PyGraphics Final Report

Facilitating Multimedia Graphics Programming in the Introductory Computer Science Curriculum

Winson Chung, David Qixiang Chen

April 12, 2007

Abstract

Over the course of this term, we made significant progress in improving and implementing the various components of the PyGraphics multimedia library in preparation for its use in the upcoming introductory Computer Science courses. This report details issues with the Picture, Sound, Movie, and Turtle Graphics components, including the 33 bugs we closed. A final post-mortem evaluation of the project is included and provide suggestions as to how future teams may improve their experience in developing this library.

Table of Contents

1 Project Overview

The PyGraphics project aims to provide Media Computation technology for use in introductory computer science courses at the University of Toronto. This technology consists of a simplified multimedia framework originally written by Mark Guzdial [1] for use in his introductory Computational Media course CS1315/16 in the College of Computing at the Georgia Institute of Technology [2]. This multimedia library allows first year students to better learn basic computing concepts through the manipulation of images, sounds, and videos through the Python programming language. In addition, the library provides professors and teaching assistants with a means to create simple interactive demonstrations to augment the traditional means of teaching.

Currently, there are two versions of the library built on the Java/Jython and CPython based implementations. Our goal for the term was to take the developmentally-neglected CPython based implementation [3] and replicate all remaining functionality provided by the Java/Jython based implementation. This required removing all existing Java dependencies in the code, relying instead on other standard Python libraries such as *pygame* [6] (general multimedia), *PIL* [7] (imaging), *Numeric* (numerical) and *tkSnack* [8] (audio).

2 Completed Tasks

The multimedia library currently consists of four main modules, Pictures, Sounds, Movies and Turtle Graphics. In order to make greater progress in the translation process, we each took on different components throughout the term. The majority of completed tasks were those planned in the initial Analysis and Estimation report which did not include minor issues we encountered along the way. Overall, of the 40 tickets filed at the beginning of this term (for the multimedia library specifically), we completed 33 tickets, and rejected five as duplicates or irreproducible.

2.1 Pictures

~~#13~~. Problem saving image files

This was an issue with the version of the media library we initially received. Using PIL to save the images now results in no such error, and allowed us to save and load from a much larger set of image file formats including the originally supported BMP, JPG, and GIF formats. See Section 3.1 for limitations using the current saving system.

~~#14~~. Copy/duplicate portions of images

We implemented this ability for users to copy portions of existing pictures into new images. This includes the ability to composite various images together into a single image; this then facilitates the implementation of dropped images in the Turtle Graphics which has yet to be implemented (see Section 3.2 for limitations).

~~#16~~. Additional overlays

We implemented the ability to draw a polygon composed of any number of points onto an existing image. There are now two new methods in Picture for this purpose: `addPolygon` and `addPolygonFilled`. The reason for having two separate methods is to stay consistent with the existing overlay drawing methods' naming convention. There are also two global methods:

`addPolygon` and `addPolygonFilled` that call the respective polygon drawing methods in Picture.

~~#17~~. Text font styles

We added support for stylizing text in the library by using basic text rendering features found in pygame. Such support is limited to common operations such as changing the color, size, weight and emphasis of the text. See Section 3.4 for limitations with text rendering in OSX.

~~#58~~. Auto-repaint images

A new feature we added is the ability for the picture to automatically repaint itself whenever its image has been updated. This prevents confusion for the users who normally would not see any changes unless they manually call `repaint`. This features has also simplified the development of the Turtle Graphics component. By default auto-repaint is disabled to conform to how the book [4] currently presents various pieces of library functionality.

~~#51~~. Event Handling

We added support for event handling in the picture class which allows users to create fully interactive applications using this multimedia library. We have currently chosen to follow the Tkinter approach to binding events [9]. Sample code which uses this event handling model can be found in the demonstrations written for the presentations (see Section 2.7), as well as other online Tkinter resources (see Section 6). Please refer to Section 3.4 for justification for implementing event handling under the picture control as well as other event handling details.

2.2 Sounds

~~#50~~. Replace/merge the sound functionalities with tkSnack

The entire Sound class has been changed from using PyGame's sound functionalities to using tkSnack. This implementation solves a number of other problems such as loading of different sound formats and sound visualization. Please see Section 3.8 for details on tkSnack.

~~#52~~. Add the ability to plot the waveform of sounds

With the use of tkSnack, this feature was easily implemented. There are now three global functions: `plotWaveform`, `plotSpectrum`, and `plotSpectrogram`. They will display the corresponding visualization of the sound file that was loaded based on the dimensions defined. Please see Section 3.9 for details.

2.3 Movies

~~#24~~. Save/load movies to file

We re-implemented the ability for users to save and load movies to and from file in the form of image sequences. Different movies can be loaded from different files and played at varying speeds. Individual frames of loaded movies can be modified and saved without issue.

~~#23~~. Create movie sequence from pictures

Unlike movies in the Java/Jython implementation, movie frames don't necessary need to be loaded from file, movies can be constructed solely from the pictures in memory. This opens up a number of opportunities such as allowing interactive applications which rely on the Picture as a graphic context (as in the Turtle Graphics component, see Section 3.5) to save a history of operations which can be reviewed and replayed.

2.4 User Interface

~~#25~~. Fix stuck Tkinter file dialog problems

One major problem that we encountered (and inherited) was an issue with the Tkinter file dialogs in the multimedia library. This issue initially arose from the transition by the original group from wxWidgets to the standard Tkinter UI toolkit.

Through further investigation, this was found to be an issue specifically with the initialization of both Tkinter and the pygame display module (which is initialized automatically with pygame unless explicitly specified otherwise). We have taken a temporary solution to this problem by preventing both Tkinter and pygame.display from initializing when the library loads. This is a viable solution (at the moment) due to the fact that all frames and images are rendered to the user onto a Tkinter Canvas instead of through pygame.

~~#26~~. Warnings, errors, and other information

We added support for users to be presented with warnings and errors through standard Tkinter dialogs.

~~#49~~. Error messages to users should be handled correctly

When we inherited the original code, there were a number of instances where exception code was being called as such:

```
if not picture.__class__ == Picture:
        print "addRect(picture,x,y,w,h): Input is not a picture"
        raise ValueError
```

We believe this was an early attempt to shield new users from the slightly perplexing messages generated by the Python errors. However, this does not work as expected, and clearly misuses Python error handling. We replaced this with a different error handling scheme in which different output is presented depending on the current debug level (set to zero for ordinary users). More details on this implementation can be found in Section 3.6.

2.5 Turtle Graphics

~~#27~~. Implement turtle graphics

One of the features that we had to re-implement was support for Turtle Graphics. This component is based on an on-screen programmable cursor which takes the form of a Turtle image and allows for simple operations such as moving, turning, and drawing. This idea was popularized by the LOGO programming language, and gives users a seemingly simple means to produce a variety of images. Like a Turing machine, these simple operations can also be chained together to produce more complex works. The original implementation of this was done in Java, with this implementation done using pygame. This implementation uses the Picture in place of Java's Graphics Context (see Section 3.5).

Due to the size of the turtle code, it has been placed in a separate file and imported by the multimedia library at runtime.

2.6 Testing

~~#28~~. Update unit tests

One of the original project tasks was to evaluate the provided unit tests (based on the Java/Jython based implementation) and remove all Java dependencies. In reviewing the tests, we found them to be overly complex, and generally problematic. As such, the first week of the implementation phase of the project was dedicated to rewriting unit tests for the core components we received (Pictures and Sound). These tests were written for each method of each class and revealed a number of problematic issues with the inherited code. The tests are run with a debug level of two to ensure that Python errors are properly thrown and caught.

In addition to the functionality added near the end of term, which has yet to be tested (see Section 4.2), the percentage of the code covered by the tests currently stands at 55.4% for `media.py` and 33.6% for `turtle.py`.

In addition, much of the code was finally tested against examples in Mark Guzdial's book [4] to ensure compatibility with the existing library. No major errors were found in this phase of the tests.

3 Current Limitations and Implementation Details

As mentioned above, there are a number of limitations and notes that future developers should be aware of before starting development on this library. Some of these issues are inherited from the code we received, while others are due to new changes we've made. These limitations and details are based on the following versions and libraries:

- Python 2.4.4/2.5
- pygame 1.7.1/1.8
- PIL 1.1.6
- Numeric 24.2
- tkSnack 2.2.10

3.1 Saving and loading of images

By using the PIL library, we are able to save and load a wide variety of image formats. The problem is that the loaded image must be converted to a format accessibly by pygame (for rendering overlays, text, and compositing images, and storing pixel data). However, when drawing the image into the Picture frame, the pygame surface must be converted back to a PIL image (since it may have changed dynamically) before it can be shown on a Tkinter canvas. This issue can be investigated further in order to find a better solution.

3.2 Loading/copying an existing image

In this library, a picture is currently assumed to be a 24bit image (RGB only), which leads to limitations when loading formats with alpha transparency. In addition, when loading and copying from existing images, there is no way to define a transform for the loaded image (a rectangular area from the source image is mapped directly to a rectangular area in the destination image).

3.3 Event Handling

The event handling is currently bound to a specific picture instead of the picture frame. This is mainly due to the fact that the picture frame is constructed and destroyed at will, meaning that the frame can not be used to store persistent information. Instead we store event handling details with the Picture, and re-bind them whenever the associated frame is created.

As mentioned earlier, we are using the Tkinter event model, which takes an event string to bind, and a callback function which is invoked when the event is handled. The handler function takes the event object as a parameter which contains details regarding the circumstances in which the event was called. More details on this can be found in the Tkinter documentation [9].

In the following example, we are binding the left mouse button press to a callback which prints "hello world".

```
# callback function
def printHelloWorld(self, event):
        print 'hello world'

# bind the left button click command to the callback
picture.addEventHandler("<Button-1>", printHelloWorld)

# show the picture so that we can interact with the picture
picture.show()
```

Additional examples can be found in the demonstrations used for individual presentations this term under the `/trunk/cpython/Demos` directory in the repository.

3.4 Text rendering in OSX

Although we technically fixed the problem with rendering stylized text (Ticket #17), users will still not see this when using the library on OSX. This is not an issue with our implementation, as pygame currently has an issue where system fonts are not loaded correctly in OSX. We expect that this will be fixed by the next release of pygame (1.8), though this is a minor issue.

3.5 Using Pictures as a Graphics Context

When implementing Turtle Graphics, we had to find some way to represent the current Graphics context, similar to that used in the Java graphics model. However, unlike the Graphics context, the picture does not store the orientation nor the various properties of the context (such as font, pen, and rendering properties). As such, we have to defer operations such as affine transformation to a pre-rendering state done by the caller rather than the context itself.

For example, when the turtle is drawn in the world, it is first rendered onto a temporary sprite, which is transformed to the current orientation of the turtle before being copied (or *blitted*) to the image surface.

While cumbersome, this implementation appears to be satisfactory for the simple use in the contexts of this library.

3.6 Error Handling

In trying solving this problem, we initially tried replacing the error output with a wrapper which printed different values depending on the current debug level (set to zero for ordinary users). Unfortunately this yielded problems when trying to pass on original errors which were thrown (necessary for unit tests which make certain assertions). Finally, we decided simply to use an exception hook to capture all exceptions passing them on directly only if necessary. As it stands, there are currently three debug levels:

Debug Level	Output/Description
0	Plain error messages printed (no notion of Python errors presented)
1	Plain Python errors are passed on to the user
2	A detailed trace of the exception stack is presented

3.7 Indexing

It should be noted that many of the indices used in the library library and Mark Guzdial's book are **one-based** and not zero-based, as common used in other libraries. This has been the source of several bugs in the inherited code, and should be kept in mind while developing in the library (if only to maintain consistency between the various components).

3.8 Replacing/merging sound functionality with tkSnack

Integrating the tkSnack library solved a number of problems including those with file formats and sound visualization. The file formats supported by tkSnack are: `wav`, `mp3`, `au`, `snd`, `aiff`, `sd`, `smp`, `csl`/`nsp`, and `raw`. It also support the encoding formats: `Lin16`, `Lin8`, `Lin8offset`, `Lin24`, `Lin32`, `Float`, `Alaw`, and `Mulaw`. The new Sound class essentially acts as a bridge to connect tkSnack to the original Sound class interface. The Samples and Sample classes are left untouched.

3.9 Add the ability to plot he waveform of sounds

Originally we wrote the code for drawing the sound waveform manually with Picture class. It was later found that tkSnack was much more feature rich, and all sound visualization has now been deferred to using tkSnack. There are three possible visualizations: Waveforms, Spectrograms and Spectrums (tkSnack calls this Sections or Power Spectra). Each have their own configuration options including drawing size, colors and start/stop times. If needed, these options, especially start/stop times can be used to implement zooming of the sound graphs.

3.10 Playing predefined musical notes

Because examples of sound manipulation was given as exercises in the Media Python book, it was decided that notes generation can be considered as an exercise for the students as it's a subset of sound manipulation with the existing library. The code base however exists. It can be found in `demo_music.py` with a list of predefined note frequencies. It uses a very simple sine wave generation algorithm that should be easy to understand with basic knowledge of trigonometry. The actual tkSnack library for sound synthesis is much more extensive.

4 Remaining Tasks

In addition to potential areas for improvement noted in Section 3, there are several tickets which have yet to be completed. They are listed in descending order of importance, with filed tickets listed where available.

4.1 Implementing a platform-independent installer (#62)

One of the last major tasks which we were not able to complete was the creation of the distribution packages. As confirmed by Mark Guzdial, the installers we received were created by hand using various platform specific tools (such as Apple's Package Maker) to produce platform-dependent installers (mpkg or exe). This may be troublesome if the development of this library reaches a point where updates are consistently being released. As such, it is preferable to migrate the installer to something platform independent, and more importantly, which can be scripted and automated.

One possible solution would be to use Python's `distutils` module to install the various dependent packages (including the media library) into the user's system. The `distutils` module allows us to separate the two fundamental tasks of distribution; the actual installation of the libraries to locations accessible by Python, and the distribution of said installer and libraries in a single accessible package.

We solve the first task using `distutils` by creating a setup script (`setup.py`) which describes all dependent libraries (and versions) as well as where to install the necessary files, and how to update Python environment variables (such as `PYTHONPATH`) to locate the library. The second task is completed by making use of the various `distutils` packagers. A packager is a module which takes a setup script and creates an accessible installer for a specific platform (such as a mpkg installer). On first glance we can use the `bdist_wininst` packager to create a self extracting zip for Windows, `bdist_rpm` packager for Linux, and `bdist_mpkg` for OSX.

Once these two parts are complete then the task of releasing an updated package should be significantly easier.

4.2 Write unit tests for Movie and Turtle Graphics (#47)

While unit tests have been rewritten for the other basic components, the newly added support for Turtle Graphics and Movies has yet to be tested thoroughly. Therefore, it is necessary to improve and extend current unit tests to ensure regressions do not happen as we add new features in the future.

Note that the current debug level must be set (after loading the library) to level 1 to ensure that proper errors are passed onto the calling code, otherwise, only error messages are printed to the console. Also note that resources for the unit tests are placed under their respective subdirectories in the `/resources` folder, and can be accessed through the `resi(…)` and `ress(…)` calls.

4.3 Implementing the explorer tools (#50, #51)

While this is not a core component to the media library, each of the explorer tools (Picture, Sound, and Movie) serve to provide users with a simpler means to interact with the library (for example, not having to type a command to select an image file) and can be written quite easily using Tkinter, or even with the media library.

The Picture tool is the simplest to implement, as it only provides the ability to load an image and display pixel data as the user hovers over the image.

The Movie tool is also quite simple as it provides the ability to load, play, and step through individual frames of a movie. It also allows the users to set specific properties related to the movie.

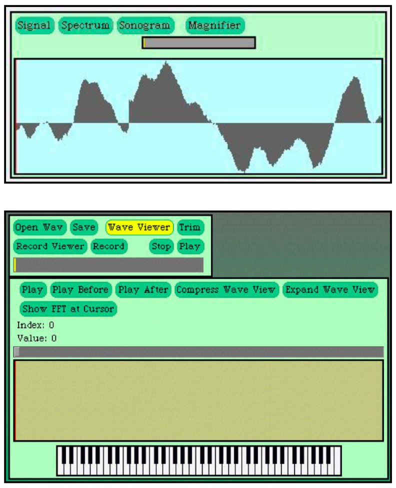

Figure 1. The sound tool in action

Potentially the most complex tool to implement is the Sound tool, which has several modes of operation. It allows users to load and save sound files, and exposes a number of other sound operations (such as the rendering of sound waves). Some of the functionality featured is not available in either version of the media library as it was written separate from the library using Java. The choice of whether to implement these features (such as sound recording) should be decided at a later time. More details on each of the explorers and their expected behaviors can be found in the book [4].

4.4 Dropped images in Turtle Graphics (#63)

The Turtle Graphics component was implemented using a model-display pattern in which the `World` extends the `ModelDisplay` class, and each world object (such as the `Turtle`) implements class `Model` class. One of the features that was not implemented with Turtle Graphics was the ability to "drop" images at the turtle's current location with the current orientation. This can easily be implemented by creating another `DroppedImage` class which extends the `Model` and is created with the current orientation. By adding this dropped image to the world, we can then render the dropped image. Note that this means that each model may have to store a priority which defines the order in which model objects are to be rendered (to prevent situations in which a dropped image is rendered over the turtle).

4.5 Play predefined musical notes (#20)

This feature is not currently implemented in the code, has we feel that it is better to be left as an exercise for the students. However the feature is already in tkSnack if there ever is a need for this to be made available in the library. If music notes generation is ever given priority in future development, this code can be used as a starting point. The ticket is currently left open for the next team (see Section 3.10 for details).

4.6 Testing with the Wing IDE

We did not have a chance to test this library with the Wing IDE [5] this term. However, the use of the library is quite standard and has been tested to work well with Eclipse and Pydev by simply ensuring the correct Python paths are set. We do not foresee any problems with the library's use with Wing, but this will have to be tested thoroughly.

4.7 Miscellaneous Tasks

Other minor issues with the library can be found on the DrProject bug tracker, listed in no specific priority with no particular milestone. These should be reviewed and ordered by the next team undertaking this project.

5 Post-Mortem Evaluation

PyGraphics began this term as a completely new CSC49X project, and understandably, there were issues which took some time to work out. Unlike other projects, there were several key differences:

- We were developing a library as opposed to an application (which made doing individual presentations more difficult).
- We had no concrete list of features to implement or bugs to fix (generally only to bring the library up to par).

That said, we each enjoyed working with Python and the multimedia library, and learned a good deal regarding project management and software development. In retrospect, there are a number of things which we could also have done better:

Communication

This involves communication both with the client, as well as with each other in the team. At the beginning of the term, we had a number of communication problems as to the scope and goals of the project, which led to a later start than other teams. The goals were very wide, and we were unsure as to the various technology involved, and which we were working on, which we were basing on, and which was just related but not used (such as JES). As such, we as a team, should have spent more time with Paul Gries in the initial stages of the project to iron these questions out. We could have made better use of the DrProject project management tools (especially the wiki) to define what we needed to do, as well as the current status.

Teamwork and Division of Labor

With Mikhail working on a separate piece of the project (VPython), we had divided the components of the media library to be completed. This provided an accelerated means to complete the necessary work, but when there were issues with implementation, we could not do much to help each other. This division of labor made it difficult to follow the priority schedule we defined in the A&E, as related bugs were completed together. While this turned out all right in this case, this may have been a serious problem had the scope of the project and goals had been different.

Lastly, one major problem was that we did not consistently hold internal meetings throughout the term. Issues were mainly discussed informally over IM chatting and in between assignments for other courses. We would have benefited from working together on the project for at least one hour a week. As noted early in the term, scheduling is perhaps one of the most difficult tasks of project management, and this term has been a wake-up call in this respect.

Conclusion

Over the course of this term, we felt that we made considerable progress in bringing the state of this library up to par. From fixing the numerous issues to implementing a number of new features, we enjoyed the experience of working with this library and improving it for use in the upcoming year. While none of the remaining issues are absolutely critical in nature, there is still much improvement to be made which we have left as suggestions for the next team to take on this project.

6 References

[1] **Mark Guzdial**
http://www.cc.gatech.edu/gvu/people/Faculty/Mark.Guzdial.html,
guzdial@cc.gatech.edu;

[2] **Media Comp at Georgia Institute of Technology**
http://coweb.cc.gatech.edu/mediaComp-plan

[3] **media.py (CPython) development page**
http://coweb.cc.gatech.edu/mediaComp-plan/117

[4] **Mark Guzdial, Introduction to Computing and Programming with Python. A Multimedia Approach**
Prentice Hall; ISBN 978-0131176553 (December 27, 2004)

[5] **Wing IDE**
http://wingide.com/wingide

[6] **PyGame**
http://www.pygame.org/

[7] **PIL**
http://www.pythonware.com/products/pil/

[8] **tkSnack**
http://www.speech.kth.se/snack/

[9] **Tkinter Event Model**
http://www.pythonware.com/library/tkinter/introduction/events-and-bindings.htm