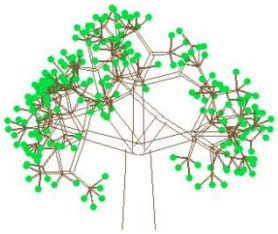
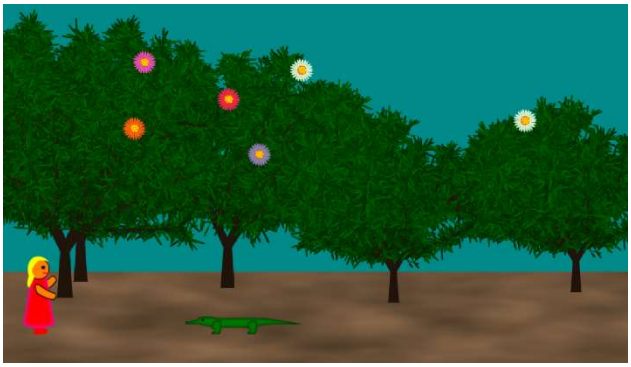
ECE220: Computer Systems & Programming **Fall 2020 ZJUI**
Lab 9 **Tuesday 17 November**

Drawing Trees



In today's lab, you will learn how to write recursive functions to draw an imaginary tree. Not a software tree, but a real tree, as shown to the right. Better than a poem by Kilmer[1], at any rate. You will also have an opportunity to reuse the code that you wrote for MP5, giving you some experience in building new abstractions out of existing elements. MP8 focuses on recursion for image processing, and you will reuse your MP5 code again for MPs 9 and 10.

Begin by checking out the **`lab9`** subdirectory in your Subversion repository. The directory contains a copy of this document (**`lab.pdf`**), slightly-modified copies of the MP5 files **`mp5.h`** and **`main.c`**, a **`Makefile`**, and a C source file, **`lab9.c`**, with which you can start this lab. Next, copy your **`mp5.c`** solution to the **`lab9`** directory. You should then be able to "**`make`**" the tree image, **`tree_image.png`**, but it will be empty until you write the main part of the code.



The Algorithm and Random Number Usage

The algorithm used to create the tree is loosely based on my recollection of that presented in Weber and Penn's 1995 SIGGRAPH paper. I used something closer to their algorithm (and my own parameter values) to generate the trees shown in the game to the right. If you are interested in learning more, the SIGGRAPH paper is a good starting point.

Before we discuss the algorithm, you need to understand the basics of pseudo-random number generation. Such generators produce "random" numbers, but do so in a reproducible way. So long as you use exactly the same starting point (called the *seed*) and the same calls for the same purposes (and the same algorithm), you get the same sequence of "random" numbers. If, however, you change anything—the seed, the algorithm, the way in which you use the random numbers, the number of times you ask for a random number—anything!—you get different results.

That said, you must use **`double drand48 (void)`**, which returns a random number uniformly distributed in the range [0,1), along with the order and equations given below. I'm not sure whether your VM and my laptop use the same implementation for **`drand48`**. If not, you cannot reproduce the tree shown above. If they do use the same implementation, and you follow the rules, you should get the same tree.

The Task

For this lab, your group must implement a recursive function to draw a branch and then recursively draw sub-branches (and, eventually, leaves).

[1]A not-very-good early 20th century poet with, so far as I know, exactly one moderately well-known poem. About trees.

The function signature is

```
static void build_tree (int32_t level, int32_t x, int32_t y,
                        double angle, double width);
```

The function emits MP5-based commands for drawing a branch from the point **(x,y)** in the direction specified by **angle**. The initial width of the branch is also given (**width**). The branch's length and final width are calculated by the function. The level parameter specifies the branching level: 0 is the trunk, 1 is the branches coming out of the trunk, and so forth. Level 5 is leaves.

You can start by writing the leaf code, although it won't be used for a while unless you add code to **main** (do so for a quick test). Leaves are green circles (I used **0x00E040** for the color) of radius 5 centered at **(x,y)**. Leaves should not call **build_tree** recursively.

For a leaf, your code should print one call to **set_color** and one call to **draw_circle**. The output of your program is written into a C source file, **draw_tree.c**, which is then compiled and executed with your MP5 code to produce **tree_image.png**. Be sure to print valid C code, or "**make**" will fail. Be sure to return after making the leaf, or your function may recurse infinitely—leaves are the stopping condition!

Now you can write the branch code. Remember that if you want to produce the tree shown in this document, you must follow the order of random number usage exactly.

First, decide on the length of the branch: $length = 60\ (\text{drand48}\ () + 1)\ 0.666666^{level}$. Note the use of **drand48**, and use the **pow** function in the math library to implement the exponent.

The other end of the branch is then at

$$(x_{end}, y_{end}) = (x + length\ \cos(angle),\ y - length\ \sin(angle)).$$

Next, calculate the width of the far end of the branch, $width_{end} = width\ \left(\frac{\text{drand48}\ () + 2}{3}\right)$.

The length and the ending width are all that you need to set the color and draw the two lines for the branch. I used 0x795B35 for the color. The two lines are from

$$(x - width\ \sin(angle),\ y - width\ \cos(angle)) \quad \text{to}$$
$$(x_{end} - width_{end}\ \sin(angle),\ y_{end} - width_{end}\ \cos(angle)), \quad \text{and}$$
$$(x + width\ \sin(angle),\ y + width\ \cos(angle)) \quad \text{to}$$
$$(x_{end} + width_{end}\ \sin(angle),\ y_{end} + width_{end}\ \cos(angle)).$$

Finally, it's time to recurse. The number of sub-branches is given by

$$numchildren = \lfloor 2 + (4\ \text{drand48}\ () \rfloor.$$

Note that the floor function can be implemented by using an **int32_t** for the number. The span of the children's new angles is given by

$$anglespan = \frac{\pi}{2}(1 + \text{drand48}\ ()).$$

Use the math library constant **M_PI_2** in place of the π/2 factor.

Use one recursive call for each child, starting from the largest angle value, $(angle + \frac{anglespan}{2})$, and ending with the smallest one $(angle - \frac{anglespan}{2})$, with angles uniformly distributed across the children, And, for the width, use $\frac{width_{end}}{2}$. That's it! Good luck with your recursion, and may your trees be beautiful!