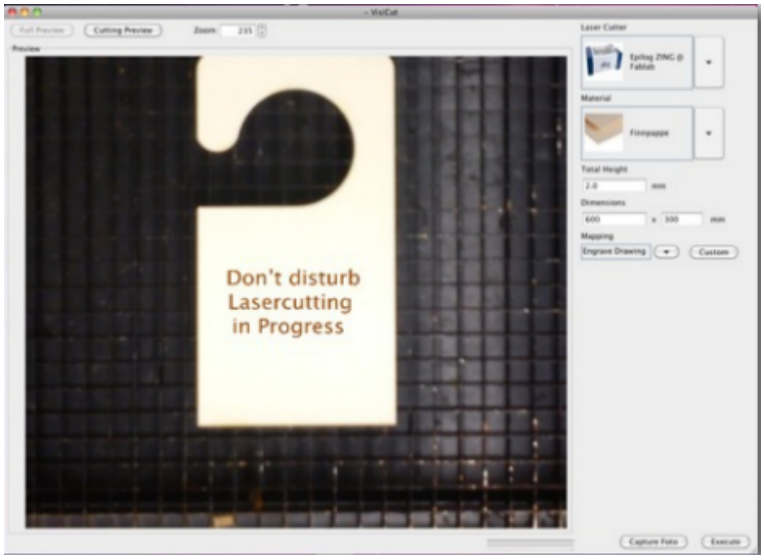

Figure 3.6: In VisiCut you can see how and where your graphic will appear

By using absolute coordinates, it is no longer necessary, to move the laser-head by hand. This is a dangerous operation anyway, because if the graphic size is too large, the laser-head can crash on the wall. VisiCut also saves information like the material thickness in the material-profile, so there is no need to focus manually, because as long as the laser-head is in position (0,0,0) at the beginning of each job, VisiCut can set the focus according to the material thickness automatically.

The camera together with material profiles removes any manual positioning and focusing of the laser-head

The Right Camera Resolution

In our setting, the laser-bed is 60cm x 30cm and we use a resolution of 500 DPI.

The absolute resolution can be calculated as: $500DPI = 500\frac{dots}{2.54cm} = 169.85\frac{dots}{cm}$

$$60cm \cdot 169.85 \frac{dots}{cm} = 11811 dots$$

$$30cm \cdot 169.85 \frac{dots}{cm} = 5906 dots$$

The laser-cutter has a resolution of 69 mega pixels

which is about 69,8 mega pixels. This means, in order to provide a preview which is as accurate as the laser, we would need a 69,8 mega pixels camera (or even more, because the offset of the image and the difference between horizontal and vertical resolution is not even considered in this calculation). This result shows that, in order to provide an at least sufficient preview, we need a really high resolution camera.

3.4.4 Implementation of VisiCut

The Camera Implementation

To keep the VisiCut platform independent, we do not want to include any camera driver into the program itself. Instead we require the camera to store the picture in the JPEG format at a local or remote URL.

Many IP cameras support this feature, but the supplied pictures are only snapshots of a video. That means they have a resolution of usually 640x480 pixels or if it is an HD camera 1920x1080. The above result shows, that this resolution is not good enough. What we need is an IP camera, which takes real pictures, either in a constant time interval or every time a client sends an HTTP request.

Since we were not able to find such a camera on the market, we had to design our own. For our prototype we used an Android phone with a camera resolution of 5 Megapixels. For the webserver we used nanohttpd[2] , a webserver implemented in a singe Java file. The app starts a webserver at port 8080 and every time a client sends a request, it takes a picture and sends it as JPEG. The implementation is provided with this thesis, but it is just a prototype.

[2] http://elonen.iki.fi/code/nanohttpd/

There is an *application*[3] *for controlling Canon PowerShot cameras from a Windows PC*. An advanced setup could connect such camera to a computer running an HTTP daemon which requests the camera to take a picture, or even just a script, which takes a picture every 5 seconds and save it on a location reachable from the computer running VisiCut. This would improve the preview quality significantly.

The Pictionaire system (see section 2.1.2—"Pictionaire") uses a "digital still camera centered above the table captures photos of the entire table on demand", which provides a resolution of "4272x2848 pixels" [Hartmann et al., 2010]. This is about 12 mega pixels and would be about 36 % of the horizontal and 48% of the vertical resolution needed, to have one pixel per laser coordinate. This camera could be used to improve the current solution, which provides only a resolution of 2592x1944 pixels, which corresponds to 22% of the horizontal and 33% of the vertical laser-cutter resolution.

General Structure of VisiCut

VisiCut itself is created using NetBeans and it's graphical UI editor Mantisse.

Most parts of the business-logic are separated in the `VisiCutModel` class and the classes in the `com.t_oster.visicut.model` and `com.t_oster.visicut.mapping` packages, however some logic is still contained in the UI classes. It is recommended to use NetBeans for further programming and compiling, but the supplied *build.xml* file should enable users to compile with apache-ant without having NetBeans installed.

The GraphicObject Interface

In order to be independent of a specific file format, the file format specific classes are wrapped by the

[3] http://www.breezesys.com/PSRemote/

GraphicElement classes and other classes found in the com.t_oster.visicut.graphicelements package. The GraphicObject interface represents an object like rectangle, path, text or image. Any GraphicObject has a bounding box and it can be rendered on a Graphics2D object. With the rendering it is possible to use a GraphicObject for engraving.

The ShapeElement class is a subclass of GraphicElement, which additionally provides the getShape method. This method provides a java.awt.Shape object representing the shape of the GraphicObject. With this shape, the library is able to use the object for vector-profiles, e.g. cutting.

Adding a New Input File Format In order to add support for a new file format, basically three things have to be done:

1. A subclass of the Importer interface, which reads a file and converts it into a set of GraphicObjects, has to be created.
2. An implementation of the GraphicObject interface, which is able to render the objects on a java.awt.Graphic2D object for raster profiles and to return a java.awt.Shape object representing the object for vector profiles needs to be provided.
3. The new Importer has to be added to the default Importers in the settings.xml or directly in the PreferencesManager class.

The Portable Laser Format

The Portable Laser Format (PLF) is designed to be simple, but powerful. It consists of three files (see 3.7), which are aggregated in a ZIP container.

The first file is the original graphic file (in later versions maybe more than one) in the original format. This allows the user to extract his untouched graphic at any later

point and re-edit it. The second component is a file named *mapping.xml*, which is a serialized `MappingSet` object that specifies how the elements of the input file are mapped to laser-profiles of the material. The third component is a serialized `java.awt.AffineTransform` object, which defines the position, size and transformations from graphic coordinates to laser-cutter coordinates. This object is stored in a file named *transform.xml*.

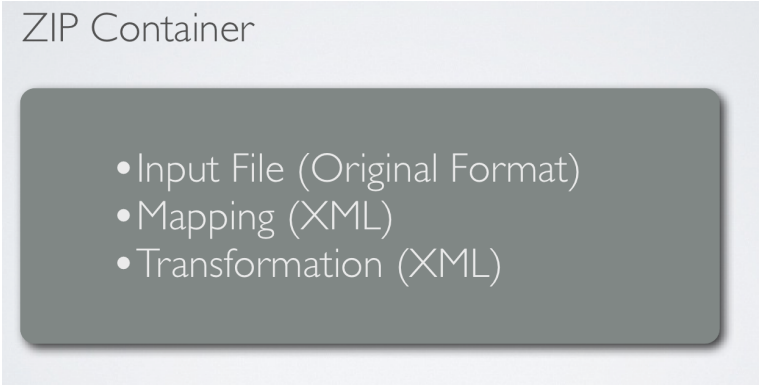

Figure 3.7: The basic structure of a PLF file

We did not include the material-profile, because a project should be material independent if possible, and since the laser-properties for a material varies from laser-cutter to laser-cutter, the profile has to be created explicitly for one laser-cutter model.

The three mentioned files are packed in a ZIP container, because it shrinks the needed file size and having just one file containing all information is easier to handle.

3.5 LibLaserCut

In this section we will present the design of the LibLaserCut. After that, we will introduce the important interfaces for implementing a new driver and using the library. Finally we will show some implementation details about the driver for the Epilog ZING, we implemented.

3.5.1 Design of LibLaserCut

The library is working on a job basis, meaning a `LaserJob` object is created and filled with all information needed to execute it on a laser-cutter. Then this job object is passed to a driver, meaning an object implementing the `LaserCutter` interface, which handles the translation of the `LaserJob` object to whatever language the laser-cutter needs.

The LaserCutter Interface

The `LaserCutter` interface is the only interface, a new laser-cutter driver has to implement. It contains a range of getters for attributes the application needs to know about the laser-cutter, like laser-bed dimensions and supported resolutions. It also contains the methods `getSettingAttributes`, `getSettingValue` and `setSettingValue`, which enables each driver to specify a set of settings, an application can manipulate.

The heart of the interface is the `sendJob` method, which is where the real driver code will resist. The driver gets a `LaserJob` object, which contains general job information and one or more of the following three parts.

Vector Part

The `VectorPart` represents a vector mode job, meaning the laser-head follows a 2-dimensional path while switching the laser on or of. This mode is similar to a usual pen plotter device receiving Pen Up and Pen Down commands. The `VectorPart` contains a List of `VectorCommands` which are listed in 3.3—"An overview of the different VectorCommands". *MOVETO* and *LINETO* are the commands, which actually move the laser-head, whereas *SETPOWER*,*SETSPEED* and *SETFREQUENCY* change the current settings. While *MOVETO* switches the laser off before moving and always moves with maximum speed, *LINETO* sets the power and frequency according to the most recent *SETPOWER* and *SETFREQUENCY* command, and moves with the speed specified by the most recent *SETSPEED* command. *SETFOCUS* can be seen as moving the z-axis, but there are laser-cutters imaginable, which alter the laser-beam (e.g. through a lens) to focus it.

command	parameters	description
MOVETO	x, y	move the head to (x,y) (in dots) with laser switched off
LINETO	x, y	move the head to (x,y) (in dots) with laser switched on
SETPOWER	power	sets the intensity of the laser beam (in %)
SETSPEED	speed	sets the speed of the head movement (in %)
SETFREQUENCY	frequency	sets the pulse frequency of the laser beam (in Hz)
SETFOCUS	focus	moves the Z-Axis to the given distance (in mm)

Table 3.3: An overview of the different VectorCommands

Raster Part

In raster-mode the laser-cutter operates line-wise on a constant speed, while switching the laser on or off on a per-dot-base according to a given bit-raster. A `RasterPart` object can contain multiple rasters, each with a start point, defining the top left corner of the raster on the laser-bed

(in dots), and a `LaserProperty` object representing the power,speed and focus settings.

3D Raster Part

3D-raster-mode is like raster-mode, but instead of switching the laser on or off at each dot, its power is scaled to the raster value. Between two raster cells, the power is scaled linearly from the current power value to the next power value. This allows engraving at different depths, creating 3 dimensional effects. Like the `RasterPart` object, a `Raster3DPart` object can contain multiple rasters, each with start point and `LaserProperty`. The power value in the `LaserProperty` is the value which corresponds to a byte with value 255 in the raster. Other values are to be scaled linearly, meaning a raster value of 128 represents 50% of the power specified in the `LaserProperty`.

3.5.2 Implementation: The Epilog Driver

The Epilog ZING cutter supports a subset of the PCL (Printer Command Language) for the Raster modes and a subset of the HP-GL (Hewlett Packard Graphic Language, see [Hewlett-Packard, 1997]), a language developed to control pen plotters, for the Vector Mode. Multiple parts can be embedded in one print job, which is sent as PJL (Printer Job Language) block.

For more information about the different printer languages, see History of Printer Command Language PCL[4] .

Vector Part Implementation

A vector part is introduced with a PCL header, specifying direction and size of the coordinate system, followed by a HPGL block with commands, very similar to the `VectorCommands`. The HPGL commands are:

[4] http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?objectID=bpl04568

PU[x_0,y_0[,x_1,y_1...]; PU stands for Pen Up and is interpreted as *switch the laser off and move along the given Polygon*. The moving speed is independent of the current speed setting. In the driver we can aggregate a whole block of `MOVETO` commands to a single PU command, because PU supports an arbitrary number of point coordinates.

PD[x_0,y_0[,x_1,y_1...]; PD stands for Pen Down and is interpreted as *switch the laser to the current Power and Frequency setting and move along the given Polygon with the current speed*. Similar to PU, we can aggregate complete blocks of `LINETO` commands.

The following commands are not part of the HP-GL specification HP-GL Graphics Language[5] :

WF$focus$; Sets the Z-Axis relative to the Z-Axis position on the start of the current job. The $focus$ value has to be between -500 and 500, otherwise this command is ignored. One unit of the $focus$ value is a distance of 0.0252mm, which means a total range of 1000*0.0252mm = 2.52cm can be achieved by the automatic movement. At the end of a job, the Z-Axis, in contrast to the X and Y axis, will remain on its position, however, when the next job starts, it will return to the position 0. Thus the only way of changing the 0 position is manually using the buttons on the laser-cutter.

XR$frequency$; Sets the laser pulse frequency to the given value. Valid values range from 500 to 5000, but every number has to be provided as four-digit value, meaning 0500 for 500 etc.

YP$power$; Sets the power of the laser beam in %. The value has to be given as three-digit value, so the range is from 000 to 100.

[5] http://cstep.luberth.com/HPGL.pdf

ZS$speed$**;** Sets the speed of the laser head (when laser switched on) in %. Like the power setting, this has to be given as three-digit value. In order to estimate job time, we empirically calculated the speed. See C—"Speed Measurement Results" for details.

Raster Part Implementation

After the PCL header, which contains the power,speed,focus and some general page information, a PCL Raster is sent line-wise. Each line starts by a pair of *$p\langle value\rangle X$ and *$p\langle value\rangle Y$ commands, which define the most left dot of the raster line. The data for every line are encoded in TIFF packbits encoding[6] , which reduces the data size significantly, if the line contains many equal pixels next to each other.

PCL command	description
y$\langle power\rangle$P;	sets the laser power
z$\langle speed\rangle$S;	sets the laser speed
y$\langle focus\rangle$A;[7]	sets the focus value

Table 3.4: PCL commands specific to the Epilog ZING

The encoded line data is announced by a *$b\langle value\rangle A$ command, containing the number of dots in the line. If the number is negative, it means that the data encoded from right to left. This command is followed by *$b\langle value\rangle W$, which contains the number of bytes, that will be sent.

An overview of the Epilog specific PCL commands is listed in table 3.4—"PCL commands specific to the Epilog ZING". For a detailed description of the different PCL versions, see [Smith, 1998].

[6] http://www.fileformat.info/format/tiff/corion-packbits.htm

[7] This command also seems to disable the auto-focus feature. Since our laser-cutter does not support auto-focus, we were not able to test it properly

3D Raster Part Implementation

The 3D raster PCL is very similar to the raster PCL and uses the same run length encoding. The difference is mainly the encoding specification in the header, which is *2M* for raster, and 7MLT for 3D-raster mode. Note that it is not possible to have raster and 3D raster parts in one job, because the laser-cutter does not differentiate between the power settings. Thus, our implementation automatically splits a job in two jobs, if it contains 3D-raster and one other part.

Sending the Job via LPD

For receiving the PJL job, the Epilog cutter runs a Line Printer Daemon (LPD), which listens for incoming connections on Port 515. A specification of the LPD protocol can be found in RFC1179[8]

3.6 Summary of Own Work

In this chapter, we presented our software system, consisting of the front-end VisiCut and the back-end LibLaserCut. For more detailed information, please look into the source code, which can be found at our GitHub project page[9]

[8] http://www.rfc-editor.org/rfc/rfc1179.txt
[9] https://github.com/t-oster/VisiCut/

Chapter 4

Evaluation

"Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted."

—Albert Einstein

4.1 Requirements

In this section, we will discuss wether and to what degree the requirements from 3.2—"Requirements" are met.

4.1.1 R1: Platform independance

We successfully tested VisiCut and LibLaserCut on Windows, (Arch- and Ubuntu-) Linux and Mac OS X. Since the application and library are all written in pure Java, there is no need to recompile the program for each architecture, instead the same java bytecode can be run on all mentioned platforms.

On the VisiCut website[1] , we provide a platform independent ZIP file, but also some prepackaged versions for Mac OS and Linux, of the afore mentioned software.

[1] http://hci.rwth-aachen.de/visicut

Thus we can say, that we achieved the goal of platform independance.

4.1.2 R2: Provide preview

Although the current preview uses only a few colors and does not respect engraving depth and burning effects, its results are suprisingly similar to the real result. A big plus is, that the user can now be sure, which lines are cut and which are engraved, which was a problem with the other software. Thus, we can say the preview functionality is a good raw estimation for the user how the result will look like.

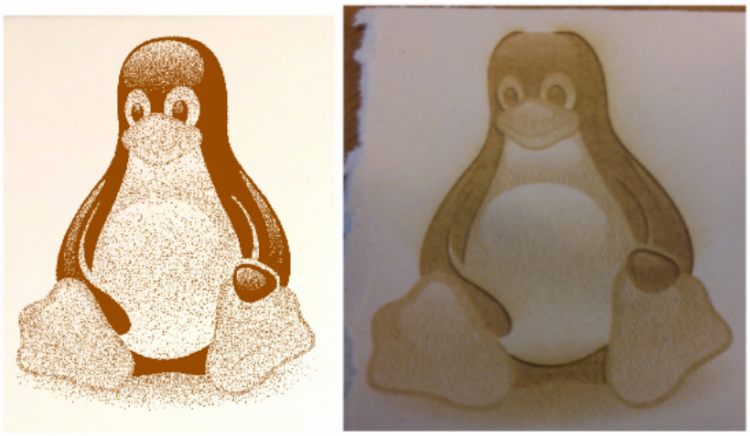

Figure 4.1: The preview generated by VisiCut on the left and the result on the right

4.1.3 R3: Reusable API

Since we only implemented one driver, we cannot say much about the generality of the API yet. However, the arbitrary attribute list in the `LaserCutter` interface should suffice any device's need of configurable parameters. In addition to that, we created an experimental driver for the

Mantis milling machine[2] , which can generate G-Code for milling along paths.

4.1.4 R4: Easy sharing and publishing of work

The PLF format (see section 3.4.4—"The Portable Laser Format") enables users to save a completely prepared laser-job, so anyone with a VisiCut compatible laser-cutter can re open it and execute it. The only thing needed is a VisiCut instance with some material profile. Additionally, a function could be added to publish the current work directly from VisiCut, rather than saving and manually uploading (see 5.2.3—"Creating a Platform for Sharing VisiCut Files and Material-Profiles").

4.1.5 R5: Store material specific settings

In VisiCut, the user does not have to deal with laser parameters, such as power, speed or focus, at all, because he just selects a material profile. However it would be nice to have some platform to share the created material profiles (see section 5.2.3—"Creating a Platform for Sharing VisiCut Files and Material-Profiles").

4.2 System Usability Scale

The System Usability Scale introduced by Brooke [1996] 2006 is a robust measure for the usability of an interactive system. In the final user test we asked each participant to fill in the a SUS questionnaire (see figure B.1). The overall result yields a usability score of 79.375. Although this is no guarantee for a good usability, it is still a good sign. According to section *5.1 What is an acceptable SUS score* in Bangor et al. [2008], this can be interpreted as our software not being a "superior product", but still in the category of "better products".

[2] http://makeyourbot.org/mantis9-1

Chapter 5

Summary and future work

"The best way to predict the future is to invent it."

—Alan Kay

In this chapter we will summarize our work and list some ideas for future improvement.

5.1 Summary and Contributions

VisiCut moves a big part of the usual workflow to the users home (see figure 5.1), which makes laser-cutting a bit more *personal* fabrication. It also removes the need for expensive software and allows re-using laser projects even on different laser-cutters. If, at some point, laser-cutters get cheap enough for home use, people can still use the same PLF files and the same VisiCut software, they can use now.

The **LibLaserCut** provides reusable low- and high-level API for laser-cutters. Since it is implemented in pure Java it runs on nearly any operating system, even on Android, which does have any printer support at all. With all benefits of open source, it can become a widely used library

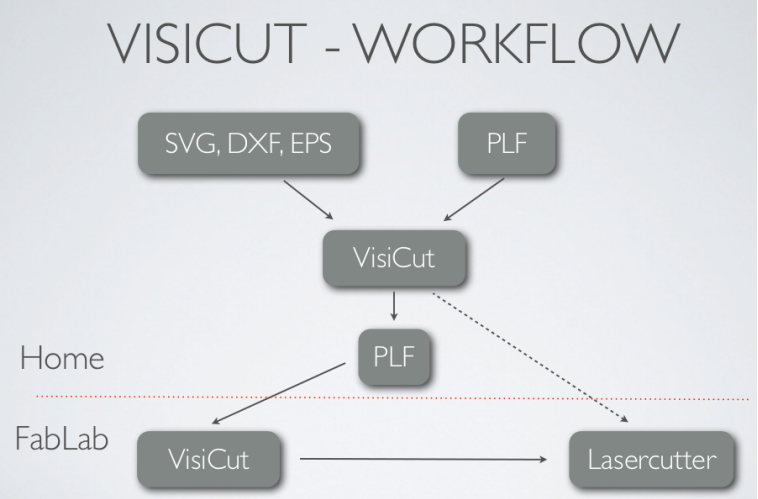

Figure 5.1: The VisiCut workflow

with increasing number of supported devices. With the implementation of VisiCut, we have shown, that the library serves all needed routines to build fully functional laser-cutter applications on top of it.

The source code can serve as a base for future projects, including open source drivers or new laser-cutter firmwares.

5.2 Future Work

5.2.1 Improving the Preview Quality by 3D-Rendering

Currently, VisiCut uses only one color for the material and another for the laser-profile to create the preview. Also the performance of the dithering and rendering routines is quite slow. An interesting addition could be support for 3D rendering (e.g. Open GL), which speeds up the rendering process and also could add nice features, e.g. a real 3D preview, respecting engraving depth and burned edge color.

5.2.2 Exctract Vector Data from Raster Files with User Support

Often people want to cut images, which exist only in a raster formats such as PNG, e.g. because they were found on a website or they are pictures from real world objects. Currently it is only possible to engrave, but not to cut raster images. The CopyCAD tool contains an algorithm to detect shapes on a camera picture (see Follmer et al. [2010]). It would be nice to integrate this algorithm into VisiCut for allowing both: vectorizing existing raster-images and recognizing shapes directly from the camera preview.

5.2.3 Creating a Platform for Sharing VisiCut Files and Material-Profiles

The PLF format allows easy sharing of the created laser-jobs and our survey has shown, that many users would publish their work if there was an easy to use function for that (see figure A.6). So one could investigate options to add support to one-button upload to Thingiverse[1] or other platforms.

Another interesting field are the material profiles. Since it requires much time and experience to find the right laser values for a certain material, there is a need of a platform to share the material profiles.

5.2.4 Engraving Non-Planar Objects by Providing a 3D-Model

Since it is possible to move the z-axis with the focus property, it is possible to engrave on non planar surfaces, if a 3D-model (height-map) is given. In vector-mode one can alter the focus during a path, but in the raster modes, it would be necessary, to split the raster into multiple rasters where each raster contains data for a certain height.

[1] http://www.thingiverse.com/

5.2.5 Optimizing the Execution Speed

Currently, VisiCut generates the laser-job in an order, mainly influenced by the structure of the original graphic file and the mapping. There is no optimization in terms of speed done. The concept to optimize the execution speed by minimizing the overall path the laser-head has to move, introduced by [Vaupotic et al., 2006], seems to be a good starting point.

5.2.6 Improving the Camera Setup

Currently we use an Android phone as IP camera. Since the laser-cutter is capable of creating jobs with 69 mega pixels, a camera with higher resolution would improve the preview significantly. We listed some alternatives to the current setup. (see 3.4.4—"The Camera Implementation").

Furthermore the current calibration procedure need user intervention, which could be simplified or fully automatized.

5.2.7 Multiple Input Files

Currently, VisiCut only supports using one graphic file at a time. However, this could be changed in order to allow users to integrate different models into one laser-job. The first question, which has to be answered before implementing such a feature, is whether each input file should use a separate mapping or not. Also, the PLF Format should be adapted in a way that is is upwards compatible.

5.2.8 Back to the Printer Driver

As outlined in the section 2.2.4—"Limitations of the Printer Driver Approach" there are reasons, not to implement VisiCut as printer driver. However, a printer driver has the advantage of being accessible with one click from nearly

any graphic application. Since VisiCut supports the EPS format, it is possible, to create a CUPS back-end, which just converts the print job to EPS and starts VisiCut.

This would allow both, using VisiCut with a single click as printer driver's dialog and still being able to run it standalone with the full power of formats different from EPS.

5.3 Conclusion

We outlined, that there is a need for user-friendly and platform independent laser-cutter software and provided a prototype, consisting of a UI tool VisiCut and a library LibLaserCut, as a basis for further development.

"Well, so long, mister. Thanks for the ride, the three cigarettes and for not laughing at my theories on life."

—from "The Postman Always Rings Twice"

Appendix A

User Survey to Determine Habits in Laser-Job Creation

The full data and evaluation can be found on the CD-ROM or as download from:

Complete results[a]

[a] http://hci.rwth-aachen.de/tiki-download_wiki_attachment.php?attId=1386&download=y

Umfrage - VisiCut

Teilnehmer Nummer: ___

1. Was ist Ihr Beruf / Studiengang	
2. Wie häufig arbeiten Sie mit Grafikprogrammen?	○ nie ○ selten ○ gelegentlich ○ oft ○ sehr oft
3. Haben Sie bereits mit einem Lasercutter gearbeitet?	○ Ja ○ Nein
4. Wie schätzen Sie den Zeitaufwand (in %) für die einzelnen Arbeitsschritte im Fablab ein?	Importieren der Datei nach CorelDraw ____% Positionieren und Nachbearbeiten ____% Ermitteln der Materialparameter ____% Konfigurieren des Lasercutters ____% Ausführen des Laserjobs ____%
5. Mit welcher Software haben Sie Ihre Datei erstellt?	
6. Unter welchem Betriebssystem haben Sie Ihre Datei erstellt?	
7. Unter welchem Betriebssystem arbeiten Sie normalerweise?	
8. Mit welcher Grafiksoftware arbeiten Sie am liebsten?	
9. Welche Laserprofile enthält Ihr Projekt?	○ Schneiden ○ Gravieren ○ 3D-Gravieren
10. Wie haben Sie in Ihrer Grafik die Unterscheidung zwischen den Laserprofilen kodiert?	○ Farbe ○ Layer ○ Füllung ○ einzelne Dateien ○ Sonstiges: _____
11. Welche Art ist für Sie am intuitivsten?	
12. Haben Sie die Anleitungen/Tipps auf der FabLab Seite gelesen?	
13. In wie fern haben diese Anleitungen ihren Erstellungsprozess beeinflusst?	
14. Wie wichtig sind für Sie folgende Funktionen eines Lasercutter Tools:	Funktion / Unwichtig ○ ○ ○ ○ ○ Sehr wichtig Gute Vorschau Projekt speichern Viele Dateiformate Performance
15. Würden Sie ein Tool von der FabLab Seite	○ Ja ○ Nein

Figure A.1

benutzen um ihren Job vorzubereiten?	
16. Würden Sie Ihr Projekt frei zur Verfügung stellen, wenn es einen einfachen Mechanismus im Programm dafür gäbe?	○ Ja ○ Nein

Figure A.2

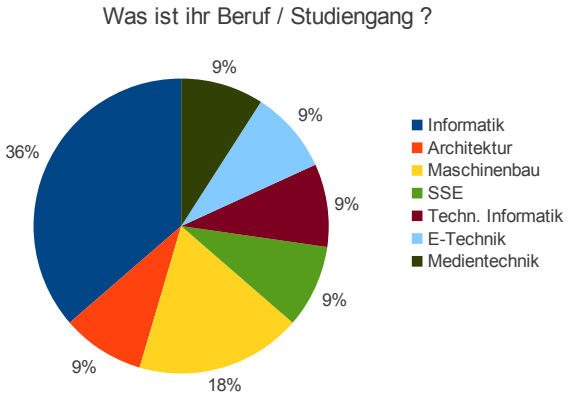

Figure A.3

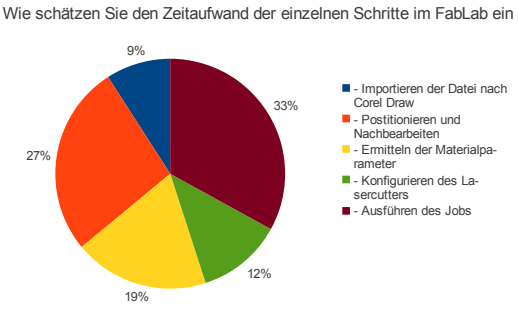

Figure A.4

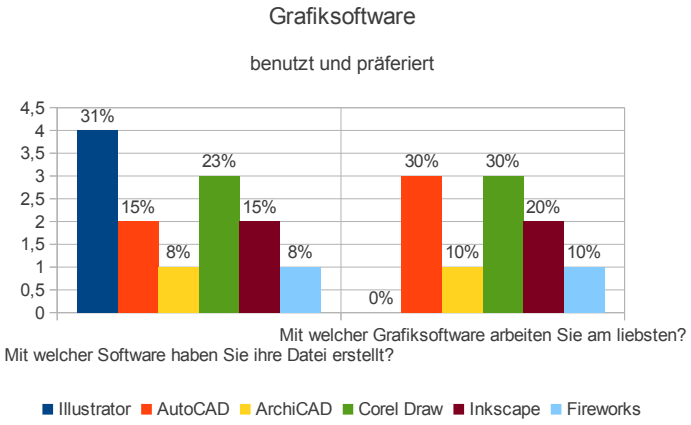

Figure A.5

Figure A.6

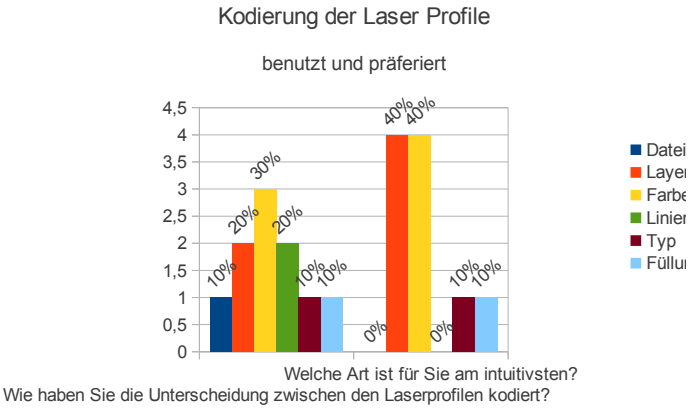

Figure A.7

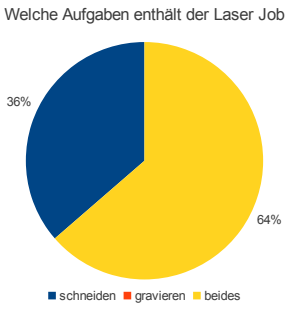

Figure A.8

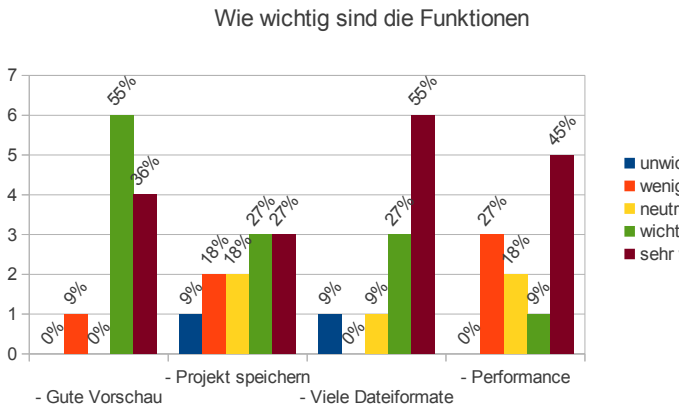

Figure A.9

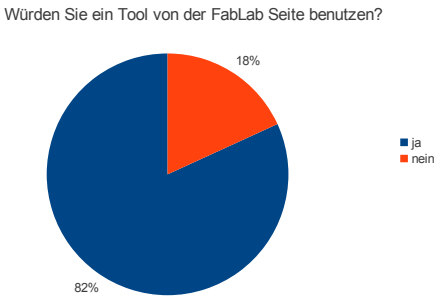

Figure A.10

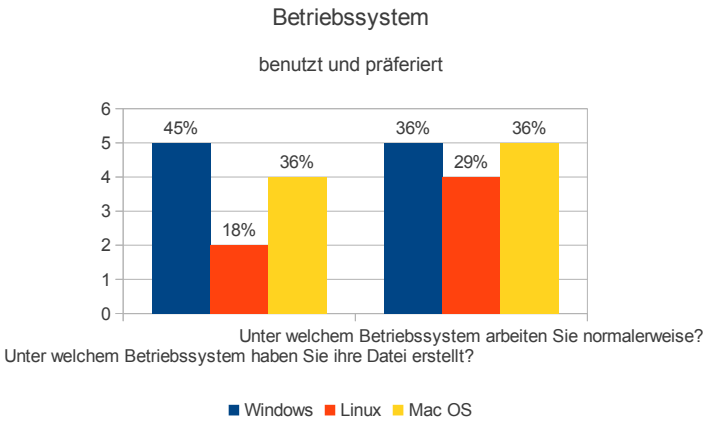
Betriebssystem
benutzt und präferiert
6
5
4
3
2
1
0
45%
18%
36%
36%
29%
36%
Unter welchem Betriebssystem arbeiten Sie normalerweise?
Unter welchem Betriebssystem haben Sie ihre Datei erstellt?
Windows
Linux
Mac OS

Figure A.11

Appendix B

System Usability Scale

After the participants of our user tests had to perform some tasks with our software, they were asked to answer the questions of the System Usability Scale (see figure B.1). The results, calculated according Brooke [1996] are presented in figure B.2.

The overall average score is 79.375.

This value indicates, that our software may be in the range of *better products* (see 5.1 What is an acceptable SUS score in Bangor et al. [2008])

System Usability Scale - VisiCut

Teilnehmer Nummer:

Frage	Ich lehne es stark ab	Ich lehne es ab	Ich weiss nicht, neutral	Ich stimme zu	Ich stimme stark zu
1. Ich denke, dass ich dieses System gerne häufig nutzen würde.					
2. Ich fand das System unnötig komplex.					
3. Ich denke, das System war einfach zu benutzen.					
4. Ich denke, ich würde die Hilfe eines Technikers benötigen, um das System benutzen zu können.					
5. Ich halte die verschiedenen Funktionen des Systems für gut integriert.					
6. Ich halte das System für zu inkonsistent. 7					
7. Ich kann mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System umzugehen.					
8. Ich fand das System sehr mühsam zu benutzen.					
9. Ich fühlte mich bei der Nutzung des Systems sehr sicher.					
10. Ich musste viele Dinge lernen, bevor ich das System nutzen konnte.					

Figure B.1: The Questions of the System Usability Scale

SUS Scores

User ID	1	2	3	4	5	6	7	8
Question 1	4	4	4	5	5	4	5	5
Question 2	1	3	2	2	1	2	2	2
Question 3	4	3	4	2	4	4	4	4
Question 4	1	3	1	2	2	1	2	2
Question 5	5	4	5	5	5	4	4	5
Question 6	2	2	1	1	1	3	2	1
Question 7	4	4	4	4	5	4	5	4
Question 8	1	2	2	2	1	1	2	1
Question 9	4	3	5	3	4	3	4	3
Question 10	1	2	2	4	2	2	3	1
Score:	87,5	65	85	70	90	75	77,5	85

Average 79,375

Seite 1

Figure B.2: The results calculated as SUS scores

Appendix C

Speed Measurement Results

C.1 Epilog Cutter Speed Tests

In order to provide a raw estimation of the duration of a laser-job, we made a few tests and measurered the time.

C.1.1 VectorPart @ 100% Speed, 500 DPI

20000px in x direction : 36.8s with laser on, 4.5s with laser off

$\Rightarrow$ $543.4782608695652\frac{px}{s}$ with laser on, $4444.44444\frac{px}{s}$ with laser off.

10000px in y direction : 18s with laser on, 2.5s with laser off

$\Rightarrow 555.555555\frac{px}{s}$ with laser on, $4000\frac{px}{s}$ with laser off.

C.1.2 VectorPart @ 10% Speed, 500 DPI

20000px in x direction : 368s with laser on

$\Rightarrow 54.34782608695652 \frac{px}{/} s$

This is about 10 % of the 100% speed value, so we conclude, that the speed value scales linearly.

C.1.3 RasterPart @ 100% Speed, 500 DPI

$50x10000$px in x direction : 32.5s

$50x5000$px in x direction : 19.5s

C.1.4 RasterPart @ 10% Speed, 500 DPI

$50x10000$px in x direction : 268s

$50x5000$px in x direction : 136s

If we assume a constant offset at each line for the head to change direction and accelerate, we can calculate it from the above values as: $0.08 \frac{s}{line}$.

Bibliography

R.T. Azuma et al. A survey of augmented reality. *Presence-Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

A. Bangor, P. Kortum, and J. Miller. An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.

J. Brooke. SUS: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. Taylor and Francis, London, 1996.

Alistair Cockburn. Using both incremental and iterative development. *CROSSTALK The Journal of Defense Software Engineering*, pages 27–30, May 2008.

Sean Follmer, David Carr, Emily Lovell, and Hiroshi Ishii. Copycad: remixing physical objects with copy and paste from the real world. In *Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 381–382, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0462-7. doi: http://doi.acm.org/10.1145/1866218.1866230. URL `http://doi.acm.org/10.1145/1866218.1866230`.

M. Fowler and K. Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

N. Gershenfeld. *Fab: the coming revolution on your desktop–from personal computers to personal fabrication*. Basic Books, 2007.

Björn Hartmann, Meredith Ringel Morris, Hrvoje Benko, and Andrew D. Wilson. Pictionaire: supporting collaborative design work by integrating physical and dig-

ital artifacts. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, CSCW '10, pages 421–424, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-795-0. doi: http://doi.acm.org/10.1145/1718918.1718989. URL `http://doi.acm.org/10.1145/1718918.1718989`.

Hewlett-Packard. *The HP-GL/2 and HP RTL Reference Guide: A Handbook for Program Developers*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997. ISBN 0201310147.

L. McLaughlin. Line printer daemon protocol, 1990.

Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *CUPS: Printer (computing), Unix- like, Operating system, Server (computing), Client (computing), Spooling, Internet Printing Protocol, Command- line interface, Line Printer Daemon protocol*. Alpha Press, 2009. ISBN 6130225768, 9786130225766.

Oberg, Horton, and Jones Sr. *Machinerys Handbook*. Industrial Press, 2004.

Tim O'Reilly. Lessons from open-source software development. *Commun. ACM*, 42:32–37, April 1999. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/299157.299164. URL `http://doi.acm.org/10.1145/299157.299164`.

Norman E. Smith. *Developers Guide to HP Printers: With Disk*. Wordware Publishing Inc., Plano, TX, USA, 1998. ISBN 1556226039.

B. Vaupotic, M. Kovacic, M. Ficko, and J. Balic. Concept of automatic programming of nc machine for metal plate cutting by genetic algorithm method. *Journal of Achievements in Materials and Manufacturing Engineering*, 14(1-2): 131–139, 2006.

Index

Typeset September 29, 2011