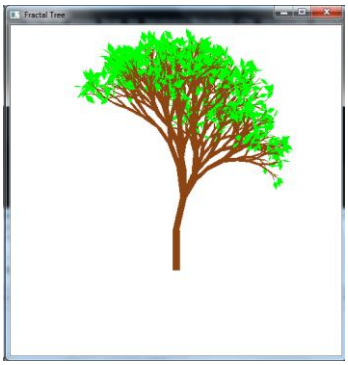
Advanced Graphics Programming

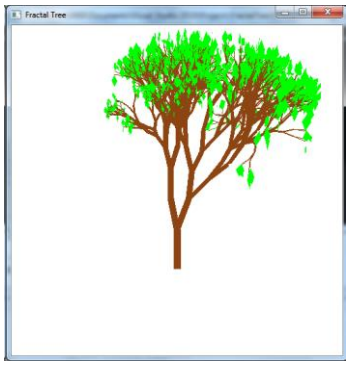
Workshop Eight (Part C) – 3D Tree and Camera Controls

In this workshop you will extend workshops 8 (part A and B) to 3D.

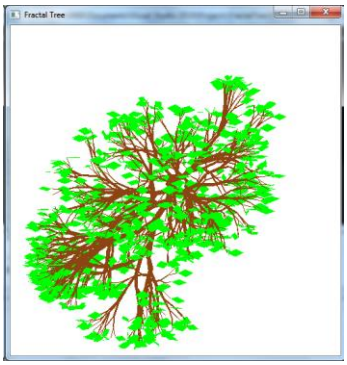
Near the end of workshop 8 (part B) you rotated the camera left and right using equations based on the polar coordinate system. Let's start by extending your camera to rotate in 3D using spherical coordinates and then apply the same concepts to rotate your branches in 3D. By the end of this tutorial you should be able to create the following tree:



Front view



Side View



Top View

Step 1: Starting with the camera controls switch from polar coordinates to their 3D representation, spherical coordinates.

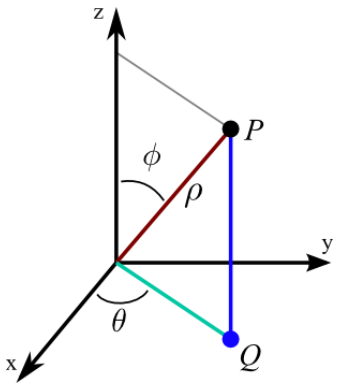


Figure 1 Spherical coordinates: ϑ is the angle between the positive x-axis and the line segment from the origin to Q. Angle φ is the angle between the positive z-axis and the line segment from the origin to P. [Math Insight]

Add a new angle variable cameraPhi and initially, set it to 0. Update your existing line of sight equations, based on the camera angles to:

- los.x = cos(cameraPhi) * sin(cameraTheta)
- los.y = sin(cameraPhi)
- los.z = cos(cameraPhi) * -cos(cameraTheta)

Run your code and check the camera controls work exactly as they did before.

Step 2: Add two new keys to rotate the camera forward and backward. When the user presses these keys either add or subtract a small amount to cameraPhi. Use the same equations as above when the user changes the angle to update the line of sight (**los**).

Run your code and check that you can rotate the camera forward and backward as well as up and down.

Step 3: Now update the code for the forward, backward and strafe controls so that you can move in any direction. You have already done this for x and z so it should be simple to update y in a similar manner.

Run your code and check that forward, backward and strafe still work, especially after rotating the camera.

Step 4: Now update the code for the top views so that when you switch to these views all the camera controls still work.

Hint: you will need to set the cameraTheta and cameraPhi to the correct values and then use the equations from step 1 to update the line of sight.

As you are now able to move your camera in the 3D environment the next steps are to generate 3D versions of the trees you created in Workshop 8 (Part A). To simplify the generation of 3D trees the next steps will guide you to change the rotation code of the branches from the equations based on Cartesian coordinates to those on Spherical coordinates.

Step 5: Update the equations used to create your branches from 2D to 3D as shown below. The rotations are similar to the camera rotations but have the added variable m. This is because the camera equations were simplified by setting m =1, as the line of sight was a unit vector but for the tree m will need to be the length of the current branch. You will need to add new variables for the phi and theta of the right and left branch, with phi (φ) initialised as 0 and theta (θ) as 90 degrees.

1) Scale the trunk (or previous branch) by ratio, r so that the next branch is smaller than the previous:
    a. $x_s = (x_1 - x_0) \times r$
    b. $y_s = (y_1 - y_0) \times r$
    c. $z_s = (z_1 - z_0) \times r$
2) Rotate anticlockwise (default) by **half** of the angle to get the left branch:
    a. $m = \sqrt{x_s + y_s + z_s}$
    b. $\theta_l = \theta + \frac{\alpha}{2}$
    c. $\varphi_l = \varphi + \frac{\alpha}{2}$
    d. $x_l = m \times \cos(\varphi_l) \times \cos(\theta_l)$
    e. $y_l = m \times \cos(\varphi_l) \times \sin(\theta_l)$
    f. $z_l = m \times \sin(\varphi_l)$

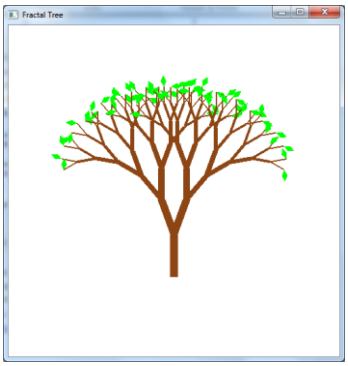
    Hint: In C++ cos() and sin() are calcuated in radians so you will need to convert degrees to radians before calling it, glm has a function to do this!

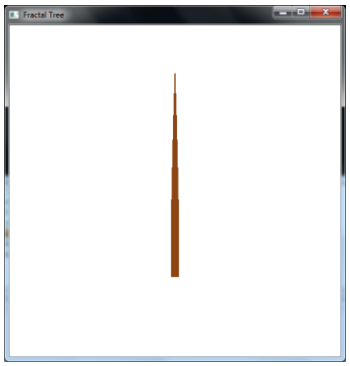
3) Move the left branch to the end of the previous branch:

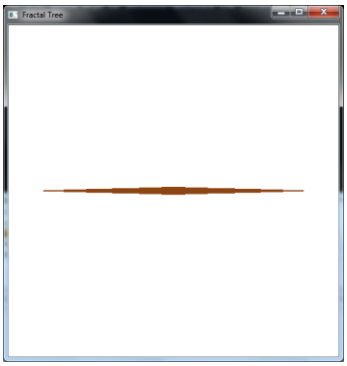
a. $x_2 = x_1 + x_l$
b. $y_2 = y_1 + y_l$
c. $z_2 = z_1 + z_l$

4) Repeat the above steps but rotate clockwise $(-\alpha/2)$ to get the right branch.

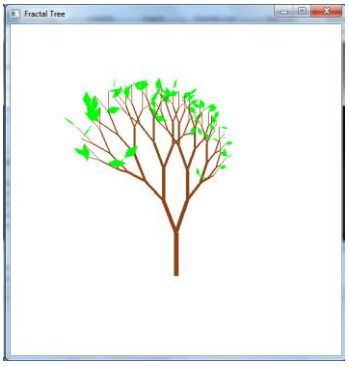
If you run your code now you should have the same 2D tree as before but you have done all the preparation to enable 3D branch rotations.

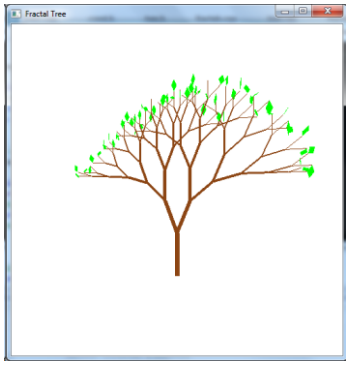

Front view

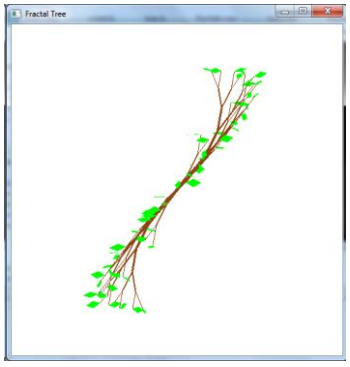

Side View


Top View

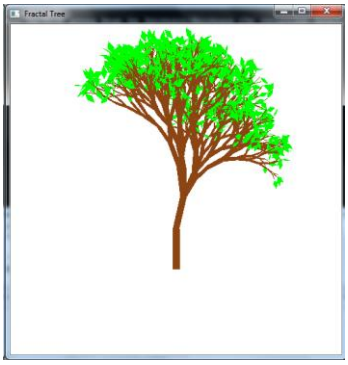
Step 6: Left and right is straightforward but then when it comes to forward and backward rotations of the branch it is not so simple. Start by rotating the right branch forward and the left branch back (by changing the angle phi) you will not get a very natural looking tree, as shown below but you will be able to test if you can rotate branches in 3D space.


Front view

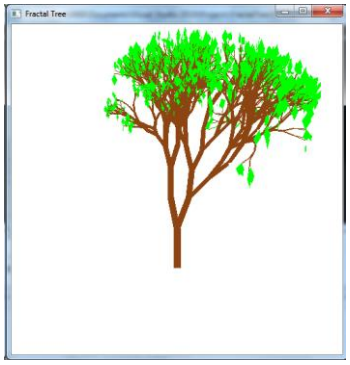

Side View


Top View

Step 7: If you now update your code to randomly decide whether to rotate a branch forward, backward, left or right, your tree will appear much more natural. You can take this further and randomly vary the angle you add and subtract at each step. In the example below I have randomly added or subtracted an angle between 10 and 20 degrees. I have also added a few more iterations to give a fuller effect. If you haven't down so already you will also need to rotate your leaves in 3D and you should end up with something like:



Front view



Side View

Top View

You can also vary the length of the branches and change the seed value of the random function to create different variations of the tree.

Modify **computeSingleBranch** function

```
void computeSingleBranch(int depth, float angle,float x0,float y0, float z0, float x1, float y1, float z1,
float &x2,float &y2, float &z2)
{
	float xs, ys, zs, xll, yll, zll, xl, yl, zl, m;
	double val;

	xs = (x1 - x0)*R;
	ys = (y1 - y0)*R;
	zs = (z1 - z0)*R;

	m = sqrt(xs*xs + ys*ys + zs*zs);

	xll = cos(angle / 2.0)*xs - sin(angle / 2.0)*ys;
	yll = sin(angle / 2.0)*xs + cos(angle / 2.0)*ys;
	zll = 0.0;

	//roate around angle / 2.0 + 3.1415926/2.0 in other axis
	srand(3); // get different srand values //seed
	if(depth % 2==0)
		val = -2.0;
	else
		val = 2.0;

	xl = cos(val*angle / 2.0 )*xll - sin(val*angle / 2.0 )*zll;
	yl = yll;
	zl = sin(val*angle / 2.0 )*xll + cos(val*angle / 2.0)*zll;

	//xl = m *cos(angle / 2.0)*cos(angle / 2.0 + 3.1415926/2.0);
	//yl = m *cos(angle / 2.0)*sin(angle / 2.0 + 3.1415926/2.0);
	//zl = m *sin(angle / 2.0);

	x2 = x1 + xl;
	y2 = y1 + yl;
	z2 = z1 + zl;
}
```