Tianliang Sun, ts2825
Jingyi Guo, jg3421
Xinan Xu, xx2153

CONCISE SLIDES LANGUAGE (CSL)
PROJECT PROPOSAL

Motivation

Presentation software has been widely used to prepare informational slideshows in meetings and conferences. Various commercial and open-source presentation software programs have been developed in the pursuit of enhancing user experience and speed up the process of making presentation materials, such as Microsoft Powerpoint, Apple Keynote, Openoffice Impress, etc. While all of those software have a huge runtime library and elaborate graphical user interface, they often do not provide the precision and simplicity required for developing professional presentations. In terms of precision, for example if the user want to resize and then align some texts or images in a slide perfectly, it is very hard to do so with a GUI-based software, because mouse control is error-prone and never precise. In terms of simplicity, what would you do if you want to create a large set of similar slides but each of them will be some minor changes, or in one slideshow you want to create a large number of similar textboxes with their positions strictly defined? Doing so in a GUI-based application could be a pain. So having a programming language capable of creating presentations in a highly structured and well defined approach will really help lots of users.

This kind of language does exist, such as VBA in Powerpoint and beamer module in Latex. But they usually have redundant and ugly language syntaxes, and are not designed specifically for presentation development. Accordingly, it takes a lot of extra development effort to do work well with this kind of tasks.

Therefore, we would like to propose our Concise Slides Language(CSL), aiming at providing an easily understood and well defined method to make the process of making presentation slides more enjoyable and efficient to users.

Description of Language

CSL is a compact programming language used to facilitate the process of making slides. While only providing minimal language support for users to create presentation slides, CSL gives users the ability to precisely define the objects in the slides, such as positioning using 2-D coordinates and coloring using RGB values. All those were hard to accomplish under traditional presentation software. Meanwhile, CSL can also keep users away from repetitive tasks, such as reformatting 100 different textboxes in 10 separate slides, with the help of the built-in if/else and for loop control structures in CSL. This is particularly important for developing and editing large sets of professional presentations with similar content under corporate environment.

The syntax of CSL is C-alike, but it removes a lot of redundancies in C which will not be used for developing presentation slides. CSL contains primitive types int, char, bool and string. It also includes built-in data structures such as Text(textbox), Image, Shape and Animation, all of which are frequently used in slideshows. To create a presentation, a user will first define required global variables (CSL does not support scoping of variables, however) such as window size and position. Then the user will define new slides and the elements in them one by one and add the slides into the slides vector slides[]. Each slide contains a content map, which is basically a Hashmap with keys being names of the elements and values being the corresponding elements in this slide. A slide also has a animation queue, which queues up all the animations in this slide in order. Once the CSL compiler finishes building the presentation, it will play the slides according their ordering in vector slides[], and each slide will animate its contents based on its animation queue.

To define the contents of a particular slide, the user creates built-in data structures such as Text and Image and then sets their properties accordingly. Those properties typically contains size, position, color, and they varies among different types of slide elements. The user defines animations and transitions by creating Animation data structures and push them into the animation queue. The required parameters for creating an Animation includes the target element, the animation function, the parameters of the animation function, and the exit parameter specifying what action it should take when an animation finishes. The user can define customized animation functions and pass them to the Animation data structure. Basically the animation function will be 2-D transition matrix to represent the pixel-wise transition of the target between two consecutive frames.

Language Syntax and Built-in Features

Whitespace and Special Symbols

Whitespace, including single space, multiple spaces, tab, etc., are used to separate tokens in CSL. All of them are considered as the same, i.e. replacing a single space with a newline or tab will not change the program.

Other special symbols, such as {}, (), “”, ‘’, and ; can separate tokens in CSL as well. {} are typically used in function definitions and control structures to indicate the scope of a code block; () are used to nest and prioritize expressions; “”, ‘’ are used to define strings and characters respectively; the ; symbol is to indicate the end of a single line of source code.

Indentation
Indentations do not matter in CSL, although it is highly recommended to use the standard indentation style to improve code quality and it is a good programming practice.

Primitive Data Types

int	a 32-bit signed integer
char	an ASCII character
bool	a standard boolean variable
string	an array of ASCII characters

Mathematical Operation

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	mod
|	or
!	not
&	and

Logical Operators

<	strictly less than
<=	less than or equal to
>	strictly bigger than
>=	bigger than or equal to
!=	not equal to
==	if identical

Comments

CSL only supports one comment style that starts with a /* and ends with */. Everything in between, including characters, symbols, and whitespace will be considered as comment. Nesting of comments is supported in CSL.

Examples:

```
/* This is for single line comment */

/*
 This comment has
 multiple lines
*/

/* Useless code block
 int a = 1;
 int b = 1;
 int c = a + b; /* integer addition */
 Still a comment
*/
```

Function Prototype

Defining a function in CSL is almost the same in C, except the func keyword at the beginning.

```
func returnType functionName(param1,param2 ...)
{
        function body
}
```

Function Call

```
functionName(para1,para2 ...);
```

Built-in Data Structures

vector<type>	a resizable array containing arbitrary data types
map<type, type>	a Hashmap holding key/value pairs

Text	a Textbox in a slide
Shape	a geometric shape in a slide
Image	an image in a slide
Animation	an animation of an element in a slide

Control Flow
CSL only supports if/else and for loop control structures, but those are sufficient to implement any algorithms that will be used in preparing presentation slides.

Examples:

```
if (expression)
{
        do something
}
else if (expression)
{
        do something
}
else
{
        do something
}

for(initialization; boolean_expression; update)
{
  do something
}
```

Example Program
helloworld.sl

```
/* define global variables */
window_width = 1024;
window_height = 768;
```

```
window_pos_x = 0;
window_pos_y = 0;

/* define a new slide */
slides[0].init();
/* define elements in the new slide and add them to the object list */
text text1; /* call the default constructor */
text1.text = “hello world”;
text1.color = COLOR_BLACK;
text1.pos_x = 10;
text1.pos_y = 10;
/* COLOR_BLACK and FONT_ARIAL are pre-defined macros in CSL, which are integer
mappings to colors, fonts, etc. */
text1.font = FONT_ARIAL;
text1.width = 100;
text1.height = 50;
 /* add this new element to the object map of slides[0] */
slides[0].addObj(“hello text”, text1);

/* define animations for the new slide and add them to the animation queue */
animation animation1(text1, resize, 2, 1000, 2000);
/* add this animation to the animation queue of slides[0] */
slides[0].addAnim(animation1);

animation animation2(text1, move, 100, 100, 1000, ANIM_RET_WAIT_MOUSE_RIGHT);
slides[0].addAnim(animation2);
 /*
ANIM_RET_WAIT_MOUSE_RIGHT and the 2000 in the previous animation definition
specify the exit actions of the animations. 2000 means it waits for 2000ms and play the
next animation, while ANIM_RET_WAIT_MOUSE_RIGHT is a macro that indicates the
presentation will wait for a right mouse click to play the next animation.
*/
/* create a new slide to say goodbye */
slides[1].init();
/* a faster way to define a new textbox */
text goodbye(“Bye!”, COLOR_BLACK, 10, 10, FONT_ARIAL, 100, 50);
slides[1].addObj(“goodbye text”, text1);
```