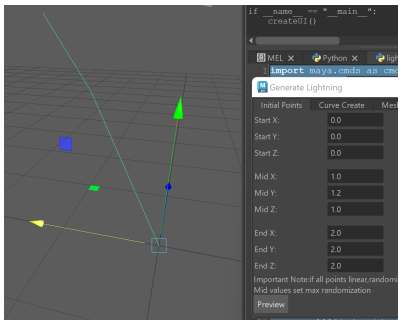
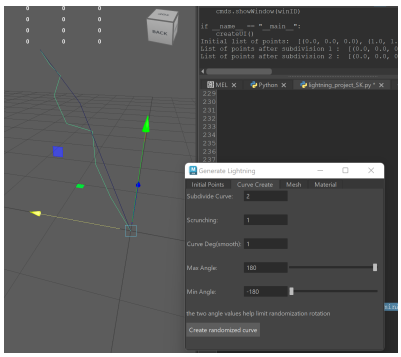
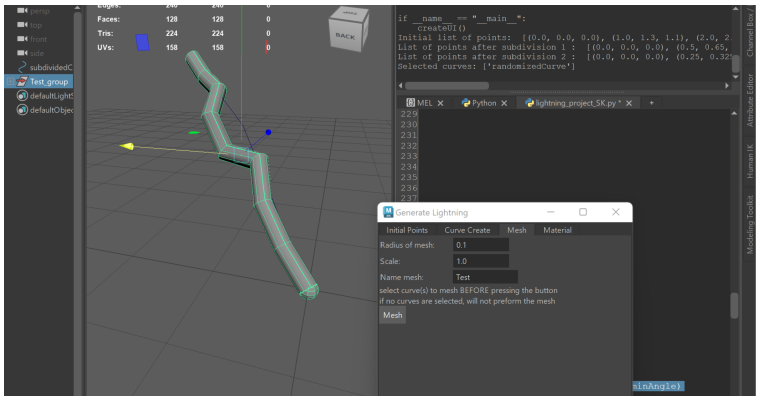
Controlled Lightning Bolt

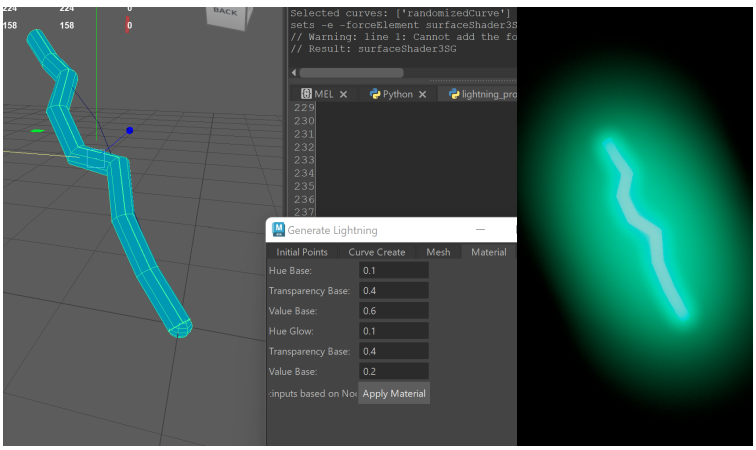
User Manual:





Four different tabs perform various actions. Initial Points allow you to see the start, mid, and endpoint with a curve having those points in there. Once satisfied with the coordinates moving to the tab curve, it controls randomisation based on inputs. However, each time it's generated, one can create any number of bolts streaming out. The next tab mesh takes the selected curve(s) and creates a simple mesh. The last tab, Material, makes and assigns a surface shader. There are initial values if a user wants to create a small bolt quickly. One can use a curve of their selection and generate their mesh. Similarly, the Material is applied to any selected mesh, even if it’s not a generated curve from this tool, and it’s render through Maya Software to get the glow effect.





Working Script Logic

Through my research, I wanted to prioritise control over how the lightning bolt, things done randomly, such as the clamp method or random displacement, were not giving the level of control I wanted to explore. I was limited to a bolt going from top to bottom. In my method, the “midPoint” serves as a way to control the level of randomisation along with other supporting variables while using Rodrigue’s Formula to rotate these points at their respective levels. To manage these variables, if some were not touched, like naming is assigned a default name if the user does not put one in or if there was a limit on a value it has, such as zero being divided, these things are controlled/ corrected.

```
138
139         if meshName:  # rename if one is picked
140             tubular_mesh = cmds.rename(mesh[0], meshName + str(curves.i
141         else:
142             tubular_mesh = cmds.rename(mesh[0], "mesh" + str(curves.inde
```

```
239     startPoint = (x1,y1,z1)
240     midPoint = (x2,y2,z2)
241     endPoint = (x3,y3,z3)
242     subdivisions = cmds.intFieldGrp(subdivi, query=True, value1=True)
243     if (subdivisions<0):
244         subdivisions=0
245     breakingDistance = cmds.intFieldGrp(breakingDis, query=True, value1=True)
246     if (breakingDistance==0):
247         breakingDistance=1#prevents division issues
248     curveDeg = cmds.intFieldGrp(cDeg, query=True, value1=True)
249     if (curveDeg<=0):
```

```
251     maxAngle = cmds.intSliderGrp(maxA, query=True, value=True)
252     minAngle = cmds.intSliderGrp(minA, query=True, value=True)
253     generator = LightningGenerator(startPoint, midPoint, endPoint)
```

However, there isn't a similar protection for the surface shader as the user would need to know what those values meant and go into the hyper-shade window and alter it there. Yet the button for it ensures there is a mesh to apply to similarly when creating a mesh.

```
def apply_glow(meshes,out1, out2, out3,glow1, glow2, glow3):
    '''surface shader created
    meshes:meshes selected to apply to
    out1:hue base
    out2:saturation base
    out3:value of base
    glow1:hue of glow
    glow2:saturation of glow
    glow 3:value/strength of glow
    return : with maya software renders out glowing mesh
    '''
    surfaceShader = cmds.shadingNode('surfaceShader', asShader=True)
    surfaceShaderSG = cmds.sets(renderable=True, noSurfaceShader=True, empty
    cmds.connectAttr(surfaceShader+'.outColor', surfaceShaderSG+'.surfaceSha

    cmds.setAttr(surfaceShader+'.outColor', out1, out2, out3, type='double3'
    cmds.setAttr(surfaceShader+'.outGlowColor', glow1, glow2, glow3, type='d

    for mesh in meshes:
        cmds.select(mesh)
        cmds.hyperShade(assign=surfaceShaderSG)
```

```
def button4Clicked(o1, o2, o3,g1, g2, g3):
    '''altering and assigning the surfaceShader to have mesh glow
    (implicit) mesh(s):lighting branches
    o1:base hue
    o2:base saturation
    o3:base value
    g1:glow hue
    g2:glow saturation
    g3:glow value/brightness
    return : 3 point curve
    '''
    meshes = cmds.ls(selection=True)
    if not meshes:
        cmds.error("No mesh have been selected. Please select mesh(s) before
        return
    out1 = cmds.floatFieldGrp(o1, query=True, value1=True)
    out2 = cmds.floatFieldGrp(o2, query=True, value1=True)
    out3 = cmds.floatFieldGrp(o3, query=True, value1=True)

    glow1 = cmds.floatFieldGrp(g1, query=True, value1=True)
    glow2 = cmds.floatFieldGrp(g2, query=True, value1=True)+0.1
    glow3 = cmds.floatFieldGrp(g3, query=True, value1=True)+0.1

    apply_glow(meshes,out1, out2, out3,glow1, glow2, glow3)
```

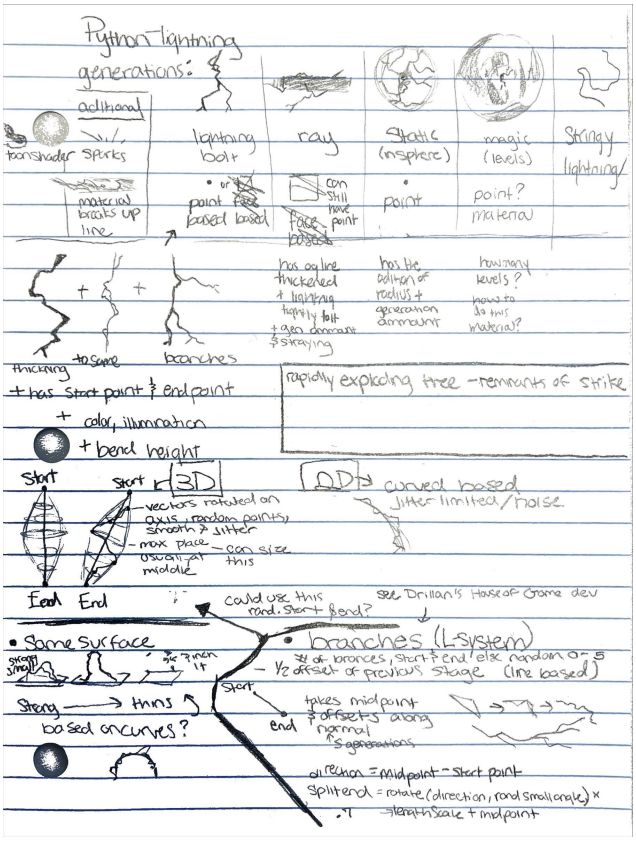
```
258 def button3Clicked(radius, expand,stringField):
259     '''meshing the selected curves
260     (implicit)curve(s):what to follow in the extrusion
261     radius:radius of the circle
262     expand:increasing the size, similar to radius but dosent change en
263     stringField:optional meshing name
264     return:mesh(s) of the curves that were selected
265     '''
266     curves = cmds.ls(selection=True)
267     print("Selected curves:", curves)
268     if not curves:
269         cmds.error("No curves have been selected. Please select curves
270         return
271     radCircle = cmds.floatFieldGrp(radius, query=True, value1=True)
272     scaling = cmds.floatFieldGrp(expand, query=True, value1=True)
273     meshName = cmds.textField(stringField, query=True, text=True)
274     meshes = create_tubular_mesh(curves, radCircle, scaling, meshName)
```

Two functions I want to discuss are curve_lightning_main and create_tubular_mesh. The first uses Rodrigue's formula, enabling me to "rotate" around the line between the start and endpoints. By manipulating the max and min angle, you can control the dimension of the "random" generation in addition to the maximum values set by the midpoint. Breaking down how to find the radius depending on where the iteration point allows for the control that the midpoint has. The axis is a means to an end to create the circle of potential new coordinates. This function is the meat of what I wanted because you can also use the angle to control its generation even more, as a bolt running along a surface isn't the same as a top-to-bottom bolt, which was one of the problems I wanted to solve. If I had to expand this, I'd implement Lichtenberg or Stochastic l-system. Create_tubular_mesh was my way of turning the curve/point-based generation into a visible mesh. I didn't like the idea of separate cylinders and worked with extrusion instead. Another way I could expand my code is to give users a choice of what shape to base the mesh on. Overall, I would also like to clean up the code of the GUI to something more readable than the wall of text it currently is.

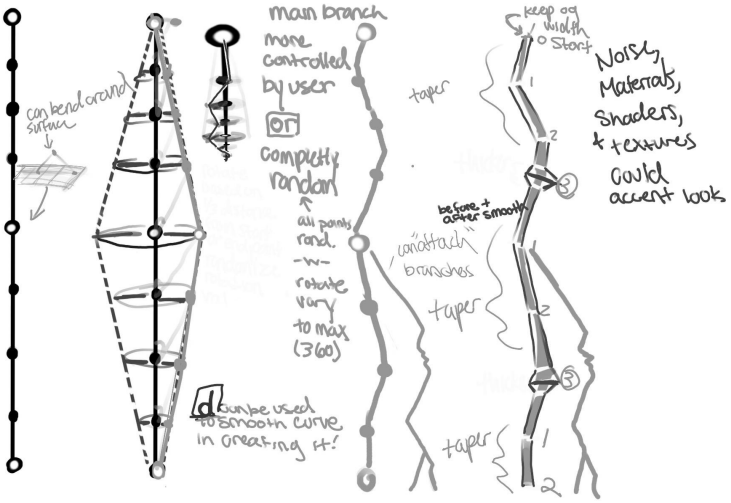
```
    def curve_lightning_main(self, curvePoints, startPoint, endPoint, breakingDistance, maxAngle, minAngle):
        '''applies rodrigue's formula and subdivision of curve to create a controlled randomized curve
        curvePoints:subdivided curve
        startPoint:start of curve
        endPoint:end of curve
        breakingDistance:ability to manipulate subdivision spread
        maxAngle:one extreme limit for angle rotate
        minAngle:other extreme limit for angle rotate
        return : altered points
        '''
        curveLightningPoints = []
        for i in range(len(curvePoints)):
            curvePoint = curvePoints[i]
            if self.get_distance(curvePoint, startPoint) < self.get_distance(curvePoint, endPoint):
                # start point closer
                rotationRadius = self.get_distance(curvePoint, startPoint) / breakingDistance
            else:
                # end point closer
                rotationRadius = self.get_distance(curvePoint, endPoint) / breakingDistance
            angle = random.uniform(minAngle, maxAngle)
            axis = self.get_axis(startPoint, endPoint)
            rotatedPoint = self.rotate_point(curvePoint, axis, angle)
            curveLightningPoints.append(rotatedPoint)
        return curveLightningPoints

```

```
def create_tubular_mesh(curves, radius, expand, meshName):
    '''turns curve into mesh
    curve:any curve desired
    radius:radius of the tube to be generated around the curve
    expand:scaling
    return : generates meshgroup and returns that grouping
    '''
    meshGroup = []

    for current in curves:
        circleEx = cmds.circle(n="circle", r=radius, ch=False)

        offset = cmds.pointPosition(current + ".cv[0]")
        cmds.move(offset[0], offset[1], offset[2], circleEx, absolute=True)

        mesh = cmds.extrude(
            circleEx, current, ch=False, rn=False, polygon=3, extrudeType=2, fixedPath=1, useProfileNormal=1, scale=expand
        )

        if meshName:  # rename if one is picked
            tubular_mesh = cmds.rename(mesh[0], meshName + str(curves.index(current) + 1))
        else:
            tubular_mesh = cmds.rename(mesh[0], "mesh" + str(curves.index(current) + 1))
        # Create spheres at the start and end points
        start_sphere = cmds.polySphere(r=radius, sx=8, sy=8)[0]
        end_sphere = cmds.polySphere(r=radius, sx=8, sy=8)[0]

        start_position = cmds.pointPosition(current + ".cv[0]")
        end_position = cmds.pointPosition(current + ".cv[-1]")

        cmds.move(start_position[0], start_position[1], start_position[2], start_sphere, absolute=True)
        cmds.move(end_position[0], end_position[1], end_position[2], end_sphere, absolute=True)

        # Group the mesh, spheres, and curve
        group = cmds.group([tubular_mesh, start_sphere, end_sphere, current], name=meshName + "_group")
        meshGroup.append(group)
        cmds.delete(circleEx)

    # Return the group containing all the mesh elements
    return meshGroup

```

Design Process/Notes



From Arcane Animation Test



Lightning (gsu.edu)^^^Lightning irl
https://www.sciencelearn.org.nz/resources/239-lightning-explained

-Rapidly exploring random tree
https://gamedev.stackexchange.com/questions/71397/how-can-i-generate-a-lightning-bolt-effect

-Vectors rotated on axis, limits arcs from becoming too disorenting
https://devforum.roblox.com/t/electric-arc-demo-with-rbxls/35433
-Normals split jagged
Drilian's House of Game Development » Blog Archive » Lightning Bolts

python - YouTube

https://www.grid.uns.ac.rs/jged/download/v7n2/jged_v7_n2_p3.pdf

lapointe_stiert.pdf (rpi.edu)

OpenGL Rotation About Arbitrary Axis (songho.ca)

https://andrewtubelli.com/portfolio/curves-to-geometry/

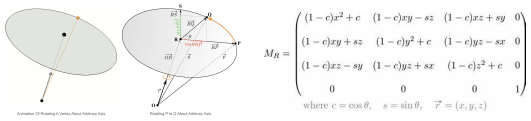
Spatial Colonization Algorithm
document (pnu.edu)
-Simplistic half split midpoint offset
https://devforum.roblox.com/t/how-to-properly-generate-lightning/140795/6

-Clamp method, highly limited
http://www.diva-portal.org/smash/get/diva2:1061502/FULLTEXT02
-Displace random points of line method
https://gamedevelopment.tutsplus.com/tutorials/how-to-generate-shockingly-good-2d-lightning-effects--gamedev-2681

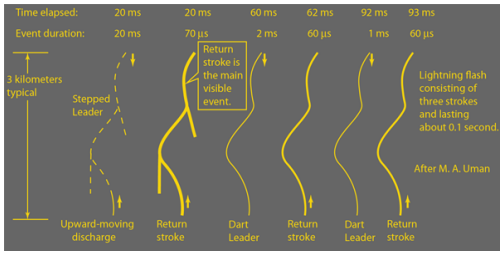
-Input ideas for GUI
https://github.com/CreativeTools/ct-lightning-generator

-Concept effect in NUKE, glitch effect, blurring, breakdown
https://erwanleroy.com/making-3d-lightning-in-nuke-using-blinkscript/
-Sparks generation on contact

https://devforum.roblox.com/t/effect-v11-lightning-beams-seamless-smooth-and-procedurally-animated-with-beam-like-properties/843534
Electric Arcs Effect - Resources / Community Resources - DevForum | Roblox
https://www.google.co.uk/url?sa=t&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DjhJvo-678Ow&psig=AOvVaw1rchrelRDBRy-9oe1id5UR&ust=1679135536917000&source=images&cd=vfe&ved=0CA8QjhxqFwoTCOjWgLvbi-0CFQAAAAAdAAAAABAE
PYTHON / Poly Cutting and Splitting | Maya Spiral



$$M_R = \begin{pmatrix} (1-c)x^2+c & (1-c)xy-sz & (1-c)xz+sy & 0 \\ (1-c)xy+sz & (1-c)y^2+c & (1-c)yz-sx & 0 \\ (1-c)xz-sy & (1-c)yz+sx & (1-c)z^2+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
where $c = \cos\theta, \quad s = \sin\theta, \quad \vec{r} = (x, y, z)$



Draft 1:MAIN BRANCH(underlined potential UI values)
inspired by https://devforum.roblox.com/t/electric-arc-demo-with-rbxls/35433
Have a curve made with 3 points

1. Iterate using midpoint and then midpoint to end or start (whatever is closer) to determine distance away from line segment
   * Adjustable break point of when to use start and end so that the largest exaggeration of the lightning can be adjusted in UI.

   * Always start upright, calculate distance from end to start, even if slanted and model that way instead of complicating it. SO if x & z value = on both start & end, then perform distance formula and save temp start. Once the operation on the main branch is complete, rotate to match the slope of the original end and start and translate.
2. Using the distance from the start or end(whichever is less), divide (test first by 3) and use to rotate around as a radius
3. Randomize based on max values, connect
4. Insert loops near the connection and make it thin out (the adjustable value is the exaggeration)
5. Smooth, then put noise(cookbook) or vice versa (noise adjustable)

MINOR BRANCHES
Inspired by Drilian's House of Game Development » Blog Archive » Lightning Bolts
^^^Broken down steps, try to follow as best as possible, end and start could be based on normals, or the endpoint of main branch going in that direction and distance from the start point

6. Select all and merge
7. Apply colour, glow amount, blur, etc for basic material, maybe have a toon shader from the cookbook
8. Make material/texture jagged magic and apply too

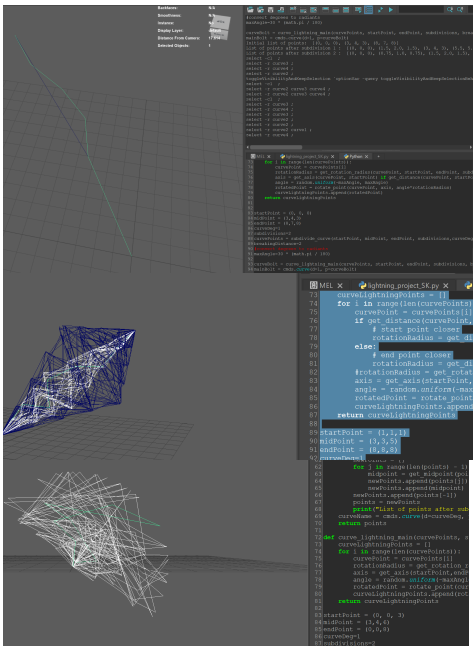
Draft 4:

Main Branch
User input:
- start, mid, and end point
-divisions(needs to be odd)
-rotation degree max(applies positively or negatively)
-exaggeration (true or false)
-thickness(suggested 0.31)

1. Generate lightning based on 3 points, the start point, the mid point, and end point, will be generating a curve
2. Iterate through till the number of divisions if 1(no iterations), odd numbers (if 3 then start to mid is another point and mid to end is and additional point, if 5 then between start and mid two additional points are added between them evenly spaced and between mid and end is two more additional points are added between them evenly space, continuing)
3. Determining the distance between each point to the mid and divide by 3 to get the radius. Rotate the points on each axis(either startpoint if before midpoint, if midpoint or after use endpoint for axsis) randomize rotation within a user controlled max value (applies positively and negatively) for the angle
4. If exaggeration is true(user choice) Every 2 points is a thickening area instead of a thinning so a point on either side about ¼ the distance from the previous and the next should be placed as well
5. Generate a mesh that is based on the curve and thickness, if exaggeration is true every 2 points is a thinking area else its thinning from the original thickness
6. Smooth

Minor Branches adapt: https://web.archive.org/web/20190816102613/http://drilian.com/2009/02/25/lightning-bolts/

Planning out how I want to organise my bolt into words, starting to break it down



'ailed methods' functions that wern't completed as the nParticle method seems more efficient than doing cylinders and spheres generated between points or using the curve to be duplicated and shifted to create 3 other curves that could guide the geometry
https://www.youtube.com/watch?v=Evbp8kuuIg4
https://forums.autodesk.com/t5/maya-modeling/build-cylinder-between-two-points-with-mel-or-python/td-p/10855803

Through the use of outputs from the script editor , the moves I made on the Node graph were recorded with MEL, combined with Maya cookbook, was able to convert to Python the above

https://www.youtube.com/watch?v=mzAzkXqtIss

More: https://docs.google.com/document/d/1u5JrfhEpEuWSYlkncOMSKeHQYxj71DxQaNjE8Asr3rU/edit?usp=sharing