Ellie Ansell
i7685565
B.A. Computer animation and visualisation
Level I
Computing for Animation 1

2. Code Documentation/Design and report on algorithms used

Files

L_System/
- Include/
  - GLWindow.h
  - MainWindow.h
  - stringmanipulator.h
  - lsystem.h
  - drawplant.h
  - weather.h
  - particle.h
- Src/
  - GLWindow.cpp
  - MainWindow.cpp
  - stringmanipulator.cpp
  - lsystem.cpp
  - drawplant.cpp
  - weather.cpp
  - particle.cpp
- UI/
  - MainWindow.ui
- Shaders/
  - Phong.fs
  - Phong.vs
  - TextureFragment.glsl
  - TextureVertex.glsl
- Obj/
  - leaf.obj
  - pot.obj
  - soil.obj
  - potTexture.jog
  - soil.jpg
  - green.jpg
- L_System
- L_System.pro
- README.md

GUI Controls

Rain : Increases the hit count to grow the plant by increasing the height, depth and angle incrementation.

Confetti : Negates the hit count of GLWindow and does the opposite of rain; the height, depth and angle increment decrease.
Sun : Toggles whether the sun is on or off.
Sun intensity : This alters the sun's colour.
Sun position : Changes the sun's position so the branches tend towards this point.
Wind Intensity : As wind intensity increases, the amount the vectors "sway" by increases.
Wind Direction : Changes the direction the wind is pointing towards.
View distance : The camera slides further away from the plant.
Height : The length of each vector of the branches.
Smoothness : Segments the vectors. So for example, a branch made of 1 vector would turn into a branch of 3 vectors if smooth Level is 3. The angle Increment also increases marginally so branches aren't uniformly straight.
Depth : Depth increases the recursive depth of the string.
Branch rotation : Rotates the branches by X, Y and Z angles.
Leaves : Adds leaves to the branch within a for-loop for leafList.
Axiom : Adds the axiom when the button is pressed.
Rule : Adds the rule when the button is pressed.

Report on Algorithms used

1) Building the string

StringManipulator.cpp: Builds the string recursively and returns to Lsystem.cpp where it is then translated into vectors.

addRule: The addRule function first takes in the axiom and rule and appends these to m_axiom and m_replaceRule.

createString: A while loop is then used to cycle through m_axiom and m_rule list. The method is run recursively until the incrementing counter is less than depth, whereby it returns the resulting string.
The string is replaced by cycling through the letters in a for loop for the length of the string. my_map is used to set the axiom to the corresponding rule in the axiom and rule lists. When read, if the string matches the axiom which is mapped to its corresponding rule, the character of that string is replaced by the rule.

addRule(_axiom : char, _axiom1 : char, _repRule : std::string, _repRule1 : std::string)
createString(_depth : int, _inputString : std::string)

2) Converting the resulting string into vectors

LSystem.cpp: Converts the final rule returned from StringManipulator.cpp to vectors

buildVectorList(_height : float, _angleIncrement : ngl::Vec3, _depth : int, _sunPos : ngl::Vec3, axiom : char, axiom1 : char, _rule : std::string, _rule1 : std::string, _smoothLevel : int, _sunIntensity : float, _wind : float, _windVec : ngl::Vec3, _frames : int, _hitCount : int)

buildVectorList: The other parameters are first altered in relation the the number of times the plant has been hit by rain.

```
{
    // Adjust the inputs relative the the amount of rain hits
    _height+=(_hitCount/100);
    _sunPos.m_y+=(_hitCount/20);
    _sunPos.m_x+=(_hitCount/150);
    _sunIntensity+=(_hitCount/20);
    _depth+=(_hitCount/80);
    _angleIncrement.m_x+=(_hitCount/10);
    _angleIncrement.m_y+=(_hitCount/10);
    _angleIncrement.m_z+=(_hitCount/10);
```

The values were set by trial and error.
The string is then loaded with the loadString function and returned to m_finalRule.
To convert this string to vectors, a for loop iterates through each letter. If “F” (meaning draw forwards), the start vector (0,1,0) is rotated using these formulae:

```
angle*=pi/180;
// Rotation in the x axis
temp = vector;
vector.m_y = temp.m_y*cos(angle.m_x) - temp.m_z*sin(angle.m_x);
vector.m_z = temp.m_y*sin(angle.m_x) + temp.m_z*cos(angle.m_x);
// Rotation in the Y axis
temp = vector;
vector.m_x = temp.m_x*cos(angle.m_y) + temp.m_z*sin(angle.m_y);
vector.m_z = -temp.m_x*sin(angle.m_y) + temp.m_z*cos(angle.m_y);
// Rotation in the Z axis
temp = vector;
vector.m_x = temp.m_x*cos(angle.m_z) - temp.m_y*sin(angle.m_z);
vector.m_y = temp.m_x*sin(angle.m_z) + temp.m_y*cos(angle.m_z);
angle*=180/pi;
```

Figure 1: Rotating the vector

“Angle” is multiplied by pi/180 (“pi” is a pre declared const float) to convert it to radians and the vector is rotated in the X, Y and Z axis.
The resulting vector is then added to the previous vector, and this vector is stored as “previous vector”.

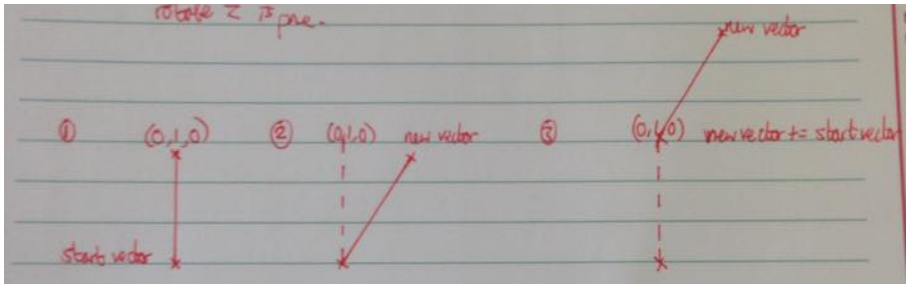


Figure 2: Adding vectors to previous vectors

If the character read is “+” or “-”, the angle vector increases/ decreases.
If “[“, the vector is pushed back onto a vector list. Then if “]” is read, this value is popped off the stack the vector is set to be equal to the previous vector in the stack, and the plant continues to iterate through the string. Here, “branch” is set to 1 and appended to the branchList. This is later used by drawPlant.cpp to specify where to clear the point drawing data (data.end()) so the branches end.

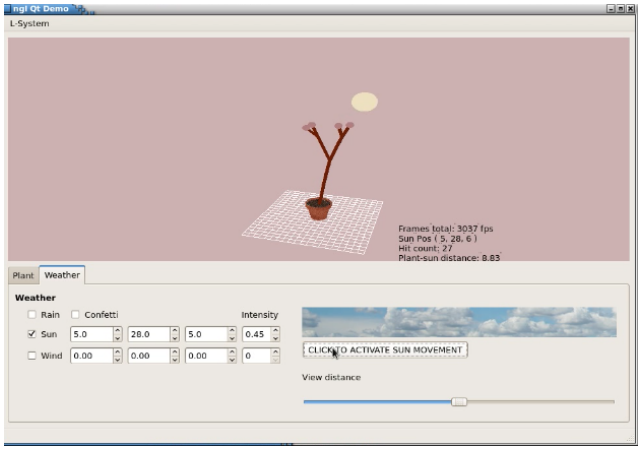
Directing the vector towards the sun

I find the direction vector which is the distance between the vector above and the sun's position.
This is normalised to give just the direction between them, and then added to the vector and multiplied by _sunIntensity. The higher the intensity, the closer the vector tends towards the sun.

```
// Finding the direction vector between tendPoint (the sun's position) and the branch vector
dir.m_x = -_sunPos.m_x - vector.m_x;
dir.m_y = -_sunPos.m_y - vector.m_y;
dir.m_z = -_sunPos.m_z - vector.m_z;

// Normalising the direction vector to find its direction
int hyp = sqrt(dir.m_x*dir.m_x + dir.m_y * dir.m_y + dir.m_z * dir.m_z);
dir.m_x /=hyp;
dir.m_y /=hyp;
dir.m_z /=hyp;

// Adding the direction vector and multiplying by sun intensity so the branch tends towards this point quickly
vector.m_x += dir.m_x * -_sunIntensity;
vector.m_z += dir.m_z * -_sunIntensity;
```



Above: The plant tends towards the sun, and tends faster if the intensity is higher.

Preventing collision between the plant and the floor or its pot

If vector.m_y<value for the floor, the vector tends towards the sun with:

// vector.m_y += dir.m_y * -_sunIntensity+1;

To prevent the branches touching the pot, I query if vector.m_x and vector.m_z are less than the pot's width, and whether y is greater than the pot's height.

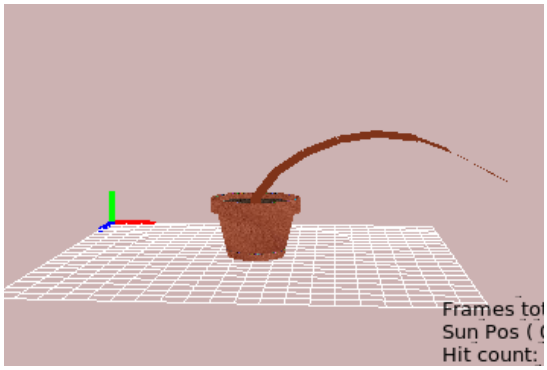


Figure 3: The plant won't go below a certain height, or collide with its pot.

Making the plant sway with wind

I used a sine function with the running frame count to make the branches "sway" by adding the resulting wind vector to the branches' "vector".

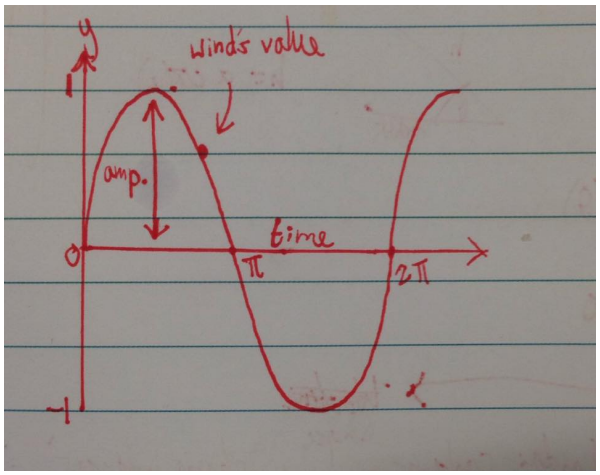


Figure 4: Diagram showing sine wave used and wind's value.
The wave used the simplified equation of:

Result = sin(time*frequency)*amplitude

Where amplitude is the intensity of the wind, time is the frame total, and frequency is a value.
// psuedo-code
wind Vector+=sin(frames_total*frequency_of_wave)*amplitude;
vector +=wind Vector;

As the program updates, the vectors oscillate as the total number of frames increment, and so the plant looks like it's swaying (figure 5).

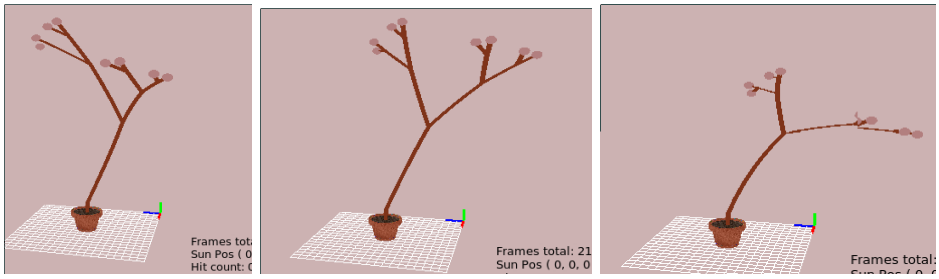


Figure 5: swaying plant

3) Drawing the plant using triangles

drawPlant.cpp: Reads the vectors and draws between the points. It also draws the sun and leaves.

buildPlant(_radius : float, _height : const float, _vectorList : std::deque <ngl::Vec3>, _branchEnd : std::deque <int>, _leafList : std::deque <ngl::Vec3>, _angleList : std::deque <ngl::Vec3>, _folliage : const int, _wind : const int)

circleData(_startPoint : ngl::Vec3, _radius : const float)

Buildplant stores the point vertex data which is later bound within the rendering paintGL() function in GLWindow.
The vectors were read from drawPlant and made into cylinders by drawing triangles between the points that formed circles. In order to draw circles, I used a for loop to cycle between the vectors for the number of segments, where segments = 360/angle, and angle was a pre determined value. I then called circleData to calculate the position the new point would be on the circle, in relation to _radius of the branch, which used pythagoras to find the x and z position of each point (figure 6).

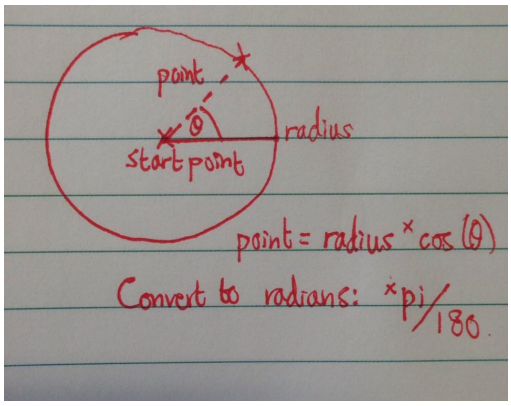


Figure 6: pythagoras to calculate the new point

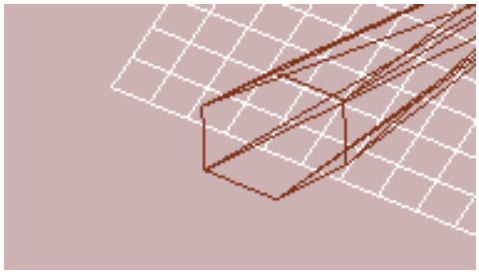


Figure 7: Number of segments and resulting circle

The result is then added to _startPoint so that the point moves the circle to the vector's position for each branch, and returned to be pushed to the vertex data.

Drawing cylinders

To draw between cylinders, as I was using triangles, I needed to draw between the points for the current circle in the list, and the previous circle as shown in figure 7:

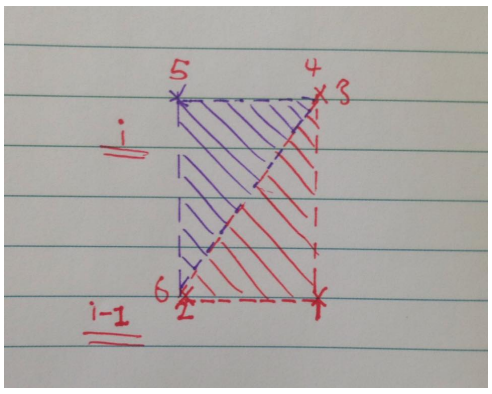


Figure 8: Triangles between points

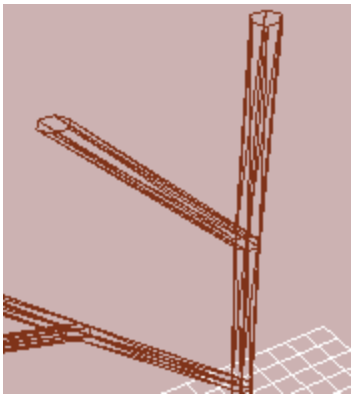


Figure 9: The drawn cylinder

I iterated through the for-loop to connect these panels together to form a cylinder. To change the radius of each cylinder so it followed smoothly into the next, the radius also had to be changed in accordance to whether it was an upper or lower point in the circle. This didn't work for branching however, so is commented out of my code, but is shown without rotation working below:

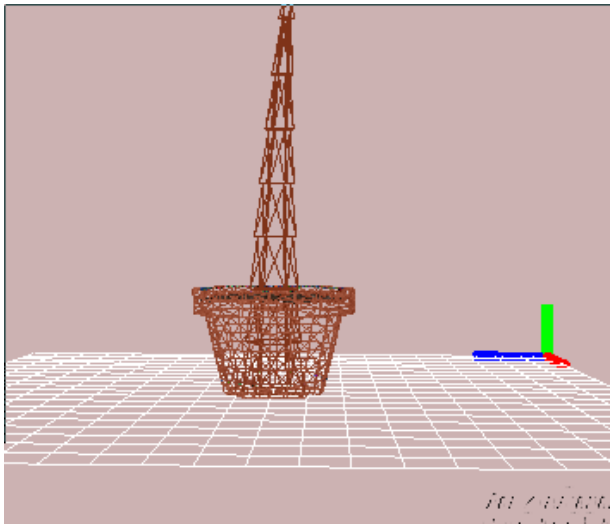


Figure 10: The radius of each branch decreasing

It worked by adjusting the radius based on whether the point was on the top or bottom of the cylinder shown below:

```
//_radius = 1-0.1*(_vectorList.size()-j);
rotateSwitch = 1;
point[2] = circleData(_vectorList[j], _radius, (angle*(i+1)), circleRotate, prevVec);
d.x = point[2].m_x;
d.y = point[2].m_y;
d.z = point[2].m_z;
data.push_back(d);

//_radius = 1-0.1*(_vectorList.size()-j+2);
```

Rotating the ends of branches

This wasn't included in the code, but could be found by first finding the angle each branch was rotated by:
vec = vector[i] - vector[i-1]
I then used the dot product to find the angle between that vector and both the Z and X axis. The points forming the circle were then rotated by these amounts, and the branch rotated with the rotating algorithm shown earlier.

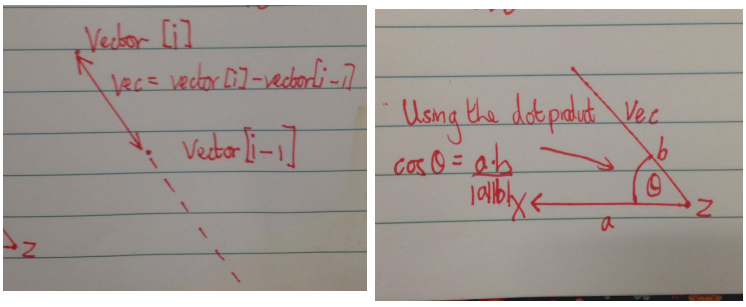


Figure 11, 12: Finding the angle between the branch and X or Z axis.

However, I encountered errors where the rotated angle “flipped”, and couldn’t fully solve the issue. For example in figure 13, the circle rotated opposite to how I wanted when the angle was >90. I needed to make the circle’s rotation angle +ve or -ve in relation to the quadratic circle.

Figure 13: Rotating the branch end

Drawing leaves

If foliage is turned on, a for loop cycles through the leaf list and draws spheres for every vector in the list. Their position also changes with the wind swaying as they use the same vectors.

4) RAIN

Weather.cpp and Particle.cpp: Weather.cpp draws the rain particles, and Particle.cpp stores the particles’ data.

Weather::Weather(_pos : ngl::Vec3, _numParticles : int)

Particle::update(_rainOnOff : bool, _confetti : bool)

Weather is initialised with a starting position and number of particles (rain droplets). “_pos.m_x” and “_pos.m_z”. are a random value between -5 and 5, and _pos.m_y is set to a predetermined value so the rain spawns from the sky. Within this function, a function to Particles is called so the positions are added to a list of m_particles, which stores the particles’ positions, colour, direction and original position.

A for loop iterates through each particle and calls the update function for each. Depending on whether confetti or rain are true/ false, the rain’s position falls and, when it hits the pot (when _pos.m_y<5), hitCount is increased or decreased depending on whether confetti or rain are enabled, and returned to weather.cpp to be read by GLWindow.cpp.

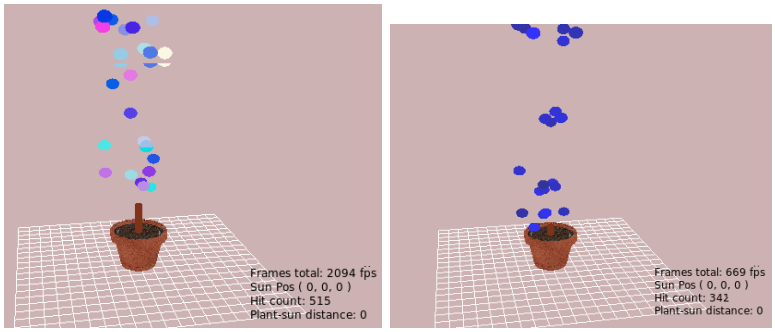
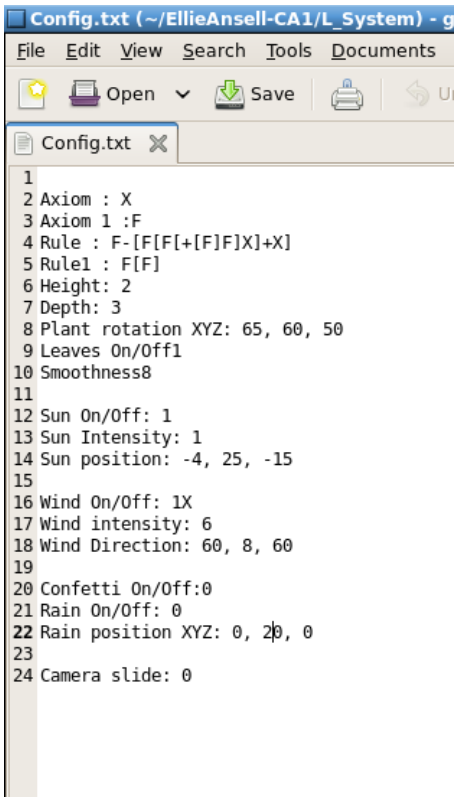


Figure 14, 15: Confetti and rain particles

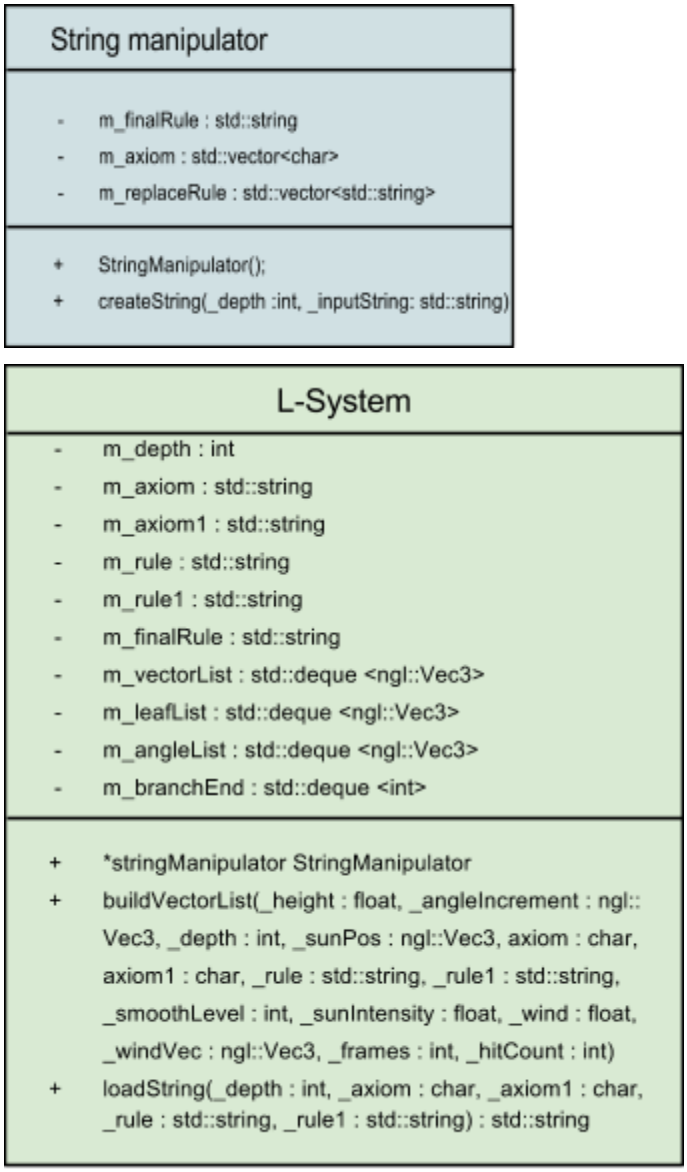
Hit count then influences other parameters, as previously described in “2) Converting the string to vectors”.

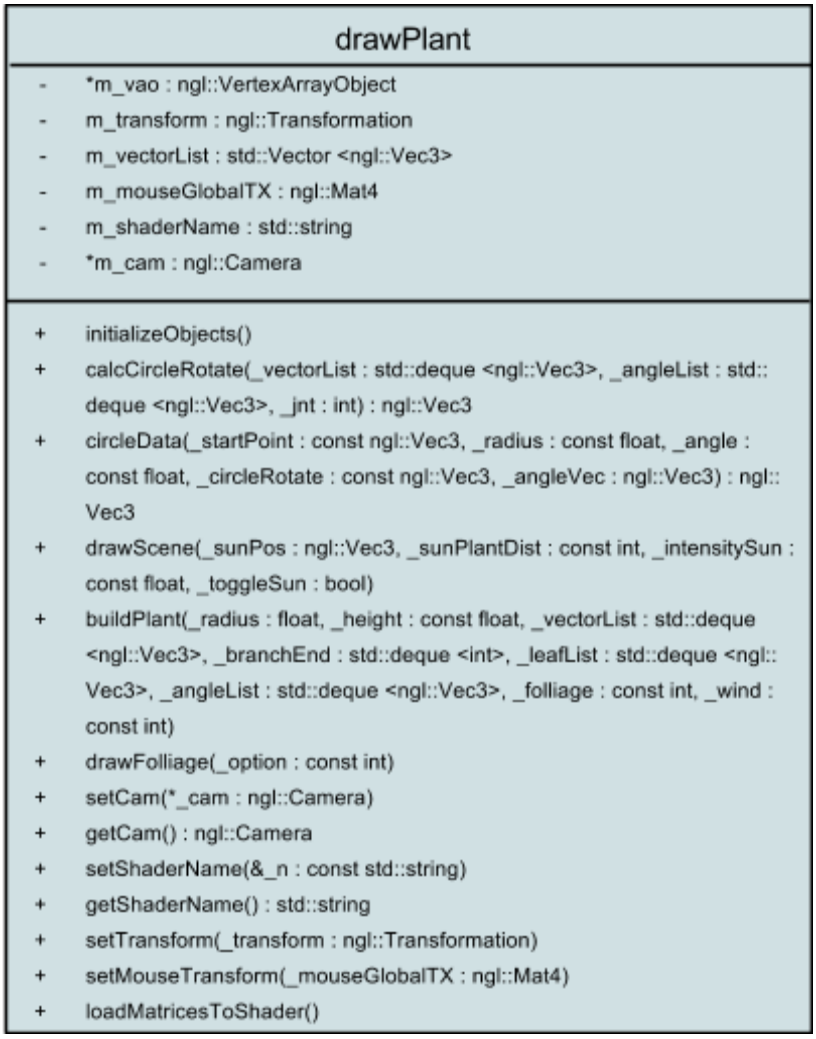
Saving to Config file

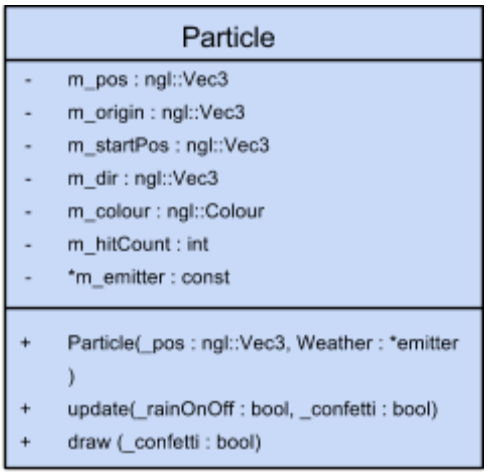
The set parameters can be saved to the config file:

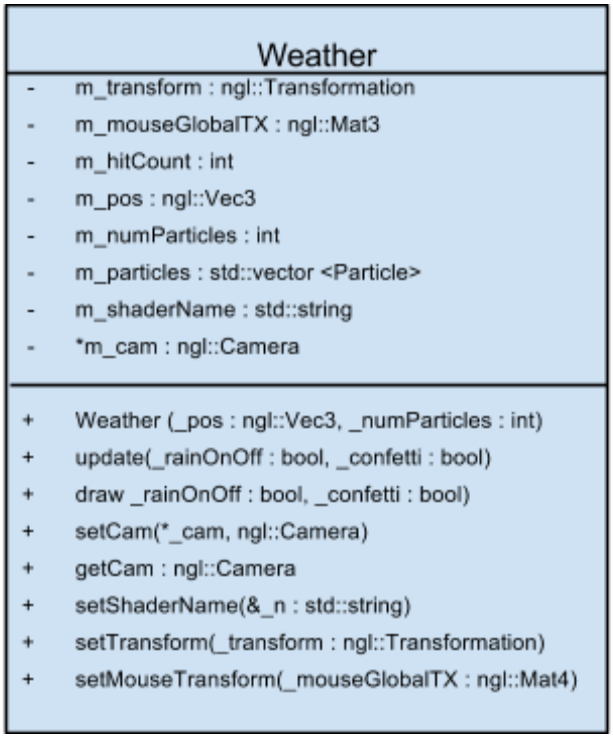


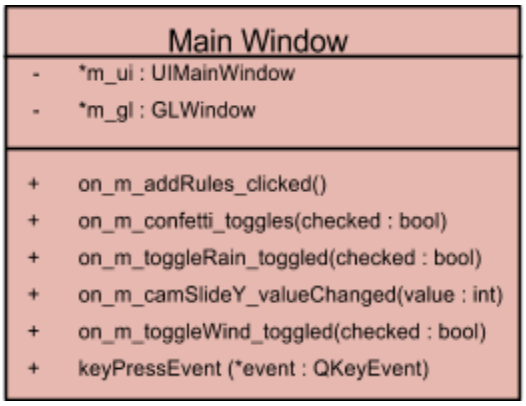
Class members and attributes shown below (they’re not connected as the diagrams are too large):


String manipulator
- m_finalRule : std::string
- m_axiom : std::vector<char>
- m_replaceRule : std::vector<std::string>
+ StringManipulator();
+ createString(_depth :int, _inputString: std::string)
L-System
- m_depth : int
- m_axiom : std::string
- m_axiom1 : std::string
- m_rule : std::string
- m_rule1 : std::string
- m_finalRule : std::string
- m_vectorList : std::deque <ngl::Vec3>
- m_leafList : std::deque <ngl::Vec3>
- m_angleList : std::deque <ngl::Vec3>
- m_branchEnd : std::deque <int>
+ *stringManipulator StringManipulator
+ buildVectorList(_height : float, _angleIncrement : ngl::Vec3, _depth : int, _sunPos : ngl::Vec3, axiom : char, axiom1 : char, _rule : std::string, _rule1 : std::string, _smoothLevel : int, _sunIntensity : float, _wind : float, _windVec : ngl::Vec3, _frames : int, _hitCount : int)
+ loadString(_depth : int, _axiom : char, _axiom1 : char, _rule : std::string, _rule1 : std::string) : std::string


drawPlant
- *m_vao : ngl::VertexArrayObject
- m_transform : ngl::Transformation
- m_vectorList : std::Vector <ngl::Vec3>
- m_mouseGlobalTX : ngl::Mat4
- m_shaderName : std::string
- *m_cam : ngl::Camera
+ initializeObjects()
+ calcCircleRotate(_vectorList : std::deque <ngl::Vec3>, _angleList : std::deque <ngl::Vec3>, _jnt : int) : ngl::Vec3
+ circleData(_startPoint : const ngl::Vec3, _radius : const float, _angle : const float, _circleRotate : const ngl::Vec3, _angleVec : ngl::Vec3) : ngl::Vec3
+ drawScene(_sunPos : ngl::Vec3, _sunPlantDist : const int, _intensitySun : const float, _toggleSun : bool)
+ buildPlant(_radius : float, _height : const float, _vectorList : std::deque <ngl::Vec3>, _branchEnd : std::deque <int>, _leafList : std::deque <ngl::Vec3>, _angleList : std::deque <ngl::Vec3>, _folliage : const int, _wind : const int)
+ drawFolliage(_option : const int)
+ setCam(*_cam : ngl::Camera)
+ getCam() : ngl::Camera
+ setShaderName(&_n : const std::string)
+ getShaderName() : std::string
+ setTransform(_transform : ngl::Transformation)
+ setMouseTransform(_mouseGlobalTX : ngl::Mat4)
+ loadMatricesToShader()


Particle
- m_pos : ngl::Vec3
- m_origin : ngl::Vec3
- m_startPos : ngl::Vec3
- m_dir : ngl::Vec3
- m_colour : ngl::Colour
- m_hitCount : int
- *m_emitter : const
+ Particle(_pos : ngl::Vec3, Weather : *emitter)
+ update(_rainOnOff : bool, _confetti : bool)
+ draw (_confetti : bool)


Weather
- m_transform : ngl::Transformation
- m_mouseGlobalTX : ngl::Mat3
- m_hitCount : int
- m_pos : ngl::Vec3
- m_numParticles : int
- m_particles : std::vector <Particle>
- m_shaderName : std::string
- *m_cam : ngl::Camera
+ Weather (_pos : ngl::Vec3, _numParticles : int)
+ update(_rainOnOff : bool, _confetti : bool)
+ draw _rainOnOff : bool, _confetti : bool)
+ setCam(*_cam, ngl::Camera)
+ getCam : ngl::Camera
+ setShaderName(&_n : std::string)
+ setTransform(_transform : ngl::Transformation)
+ setMouseTransform(_mouseGlobalTX : ngl::Mat4)

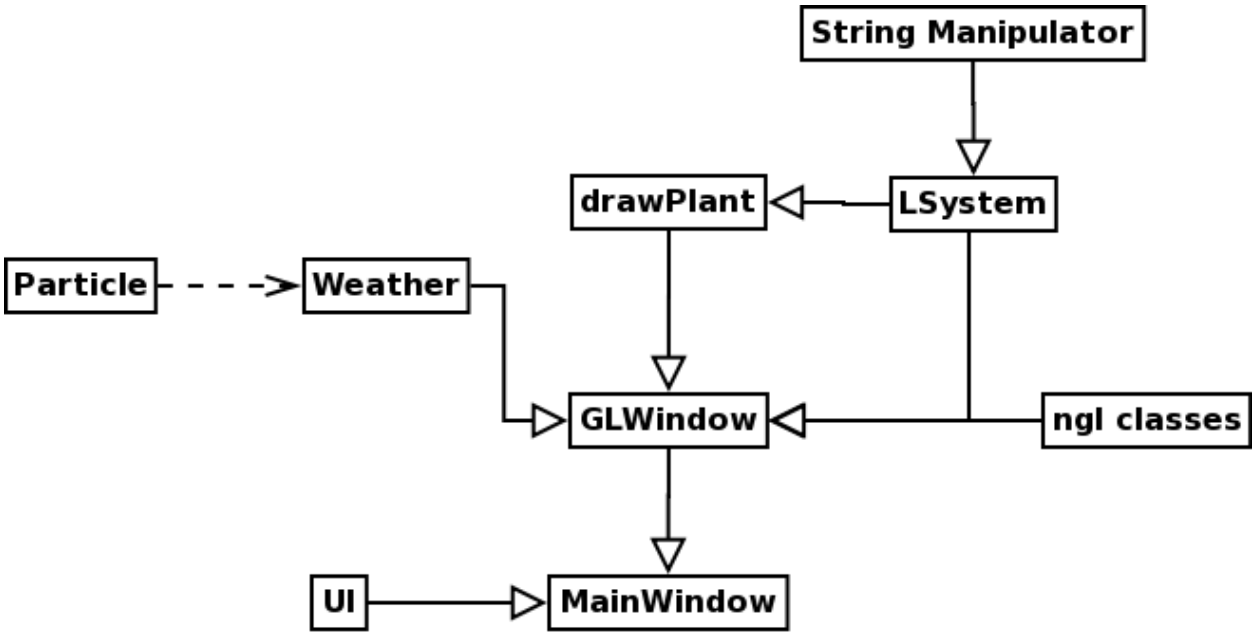

Main Window
- *m_ui : UIMainWindow
- *m_gl : GLWindow
+ on_m_addRules_clicked()
+ on_m_confetti_toggles(checked : bool)
+ on_m_toggleRain_toggled(checked : bool)
+ on_m_camSlideY_valueChanged(value : int)
+ on_m_toggleWind_toggled(checked : bool)
+ keyPressEvent (*event : QKeyEvent)

GLWindow

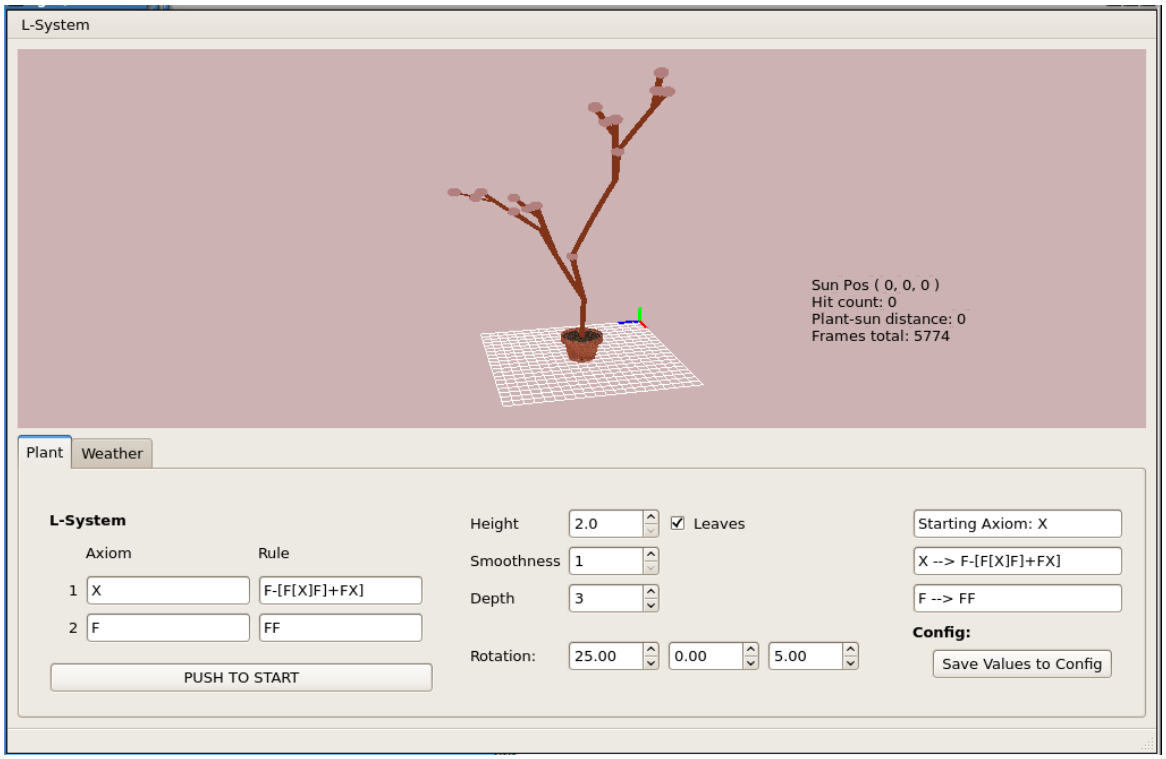
- m_sunPos : ngl::Vec3
- m_rainPos : ngl::Vec3
- m_windVew : ngl::Vec3
- m_camSlideY : float
- m_boolWindOnOff : bool
- m_angleIncrement : ngl::Vec3
- m_scalePlant : float
- m_wind : float
- m_radius : float
- m_height : float
- m_depth : int
- m_hitCount : int
- m_wireFrame : bool
- m_sunOnOff : bool
- m_rainOnOff : bool
- m_confetti : bool
- m_axiom : char
- m_axiom1 : char
- m_smoothLevel : int
- m_rule : std::string
- m_rule1 : std::string
- m_intensitySun : float
- *m_test : ngl::text
- *m_pot : ngl::obj
- *m_soil : ngl::obj
- m_vectorList : std::deque <ngl::Vec3>
- m_leafList : std::deque <ngl::Vec3>
- m_branchEnd : std::deque <int>
- m_angleList : std::deque <ngl::Vec3>
- m_startVector : ngl::Vec3
- *cam : ngl::Camera
- m_transform : ngl::Transformation
- *m_light : ngl::Light
- m_spinXFace : int
- m_spinYFace : int
- m_origX : int
- m_origY : int
- m_origXPos : int
- m_origYPos : int
- m_translate : bool
- m_rotate : bool
- m_modelPos : ngl::Vec3
- m_mouseGlobalTX : ngl::Mat4
- m_scale : ngl::Vec3
- m_position : ngl::Vec3

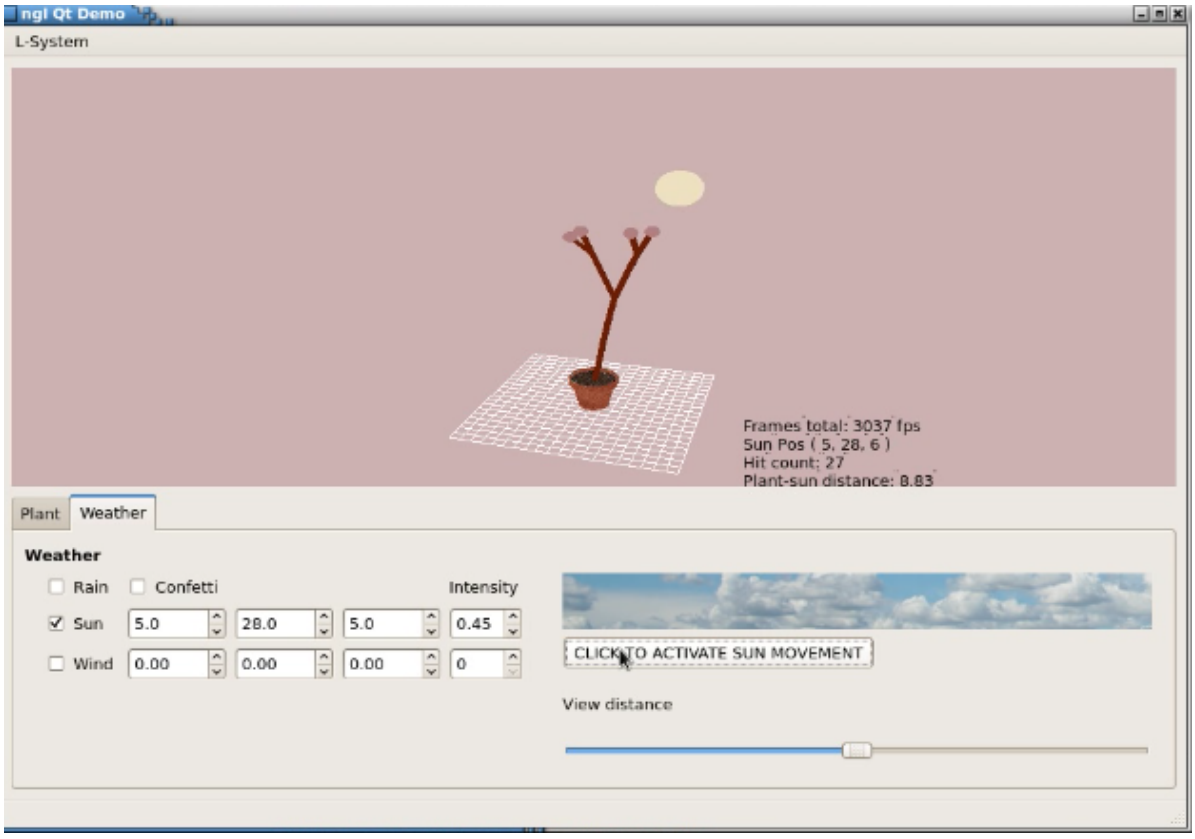
+ GLWindow(_format : const QGLFormat, *_parent : QWidget)
+ GLWindor()
+ setDepth(_i : int)
+ smoothLevel(_i : int)
+ sunPosX(_x : double)
+ sunPosY(_y : double)
+ sunPozZ(_z : double)
+ rainPosX(_x : double)
+ rainPosY(_y : double)
+ rainPosZ(_z : double)
+ sunIntensity(_i : double)
+ windIntensity(_i : double)
+ windDirX(_x : double)
+ windDirY(_y : double)
+ windDirZ(_z : double)
+ toggleSun(_mode : bool)
+ toggleWind(_mode : bool)
+ toggleLeaf(_mode : bool)
+ toggleConfetti(_mode : bool)
+ angleIncrementX(_x : double)
+ angleIncrementY(_y : double)
+ angleIncrementZ(_z : double)
+ plantHeight(_i : double)
+ camSlideY(_y : int)
+ getAxiomString(_axiomStr : char)
+ getAxiom2String(_axiom2Str : char)
+ getRuleString(_ruleStr : std::string)
+ getRule2String(_rule2Str : std::string)
+ initializeGL()
+ resizeGL(_w : const int, _h : const int)
+ paintLG()
+ mouseMoveEvent(*_event : QMouseEvent)
+ mousePressEvent(*_event : QMouseEvent)
+ mouseReleaseEvent(*_event : QMouseEvent)
+ timerEvent(QTimerEvent)
+ loadMatricesToShader()

Class diagram:



Main window plant and weather tabs screenshots:




ngl Qt Demo
L-System
Frames total: 3037 fps
Sun Pos (5, 28, 6)
Hit count: 27
Plant-sun distance: 8.83
Plant
Weather
Weather
Rain
Confetti
Intensity
Sun
5.0
28.0
5.0
0.45
Wind
0.00
0.00
0.00
0
CLICK TO ACTIVATE SUN MOVEMENT
View distance