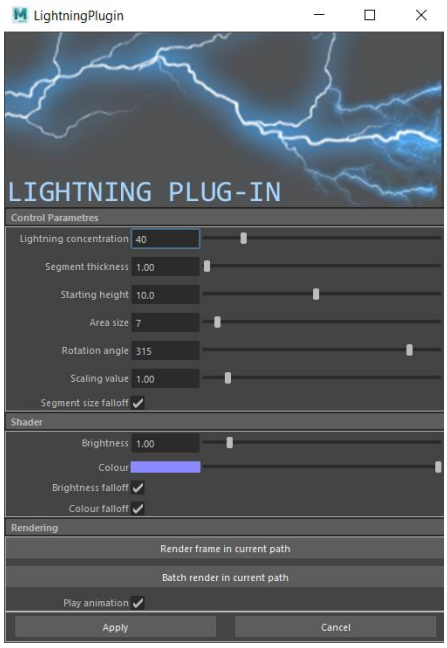
PROCEDURAL LIGHTNING USING THE SPACE COLONIZATION ALGORITHM

User Manual



The user should first make sure the file path for the GUI picture is the correct one, and, if necessary, change it manually (line 251). Running the script will create the user interface, and the 'Apply' button will generate the lightning. The list underneath illustrates each parameters' use:

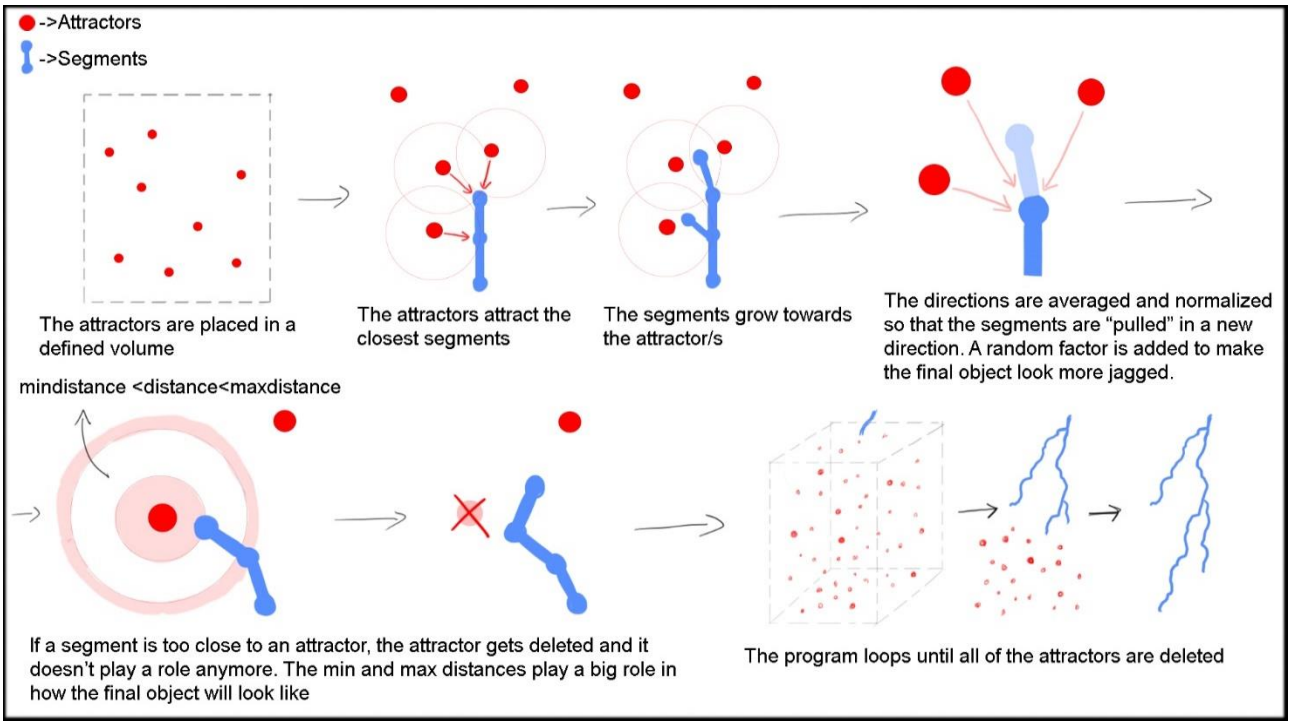
Lightning concentration: it controls how 'dense' the lightning will be (min=10, max=200)
Segment thickness: it controls the radius of each lightning segment (min=1, max=5)
Starting height: where the lightning starts generating (min=1, max=20)
Area size: the area of influence of the bolt generation (min=5, max=50)
Rotation angle: the angle for wich the lightning is rotated. It's pointing downwards at zero degrees (min=0, max=360)
Scaling value: the value for wich the lightning is scaled (min=0.1, max=10)
Segment size falloff: checkbox to control if the radius of the segments should get smaller as the lightning grows

Brightness: it controls the brightness of the lightning (min=0, max=10)
Colour: it controls the colour of the lightning
Brightness falloff: checkbox to control if the brightness of the segments should get smaller as the lightning grows
Colour falloff: checkbox to control if the colour value of the segments should get smaller as the lightning grows

Render frame in current path: renders the current frame
Batch render in current path: batch renders
Play animation: checkbox to control if the animation would play upon the bolt creation

Algorithm Explanation

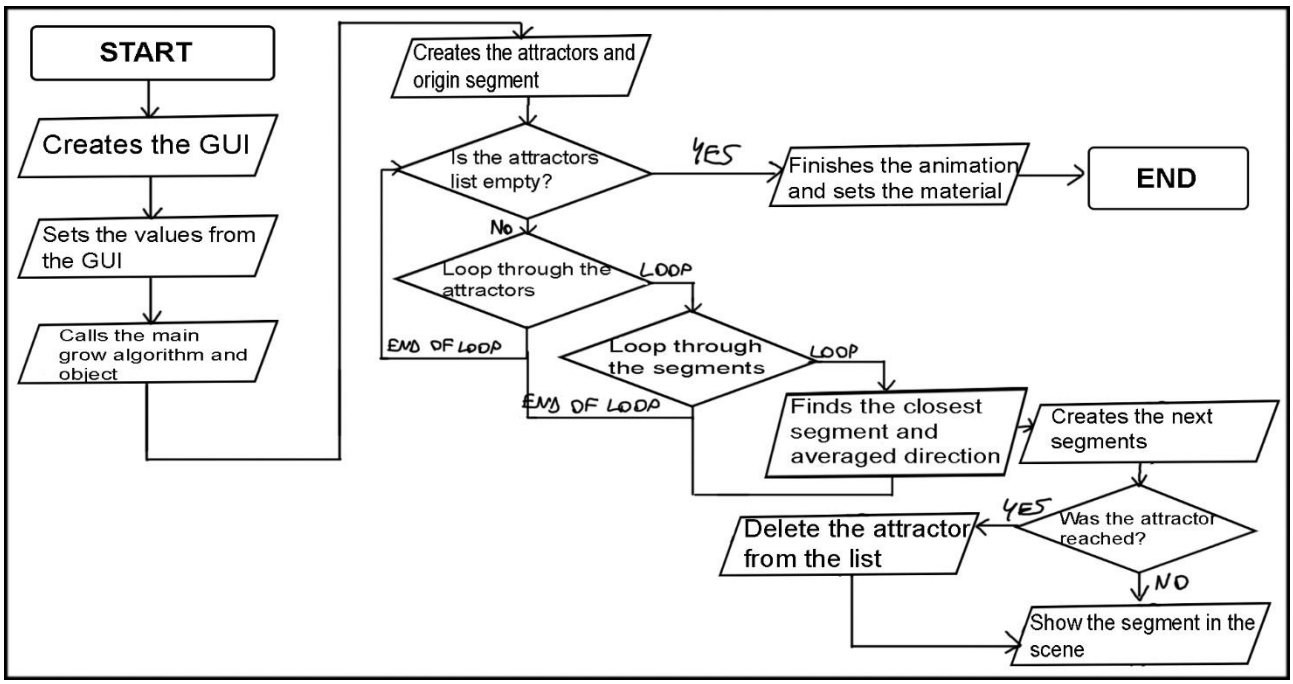
The Space Colonization Algorithm models the growth of branching structures: it was utilized here to create lightning. The algorithm uses two fundamental elements, attractors and segments. The attractors are placed in a volume. A loop through the attractors begins and the closest segment to each attractor is found. A segment is influenced when it is between the minimum and maximum distance from the attractor. Since multiple attractors can influence a single segment, for each segment the average direction towards the attractors influencing it is calculated and normalized. The next segment is placed at the designated position. If a segment is too close to an attractor, the latter will be deleted. The program loops until all the attractors are removed (referenced from *Modelling Trees with a Space Colonization Algorithm* (2007)).



The level of complexity is high: the algorithm requires a nested for loop to function, together with many if statements and while loops. The great amount of customization for the user also creates many differing variables. Thankfully the use of object-oriented programming and comments help make the code more readable. The code is also innovative, as the main algorithm was originally meant for the creation of procedural trees.

Scripting Structure

Running the script will create the user interface via the createUI() function. Pressing the apply button will call actionProc(), which sets the values from the GUI and calls the main grow() function.
The three main objects are attractors (instances of the Point class), segments (instances of the Line class with its methods) and the lightning itself (instance of the main Bolt class). The latter contains lists of attractors and segments and the heart of the program, the grow() method, which creates the attractors and the origin, and then loops through the attractors and segments and determines which ones are to be attracted, based on which segment is closest. A new position is calculated by averaging the directions of the segments (referenced from *Coding Challenge #17: Fractal Trees – Space Colonization* (2016), from 30:23 to 31:20). The attractors that are reached by a segment get deleted. Within grow() the methods of the Line class are utilized: resetFunc() resets the count used for the averaging and the next() creates the new segments to be added. The showMesh() functions is instead used to visualize the lightning with small cylinders and it creates the animation together with unshowMesh(). Finally, setMaterial() creates nodes for the surface shader, giving the segments colour.



Error checking:

```
56      radius = (0.1 - (radiusDecrease))
57      if (radius <= 0):
58          radius = 0.0001
```

-> If the radius of a segment is less/equal to zero it gets set to a positive number

```
183     '''This part groups all the segments together'''
184     for i in range(len(self.segmList)):
185         if (self.segmList[i].father is not None):
186             cmds.select("segment_"+str(i), add=True)
187     cmds.group(name='Lightning')
188
```

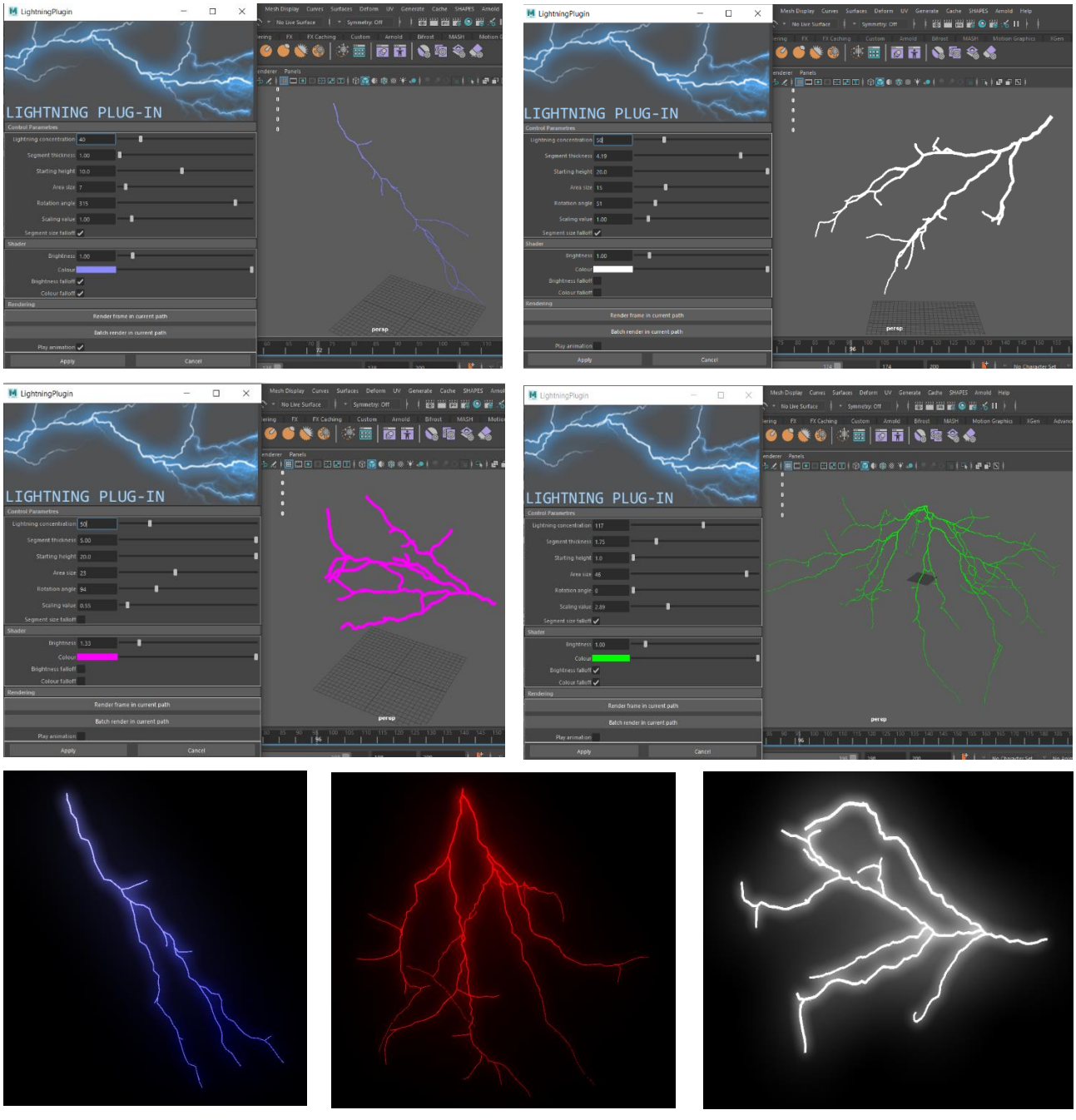
-> The code doesn't allow external objects to be grouped with the lightning

```
250 imgFileName = "H:\PYTHON\lightningProjectFINAL\Docs\Lightning_GUI2.tif"
251 if cmds.file(imgFileName, query=True, exists=True):
252     cmds.image(image=imgFileName, width=500, height=200)
253 else:
254     print("picture doesn't exist or path is not correct")
255
```

-> If the file path for the GUI image is incorrect an error message is printed

Results

The name of the lightning or the segments should not be changed, as that could cause multiple objects having the same name if another bolt was generated. Also, the rendering should be done with Maya Software.



References:

-Runions, A., Lane, B., Prusinkiewicz, P. (2007) *Modelling Trees with a Space Colonization Algorithm*. Canada: University of Calgary

-Webb, j. (2020) *Modelling organic branches structures with the space colonization algorithm and Javascript.* Medium.

-(2019) *Generating a 3D growing tree using a space colonization algorithm.* Ciphrd.

-(2014) *Procedurally Generated Trees with Space Colonization Algorithm in XNA C#.* jgallant.

-The Coding Train (2016) *Coding Challenge #17: Fractal Trees – Space Colonization.* YouTube.

-The Coding Train (2016) *Coding Challenge #18: 3D Fractal Trees.* YouTube.