COMS W4115 : Programming Languages and Translators

TMIL : Text Manipulation Imaging Language

Project Proposal

Eli Hamburger (eh2315@columbia.edu)
Michele Merler (mmerler@cs.columbia.edu)
Jimmy Wei (jw2553@columbia.edu)
Lin Yang (ly2179@columbia.edu)

September 24, 2007

1 Introduction

TMIL (pronounced TEE-mil), short for Text Manipulation Imaging Language, is a revolutionary high level programming language that allows users to manipulate text programatically on an image. Users of the language can generate small programs that can do sophisticated text manipulations on images, without having to resort to complicated graphics libraries or painting programs such as Adobe Photoshop. Text manipulation has a wide range of applications, specifically geared towards web development. Some interesting scenarios include:

- Allowing a user of a Content Management System to display text to website users in fonts that users don't have on their computer
- Allowing a web site template designer to create template that are easily adaptable
- Allowing users to interactively label specific regions of an image.
- Generating CAPTCHAs, an image based challenge-response tests used on many web site registration forms.
- Manipulating text on multiple images to create an animation.

There are image processing libraries available that can manipulate text, but they are very difficult and cumbersome to use. TMIL was designed from the ground up to have a clean and simple syntax so that users can do repetitive and complicated imaging tasks quickly and efficiently.

2 Motivation and Features

Our primary motivation for creating TMIL is to create a specific purpose language that is easy enough to enable developers of all backgrounds to use, while remaining powerful enough to draw exactly what the programmer wants. The Java Paint2D and the GD2 library for C++ are both powerful, but require a lot of effort for even the simplest projects. Other command based image editing application such as ImageMagick require fine tinkering of command line arguments in un-understandable order. While the simple MSPaint and expansive Adobe Photoshop offer similar text on image capabilities, they require user interaction. The TMIL Language offers the developer a way to automatic adding text to images in a construct that feels **natural**.

The specificity of the language also brings with it **security**. Administrators can give free compilation and execution rights for TMIL code and applications knowing the user is limited in his power. The user is constraint to editing images and has no access to other parts of the system through TMIL code.

TMIL uses standard and recognizable constructs such as if statements and while loops. The language also supports native objects such as integers and strings as well as standard functions that are normally available for such objects. The simple tools give a programmer a lot of manipulation ability.

TMIL code flow is **intuitive**, allowing developers to lay down text in the code the same way they would think about doing it interactively. The user sets the properties of the text such as font, font-size, color etc he/she wishes to set on the image, and stamps it on. The location of the text and even whether or not to rotate the text are all properties of the text object and can be changed anytime until the text is stamped. This allows the programmer to do what he/she feels most natural.

TMIL is **flexible**. The user can create one text object and stamp it onto many images, or stamp many text objects sequentially onto a single image. Obviously, he/she can do a combination of both.

3 Example of Syntax

The simple code example described here loads an image and writes text on it at different positions and with different colors. Many other attributes, such as rotation, size, and font can be manipulated, but are not shown in these examples. Arrays of types are also supported.

```
int i = 0;

image im = open("namefile.png");   // load image

text t;

t::string = "dog";              // assigns value "dog" to property string of t
```

```
t::font = "arial";              // assigns arial font to t

t::size = 18;                   // assigns a size to t

color c = [0, 0, 0];

for (i=0:2) {

      c[i] = 255;

      t::color = c;                 // assigns value c to the color property of t

      t::position = [i*10,i+20];    // assigns a value to the position of t

      im <-- t;                     // stamps t to the image im

}

save(im, "namefile.png");           // saves the result
```

Input and output of this sample code are presented in Figure 1 (a) and (b).

(a) (b)

Figure 1: (a) Input and (b) output of the sample code presented.