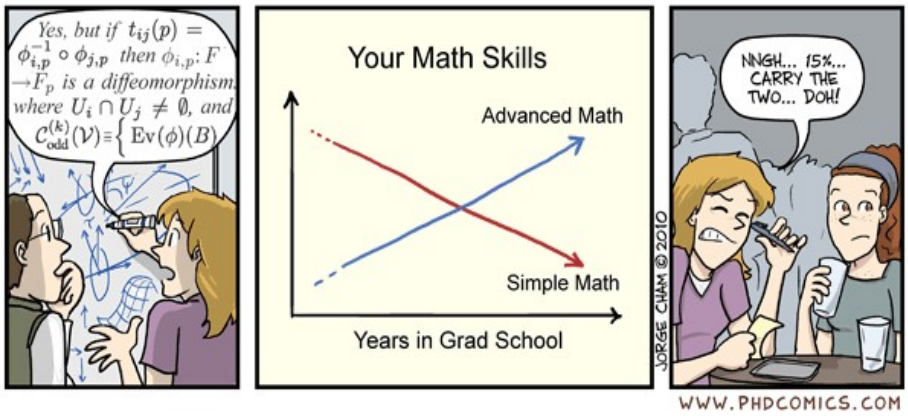
CS 218 – Assignment #10

Purpose: Become more familiar with data representation, program control instructions, function handling, stacks, floating point operations, and operating system interaction.
Due: Tuesday (3/06)
Points: 150

Assignment:

In mathematics, parametric equations[1] are commonly used to express the coordinates of the points that make up a geometric object such as a curve or surface.



Write a simple assembly language program to plot the provided parametric equations. The moving figure will appear somewhat like a jellyfish[2].

Use the provided main program which includes the appropriate OpenGL initializations. You will need to add your code (three functions) in the provided functions template.

The program should read window height, window width, draw speed, point size and color values from the command line. For example:

```
./jelly -h 500 -w 500 -sp 3 -ps 2 -cl 9484807
```

The required format for the date options is: "-h <base36 number>", "-w <base36 number>", "-sp <base36 number>", "-ps <base36 number>", and "-cl <base36 number> with no spaces in each specifier. The program must verify the command line format, read the arguments, and ensure the arguments are valid. The program must also ensure that the ***height***, ***width***, ***speed***, ***pointSize***, and ***color*** values are between the specified ranges (provided constants). If there are any command line errors, the program should display an appropriate error message and terminate. Refer to the sample executions for examples of the error handling.

Submission:

Only the functions file will be submitted. As such, the main file cannot be altered in any way. When complete, submit only the functions source file:

- A copy of the functions **source file** via the class web page (assignment submission link) by 23:55 PM. ***Assignments received after the due date/time will not be accepted.***

1 For more information, refer to: https://en.wikipedia.org/wiki/Parametric_equation
2 For more information, refer to: https://en.wikipedia.org/wiki/Jellyfish

Functions

The provided main program calls the following routines:

- Boolean function ***getArguments()*** to read and check the command line arguments (***height***, ***width***, ***drawSpeed***, ***pointSize***, and ***color***). The function must read each argument, convert each ASCII/base36 string to integer, and verify the ranges. The ranges for ***height***, ***width***, ***drawSpeed***, ***pointSize***, and ***color*** are provided in constants H_MIN/H_MAX, W_MIN/W_MAX, SP_MIN/SP_MAX, PS_MIN/PS_MAX, and CL_MIN/CL_MAX (inclusive). If all arguments are correct and within the ranges, return the values (via reference) and return to the main with a value of TRUE. If there are any errors, display the appropriate error message (provided) and return to the main with a value of FALSE. This function should call the ***cvtB362int()*** function (five times).
- Boolean function ***cvtB362int(strAddr, intAddr)*** to convert the passed base-36 string into an integer. If the conversion is successful, the function should return the number (via reference) and the TRUE. Otherwise, the function should return FALSE. The code from a previous assignment should be converted into a function.
- Void function ***drawJellyfish()*** to plot the torus functions noted below. The functions should be iterated and will generate a series of (***x***,***y***,***z***) values which must be plotted.

$$\textit{for}\left(\textit{double}\, u = 0.0;\ u < (2.0 * \textit{pi});\ u \mathrel{+}= \textit{tStep}\right)$$

$$\textit{for}\left(\textit{double}\, v = -1.0;\ v < 1.0;\ v \mathrel{+}= \textit{tStep}\right)$$

$$x = \sqrt{\left((2.0 * s - 1.0) * v + 0.2\right)} * \cos(u)$$

$$y = \sqrt{\left((2.0 * s - 1.0) * v + 0.2\right)} * \sin(u)$$

$$z = \left(1.0 - (2.0 * s - 1.0)^2\right) * v$$

The ***t*** values should range between **0.0** and ($\mathbf{2.0 \times \pi}$) and the ***u*** values should range between **-1.0** and **1.0**. Each should be incremented by ***tStep*** (predefined) each iteration. As such the first loop will require ($\mathbf{2.0 \times \pi}$) / ***tStep*** iterations and the inner loop will require **2.0** / ***tStep*** iterations. *Note*, calculating the number of iterations will avoid a floating point compare. Before leaving the function, the ***s*** value should be incremented by ***sStep*** which must be set as follows;

$$\textit{sStep} = \frac{\textit{drawSpeed}}{\textit{fltScale}}$$

The function is called repeatedly which generates the animation (based on the changing ***s*** value between successive calls).

OpenGL Installation

For this assignment, we will be using the openGL (graphics library) to provide some basic windowing capabilities. As such, the OpenGL development libraries must be installed. This can be done via the command line with the following commands.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install binutils-gold
sudo apt-get install libgl1-mesa-dev
sudo apt-get install freeglut3 freeglut3-dev
```

It will take a few minutes to install each. You must be connected to the Internet during the installation. After the installation, a re-install of Virtual Box Guest Additions may be required.

OpenGL Calls

The basic OpenGL library calls are included in the provided main and functions template. In order to set the draw color and plot the point, the following OpenGL functions will be required.

```
call glColor3ub(red, green, blue);
call glVertex3d(x, y, z);
```

The ***red***, ***blue***, ***green*** variables are unsigned bytes. The ***x, y,*** and ***z*** variables are double precision floating point values. These calls follow the standard calling convention.

Assembly:

The program must be linked with the OpenGL libraries. An assembly and link script file (asm10) is provided and must be used. *Note*, the script file will require execute privilege (i.e., **`chmod +x asm10`**). To assemble/link script can be executed as follows:

```
ed-vm% ./asm10 jelly a10funcs
```

Assuming the main file is named **`jelly.cpp`** and the functions file is named **`a10funcs.asm`**. *Note*, the **`ed@vm%`** is the prompt on my machine.

Debugging -> Command Line Arguments

When debugging a program that uses command line arguments, the command line arguments must be entered after the debugger has been started. The debugger is started normally (ddd <program>) and once the debugger comes up, the initial breakpoint can be set. Then, when you are ready to run the program, enter the command line arguments. This can be done either from the menu (Propgram -> Run) or on the GDB Console Window (at bottom) by typing **`run <commandLineArguments>`** at the (gdb) prompt (bottom window).

Example Executions (with errors):

Below are some example executions with errors in the command line. The program should provide an appropriate error message (as shown) and terminate. *Note*, the **`ed@vm%`** is the prompt.

```
ed-vm% ./jelly
Usage:  ./jelly -h <base36 number> -w <base36 number> -sp <base36 number>
                -ps <base36 number> -cl <base36 number>
ed-vm%
ed-vm% ./jelly -h go
Error, invalid or incomplete command line arguments.
ed-vm%
ed-vm% ./jelly -h large -w g0 -sp 200 -ps 2 -cl 73
Error, height value must be between 2S(36) and XC(36).
ed-vm%
ed-vm% ./jelly -h go -w go -sp 1100 -ps 2 -cl 73
Error, draw speed value must be between 1(36) and RS(36).
ed-vm%
ed-vm% ./jelly -h go -w go -sp 30 -ps 2 -cl 7
Error, color value must be between 1E(36) and 9ZLDR(36).
ed-vm%
ed-vm% ./jelly -ht go -w go -sp 30 -ps 2 -cl 73
Error, height specifier incorrect.
ed-vm%
```

openGL Errors

Based on the specific hardware configuration and/or virtual box configuration, the following warning message may be displayed.

```
OpenGL Warning: Failed to connect to host. Make sure 3D acceleration is enabled
for this VM.
```

This warning message can be ignored. *Note*, some hardware configurations, especially older hardware, using virtual box may not be able to use openGL. An *openGLtest* program is provided to verify the openGL installation.

Testing

A script file to execute the program on a series of predefined inputs will be provided. The test script executes the program and performs a series of error tests (with expected output). Refer to the examples for output formatting and error handling. The test script, named **`a10tst`**, can be executed as follows:

```
ed-vm% ./a10tst jelly
```

The expected error will be shown and then the program is executed (with the specified error).

Example Execution:

Below is an example execution showing the resulting image.

```
ed-vm% ./jelly -h go -w go -sp 10 -ps 2 -cl 9Y70F        (left image)
ed-vm% ./jelly -h go -w go -sp 10 -ps 2 -cl 9Y70F        (right image)
```

When functioning, the image can be rotated by typing the up, down, right, or left arrow keys. The animation speed can be increased by typing a '***f***' or decreased by typing a '***s***'. The window must be selected.