



Master of Science HES-SO in Engineering
Information Technology orientation

Cracked records with 3D/IRENE

Supported by the Lawrence Berkeley National Laboratory (LBNL)

Under the supervision of

Carl HABER & Earl CORNELL

Written by

Jérémy SINGY

Under the direction of

Frédéric BAPST & Ottar JOHNSEN

Expert

Noé LUTZ

Berkeley, February 8, 2013

Accepted by HES-SO//Master (Lausanne, Switzerland) on a proposal from
Prof. Frédéric BAPST, Master thesis Supervisor

Berkeley, February 8, 2013

Supervisor:
Jacques BAPST

Head of MRU:
Omar ABOU KHALED

Abstract

Recovery of old records using optical technologies has been done for years by Carl Haber and his team at Lawrence Berkeley National Laboratory (LBNL). So far, a lot of them have already been restored using 2D or 3D scanning for the acquisition and specialized processing programs to get the original sound from the acquired material. However, some of these records are heavily damaged making them difficult to process. A common issue is when the lacquer layer shrinks resulting in so-called cracked records. This project aims to find solutions in order to process different types of damaged records, whether by improving user input and control or by proposing a more automatic way. The results have shown that a fully-automated processing is hardly practical on a heterogeneous set of records but works in specific situations, though still relying on human decision for matchings. For the most difficult cases, the user may rely on improved manual features.

Keywords: Image and sound processing, Gramophone records, LBNL, Programming

Contents

Acknowledgments	xi
List of Figures	xiii
List of Tables	xvii
List of Listings	xix
1 Introduction	1
1.1 Context	1
1.2 Project goals	2
1.3 Report structure	3
2 Phonograph cylinders and records	5
2.1 History	5
2.2 Basic principle	6
2.2.1 Edison phonograph	6
2.2.2 Record types	7
2.3 Summary	8
3 Acquisition and processing systems	9
3.1 Acquisition hardware	9
3.1.1 Components	9
3.1.2 Z-correction	11
3.2 Mapping characteristics	11
3.2.1 Disc mapping	11
3.2.2 Cylinder mapping	13
3.3 Binned representation	13
3.4 IRENE	14
3.4.1 Acquisition	14
3.4.2 Processing	16
3.4.3 Architecture	17
3.5 3D Probe	18

3.5.1	Acquisition	18
3.5.2	Processing	20
3.5.3	Architecture	21
3.6	Typical record issues	22
3.6.1	Shifting	23
3.6.2	Focus	23
3.6.3	Other issues	24
3.7	Sample recordings	24
3.7.1	Graham Bell disc	24
3.7.2	Dickson cylinder	25
3.7.3	Boas recording	26
3.7.4	Brigance recording	27
3.7.5	Conclusion	28
3.8	Summary	28
4	Manual tracking	29
4.1	Current implementation	29
4.1.1	Limitations	29
4.1.2	Suggestions for improvement	31
4.1.3	Summary	32
4.2	Interactive tracking	32
4.2.1	Main features	32
4.2.2	Graphical design	34
4.2.3	Bitmap creation	36
4.2.4	Drawing	37
4.2.5	Groove center approximation	40
4.2.6	Managing tracked points	41
4.2.7	Assisted tracking	41
4.2.8	Design and classes organization	41
4.2.9	Adaptation for RENE	42
4.3	Finding grooves	44
4.3.1	Finding groove bottoms	45
4.3.2	Preparing data and filtering results	47
4.4	Summary	49
5	Towards an automatic processing	51
5.1	Existing tracking methods	51
5.1.1	Tracking line	51
5.1.2	Tracking depth	52
5.1.3	Tracking Fourier	52
5.1.4	Conclusion	52
5.2	Chunks detection	53
5.2.1	Cracked records characteristics	53
5.2.2	Filtering	55

5.2.3	Segmentation	57
5.2.4	Cleaning	60
5.2.5	Contours extraction	61
5.2.6	Tests and results	62
5.2.7	Design and integration into PRISM	65
5.3	Establishing frontiers	66
5.3.1	Difference with VisualAudio	66
5.3.2	Goal of this step	67
5.3.3	Frontier extraction	68
5.3.4	Interpolation	71
5.3.5	Design	74
5.4	Groove detection	75
5.4.1	Preparing the binned image	75
5.4.2	Creating signal	77
5.4.3	Finding grooves	77
5.4.4	Design and class organization	77
5.5	Groove tracking	78
5.5.1	Differences from current tracking	78
5.5.2	Tracking overview	78
5.5.3	Finding center	80
5.5.4	Tracking using depth	80
5.5.5	Tracking using curve fitting	80
5.5.6	Stop condition	83
5.5.7	Design	85
5.6	Linking and building tracks	85
5.6.1	Frontier linking	86
5.6.2	Audio characteristics	91
5.6.3	Audio comparison	97
5.6.4	Track building	100
5.7	Integration and other considerations	103
5.7.1	Integration in the existing software	103
5.7.2	User configuration	103
5.7.3	Drawing result	103
6	Tests on sample recordings	107
6.1	Interactive tracking	107
6.1.1	Dickson cylinder	107
6.1.2	Boas recording	108
6.1.3	Brigance recording	109
6.1.4	Summary	109
6.2	Automatic tracking	109
6.2.1	Bell disc	110
6.2.2	Dickson cylinder	111

6.2.3	Boas recording	112
6.3	Conclusion	112
7	Conclusion	113
7.1	Objectives review	113
7.1.1	Analysis	113
7.1.2	Implementation	113
7.1.3	Other	114
7.2	Future work and improvements	114
7.2.1	Interactive tracking	114
7.2.2	Cracks and groove detection	115
7.2.3	Tracking in chunks	115
7.2.4	Linking and audio analysis	115
7.3	Personal conclusion	116
	Glossary	117
	Bibliography	119
A	External files structure	121
B	Interactive Tracking User Manual	123
1	Introduction	123
2	General description	123
2.1	User interface overview	123
2.2	Description of controls	124
3	Loading a new acquisition	126
3.1	Contrast normalization	126
4	Tracking interactively	126
4.1	Piloting the tracking	126
4.2	On the binned image	127
4.3	Starting a new revolution	128
4.4	Letting the program do it	128
4.5	Correcting an error	129
4.6	Copying an existing pattern	129
5	Starting record processing	130
6	Other considerations	130
6.1	Saving and loading tracking	130
6.2	Tracking direction	131
7	Interactive tracking in RENE	131
C	Automatic Processing User Manual	133
1	Introduction	133
2	Overview of the system	133

2.1	Typical use cases	134
2.2	Configuration panel overview	134
3	Starting the cracks detection	135
4	Setting up the cracks detection	136
5	Groove tracking	137
5.1	Check groove detection	138
5.2	Improve tracking	139
6	Groove linking	139
6.1	Linking options	139
6.2	Linking review and processing	140
7	Conclusion	142
D	Requirements	143
1	Introduction	143
2	Context	143
3	Objectives	144
3.1	Main objectives	144
3.2	Secondary objectives	144
4	Tasks	144
4.1	Analysis	144
4.2	Design/Implementation	145
4.3	Tests	145
4.4	Documentation	145
5	Planning	145
5.1	Important dates	145
5.2	Gantt Chart	146
6	Initial specifications	148

Acknowledgments

Firstly, I would like to thank my responsible professors, Messrs. Ottar Johnsen and Frédéric Bapst. They made possible this unique opportunity of a great experience, working for almost six months in the famous Berkeley Lab in the USA. Moreover, Mr. Johnsen was a precious help for technical difficulties in the signal processing field, while Mr. Bapst kept a careful look at the project and at my numerous typos in this report.

A special thank goes to Carl Haber, without whom this thesis would not have been possible, and who brought ideas and suggestions when I was in a deadlock. Another thanks to Earl Cornell, who brought a lot of help on technical details. He was always open for an interesting conversation about the implementation of IRENE.

I am also grateful to the Hirschmann Foundation for their generous fellowship which was a great help for my life in the USA.

Finally, many thanks to my parents, family and friends who supported me in this experience and allowed me to keep in touch with my homeland during the semester.

List of Figures

1.1	An example of cracked disc.	2
2.1	Pictures of Edison wax phonograph and record.	6
2.2	Schema of a simple phonograph.	7
2.3	The two types of groove modulation.	8
3.1	The acquisition hardware to acquire discs as installed at LBNL.	10
3.2	The acquisition hardware to acquire cylinders as installed at LBNL.	10
3.3	Schema representing mapping of a flat disc.	12
3.4	Example of a mapped entire revolution acquisition with off-axis.	12
3.5	Schema representing mapping of a cylinder.	13
3.6	IRENE capture excerpt.	14
3.7	Visualization of scanned grooves.	15
3.8	Coaxial illumination, light emission and reflection.	15
3.9	The IRENE user interface.	16
3.10	General architecture of RENE.	18
3.11	Chromatic aberration through a lens.	19
3.12	The PRISM user interface.	21
3.13	General architecture of PRISM.	22
3.14	Grooves shifted because of a crack.	23
3.15	Blurred grooves on the bottom part under the crack.	24
3.16	The Graham Bell 287700 disc.	25
3.17	A crack on the 287700 disc as it appears in PRISM (detail image).	26
3.18	Damages on the Dickson cylinder.	26
3.19	Excerpt of the acquisition of the 0724 cylinder.	27
3.20	The Brigrance disc as viewed from the RENE program.	28
4.1	Manual tracking example in PRISM.	30
4.2	General workflow of a whole processing using PRISM or IRENE.	30
4.3	Interactive tracking synchronized with the global view.	33
4.4	Example of the groove pattern cloning.	35
4.5	Interactive tracking panel in PRISM.	36
4.6	Illustration of global and local contrast adjustment on the Dickson cylinder.	37

4.7	Panel and record coordinate systems in the interactive tracking panel. . . .	38
4.8	Class diagram for the interactive tracking in PRISM.	42
4.9	Screenshot of the interactive tracking working in RENE.	43
4.10	Groove highlighting as implemented in PRISM.	45
4.11	Result of the peak detection algorithm.	46
5.1	Workflow representing a whole processing using the automatic tracking. . .	53
5.2	A crack in the Bell record visualized as depth and brightness intensity. . .	54
5.3	Visualization of the two-dimensional Gaussian with $\sigma = 0.8$	56
5.4	Visualization of the different filtering methods.	57
5.5	Histogram of a portion of the cracked record.	58
5.6	Result of the thresholding operation on a portion of a cracked record. . .	59
5.7	Result of the brightness normalization in PRISM.	60
5.8	Five passes closing to eliminate black pixels in the chunks.	61
5.9	Illustration of the contour detection and approximation.	62
5.10	Screenshot of the cracks detection test program.	63
5.11	A very thin crack not entirely detected on the bell disc.	64
5.12	Class diagram for chunk detection.	66
5.13	Visualization of a non-convex chunk.	67
5.14	The chunk with highlighted frontiers.	68
5.15	Frontier detection based on horizontal variation.	69
5.16	Relation between peak and frontier detection.	70
5.17	Class diagram for frontier extraction.	74
5.18	Schema representing the groove detection on each frontiers of a chunk. . .	75
5.19	Illustration of the result using linear regression for slope correction. . .	76
5.20	Class diagram for groove detection.	77
5.21	Tracking being applied on a frontier of a chunk.	79
5.22	Example of accuracy problem with the tracking depth method.	81
5.23	Example of tracking using curve fitting.	82
5.24	Comparison of tracking methods.	82
5.25	Examples of problems with curve fitting method.	83
5.26	Illustration of the stop condition.	84
5.27	Class diagram for groove tracking.	85
5.28	Schema representing frontiers linking.	86
5.29	Example of the result after frontier linking.	88
5.30	Class diagram for frontier linking.	90
5.31	Illustration of outliers removing on a sample signal (one big outlier). . .	92
5.32	Comparison of audio analysis methods.	94
5.33	Example of high-pass filter applied on a distorted surface.	95
5.34	Comparison of FFT with the use of different window functions.	96
5.35	Different examples of shifting leading in different paths.	98
5.36	Class diagram for audio characteristics computing and comparison. . . .	100
5.37	An example of cracked record tracked.	102

5.38	Class diagram for groove linking.	102
5.39	Class diagram representing the automated processing integration.	104
5.40	Tracking, cracks and groove bottoms as drawn in PRISM.	105
6.1	Test of interactive tracking on the Dickson cylinder.	108
6.2	Test of interactive tracking on the Boas recording.	108
6.3	Test of interactive tracking on the Brigance recording.	109
6.4	Examples of grooves not detected on a frontier.	110
6.5	Tracking of the Bell disc using automatic tracking.	111
6.6	Tracking of the Dickson cylinder using automatic tracking.	111
6.7	Tracking of the Boas cylinder using automatic tracking.	112
B.1	Location of the interactive panel in PRISM.	124
B.2	Visualization of the controls in the interactive panel.	125
B.3	Example of starting position.	127
B.4	A groove being tracked.	128
B.5	Example when another revolution as started.	129
B.6	Example of the same pattern copied for each revolutions.	130
B.7	Location of the interactive panel in RENE.	131
C.1	Visualization of the controls in the <i>Cracks</i> panel.	135
C.2	Example after loading a record using automatic tracking for cracked records.	136
C.3	Chunk detection visualization.	137
C.4	Example of missing groove as seen on the detailed image.	138
C.5	Controls related to groove linking on the detailed view. The	140
C.6	Multiple tracks as seen on the binned view.	141
D.1	Gantt Chart for the entire schedule of the thesis.	147

List of Tables

4.1	Transformation of the different objects in the interactive panel.	40
B.1	Explanation of the different objects in the interactive panel.	125
C.1	Explanation of the main parts in the <i>Cracks</i> panel.	134

List of Listings

4.1	Interactive panel drawing and coordinate systems transformation.	39
4.2	Pseudocode of the peak detection algorithm.	46
4.3	Pseudocode of the row normalization function.	48
5.1	Pseudo for the frontier extraction, using the peak detection algorithm. . . .	70
5.2	Basic linear interpolation algorithm.	72
5.3	Linear interpolation of points as implemented into PRISM.	73
5.4	Main algorithm for tracking.	79
5.5	Pseudocode representing localization of the end position and frontier. . . .	84
5.6	General algorithm for frontier linking.	88
5.7	Algorithm for finding closest end frontier from a given groove.	89
5.8	Algorithm for finding the closest groove.	90
5.9	Algorithm to remove outliers.	91
5.10	Audio analysis using curve fitting.	93
5.11	Algorithm for final track building.	100

Chapter 1

Introduction

1.1 Context

For several years, Carl Haber and his team at Lawrence Berkeley National Laboratory (LBNL) have been working on the sound recovery of old mechanical records. They developed systems able to recover the sound from mediums using optical technologies, without any physical contact. These solutions are able to process many different record types, from early Edison cylinders to vinyl discs, including shellac or lacquer composed discs and others.

Nowadays, a lot of unique records remain in archives stored in different libraries, museums and academic institutions, containing materials of important historical and cultural value. Since most of them are recorded on aging and deteriorated mediums, it is often too delicate to play them with a normal mechanical phonograph. Non-contact feature of optical methods is then important.

The team at LBNL first started to develop a system called IRENE. It enables to extract the sound using digitized images of a disc. The installation for acquisition is completed by a software package that processes the extraction step, called RENE. Some time later, the development of a new project called 3D Probe started. Instead of capturing 2D images, the acquisition uses a special probe able to measure the real depth on the surface of the scanned record. This new system has enabled to process other types of records such as cylinders and is able to offer a better outcome with some specific mediums.

Even earlier in Switzerland, at the College of Engineering and Architecture of Fribourg, another project called VisualAudio has been set up. It aims to archive discs that are deteriorating in a durable way so that the recordings can be heard years later. The discs are stored on a photographic film which is a support with a long life span. An application has also been developed, enabling to extract the sound using non-contact optical scanning in a way similar to IRENE.

1.2 Project goals

Old records can suffer from degradations. One of the most common is known as “cracked discs”. This problem appears when the lacquer coating shrinks as the disk is getting older. This causes cracks where the underneath hard support is visible, as seen in Figure 1.1. Some records are also entirely broken into pieces. The acquisition is then performed in several steps by strapping the pieces together. The result is also some discontinuous grooves between the different pieces.



Figure 1.1 *An example of cracked disc.*

Up to now, IRENE and 3D Probe cannot automatically process such damaged records, because the groove traces are sometimes widely separated and may be shifted from each other. Though a special feature is implemented, enabling the user to manually track the traces, it is not a perfect tool for practical use. Moreover, even with such a feature, it could be really difficult to visually find the correct shifting when the traces look similar.

The aim of this Master’s thesis is to find ways to correctly read these kinds of degraded records with the solutions developed at LBNL by Carl Haber and his team. In a first step, an improvement of the manual tracking will be implemented. It is still useful to keep it for some cases, e.g for special early recordings or when a disc is heavily cracked.

Then, the next step will be to design and implement a tracking feature able to process

cracked records and link their traces automatically. VisualAudio already implements a feature enabling to track cracked discs in an automatic or semi-automatic way. It can then be taken as an example, though it will obviously not be possible to directly adopt its implementation, as the systems are quite different.

1.3 Report structure

This report is separated into three main sections. The first one corresponds to the analysis performed in order to have a good understanding of the subject before starting on the actual project. Chapter 2 details the important points to know about phonograph records, history as well as the main techniques implemented between the 19th and the 20th century. Then, Chapter 3 will presents the current solutions developed at LBNL. Both the 2D and 3D acquisition systems will be presented, as well as the processing part involving the different softwares.

The second part will present the main work realized during the thesis. Chapter 4 will detail the improvements to help the user for the restoration of profoundly damaged records, using a manual solution. Then, the implementation of a prototype able to automatically process damaged records in certain circumstances while be presented in Chapter 5.

The general results and tests with different records will be presented in Chapter 6. Finally, Chapter 7 will summarize the objectives and results, explain the remaining problems and propose further work related to the subject.

Chapter 2

Phonograph cylinders and records

This chapter will give a quick historical overview of the different mechanical phonograph technologies and the types of mediums encountered. It will also explain the main principles behind the analog recording.

Some information, mainly historical, is taken from [10], [12] and [13].

2.1 History

The first known sound recording was realized by the French printer Édouard-Léon Scott de Martinville in 1860. However, the used device called the *phonautograph* was not able to playback the recordings. It rather acted as an early mechanical oscilloscope and was later used by scientists to study visual representations of the sound.

The phonograph was created by Thomas Edison in 1877. It was the first device able to record *and* reproduce the sound. The general principle was to mechanically engrave the audio signal on a rotating cylinder. The resulting groove undulated then vertically. Then, for the playback, a stylus (needle) retraced the groove and was able to reproduce the sound waves from the generated vibrations. Finally, the signal was efficiently coupled to the room environment with a horn so that the recording was more audible. A newer version of this phonograph and an example of cylinder are presented in Figure 2.1.

Since then, the phonograph has been used as the primary device for sound reproduction until around the 1950's when the magnetic tape became widely available. From the Edison's first version, a lot of improvements have been made. Edison used first a tinfoil around the cylinders into which the groove was embossed. However, this material gave bad quality and was rapidly degrading. Alexander Graham Bell started to use wax-coated cylinders and discs instead of the tinfoil, which improved a lot the sound quality.



(a) An Edison phonograph.



(b) An Edison wax-cylinder.

Figure 2.1 Pictures of Edison wax phonograph and record.

Then, in the early twentieth century, the cylinders were gradually replaced by the gramophone records which were flat discs that could be double-sided. On the latter, the groove undulated laterally instead of vertically.

After the wax-cylinders, a lot of different materials have been used for the discs covering. For example, the shellac discs became the first widely used medium. Some discs were also using a special lacquer coat. Finally, the vinyl disc is one of the last types of gramophone record and is still being used today. The different existing recording types will be further explained in Subsection 2.2.2

2.2 Basic principle

2.2.1 Edison phonograph

With the early Edison phonograph [4], the same device was used to record and play the sound. As already explained, the sound is mechanically embossed with a needle on a tinfoil placed around the cylinder. In fact, the needle is connected to a diaphragm that vibrates when audio waves are emitted. The cylinder is put on a mechanism which can be rotated by a crank and that shifts slowly resulting in helical grooves. To improve acoustic match, a horn (tube) is used, as represented in Figure 2.2.

Then, to reproduce the recorded sound, the cylinder is repositioned at its starting position. In the original Edison phonograph, there is in fact another needle on the other side. This needle is connected to a more sensitive diaphragm so that when the crank is turned, the needle follows the previously embossed groove which causes the diaphragm to vibrate and returns the sound waves through the tube. This is then approximately the inverse process as the recording.

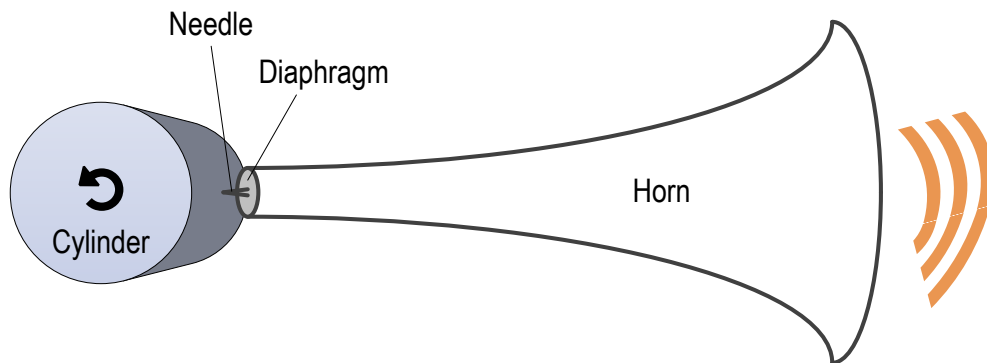


Figure 2.2 *Schema of a simple phonograph.*

2.2.2 Record types

Direct vs. stamped recordings

The first important distinction is the way records are recorded. This also defines the different used materials. Firstly, the records made for direct recording are made to be recorded one time and play back directly with the adapted phonograph. This category is the most important for recovering nowadays because they are originally no other copies available. It includes the following record types:

- Wax cylinders and discs
- Lacquer (also called acetate discs)
- Aluminum discs
- Plastic belt (e.g. used by the Dictabelt recorder)

The other main category are the stamped (or molded) records. These are the recordings typically encountered on the market. The original recording is firstly molded on a master, and a lot of different copies can be created by stamping on other discs. The typical examples are the following:

- Shellac records
- Vinyl records

Groove modulation

The Edison phonograph uses a groove that undulates vertically. However, a lot of records, particularly the discs, have later used laterally modulated grooves. This way, it is no more the depth that determines the signal, but rather the lateral position of the groove bottom. The difference can be clearly seen in Figure 2.3.

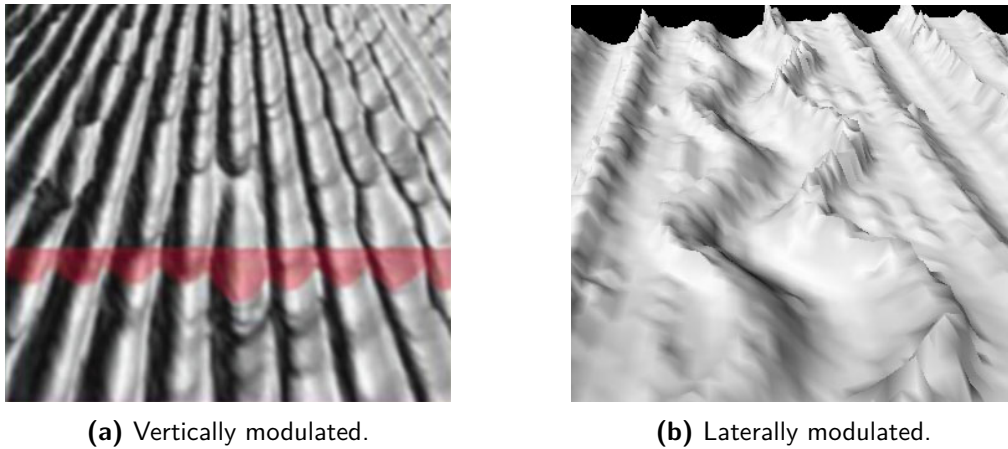


Figure 2.3 *The two types of groove modulation.*

It exhibits a 3D representation of record surfaces for both groove types. In Figure 2.3a the boundaries are straight and the depth varies while in Figure 2.3b they form an horizontal wave.

Revolution per minute

Another important parameter is the number of revolutions per minute, abbreviated *rpm*. It represents the rotational speed of the record while recording or playing. The speed of early records was not standardized, and it could vary from 60 to 160 rpm for cylinders. Then, the speed has been progressively standardized to the common 78 rpm. A lot of recording material to be restored now uses this format. The newer vinyl discs have then used the well-known speeds of $33\frac{1}{3}$ rpm and 45 rpm.

2.3 Summary

This chapter gave an idea of the phonograph history and different devices, showing the heterogeneous nature of the audio recording, with a lot of different techniques and materials. This outlines the difficulties that might appear while trying to recover old recordings.

The next chapter will discuss the technology developed at LBNL in order to extract optically the sound from old records.

Chapter 3

Acquisition and processing systems

This chapter details the systems used in the audio laboratory at LBNL to extract records. For both systems, the main process is separated in two main steps: the acquisition and the processing. The first section is an overview of the setup and hardware used to acquire records, followed by the characteristics of this acquisition step. The last two sections present the specificities of each system, both about acquisition and processing steps.

Some information is taken from previous thesis or technical report realized at LBNL, such as [1], [6] and [5].

3.1 Acquisition hardware

Both IRENE and 3D Probe use the same hardware for the acquisition. It consists of different sensors and motors connected to a controller. This controller is itself driven by a software, using a special library embedded in a LabVIEW environment.

3.1.1 Components

There are two main different supports. The first one is a turntable specifically designed for flat discs with the scanner put vertically and the disc rotating horizontally. It is viewed in Figure 3.1 as installed with the 2D camera .

The second one enables cylinder scanning, the probe being placed horizontally and directly facing the record. It can be represented in Figure 3.2 as installed in the laboratory, with the 3D probe.

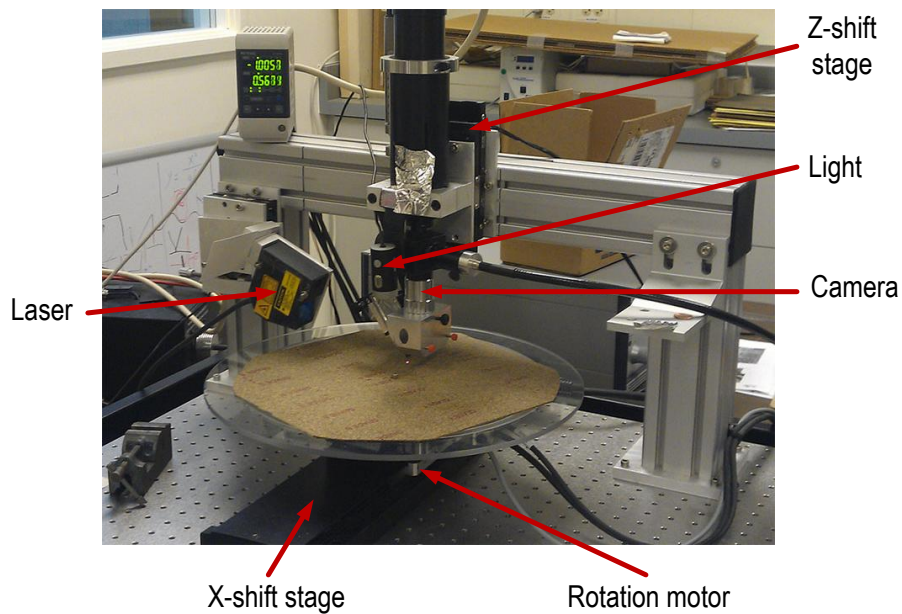


Figure 3.1 *The acquisition hardware to acquire discs as installed at LBNL.*

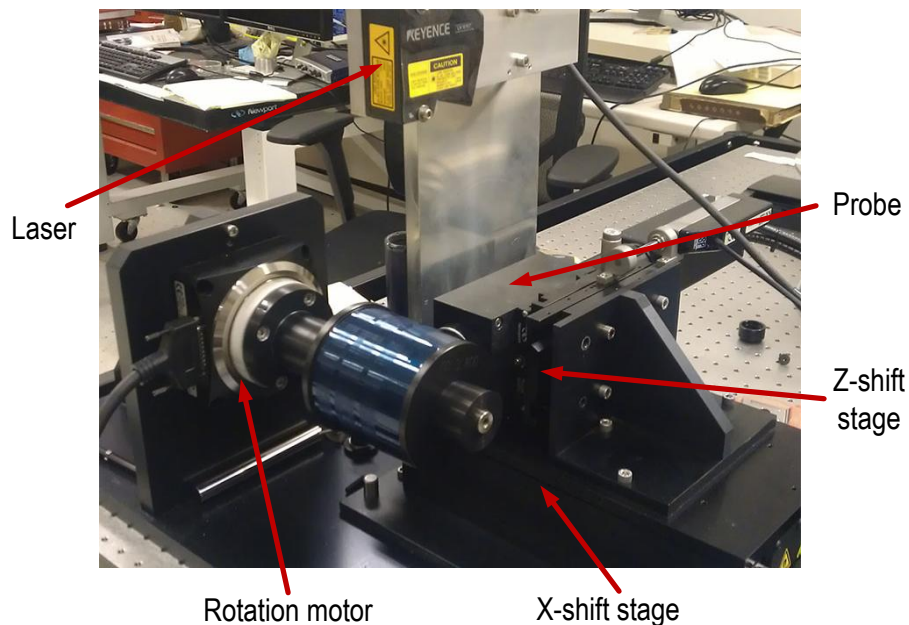


Figure 3.2 *The acquisition hardware to acquire cylinders as installed at LBNL.*

In both cases, in addition to the rotary motor, there is a X- and Z- shifting motor. The X-motor enables to shift the record to scan another portion. The Z-motor is used to adapt the sensor height for a correct focus.

3.1.2 Z-correction

Small differences can cause the 2D image to be blurred or the 3D scan to be out of range. To avoid this problem, a Z-shift correction is applied. The height shifting is controlled using a laser displacement sensor. It follows up the record and adjusts the sensor height, acting as an “autofocus system”.

3.2 Mapping characteristics

Before explaining the specificities of each system, it is noteworthy to explain the basic concepts about record acquisition used in both 2D and 3D systems. The most important is to understand the mapping of a record onto a 2D image. This characteristic is slightly different from the two categories, cylinders or discs.

3.2.1 Disc mapping

As previously seen in Figure 3.1, the setup for flat discs acquisition looks similar to an ordinary phonograph. The disc is placed on the turntable and rotates while the sensor takes a single line of capture at regular intervals.

The first main difference is that an entire revolution consists of a sequence of lines, each of which mapped in a single groove profile. The line width and other properties depend on the sensor and optics and will be detailed in the next sections.

Since each captured line are part of a sequence, an entire revolution finally results in a ring-shaped capture which is mapped in a rectangular image. The groove then maps to straight parallel lines instead of the original spiral. This mapping is more clearly seen in Figure 3.3.

It is important to note that as the mapping represents a revolution, the resulting *top* and *bottom* of the acquisition are in fact at the same position. As the groove is embedded as a spiral onto the disc surface, the result is not exactly vertical but slightly inclined. The groove section ending at the image bottom continues at the same horizontal position at the top (or vice versa depending on the capture direction). A radial line maps to a line (almost) perpendicular to the grooves

Off-axis problem

This section explained the general idea of mapping. However, in practical situations, the groove shape is rarely so straight on the mapped image. Instead, it results in waved repeated sections, as seen in Figure 3.4.

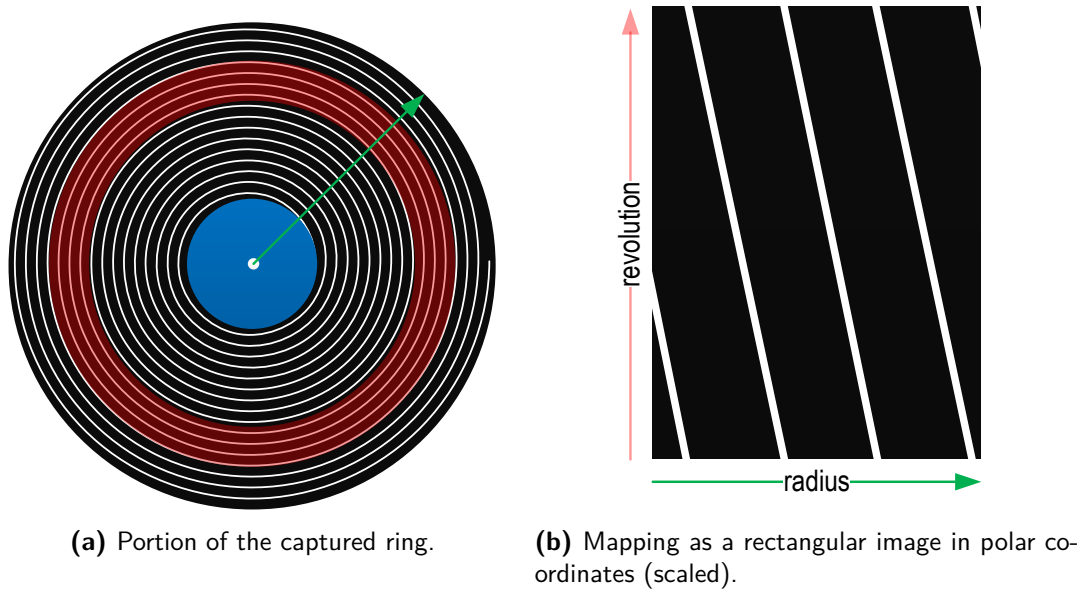


Figure 3.3 Schema representing mapping of a flat disc. The example groove is enlarged a lot from reality for a proper visualization. The right image corresponds to the red portion on the disc.

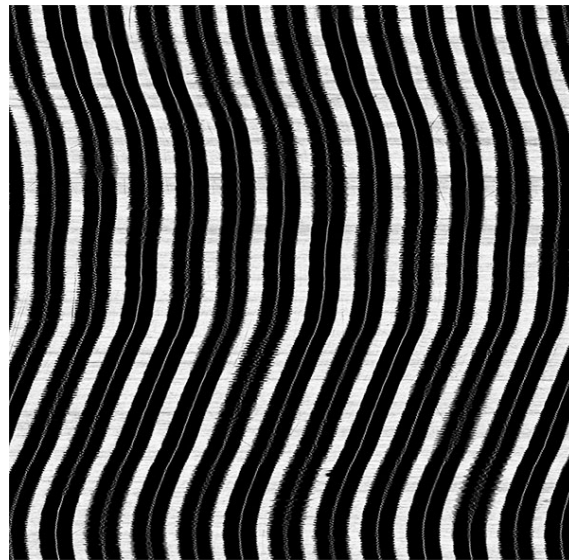


Figure 3.4 Example of a mapped entire revolution acquisition with off-axis.

This result comes from the fact that the disc is hardly *exactly* put at the center of the turn table. As a groove is in reality very tight, a small off-axis of e.g. 0.5 mm is already clearly visible on a capture of a whole revolution.

However, the distortion is only problematic for visualization. The resulting sound is not strongly affected, even for laterally-modulated records because the frequency is very low compared to the actual recorded signal. Moreover, the original signal is differentiated, making the off-axis insignificant.

3.2.2 Cylinder mapping

Because of the geometrical shape, the cylinder mapping is slightly different. Again, the cylinder rotates in a way similar to the original Edison phonograph, while the sensor captures lines. A whole revolution gives an image equivalent to the disc mapping, as seen in Figure 3.5.

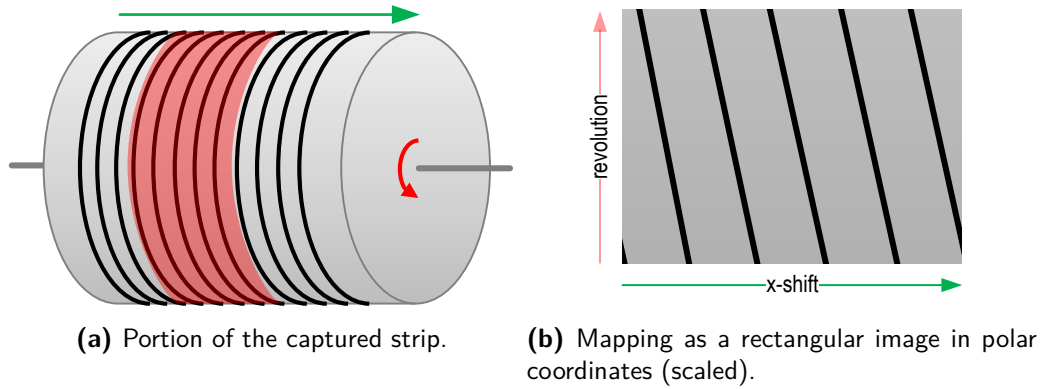


Figure 3.5 Schema representing mapping of a cylinder. The example groove is enlarged a lot from reality for a proper visualization. The right image corresponds to the red portion on the record.

This time, the mapped representation corresponds to an unrolled cylinder strip. Another way of viewing this is if we unwrap the tinfoil around an Edison cylinder and look at it unfolded. This time, a line perpendicular to the grooves maps also to a perpendicular line on acquired image, and a vertical line remains vertical. The Y coordinates corresponds to the rotation angle.

3.3 Binned representation

A record complete acquisition gives an image in very high resolution, so that the sample rate is big enough ¹. It is often useful to have a more general view of the record. Particularly, to view an entire revolution, all points in the vertical axis are not useful. Both

¹The complete discussion about sample rate is given in the corresponding next sections for IRENE and 3D Probe.

systems create then a *binned* image, which corresponds to the acquired image with the height divided by a certain factor d , typically around 20–100 that can be changed according to the original resolution. It is created by averaging the value of the actual pixel values every d lines. The result is an image such as the one already viewed in Figure 3.4.

In addition to the visualization, this representation will be very useful for a step of the processing algorithm, which will be explained in the next sections.

3.4 IRENE

3.4.1 Acquisition

As already explained, IRENE uses numerical 2D capture for the acquisition. The scanner takes some monochromatic pictures of the disc. The resulting image is influenced by the shape of the surface because of the light reflection. This enables to visually find the grooves on the resulting picture. An example of such a picture is shown in Figure 3.6.

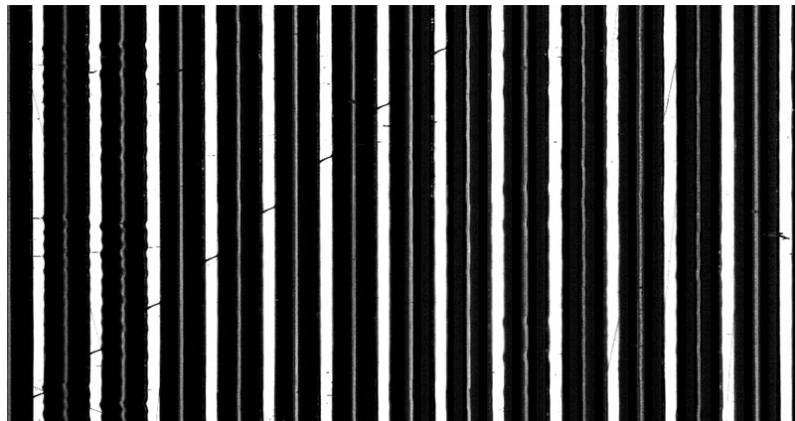


Figure 3.6 *IRENE capture excerpt.*

A single groove can be seen as two black strips with a thin bright line in between. The black parts represent the edges while the thin line is the bottom of the groove. Between two grooves, the interval also appears in white. Figure 3.7 describes the different parts.

Figure 3.7a represents two grooves as they appear from the acquisition part. The approximate corresponding shape in a sectional view is visualized in Figure 3.7b.

Coaxial illumination

The system uses coaxial illumination. Every light beam is parallel and orthogonal to the record surface. Therefore, the intensity of the resulting pixels will finally depend on the slope of the surface, as visualized in Figure 3.8.

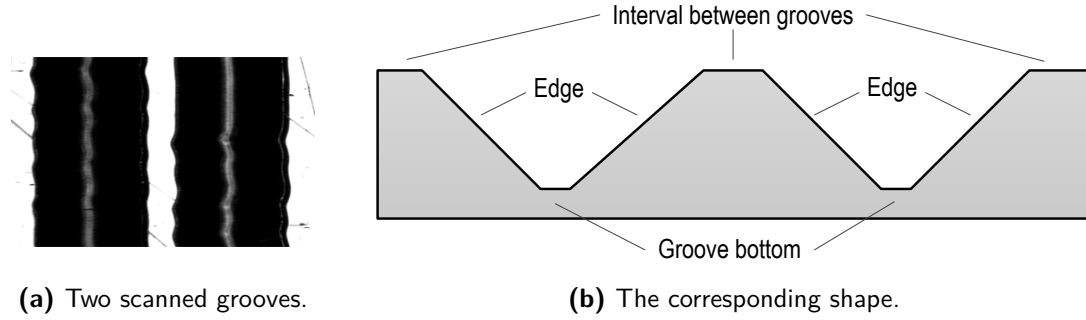


Figure 3.7 Visualization of scanned grooves.

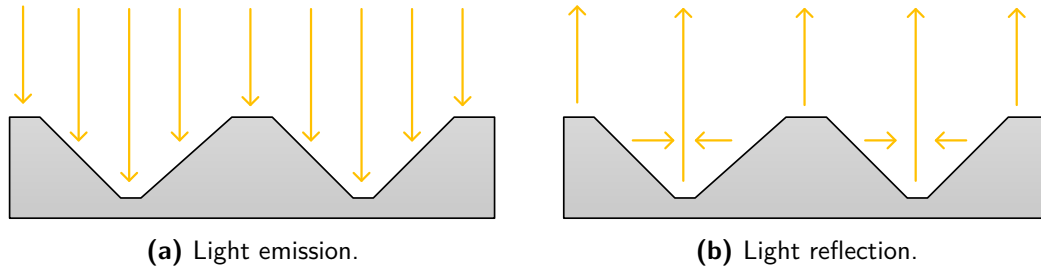


Figure 3.8 Coaxial illumination, light emission and reflection. The light is not reflected back to the source on groove edge, resulting in black parts on the acquired image.

On the exact groove bottom, the slope is null and then the light is orthogonally reflected. This is also valid for the interval between two grooves, as the surface is almost flat. On the other hand, the groove edges are very sloping and the light is not reflected directly to the source. The corresponding pixels will then appear black.

Capture properties

The camera cannot take a whole picture of one disc in a snapshot. It captures a line of 3.072 mm with 4096 pixel samples. For a total disc revolution, 80 000 lines are captured this way. Therefore, it results in an image of $4096 \times 80\,000$ px representing a ring-shaped disc capture in polar coordinates. To simplify the manipulation, the high resolution image is exported as eight separate pictures of $4096 \times 10\,000$ px easier to process.

For a 78 rpm disc, the time for one revolution t_{cyc} is the inverse of the rotational speed ω_{cyc} . Therefore, the sampling rate f_s for the system is given by

$$f_s = \frac{n_{sample}}{t_{cyc}} = \frac{n_{sample}}{\frac{1}{\omega_{cyc}}} = \frac{80\,000}{\frac{1}{78 \text{ rpm} \cdot \frac{1}{60} \text{ m s}^{-1}}} \approx 104 \text{ kHz} \quad (3.1)$$

This value satisfies the sampling theorem² as it is greater than the minimum of 40 kHz.

3.4.2 Processing

During the acquisition, the original record is digitized and can be archived on digital media. To extract the sound, the next step is to use the RENE program. It offers features to process the record images and output the final corresponding sound as a WAV file.

RENE embeds a lot of algorithms and parameters to adapt at best the processing to the different record types. It is written in the C[#] programming language, enabling to easily build a complete graphical application without the use of third-party libraries. The GUI simplifies its utilization and offers the possibility to visually follow the tracking, as seen in Figure 3.9.

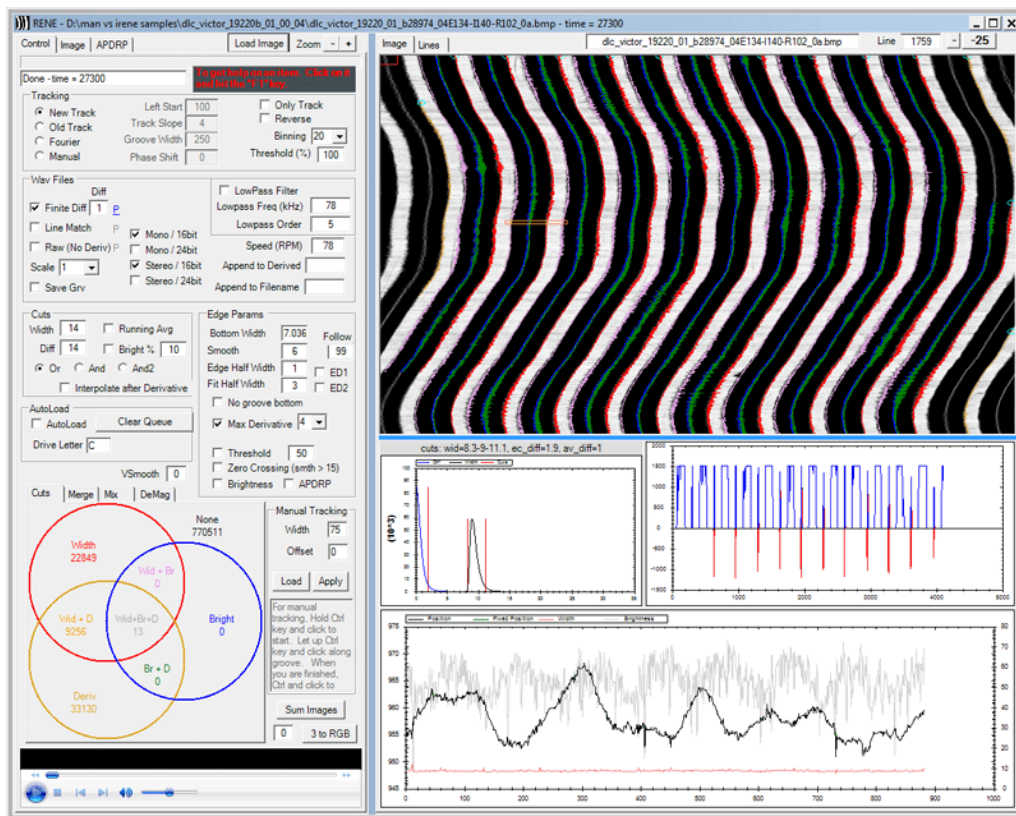


Figure 3.9 The IRENE user interface.

The left panel is used to specify the different parameters and algorithms for the separate

²The Nyquist–Shannon sampling theorem states that the minimum sampling frequency should be greater than twice the maximum of the sampled signal to reconstruct it properly. As the human hearing have a range from 20 to 20 000 Hz, the sample rate for audio signal must be at least 40 kHz.

steps and more generally to control the application. The upper right panel shows the processed record image. When the tracking is applied, the detected edges are directly drawn on top of the image. The panel at the bottom presents diverse information and statistics on the records.

The whole processing is applied in several steps.

Tracking

The tracking is the first step to process a disc. It enables to approximately find the groove positions on a record using different detection algorithms. These latter are performed on the binned image (see Section 3.3). It is actually the one seen in the upper right panel in the GUI (see Figure 3.9).

Due to the small bitmap size, this processing is consequently very efficient and not subject to possible noise or imperfections which are smoothed by the binning. In the user interface, the result of this step can be viewed on the top-panel as the purple and red external lines delimiting the grooves as seen in Figure 3.9.

Several different algorithms (explained in detail in Section 5.1) can be selected to track the grooves. The best one to apply depends on the scanned medium and its general condition. As already mentioned, there is also an option to manually define the groove position when automatic methods are not applicable (e.g. if the disc is too damaged). The user can directly select points that define an interpolated line, which must follow the edge of the groove. This feature will be further explained in Section 4.1.

Image processing

The next step is the actual processing. When the grooves are tracked, it remains to precisely find the groove center, which defines the sound information. Again, different algorithms can be used as methods using the derivative, thresholding, etc.

The result is the precise groove bottom defined between the blue and green traces on Figure 3.9.

Sound processing

Finally, it remains to create the output WAV file from the tracked groove bottom positions. At this step, some filters can also be applied to improve the sound quality.

3.4.3 Architecture

The diagram in Figure 3.10 presents the general architecture of RENE.

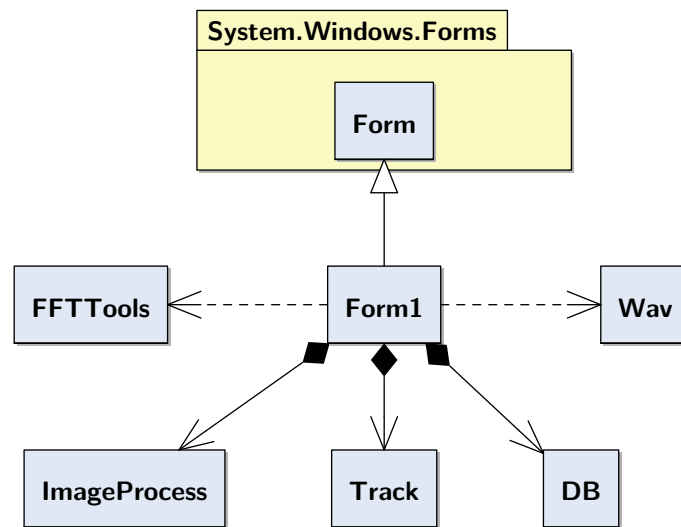


Figure 3.10 *General architecture of RENE.*

The program does not use object-oriented design. The business logic and processing parts are embedded in the `Form1` class which also represents the view of the application (inheriting from the .NET WinForms API).

Dependencies between the different classes are then fairly simple. Some tasks have been separated in external classes contained in the `Form1` class, as for the groove tracking or image processing which includes in fact routines to load images into bitmaps. The `DB` class handles connection and communication with a database, used to automatically store information and statistics on the processing results in a centralized database.

Some common processing tasks are separated in different utility classes, e.g. FFT algorithm or WAV processing to output the sound file.

3.5 3D Probe

3.5.1 Acquisition

With 3D Probe, the acquisition stage is the main difference. Instead of using a regular camera, a special probe is used, getting the actual depth for each point over a scanned part. This gives a real heightmap of the disc surface, as shown in three-dimensional view in Figure 2.3.

To measure the height of the scanned surface, the sensor takes advantage of the chromatic

aberration³. Light is emitted and passes through an objective. Because the refraction index depends on the wavelength, the light is refracted differently regarding the wavelength, as shown in Figure 3.11.

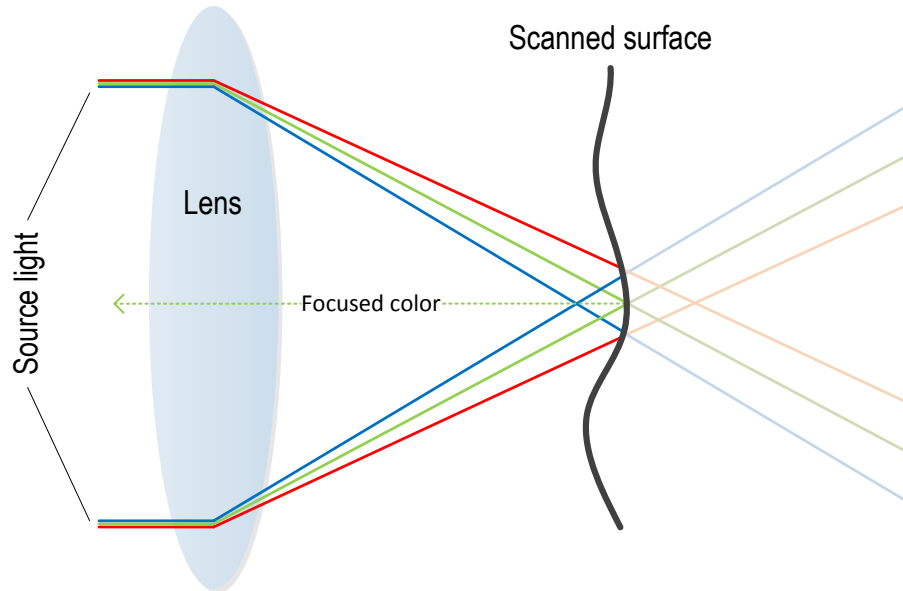


Figure 3.11 *Chromatic aberration through a lens.*

Hence, the color reflecting in focus on the surface will depend on the distance from the lens. The sensor captures then the reflected wavelength and can deduce the height. The probe can also give the value of the brightness, that is, the captured light intensity for a given point. The technical specification from the manufacturer is available in [8].

Capture properties

In a single snapshot, the sensor is able to capture 180 points in one line of 1.8 mm in a way similar to the 2D camera with IRENE. The depth is measured in a range of 400 μm with a resolution of 0.125 μm .

The acquisition software outputs the data in a specific binary file format with extension *.pri*, containing essentially floating-point values corresponding to the depth captures. The brightness is stored separately in a *.bri* file, using exactly the same structure.

In this case, the number of samples for a full revolution depends on the angle α by which the cylinder is rotated between two samples. Therefore, the sampling rate is not fixed and

³Optical distortion occurring when a lens cannot properly focus all colors of a same point due to different refractive indices for different wavelengths of the light.

the formula for a cylinder at 160 rpm become

$$f_s = \frac{n_{sample}}{t_{cyc}} = \frac{n_{sample}}{\frac{1}{\omega_{cyc}}} = \frac{\frac{360^\circ}{\alpha}}{\frac{1}{160 \text{ rpm} \cdot \frac{1}{60} \text{ m s}^{-1}}} \quad (3.2)$$

For example, with an angle α of 0.02° , the number of samples is $\frac{360^\circ}{0.02^\circ} = 18\,000$ and the sampling rate become $\frac{18\,000 \cdot 160}{60} = 48 \text{ kHz}$.

Multiple-pass acquisition

From the capture properties, the distance between two points is $10 \mu\text{m}$. The corresponding resolution is then quite low and could not be enough in some cases.

Therefore, the LabVIEW acquisition software is able to perform a multiple-pass scanning. The same part a record is captured several times with a different shift. For example, by using two-pass, the number of points, and so the resolution are multiplied by two resulting in 360 points for the same distance.

3.5.2 Processing

The processing is done with the program called PRISM. It uses the data stored in the *.pri* and *.bri* files to process and output the sound. The application is very similar to RENE and is also written in C#. There are also a lot of different parameters to tune the tracking and processing at best for the corresponding record type. Some parameters are also specifically related to cylinder processing.

The user interface (Figure 3.12) embeds approximately the same main parts as RENE, though differently disposed.

The parameters and algorithms can be set on the top panel. In the middle and the right, the scanned media is visualized as a 2D heightmap. The dark and bright colors emphasize the higher, respectively the lower points on the surface. The image at the left-hand side is a general view while the other one shows a full resolution portion of the record (selected with the cursor and highlighted with the red square on the left panel). The lower panel gives the same type of information as the other program.

Processing steps

The processing part is performed through the same steps as with RENE (see Subsection 3.4.2). The main point is to take into account that a pixel value now denotes a real height and no more a brightness difference as on a photography. Likewise, the 3D acquisition suffers from bad points, that is, pixels with incorrect height value coming from the

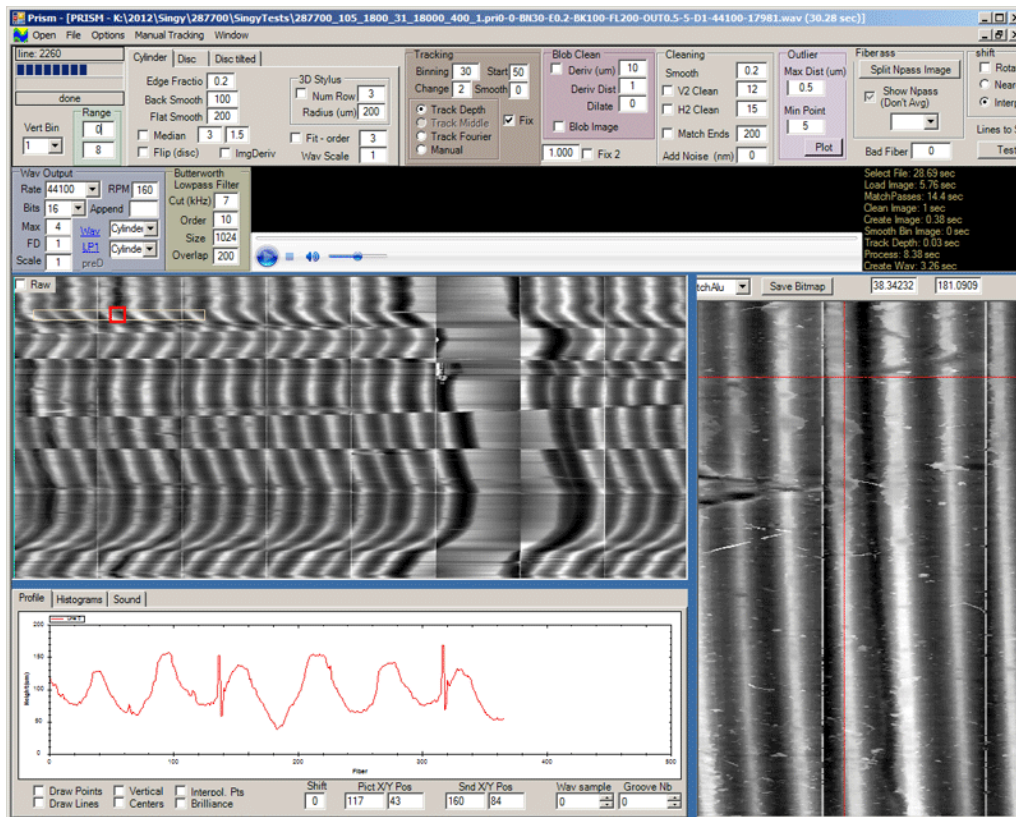


Figure 3.12 The PRISM user interface.

acquisition. Special algorithms are then set up to clean the scanned data and suppress these bad points.

Finally, the cylinder processing algorithm is specific, as it considers of course the vertically engraved nature of these type of record (see Subsection 2.2.2). The waveform is indeed defined by the surface height in the groove bottom (the intensity in the heightmap) and not by its lateral position.

3.5.3 Architecture

The general architecture of PRISM is presented in Figure 3.13.

The PRISM program have a refined architecture, using more object-oriented facilities. The GUI is handled by the `frmMain` class. The big part of the business logic has been separated in its own class `Hardware`. This class represents an abstract record which is processed by the application. A inheritance hierarchy is built from it, mainly separated in `Cylinder` and `Disc` types representing vertically and laterally engraved records. This design enables to specialize the algorithms for any types of records.

3.6.1 Shifting

Visible cracks on a record as shown in Figure 1.1 is only the tip of the iceberg. Usually, it involves other problems to be solved algorithmically. One of the most common problem is shifting. When different parts of a disc surface are separated because of the cracks, they typically move slightly from each other, resulting in grooves mismatch. An example of this kind of issue is illustrated in Figure 3.14.

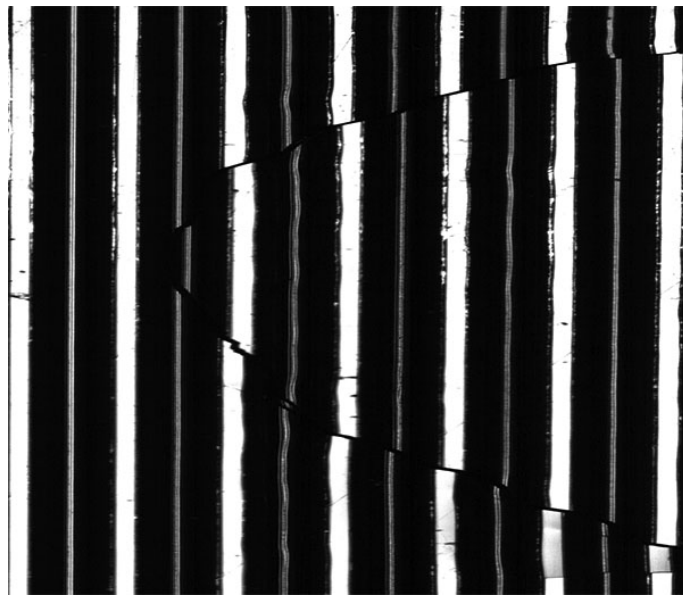


Figure 3.14 *Grooves shifted because of a crack.*

The acquisition shows clearly the two distinct pieces separated by a arch-shaped crack. In this case, the grooves are only slightly shifted. However the difference can be much larger and exceed up to several grooves large, causing difficulty to visually find the correct matching.

3.6.2 Focus

As explained in Subsection 3.1.2, the acquisition needs always to stay in focus to get a sharp image. However, when the height varies abruptly, the laser is not able to directly compensate the Z-shift. This can happen sometimes when the lacquer inflates resulting in bubbles on the surface. The result in the acquisition is shown in Figure 3.15.

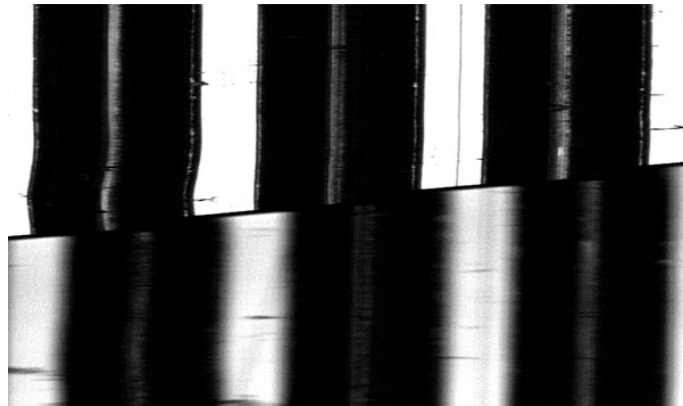


Figure 3.15 *Blurred grooves on the bottom part under the crack.*

3.6.3 Other issues

Other problems may occur because of the cracks. For example, because of the lacquer deformation, the pieces can sometimes expand or shrink, which results in other corrections to apply. Likewise, the gaps introduced by the cracks can sometimes be large and the resulting sound affected. Most of time, the gap is not a loss of content but rather a separation due to the shrinkage. Yet the reciprocal can also happen, i.e. when a piece of lacquer coat is folded on another one.

The trouble is to find out what is the best interpolation scheme to match the sound between the cracks. Furthermore, these latter issues become more important when a sound has to be synced with a visual representation, as for a movie sound track for example. The sound must match the picture so that the film is properly watchable.

3.7 Sample recordings

To investigate properly on the previously discussed issues, a canonical test set has been provided by the laboratory. These samples gather the typical problems that the current software components are unable to process correctly. Their properties as well as a general analysis of the related issues will be detailed in the next sections.

3.7.1 Graham Bell disc

A collection of Graham Bell records⁴ has been recovered using the IRENE 3D Probe at the Library of Congress in Washington DC. This collection comes from the Volta Laboratory

⁴More information on the recovered collection can be found on this IRENE web page: <http://bio16p.lbl.gov/volta-release.html>.

[15] created by Alexander Graham Bell. One of the disc in the collection, labeled 287700, is a vertically-cut wax disc. It can be viewed in Figure 3.16.



Figure 3.16 *The Graham Bell 287700 disc. The radial cracks are clearly visible on the photography.*

This record is cracked and the specific cracks are clearly visible as black traces on the support, revealing the material underneath the wax layer visible in a yellowish color. This disc has been acquired with the 3D Probe system, but the program is not able to properly process it. Indeed, one can clearly see that groove shifting can appear when a crack is reached, as in Figure 3.17, making it impossible to track.

On the whole record, the shifting does not seem to be very large. There is no more than one groove width of shifting in this case.

3.7.2 Dickson cylinder

This specific wax cylinder comes from an experimental project by William Dickson and Thomas Edison [14]. It is the soundtrack recording for the first film with recorded sound in addition to the motion picture. It is then an early prototype of a sound-film, but without real synchronization between picture and sound. This first attempt of a sound-film is known as the *Dickson Experimental Sound Film*.

This cylinder is seriously damaged and contains, in addition to cracks, big areas where the grooves are removed, probably due to a piece of layer that has been entirely broken. This problem can be visualized in Figure 3.18a. A smaller example of a crack is presented in Figure 3.18b as it is viewed from PRISM in full resolution.

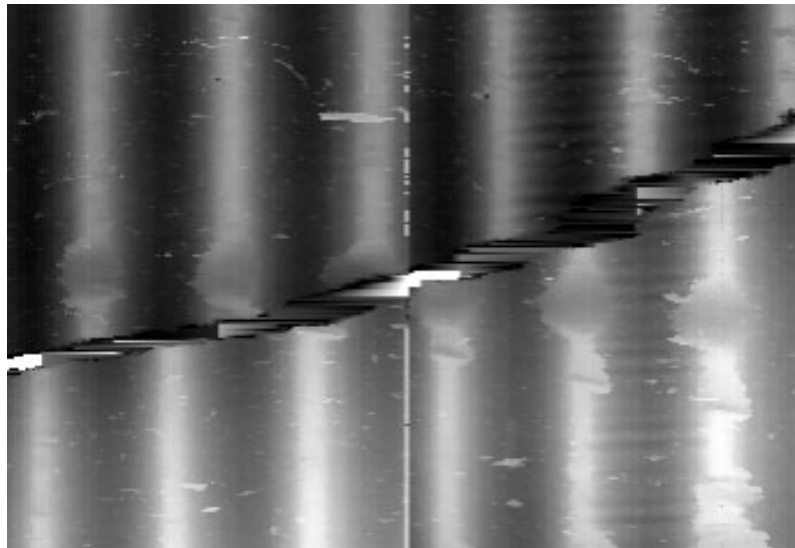
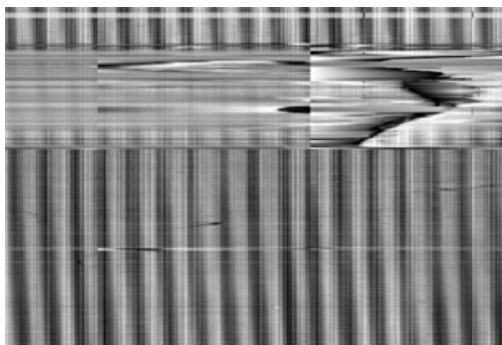
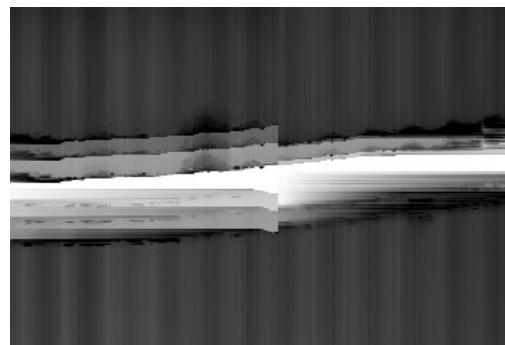


Figure 3.17 A crack on the 287700 disc as it appears in PRISM (detail image).



(a) A damaged area as seen in binned view.



(b) A small crack in detailed view.

Figure 3.18 Damages on the Dickson cylinder. In both images, the vertical grooves are hardly noticeable because when an irregularity occurs the depth difference is much bigger than the usual groove height (this is only a visualization issue).

However, even with these missing parts, the shifting remains once more quite small, and the right path is quite easy to spot for a human.

3.7.3 Boas recording

Franz Boas was a famous anthropologist born in 1858 in Germany [9]. He moved to the United States in 1887 where he started to study Native North American populations. During his researches and expeditions on Vancouver Island in 1930, Boas recorded sound. Some of these records have been acquired with 3D Probe for restoration some time ago.

One of the disc in the collection, labeled 0724, is a damaged wax cylinder.

As with every cylinder, it was acquired with the 3D Probe system. This record suffers from a lot of cracks separating the data in several pieces. These separated pieces can then be quite small. Figure 3.19 represents a portion of the disc after the acquisition.



Figure 3.19 *Excerpt of the acquisition of the 0724 cylinder as seen from the binned image in PRISM.*

The grooves are clearly visible but when a crack appears, the correct matching between the different parts is difficult to spot because they are almost always shifted. Even for human eyes, the right way is difficult to follow. PRISM currently fails to properly track the grooves on this record automatically.

3.7.4 Brigrance recording

William Norwood Brigrance was a teacher at Wasbash College and was also the leader in the Speech Association of America. He is well-known for his work in rhetoric, teaching people how to effectively speak and improve the communication. Some of his audiovisual material has been brought to the laboratory for restoration. One of these records is a lacquer disc with many cracks.

This disc is the only one in the test set that has been acquired with the IRENE system, as it is a laterally-cut disc. Many cracks can be seen on the scanned medium, as viewed from the acquisition in Figure 3.20.

We can clearly see from the binned image in Figure 3.20a many irregularities due to the cracks. However, the shifting is narrow and the general path to follow seems obvious. The

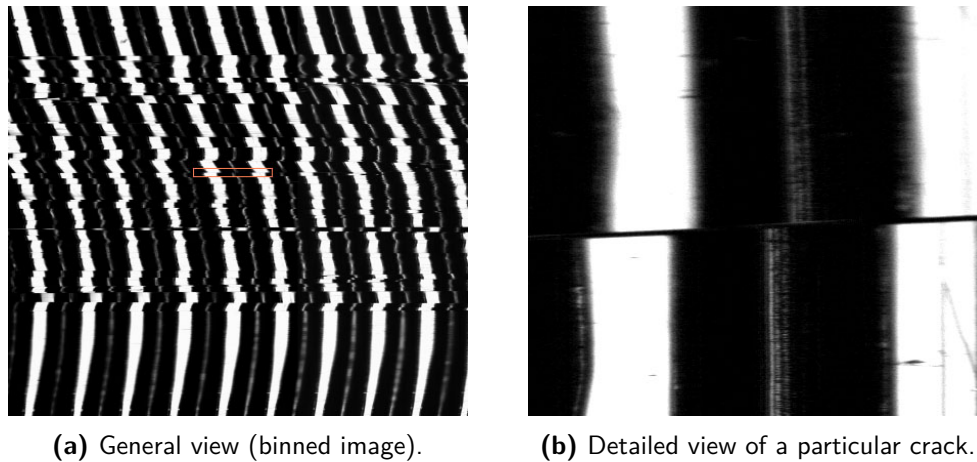


Figure 3.20 *The Brigance disc as viewed from the RENE program.*

detailed view in Figure 3.20b shows however that the crack is very thin, making it difficult to detect.

In fact, these cracks are due to bubbles spread on the whole record. The main issue here is that the groove center is blurred, due to the height difference where bubbles arise. Thus, the acquisition system is unable to focus fast enough.

3.7.5 Conclusion

All provided samples are unique and show a lot of different properties according to the mediums and their condition. This means that a general solution for this problem is difficult to come up with. Some parameters will then probably have to be chosen to adapt the future solution to different types of medium. In the worst cases, the user intervention to help tracking the grooves may be mandatory.

3.8 Summary

This chapter presented an overview of the solutions developed at LBNL. It covered the two systems, IRENE and 3D Probe, both in terms of software and hardware. It also presented the common issues appearing while processing damaged audio records, and the test set provided by the laboratory with its specific properties. These samples will be used to test the new features.

The following chapters will cover the existing ways of figuring out these issues, and present new features implemented during this thesis to improve and ease the recovery of these damaged records.

Chapter 4

Manual tracking

This chapter explains how the manual tracking may be improved. Because there are a lot of different mediums and the same amount of issues coming with, it is unlikely to find the best solution to solve each of them. There are some recordings to be recovered that have particular and unique issues, appearing only in a small section of the record. Likewise, some small artifacts can appear visually easy to catch and fix for the human eye, but very tricky to solve with a particular algorithm.

Therefore, for this kind of thing, falling back to a manual solution may be the most appropriate alternative.

4.1 Current implementation

As summarized in Chapter 3, both RENE and PRISM feature similar manual groove tracking. The system is simple but useful on specific cases. To start tracking, the user must select the proper option in the tracking options. When clicking on the general record view with the *Ctrl* key pressed (left image on PRISM), a first tracking point is stored. Then, each successive click adds a new point and a polyline is drawn connecting all previous points, giving the general track path as seen in Figure 4.1.

The program offers the possibility to store the tracked coordinates in a file to be used later. Finally, when the user applies the tracking, all points are interpolated and the processing step starts as usual. A summary of the required actions is presented in Figure 4.2 to better understand the general procedure and how the tracking fits into it.

4.1.1 Limitations

The current implementation is suitable to handle some complicated cases but suffers from several limitations:

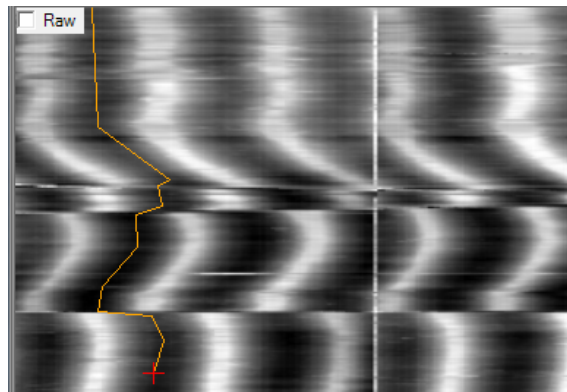


Figure 4.1 *Manual tracking example in PRISM.*

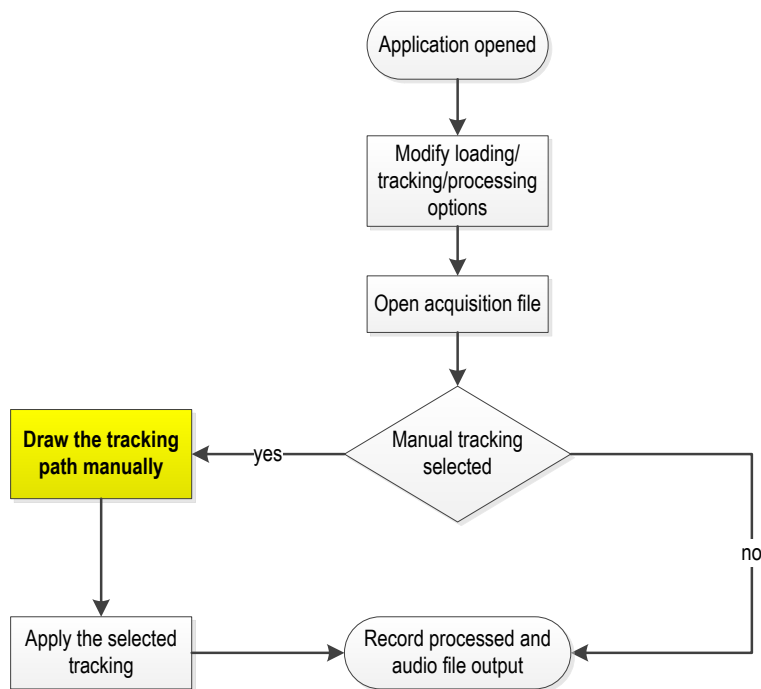


Figure 4.2 *General workflow of a whole processing using PRISM or IRENE.*

- Unnatural way to draw a path over an image.
- Tracking points can be manually put only on the binned (left) image, which can lack of precise information, especially if the scanned part is big and the record is cracked or damaged.
- No possibility to undo parts already traced.
- Tracking a whole record manually is a very slow process.

4.1.2 Suggestions for improvement

Regarding the previous limitations, several proposals were made at the project startup to find ways to improve the user experience and enhance the tracking efficiency.

Touchscreen interface

One of the concepts was to use a touchscreen interface. Nowadays, a lot of people are used to these interfaces such as tablets, smartphones, etc. Moreover, following a trace on a touch screen with the finger or a stylus seems to be user-friendly. This interface could also be convenient to re-align broken discs with shifted grooves (see Subsection 3.6.1).

However, using a tablet would imply working on a specific operating system which would not be compatible with the current applications. It would be possible to implement a communication interface between the standard application and the UI but another problem could arise: these devices have mostly a limited capacity, while the amount of data processed for the sound recovery is very large. Nevertheless, using a simple touch screen within the current system and adapt the interface accordingly is still worth considering.

Interactive tracking

Another idea was to develop an *interactive* way of tracking the grooves. The interface would display the groove scrolling *continuously*, and the user would have to correct the lateral position to stay over the groove center.

The scrolling speed must be adaptable so that the tracking is convenient in different situations. Over areas where the surface is clean, it can be fast because there is no big correction to fix the track. When the right way is less obvious, the user can slow down and stop to choose the right way.

The zoom could also be adapted regarding of the speed. When the speed is slowed down, the zoom factor is greater to see precisely a specific part. When the speed is high, the user does not need to see the groove scrolling rapidly but should instead have an overview to stop at the next difficult point.

In fact, this feature could also be seen as a more entertaining way for this kind of work, because it could look like a “game” because of its animated and lively interface.

Further improvements

Beyond a brand new specific interface, other improvements to the user experience can be made to ease and accelerate the manual tracking.

The ability to undo what was done so far to be able to correct a wrong decision is a useful feature. Likewise, it could be gainful to be able to put some annotations before actually track the grooves. RENE and PRISM already offer information and statistics about the scanned record. For example, one could notice that the signal on the graph between two shifted grooves is similar. It would then be possible to click on the specific ends to indicate that they are likely to be linked together for the tracking.

4.1.3 Summary

The current user interface for manual tracking is functional but can still be improved in many ways. The concept for a touchscreen interface is interesting but not easily applicable in the current state of the system. Therefore, the interactive method seems to be a good idea and its design and implementation is detailed in the next sections.

4.2 Interactive tracking

This section presents the interactive tracking feature as it has been integrated into the PRISM program. The next two subsections present it in terms of main features and graphical interface, before detailing the implementation and software design.

4.2.1 Main features

To use the interactive tracking, the user first selects the corresponding button in the tracking options. Then, when the record is loaded, the corresponding disc is visible in the interactive panel. A red triangle acts as a beacon to clearly see the current tracked position.

When the user clicks on the panel and moves the cursor, the horizontal position moves accordingly. The goal is then to place the triangle tip on the center of the groove. While maintaining the mouse button pressed, the user may then increase the speed by using the mouse wheel. The record scrolls vertically while the horizontal position is still following the cursor.

To actually start the tracking, i.e. keep track of the points interpolated, the user must click the *Play* button on the left.

Synchronization between views

The interactive view and the global view on the left are always synchronized. The current tracked position is visualized on the left panel of the application as a small red square and moves accordingly. When the tracking is performing, the corresponding polyline is drawn

on the left part, as seen in Figure 4.3. The path is also represented in the right panel, so that the user is able to see which was the previous tracked groove before starting the next one.

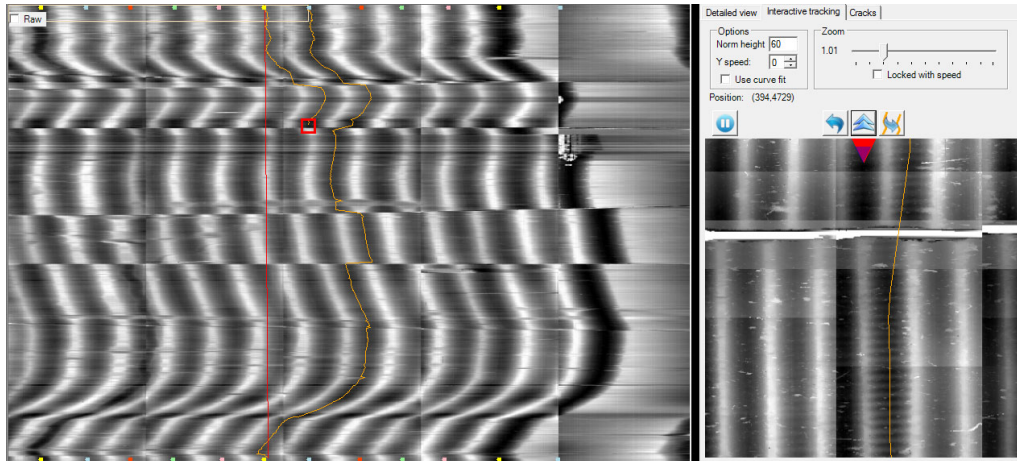


Figure 4.3 *Interactive tracking synchronized with the global view.*

When the left panel is clicked, the position in the interactive panel is also changed accordingly. When the tracking is running (the *Play* button is pushed), a click on the panel also has the effect of adding an interpolation point. In fact, this acts in the same way as the original manual tracking. It is useful for some areas where the disc is in good quality and passing through the entire groove interactively would be too long. The user can easily switch between the two alternatives while tracking a record.

Undo option

One of the problems that happens with the manual tracking is the lack of an option to undo the last actions. It appears quite frequently that the last action is improper resulting in a bad tracking. The interactive tracking offers the possibility to undo the steps up to the starting point.

Groove center correction

Another feature which has been added is the correction of the actual position to match the very center of the groove. In fact, the system is able to precisely find the groove center once the it is known which one is the current one to track. The user involvement is then more a way for helping the system to choose the correct groove from an approximated position.

The user will then see two triangles. A red one which is fixed and represents the position

that the user points out, and the blue one which shifts to the deepest position in the surrounding area.

Assisted tracking

Sometimes, an automatic tracking is partly possible in an undamaged section of a record, e.g. between two points where the surface is not crossed by a crack. In this case, the existing algorithms would still fail to globally track the record because of the damaged areas.

A feature has been added able to automatically track the record. Instead of drawing a straight line, a tracker tries to follow the grooves and stops at the cursor position.

This is particularly useful when the grooves are not straight, e.g. in the case of off-axis (see Subsection 3.2.1).

Groove pattern cloning

It also often happens, in the case of cracks, that once the correct way has been figured out for a revolution, the same pattern applies in the near area. A feature enables to copy the current pattern from the previous revolution has been added. An example can be viewed in Figure 4.4.

One can see that even if the path is not evident, once it has been found the exact same pattern can be copied for every revolution.

Magnifier

Given the properties of the interactive tracking another interesting feature would be also to act as a magnifier to help the user analyzing the record before actually tracking it. In fact, the interactive view provides an option to change the zoom level. As the drawing is interactive and drawn several times per second, the showed position can also be adjusted directly when the mouse is simply moving on the binned image but without tracking. When the mouse leave the area, the interactive directly returns to its last tracked position.

4.2.2 Graphical design

Location and integration

As the current interface is already overflowed with all existing parameters and options, it is important to add the new feature so that it is integrated smoothly in the existing application, without making it tedious to use.

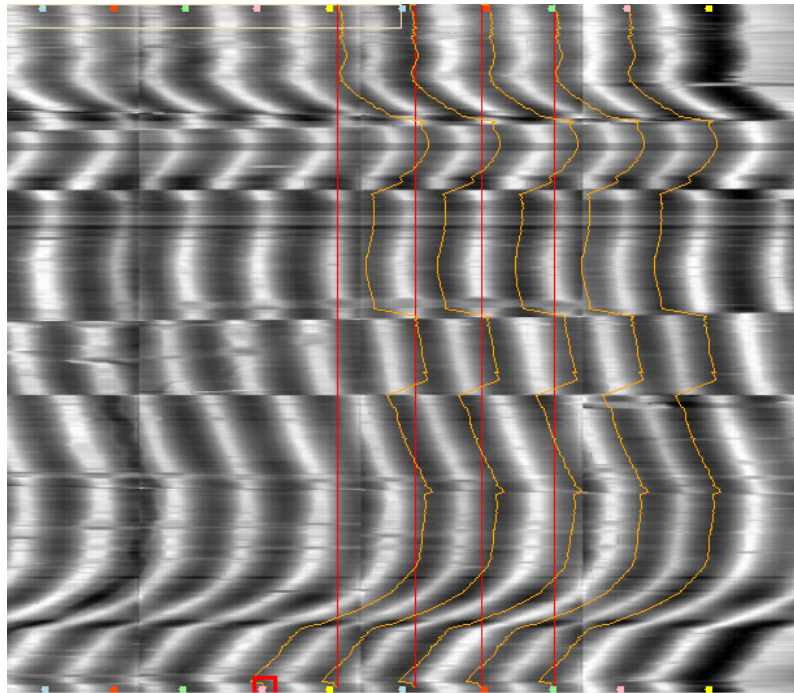


Figure 4.4 *Example of the groove pattern cloning.*

As seen in Figure 3.12, the interface is already able to show a detailed part of the disc on the right panel. When the user clicks on this detailed part, he can visualize other information on the signal. However, it is not designed to be interactive. In fact, each time the user selects a part on the right image, the corresponding part is reloaded and stored in a bitmap to be drawn on screen. The whole process is too time-consuming for an interactive drawing though it is suitable for its purpose.

The new feature has then to be separated from this detailed view. It has been chosen to put a tabbed panel to choose between the two configuration. As the user will not need both features at the same time, this is a suitable solution.

Tracking panel

The interactive tracking panel is separated in two parts: controls and visualization. The first part enables the user to manage and edit the different options while the second represents the record scan which the user has to follow to indicates the grooves. Figure 4.5 represents the whole panel.

The controls enable to monitor the tracking by changing several parameters such as the speed or the zoom level. As explained in Section 4.1.2, the zoom can also be locked regarding to the speed, using the corresponding option.

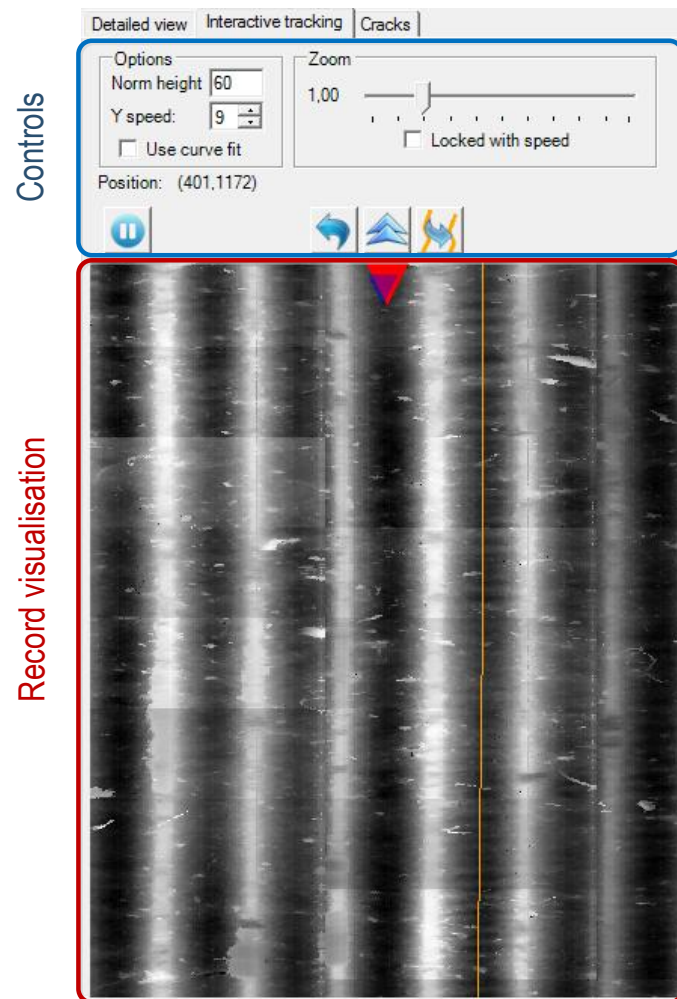


Figure 4.5 *Interactive tracking panel in PRISM.*

4.2.3 Bitmap creation

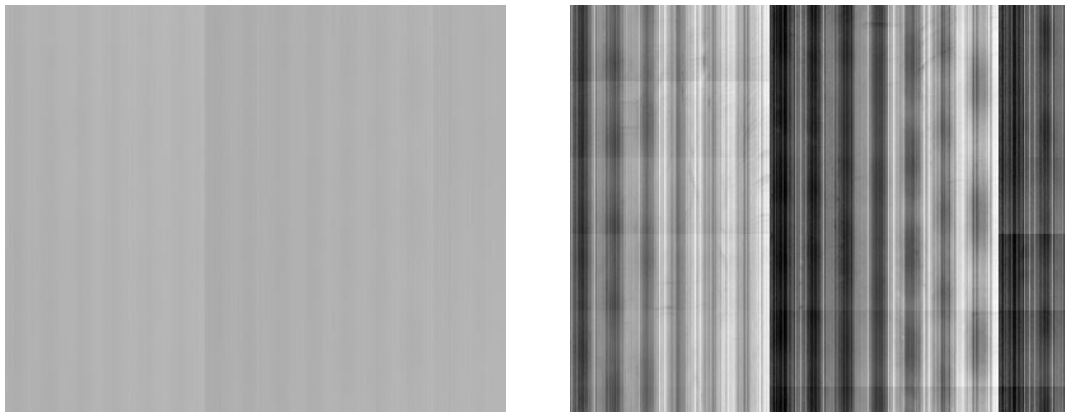
At creation, the scanned record must firstly be converted to a bitmap, so that it can be drawn on the panel. Most part of the process is implemented in the `BitmapFromData()` in the utility class.

As already explained, the raw data of the scanned record is stored as an array of floating-point values (see Section 3.5.1). They must then be converted and normalized to be stored as values in the range 0–255 to represents pixel values. The bitmap created has the 8 bpp format, using a 256 colors palette of grayscale values, so that the memory is not wasted with other colors.

Contrast adjustment

Moreover, the normalization must not be applied on the whole image in one step. Given some record values, the contrast would be very low in this case. This is not an issue for the existing detailed bitmap as the contrast is optimized for the viewed part at each click. However, this would take way too much time if the bitmap would have to be recreated (almost) every time the view changes.

To address this problem, the bitmap is normalized in several rectangle pieces. The width corresponds to the size of a single scan (180 points in one pass for the 3D Probe, see Section 3.5.1). Because there is sometimes a slight Z-shift between two revolution. The height may be changed by the user. Getting a good contrast depends a lot on the acquisition. On damaged records with out-of-range values, the overall contrast may be very bad. The default value of 60 px works in most cases. The difference between local or global adjusted contrast can be viewed in Figure 4.6.



(a) Contrast adjusted from the whole image range. (b) Contrast adjusted in subrectangles of (180, 60).

Figure 4.6 Illustration of global and local contrast adjustment on the Dickson cylinder. Because of the height irregularities, the depth of grooves is barely visible on the left image.

4.2.4 Drawing

The main implementation point is to update and draw the panel. This part is handled using GDI, provided in Microsoft .NET through the `System.Drawing` classes.

As its name implies, *interactive* means to draw the objects several times per second, so that it is possible to output a smooth animation. The way WinForms are usually handled is not meant to perform animation. The panels are refreshed only when required, e.g. when a window has been hidden by another, or when it is resized to replace the components. To

refresh the panel manually, it must be marked as invalid by calling the `Invalidate()` function.

To perform an animation, one may then use a `Timer` (provided in the API) and, at each tick, update the view before invalidating the panel. The interval should be set to a value giving a high enough refresh rate. For example, with an interval of 40 ms, the corresponding refresh rate is $1/0.04 = 25$ fps.

Linear transformations

Another important consideration is the transformations. Although there are not a lot of objects drawn on the panel, the correct positioning of all elements is the main point.

For example, the bitmap representing the record is placed regarding the user inputs. However, the relevant position is, from an external point of view, the coordinates in the record where the red triangle is pointing. On the other hand, from the drawing area perspective, the triangle is fixed at the top center. Therefore, to draw the bitmap at the correct location, the drawing coordinates must be inverted and shifted. When the targeted position is 30 pixels from the left and 200 from the top, the bitmap must in fact be translated by $(-30, -200)$ from the triangle tip.

There are different coordinate systems, as viewed in Figure 4.7. Likewise, the zoom level influences the final position as well, and the final shift also depends on this factor.

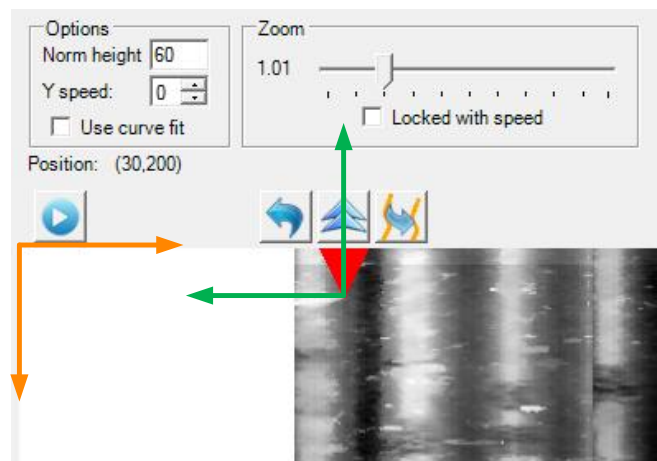


Figure 4.7 Panel and record coordinate systems in the interactive tracking panel. The first one is represented in orange and the second in green. When the record is drawn at $(30, 200)$, the bitmap is translated to the left and top.

In fact, each object has its own properties. To be properly drawn on screen, their position expressed naturally must be converted to the panel coordinate system. The needed transformations for each element are summarized in Table 4.1.

The most effective and flexible way to easily handle these conversions is to alter the global transformation matrix before rendering each object, instead of applying manually the transformations to the point. This enables to keep the position in their natural logical coordinates. The corresponding conceptual code is presented in Listing 4.1.

Listing 4.1 *Interactive panel drawing and coordinate systems transformation.*

```

1 // Start all drawing at the panel center and tip of the red triangle
2 g.TranslateTransform(this.Width / 2, trianglePos.Size.Height);
3
4 // Save this base system
5 Matrix savedTransform = g.Transform;
6
7 // Coordinate system transformation
8 g.TranslateTransform(this.Width / 2, trianglePos.Size.Height);
9 g.ScaleTransform(zoomLevel, zoomLevel);
10
11 // Draw the record bitmap
12 Point renderPos = new Point(-position.X, -position.Y);
13 g.DrawImage bmpRecord, renderPos;
14
15 // Set the translation transformation for the tracked points
16 g.TranslateTransform(-position.X, -position.Y);
17
18 // Draw tracked points...
19
20 g.Transform = savedTransform;
21
22 // Draw red triangle (no transformation needed)
23 trianglePos.Draw(g);
24
25 // Transformation for the blue triangle position
26 Matrix m = new Matrix();
27 m.Translate(this.Width / 2, 0);
28 m.Scale(zoomLevel, zoomLevel);
29 m.Translate(-position.X, 0);
30
31 Point[] trackPos = new Point[] { new Point(TrackedGrooveX, 0) };
32 m.TransformPoints(trackPos);
33
34 // Draw the blue triangle at right position
35 triangleTrack.TargetPosition = trackPos[0];
36 triangleTrack.Draw(g);
37 g.Transform = savedTransform;

```

Firstly, the actual panel coordinates are translated to the center of the screen and below the red triangle. This sets the new basis for the panel coordinate system, with the origin starting at the tip of the latter. This initial transformation matrix is saved to be restored later on.

Table 4.1 *Transformation of the different objects in the interactive panel.*

Object	Coordinates relative to	Necessary transformations
Record bitmap	position in the record where the user is pointing	Inverse translation of the position Scaling of the current zoom level
Tracking path	position in the record of the points building the line	Idem as the bitmap
Red triangle	top center in panel coordinates	Translation to the center of the panel
Blue triangle	X: real tracked position in record coordinates Y: fixed in panel coordinates	X: idem as the bitmap Y: no transformation needed

The bitmap and the tracked points are drawn in the same coordinate system. One can note that for the bitmap, the inversion is not applied to the current matrix using a negative scale, but is manually set by defining the `renderPos` variable. This is to avoid that the bitmap is rendered as if it was flipped, which is not the desired behavior. We just want its starting position to be altered.

The matrix is then reset to draw the red triangle. No transformation is needed as it is directly positioned at the very top center. Then, to draw the actual tracked position indicated by the blue triangle, the same transformations as for the bitmap must be applied for the X coordinates. However, we cannot just alter the current matrix. Indeed, the zoom would alter the triangle size as well. As for the record, we only want the position to be altered. The only way is then to define a new matrix and only apply the transformation to the point, before setting the position.

4.2.5 Groove center approximation

As explained in the previous section, the groove center is refined from the position indicated by the user. This is done by iterating n points before and after the current point in the same row, and returning the deepest position. The n value is determined using an approximated groove width and a tolerance factor t in the range $[0, 1]$. For example, if the groove width is 60 and the tolerance 0.8, then $n = \frac{60}{2}t = 24$. The groove width can be fixed or determined by analyzing the record. This step will be explained in Section 4.3.2.

Later in the project, the implementation of the center approximation has been improved and extracted to be used in the automatic cracks processing, explained in the next chapter. The code is then reused and enables to get a more accurate and stable approximation by using curve fitting. The implementation is detailed in Section 5.5.

4.2.6 Managing tracked points

Several features have been added to help the user. It includes the possibility to undo an operation or copy the pattern tracked in the previous revolution (see Subsection 4.2.1). From the implementation's point of view, these features concern the access to the tracked points.

From the existing implementation, the collection storing the points is already present as a list called `ManualTrack` in the existing `Hardware` class and publicly accessible by the `frmMain` class. This collection is directly used as it enables to integrate the new interactive features in the application while taking advantage of the previously implemented features such as save and restore the tracking to and from a file.

However, it is more convenient to encapsulate this collection in a class managing the options such as undo and clone. When adding or removing a point we need to keep track of different things. For example when starting a new revolution (at the top of the mapped image), the last point index must be stored, in order to know from which to copy the last revolution.

Likewise, when several points is added in one step, for example with the assisted tracking or by copying the last revolution, it is more convenient for the user if the undo options remove all added points in one shot, instead of having to remove all automatically tracked points if the result is not satisfying.

For this purpose, a class called `TrackedPoints` acts as the collection of points, implementing methods such as `Add()` or `RemoveAt()` but also a method `Undo()` and `CopyLastRevolution()` to handle the corresponding features. At construction time, it takes the initial collection of tracked points and performs directly on it.

4.2.7 Assisted tracking

As already explained, this feature enables to automatically track a part of a disc. When the user clicks on the binned image while pressing the *Shift* key, the program is able to follow the groove center automatically. In fact, using the routines written for the center approximation (see Subsection 4.2.5, the implementation is quite straightforward. It consists of looping between the two vertical position, and applying the approximation algorithm to adjust the center at each step.

4.2.8 Design and classes organization

This part of the implementation is much about user interface and does not require a lot of processing. The main point is to find an appropriate way of integrating the new feature without blowing up the existing codebase which is already quite complex (see Subsection 3.5.3).

The new feature has been implemented essentially in two new classes called `InteractiveControl` and `DrawPanel` as represented in the diagram of Figure 4.8.

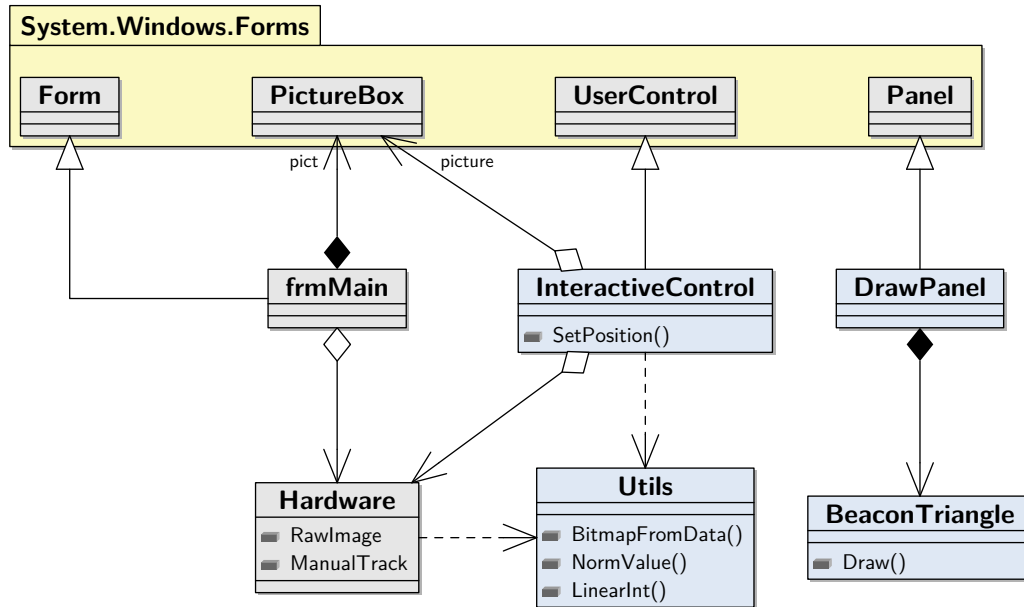


Figure 4.8 Class diagram for the interactive tracking in PRISM.

The first one embeds the whole control with the controlling part (toolbox) and the visualization path. It is a specific control and inherits the `UserControl` class of the WinForms API. The specific interactive part drawing the record is handled by the second class. It inherits from the WinForms `Panel` class. Another class called `BeaconTriangle` has been created to simplify the rendering of the blue and red triangles indicating the tracked position.

The interactive panel is owned by the `frmMain` class already created in PRISM. It uses the `Hardware` class for the communication with the record information, e.g. to get the original bitmap or set the points building the tracking path. The reference to `PictureBox` called `pict` and `picture` point to the same instance. It represents the left panel showing the global view. The `InteractiveControl` class needs it to force the drawing when the values are updated (e.g. while points are being added).

An utility class called `Utils` stands for common operations used specifically for the new feature.

4.2.9 Adaptation for RENE

Although all the analysis was done in the PRISM program, this feature is also useful for RENE, to recover very damaged discs acquired with the IRENE 2D system. Most of the

code implementing GUI was fairly quickly adapted for the program, as the custom controls deriving from the Windows Forms API are directly reusable.

Integration in the application

In terms of user interface, the new interactive panel was added as a new tab on the left part, as viewed in Figure 4.9). The feature with the cursor following in the binned image as also been added. As almost all business logic is implemented directly in the `Form1` class, which also contains the sources of the main GUI, this specific code has been directly added in this class, and the class is linked to the `InteractiveControl` class (composition).

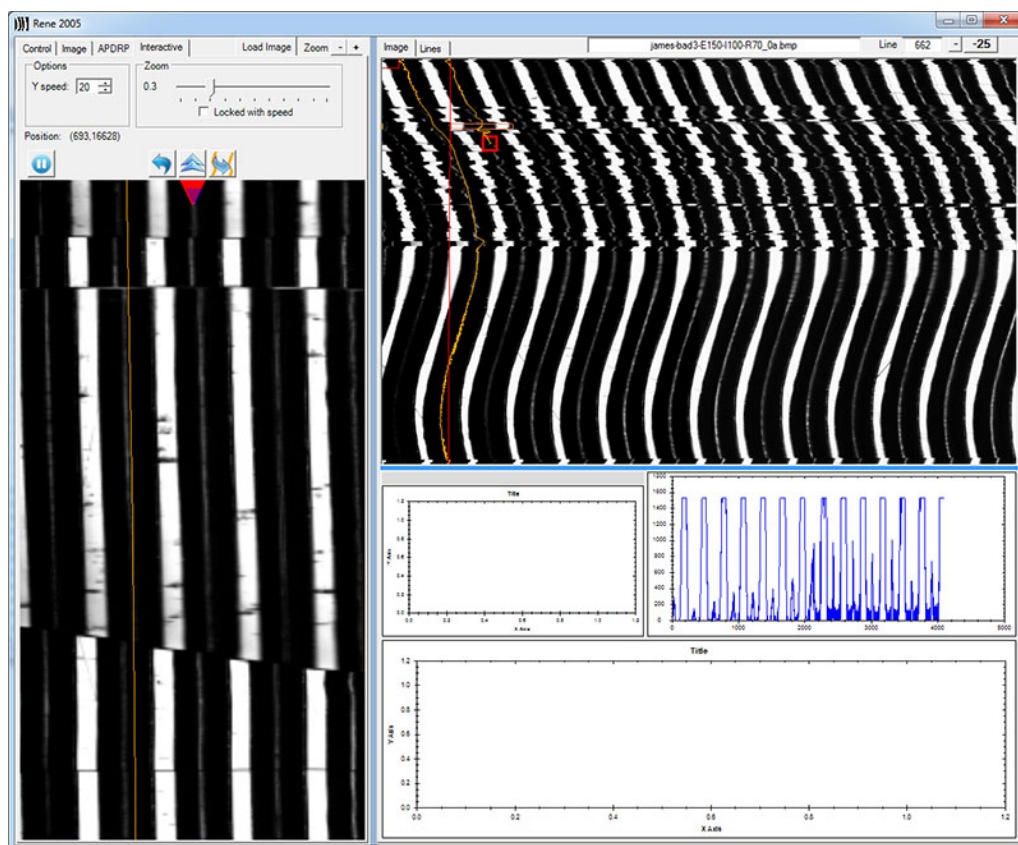


Figure 4.9 Screenshot of the interactive tracking working in RENE.

Groove center approximation

The tracking step in RENE differs from the PRISM implementation. In this case, the groove center is not already sought at this step, but instead the top edges delimiting a

groove (the intervals as represented in Figure 3.7). Then, in the processing part, the center is precisely found by looking inside the whole groove. However, while manually tracking a groove, the user does not trace both edges. Instead, he just highlights the approximate center, and the edges are computed by adding/subtracting a value conform to the groove width from this center. As the interactive tracking is based on the manual feature, the user will also follow the center.

Therefore, the main difference between RENE and PRISM is how to approximate this center. In PRISM, the real height can be used and it is straightforward to keep track of it by finding the lowest point in the neighborhood or using parabola fitting. In this case, to keep the real center, an optimal solution would be to detect the edge sides and find the center, in a way similar to what is already done for the processing step.

However, at this point, the tracking is just a rough approximation, and the real useful values are the top edges delimiting the groove. It will then be sufficient to keep track the brightest point in the neighborhood of the part pointed by the user to improve the tracking. The curve fitting method is then irrelevant for RENE and the option is removed.

Original image resizing

An issue that appeared in RENE is that a single scanned ring loaded in the program has a huge resolution of $4096 \times 80\,000$ px. It appeared that handling the dynamic drawing of such a big bitmap, though feasible, was quite slow and made the tracking uncomfortable for the user.

The solution was to resize the original image before drawing it by a certain factor. After several tests, a factor of 8, resulting in a $512 \times 10\,000$ px image was sufficient.

4.3 Finding grooves

Another feature that can help the user while manually tracking a record is to highlight the groove bottoms at the start and the end of a revolution. Even if they are clearly seen as a lower intensity in PRISM (and a lighter brightness with RENE), the match from the start and the end is not evident when a record is heavily cracked and then separated into pieces.

The main point for the user is then to correctly match the pieces together, while the tracking inside a piece itself could be easy enough to perform manually. The visual result will be the coloration of the groove bottoms with the same color sequence at the top and bottom of the binned view in the program. The result after implementation can be viewed in Figure 4.10.

Seeing the colored dots, the user then knows that if the track has the same color at the start than at the end, the tracking has a substantial possibility to be correct.

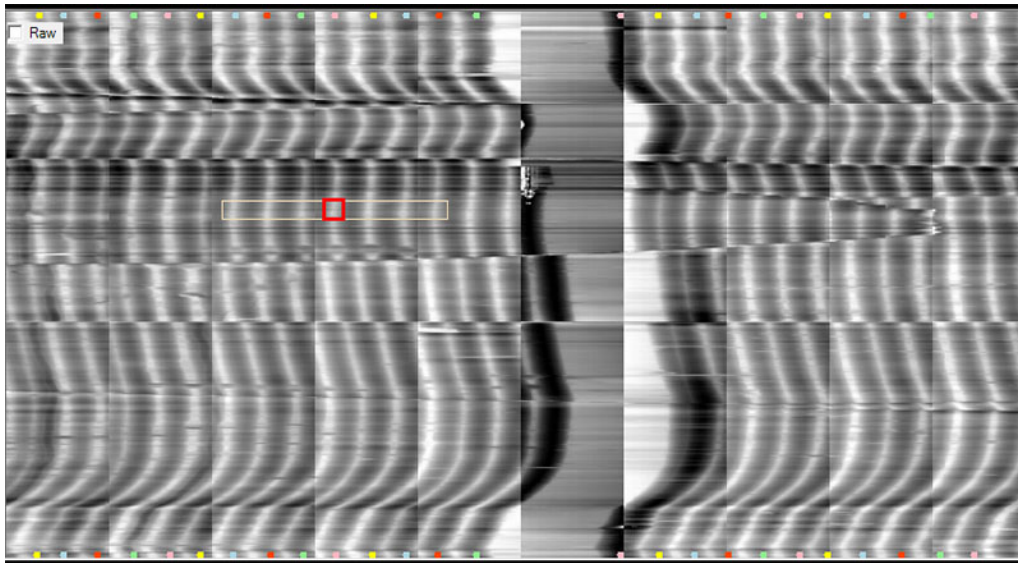


Figure 4.10 *Groove highlighting as implemented in PRISM.*

The techniques used to implement this feature will also be useful to detect groove for an automatic tracking system, to be able to reassemble them in a next step. This step will be explained in more detail in Section 5.4.

4.3.1 Finding groove bottoms

The work is to find the groove bottoms on a given Y-position. The usual tracking methods (see Section 5.1) do not need to find separate bottoms because once a groove is found, it can be followed until the end. This property is no more true on a damaged disc where the different parts can be shifted or even sometimes lost.

However, pointing out the position where the bottoms are located in a horizontal line is possible, using an adequate algorithm.

Peak detection

The values on an horizontal line on a acquired record (mapping to a radial line on a disc and a line perpendicular to the grooves on a cylinder, see Section 3.2) can be seen as a one-dimensional signal showing grooves in a sectional view.

To find the groove bottoms on the signal, one cannot use the simple mathematical derivative because the signal is too noisy. A lot of variation can happen between two different values, because the signal is not smooth as if given by a simple mathematical function. One must then use a peak detection algorithm.

Different algorithms exist. For example, one can just smooth the signal by averaging it (low-pass filter) and then use the zero-derivate method. A simple well-suited method that does not alter the original signal is presented in [3]. The idea is to define peaks and valleys. In fact, a peak is a location where the value is the highest between two valleys and a peak a location where it is the lowest between two peaks. A delta parameter defines the minimum Y distance between a peak and the next valley to really taking it into account.

Figure 4.11 presents a graph with examples values taken from the Bell disc and the points found by the algorithm.

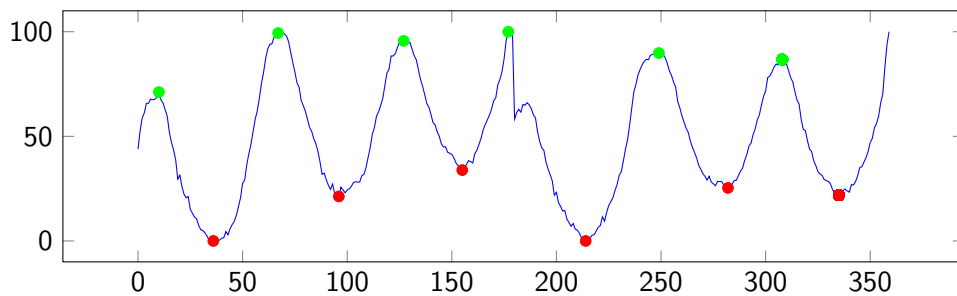


Figure 4.11 *Result of the peak detection algorithm.*

The algorithm first defines if it is looking for a maximum (peak) or minimum (valley). Then, by iterating through the data, it keeps track of the local minima and maxima. If at some point the height difference from the current point to the second one exceeds the defined delta, it means that a new extremum has been found. The point is then added and the state changed, before continuing the iteration.

The pseudocode of the algorithm is presented in Listing 4.2.

Listing 4.2 *Pseudocode of the peak detection algorithm.*

```

1 function FindPeaks(float[] data, double deltaY) {
2     min = Infinity; max = -Infinity
3     minPos = 0; maxPos = 0
4     minList = {}; maxList = {}
5     isFindingMax = true
6
7     for (i = 1 to data.Len - 1) {
8         if (data[i] > max) {
9             max = data[i]
10            maxPos = i
11        }
12        if (this < mn) {
13            min = data[i]
14            minPos = i
15        }
16
17        if (isFindingMax) {
```

```
18         if (data[i] < max - deltaY) { // Max has been found
19             maxList.add(maxPos)
20             min = data[i]; minPos = i
21             isFindingMax = false
22         }
23     }
24     else {
25         if (data[i] > min + deltaY) { // Min has been found
26             minList.add(minPos)
27             max = data[i]; maxPos = i
28             isFindingMax = true
29         }
30     }
31 }
32
33 return minList, maxList
34 }
```

4.3.2 Preparing data and filtering results

This algorithm works very well on test data set, but in practice the acquired data is sometimes not clean enough. The previous algorithm enables to get rid of small noise but may still give false positives in some cases.

Normalizing data

The data range retrieved by the acquisition is not normalized. Given the record type, the acquisition parameters or other smoothing options in PRISM, these values and their relative distance may vary a lot. Before finding the peaks on the data, the raw values must then be normalized so that the behavior remains the same regardless the situation. It was decided to normalize the values in the range [0..100]. This way, the minimum Y-delta chosen from the peak detection algorithm can be viewed as a percentage of the maximum delta between all values.

Also, as explained in Section 3.5.1, the 3D sensor takes 180 points in a snapshot. This is also known as the number of fibers on the probe. Sometimes, it may happen that the beam is not exactly orthogonal to the record surface. This results in big differences between two snapshots that influence a lot the final shape. PRISM already implements a feature known as Slope Correction. The user must enter manually a value representing the difference between the start and the end of a capture and the corresponding slope is subtracted from the original signal.

However, sometimes this correction is not sufficient. The slope is almost removed but the signal has not the same height between two snapshots and results in a “staircase-

shaped” signal. The normalization is then not directly performed on the whole row, but in portions of the scan (180 points). This enables to smooth the signal and minimize the height differences.

The final pseudocode for the method `NormalizedRow()` can be found in Listing 4.3.

Listing 4.3 *Pseudocode of the row normalization function.*

```
1 function NormalizedRow(float[,] binImage, int row, float maxMagnitude) {
2     float[] res
3
4     // Copy data
5     for(int i = 0; i < width; ++i)
6         res[i] = binImage[row, i];
7
8     float maxs[width];
9     float[] mins;
10
11     // Populate mins and maxs with the extremums for
12     // each scan (every 180 points)
13     // for(...) {...}
14
15     for (int i = 0; i < width; ++i)
16     {
17         int crtId = i / 180;
18         res[i] = (res[i] - mins[crtId]) / (maxs[crtId] - mins[crtId]) *
19             maxMagnitude;
20     }
21     return res;
22 }
```

Fix bad fiber

Another problem that came in the first place was about bad fiber detection. The probe may have a manufacturing defect affecting one or two points (that remain the same) from all the 180 ones on the probe. Because the corresponding pixels will have the maximum possible value (as for bad points), they are always seen as whitish vertical lines on the image. In PRISM, there is already a feature to remove a bad fiber. The user can enter a value from 1 to 179 and the value is fixed by computing the mean from the two adjacent values.

However, the Bell disc has been acquired by the Library of Congress which possesses their own installation. It seems that it has two bad points at the border of the scan. This results in white lines at each scan borders, as seen in previous screenshots of this record from PRISM (e.g. in Figure 4.3 on the binned image).

The current implementation has two limitations: it cannot handle multiple bad fibers and cannot fix points from the border, as the mean of both values is used (at this time, only the 180 points of the current scan are known). An option has then been added to remove bad borders. The user can enter the number of bad fibers from the start or the end of the scan. Then, it simply overwrites the values from their neighbors inside the scan.

Filtering results

Even with all the previous steps, there may still be some wrong values that come out from the peak detection. For example, in some situations with bad values, a groove may be detected twice. To avoid this, the filtering method analyzes the common distance groove size and ensures that the it is not too short, giving a tolerance factor.

The common distance is evaluated by the median of all distances between two consecutive points. This gives a pretty good estimation. Using the average would be badly influenced by uncommon distance, as seen for example in the middle of the scan in Figure 4.10. This value is also used for the interactive tracking, as viewed in 4.2.5.

4.4 Summary

This part of the project ended with a new feature to the current system. The new interactive tracking enables to simplify the sometimes painful manual tracking. It provides a new way of tracking the grooves and several enhancements helping the operator to restore records. This feature is however not sufficient in practice, as in most cases the difficult part is to find the correct path when the record is cracked and the grooves possibly shifted.

The next chapter will present some ways to detect the cracks in a record. This will be a first step for the implementation of an automatic system to track damaged records, as well as a way of providing further information to the user for the manual tracking feature.

Chapter 5

Towards an automatic processing

The main problem to address in this project is that the current algorithms are unable to properly track (find) the grooves when a record is cracked.

This chapter firstly summarizes the current implemented tracking algorithms and explain why they cannot be used as such on cracked records. Then, it explains a way of finding the cracks and distinguishing the different pieces delimited by them. Finally, the remaining work is to find the grooves inside these chunk, track them and finally reassemble all the pieces and construct the final track to be processed by the program.

5.1 Existing tracking methods

5.1.1 Tracking line

One of the currently implemented automatic tracking method, called *track line*, tries to find the approximate edges of a groove. It is especially used on disc records in RENE. This is achieved by finding the peaks in intensity and keeping track of the groove edges by detecting the changing intensity (as seen in Subsection 3.4.2). Then, it follows the corresponding X-position scanning the record downward and adjusting it to the deepest value.

When the *bottom* of the scanned record is reached, the tracking can simply continue at the top, meaning that a full revolution has been reached (see Section 3.2). The tracking is completed when the side of the record has been reached.

This method fails when there is just a little crack, because the depth is obviously completely wrong at this place, and the algorithm fails to find its way on the disc.

5.1.2 Tracking depth

The second method is the counterpart of the tracking line algorithm, operating in a similar way but specifically used for cylinders in PRISM. The difference is that it tries to directly find the groove center instead of the edges. As the height values are expressively indicated by the 3D probe scanning, this can be done by looking for the deepest point in a local region of the record. The disc is then scanned downward the very same way as with the previous method.

5.1.3 Tracking Fourier

The second method uses a different approach. The binned image is scanned horizontally, giving the groove shapes for each row. Then, the Fast Fourier Transform algorithm is applied giving the frequency space of the signal. This enables to get the peak frequency and deduce the phase which gives the distance between two grooves. When the first groove has been tracked, the other ones can then be deduced by the computed phase.

This method works well when the records are not distorted, as the distance between two grooves remains invariable in the same row. Therefore, it is also inapplicable for cracked records for the same reasons.

5.1.4 Conclusion

The existing automatic tracking methods propose different ways which have proven to be effective for a large variety of records. However, this simple approach cannot be used anymore in the case of cracked records. The biggest issue is that when a crack appears, no information is relevant in the corresponding signal. It could be avoided until the damaged part ends up, but in this case the typical problem of shifting (see Subsection 3.6.1) would make it almost impossible to follow correctly the good path.

Somehow, these cracks must be firstly detected in order to process the record correctly. Then, the different pieces may be tracked separately before being reassembled in a final step. After a first linking step, the user could be involved to check and possibly modify the result, before starting the final processing itself to get the final audio. The general workflow is summarized in Figure 5.1.

The next sections explain in details ways to find cracks, to track the different pieces and before linking them together.

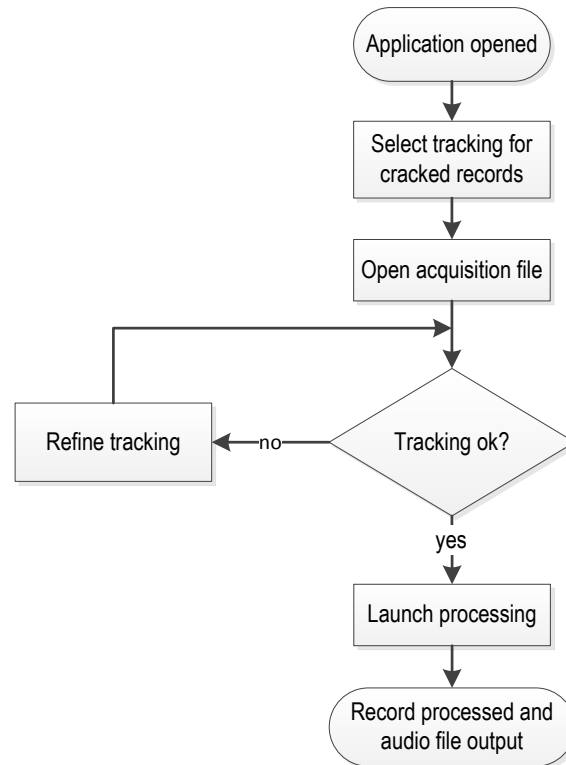


Figure 5.1 Workflow representing a whole processing using the automatic tracking for cracked records.

5.2 Chunks detection

As already seen in the previous sections we need a way to differentiate the pieces separated by cracks, which will now be called *chunks* in the rest of this report. Rather than finding the cracks, the goal will be to find the chunks to be able to process them separately.

To detect them properly, a characteristic which differs in parts representing cracks and normal part of a record must be found. Then, some image processing algorithms must be applied in order to extract the contour of the chunk. These steps are explained in the next subsections.

5.2.1 Cracked records characteristics

The recording test set with specific damaged examples has already been presented in Section 3.7. However, at this point, we need to investigate more deeply to find a way to clearly distinguish the cracks from the rest of a record.

For the records processed with 3D Probe, an early response to this problem is the infor-

mation of the brightness intensity reflected by the probe, in order to see what looks like a crack in terms of brightness. This first example in Figure 5.2 comes from the Graham Bell disc.

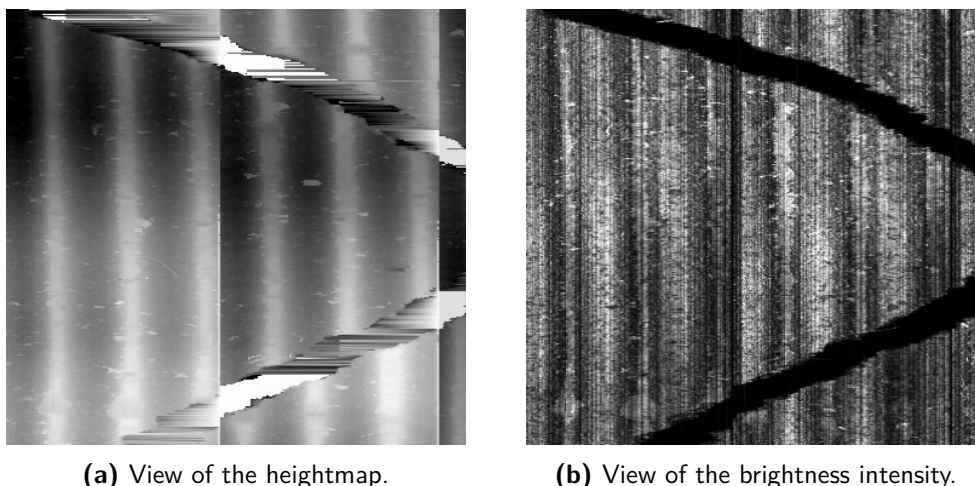


Figure 5.2 *A crack in the Bell record visualized as depth and brightness intensity.*

In Figure 5.2a, it is quite difficult to spot a clear difference. The crack area seems to be brighter but this is not very clear and some values seem to be erroneous. In fact, these peculiar values inside a chunk occur because of a feature implemented in PRISM. When the acquired values of the beam are out of range, the probe return the maximum height value, normally appearing in white in the image. However, at loading time, the bad pixels detected as totally wrong are replaced by the weighted average of their neighbors. When a lot of bad points are detected, this results in these horizontal lines seen in Figure 5.2a making difficult to extract the cracks. This feature can be deactivated, making the cracks more visible in bright values, but also resulting in other bad points (not necessarily in cracks) not being fixed.

On the brightness image (Figure 5.2b), the pixels appear much darker in the crack area. This happens in this case because the material of the underneath support reflects less brightness than the normal surface. Of course, in the general case the intensity depends on the material. For example, in a disc with metallic support, the brightness would be much higher instead. However, in all the samples used in this project from 3D Probe, the reflected brightness is clearly lower than the surface. This property will then be used to find the chunks.

Differences with VisualAudio

On the cracked discs processed by VisualAudio, the cracks are quite big and usually bigger than the actual grooves. The process to find cracks starts by decimating the image, e.g. by a factor of 16. The cracks detection can then be processed quickly.

However, in our case, the cracks appearing in wax cylinders and discs can be very small, usually smaller than an actual groove. If the image is binned, too much information is lost to properly find all cracks. The processing must then be applied on the original image. The size is bigger but this is not an issue with 3D Probe. Firstly, the scan is not separated in different files and has not to be merged. The resulting image is also smaller, as the number of points per line is smaller (see Section 3.5.1) as well as the sampling frequency detailed in (3.2). The multi-pass scan is not an issue, as for vertically-cut records the passes are usually averaged instead of kept as individual values. Therefore, the final image size is not multiplied.

5.2.2 Filtering

When detecting a feature on a digitalized image, the first task is to apply a filter to eliminate or rather reduce the noise present in the image. What is typically needed is a way of smoothing (blurring) the image. Different filters can be applied to clean the source image.

Convolution

The mathematical operation for applying a filter is called a convolution. In the case of an image, the source can be viewed as a matrix. The new image is computed by applying a convolution matrix, also called *kernel*, which contains different weights. Convolution then replaces one pixel value with a new value determined as a function of a group of pixels. The size of the group is determined by the kernel size and the (linear) coefficients are the kernel values. It is also possible for the function to be non-linear but that can be considered as a special case. For the linear this means

$$b_{mn}^* = \sum_{i=m-a, j=n-b}^{m+a, n+b} K_{ij} p_{ij} \quad (5.1)$$

with A the source image, B the destination, K the kernel and a, b determined by the kernel size such as $a = \left\lfloor \frac{size_x}{2} \right\rfloor$ and $b = \left\lfloor \frac{size_y}{2} \right\rfloor$.

With the mathematical notation $*$, not to be confused with matrix multiplication, the equation for a 3×3 kernel would then simply become

$$B = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} * A. \quad (5.2)$$

If the coefficients of K were all the same in (5.2), the kernel would result in a simple blur consisting of the average of all pixel values around. The dimensions of a convolution matrix must be an odd value but it is not necessarily square.

Blurring types

Other blurring types can be used depending on the desired results. A well-known filter is the Gaussian blur. Instead of using the same coefficients for every pixels, they are calculated using a Gaussian distribution in two dimensions. It is computed from the product of two Gaussians parameterized by x and y . The corresponding formula is given in (5.3).

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (5.3)$$

An example of a 5×5 kernel applying a Gaussian blur with $\sigma = 0.8$ is approximated in (5.4). The result is that the current pixel is much more weighted than the distant neighbors. The visual representation of the two dimensional Gaussian is viewed in Figure 5.3.

$$\begin{bmatrix} 0.0005 & 0.0050 & 0.0109 & 0.0050 & 0.0005 \\ 0.0050 & 0.0521 & 0.1139 & 0.0521 & 0.0050 \\ 0.0109 & 0.1139 & 0.2487 & 0.1139 & 0.0109 \\ 0.0050 & 0.0521 & 0.1139 & 0.0521 & 0.0050 \\ 0.0005 & 0.0050 & 0.0109 & 0.0050 & 0.0005 \end{bmatrix} \quad (5.4)$$

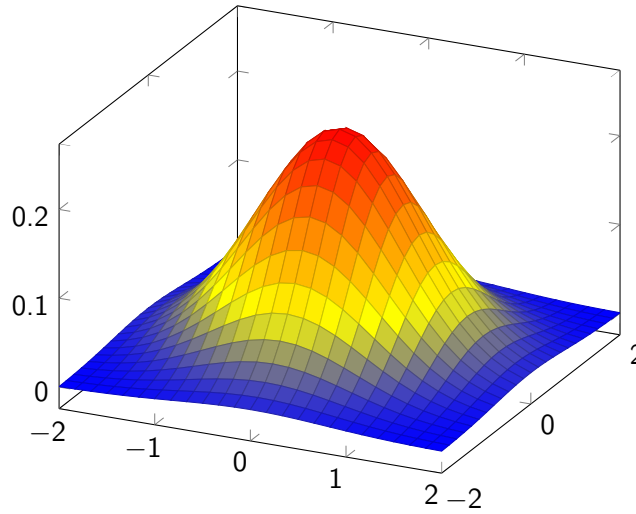


Figure 5.3 Visualization of the two-dimensional Gaussian with $\sigma = 0.8$.

Another common filtering method is the median blur. The principle is slightly different than a convolution. Instead of computing a weighted sum, the new pixel value is computed by taking the median of the values selected by the kernel. It is no longer an averaging but a sorting operation. The selected value is thus an existing one and is less subject to degenerate pixels. The outcome is that the median filter preserves edges better than the other filters. The different filtering methods can be seen in Figure 5.4.

All the filters in this example reduce the noise in the image. The regular blur seems to remove information on the cracks while the Gaussian filter gives a better result (the crack edges are less blurred). The crack area is even better preserved with the median filter.



Figure 5.4 Visualization of the different filtering methods with kernels of dimension 7×7 . The top left part is the source image and on the right is the simple constant blur. The two bottoms parts are respectively the Gaussian and median filters.

5.2.3 Segmentation

Once the noise is reduced in the source image, the next step is to segment the image. This process groups the pixels with common properties that we want to keep as valuable information (commonly called the object), and distinguishes them from the rest (the background). In this case, we want to keep the chunks areas and remove the cracks.

Histogram

A common tool helping for the segmentation operation is the histogram. A histogram represents the statistical distribution of a variable. Therefore, computing the histogram of an image means to compute the number of occurrences of each pixel values (for example

in the range 0–255 for an 8-bit greyscale image). In our case this corresponds to the brightness intensity.

The histogram of a portion of the brightness image from a cracked disc can be seen in Figure 5.5.

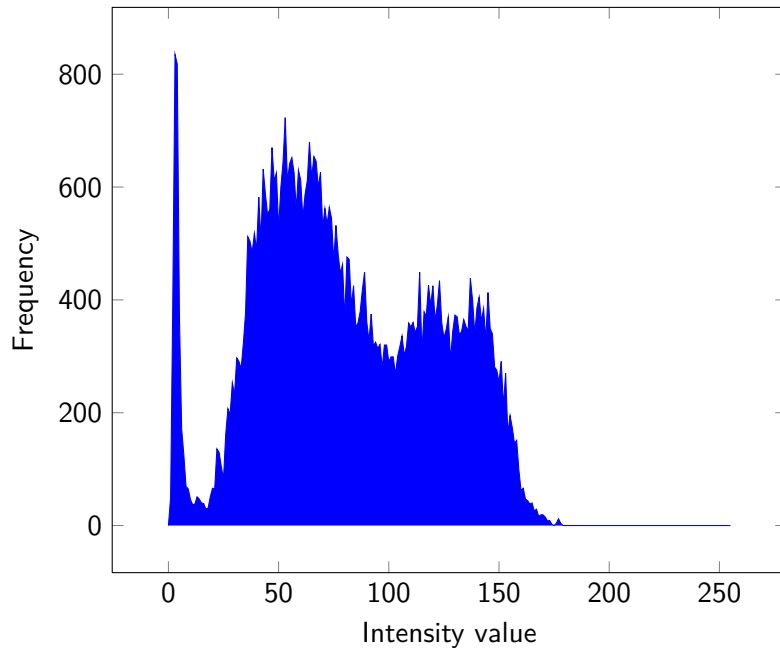


Figure 5.5 *Histogram of a portion of the cracked record.*

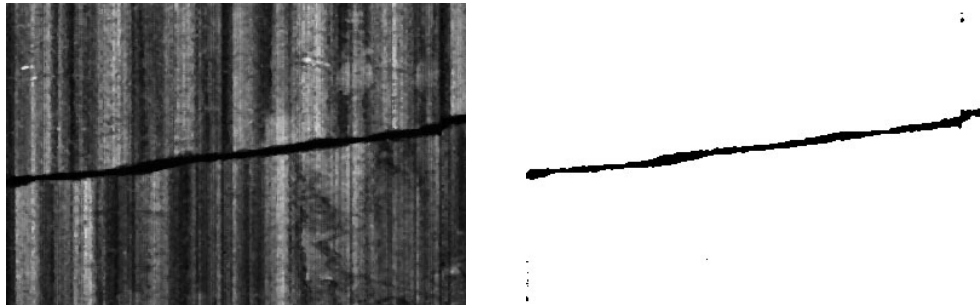
This histogram shows clearly different peaks on the intensity value. The first peak represents very dark pixel values. As predicted, this is the intensity inside the crack area, which is clearly detached from the rest of the image. The crack is approximately represented by the intensity between 0–10 and the rest of the image in the intensity 11–255.

Thresholding

With the histogram, we have the information to separate the regular record surface from the cracks. The thresholding is one of the existing methods to segment the image. It simply consists of creating a binary image from a defined threshold. If the pixel value in the source is smaller than the threshold, the destination will get the value 0. Otherwise, it will get the value 1 (in practice, often the maximum intensity value, e.g. 255 for a byte-depth bitmap), as summarized in (5.5).

$$\text{dst}(x, y) = \begin{cases} 1 & \text{if } \text{src}(x, y) > \text{thresh}, \\ 0 & \text{otherwise.} \end{cases} \quad (5.5)$$

Figure 5.6 shows the result of the thresholding applied on a portion of the Bell record.



(a) Before the thresholding (grayscale image). (b) After the thresholding (binary image).

Figure 5.6 *Result of the thresholding operation on a portion of a cracked record.*

The crack is well separated from the rest of the surface, except for some small artifacts in the top right and bottom left of the binary image. The methods to get rid of the latter will be explained in the next section.

Brightness normalization

Another way to improve the cracks detection will be to normalize the brightness signal. The raw information from the probe might indeed not be constant for every 180 points in a pass. Likewise, on the histogram example of Figure 5.5, it seems that the the range in the intensity values is not entirely used. The maximum around 180. This explains why the brightness image looks quite dark.

In fact, PRISM already implements a function normalizing the brightness in the vertical direction. Each captured points on the whole record are then rescaled from their specific properties.

The implemented algorithm computes a histogram for each column of the record. Then, in each of these, it iterates downward from the last intensity and stops when $\frac{4}{10}$ of the brighter pixels are reached. The most represented intensity (peak) in this portion is stored and then all values in the corresponding column are divided by its value. This results in a loss of information for the brighter (but underrepresented) values while the whole normalized image will be brighter and better emphasize the difference from lower and middle intensities. The visual result of the algorithm is visualized in Figure 5.7.

The cracks is more evidently visible in the normalized image. Some artifact are now visible in the crack itself, as the range is greater. The threshold must then be adapted when working with normalized brightness.

Using this feature will be interesting in cases where the threshold can hardly differentiates the object from the background.

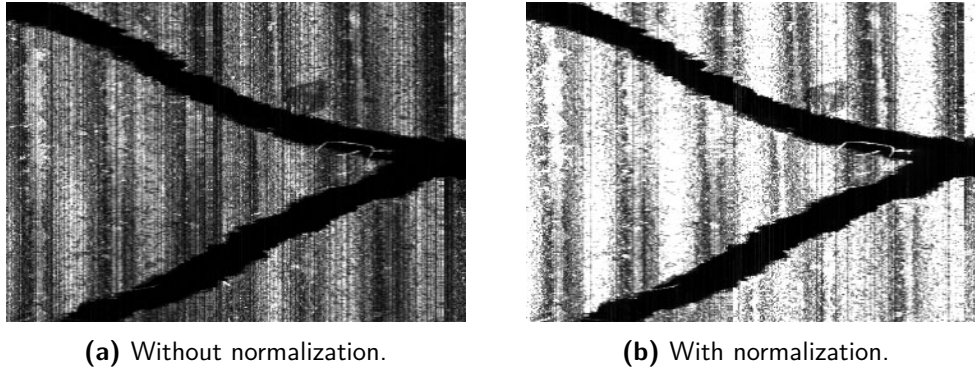


Figure 5.7 Result of the brightness normalization in PRISM.

5.2.4 Cleaning

The thresholding which segments the object from the background only takes into account the value of a single pixel to decide if it is part of a chunk or a crack. In practical cases, the distinction is almost never that evident. The histogram of an image shows peaks as statistical information but the final chosen threshold is a compromise to separate at best the two parts. There will then be some background pixels considered as being part of the object and vice versa. One possibility to clean these regions is to use the morphological operations.

Morphological operations

Morphological operations are a technique to clean an image based on the pixels in the neighborhood. These are then boolean masks applied on the binary image in a way similar to the filtering operations seen before. The theoretical explanations in this section are partly inspired from [11].

As the coefficients are 0 or 1, a mask is usually defined by its shape and is called a *structured element* in this context. A typical shape is the cross, as defined in (5.6).

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (5.6)$$

The mask M is evaluated on each pixel as a common filter. The result is then the number of pixels contained in the cross shape.

The two basic operations using these structuring elements are called erosion and dilation. If n is the number of pixels from the structuring elements, these operations E and D are defined as in (5.7).

$$\text{dst}(E) = \begin{cases} 1 & \text{if } n = 1, \\ 0 & \text{otherwise.} \end{cases} \quad \text{dst}(D) = \begin{cases} 1 & \text{if } n > 1, \\ 0 & \text{otherwise.} \end{cases} \quad (5.7)$$

The first one removes some pixels from the object and the second one add pixels to the object. These operations are most often applied together. Performing a dilation on an erosion ($E \rightarrow D$) is called an opening and the inverse ($D \rightarrow E$) a closing.

Results

Which operation to use depends on the context. The opening removes small regions considered as the object that are smaller than the structuring element. Closing adds small parts in the object that were considered as background, i.e. it “closes the holes”.

The example in Figure 5.8 shows the difference after applying a closing on a segmented binary image from a cracked record. In this case, the erosion and dilation elements are both 3×3 and each of them is applied five times (in 5 passes).

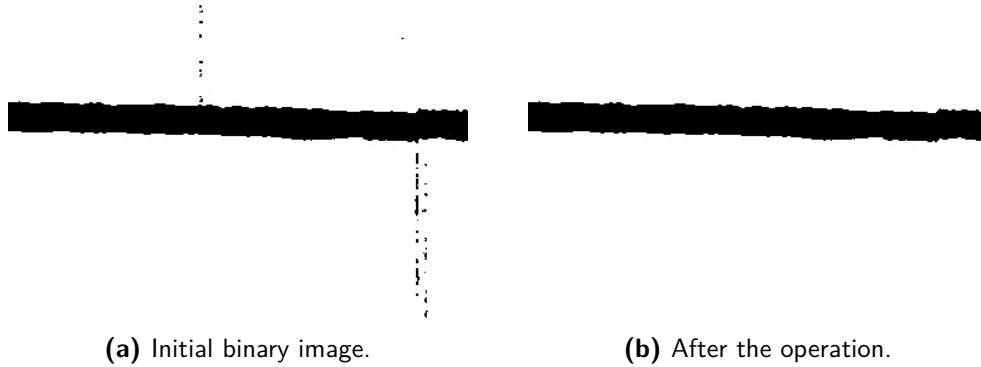


Figure 5.8 Five passes closing to eliminate black pixels in the chunks.

5.2.5 Contours extraction

Finally, when the chunks are separated from the cracks, the remaining work is to extract the contours, which will define and distinguish all the chunks in the record. The output of this step is a list of polygons (itself a list of points), describing the contours of each chunks.

The standard behavior is to describe every pixel of the contour in the image. However, one step further is to approximate a simple polygon from this set of points, as viewed in Figure 5.9. This enables to remove small noise from the contours detection and lighten the storage.

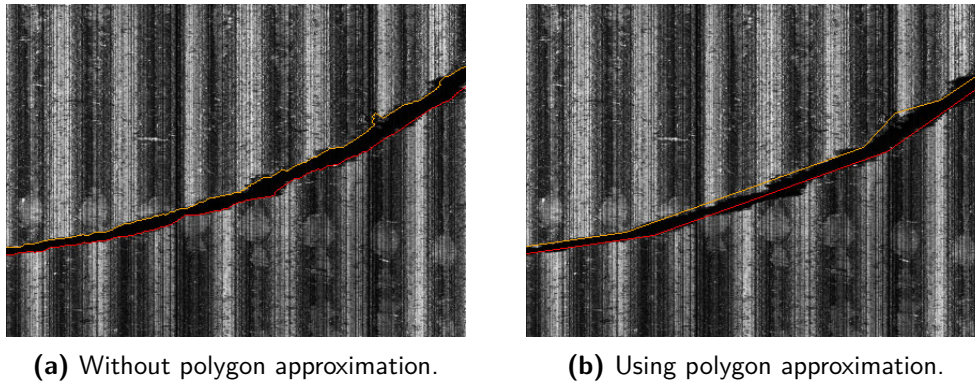


Figure 5.9 *Illustration of the contour detection and approximation.*

5.2.6 Tests and results

The previous sections explained the techniques to implement the chunks detection. As this step implies a lot of experimentation to properly find the best parameters to use, a separate application has been written enabling to easily visualize the results and change all parameters. This enables to test quickly the best parameters to use for a proper detection, without the overhead of loading a whole acquisition using PRISM.

Cracks detection application

This application has been written in C[#] in order to simplify the integration into PRISM. Also, for the image processing algorithm, the OpenCV library (more precisely its C[#] wrapper called Emgu CV) has been used. Almost all techniques and algorithms detailed in the sections above are implemented in an efficient way in this library.

The program loads directly pictures from the filesystem. For the early tests, some parts of records have been loaded into PRISM and saved as pictures. A screenshot of this program is presented in Figure 5.10.

The four steps explained in the previous section are visualized in the program. This enables to clearly see the results for each steps separately and decide how to change parameters to improve the final detection. The four images scroll together be able to easily compare the results.

In the beginning, several edge detection algorithms have been tested, e.g. the well-known Sobel and Canny filters. In the end, they were not suitable for our purpose. The thresholding as detailed in Section 5.2.3 (page 58) is the only algorithm that has been implemented in PRISM.

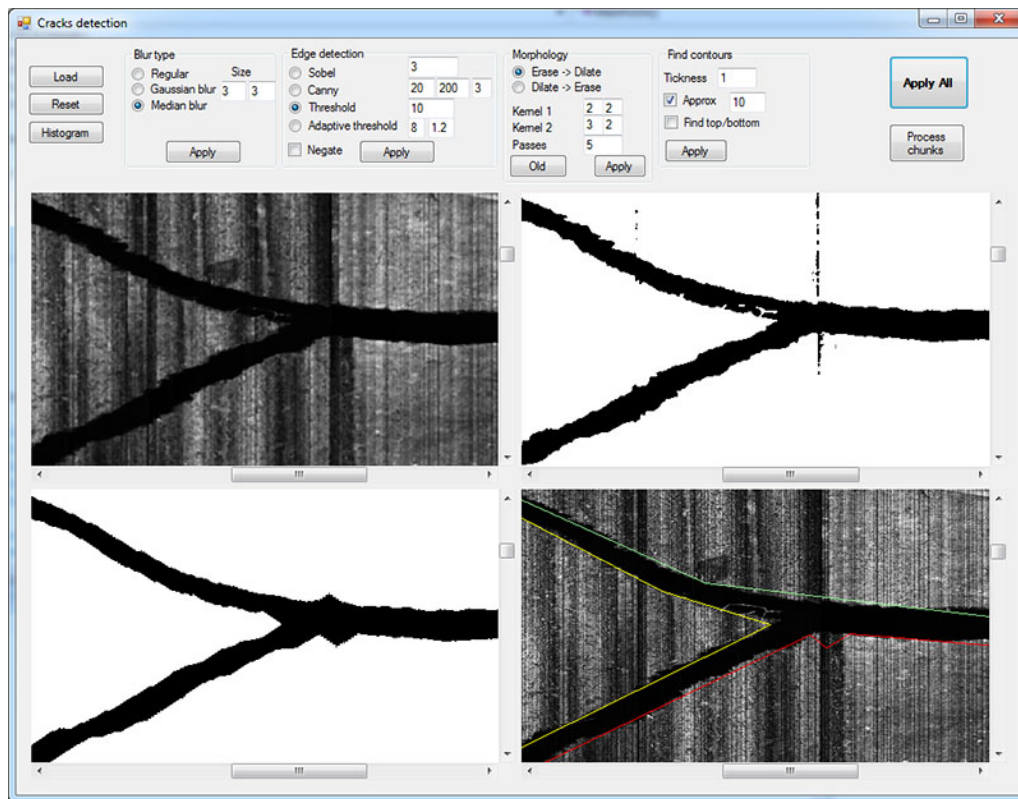


Figure 5.10 Screenshot of the cracks detection test program.

General results

The results of the crack detection algorithm vary a lot depending on the type of record and damage. Some are easily detected by correctly adapting the different parameters. However, others are really hard to detect, especially when they are very tight. In this case, the tracking algorithm will have to be robust enough to pass through very thin cracks, when the shift remains small. Indeed, if the image is binned by a factor of 30, any crack being smaller than around 15 pixels is not even spotted by the tracking algorithm.

The main issue will be when a crack is not entirely detected. The frontier of the chunk can stop anywhere, as seen in Figure 5.11 causing probable linking problems.

In VisualAudio, even if the cracks are much bigger, this issue may also happen. The workaround is to manually edit the exported image to “draw” the missing part. In the case of PRISM, we will try to avoid it, and the frontier detection (as it will be explained in Section 5.3) should not be affected by this issue.

The next sections briefly presents the result on the records acquired with 3D Probe. However, this is at this step only informative and related to chunk detection. The final tests (see Chapter 6) will explain the final results and the configurations will be added in

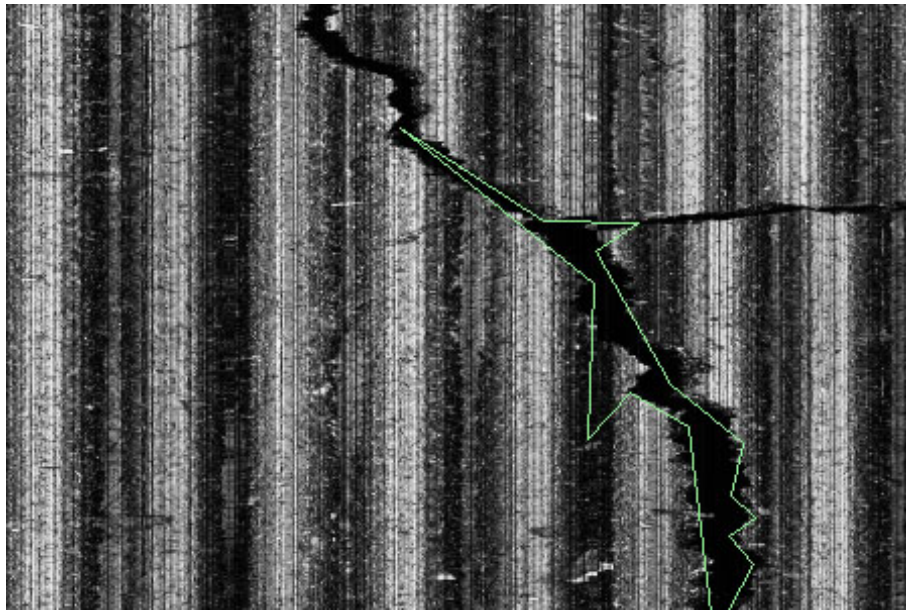


Figure 5.11 *A very thin crack not entirely detected on the bell disc.*

appendices (see Appendix A).

Bell disc

On the bell disc, some cracks are easily detected while others are really hard to spot. Fortunately, when it is the case there is almost no groove shifting. The following parameters gave good results.

- Not using brightness normalization
- Using 3×3 median blur
- Threshold of 20
- 5 passes opening morphology
- Bad fibers: two bad fibers at the start of each capture

Using the brightness normalization as seen in Section 5.2.3 does not really change the results with an adapted threshold.

Dickson cylinder

As described in Section 3.7, this record contains also some parts where the wax layer is broken and no groove is visible. Apart from this, a single radial crack crosses the whole

record at the beginning. This crack is easy to detect as it is wide. At some point the crack is getting tighter and difficult to distinguish from the rest of the record. It is however in a very damaged part where the grooves are very difficult to distinguish as well, even for a human.

The parameters do not influence a lot the detection and will then remain the same as for the Bell disc above. It would be possible to extract several parts of it but probably not entirely automatically. The only difference is that there is no bad fiber from this acquisition.

Boas cylinder

For this record, the brightness normalization must be used, because the range of intensity is far bigger than with other records, making the cracks almost invisible if not used. The final threshold to use is then greater than for the other records.

On this particular acquisition the height between different chunks varies also a lot. This should however not interfere with the detection and tracking of grooves as it will be done separately between the different chunks. A more troublesome issue is that with these parameters, some small regions are detected as chunks. The code will handle this by removing too small areas, but the final tracking and linking steps will be probably very difficult on some parts.

- Using brightness normalization
- Using 3×3 median blur
- Threshold of 80
- 5 passes opening morphology with 3×2 for erosion and 2×2 for dilation
- Bad fibers: one bad fiber at the end of each capture

Regarding all the issues from this particular record, it seems that it will be a better candidate for the manual and interactive tracking features, at least for some parts of the record.

5.2.7 Design and integration into PRISM

Almost all the implementation of the automatic tracking feature has been done in the namespace and folder `prism.CracksProcessing`. This way the specific code is easily localizable. The schema with the new added objects is represented in Figure 5.12

Firstly, the `ChunkDetector` class contains all processing code related to chunk detection. Most algorithms use Emgu CV for the specific step. From the user's point of view, the only interesting method is `Process()` which will analyze the loaded record and return

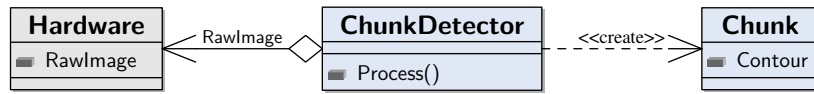


Figure 5.12 Class diagram for chunk detection.

a list of chunks. The constructor takes in parameters the Hardware object representing the record.

A chunk is itself described in its own class named `CrackedChunk`. The contour of the chunk can be accessed via the property `Contour`. The rest of the implementation will be explained as other properties are added.

The way all the parameters (thresholding, number of passes, etc.) are specified from the GUI to the object will be explain more generally in Section 5.7.

5.3 Establishing frontiers

At this stage of the processing, a collection of objects describing a physical chunk has been computed. It consists of a list of points describing a simple polygon (a polygon without intersecting lines). However, this polygon can still be non-convex. In the general case, it could be a non-trivial shape.

This section explains how to separate a chunk into several *frontiers* which will be the starting point for the groove detection and the tracking.

5.3.1 Difference with VisualAudio

At this point, the choice has been made not to directly follow the idea of VisualAudio. In the latter, the next step after the chunks detection is directly the tracking (or even processing, as there is no previous tracking step as with IRENE). The portion inside the chunk is isolated by applying a mask, and the grooves are found and tracked by a method using histograms (see Chapter 6 in [7]). It is only after this step that the chunks (now defined by a set of traces, or grooves) are separated in frontiers to continue towards the linking step.

This choice was made after analyzing the VisualAudio code and documentation. This part was quite complex and it required some algorithms that did not look relevant in the case of PRISM. Instead, it was decided to geometrically separate the chunk into frontiers, before finding and tracking the groove from the detected frontiers.

5.3.2 Goal of this step

To correctly link the trace, it is mandatory that the different pieces to link are convex or, more precisely, at least monotone in respect to the horizontal axis¹. As the grooves are scanned vertically from the acquisition, it is not a problem if the concavity appears on the horizontal direction. However, in the case of a chunk as illustrated in Figure 5.13a, it looks obvious that the grooves on the left part are split by the frontier. A missing part of these grooves are located in another part of the record.

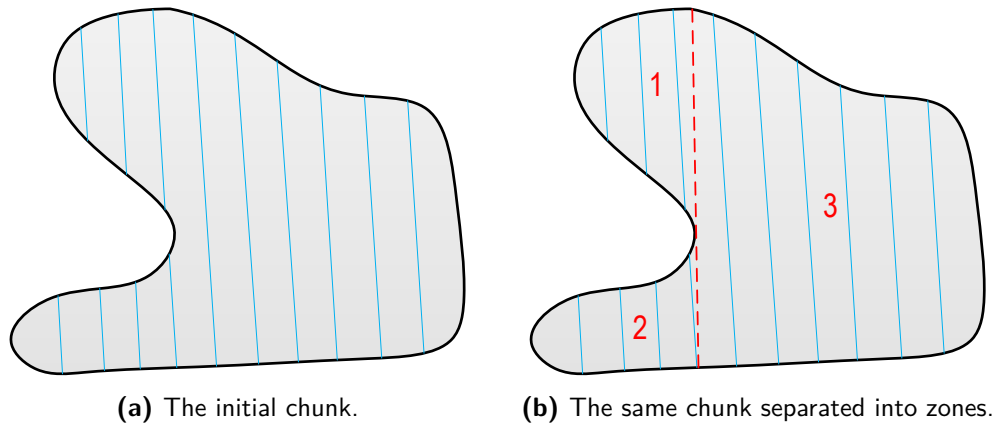


Figure 5.13 Visualization of a non-convex chunk. The blue lines represent the grooves inside the chunk.

In fact, the problem can be solved by splitting the chunks into well-defined parts. In VisualAudio, these parts are called *zones*. The corresponding zones from the previous example can be seen in Figure 5.13b. With this separation into zones, the tracking of grooves inside each zone is no more problematic.

This separation looks evident for the human eyes, but the exact definition is more difficult to spot. We must firstly define what is a frontier.

Frontier definition

A frontier is either at the top or the bottom of a chunk. A top frontier has only some content of a chunk below line, while a bottom frontier has only some content above. A convex-shaped chunk has then only one top frontier and one bottom frontier. Figure 5.14 visualizes the frontiers of the previous non-convex chunk.

A property is that if there is n top frontiers in a chunk, there is also n bottom frontiers. The zones are then defined by a common top and bottom frontier of a chunk. Of course,

¹A polygon monotone in respect to a line L means that every line parallel to L , in this case the x-axis (horizontal), intersects the polygon no more than twice.

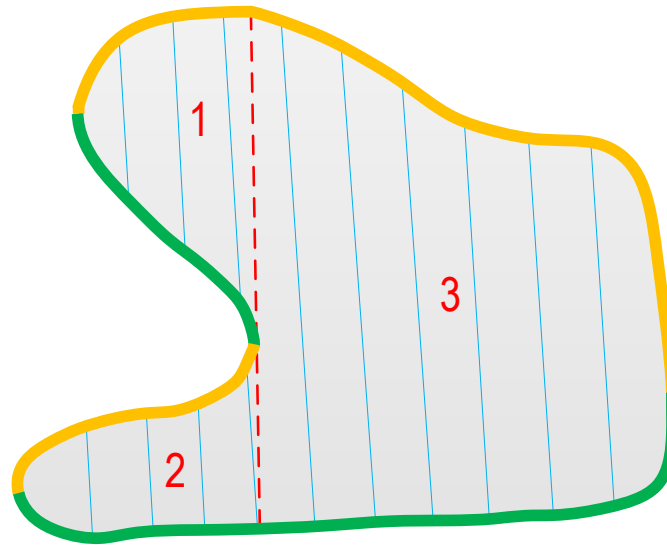


Figure 5.14 *The chunk with highlighted frontiers. The three zones are defined by the common top and bottom frontiers.*

one of the two may be smaller but a single zone is always only between the mingled part of these frontiers. The next step is to find an algorithm to extract the frontiers.

5.3.3 Frontier extraction

Detection

To extract the frontiers, an analysis of the horizontal variation of the shape must be done. Indeed, as explained at the beginning of this section, we are looking for pieces monotones in respect to the x-axis. A simple algorithm is then to iterate through all the points of the shape and detect the changes in direction. If the points are listed in a counterclockwise rotation, the X coordinates for bottom frontiers are increasing, respectively decreasing for top frontiers.

Whenever the X direction changes, a new frontier is detected. This method is visualized in Figure 5.15.

The corresponding algorithm is pretty simple to implement. To avoid that a single frontier is detected separately, the iteration must start at either the leftmost or rightmost point of the shape.

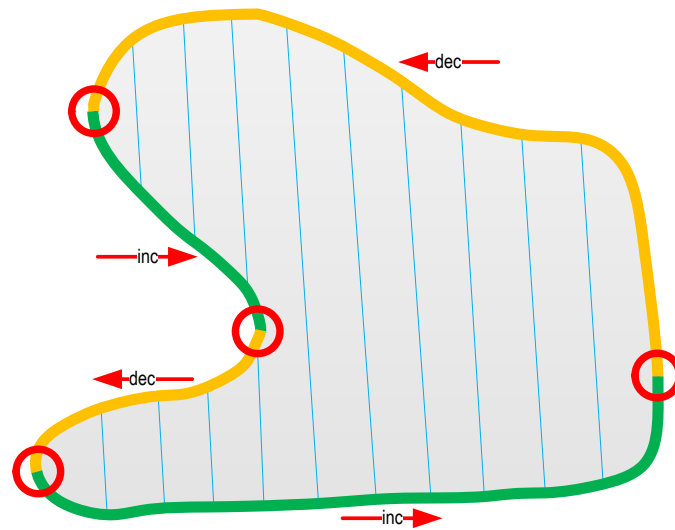


Figure 5.15 *Frontier detection based on horizontal variation.*

Improvement

The solution explained above is not sufficient in practical cases. A chunk is not necessarily smooth-shaped, but may comport several local modifications that mean no real change of direction. This problem can be minimized with the polygon approximation (already implemented as part of the chunk detection, see Subsection 5.2.5), but even in this case some wrong results are possible.

In fact, we want to define a new frontier that has a defined minimum length, i.e. when the delta (in X coordinate) between the current position and the local extremum of the current frontier is large enough. Then, the previous frontier stops at the last extremum and the current one starts at the same location.

Actually, this algorithm has already been discussed and is the peak detection algorithm (see Section 4.3.1). Figure 5.16 demonstrates the relation between the peak and frontier detection on an example. In Figure 5.16b, the graph corresponds to the X coordinates of the polygon points plotted from the leftmost one.

It is clear that the frontier extremities are defined by the peaks and valleys in the corresponding signal. Bottom frontiers are located between a peak and a valley in this order and top ones between a valley and a peak.

One can note that the chunk is previously polygon approximated (the whole shape is described with 28 points).

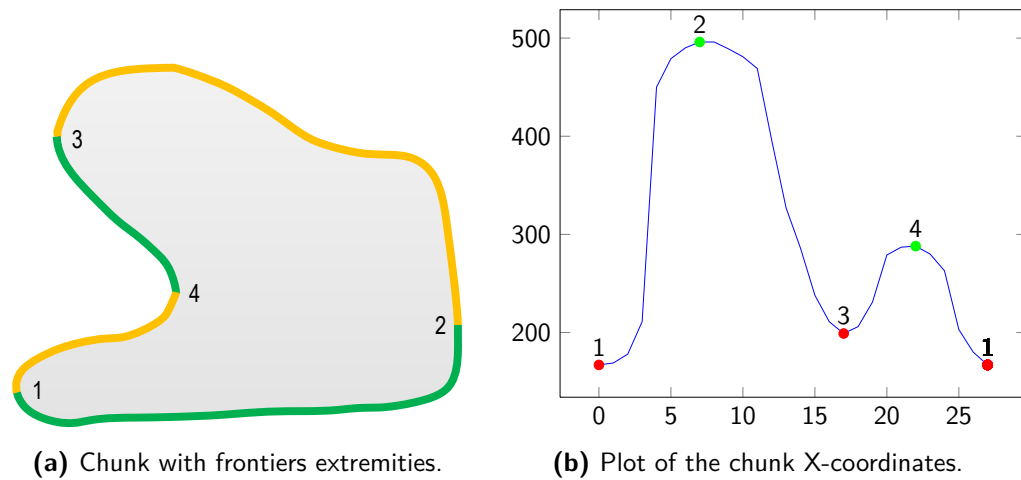


Figure 5.16 *Relation between peak and frontier detection inside a chunk. The extremities of each frontiers are numbered in both cases and correspond to the extrema on the resulting signal (right side).*

Final algorithm

The peak detection algorithm does the main work of finding the extremities. From these values, it remains to extract the different frontiers. In the easiest case, when the chunk is convex, the detection will return one maximum and no minimum. The unique bottom frontier is then located between the left-most point to the maximum, and the top one from the maximum to the end.

The pseudocode in Listing 5.1 presents the final algorithm.

Listing 5.1 *Pseudo for the frontier extraction, using the peak detection algorithm.*

```

1 function FindFrontiers(float[] chunk, double deltaY) {
2     int minIndex = x; // Index of left-mostpoint
3
4     // Copy X coords from the left-most point and repeating the first one
5     float[] pointsX = new float[Contour.Len + 1];
6
7     for (int i = 0; i < Contour.Len + 1; ++i)
8         pointsX[i] = Contour[(i + minIndex) % Contour.Len].X;
9
10    // Find tops/bottoms transitions using peak detection
11    int mins[], max[] = FindPeaks(pointsX, MIN_LENGTH);
12
13    // First bottom (always)
14    AddNewFrontier(0, maxs[0], minIndex, false);
15
16    // Mid tops/bottoms (if existing)
17    for (int i = 0; i < maxs.Len - 1; ++i)

```

```

18     {
19         AddNewFrontier(maxs[i], mins[i], minIndex, true);
20         AddNewFrontier(mins[i], maxs[i + 1], minIndex, false);
21     }
22
23     // Last top (always)
24     AddNewFrontier(maxs[maxs.Len - 1], pointsX.Len - 1, minIndex, true);
25 }
26
27 function AddNewFrontier(int first, int last, int minIndex, bool isTop) {
28     List<Point> frontier = new List<Point>();
29
30     // Copy points
31     for (int i = first; i <= last; ++i)
32         frontier.Add(Contour[(i + minIndex) % Contour.Len]);
33
34     // Add frontier if large enough
35     if (abs(frontier.Last().X - frontier[0].X) > MIN_X_FRONTIER) {
36         if (isTop) {
37             frontier.Reverse();
38             tops.Add(frontier);
39         }
40         else
41             bottoms.Add(frontier);
42     }
43 }

```

The `FindFrontiers()` function firstly finds the left-most point and stores its index. Then it copies the X coordinates of all points starting by this index in `pointsX`. The peak detection is applied on this array. Finally, the corresponding frontiers are created from the found extremities. In the for loop are found the possible mid frontiers.

The `AddNewFrontier()` function simplifies the storage of the results, separating the top frontiers from the bottom ones. It also verifies that the frontier has a minimal size before adding it to the corresponding collection.

5.3.4 Interpolation

At this point, we have a collection of frontiers per each chunk. To have all information available for future steps, another task must be applied: the interpolation of the frontier border. For the moment, the list of points defining the border has only the property of being defined from the left-most point to the right-most point of the frontier. However, following the definition of a frontier and the algorithm used for its detection, a small change of direction may still occur within the frontier, possibly resulting in several points with the same X coordinate in different values. Moreover, as the border may have been approximated (see Subsection 5.2.5), the distance between two points is not defined.

More theoretically, we need a mathematical discrete function that is defined from the left to the right point, with the standard definition that for each input X , only one output Y is defined. The border must then be interpolated in the horizontal direction.

Linear interpolation

As the list of points defines a well-approximated polygon, we can directly use a linear interpolation. The basic algorithm is given in Listing 5.2.

Listing 5.2 *Basic linear interpolation algorithm.*

```

1  function Point[] Interpolate(Point[] points)
2  {
3      Point[] interpolated = {};
4
5      for (int i = 1; i < points.Count; ++i)
6      {
7          float x1 = points[i - 1].X;
8          float x2 = points[i].X;
9          float y1 = points[i - 1].Y;
10         float y2 = points[i].Y;
11
12         for (int j = x1; j < x2; ++j)
13         {
14             Point p = new Point();
15             p.X = j;
16             p.Y = y1 + (y2 - y1) * (j - x1) / (x2 - x1);
17
18             interpolated.add(p);
19         }
20     }
21
22     return interpolated;
23 }

```

The main formula is on lines 15 and 16, where the point is created. The X coordinate is directly the consecutive one and the Y is computed by interpolating the two out points. This algorithm is simple but not sufficient. As already said, it is possible that two points have the same value. This algorithm crashes when it happens because of a division by zero on line 16 ($x2 - x1$). A condition must be added between before the inner loop to ensure that $x1$ is different than $x2$. If the horizontal distance between two points is null, there is no need to interpolate between these ones.

Moreover, the strict order is not implied and two non-consecutive points may appear on the same vertical line. This would give wrong results for the interpolation. A solution to ensure the correct result is to use a proper data structure, as a sorted dictionary

with the key corresponding the X position. This ensures the two properties, order and uniqueness.

Final algorithm

With the last additional changes, the algorithm is functional but may still be improved. In an extreme example, e.g. as the one previously seen in Figure 5.11, the values from two vertical points can vary significantly. The question is to find which one is the most appropriate to choose. Using directly a dictionary would for example just add the last computed point as the final point.

A better way is to be able to choose if we want the highest or the lowest points. Indeed, for the future groove detection, we want to avoid wrong values as much as possible. From a frontier at the top of a chunk, the values will be better at the lowest points, as the highest ones are more likely to cross the crack and give bad values. We can add a parameter and perform a test if a point already exists at this horizontal position. The final C# algorithm including all tests is presented in Listing 5.3.

Listing 5.3 *Linear interpolation of points as implemented into PRISM.*

```

1 public static List<Point> Interpolate(List<Point> points, bool keepPointUp)
2 {
3     // Key: x-coordinate, value: point
4     SortedDictionary<int, Point> interpolated = new SortedDictionary<int,
        Point>();
5
6     for (int i = 1; i < points.Count; ++i)
7     {
8         float x1 = points[i - 1].X; float x2 = points[i].X;
9         float y1 = points[i - 1].Y; float y2 = points[i].Y;
10        int xDiff = (int)x2 - (int)x1;
11
12        if (xDiff == 0)
13            continue;
14
15        for (int j = 0; j < xDiff; ++j)
16        {
17            int xVal = (int)x1 + j;
18
19            Point p = new Point();
20            p.X = xVal;
21            p.Y = (int)(y1 + (y2 - y1) * (xVal - x1) / (x2 - x1));
22
23            if (interpolated.ContainsKey(p.X))
24            {
25                // Add if upper and we want it or if lower and we want it
26                if (keepPointUp == p.Y < interpolated[p.X].Y)
27                    interpolated[p.X] = p;
28            }
29        }
30    }
31}
```

```

29         else
30             interpolated[p.X] = p;
31     }
32 }
33
34 // Return the values (points) as a list
35 return new List<Point>(interpolated.Values);
36 }

```

5.3.5 Design

This section presents the specificities in terms of class design from the last discussed features. The general involved classes are summarized in Figure 5.17.

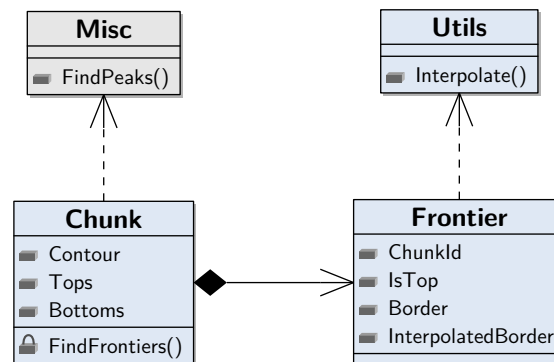


Figure 5.17 Class diagram for frontier extraction.

The frontier extraction is handled by the `CrackedChunk` class itself and is performed automatically at the object's creation by the private method `FindFrontiers()`. The peak detection algorithm used for this purpose was already implemented previously in an external routine from the existing utility class called `Misc`.

The created frontiers are stored and exposed in two separated lists, `Tops` and `Bottoms`. A single frontier is described by its own class `Frontier` which contains several information, mainly the points composing its border in the `Border` property. The interpolation is also performed directly in the constructor, using the `Interpolate` method previously discussed and implemented in `Utils`, which contains utilities used specifically by the cracked records processing and put in the same namespace.

5.4 Groove detection

The particular problem of finding grooves has already been discussed in Section 4.3. Nevertheless, some necessary steps must be done before and after applying the discussed algorithm. These are explained in this section.

The process detailed here is to be applied for each frontier at the top of each chunk, so that every groove can be found over the record. The bottom frontiers do not need to be taken into account. They will rather be used at the tracking step which will be explained in Section 5.5. A simple schema representing visually the result of this step on a chunk can be seen in Figure 5.18. Found grooves are represented as red dots.

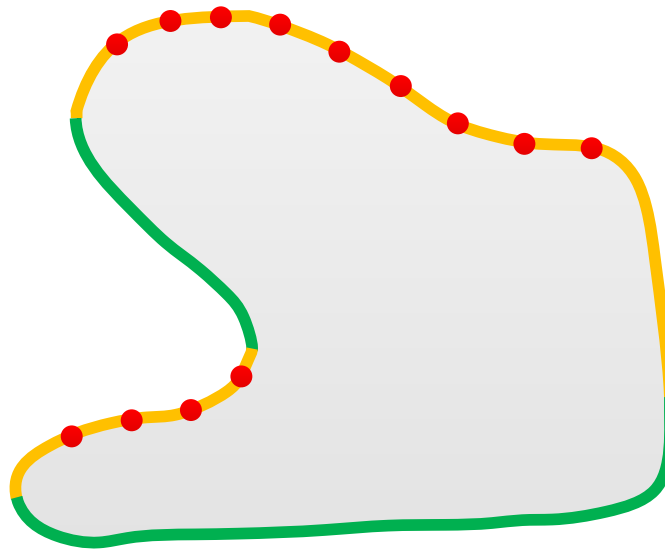


Figure 5.18 *Schema representing the groove detection on each frontiers of a chunk.*

Before talking about the actual algorithm, the next section will explain other routines added during the implementation to improve the groove detection and tracking.

5.4.1 Preparing the binned image

The algorithms that will be explained in this section and the next one depend a lot on the binned image created previously by the program from the acquisition. During implementation, several issues appeared, mostly due to the surface irregularities. Some – e.g. the bad fibers and basic slope correction – have already been explained during the interactive tracking implementation (see Subsection 4.3.2).

Slope correction

The slope correction already implemented in PRISM is applied at loading time, directly on the detailed acquisition and using a fixed user-defined value. During tests, it has been discovered that the slope may change from an area to another, making this feature insufficient. For the existing tracking methods, this is usually not an issue, but the signal was sometimes too degenerate to detect and track the grooves properly.

A routine has been added, enabling too fix the slope on the whole binned image after its creation. This enables to improve the tracking without affecting the processed audio. The principle is to compute a linear regression of all samples from all scans and subtract the resulting line from the initial signal. The result is visualized in Figure 5.19.

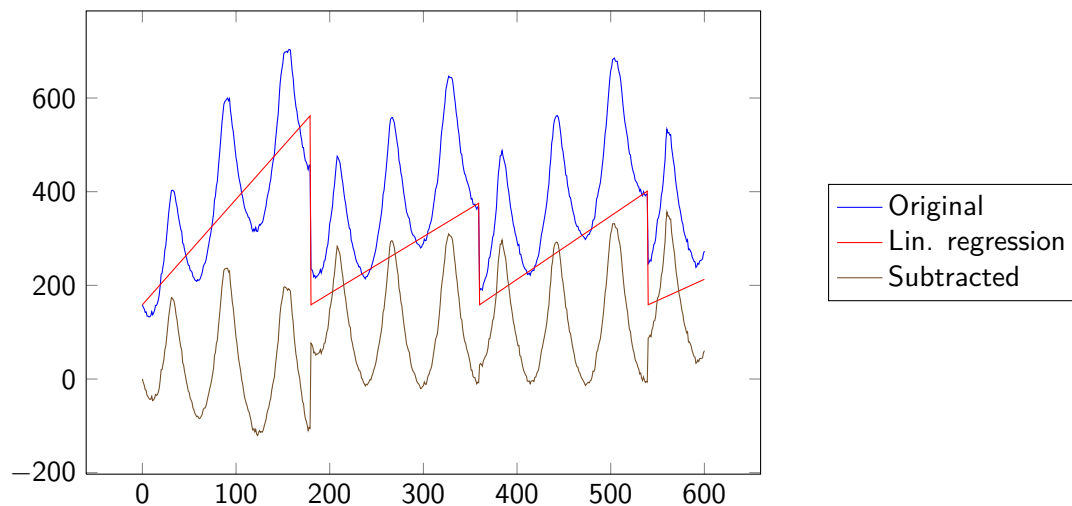


Figure 5.19 *Illustration of the result using linear regression for slope correction.*

This slope correction may be activated by the user by selecting a new option in the GUI (see Appendix C).

Subtract average

A further step, after applying the slope correction, is to subtract the height average so that all separated part have approximately the same height, improving the final signal waveform. To avoid outliers at the scan limits (every 180 points), the starting height of a scan is also equalized the last one of the previous scan, removing the last visible differences.

5.4.2 Creating signal

From the list of interpolated points computed previously (see Subsection 5.3.4), the first task is to get the corresponding signal from the line defined by these points, i.e. the surface height information. At this point, an important thing to take in account is that the tracking is performed over the binned image (see Section 3.3) while the cracks detection has been done on the detailed image, for the reasons already explained in Subsection 5.2.1. The groove finding will also be performed on the binned image. This enables to get rid of bad points values, especially around a cracked area, as the binned image is smoothed.

The values will also be normalized in the same way as seen in Subsection 4.3.2 so that the height differences depend less on the record type. The main difference in the function signature is that the parameter is no more a row index, but the list of previously computed points, as it is not necessarily straight and completely horizontal.

Another parameter will be added, which enables to specify the distance from the actual border from which to get the signal. Indeed, even in the binned image, the values could be altered around a crack. To ensure the location to be properly *inside* the chunk for groove detection, this value will be added to the specified frontier from which to get the signal.

5.4.3 Finding grooves

From the signal, it remains to apply the peak detection algorithm and properly clean wrong values (see Section 4.3.1). Finally, the result of this step is that for each chunk, the starting point of all grooves are detected, to be tracked later on.

5.4.4 Design and class organization

The objects involved in this step are represented in Figure 5.20.

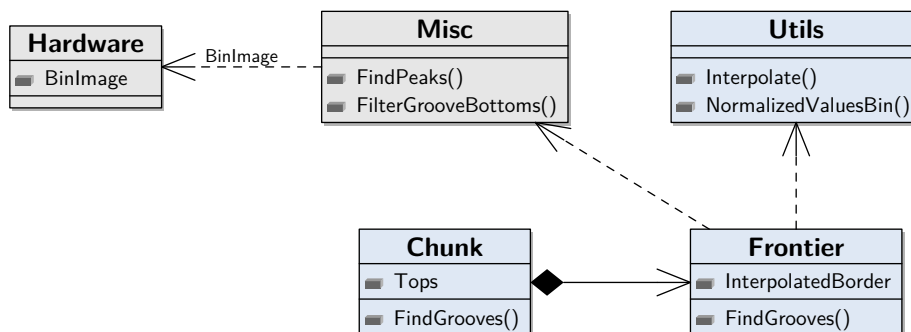


Figure 5.20 Class diagram for groove detection.

The groove detection is handled by the `Frontier` object. The method `FindGrooves()` takes the binned image and other parameters for configuration. After this step, the groove centers are stored and exposed by the `GrooveCenters` property which give the exact points where the groove is starting on the frontier. The `CrackedChunk` class also provides a method with the same signature which will basically apply the groove detection on all its composing frontiers by calling the proper method.

The method for groove finding once again uses `FindPeak()` algorithm but also `NormalizedValuesBin()` which creates the normalized signal from the interpolated points. The method `FilterGrooveBottoms()` verifies if all detected points are really groove bottoms.

5.5 Groove tracking

At this point of computing, the chunks on the record were detected, the frontiers of all chunks in the record extracted and defined whether if at the top or bottom of a chunk, and the groove bottoms are identified at the top frontiers. The next step is to track all the grooves inside each chunk.

5.5.1 Differences from current tracking

PRISM already implements different tracking types that have been discussed in Section 5.1. However, the existing implementation cannot be directly adapted because the point of view is quite different than with non-damaged records. All tracking methods need to analyze the record in its entirety. For example, the tracking depth method follow the groove bottom until the end of a revolution and continues when it is reached until the whole disc is processed. The tracking using Fourier also needs to analyze all rows to discover a common pattern in the groove shapes.

With this implementation, it is a groove *section* that will be tracked, that is, a part of groove inside a chunk. The next sections will explain the used approach.

5.5.2 Tracking overview

Beyond finding the best track matching a groove, this algorithm must also know the limits to be able to properly stop. The main steps are:

1. Start tracking using the position of the detected groove bottom
2. Follow the groove downwards by using an appropriate method to remain correctly at the center (groove bottom)

3. Verify that the position is still on the chunk area. If it is not the case, stop the tracking at this point.

The simplified main algorithm is summarized in Listing 5.4.

Listing 5.4 *Main algorithm for tracking.*

```

1 function Track(Point start) {
2     Point[] points;
3
4     for(int row = start.Y; row < nextStop; ++row) {
5         points.Add(FindCenter());
6         nextStop = FindStopPos();
7     }
8
9     return new Groove(points);
10 }
```

All tracked groove sections are still separated after this step, and attached to a particular frontier. An example of the state while grooves inside a chunk are being tracked can be seen in Figure 5.21.

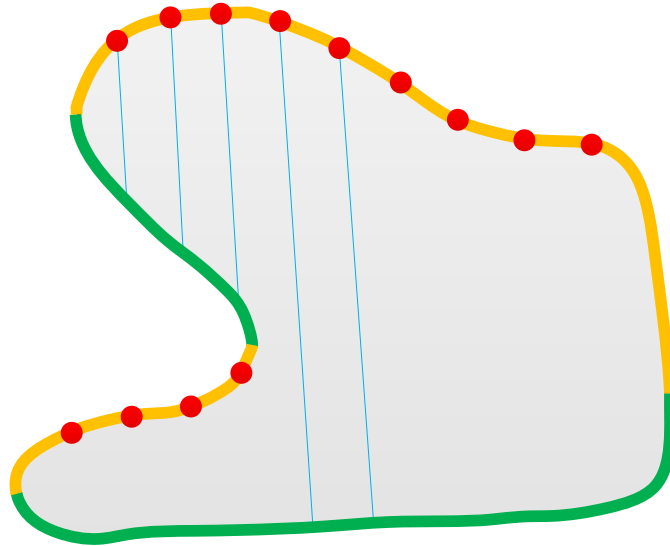


Figure 5.21 *Tracking being applied on a frontier of a chunk.*

At this point, five sections have been tracked. One can notice that not all grooves end on the same frontier. The first three leave off at the same frontier and the last two at the main chunk bottom. A solution has to be found to stop the tracking accordingly.

5.5.3 Finding center

As explained, the starting point is the groove center already detected before. Then, when the groove is followed downwards at each iterations (each line), it remains to find a good way to follow the groove bottom so that the center is properly tracked (`FindCenter()` in the pseudocode).

The main idea is that on the next line of the binned image, the groove center should remain very close to the previously computed one as long as there is no cracks (at this point, no crack should appear inside a chunk). Then, the actual center position is refined using a specific algorithm. The subproblem is then to find the center on a groove line from an approximation.

In the general case, this is not a simple problem. Regarding all record's specific characteristics, there could be a lot of differences. The sensitivity must of course be configurable by the end-user, but there are also some algorithms more adapted depending on these characteristics.

During this project, two different algorithms have been tested. They are described in the sections below.

5.5.4 Tracking using depth

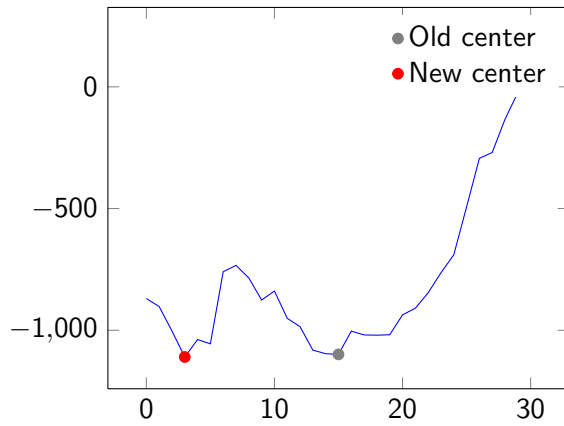
The first algorithm is based on the simple idea of tracking the deepest point at the neighborhood of the previously tracked points. At each iteration, n values before and after the current point are tested. The point with the deepest value is considered at the bottom center.

This tracking method works quite well in practice and rarely fails to follow the groove. However, its main flaw is that it is very prone to local variation, mainly due to the surface or measurement irregularities. An example of this problem can be seen in Figure 5.22.

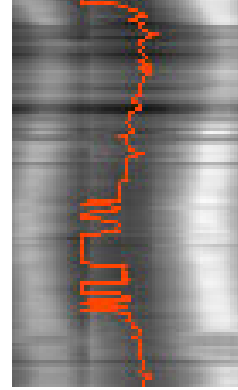
In this case, the groove bottom is not clear, in fact due to deviation from different scans. As the old minimum point is located at 15 while the new one is at 3. The groove will be correctly tracked, but all these irregularities along the tracking will give an inaccurate and noisy result, as seen on Figure 5.22b.

5.5.5 Tracking using curve fitting

To avoid problems with these inaccuracies, a solution is to somehow smooth the values. A simple averaging would however not be a good choice. On the previous example the choice between the two values will not be improved by a simple smoothing. In fact, we know that the general groove shape is, in a lot of cases, similar to a parabola.



(a) A problematic example as seen at a step.



(b) Visualization of the result on the groove section.

Figure 5.22 Example of accuracy problem with the tracking depth method.

Parabola equation

Mathematically, a parabola is the result of plotting a quadratic function such as

$$y = ax^2 + bx + c. \quad (5.8)$$

It has the interesting property of having its only vertex at the coordinate

$$x = -\frac{b}{2a}. \quad (5.9)$$

Therefore, knowing the equation of the parabola enables to easily find the center point, located at the vertex.

Curve fitting

To find this equation, a curve fitting method must be used. This process enables to approximate a mathematical functions that fits some discrete points. There are different kinds of curve fitting, for example polynomial interpolation which tries to exactly match the points at the discrete values. The degree then depends on the number of points.

In our case, we want the general shape to be approximated by a function of the wanted degree. All points do not necessarily have to exactly match the curve as the wanted behavior is too smooth the signal. A well-known method for fitting is for example the least-squares which enables to approximate a function by minimizing the squared error from the approximated curve to the data points. The general shape is then smoothed by a function of the wanted degree.

In fact, curve fitting methods are already implemented in PRISM. They are for example used by a specific algorithm to process cylinders. The `PolyInterpolation` class implements several methods as `fitP()`. By passing two arrays representing the X and Y values and the order of the fitted function, the method automatically returns the corresponding coefficients in an array. Using this method to find the center is represented in Figure 5.23.

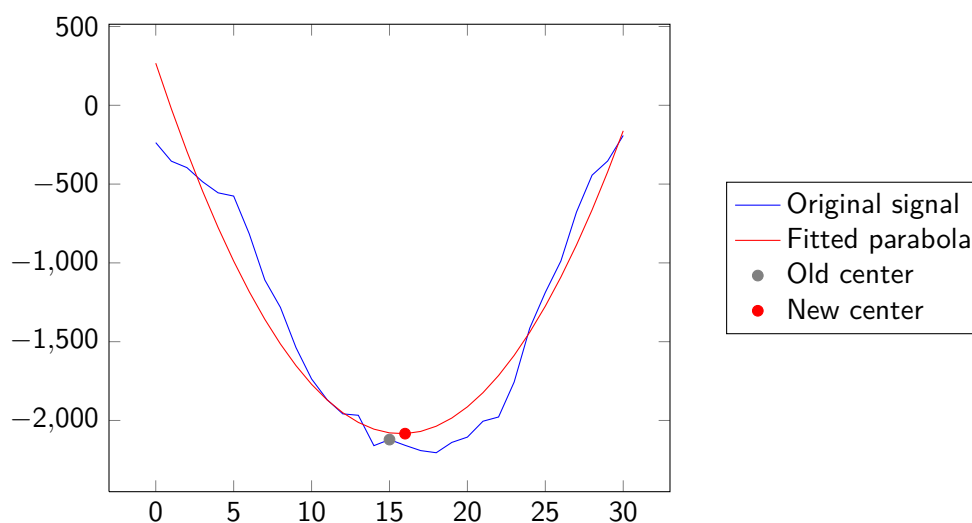


Figure 5.23 Example of tracking using curve fitting. The gray point is the approximation from which the other points have been measured. The red point is the new center value (vertex of the parabola).

On the general results, the curve fitting method allows to track more properly a record, with smoother lines. This difference is obvious in Figure 5.24.

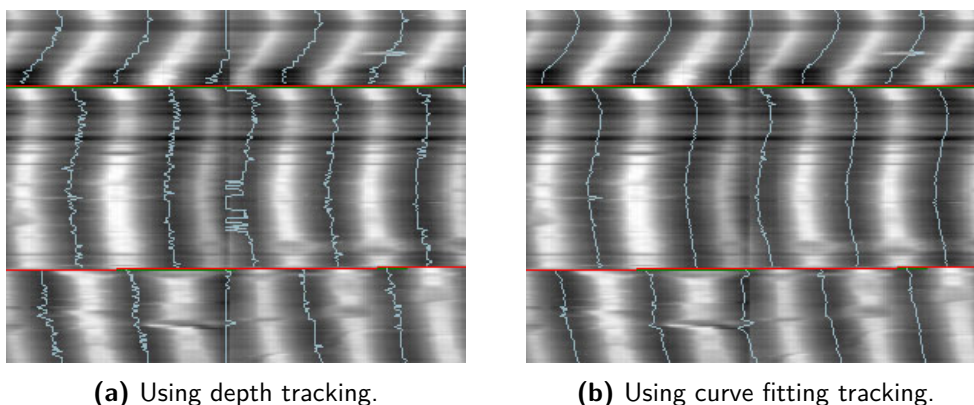


Figure 5.24 Comparison of tracking methods, results on a portion of the Bell record.

Final implementation

As often, the raw results cannot be directly use in some cases. Firstly, what happens if a signal is very noisy or flat? The effective minimum found by (5.9) could be located far away the current area, even out of the record. If the shape is really bad at certain place, the fitted parabola direction could also be inverted, making the vertex a maximum instead of a minimum. Figure 5.25 presents such problematic cases.

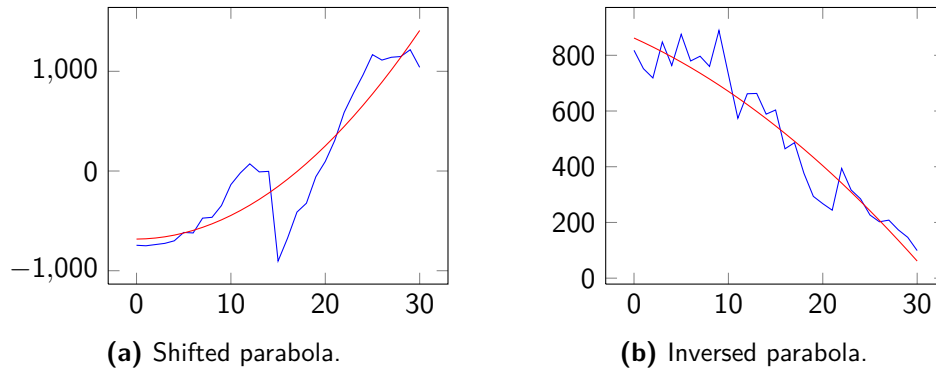


Figure 5.25 Examples of problems with curve fitting method.

This eventuality has to be checked with the proper code. The simple test is to ensure that the vertex is located in the neighborhood by checking the distance from the last point. If too far, the current center may be set to the previous value. The parabola direction can also be tested with the sign of the a coefficient. If $a \geq 0$, the corresponding vertex should not be taken into account.

5.5.6 Stop condition

At this point, the groove center may be tracked with different methods. The remaining work is to find a stop condition for the algorithm, i.e. when the chunk contour has been reached (routine `FindStopPos()` in the previous pseudocode). As seen in Figure 5.21, the groove can reach any of the groove's bottom frontiers.

At each loop iteration, all frontiers at the bottom of the chunk must be tested. If the current point is in the range of the frontier and is still located above it, this frontier is a potential candidate. The next one to be reached is therefore the closest one. This procedure is better explained in Figure 5.26.

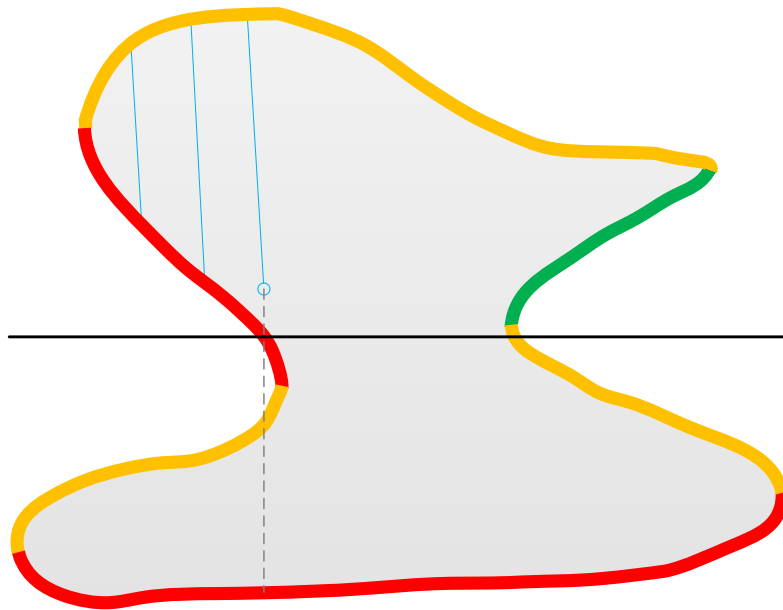
The current point (blue dot) is in the range of the two red-colored frontiers. Both these frontiers are below the point are potential candidates. The closest one is the one above so that the next stopping point, represented by the horizontal line, is at the height of this frontier at this position. The algorithm to find the bottom position is summarized in Listing 5.5.

Listing 5.5 *Pseudocode representing localization of the end position and frontier.*

```

1 function FindStopPos(...) {
2     int nextStop = int.MaxValue;
3     Frontier frontierStop;
4
5     // Test the next frontier to stop at
6     foreach (Frontier b in bottoms) {
7         int height = b.HeightFromX(crtX);
8
9         if (height > 0 && height / binFactor >= row && height < nextStop) {
10             nextStop = height;
11             frontierStop = b;
12         }
13     }
14
15     return nextStop, frontierStop;
16 }

```

**Figure 5.26** *Illustration of the stop condition.*

The routine `HeightFromX()` enables to get the height (or Y position) on the frontier at the current X coordinate. If the coordinate is not in the frontier's range, it returns a negative number. From this height of frontiers, the closest one is stored which corresponds to the stop position and frontier.

5.5.7 Design

As previously explained, the complete tracking of a chunk is quite complicated and has been decomposed in multiple steps. This is also reflected in the final design. The process involves already discussed classes such as `Chunk` and `Frontier` but the processing has been separated in their own modules as viewed in Figure 5.27.

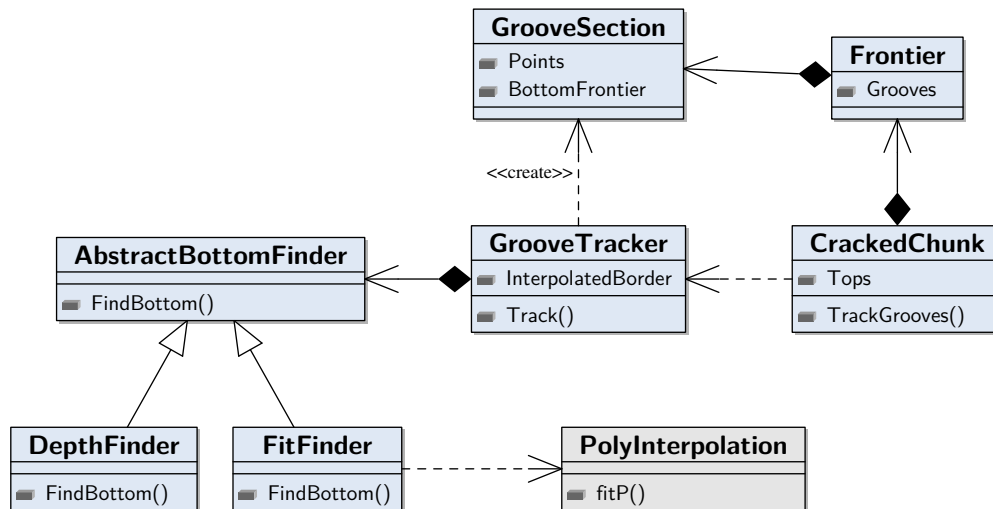


Figure 5.27 Class diagram for groove tracking.

Again, the starting point is the `Chunk` class with the method `TrackGroove()`. As the tracking algorithm needs all frontiers of a single chunk, the process is started by this class. The main algorithm is implemented in the `GrooveTracker` class. It creates the final groove stored in an object of type `GrooveSection` which contains all useful information such as the tracking points and a reference to frontier on which the groove is ending. A list of tracked groove sections is finally stored by the frontier where it starts, but also where it ends (whether the type of frontier it is).

The two different algorithms for maintaining the groove centers are implemented in a class hierarchy inheriting from `AbstractBottomFinder`. To simplify the creation of the desired algorithm, a factory class is provided (not detailed in this diagram).

5.6 Linking and building tracks

At this point of the processing, the last remaining big step is to reassemble the pieces of the puzzle. The list of grooves is known per each frontier where it starts, and the end frontier is also linked to the groove section.

The main linking takes part in three big steps:

Frontier linking Link two frontiers of two different chunks together

Groove linking From the common frontier established in the first point, actually link the grooves together (find corresponding match) by performing an analysis of the signal to find which linkage is more likely.

Track building Once the linking is performed, put all the sections together to build the final track which will be processed by PRISM with the usual algorithms.

These main steps are described in the sections below.

5.6.1 Frontier linking

Before actually linking all groove sections together, a previous step must be performed. The common frontiers must be identified. At this point, the missing piece is a link between the chunks, as all grooves should be entirely tracked. This connection will be done on the frontier basis. The current known information is a groove, which starts on a frontier at the top of a chunk and ends to another frontier at the bottom of the same chunk. Somehow, to eventually link the groove sections together, an association between a *bottom* frontier and a *top* frontier has to be done. An example of correct frontier linking can be visualized in Figure 5.28.

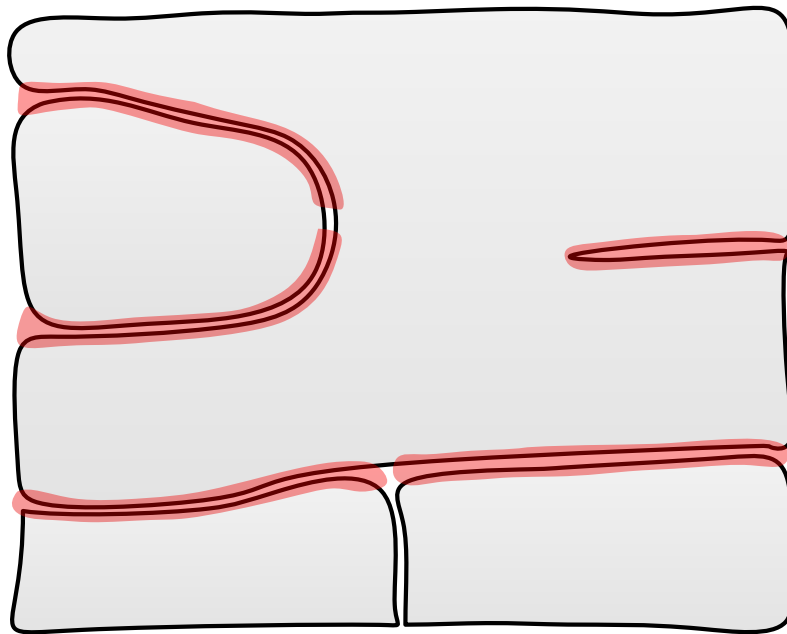


Figure 5.28 Schema representing frontiers linking.

The first observation is that a single frontier can be linked to several different other ones, for example when two chunks are located beside the same frontier. Another important

result is that it is possible that two frontiers of the same chunk have to be linked. This can happen for example in the case of thin crack not entirely detected, as already seen in Figure 5.11.

This step must be seen as a general problem over the record: at this point it is assumed that we have two collections containing all bottom and top frontiers of all chunks in the record. The algorithm iterates over all bottom frontiers and tries to assign their corresponding linked frontiers.

Frontier section

For this process, a new object type has to be defined: a frontier section. A frontier section defines the range of a frontier that has been linked with another one. This range is important as we saw that several frontiers may be linked to a single one. A section stores the following information:

- Reference to the linked frontier
- Index of the first linked groove in the linked frontier
- Size, i.e. number of grooves in the section

A class `FrontierSection` containing the corresponding attributes is defined for the following algorithms. It will also be useful for the next processing step (actual groove linking).

General algorithm

At least one frontier section is created per bottom frontier. Then, the iteration is performed on each groove of the current computed frontier. For each groove ending position, the closest frontier is found. If there is no frontier below this position, this means that the end of a revolution has been reached. It is located at the bottom of the mapped record and no other chunk is present below.

Additionally, when a new section starts, the index of the closest groove index in the linked frontier is sought. The index reflects the position on the frontier where the groove is located. This will temporarily be the groove section that will be linked to the current one. The section is then stored in a collection.

Finally, the process repeats for every groove in the frontier. As long as the closest frontier remains the same, the section is conceptually enlarged by adding a groove. If the closest frontier changes, it means that a new section must be defined. The process then continues similarly. A full example showing the result of this processing is presented in Figure 5.29.

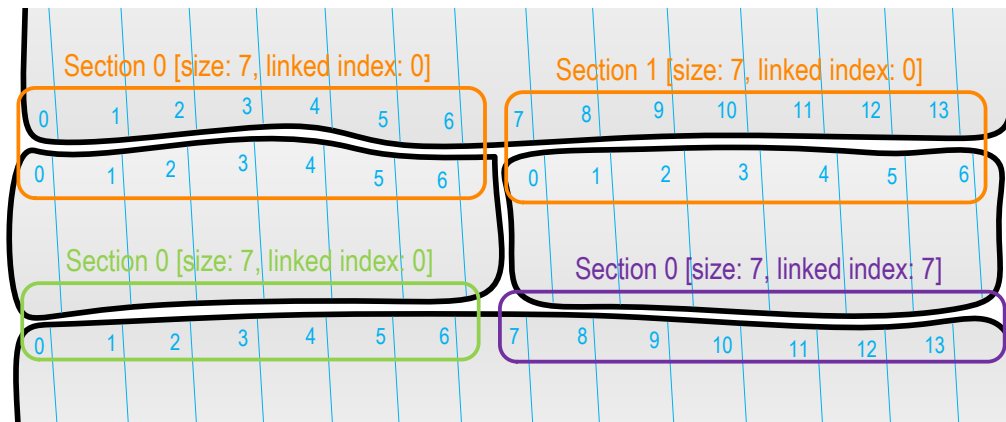


Figure 5.29 Example of the result after frontier linking. The sections of the same frontier are represented by the same color. The section sizes and linked groove indices are also indicated.

The parameters stored by a section are more easily visualized in the corresponding rectangles. This example also shows different situations that may happen. If, as explained above, a chunk is followed by several others, the frontier is split as for the two orange sections in the example. However, for the green and purple sections, the linked frontier remains the same for all grooves. The purple one also shows that the first linked groove is not the first one (index 7).

Following these definitions and explanation, the whole algorithm is summarized in Listing 5.6.

Listing 5.6 General algorithm for frontier linking.

```

1 function Link(Frontier[] bottoms, Frontier[] tops) {
2     FrontierSection[] sections = {};
3
4     foreach (Frontier b in bottoms) {
5         // Sort grooves by X-coordinate (order not always defined!)
6         b.Grooves.SortByXCoord();
7
8         FrontierSection crtSection = new FrontierSection();
9
10        for (int i = 0; i < b.Grooves.Count; ++i) {
11            if (b.Grooves[i].NoBottom)
12                continue;
13
14            // null if no frontier
15            Frontier f = ClosestFrontier(b.Grooves[i]);
16            if (f == null)
17                b.Grooves[i].IsEndRevolution = true;
18
19            // New frontier to create
20            if (f != crtSection.LinkedFrontier) {

```

```

21         int linkedIndex = ClosestGrooveIndex(f, b.Grooves[i]);
22         crtSection = new FrontierSection(b, f, i, 1, linkedIndex);
23         sections.Add(crtSection);
24     }
25     else // Just enlarge the section
26         crtSection.AddGroove();
27 }
28 }
29
30 return sections;
31 }

```

Closest frontier

From this main algorithm, the first subproblem is to find the closest frontier from a given groove. For this purpose, the only choice is to do it geometrically, as there is still no connection between different chunks or frontiers. Basically, the goal is to find the minimum Y distance between all frontiers and the groove's last point, whereas the frontiers are *below* the point.

In specific situations, e.g. when the crack is almost vertical (see Figure 5.11), it is possible that the next frontier is slightly above the end groove position. A small negative number is then allowed. The typical value is for example 5 pixels, so that it would not be possible for a wrong frontier to be detected (the chunk should have a height less or equal to 5 pixels). The algorithm is described in Listing 5.7.

Listing 5.7 Algorithm for finding closest end frontier from a given groove.

```

1 function ClosestFrontier(GrooveSection groove) {
2     int minDistY = Infinity;
3     Frontier found = null;
4
5     foreach (Frontier t in topFrontiers) {
6         int height = t.HeightFromX(groove.End.X);
7         int distY = height - groove.End.Y;
8
9         if (height > 0 && distY >= -MAX_NEG_DIST && distY < minDistY) {
10             minDistY = distY;
11             found = t;
12         }
13     }
14
15     return found;
16 }

```

Closest groove

When the closest frontier has been found, it remains to find the groove in this frontier that is the nearest to the current one. This algorithm consists simply to run through the grooves in the linked frontier, compute the horizontal distance from the current groove and return the one with the smallest distance. It is presented in Listing 5.8.

Listing 5.8 *Algorithm for finding the closest groove.*

```

1 // Get the index of the closest groove to be linked to the next one
2 function ClosestGrooveIndex(Frontier frontier, GrooveSection groove) {
3     int minDist = Infinity;
4     int minIndex = 0;
5
6     if (frontier == null)
7         return -1;
8
9     for(int i = 0; i < frontier.Grooves.Count; ++i) {
10         int xDist = Abs(frontier.Grooves[i].Start.X - groove.End.X);
11
12         if(xDist < minDist) {
13             minDist = xDist;
14             minIndex = i;
15         }
16     }
17
18     return minIndex;
19 }
```

Design

The classes involved in this step are listed in Figure 5.30.

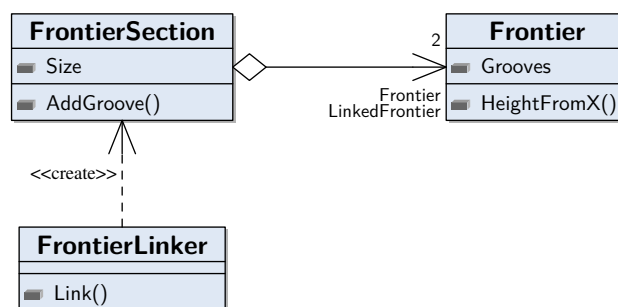


Figure 5.30 *Class diagram for frontier linking.*

The FrontierLinker class contains the algorithms previously explained and will link

all frontiers together. The `Link()` method takes all frontiers in parameters and builds a list of `FrontierSection` objects that is directly returned.

5.6.2 Audio characteristics

From the previous step, all frontiers are now logically linked to another when possible (if not, the frontier is considered as being at the end of a revolution) using a frontier section. The final track could already be linked with the current information if there was no shifting between the chunks (see Subsection 3.6.1). However, the best match is not necessarily from the `closest` groove as it has been computed. In fact, this specific solution would correspond to a match with no shift (or a null shift) between all frontier sections.

Finding the real match is not a trivial task as well. In some cases, the correct link is pretty obvious and can be easily confirmed by the user. Sometimes, it is more difficult to spot. A way of finding the right match is to analyze and compare characteristics of the actual audio signal from a section before – and after – the crack. `VisualAudio` uses such an implementation for the linking and the same kind of analysis will be done for `PRISM`.

Before the comparison, the actual audio signal must first be extracted. Even if the groove is already tracked, in the binned image, the audio must be analyzed from the detailed acquisition. It can be seen as the final processing on a small section of the record. For this purpose, it has been difficult to find a very suitable method. Finally, a comparison method must be used to be able to give the matching likelihood between two audio signals. The general process followed during implementation is explained in the following sections, before detailing the final implementation.

Audio analysis

In the first implementation, the idea was to simply look at the tracked horizontal position and follows vertically the approximated groove center. This early approach suffered from different flaws. Firstly, it is a bad idea to only rely on the tracked position to follow the exact center. The binned image can be decimated at a high factor (e.g. 100), resulting in inaccurate measures. Also, especially on the detailed raw image, some outliers may appear (e.g. because of bad points) distorting the original audio signal.

In a first attempt, the outliers were removed by cleaning the signal afterwards. The method to do so is to compute the signal mean, and then keep track of all Δs_i (absolute difference between a value i and the mean). Then, the average $\overline{\Delta S}$ of all deltas is itself computed. Finally, based on a tolerance factor t , the outliers are fixed in i if $\Delta s_i > t\overline{\Delta S}$. The corresponding algorithm is presented in Listing 5.9.

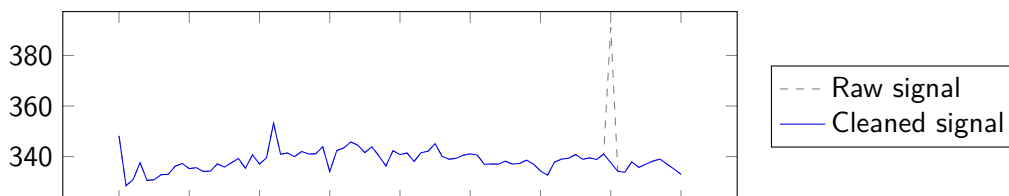
Listing 5.9 *Algorithm to remove outliers.*

```

1 function RemoveOutliers(double[] data, double tolFactor) {
2     if (data.Length == 0)
3         return;
4
5     double avg = data.Average();
6     double[] deltas = new double[len];
7
8     // Compute deltas from average
9     for (int i = 0; i < len; ++i)
10         deltas[i] = Math.Abs(data[i] - Avg);
11
12     double avgDelta = deltas.Average();
13
14     // Clean signal by comparing delta values
15     for (int i = 0; i < len; ++i) {
16         if (deltas[i] > tolFactor * avgDelta) {
17             if (i > 0 && i < len - 1)
18                 data[i] = (data[i - 1] + data[i + 1]) / 2;
19             else if (i > 0)
20                 data[i] = data[i - 1];
21             else if (i < len - 1)
22                 data[i] = data[i + 1];
23         }
24     }
25 }

```

Bad values are replaced with a interpolation of the neighbors. The result of this algorithm can be visualized in Figure 5.31. In this case, especially one big outlier is visible that is removed after the cleaning.

**Figure 5.31** *Illustration of outliers removing on a sample signal (one big outlier).*

Adding curve fitting

Even with outliers correction, the result is not perfect and could be noisy. Another better way to get the audio signal is to use curve fitting, as explained for tracking in Subsection 5.5.5. Thi time, the vertex of the parabola is used for the signal value. However, unlike tracking, the center position is not corrected. Indeed, it varies not much and taking

the vertex is sufficient to get a proper value. This method also has the benefit of smoothing noisy values. The difference with both methods is visualized in Figure 5.32.

As for the tracking, the eventuality of a degenerated parabola must be tested. If it is inversed or the vertex is away from the tracked position, the signal value is computed from the tracked position. The corresponding algorithm is presented in Listing 5.10.

Listing 5.10 *Audio analysis using curve fitting.*

```

1 function FittedSignal(float[] rawImg, int start, int length) {
2     double[] signalRes = {};
3     int startRow = groove.Points[start].Y * binFact;
4     int endRow = Min(groove.Points.Last().Y * binFact, startRow + length);
5
6     for (int j = startRow; j < endRow; ++j)
7     {
8         int binPosX = groove.Points[(j - startRow) / binFact + start].X;
9         int left = binPosX - FIT_WIDTH;
10        int right = binPosX + FIT_WIDTH;
11        double[] valuesX = new double[right - left + 1];
12        double[] valuesH = new double[right - left + 1];
13
14        // Get the line X and height
15        for (int w = 0; w <= right - left; ++w)
16        {
17            valuesX[w] = w + left;
18            valuesH[w] = rawImg[j * hardware.Width + w + left];
19        }
20
21        // Fit parabola
22        double[] par = PolyInterpolation.fitP(valuesX, valuesH, valuesX.
23            Length, Constants.ORDER_PARAREG);
24        double topX = -par[1] / (2 * par[2]);
25
26        // Default: tracked value
27        double finalValue = rawImg[j * hardware.Width + binPosX];
28
29        // If appropriate, fitted value
30        if (par[2] > 0 && Math.Abs(topX - binPosX) <= FIT_WIDTH)
31            finalValue = par[2] * topX * topX + par[1] * topX + par[0];
32
33        signalRes.Add(finalValue);
34    }
35    return signalRes;
36 }

```

The outliers removing is still relevant with this method and may also be used to help cleaning the resulting signal. For example, if the parabola is ill-formed and the value from the tracked position is indeed an outlier, it must be fixed.

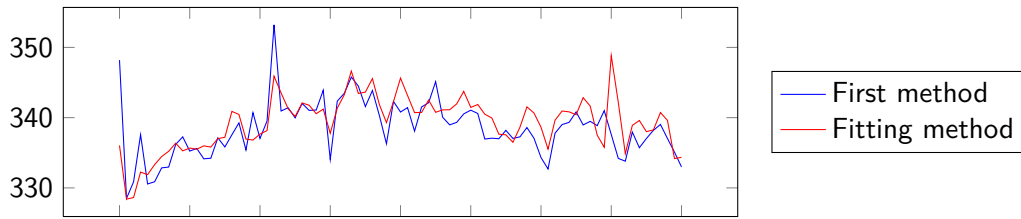


Figure 5.32 *Comparison of audio analysis methods.*

The next sections will explain the different characteristics that will be computed over this audio signal. Some of the implemented techniques are derived from the work in VisualAudio, from [7] and [2].

Standard deviation

The first used characteristic is the standard deviation of the signal. This is the parameter used in VisualAudio for comparison. The standard deviation σ over a discrete set of data is computed with

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} . \quad (5.10)$$

It is a well-known parameter informing on the dispersion of the values around the mean of the signal energy. The main advantage of using this parameter is that it gives a single value which is then easy to compare. However, an important issue appeared in the first tests. In fact, the surface of the record is sometimes distorted because of a crack. This height difference may result in very big signal modification. Therefore, the standard deviation is influenced a lot and become very large even if the signal remains silent.

The way to solve this issue is to apply a high-pass filter over the signal. This surface variation is at very low frequency and it does not come from the audio itself, it can then be removed without any loss on the audio. It has been chosen to use the FFT to apply the filter. Transforming the signal into the frequency domain enables to place the cutoff at a specified frequency. Then, the original signal is converted back using the inverse transform. A visualization on a audio sample is presented in Figure 5.33.

This example is applied on an almost silent section resulting in a very small signal energy. However, the standard deviation on the raw signal would be very high because of the general surface shape. The high-pass filter cleans it using a cutoff frequency and removing values below it.

One can note that the signal is altered at the beginning and the end. This is due to the transformation, particularly to the window function. The details about applying FFT and

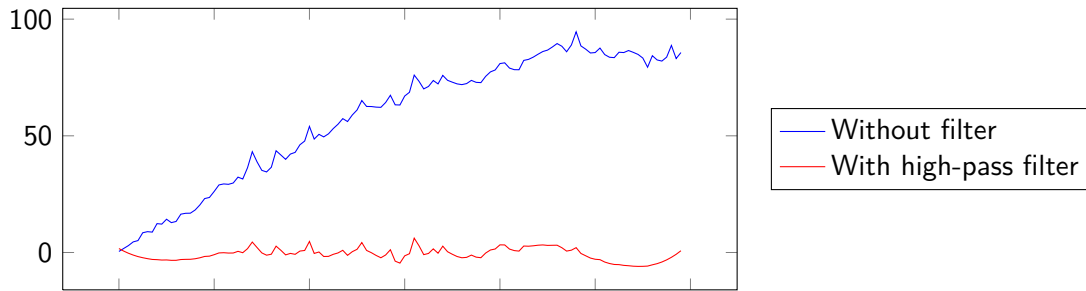


Figure 5.33 Example of high-pass filter applied on a distorted surface. The cutoff frequency is 120 Hz. The original signal has been translated to 0 for a better view.

windowing is explained in the next subsection. The algorithm using existing routines can be summarized in the following steps:

1. Apply the FFT on the original signal by previously using a proper window function (see next subsection).
2. Remove low frequencies corresponding to the cutoff
3. Convert back the signal using the inverse FFT

From this filtered signal, the standard deviation as developed in (5.10) is applied and stored.

FFT

During the project, different solutions have been tried to get a proper characteristic enabling to compare audio signal. With all issues such as surface distortion or outliers, using the FFT has been also implemented. Indeed, the frequency analysis is almost not influenced by these problems.

A `fft()` function is already implemented in PRISM in the `Misc` class, enabling to compute FFT coefficients of data. However, this routine is low-level and requires some pre-processing before using it. Firstly, the parameter is an array of size $2(n + 1)$ representing the complex coefficients. The real parts are indexed at $2i + 1$ and the imaginary at $2i + 2$. Moreover, a precondition of the FFT is that the data size must be a power of two. The function then returns an array of the same size containing the complex coefficients in the frequency domain. In this case, we just need the frequency magnitudes, i.e. the modules of the complex numbers, which is sufficient to know the distribution of frequencies.

Also, another common practice is to use a specific window function to improve the results. A signal is multiplied by a window function to specify an interval of values. These functions have values in the interval $[0, 1]$. In fact, the signal corresponds itself to an observation and

can be viewed as windowed from the first and last sample by a rectangular function (1 from 0 to n , zero otherwise).

The Fourier transform causes what is called *spectral leakage*. A simple periodic function such as $\sin(\omega t)$ results in frequencies other than ω to be viewed in the spectrum. To soften this effect, other windowing functions must be used. In this case, the Hann window have been used, as defined in (5.11).

$$w(i) = 0.5 - 0.5 \cos\left(\frac{2\pi i}{N-1}\right) \quad (5.11)$$

Multiplying the signal by this window smooths the result. The difference with or without the Hann window can be seen in Figure 5.34.

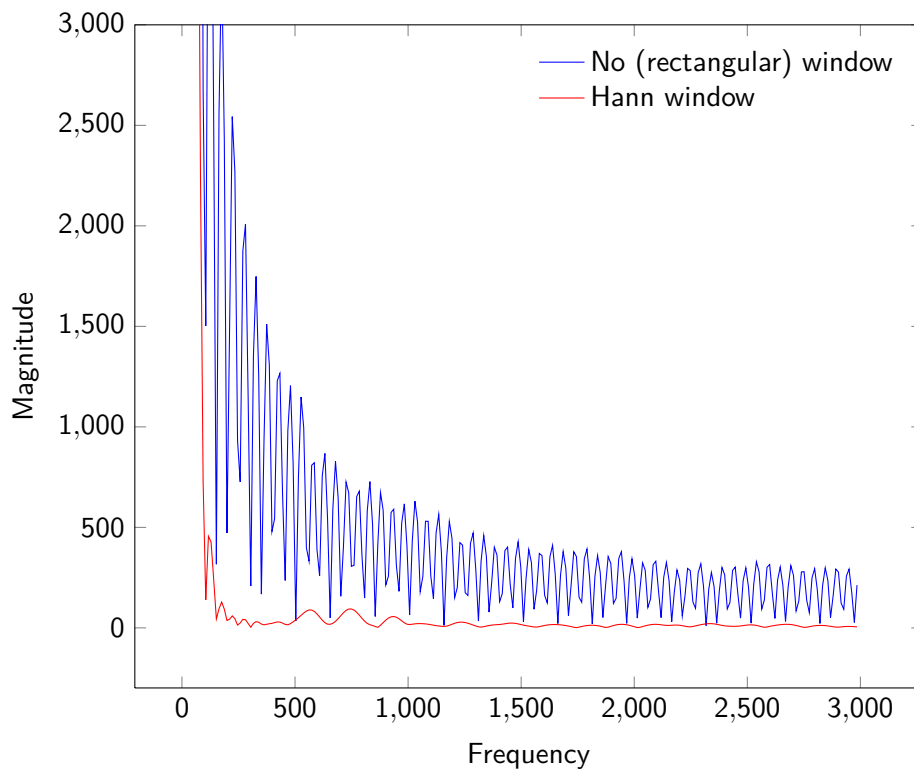


Figure 5.34 Comparison of FFT with the use of different window functions.

The resulting spectrum, using a Hann window suffers much less from the spectral leakage. If the low frequencies were removed and the graph properly rescaled, three harmonics would be more clearly identified between 500–1000 Hz.

To simplify the use of Fourier transforms, some utilities have been added to a class `SignalUtils`. The function `FftAmplitudes()` automatically computes the amplitudes of frequency coefficients from a real signal and applying a Hann window. The FFT size

can be defined or it is determined as the smallest power of two greater than the signal length. When the FFT has been computed, the modules are directly calculated and returned.

To properly compare the spectra, a frequency range must be defined, to compare only the valuable frequencies from the source audio. The `fft()` routine does not return results mapped to actual frequencies. In fact, on a discrete signal, the frequencies depend of the size of the FFT N and the sample frequency f_s . Thus, the corresponding frequency at position i is

$$f = \frac{if_s}{N}. \quad (5.12)$$

Given parameter values, only frequencies between the defined range are then stored as characteristic.

5.6.3 Audio comparison

Now that different properties have been computed at the beginning and end of every groove section, we must find a way to compare them together and get a value indicating the likelihood of a match. Comparing the audio cannot give an exact result in the general case. We cannot say that two grooves match together just because their audio signal is similar. The following list presents different cases that can happen.

Audio variation Firstly, it is quite possible than the audio changes entirely just where the crack happens. In this case, there could be almost no similarity between the two signals.

Silent parts On a part where there is a lot of silence, the sole audio signal is noise, and there is no real audio properties to be compared between two groove sections.

Similar audio If the recorded sound remains similar for a long time (e.g. a repetitive musical section), the differentiation between grooves becomes difficult.

These facts show that it is almost impossible to match only two grooves by a simple comparison. However we need to match entire frontiers and not single grooves. In most cases, there will be multiple groove sections to test between each other. Then, not only the precision is better (because more data can be compared), but the sound is also more likely to vary when the area is larger.

Shifting

In fact, the final result of this step will be the shifting corresponding to the best match between all tested grooves. Different shifting examples are given in Figure 5.35.

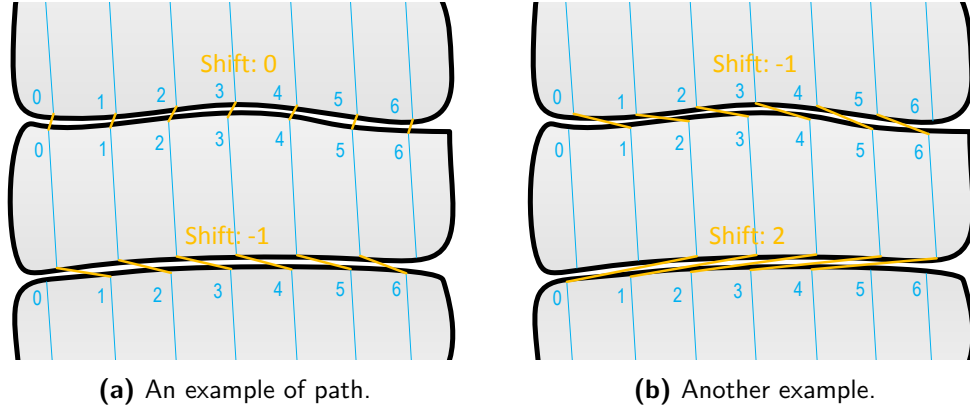


Figure 5.35 *Different examples of shifting leading in different paths.*

In the chosen convention, the shift is relative to the displacement of the first frontier (upper part of the crack) against the other. The class `FrontierSection` already discussed enables to easily change shift and compare values. The two methods `IncrementShift()` and `DecrementShift()` logically change the shift from the current position. `LinkedGroovesCount()` gives the number of linked groove corresponding to the current shift.

To match the frontiers from the computed characteristics, the method `BestMatching()` takes the maximum allowed shift (e.g. if 3, it looks from -3 to 3), tries all positions and sets the shift corresponding to the best match. The likelihood is computed on the maximum number of possible groove sections and averaged before being compared. The comparison ratio between two grooves regarding the different used characteristics is explained in the following sections.

Using standard deviation

Comparing two standard deviation is quite easy. The implementation is based on what is done in `VisualAudio`, especially from the results in [2]. A ratio σi_{rat} is computed between the standard deviations of two groove sections σ_1 and σ_2 with:

$$\sigma i_{rat} = \min \left(\frac{\sigma_1}{\sigma_2}, \frac{\sigma_2}{\sigma_1} \right). \quad (5.13)$$

This results in a ratio in the range $[0, 1]$. To improve the results, this ratio is multiplied by the mean $\bar{\sigma}$ between the two sections. This helps to favor traces with more energy and distinguish between silent and noisy parts. Finally, the likelihood l for a complete match with N groove sections is then computed as in (5.14).

$$l = \frac{1}{N} \sum_{i=1}^N \sigma i_{rat} \bar{\sigma} \quad (5.14)$$

In VisualAudio, there is also a notion of *reliability*. The reliability is a computed value used to determine whether the shift may be fixed as-is or if the user should review the linking. In the current implementation, given the first results, it was decided to show all linkings to the user and therefore the reliability is not calculated. This way, the user may fix any previously computed linking.

Using FFT

Using FFT to compare signal is more problematic. In the first tests, an idea was to find the n greatest peaks in the spectrum representing the most represented frequencies and compare the distance between frequency and/or magnitude. After several tries, it did not give better results and was difficult to implement properly in the general case.

An FFT comparison has however been implemented, which basically compares the overall difference between the two results. As explained in Subsection 5.6.2, the stored spectrum is already filtered to remove very low and high frequencies, so the size is also the same for every sections. A formula to get the normalized distance between two signals \mathbf{s}_1 and \mathbf{s}_2 is then

$$\mathbf{s}_{\text{dist}} = \frac{|\mathbf{s}_1 - \mathbf{s}_2|}{\mathbf{s}_1 + \mathbf{s}_2} \quad (5.15)$$

with the division as an element-wise operation. This results in a distance vector in the range $[0, 1]$. The value tends to 0 when values at the same frequency are very close, and to 1 when they are distant. From this, the likelihood ratio between the two signals can finally be approximated by computing the average and taking the opposite $1 - \overline{\mathbf{s}_{\text{dist}}}$.

Design

The classes organization about signal comparison is quite simple and detailed in Figure 5.36.

The characteristics for the beginning or the end of a groove section are stored in a class called `GrooveProperties`. The corresponding methods `ComputeStdDev()` or `ComputeFft()` compute the corresponding characteristics. Each groove section holds two instances of this class representing the characteristic at the start and the end. These are constructed upon the call of `CreateProperties()`.

The method `BestMatching()` in `FrontierSection` will then try different shifting values, compute the matching likelihood by comparing the characteristics and set the shift according to the best result.

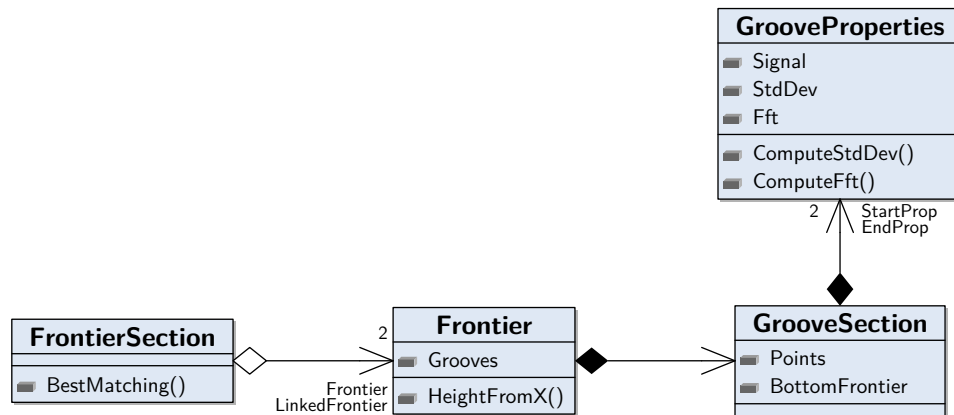


Figure 5.36 Class diagram for audio characteristics computing and comparison.

5.6.4 Track building

At this point, the linking is almost done. Before starting the regular processing from the tracked data, it remains to create this data. The final tracked points are defined as tracks. A single record may have more than one track. For example, it may happen that the groove cannot continue at a defined point if it is not linked. In this case, the current track is stopped and a new one is started from the next scanned revolution. This enables to track records in several steps even if some parts were not entirely well detected.

Overview

Firstly, the sections in a same revolution are linked together. The class `FrontierSection` embeds a function named `LinkGrooves()`, which iterates through all grooves in it and, following the appropriate shift, links them together by setting the `NextGroove` property. At this point every groove section forming revolution is linked.

Then, the grooves section known as *starting* sections must be detected. This is done by keeping track of all frontiers sections that are not linked to other ones, meaning that they are located at the start of a revolution, i.e. at the top of the capture. At that time, they are also sorted by their X-coordinate to ensure the continuous tracking.

The very final step is the iteration over the starting sections. For each of them, the tracked points are added and the next groove followed until a revolution is reached. Then, it continues to the next starting sections until the whole record is scanned. The corresponding algorithm is summarized in Listing 5.11.

Listing 5.11 Algorithm for final track building.

```

1 function LinkFrontier(Frontier[] tops, FrontierSection[] sections) {

```

```

2      // Link grooves together between cracks from the selected shifts
3      foreach (var s in sections)
4          s.LinkGrooves();
5
6      // Find and sort starting grooves, not detailed here
7      GrooveSection[] startGrooves = FindStartGrooves(tops, bottoms);
8      startGrooves.SortByXCoord();
9
10     // Follow tracks and return them
11     return FindTracks(startGrooves);
12 }
13
14 function FindTracks(GrooveSection[] startGrooves) {
15     LinkedTrack[] tracks = {};
16     LinkedTrack crtTrack = new LinkedTrack();
17
18     foreach (var groove in startGrooves) {
19         crtTrack.AddPoints(groove.Points);
20         GrooveSection crtGroove = groove;
21
22         while (crtGroove.NextGroove != null) {
23             crtTrack.AddPoints(crtGroove.NextGroove.Points);
24             crtGroove = crtGroove.NextGroove;
25         }
26
27         // Not in the end of a revolution, start a new track
28         if (!crtGroove.IsEndRevolution) {
29             tracks.Add(crtTrack);
30             crtTrack = new LinkedTrack();
31         }
32     }
33
34     // Add last track if any
35     if (crtTrack.NbPoints > 0)
36         tracks.Add(crtTrack);
37
38     return tracks;
39 }

```

One can note that this algorithm continues to link from the starting grooves without actually verifying if the end of the revolution matches correctly. In fact, this is a willing behavior. As with other tracking methods, the result will be shown to the end-user. This enables to clearly identify if the match is correct, i.e. if the links between the bottom to the top are vertical (they should have approximately the same horizontal position). Otherwise, the user can use the interface to easily correct the matches (see Appendix C).

Even with an incorrect match, the processing is still possible. This can be useful, in the case of imprecise crack or groove detection, if it is not possible to get a correct result even manually. An example of a correctly tracked record can be seen in Figure 5.37, as it is viewed in PRISM on the binned image.

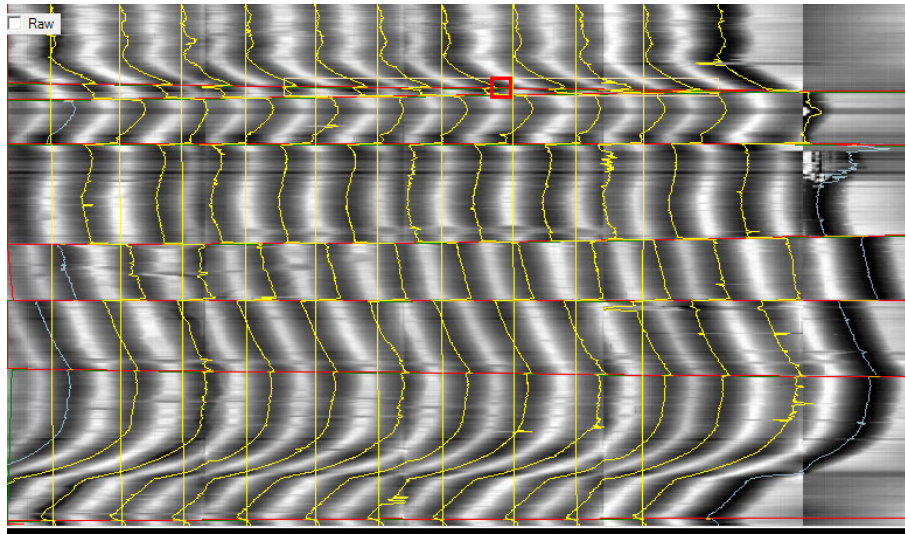


Figure 5.37 An example of cracked record correctly tracked, with final correct matches. The links from bottom to top are then vertical.

Design

This section presents an overview in terms of classes organization. The design with involved classes is summarized in Figure 5.38.

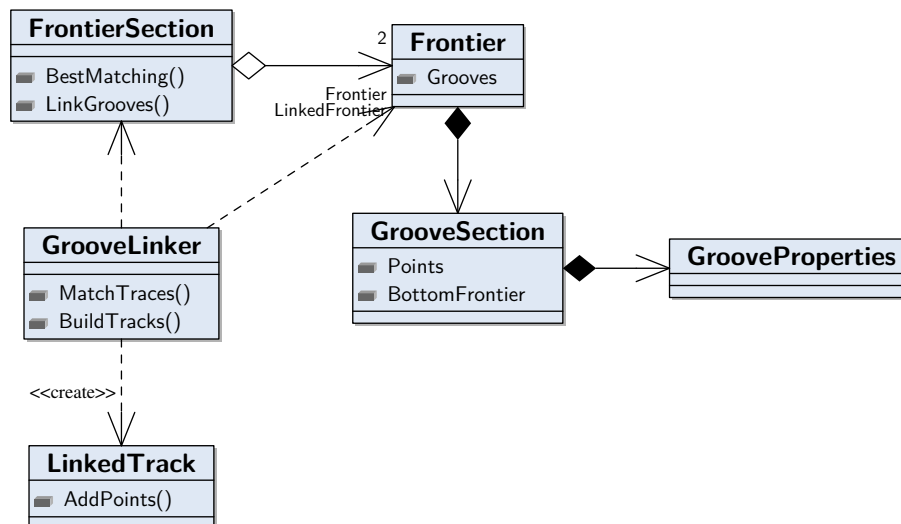


Figure 5.38 Class diagram for groove linking.

The main point is the `GrooveLinker()` class which implements the operations detailed above. For the best matching, it indirectly uses the properties of the grooves attached to the corresponding frontiers, through the `FrontierSection` class. Then, the `Build-`

`Track()` method returns the result as a list of `LinkedTrack` objects.

5.7 Integration and other considerations

5.7.1 Integration in the existing software

As seen in the previous sections, the whole procedure to automatically build an appropriate tracking on a cracked records requires a lot of different steps, and thus the final architecture becomes rapidly complicated.

To simplify the integration and use within the current PRISM implementation and avoid bloating the existing code base, a class named `CracksProcessor` has been added. This latter acts as a facade between the classes described in the sections above (representing business objects) and the rest of the program. It simplifies the whole process by providing simple functions and using the appropriate submodules. Moreover, it is useful to make the link between all parameters that are configurable by the end-user. A summary of the main classes composing the new module and their integration in the rest of the program is presented in Figure 5.39.

5.7.2 User configuration

`CracksConfig` is a user interface control. This control is embedded in the main form (class `frmMain`) and displays all configurable parameters related to the processing of cracked records. In the GUI, this is displayed in a new tab right next to the interactive panel tab already added before (see Section 4.2). The description of the parameters is explained in detail in the User Manual (see Appendix C). The `CracksProcessor` class uses it to get the current user configuration.

5.7.3 Drawing result

The `ObjectDrawer` class is a utility to draw all results, as the actual tracking on the record, the position of the chunks or the grooves on the existing panels in PRISM. An example is shown in Figure 5.40.

The existing drawing methods, `pict_paint()` and `snd_paint()` are already implemented in the `frmMain` class. To draw these new objects, they call the `DrawObjects()` method to the facade which then dispatches the drawing properly. The exact same code is used to draw the objects on the binned or the detailed image. The corresponding matrix transforms are computed priorly in the paint methods (this technique is described in Subsection 4.2.4).

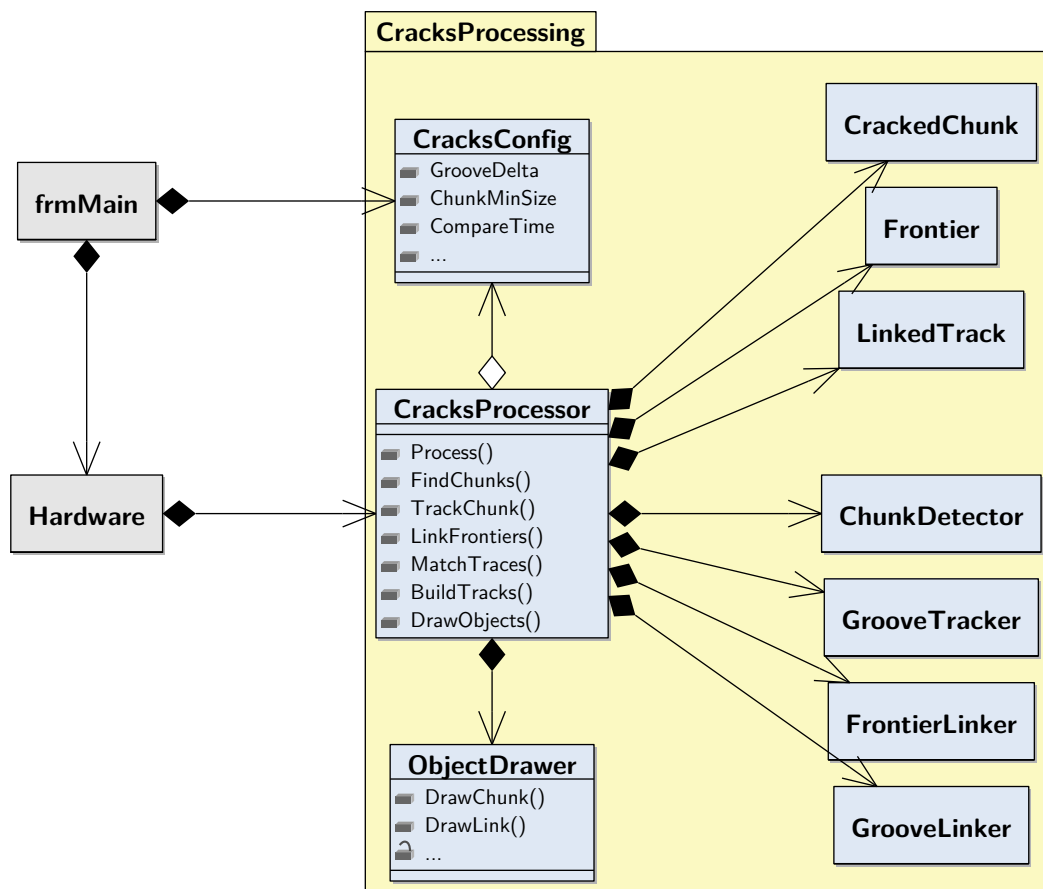


Figure 5.39 Class diagram representing the automated processing integration into the existing PRISM program. The links are simplified to the main compositions between the principal objects.

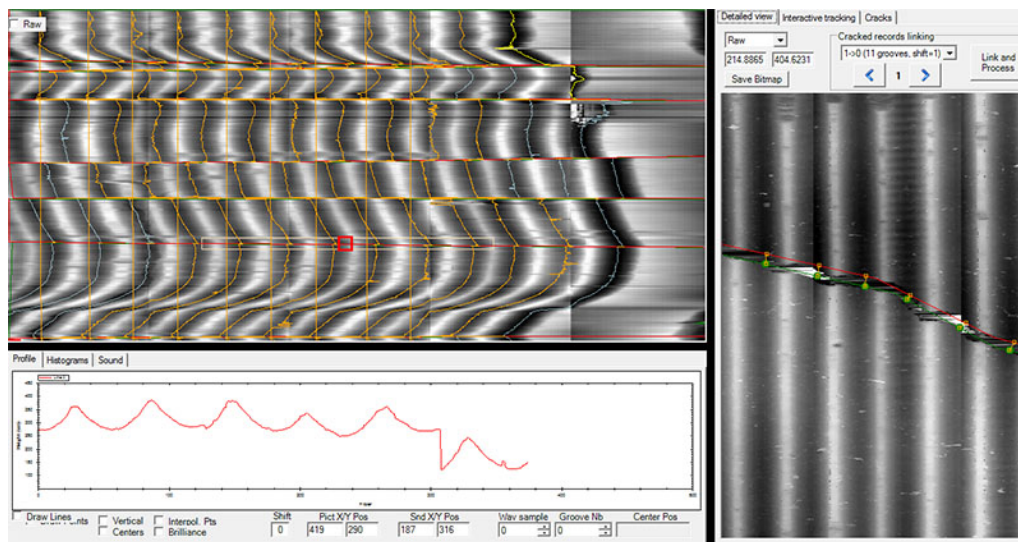


Figure 5.40 Tracking, cracks and groove bottoms as drawn in PRISM. The left panel shows the crack borders and the tracking in the binned view. The right panel shows the links between two chunks.

Chapter 6

Tests on sample recordings

This chapter presents an overview of the final tests that have been performed in order to test that the final solution is functional and behaves in a natural way for the end-user. The first section will explain the tests and results about the interactive tracking whereas the next one will talk about the automatic processing

The sample recordings will be used to apply the tests. The exact parameters used for the test as well as the final resulting audio may be found in the external files (see Appendix A).

6.1 Interactive tracking

This part of the tests corresponds to a GUI and final user test. The goal is to verify that the behavior corresponds to the user expectation in all situations. The tests are applied on all records except the Bell disc, which has been successfully processed with the automatic tracking (see Section 6.2). The two first tests are done with the implementation in PRISM and the last one (Brigance) with RENE.

6.1.1 Dickson cylinder

Tracking this record is quite easy with the interactive tracking. On the main part, there is only one big crack and scratch, but the shift is not big and easily findable. Once the pattern is found, it may be copied multiple times, so that the record is quickly tracked. A part of the result is viewed in Figure 6.1.

The first run (run A) of this record has been recovered with this method. The tracking has been applied in three steps from different ranges. The first part is from 6 to 10 (there is no audio before) which contains the violin music. the second part is from 11 to 18 and contains speech, then the music starts again in the last part from 18 to 20.

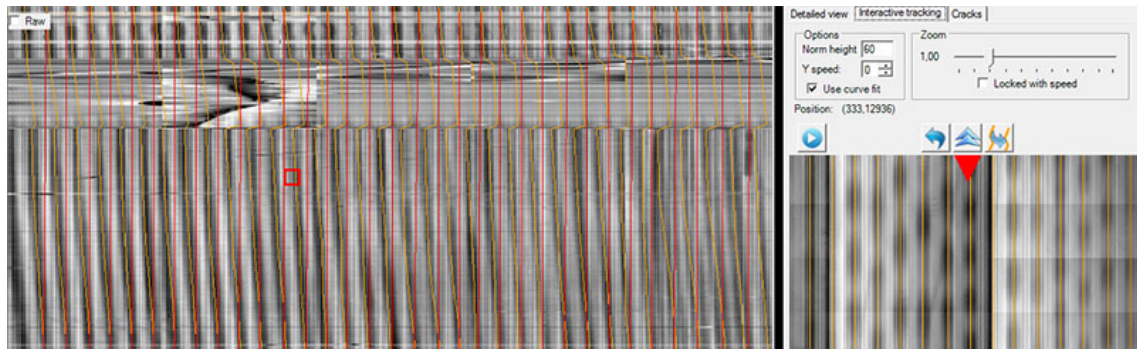


Figure 6.1 *Test of interactive tracking on the Dickson cylinder.*

The final sound quality is correct though the volume is quite low. The music is audible but the speech difficult to understand. In this test, it has not been tried to optimize the processing part. However, the blob correction feature has been used to minimize the noise.

6.1.2 Boas recording

This record is more difficult to track. The main point is to find the correct shift between the cracks, which is quite tricky in the first place. However, as usual, when a first revolution is correctly tracked, tracking the rest of the record is very quick. An overview of this record correctly tracked is presented in Figure 6.2.

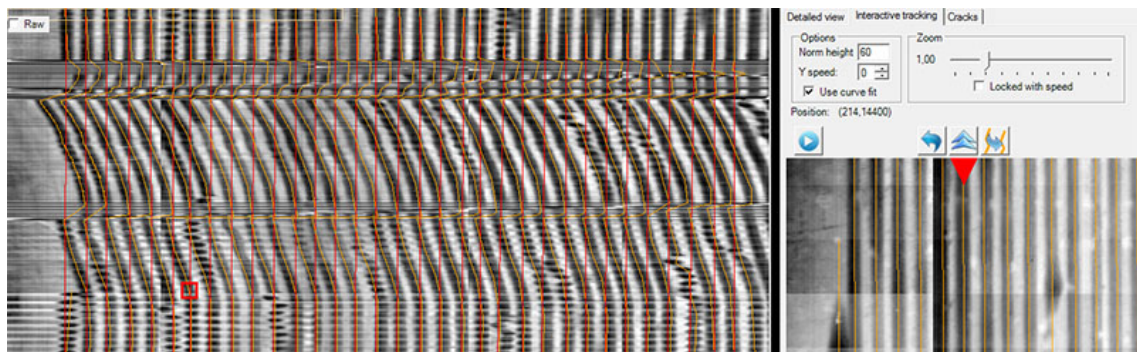


Figure 6.2 *Test of interactive tracking on the Boas recording.*

The first run (beg 3) of this record has been successfully recovered. The tracking has been applied in two different steps, from range 1 to 6 and 6 to 10. It seems to be a recording of a singing with percussions, probably recorded from a Native American population. Because of noise, the final audio quality is very bad and the processing should be tuned for a better result.

6.1.3 Brigance recording

This record has been used to test the version of the interactive tracking implemented in RENE. The result for the tracking is good, as viewed in Figure 6.3.

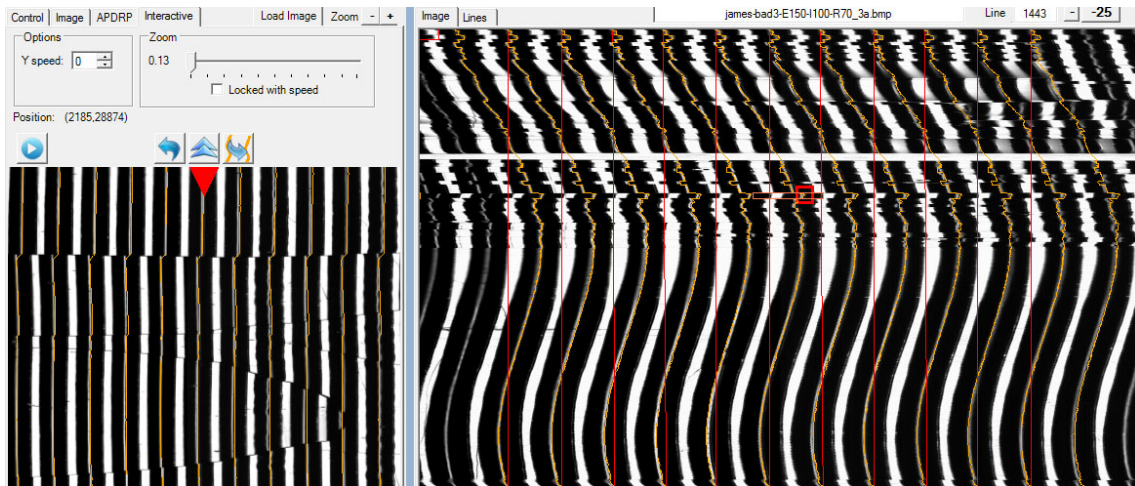


Figure 6.3 *Test of interactive tracking on the Brigance recording.*

The assisted tracking (see Subsection 4.2.1) works also well on this record, centering to the position where the brightness is the highest. The audio result is a speech but the sound is once more very noisy. As explained in Subsection 3.7.4, the record also has some bubbles on its surface. Therefore, the processing step is not able to find properly the edges when the groove bottom is blurred.

6.1.4 Summary

These tests enabled to show that the interactive tracking works well in different situations to help the user tracking some records. All tests worked properly and allowed to output a final audio result. Of course, it would take some more time to improve the sound quality, by correctly tuning the processing options.

Also, as a result from these tests, some future useful improvements for this feature have been identified. These will be explained in Section 7.2.

6.2 Automatic tracking

The second main part of the project was about developing a solution able to automatically track some cracked records. From the implementation, a lot of different issues have been identified. The two most difficult parts are the crack and groove detection. If the cracks

or the grooves are not properly detected, the whole mechanism cannot work. Indeed, the link cannot match if a groove is missing, as seen in Figure 6.4.

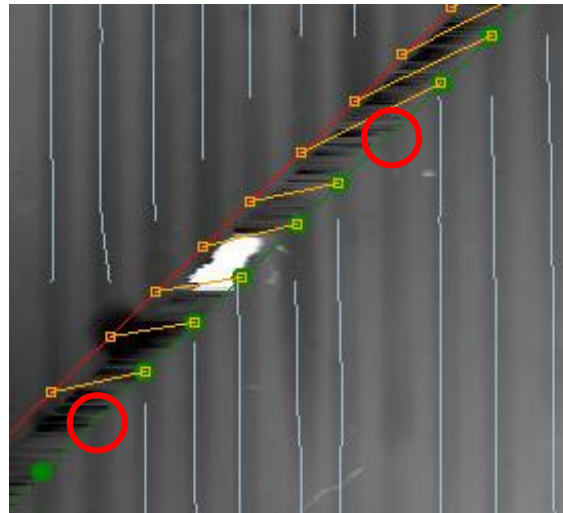


Figure 6.4 *Examples of grooves not detected on a frontier. This makes the linking unusable.*

Another important point is the tracking, which is difficult to perform if there are a lot of scratches, making it impossible to work properly. If the tracking moves out of line, it may be mingled with another groove section.

Finally, the linking, which is only based on audio comparison, is not precise enough in all cases. It is quite good when the comparison occurs between silent parts and well-formed audio. However, if the audio remains similar or silent, the user may have to fix the final linking. In most cases it is quite easy for a human to fix it, by viewing the general record geometry.

6.2.1 Bell disc

This record has been entirely restored using this feature. The cracks are detected quite easily because the brightness is very different compared to the regular surface (see Figure 3.16). Also, all the grooves are found because they are deeply engraved to the surface. However, the binned image must be transformed with the fixing algorithms (see Subsection 5.4.1) because the surface height is very changing, both because of the scan (probe not aligned) and the cracks (the surface is sometimes altered when it occurs).

The result after restoring the second part of the record is viewed in Figure 6.5.

The record has been recovered in four steps, as it has four clearly separated tracks. A talking man can be heard on all these parts, though it is difficult to understand what he says because of the high noise level. However the tracking is consistent for all parts in this example.

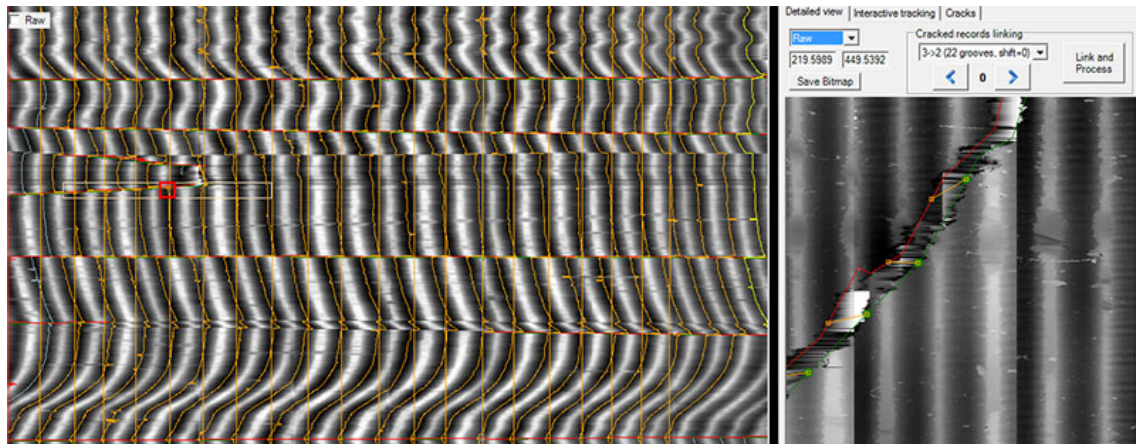


Figure 6.5 *Tracking of the Bell disc using automatic tracking.*

6.2.2 Dickson cylinder

This record was more problematic to track automatically. In fact, the cracks are pretty well detected, but the big scratches result in weirdly-shaped chunks, making the tracking very difficult. Moreover the groove detection and tracking is difficult because the grooves are very weakly engraved and quite thin.

In the end, with proper parameters, although almost all grooves are correctly tracked, the linking step does not work and the automatic tracking is unusable. An overview of the result in a section of the acquisition is presented in Figure 6.6.

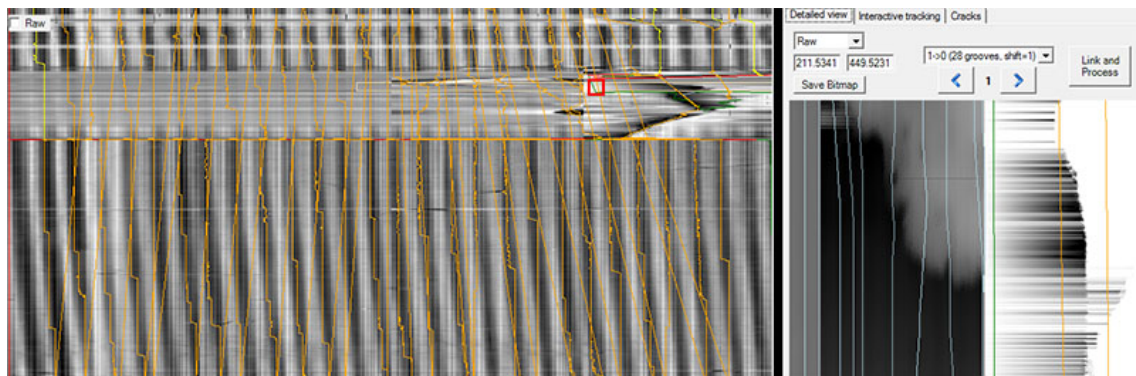


Figure 6.6 *Tracking of the Bell disc using automatic tracking.*

In this example, the left part is correctly tracked. The main problem comes with the big scratch on the right side. As seen in the detailed view, the surface at this position is completely distorted, without any groove seen, which makes the linking impossible.

6.2.3 Boas recording

Another test was done on the Boas cylinder. The results are quite similar to the Dickson one. Again, even with very thin cracks the detection looks good, but some missing grooves and bad values because of ill-shaped cracks give an unusable result, as seen in Figure 6.7.

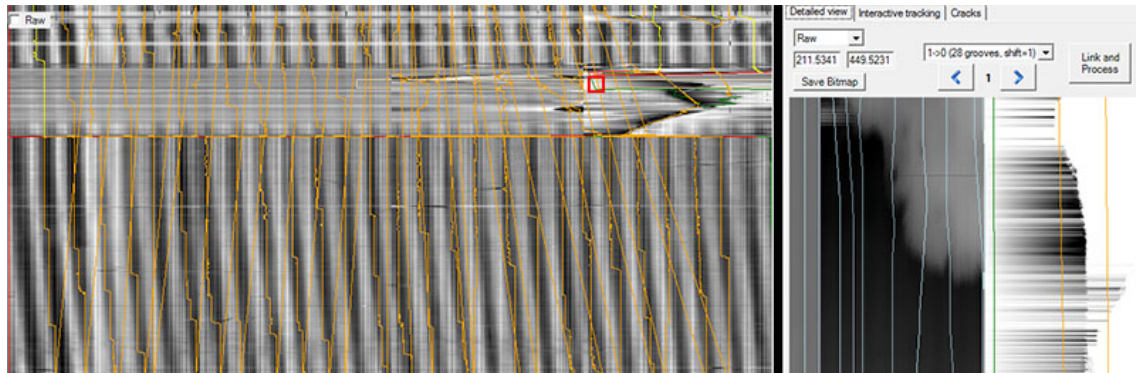


Figure 6.7 *Tracking of the Boas cylinder using automatic tracking.*

Almost all cracks are correctly tracked, but the final linked result is inconsistent.

6.3 Conclusion

These different tests have shown that the interactive tracking is an interesting feature to help the user on any type of record, even deeply damaged. The new features enhance the productivity by providing “shortcuts” and new tools.

On the other hand, the automatic tracking, even if working on specific records, is not an appropriated tool in the general case. In fact, each internal function – cracks detection, chunk detection, audio comparison – represents a new “brick” that may help tracking a record. However, put together, it is very difficult to get an appropriate result in every situation.

This leads to the idea of a possible solution that would stand in between a fully manual or automatic system. Specific parts that are problematic could be done or fixed manually by the user, like drawing a chunk border by hand, indicating missing grooves or fixing a bad tracking. These improvements will be explained more specifically in Section 7.2.

Chapter 7

Conclusion

To conclude this thesis, the current chapter reviews the initial objectives and summarizes problems encountered during the project. It also proposes different views for future work and improvements based upon the current state of the project. Finally a personal conclusion is presented.

7.1 Objectives review

The main objectives of the project are detailed in the requirements (Subsection 3.1). A summary of the achieved objectives is presented in the next sections.

7.1.1 Analysis

The first main objective was about analyzing the existing programs and becoming familiar with the codebase and the involved algorithms. After a phase of pure code analysis, the majority of the application was gradually studied while adding the first new features, with the early tests of the interactive tracking feature.

An analysis of the cracked records algorithm in VisualAudio was also made, but more theoretically by studying [7]. Later in the project, the differences between the systems showed that although the basic idea could be used, the implementation would be quite different and require specific solutions.

7.1.2 Implementation

The main part of the project was to find different ways to restore cracked record in an efficient way. The work to improve the manual tracking led the new interactive tracking,

adding different features which proved to be very useful for the user. The feature has been implemented both in RENE for 2D acquisitions and in PRISM for 3D.

As a further step, the next goal was to find a way of automatically processing some damaged records, as it is possible in certain situations with VisualAudio. The final implementation is a complete system which tries to find the cracks for tracking damaged records. This solution works in some cases but may still be improved in several ways (see Section 7.2).

Regarding the available recordings and the types of damage, only an implementation for PRISM has been completed. The specific damages on the Brigance disc, the only tested 2D acquisition, were almost impossible to find with image processing. Moreover, this record suffers from more important problems such as focus and is easily tracked with the interactive tracking.

7.1.3 Other

About integration and avoiding regression, the new features have been implemented as separate as possible from the rest of the program, so that it is easily maintainable (for example with the use of a facade). A secondary objective was to improve the software architecture. Some technical details have been improved and some bugs fixed, but for the general design, a deep software reengineering should be applied, which was out of the scope of this thesis.

7.2 Future work and improvements

The final results from the researches conducted during this thesis open the way to many possible future work. Not only some specific points regarding the implementation may be improved but some new ideas also emerged given the current stage. Some of these are explained in the next sections.

7.2.1 Interactive tracking

The interactive tracking already offers some useful new features. While tracking records, other ideas came to improve it. Firstly, it could be useful to be able to fix an area already tracked, without being forced to cancel everything after it. One could imagine a way of moving or removing an existing point by clicking on it.

Another idea is to be able to “jump” over a crack while tracking, that is, to be able to cut off the tracking in several pieces (with a result similar to what is done in the automatic tracking). Until now, the path is always continuous and the user just passes through

the cracks. In the final audio, this often results in some heavy clicks adding disturbing noise.

As a detail, one can also note that in the final implementation the groove center is always automatically fixed (using fitting or depth correction). In some bad areas, the user might want to exactly set the position and avoid badly corrected center. It could also be possible to set the maximum deviation (currently fixed from the groove width).

7.2.2 Cracks and groove detection

The cracks and groove detection was the most problematic part during the implementation. Particularly for groove detection, there is a lot of other possible things. The peak detection followed by a cleaning pass is not really satisfactory. In some cases, it would be better to base the detection on a Fourier analysis in a way similar to the existing Fourier tracking. Knowing the phase from the groove signal could give results less sensitive to outliers.

Regarding the cracks detection, one possibility is also to allow the user to draw the chunk borders, for example on the binned image. On certain records, the main chunks may be obvious to the human eye. In fact, an early attempt in this direction has already been tested. In the `CracksProcessor` class, a method called `ProcessTestManual()` has been implemented, that embeds fixed-shaped chunks. An adaptation of the user interface would allow to interactively create those chunks.

In a step beyond, one could imagine to allow the user to fix or manually perform any particular step resulting in wrong effects. The rest of the mechanism would be able to do the remaining tasks automatically. The current implementation is flexible enough to be used from any step to another.

7.2.3 Tracking in chunks

The implemented tracking methods could be also improved. It would also be useful to detect when a groove is mingled to another, warning that one of the initial grooves is badly tracked.

7.2.4 Linking and audio analysis

As already stated, the audio analysis for linking is not perfect for all cases. It would be possible to improve it by refining the signal creation or the comparison formulas. For example, the high-pass filter has been implemented from scratch. Particularly, it sets all coefficients to zero until the cutoff frequency is reached, which may lead to unstable with FFT transforms. Instead, it would be possible to use a common filter, for example Butterworth.

To conclude, it might be worth trying a totally different approach. In essence, analyzing the audio is not sufficient for comparing grooves. One could imagine a solution using the geometry, e.g. if two grooves are the first ones from the record border, they should probably be linked together.

7.3 Personal conclusion

From a personal perspective, working on this project has been very informative and rewarding. It allowed me to learn several aspects of signal processing by contributing to an actively used and maintained project.

Of course, in the beginning it was not easy to get used to all theoretical principles and to apply them to an existing complex software. But finally, I think that all efforts enabled to bring new ideas and tools that will help people involved in the restoration of recordings with IRENE. Also, the fact that the final outcome is tangible (i.e. hearing sounds from historical and early recordings) was really motivating.

More generally, the experience of living in Berkeley for a whole semester was a great opportunity, allowing me to work in a famous Laboratory. I discovered new places, met new people and could practice English speaking and writing.

Glossary

3D Probe The second system developed at LBNL to extract the sound using a 3D probe. 1, 2, 9, 13, 18, 22, 24–28, 37, 53–55, 63

API Application Programming Interface, software interface for a reusable component, such as a library.. 18, 38, 42, 43

FFT Fast Fourier Transform, well-known algorithm to compute a Fourier transform in a discrete way.. 18, 94–97, 99

GDI Graphics Device Interface, API embedded in Microsoft Windows systems to draw 2D graphics to the screen or other devices.. 37

GUI Graphical User Interface. 16, 17, 21, 43, 66, 76, 103, 107

IRENE Image, Reconstruct, Erase Noise, Etc. The first system developed at LBNL to extract records using optical methods. 1, 2, 9, 13, 14, 19, 22, 27, 28, 42, 66, 116

LabVIEW Laboratory Virtual Instrumentation Engineering Workbench. Environment and development platform mainly used for instrument control and data acquisition. 9, 20

LBNL Lawrence Berkeley National Laboratory is a scientific laboratory from the U.S. Department of Energy located in Berkeley, California. 1, 3, 8, 9, 28

PRISM The software used to process acquired records in the 3D Probe system. 20, 21, 25, 27, 29, 32, 42–44, 47, 48, 52, 54, 59, 62, 63, 66, 76, 78, 82, 86, 91, 95, 101, 103, 107, 114

RENE Reconstruct, Erase Noise, Etc. Name of the software that processes acquired records in the IRENE system. 1, 16, 20, 29, 32, 42–44, 51, 107, 109, 114

VisualAudio Solution developed at the College of Engineering and Architecture of Fribourg in order to archive and restore old discs.. 1, 3, 54, 63, 66, 67, 91, 94, 98, 99, 113, 114

WAV Contraction for Waveform Audio File Format, a standard used to store audio data stream on file systems. 16–18, 22

Bibliography

- [1] Alain Benninger. “2D and 3D Scanning of Metallic Records and Masters”. Bachelor’s thesis. College of Engineering and Architecture of Fribourg, Aug. 2012.
- [2] Alain Benninger. *Chunk Linking of Records*. Semester project. College of Engineering and Architecture of Fribourg, May 2012.
- [3] Eli Billauer. *Peak detection using MATLAB*. URL: <http://billauer.co.il/peakdet.html> (visited on 10/19/2012).
- [4] Thomas A. Edison. “Phonograph or Speaking Machine”. 200,521 (United States). Feb. 1878.
- [5] Carl Haber. *New Technologies for Preservation and Access to Recorded Sound History*. Tech. rep. Berkeley Lab, Mar. 2011.
- [6] Michael Heinzer. “Adaptive Parameter Determination for Record Processing”. Bachelor’s thesis. College of Engineering and Architecture of Fribourg, Aug. 2012.
- [7] Lionel Seydoux. *Recovery of fragmented records using VisualAudio*. Technical report. College of Engineering and Architecture of Fribourg, Mar. 2008.
- [8] Stilsa. *Non-contact “line” sensors*. URL: http://www.stilsa.com/catalog2/pdf/STILSA_MPLS.pdf (visited on 10/16/2012).
- [9] Wikipedia. *Franz Boas*. URL: http://en.wikipedia.org/wiki/Franz_Boas (visited on 11/29/2012).
- [10] Wikipedia. *Gramophone record*. URL: http://en.wikipedia.org/wiki/Gramophone_record (visited on 10/09/2012).
- [11] Wikipedia. *Mathematical morphology*. URL: http://en.wikipedia.org/wiki/Mathematical_morphology (visited on 12/11/2012).
- [12] Wikipedia. *Phonograph*. URL: <http://en.wikipedia.org/wiki/Phonograph> (visited on 10/09/2012).
- [13] Wikipedia. *Phonograph cylinder*. URL: http://en.wikipedia.org/wiki/Phonograph_cylinder (visited on 10/09/2012).
- [14] Wikipedia. *The Dickson Experimental Sound Film*. URL: http://en.wikipedia.org/wiki/The_Dickson_Experimental_Sound_Film (visited on 11/29/2012).
- [15] Wikipedia. *Volta Laboratory and Bureau*. URL: http://en.wikipedia.org/wiki/Volta_Laboratory_and_Bureau (visited on 11/29/2012).

Appendix A

External files structure

This section presents an overview of the folder structure containing the files external to this report. These files are located on the project's website ¹ and, for printed versions, on the attached CD-ROM. The general structure is represented as follows (a `readme` file is also present in the root folder).

/documents Contains used documentation in electronic form.

/presentations Contains the slides of the presentation done during this project (the last version contains the slides of the thesis defense).

/repo Contains the last version of the `Git` repository used throughout the project. A commit log has been added in the file `git-commits.log`.

/report Contains this thesis report as well as \LaTeX sources and images.

/requirements Contains the project's requirements and the initial planning in MS Project 2010 format.

/tests Contains the results of the tests performed on the example records, as well as the required configuration to reproduce them with the original application.

¹ Accessible using an authorized account with the following URL: <https://forge.tic.eia-fr.ch/projects/cracked-records-3d-irene>.

Appendix B

Interactive Tracking User Manual

1 Introduction

This document aims to explain the new *Interactive Tracking* feature developed during this thesis from the end-user's point of view. It will detail the steps required to use all new functions and to be able to recover a record using the new features.

As a prerequisite, the user must already be familiar with the existing PRISM and/or RENE programs to restore recordings acquired with IRENE. This document is focused on the version as integrated in PRISM. Almost all explanations in this document also apply to the RENE version. The differences are explained in Section 7.

2 General description

2.1 User interface overview

In the application user interface, the main difference is the addition of a new panel on the right side, where the detailed record view also stands (see Figure B.1). It is accessible by clicking on the *Interactive Tracking* tab. The detailed view as already existing remains unchanged, and one can switch between the two views at any time. A new tracking option has also been added in the list of tracking methods.

Almost all the new features are located on this panel. The upper part contains the controls and some information about the record being tracked. The remaining space is used to draw the record while tracking. When no record is loaded, this part is blank except for a red arrow.

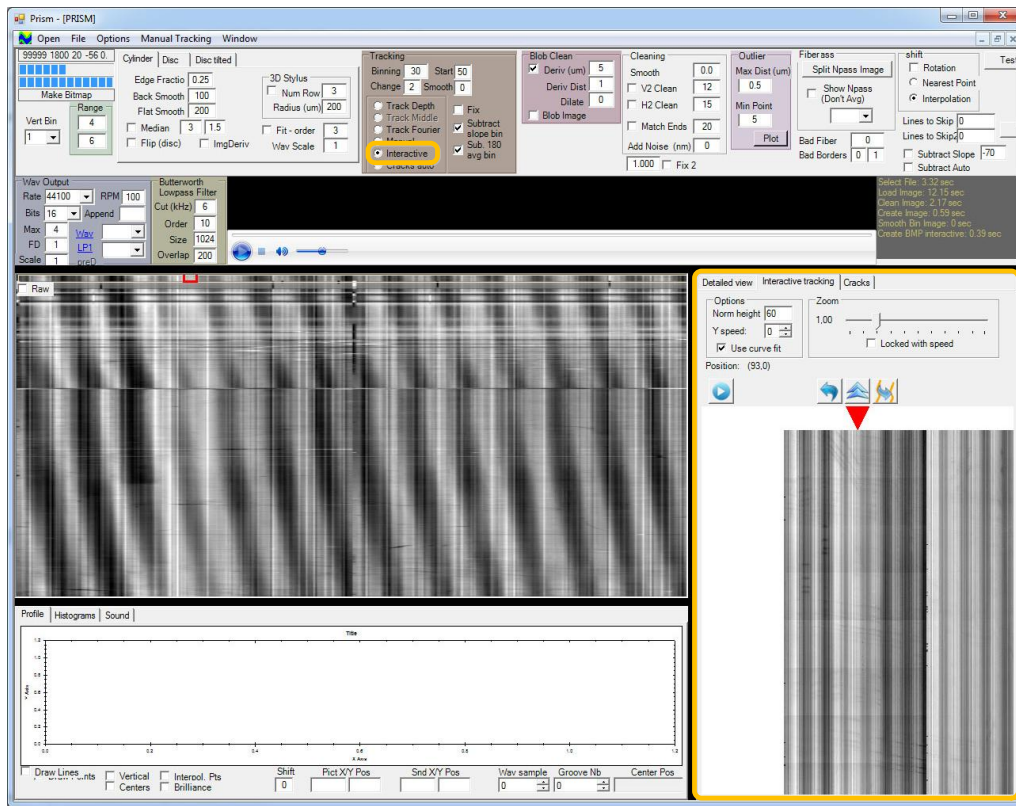


Figure B.1 Location of the interactive panel as well as the new tracking option highlighted in orange, as viewed in PRISM.

A description of this panel can be seen in Figure B.2. Specific controls are also detailed in Table B.1.

2.2 Description of controls

The upper part enables to control different parameters for the tracking. The first one, *Norm. height* may remain unchanged most of time. Its behavior is detailed in Subsection 3.1. The *Y speed* is for setting the speed while tracking a record. One can directly set the value or increment it. While tracking the record, it is also possible to increment and decrement the speed using the mouse wheel, without stopping the movement.

The option *Use curve fit* enables to improve the tracking by adapting the center of the groove using parabola fitting. If unchecked, the smallest value in the neighborhood is defined as the real center.

To change the zoom level, whether to see more details or an overview of the scan, one can slide the *Zoom* control. If the check box *Locked with speed* is enabled, the zoom

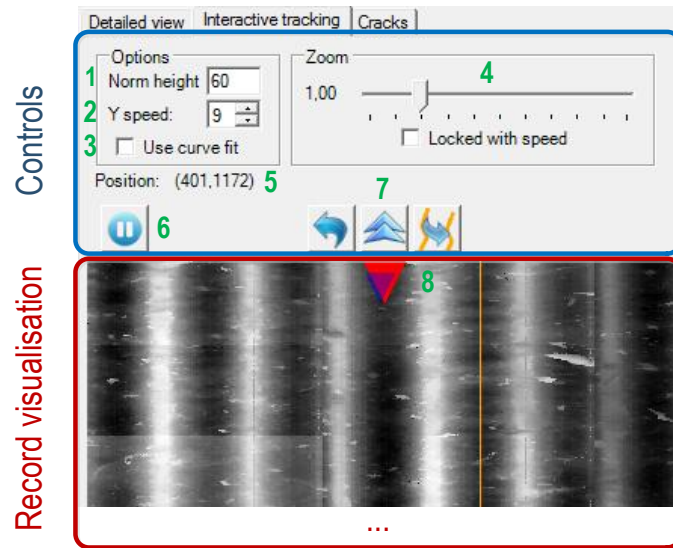


Figure B.2 Visualization of the controls in the interactive panel.

Table B.1 Explanation of the different objects in the interactive panel.

No.	Description
1	Text box to change the height for bitmap contrast normalization (see Subsection 3.1)
2	Tracking speed selector
3	Use curve fitting to make tracked position more smooth accurate
4	Zoom control
5	Indication of the current position pointed by the red arrow
6	Start/Stop tracking button
7	Tracking controls (see below)
8	Arrows indicating the currently tracked position

is automatically adjusted according to the tracking speed. This may be useful when a specific part needs a more accurate tracking. Slowing down will then directly show more details by expanding the zoom.

By default, the tracking is not active. To actually follow the groove, the *Start/Stop* button must firstly be pressed. The tracking can be stopped at any time, e.g. to look over other parts and continue later.

3 Loading a new acquisition

Before loading the acquisition, the first step is to select the proper option, *Interactive Tracking*, in the list of tracking types. Then, it remains to select the needed usual options for loading and processing and open the acquisition file into PRISM.

Once the record has been loaded, if the interactive tracking was selected, the heightmap image is directly viewed on the interactive panel, with the arrow pointing at the top-left corner.

3.1 Contrast normalization

The only option that must be set before loading the disc is *Norm. height*. In some cases, the absolute surface height varies a lot on big areas, making the structure of grooves negligible and almost invisible if the contrast was normalized from the entire range of values (for more information, see Section 4.2.3). The original detailed view in PRISM is not directly concerned by this issue, as the contrast is readjusted each time the user selects a new area.

However, recomputing it would be too slow to draw the record interactively. Therefore, the normalization is performed in smaller area. The width corresponds to the width of a pass and the height may be controlled with this option. However, a value of 60 should be adequate in most situations.

4 Tracking interactively

This section details all options available to the user to help manually tracking a record as quickly and correctly as possible.

4.1 Piloting the tracking

In the interactive panel, the drawn record can be seen as a draggable object. When the mouse is pressed, it is possible to shift it horizontally by moving the cursor. The red arrow represents the currently tracked position and remains fixed at the top-center. It should therefore point at the center of the groove. An example of starting position can be seen in Figure B.3.

Once the groove has been selected, the tracking may start. To actually keep track of the position while moving, the *Start/Stop* button must be pressed. Then, to start tracking, the most convenient way is to click on the record image and add speed using the mouse wheel. The image then starts scrolling down, representing the record being tracked.

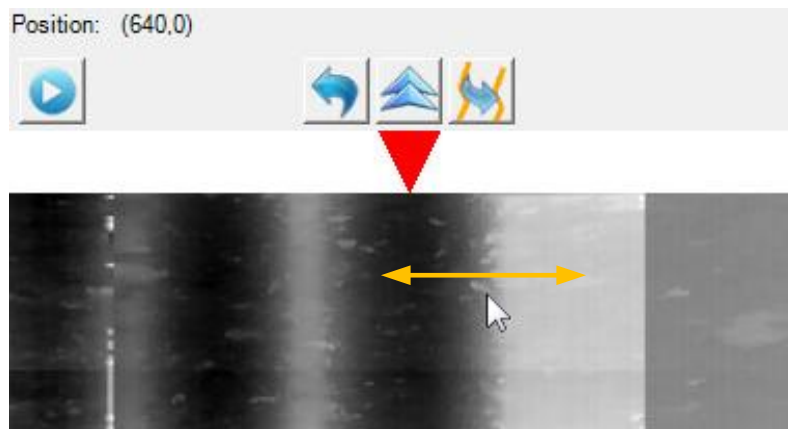


Figure B.3 Example of starting position. The record is dragged to the wanted position. The vertex of the arrow represents the position to be tracked. Here the tracking is not active, as the Start/Stop button has not been pressed yet. In this specific case, the starting groove is located on the right (tracking from right to left).

The main goal is to keep the top of the arrow as close as possible to the groove center, ensuring a proper tracking. The speed can be adjusted at any time with the mouse wheel. The user can also release the click, change the speed on the control and click again on the record to continue tracking.

4.2 On the binned image

At any time, the current position can be viewed globally on the binned image as a little red square. This is very useful to locate the current position on the whole record. Another important feature is that it is also possible to click on the binned image to move to that position. If the tracking is enabled at this moment, this will also add a new tracking point. In fact, it will act the very same way as the former manual tracking. This is very useful when a specific area does not require a very precise analysis, enabling to track it very quickly.

The two views are always synchronized, each of them giving information at another level. The current path representing the part already tracked is also represented in orange. Figure B.4 represents an example of the part of a groove being tracked and visualized on both views.

A magnifier tool

When the cursor is moving over the binned image (without clicking on it) the portion viewed in the interactive panel is temporarily moved accordingly. This then acts as an

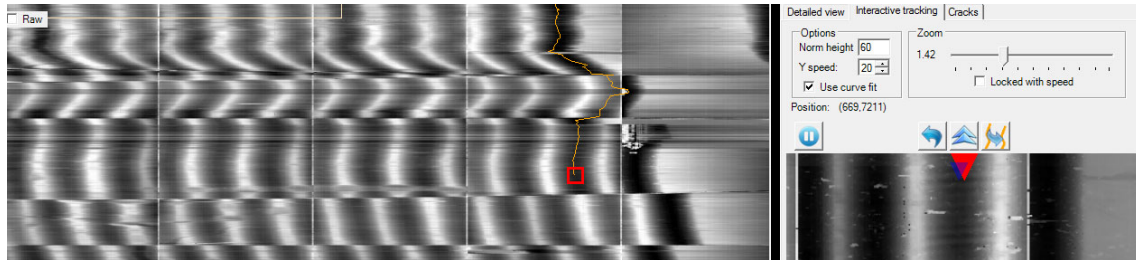


Figure B.4 *A groove being tracked. The two positions are synchronized and the path is drawn in orange.*

interactive magnifier (adjustable with the zoom), which may be useful on a former analysis step or while tracking, e.g. to find the best matching when a crack appears.

4.3 Starting a new revolution

When an entire revolution has been performed, the final position is at the bottom of the mapped image. If the tracking is correct, the tracking may continue directly at the same horizontal position. The system is able to move back by setting a negative speed. However, to simplify this particular case (setting the position back to the top), a button has been added (upward pointing arrow). In reality, it does three things:

- Go back to the top (without changing horizontal position)
- If the tracking is active, add a point at this position (representing the starting point of the new revolution)
- Sets the speed to zero.

Resetting the speed is useful so that the position can be adjusted before continuing. The tracking can then start easily from this point. One can note that the tracking is drawn not only in the binned image, but also on the interactive panel, which gives a more detailed view of the result. It is also useful to compare the current track to the previous revolution. On the binned view, the link from the end to the start of the next revolution (conceptually the same position) is drawn in red, as viewed in Figure B.5.

4.4 Letting the program do it

Manual or interactive tracking are primarily useful with heavily damaged records that cannot be tracked with an automatic algorithm. However, though it is not directly possible to track e.g. over a crack, it could be possible to let the program find the way when on an area where no irregularity occurs.

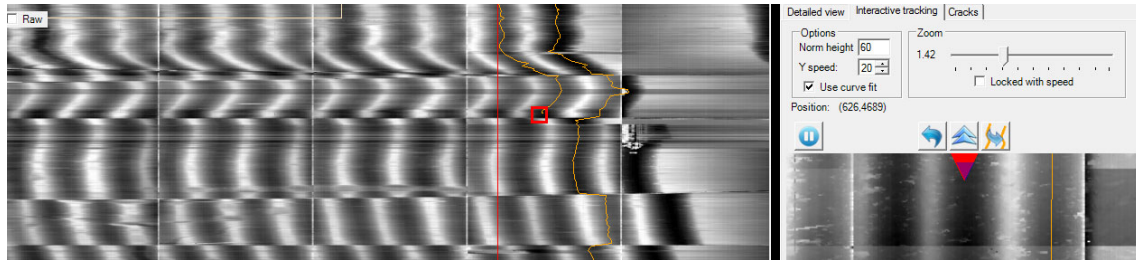


Figure B.5 Example when another revolution as started.

This option is able to automatically track a section from the current position to a specified one, acting as an *assisted* tracking. The user must simply press the *Shift* key while clicking on the record. This way, it is not a straight line that is drawn but the real center is followed until the clicked position.

4.5 Correcting an error

Manually tracking a record is an error-prone process. It frequently happens that a particular point is wrong or that a section of the track is wrong. An option to undo the last operation (leftwards arrow) has been added for this purpose. It reverses the new performed action and sets the position back to the previous one.

Usually, it directly removes the last tracked point. However, when the previous action added several points in one step, like with the assisted tracking or the copy pattern function (see Subsection 4.4 and Subsection 4.6), it will directly remove the entire action, which is more convenient e.g. if the track was totally wrong in a too large section.

4.6 Copying an existing pattern

Another useful feature is the copy of a pattern that remains common between several revolution. It often happens that once the path has been found for a revolution (from top to bottom), the same pattern is repeated almost exactly in the same area.

To avoid doing the same job multiple times, the option (available through the last button on the GUI) automatically copies the pattern from the last revolution, starting at the current position and shifting it accordingly. An example of copied pattern is visualized in Figure B.6.

For this feature to work properly, it is worth noting that the new revolution must have been previously started (i.e. the position has been put back to the top of the mapped image). The usual sequence is then (Go to top, Copy, Go to top, copy, ...). The horizontal shift may also be readjusted in between to precisely match the groove center. It may also be combined with the *Undo* option.

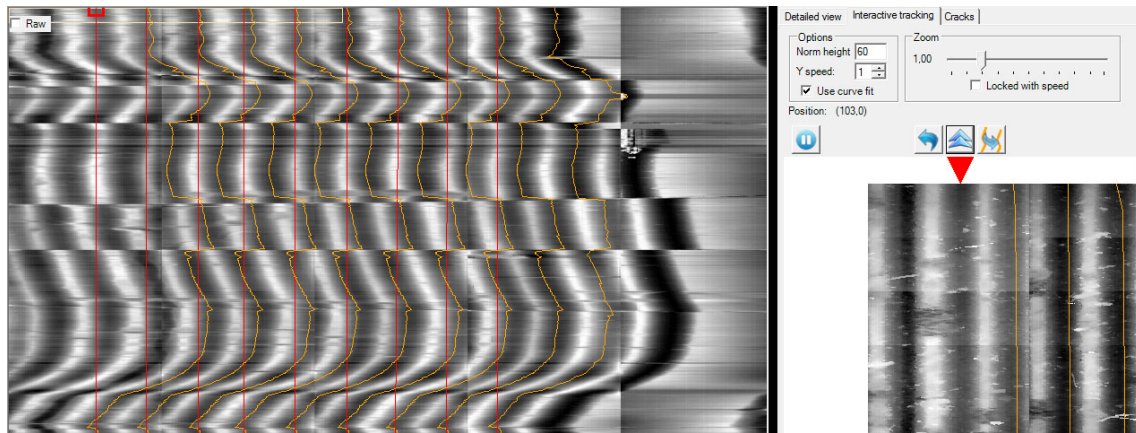


Figure B.6 Example of the same pattern copied for each revolutions.

5 Starting record processing

Once the tracking is completed, the remaining step is to launch the processing to finally get the audio file. In fact, this is done the same way as with the existing manual tracking. The *Apply* action in the *Manual Tracking* top menu will interpolate the track and launch the corresponding processing. The options can be set as usually regarding the record type (not detailed in this documentation).

6 Other considerations

6.1 Saving and loading tracking

This new way of tracking has been integrated in a way to minimize changes in the existing application. For the program, there is no difference between a record tracked with the old manual feature or with the new one. Thus, all existing options may also be used, as it is the case e.g. to apply the tracking and launch the processing.

Another useful feature already existing is the possibility to save the tracking to the hard drive, and load it back later on. The *Load* and *Save* options may be use as-is with the new implementation. The file is automatically saved in the record directory with a name postfixed by the starting range position. This enables to track several main parts of the record starting at different positions and at different times.

6.2 Tracking direction

Due to its way of operating, the interactive tracking can track only from *top to bottom*, i.e. in one rotational direction. For records that have been recorded in the opposite direction, the acquired record must be flipped vertically at loading time, so that the sound will not be played backwards. This option already exists in both PRISM and RENE.

7 Interactive tracking in RENE

This section explains the difference in the implementation specifically for RENE. The first important thing is that the new tab is this time located on the left side of the user interface, as seen in Figure B.7. The new tracking option has been added directly after the existing ones, in the *Control* tab.

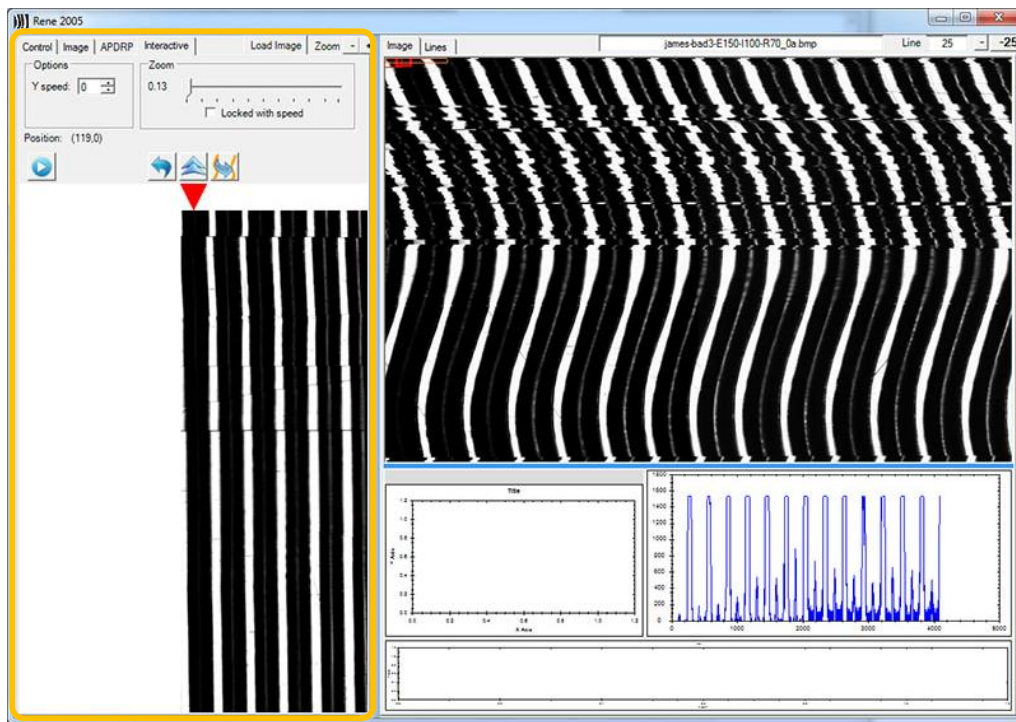


Figure B.7 Location of the interactive panel highlighted in orange, as viewed in RENE.

The other main change is the disappearance of the fit curve option. Indeed, with 2D acquisitions, this option is irrelevant. The tracking simply follows the brightest part representing the center of the groove.

Appendix C

Automatic Processing User Manual

1 Introduction

This document is intended to explain the use of the new automatic tracking feature for cracked or damaged records, implemented in PRISM. It explains all options and how to use them when it is possible to automatically track a cracked record.

As a prerequisite, the user must already be familiar with the existing features of PRISM to restore recordings acquired with the 3D Probe.

2 Overview of the system

The automatic tracking for cracked records has been created to offer an alternative to a manual system in certain circumstances (see Subsection 2.1). To enable this, the procedure is decomposed in several main steps:

1. Find cracks on the record surface and deduce the main parts separated by cracks, named chunks. There should be no other crack inside a chunk.
2. Proceed with the groove tracking within all chunks.
3. Link the tracked grooves from all chunks and build the final global track, before launching the processing.

For the first step, the configuration impacts the result and may be different regarding the medium type. This particular step is explained in Section 3. Also, the linking step often requires the user to check that the final matching results to a proper solution (see Subsection 6.2).

2.1 Typical use cases

This system works quite well under certain conditions but may fail in the general case. The sought properties for a better result are the followings:

- The cracks are clearly visible and identifiable (not too thin). This is the case if a significant different of brightness occurs between the regular surface and the crack.
- The grooves are also identifiable and may be tracked using the depth information. If big scratches occur too often inside a chunk, the tracking may not work properly.
- For a correct linking, the audio signal should be clearly identifiable, without too many silent part. This point is however not critical as it is possible to manually modify the linking.

The cracks detection is based on the properties of the brightness image. The processing then requires that the *.bri* file exists.

2.2 Configuration panel overview

A new panel, accessible through the tab *Cracks* has been added next to the detailed view and the interactive panel. It presents all options related to this new tracking method. The main parts of the user interface are visible in Figure C.1 and detailed in Table C.1.

Table C.1 *Explanation of the main parts in the Cracks panel.*

No.	Description
1	Options related to the cracks detection
2	Enables to show the specific results about cracks detections (if the delimited frontiers are correct)
3	Options related to groove and frontier detection
4	Tracking options
5	Options related to groove linking
6	Drawing options
7	Enables to restart the process using the new parameters

Configuration saving

All these configurable options are saved in a way similar to the general PRISM options, in an *INI* file. The main difference is that this new file, called *CracksConfig.ini*, is stored locally in the record folder, and not directly the application one.

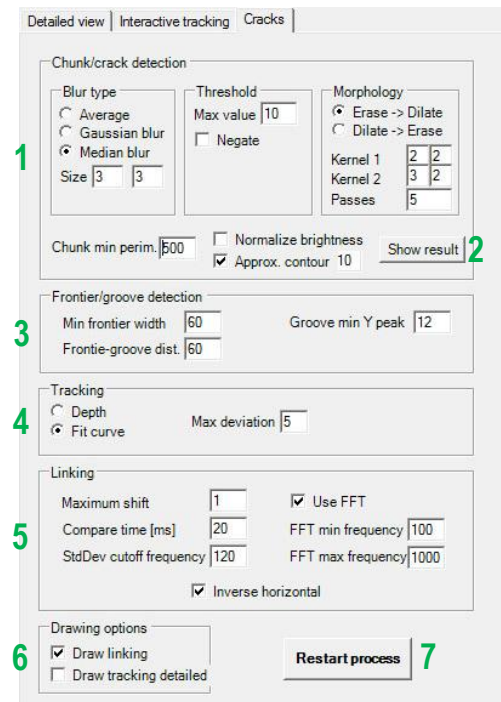


Figure C.1 Visualization of the controls in the Cracks panel.

Indeed, most of the configuration is specific to the current record and it enables to have different saved configurations for each record. It is worth noting that the configuration is loaded at the same time as the record acquisition, and saved back when the application exits. As a consequence, loading a new record overwrites the changes in the configuration. However, the modifications are updated to the file when the processing is restarted (using the button *Restart processing*) or when the program exits.

3 Starting the cracks detection

First of all, to use this tracking method, one must select the *Cracks auto* tracking option in PRISM which has been added after the existing ones. Then, when loading the record, the program will try to find the cracks, track the record and stop before the final processing.

Furthermore, to improve groove detection and tracking, some cleaning functions may be applied by selecting the new options *Subtract slope* and *Average 180* next to the tracking options. These allow to improve the created binned image by removing unintended height differences due to the multiple scans and therefore emphasize the actual groove shape once the signal is normalized (see Subsection 5.4.1).

At this point, a view similar to Figure C.2 is presented.

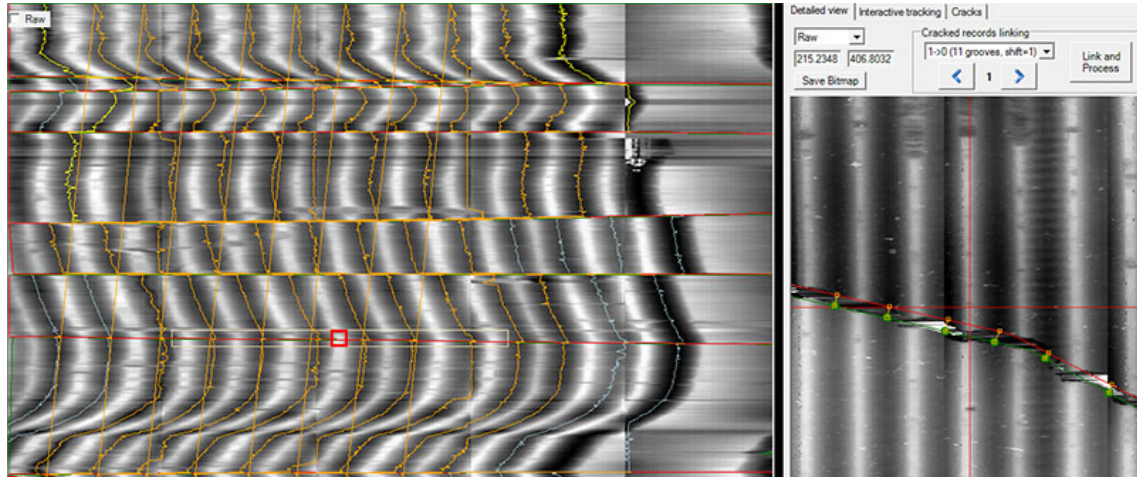


Figure C.2 Example after loading a record using automatic tracking for cracked records.

The chunk/crack frontiers are drawn on the binned view as well as the detailed view. On frontiers, small dots represent the detected grooves, as their starting position. The tracking is also visualized on the binned image.

4 Setting up the cracks detection

At this point, a complete analysis has already been performed. However, the result may not be correct and satisfactory in the first place. Firstly, we must focus on the cracks detection. All options are located in the cracks panel (Figure C.1, no. 1).

Blur type Should normally not be changed. The median blur is convenient for most cases. If the acquired brightness is very noisy, the size of the blurring kernel may be increased (e.g. to 5×5).

Threshold This value represents the pixel intensity in the range $[0, 255]$ until which it is considered as a crack. This definition is convenient for wax discs or cylinders, where the cracks appear darker than the regular surface. For some recording types, when the cracks appear brighter than the regular surface, the *Negate* option can be used to inverse the result of the thresholding.

Morphology These options enable to refine the parameters for the morphological operations. For more information, see Section 5.2.4 in the main report. For example, if the cracks are very thin, the dilation kernel may be enlarged, though this will also enlarge the possible noise.

Chunk min perimeter This value enables to exclude resulting chunks that are too small. The unit is in pixel.

Normalize brightness Use this option to apply the brightness normalization beforehand (as already implemented in PRISM).

Approximate contours Enables to approximate the found contours, resulting in a smoother shape. The approximation accuracy may be changed. It represents the maximum distance between the original shape and its approximation.

As this step may require several tests on its own, the final result may be seen in a separate window by clicking on the *Show result* button. The corresponding view is presented in Figure C.3.

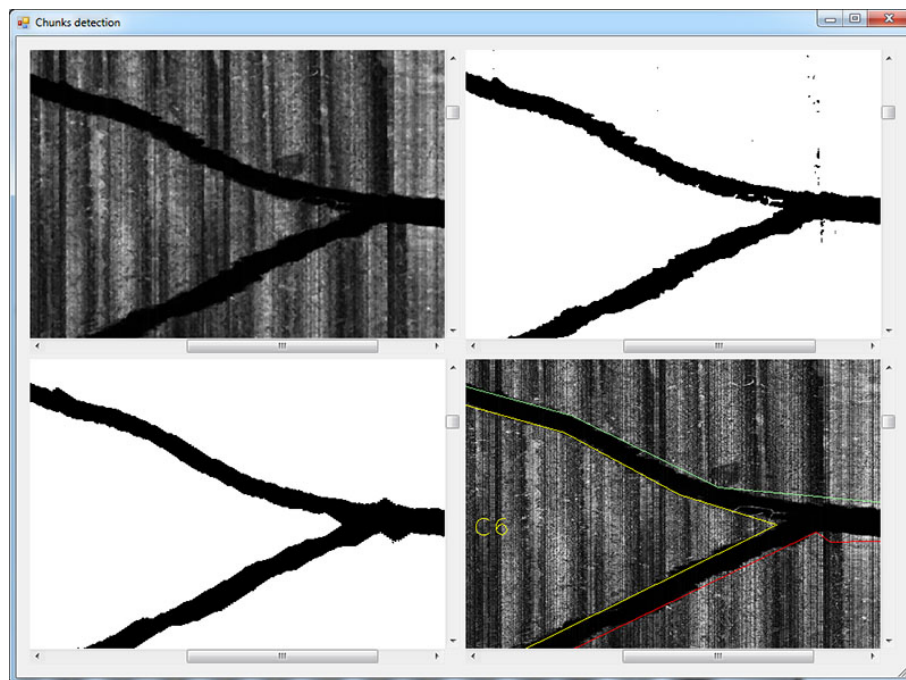


Figure C.3 *Chunk detection visualization. It shows the result after (from left to right, top to bottom) blurring, thresholding, morphology, and contours detection (drawn over the original brightness image).*

This view shows more clearly the result of the different steps applied by the detection algorithm. When the parameters have been adapted, one can apply them by restarting the whole process with the corresponding button and check the result again.

5 Groove tracking

Once all chunks are completely detected, the next step is to verify that all grooves are found and correctly tracked. On the binned image, all tracked grooves *inside* a chunk are

displayed in light blue. Furthermore, grooves that are already linked to a single track are displayed in yellow or orange (see Subsection 6.2).

Thus, if a groove has no line, it has not been found, or the tracking is wrong (may be mingled with another groove).

5.1 Check groove detection

To verify if the groove has been found, one can open the detailed view and check whether a green dot appears. An example is viewed in Figure C.4.

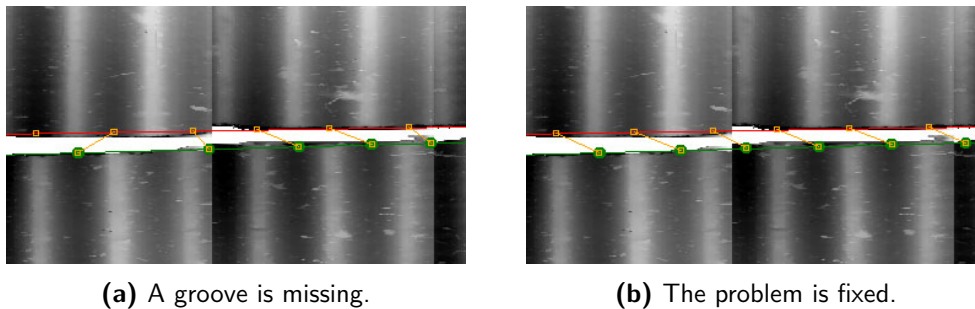


Figure C.4 *Example of missing groove as seen on the detailed image.*

In Figure C.4a, one groove is not properly detected. The problem is fixed in Figure C.4b. The useful options for customizing the groove detection are the following (Figure C.1, no. 3):

Minimum frontier width Specifies a minimum width for a frontier to be taken into account. A frontier is either a upper or lower boundary of a crack (for more information, see Section 5.3 in the main report). This value may impact grooves detected on small frontiers.

Frontier-groove distance Specifies the margin size between the detected frontier position and where in the chunk the groove is actually detected. If the surface is particularly noisy around the cracks, this value may be increased.

Groove minimum peak This value represents the minimum peak height so that it is considered as a groove border. The value corresponds in fact to a percentage from the height difference (which is normalized in the range $[0, 100]$ for the detection). If the grooves are heavily embossed, the value may be increased. If they are smaller, a lower value may help to find all grooves but may also find false-positives. For more information about peak detection, see Section 4.3 in the main report.

5.2 Improve tracking

Even if the grooves are correctly detected, the tracking sometimes needs to be fixed. Two algorithms are implemented for tracking grooves. The default one, using parabola fitting, usually gives better and smoother results. However, in some cases, the depth fitting may be better suited. The parameter *Max deviation* enables to set the “dispersion” of the algorithm, by fixing the maximal deviation from the current position while tracking.

6 Groove linking

Finally, the last step is to ensure that the final linking is suitable and results in a well-formed track. An algorithm tries to find the best link using sound analysis. However, tests have shown that it may be wrong. Therefore, after a first automatic step, one must usually manually review the linking.

6.1 Linking options

The linking options are used mainly for the automatic linking. The algorithm tries several possibilities and computes a score by comparing the linked sections. The linking with the best score is then selected as the default solution. The parameters are the following (Figure C.1, no. 5):

Maximum shift Represents the maximum shift value tried by the algorithm. For example, if set to 3, the algorithm will compare the current groove with the closest next one, and the three grooves on the left and right (seven tests in total). If it is evident that a record does not suffer a lot from shifting, a smaller value increases the likelihood of a correct linking.

Compare time Enables to set the signal length, in ms, that will be extracted for each sections to be compared to each other. The value in time is independent of the sample rate. Note: for a correct calculation, the RPM option in PRISM must be set correctly.

Use FFT If checked, the algorithm will compare the spectral domain of the grooves by applying a Fourier transform. Otherwise, the algorithm will compare the standard deviations of the signals, after having applied a high-pass filter.

StdDev cutoff For standard deviation comparison. Cutoff for the high-pass filter, enabling to get rid of the surface variability not due to the original audio signal.

FFT min/max frequency For FFT comparison. Minimum and maximum frequency that will be tested on the frequency domain (lower and higher frequencies are usually irrelevant).

Inverse horizontal This parameter is not related to audio comparison but to how the final tracks are created. Usually, the tracking is performed from left to right. For a tracking starting to the right side, this box must be checked.

Once the options are correctly set, a last pass may be applied by clicking on the button *Restart process*.

6.2 Linking review and processing

For the final part, one must go back to the *Detailed view* tab. When the tracking option is created, new controls appear above the record visualization. A list shows parts of chunks that have been linked. When an element of the list is selected, the view is automatically moved to see the corresponding area. The value shows the shift that has been automatically selected.

Then, the shift can be adjusted by pressing on the corresponding buttons. The value is incremented or decremented accordingly, and the result directly changes visually, as seen in Figure C.5.

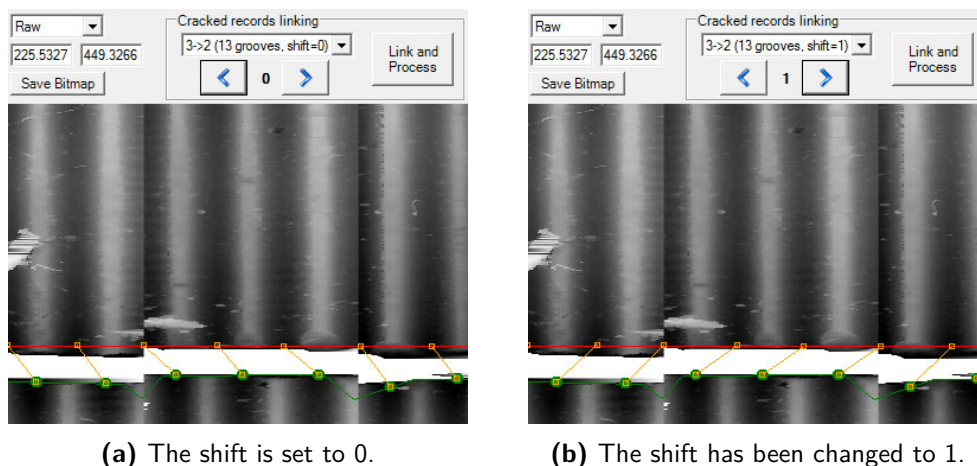


Figure C.5 Controls related to groove linking on the detailed view. The

With this view, it is easy to review the linking to get a final correct result. The path is also displayed on the binned image, enabling to see if the overall result is reasonable. If it is the case, the links from bottom to top are vertical, meaning that the revolution is correct in accordance with the record mapping.

Final tracks

The final tracks are drawn in yellow or orange. In fact, this simply enables to distinguish the different tracks. The final result may sometimes lead to multiple tracks. For example,

when it is impossible to link a groove in a specific area (record border or no groove to link), the program stops and tries to continue to link another track. An example is shown in Figure C.6.

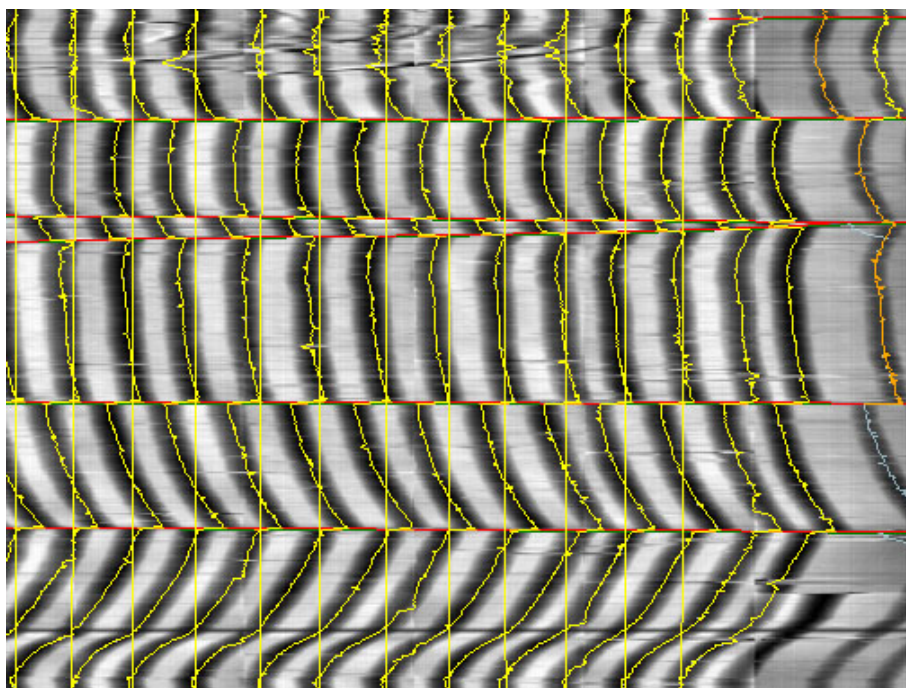


Figure C.6 *Multiple tracks as seen on the binned view.*

A first small track is drawn in yellow in the upper-left corner of the record. The tracking has then been stopped because the border of the record was reached. The second track (in orange) also stops at the record border. Finally, the rest of the record is properly tracked. One can note that the links are vertical, meaning that the revolutions (from bottom to top) are correctly linked.

Processing

Finally, the record may be processed using the existing algorithms by clicking on the button *Link and Process*. As a result, one audio file is created for each track. The name of the files is the usual names with information on the processing, postfixed with `-track-N`, with N being the track number.

7 Conclusion

Using the automatic tracking on a new type of record requires quite a lot of tuning. The results may or may not be adequate regarding the level of damage or the record quality.

This manual gave an exhaustive description of the parameters that may be changed on special situations. However, when applied on a set of similar record that corresponds to the typical use case (see Subsection 2.1), it may save a lot of time compared to a fully-manual tracking.

Sometimes, it is also possible that the properties change slightly from a section of a record to another. For best results, it is advisable to process it in several part, by using the ranges appropriately.

Appendix D

Requirements

1 Introduction

This document aims to define the specifications of the *Cracked records with 3D/IRENE* Master Thesis. It describes the main project objectives and specifies the needed tasks.

The important milestones during the semester are also defined.

2 Context

IRENE and 3D Probe are solutions maintained by Carl Haber and his team at the Lawrence Berkeley National Laboratory (LBNL). They consist of hardware and software which are able to recover the sound information from different types of old records without physical contact. The first system uses a common camera to scan the records while the second one uses a special 3D probe.

To get the sound information from the acquisition, the two programs called RENE and PRISM read the raw data and output the final processed sound by applying image and signal processing algorithms.

However, with some old material, several problems can happen. One of the most common one is cracked discs, that is, discs that are deteriorated because of the shrinkage of the lacquer layer. The signal is then cut by the cracks, which makes difficult to track the grooves. RENE and PRISM implement already a feature enabling to manually add some points to track the groove, but the solution is still not entirely satisfying because it is slow. It may also be difficult to find manually the correct match when shifts occur.

In a first step, the goal will be to improve the user interface to enhance the manual tracking feature. Then, the final goal will be, in some situations, to be able to automatically

track cracked discs or other damaged records by implementing new algorithms in the application.

3 Objectives

This section details the main objectives to achieve during the thesis. The secondary objectives are indicative and may be modified over time.

3.1 Main objectives

- Analyze the RENE and PRISM programs to be able to use them in common situation and program new features in the code base.
- Analyze the cracked records algorithm in VisualAudio to be able to extend it for the IRENE program. Find out the main differences in VisualAudio compared to the 3D/IRENE systems.
- Improve the current manual tracking feature to be more efficient and user-friendly.
- Add a new feature able to process some cracked records automatically or helping the user finding the best match by analyzing the signal.
- Be able to restore the sample recordings with the tools developed in this project.
- Be careful to maintain the application in stable condition and to avoid introducing regression.

3.2 Secondary objectives

- Improve the application code architecture for a good maintainability.
- Maintain/improve the code documentation.

4 Tasks

This section presents the tasks (from the objectives and the initial plan included in Section 6) that have to be planned.

4.1 Analysis

- RENE and PRISM familiarization and analysis.

- Learn the cracked records implementation in VisualAudio and make a presentation of it to the weekly student seminar at LBNL.
- Consideration about a more interactive user interface for the manual tracking.
- Consideration about the integration/adaptation of the solution implemented in VisualAudio.
- Consideration about the possible scaling corrections needed and the gaps caused by the cracks.

4.2 Design/Implementation

- Design and implementation of new GUI features improving the manual tracking.
- Design and implementation of an automatic reassembly system for cracked records (correct matching of the shifts).

4.3 Tests

- Functional tests over the test set.
- GUI tests.

4.4 Documentation

- Report writing
- User documentation
- Thesis defense preparation

5 Planning

5.1 Important dates

2012-09-17 Project start

2012-10-01 Weekly student seminar presentation

2013-02-08 Project end (report restitution)

2013-03-04 to 2013-03-15 Oral defense (in Switzerland, exact date not defined)

5.2 Gantt Chart

The tasks appear in the Gantt Chart in Figure D.1. The project main milestones are also specified. Nevertheless, this planning is more a general overview than a detailed schedule. Some objectives may be modified in the future regarding the future progress of the project.

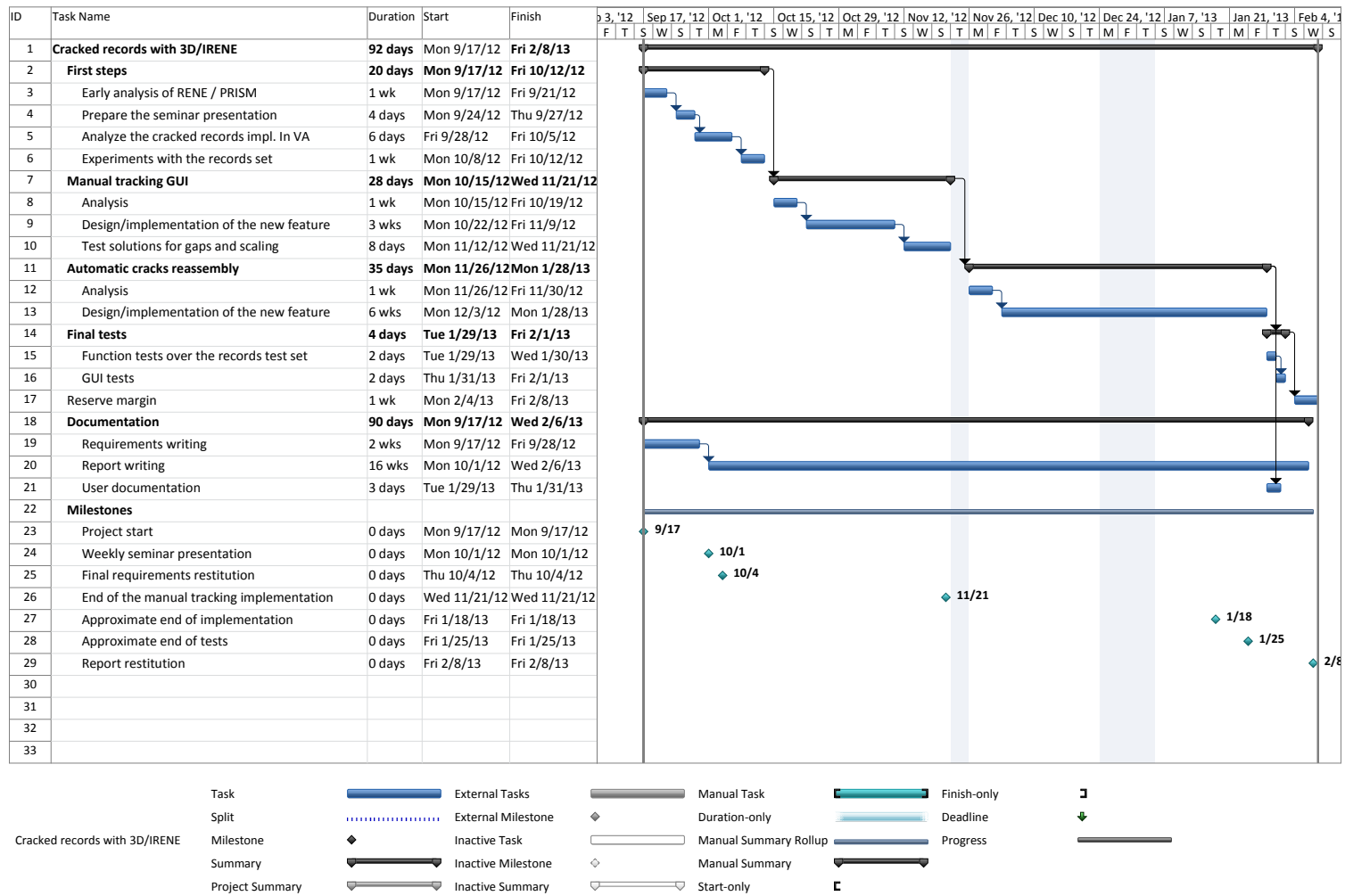


Figure D.1 Gantt Chart for the entire schedule of the thesis.

6 Initial specifications

At Lawrence Berkeley National Laboratory, Carl Haber, Earl Cornell, and collaborators have been working on the extraction of sound from phonographic records. They have developed both a 2D (image, referred to as IRENE) and a 3D (depth) system. The proposed project is part of a collaboration between this research team and EIA-FR, who have developed their own solution VisualAudio.

Old disc and cylinder records and other early experimental recordings can suffer from different kinds of degradations. Cracked transcription discs represent a frequent defect: the lacquer coating shrinks, which causes cracks where the plate support (metal or glass) is visible. Wax cylinders are brittle and on occasion break into two or more separate pieces. Early experimental recordings come in a variety of obsolete formats. These include foil sheets, wax discs on a number of different backing materials, metal plated depositions derived from wax masters, and glass discs with photosensitive coatings. Among this variety are many examples of breakage and deformation.

For the moment, the IRENE and 3D prototypes at LBNL are not able to process these kind of records automatically. The software does contain features which allow the user to manually guide the tracking algorithm across cracks, but this is an inconvenient tool to use in practice. None-the-less a number of broken items have been successfully scanned after temporary re-assembly.

The goal of this thesis project is to design, realize and evaluate the necessary modifications, so that the sound can be extracted from cracked recordings in a more robust and convenient way. One key aspect of this study will be to understand the balance between an automatic analysis, one which finds the cracks and derives the necessary corrections as part of the data processing, and one which includes user input and control. Particularly because the early experimental recordings are so diverse, and relatively rare and small in number, the latter approach may be in the end more powerful and easier to implement for the special cases.

The VisualAudio solution to the problem of cracked discs can bring some ideas, but it has itself some limitations, and the acquisition system is quite different.

The steps in this project will be as follows.

1. Following the initial discussions, and an examination of some of the example cases to be studied, Mr. Singy should make a presentation to the weekly student seminar about his experience with VisualAudio and the cracked disc restoration features.
2. A set of sample recordings will be chosen as the canonical test cases to use in this study. This set will include both common types and unusual experimental recordings.
3. Familiarization with the 3D/IRENE hardware and software.
4. A consideration of how the VisualAudio approach might be applied or generalized

to 3D/IRENE.

5. A consideration of how to apply a more user interactive approach to the restoration of broken recordings. This would include a considerable GUI feature whereby the user can guide the software through the tracking process in a fast and efficient manner.
6. A consideration of scaling corrections and how to best apply them. This is because once separated pieces actually shrink or expand with time and no longer actually fit together correctly.
7. A consideration of gaps - do these contain extra time which must be accounted for or deleted in the sampling and process? What are the best interpolation schemes to connect data across these gaps?
8. Even with manual intervention it can sometimes be ambiguous, to the eye of the user, how to match tracks across a crack, due to shifts. This step should investigate whether some sort of track counting (perhaps using Fourier analysis and phase measurement) could be used to guide better the track connection process.
9. These various studies and considerations will result finally in a software package and/or enhancements to the existing code packages (these are written in C# which combines the most appropriate features found into a set of tools, automatic and/or user interfaced, to restore a variety of broken media.
10. A best effort will be made to restore the canonical test set with the tools developed in this study.