CSC 299/499
Bluetooth Communications with C
Henry Walker

Image Processing in the revised MyroC 2.1

Vasilisa Bashlovkina, Anita DeWitt

GRINNELL COLLEGE

Introduction

Since 2011, a student-faculty team at Grinnell College Computer Science Department has been developing and refining a complete course package called MyroC, comprising a C-based infrastructure for working with Scribbler 2 robots and other course materials for CSC-161, the second course in the introductory CS sequence. During the recent efforts, the old framework that relied on the original C++ implementation has been almost completely replaced with a self-standing C-based infrastructure. However, converting the image processing functionality from C++ to C proved to be a non-trivial process because:

- Scribbler robots take pictures in **YUV** format
- Students usually think in **RGB** format
- Viewing software normally expects **jpeg** or **png** files

This poster describes the problems and the solutions that resulted in a redesign of the familiar picture functions in pure C.

Issues with Pictures in MyroC 1.0

The previous version of MyroC solved the problem of juggling multiple formats (YUV, RGB, png) with the help of the extensive C++ image processing libraries. While this approach certainly worked, it had a few flaws.

- **Portability**
  The dependence on a C++ infrastructure created a major issue of portability, which limited the use of MyroC to Linux environments. According to experimental data, porting the numerous auxiliary packages for the C++ materials from Linux to a Macintosh required about 4.5 hours.

- **Unnecessary level of abstraction in the Picture struct**
  The declaration of the Picture struct obscured its intuitive nature as a two-dimensional array of pixels and added an unnecessary level of abstraction. Consider the task of making a Picture pic 10% less red. Inside a nested for loop iterating over rows and columns of the image, one would have to perform three steps:

```
// Step 1: get the pixel
Pixel pix = rGetPicturePixel(pic,col,row);
// Step 2: change it
pix.R = pix.R * 0.9;
// Step 3: record the change
rSetPicturePixel(pic,col,row,pix);
```

Clearly, this setup convoluted the underlying nature of a Picture and obstructed the access the individual pixels.

A New Implemenation

The challenges of the image processing implementation in MyroC 1.0 inspired a redesign of the Picture struct and supporting functions in the revised MyroC 2.1. The new approach utilizes a 2D array of RGB pixels as its base and supplies functions to transform the basic image to and from jpeg format for storage, retrieval, and display. A well-defined structure specifies width, height, and a 2D array of Pixels representing a Picture. With this setup, students can manipulate image pixels directly, gaining experience with nested loops, 2D arrays, and structures. As a result, a student can now make an image 10% less red with a single line of code inside a nested loop:

```
pic.pix_array[col][row].R *= 0.9;
```

MyroC 2.1 relies on widely used software to handle image-specific tasks:

- ***libjpeg***, a common C library, converts 2D arrays of pixels to jpeg files and back
- ***ImageMagick***, an open source software suite, handles viewing functionality

The use of standard, open source software is essential when considering the possibility of porting MyroC onto a different operating system than Linux.

Outcome

As a result of this semester's work, students can use the current MyroC framework to import or create images, manipulate them directly in a 2D array framework, and store, retrieve, and display them in jpeg format. The updated Picture struct looks like this:

```
typedef struct
{
  int height;
  int width;
  Pixel pix_array[256][192];
} Picture;
```

With these changes, the hope is that students will be able to gain more experience with nested loops, 2D arrays, and structs. Moreover, thanks to the widely used open source software utilized in MyroC 2.1, the framework is now more easily portable to environments other than Linux, which opens up the nationally recognized CSC-161 curriculum to other colleges.

Acknowledgements

This on-going project continues to build upon efforts of successive development teams. We would like to extend our special thanks to past team members, whose work paved the path to our results: David Cowden, Spencer Liberto, April O'Neill, Erik Opavsky, Dilan Ustek, and Jordan Yuan. In addition, we would like to acknowledge our collaborators Nicolas Knoebber and Anqing Liu. Finally, we thank Professor Henry Walker for his guidance throughout the development process.

References

1. ACM/IEEE-CS Joint Task Force on Computing Curricula. *Computer Science Curricula 2013*, ACM Press and IEEE Computer Society Press. , December 2013.
2. Cowden, D., O'Neill, A, Opavsky, E., Ustek, D., Walker, H. M., A C-based introductory course using robots, *Proceedings of the 43rd ACM technical symposium on computer science education*, pp. 27-32, Raleigh, NC, February 29-March 3, 2012, Raleigh, NC
3. Ustek, D., Opavsky, E., Walker, H. M., and Cowden, D., Course development through student-faculty collaborations: a case study, *ITiCSE '14: Proceedings of the 2014 conference on innovation & technology in CS education*, Uppsala, Sweden, pp. 189-194, June 2014.