## CS-AD-216: Foundations of Computer Graphics

## Assignment 2, Due: September 22

---

**Instructions:**

- Please read Chapters 2 and 3 of the textbook carefully before attempting the exercises.
- Assignments can be submitted in groups of at most three. The purpose of groups is to learn from each other, not to divide work. Each member should participate in solving the problems and have a complete understanding of the solutions submitted.
- Submit your assignments as a zip file (one per group) which includes the Common directory and a separate directory for each of the assignments so that I can run your code by just extracting your files and double-clicking the html files.

---

**Problem 1** (10 points)**.**
Write a program that displays a cube, each of whose faces is monochromatic but different faces should have different colors. You program should also provide three sliders whose ranges are 0 to 360 with step size of 5. Let $\alpha$, $\beta$ and $\gamma$ be the values corresponding to these sliders. Then your program should display a cube that is obtained from the original cube after the following transformations: rotation about $x$ axis by angle $\alpha$, followed by the rotation about the $y$ axis by angle $\beta$, followed by rotation about the $z$ axis by angle $\gamma$.

Note that the order of rotations is fixed. Since you know how to do each transformation separately, you just need to do them one by one in the prescribed order in the vertex shader. The angles are specified in degrees. Don't forget to convert to radians and don't forget to enable depth test. You may want to write functions that make it easier for you to create the arrays of vertex positions. For instance, a face of the cube defined by the corners $a$, $b$, $c$ and $d$ is defined by two triangles. So, you may want to write a function `quad(a,b,c,d)` that inserts the data for the two triangles into the appropriate array.

**Problem 2** (10 points)**.** (**Turtle Graphics and L-Systems**)
In 2D turtle graphics, we have a turtle that can move around in the plane and draw line segments. In this exercise you will implement a turtle which is given instructions for drawing using strings of characters, each character representing a specific action. At any point in time the turtle has a "current configuration" which is represented by a triple $(x, y, \theta)$ where $(x, y)$ is the location of the turtle in the plane and $\theta$ is the angle made by the direction the turtle is facing with the positive $x$ axis. For example, for a turtle sitting at the location $(-0.4, 0.3)$ and oriented in the direction of the positive $y$ axis, the triple representing its configuration is $(-0.4, 0.3, \pi/2)$. Below are the characters we will use along with the corresponding actions:

- $f$ : move forward by distance 1 without drawing anything
- $F$ : move forward by distance 1 while drawing a line segment from the old position to the new position
- $+$ : turn left by a pre-specified angle $\alpha$
- $-$ : turn right by a pre-specified angle $\alpha$

- [ : save current configuration into the stack
- ] : pop the top element from the stack and use the popped element as the current configuration

Note that we need to specify in the initial configuration and the turn angle $\alpha$ before the turtle can interpret our commands. We are fixing the distance we move in 'f' and 'F' to 1. In general, one could make it a variable parameter.

**Examples:** If the initial configuration is $(0, 0, 0)$, and $\alpha = \pi/2$, then the instruction "$F+F+F+F$" makes the turtle draw an axis parallel square of unit side length having its bottom left corner at $(0, 0)$. The instruction"[F]+[F]+[F]+[F]" makes the turtle draw four segments of unit length each having one end point at $(0, 0)$ and going in the four directions corresponding to the positive and negative $x$ and $y$ axes.

**Note:** In general, the instruction string may have characters other than the ones mentioned above, in which case the turtle should ignore them. So, in the above example, the instructions "$F + F + F + F$" and "$XF + F + YF + ZF$" should have the same effect as 'X', 'Y' and 'Z' do not correspond to any actions.

We also want to have "production rules" that specify how new strings can be obtained from a given string by repeatedly replacing each occurrence of a character with another string.
For instance, let us say we have an initial string "$FX$" and the production rules "$X \mapsto X+YF+$" and "$Y \mapsto -FX-Y$". The production rules say that every instance of $X$ is to be replaced by the string "$X+YF+$" and every instance of $Y$ is to be replaced by "$-FX-Y$". Applying the production rules once to the initial string gives us the string "$FX+YF+$". Applying the production rules again to this new string gives us the string "$FX+YF++-FX-YF+$". The initial string is often called an ***axiom.*** We will assume that there is only one axiom.

In general, we start with the axiom and apply the production rules a few times and the use the resulting string as instruction for the turtle. This final string may have characters (like $X$ and $Y$ in the example above) that don't represent any action and, as mentioned before, are to be ignored. These characters are used only to guide the evolution of the string through production rules.

Your goal in this exercise is to write a program in which once you specify the initial configuration, the turn angle $\alpha$, the axiom, the production rules and the number of times the production rules are to be applied, the resulting picture drawn by the turtle is displayed on the screen.

Since WebGL displays only the portion of the 2D plane with $x$ and $y$ coordinates between $-1$ and 1, you will need to scale the picture appropriately so that so that it fits on the screen.

You can use objects of the following type for configurations:

```
var initial_config = {
    x: 0,
```

```
    y: 0,
    theta: Math.PI/2
};
```

For production rules, you can use objects like the following:

```
var production_rules = {
    X: "X+YF+",
    Y: "-FX-Y"
};
```

Write a function:
`turtle(initial_config, alpha, axiom, production_rules, num_productions)` $\{\cdots\}$

which takes the necessary parameters described above and returns a javascript array containing the segments drawn by the turtle. Each segment has two end points with two coordinates each and therefore is described by four numbers. The first four numbers in the array should describe the first segment, the next four numbers should describe the next segment and so on. The data in this array can then be stored in a buffer in the GPU and displayed via a call like `gl.drawArrays(gl.LINES,0,num_vertices);`

Please use the skeleton code attached.

Here are a few things to try:

- initial configuration: $(0, 0, 0)$, $\alpha$: $\pi/2$, axiom: F, production rules: $F \mapsto FF + F + F + FF + F + F - F$
- initial configuration: $(0, 0, 0)$, $\alpha$: $\pi/2$, axiom: F+F+F+F, production rules: $F \mapsto F + F - F - FF + F + F - F$
- initial configuration: $(0, 0, \pi/2)$, $\alpha : \pi/8$, axiom: F, production rules: $F \mapsto FF + [+F - F - F] - [-F + F + F]$
- initial configuration: $(0, 0, 0)$, $\alpha$: $\pi/9$, axiom: X, production rules: $F \mapsto FF$, $X \mapsto F[+X]F[-X] + X$.

You may also want to figure out how to draw the "Sierpinski arrowhead curve" (check wikipedia) using your program.