Gamera: Optical music recognition in a new shell

Karl MacMillan, Michael Droettboom, and Ichiro Fujinaga

Peabody Conservatory of Music
Johns Hopkins University
1 East Mount Vernon Place, Baltimore MD 21202
email: {karlmac,mdboom,ich}@peabody.jhu.edu

ABSTRACT

An optical music recognition system has been completely overhauled and reformatted into a new framework called Gamera. The new open-source software is not only designed to recognize various music notations, including handwritten scores, but can be used to develop systems that can recognize many other structured documents. Gamera is intended to be used by domain experts with particular knowledge of the documents to be recognized but without strong programming skills. Gamera contains image processing and recognition tools in an easy-to-use, interactive, graphical scripting environment. Additionally, the system can be extended through a C++ and Python plugins.

Keywords: optical music recognition

1. INTRODUCTION

An optical music recognition (OMR) system (Fujinaga 1997) has been completely overhauled and reformatted into a new framework called Gamera. This system combines image processing and recognition tools in an easy to use, interactive, graphical scripting environment for the creation of domain-specific document recognition systems by document experts

The goal of the system is to leverage the user's knowledge of the target documents to create custom applications rather than attempting to meet the needs of diverse users with a monolithic application. The applications created by the user are suitable for use in a large-scale digitization project and they can be run in a batch processing mode and easily integrated into a large-scale digitization framework. Additionally, a module (plug-in) system allows experienced programmers to extend the system. This paper will give an overview of Gamera, describe the user environment, and briefly discuss the plug-in system.

2. MOTIVATION AND GOALS

Gamera is being created as part of the Lester S. Levy Sheet Music Project (Choudhury et al. 2001). The Levy collection represents one of the largest collections of sheet music available online <http://levysheetmusic.mse.jhu.edu>. The Collection, part of the Special Collections of the Milton S. Eisenhower Library at the Johns Hopkins University, comprises nearly 30,000 pieces of music which correspond to nearly 130,000 sheets of music and associated cover art.

The goal of the Levy Project (Phase Two) is to create an efficient workflow management system to reduce the cost and complexity of converting large, existing collections to digital form. From the beginning of the project, optical music recognition software was a key component of the workflow system. The creation of a flexible OMR tool is necessary because of the historical nature of the Levy collection. Existing OMR systems are not designed to handle the wide range of music notation found in the collection or deal with the potentially degraded documents. OMR alone is not sufficient for the complete recognition of the scores in the Levy collection as they are not comprised only of musical symbols. Text is also present as lyrics, score markings, and metadata. It was hoped, however, that an existing optical character recognition (OCR) system would be able to process such text. Early trials of existing systems revealed there are many problems with the current generation of OCR software within this context.

To address the need for OCR in the Levy project the Gamera system was created. Gamera is an extension of the existing OMR system to a general symbol recognition system. The recognition of text uses the same technology that allows the OMR system to perform well on the musical portions of the Levy collection. In addition to serving the needs of the Levy project and music recognition in general, we hope that the system may be used in the future for the recognition of historical documents and any other structured documents that current recognition systems do not adequately address.

In addition to generalizing the system, a graphical programming environment has been added to ease the adaptation of the system by users with expert knowledge of the documents to be recognized. This environment provides an easy-to-learn scripting language coupled with a graphical user interface. The goal is to allow the user to experiment easily with algorithms and recognition strategies during the creation of custom scripts for the recognition process. This will allow users to leverage their knowledge of the documents to customize the recognition process. It is hoped that users without extensive computer experience can effectively use this environment with a small amount of training. The scripting environment does contain, however, a complete, modern programming language that will allow advanced users considerable flexibility and power.

3. ARCHITECTURE OVERVIEW

Gamera is primarily a toolkit for the creation of applications by knowledgeable users. It is composed of modules (plug-ins), written in a combination of C++ and Python, that are combined in a very high-level scripting environment to form an application. The overall design is inspired by systems like MathWorks Matlab and spreadsheet macros. In Gamera, modules perform one of five document recognition tasks:

1. Pre-processing
2. Document segmentation and analysis
3. Symbol segmentation and classification
4. Syntactical or semantic analysis
5. Output

Each of these tasks can be arbitrarily complex, involve multiple strategies or modules, or be removed entirely depending on the specific recognition problem. Additionally, keeping with the toolbox philosophy of Gamera, the user of the system has access to a range of tools that fall within the general category of these tasks. The actual steps that make up the recognition process are completely controlled by the user.

In addition to flexibility, Gamera also has several other features that are important to the Levy project and to large-scale digitization projects in general, such as batch processing mode, open source for customizability, and portability (currently the system runs on Unix/Linux, Windows, and MacOS).

3.1 Pre-processing

Pre-processing can involve almost any standard image-processing operation including noise removal, blurring, de-skewing, contrast adjustment, sharpening, binarization, or morphology. Any number of operations may be necessary to take a raw input image and prepare it for recognition, but the output of this step must be a binary image for the rest of the recognition process.

Many documents, particularly historical documents like those in the Levy collection, will depend on this part of the recognition process to ensure overall good performance of the system. Discoloration of the documents makes binarization difficult and often requires locally-adaptive algorithms. Additionally, broken lines often cause problems in the segmentation of symbols.

3.2 Document segmentation and analysis

Before the symbols of a document can be classified, an analysis of the overall structure of the document is often necessary. The document segmentation and analysis process is designed to analyse the overall structure of the document, segment it into sections, and perhaps remove elements. For example, in the case of music recognition, it is necessary to identify and remove the staff lines in order to be able to properly separate the individual symbols. The proper identification of the staff lines and the grouping of the lines into staves and systems is essential to the classification of symbols and later to the interpretation of those symbols. Similarly, text documents may require the identification of columns, paragraphs, lines, or tables.

3.3 Symbol segmentation and classification

The segmentation and classification of symbols is the core of the Gamera system. The current implementation provides tools for the creation of simple heuristic classifiers, template-based image matching, and a learning classifier using the k-nearest neighbor algorithm enhanced with a genetic algorithm. Other possible classification algorithms include neural-nets, decision trees, or hidden Markov models. The use of both learning and heuristic classifiers allows for the balancing of flexibility, training time, and recognition speed.

3.4 Syntactical or Semantic analysis

This process reconstructs a document into a semantic representation of the individual symbols. Examples include combining stems, flags, and noteheads into musical notes, or grouping words and numbers into a table. Obviously, this process is entirely dependent on the type of document being processed and is a likely place for large customizations by knowledgeable users. Fortunately, generalized tools are provided by Gamera, including tools for structural and syntactic analysis, and theorem and constraint solvers.

3.5 Output

The output functions convert the raw symbols or the post-structural interpretation data into a suitable format for storage. In general, Gamera takes advantage of the rich set of XML tools provided by Python, though using XML for output is optional. Currently, for music, GUIDO and MIDI output are supported.

4. USER ENVIRONMENT

The goal of the user environment is to leverage the knowledge and skills of the user about the documents being recognized. This is accomplished by creating a dynamic scripting environment and graphical user interface where users can experiment with various Gamera modules.

4.1 Scripting Environment

Gamera includes a complete scripting environment that a user can use to create custom recognition systems quickly and easily. The scripting environment tries to be easy to use, flexible, and extensible.

4.1.1 Ease of Use

Perhaps the most important aspect of the Gamera scripting environment is ease of use by users with limited computer programming experience. As previously stated, the targeted user is a person with expert knowledge of the documents to be recognized that may or may not have computer programming experience. In order to meet this goal Python (Gauld 2000) was chosen as the foundation and extensions were written that are as easy to use as possible.

In order to transform Python from a general purpose scripting language to a scripting environment tailored to the needs of Gamera users, a set of extensions were written in a combination of Python and C++. Example 1 shows a script for OMR. This script gives a good indication of the high-level of Gamera scripts.

```
# load an image
image = Image()
image.load_image('example.tiff')
# Convert to binary using the Otsu thresholding
image.otsu_threshold()
# Remove staves and store information about
them
staves = image.remove_staves()
# Perform recognition on the image - this is a
# two step process. First the image is
# segmented and then the K-nn classifier is
# used on the individual symbols
symbols = image.connected_components()
classified_symbols = knn_classify(symbols,
'knn-database.knn')
# Interpret the symbols with the Optical Music
# Interpretation object
omi = OMI()
omi.interpret(image, staves,
classified_symbols)
# Output GUIDO and the MIDI
omi.save('example.gmn', 'guido')
```

4.1.2 Flexibility

Flexibility is the second most important goal for the scripting environment. Again, this aspect of the scripting environment is facilitated by the choice of Python. Because Python is a general-purpose programming language, a large portion of the system can be implemented directly in standard Python. In general, only those algorithms that need direct access to image pixels are written in C++. This allows users to customize existing modules written in Python, combine the low-level building blocks into new modules, or write modules from scratch.

4.1.3 Extensibility

Despite the flexibility of the scripting environment, not all algorithms can be suitably implemented in Python. For this reason, a C++ module system for use by experienced programmers has been developed. Some of the features of this system are:

1. Automatic binding of C++ code to Python.
2. Runtime addition of C++ modules as methods to Python classes.
3. Abstraction of the data storage format of image data using C++ templates to allow convenient access to compressed images.
4. Flexible programming interface allows the easy conversion of existing C and C++ code that uses a variety access methods to image data.

4.2 Graphical Interface

In addition to the scripting environment, Gamera includes a graphical user interface that allows the interactive display and manipulation of images using the scripting environment. This can be as simple as displaying the results of a pre-processing algorithm or as complex as complete interface for training. Figure 1 shows a sample Gamera editing session. Again, like the scripting environment, the graphical interface is created with standard tools entirely in Python allowing users to extend and modify the system.

5. CONCLUSION

A graphical programming environment for the creation of document recognition applications was described. This system, called Gamera, is designed to be used by people with expert knowledge of the documents to be processed. These users are not required to have extensive computer experience and the system can be effectively used with a small amount of training. Users with considerable programming experience can also create custom modules in

Python or C++ to extend the system. The applications created by this system are suitable for large-scale digitization projects because they can be run in batch mode and integrated into the digitization framework.

6. ACKNOWLEDGEMENTS

The second phase of the Levy Project is funded through the National Science Foundation's Digital Library Initiative 2 (Award #9817430), an Institute for Museum and Library Services National Leadership Grant, and support from the Levy family.

7. REFERENCES

Choudhury, G. S., T. DiLauro, M. Droettboom, I. Fujinaga, and K. MacMillan. 2001. Strike up the score: Deriving searchable and playable digital formats from sheet music. D-Lib Magazine 7 (2).

Fujinaga, I. 1997. Adaptive optical music recognition. Ph.D. Dissertation. McGill University.

Gauld, A. 2000. *Learn to program using Python*. Boston: Addison-Wesley.

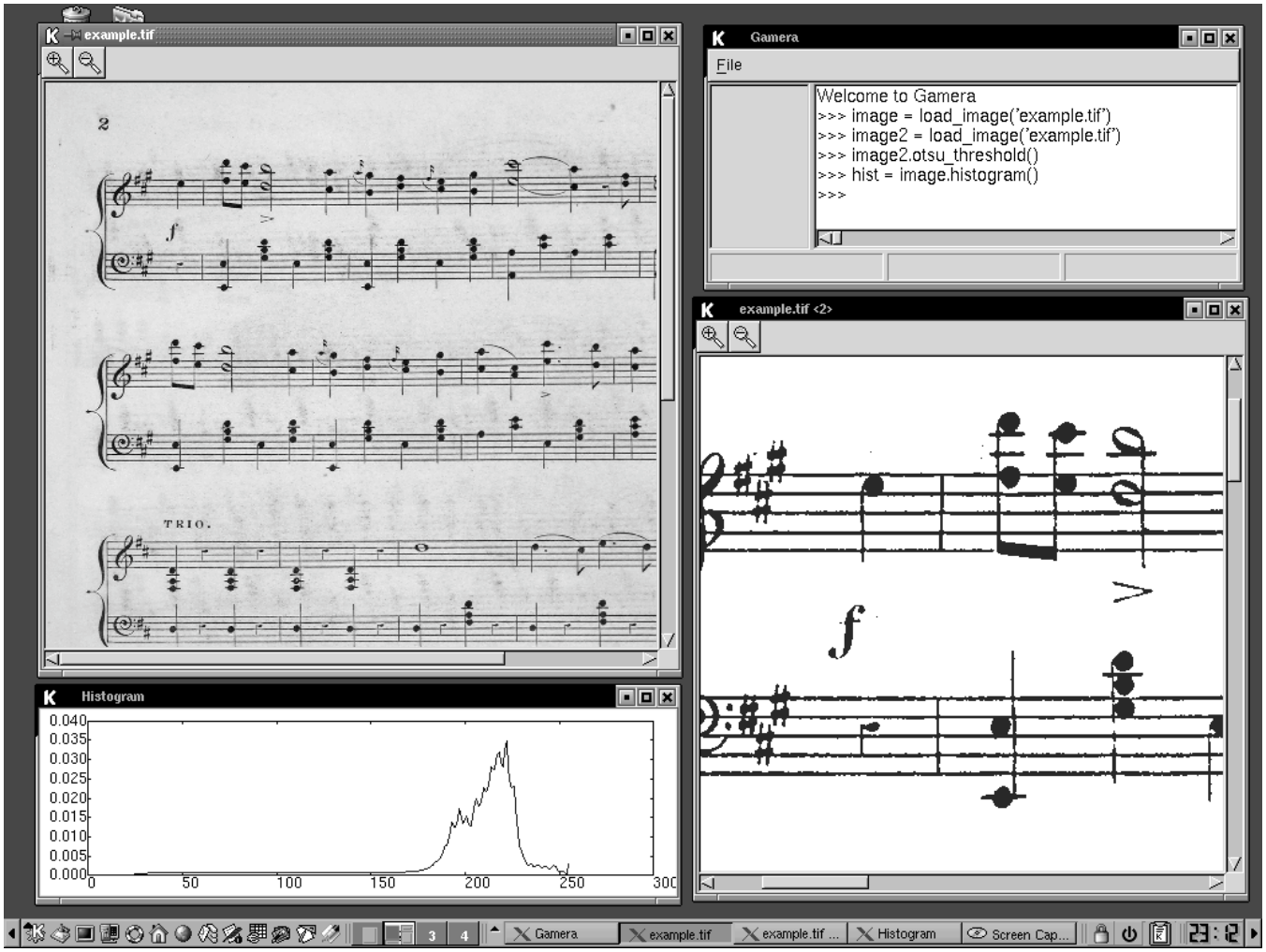

Figure 1. Screenshot of a grey-scale image, its histogram, the script window, and a copy of the image converted to binary format.