LIGHTTNING GENERATION IN PYTHON IN MAYA

Important Resources:

Maya Programming with Python cookbook
https://www.cs.rpi.edu/~trink/Courses/RobotManipulation/lectures/lecture6.pdf
https://songho.ca/opengl/gl_rotate.html
https://www.grid.uns.ac.rs/jged/download/v7n2/jged_v7_n2_p3.pdf
OpenGL Rotation About Arbitrary Axis (songho.ca)
https://devforum.roblox.com/t/electric-arc-demo-with-rbxls/35433
https://web.archive.org/web/20190816102613/http://drilian.com/2009/02/25/lightning-bolts/
https://andrewtubelli.com/portfolio/curves-to-geometry/
https://www.youtube.com/watch?v=mzAzkXqtIss
https://www.sciencelearn.org.nz/resources/239-lightning-explained
-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
-further development
https://en.wikipedia.org/wiki/Lichtenberg_figure
https://www.semanticscholar.org/paper/Volume-Lightning-Rendering-and-Generation-Using-Lapointe-Stiert/6206e70d8464eae058596a60296b879ebae34638
https://redirect.cs.umbc.edu/~ebert/693/TLin/node18.html
https://www.michael-hansmeyer.com/l-systems

More images belo

LIGHTTNING GENERATION IN PYTHON IN MAYA

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 (psu.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=i&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DijMyo-6T8Ow&psig=AOvVaw1rchwzlRDBRr-9oeIid50R&ust=1679135536917000&source=images&cd=vfe&ved=0CA8QjhxqFwoTCOjWgLvh4v0CFQAAAAAdAAAAABAE
PYTHON / Poly Cutting and Splitting | Maya Spiral

Lightning

+Bolt
issue: built into Maya somewhat, other examples of a bolt with Bifrost along with sparking at secondary surface contact
+Cloud generated-Surface-Interior-Surface to Surface
issue: can I do all 3 in time? How can I do a toon shader but in Python and not with node graph?
+Orb
issue: 3-4 layers, would be doing part of above's interior is it in addition to that? Is it complex enough? How can I have the surface of a sphere partially disappear and morph without using the node graph?

Forest Growth

Issue: how would I map out the areas? Trees we did partially in class. To curve mushroom stems how would I accomplish that and not cross them/ have them intersect? Is it too overused? Especially when it was done in class

Crochet pattern to 3D model & model with crochet material

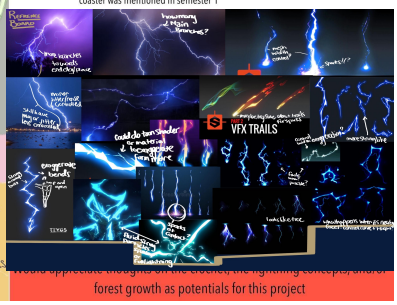
Issue: generating color changes, sewn parts, scaling it up. Would stitches be modeled prior? How will I define all stiches, such as bobble stiches that are raised? How can I let people add other stiches? Is it too niche to be practical? Is it too easy with text conversion?

Trinity College/Bookshelves

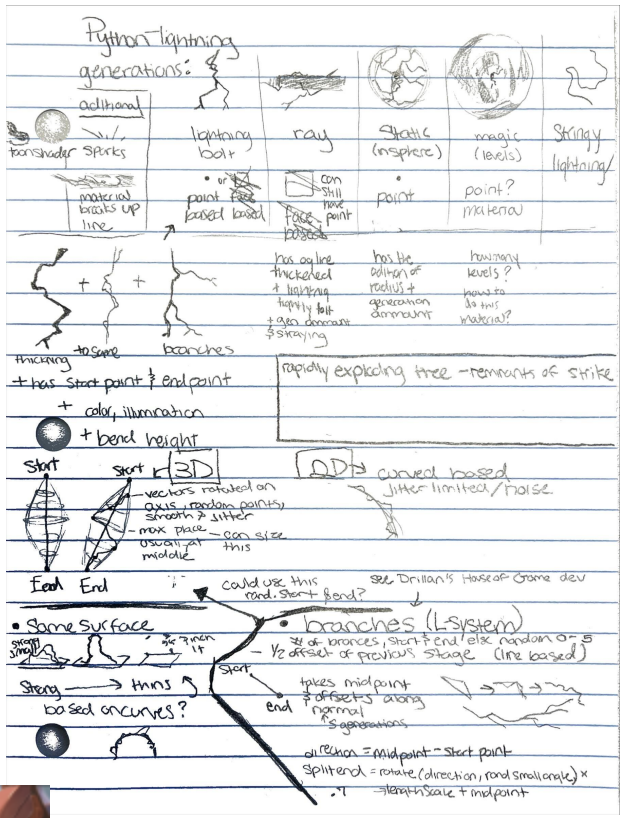
Issue: books and book shelves overused, potential loss of creativity, most of it has been done in class, would like to save it as a back up if everything else fails

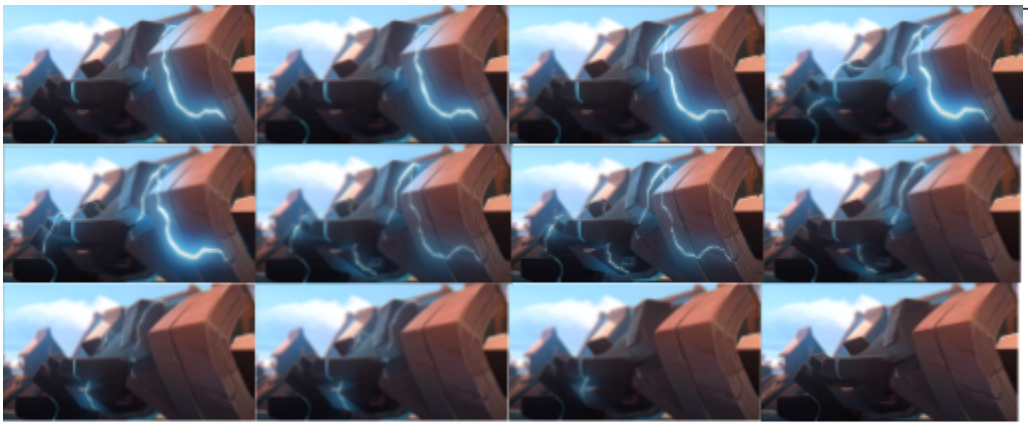
Chain/Rope with shader/materials

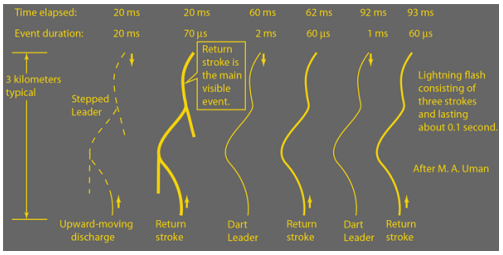
Issues: potentially too simplistic for just a chain, very similar to the bridge. While not covered in class lectures, is avaible and roller coaster was mentioned in semester 1



forest growth as potentials for this project







Arcane: Animation Test (2 second scene)

LIGHTTNING GENERATION IN PYTHON IN MAYA

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

[redacted]

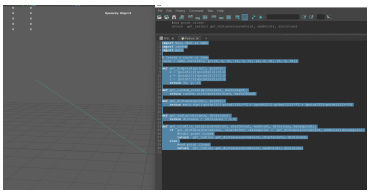
Draft 4:

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 ⅓ 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

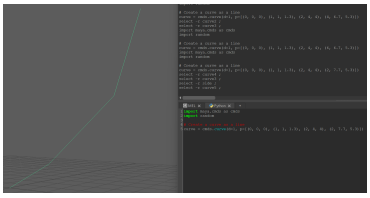
Minor Branches adapt: https://web.archive.org/web/20190816102613/http://drilian.com/2009/02/25/lightning-bolts/

```
import maya.cmds as cmds
import random

# Create a curve or line
curve = cmds.curve(d=1, p=[(0, 0, 0), (1, 1, 1), (2, 2, 2), (3, 3, 3)]
-Midpoint formula (x1+x2/2,y1+y2/2,z1+z2/2)-Math class

def get_midpoint(point1, point2):
    x = (point1[0]+point2[0])/2.0
    y = (point1[1]+point2[1])/2.0
    z = (point1[2]+point2[2])/2.0
    return (x, y, z)
```



- random.uniform(*low=0.0, high=1.0, size=None*)

https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html#numpy.random.uniform



-Distance formula, NOT following curve because if it follows and on same surface could pass through "floor" if its a long n shaped curve square root((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)

```
def get_distance(point1, point2):
    return math.sqrt((point2[0]-point1[0])**2 + (point2[1]-point1[1])**2 +
(point2[2]-point1[2])**2)
```

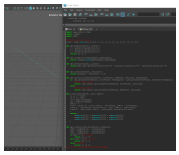
-Radius is distance from start or end(determined by breakpoint and which is closer and divisions(user controlled

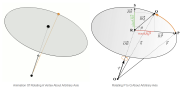
```
def get_radius(distance, divisions):
    return distance / (divisions * 3.0)

def get_rotation_radius(curvePoint, startPoint, endPoint, divisions, breakpoint):
    if get_distance(curvePoint, startPoint) +breakpoint< get_distance(curvePoint,
endPoint)+breakpoint:
        #startpoint closer
        return get_radius( get_distance(curvePoint, startPoint), divisions):
    else:
        #end point closer
        return get_radius( get_distance(curvePoint, endPoint), divisions):
```

https://download.autodesk.com/us/maya/2011help/CommandsPython/curve.html#:~:text=You%20can%20replace%20an%20existing%20curve%20by%20using,To%20create%20a%20curve-on-surface%2C%20use%20the%20curveOnSurface%20command.
Replacing curve with new one that has midpoints (example)

```
cmds.curve( 'curve1', r=True, p=[(0, 0, 0), (3, 5, 6), (10, 12, 14), (9, 9, 9)] )
```

https://www.grid.uns.ac.rs/iged/download/v7n2/iged_v7_n2_p3.pdf
OpenGL Rotation About Arbitrary Axis (songho.ca)
Rotate point def
Rodrigues' rotation formula:
https://www.cs.rpi.edu/~trink/Courses/RobotManipulation/lectures/lecture6.pdf





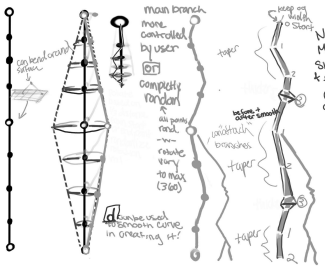
```
def rotate_point(point, axis, angle):
    x, y, z = point
    u, v, w = axis
    cos_a = math.cos(angle)
    sin_a = math.sin(angle)
    matrix = [
        [cos_a+(u**2)*(1-cos_a), u*v*(1-cos_a)-w*sin_a, u*w*(1-cos_a)+v*sin_a],
        [u*v*(1-cos_a)+w*sin_a, cos_a+(v**2)*(1-cos_a), v*w*(1-cos_a)-u*sin_a],
        [u*w*(1-cos_a)-v*sin_a, v*w*(1-cos_a)+u*sin_a, cos_a+(w**2)*(1-cos_a)]
    ]
    return (
        x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0],
        x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1],
        x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2]
    )# needs to be modified to get a point
```

$$M_R = \begin{pmatrix} (1-c)x^2 + c & (1-c)xy - sz & (1-c)xz + sy & 0 \\ (1-c)xy + sz & (1-c)y^2 + c & (1-c)yz - sx & 0 \\ (1-c)xz - sy & (1-c)yz + sx & (1-c)z^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $c = \cos\theta, \quad s = \sin\theta, \quad \vec{r} = (x, y, z)$

Note on creating a curve, d=??? Can be used to "smooth" rough edges , could be a factor to use

```
def subdivide_curve(startPoint, midPoint, endPoint, subdivisions,curveDeg):
    points = [startPoint, midPoint, endPoint]
    print("Initial list of points: ", points)
    for i in range(subdivisions):
        newPoints = []
        for j in range(len(points) - 1):
            midpoint = get_midpoint(points[j], points[j+1])
            newPoints.append(points[j])
            newPoints.append(midpoint)
        newPoints.append(points[-1])
        points = newPoints
        print("List of points after subdivision", i+1, ": ", points)
    curveName = cmds.curve(d=curveDeg, p=points)
    return points

startPoint = (0, 0, 0)
midPoint = (1, 2, 3)
endPoint = (4, 5, 6)
curveDeg=1
subdivisions = 2
subdivide_curve(startPoint, midPoint, endPoint, subdivisions,curveDeg
```



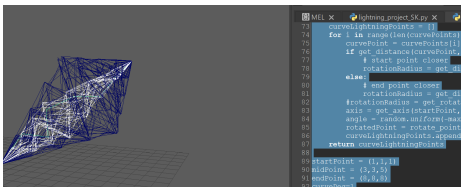

```
startPoint = (0, 0, 0)
midPoint = (1, 2, 3)
endPoint = (4, 5, 6)
curveDeg=1
subdivisions = 2
subdivide_curve(startPoint, midPoint, endPoint, subdivisions,curveDeg)
```

```
import maya.cmds as cmds
import random
import math

#distance formula square root((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)
def get_distance(point1, point2):
    return math.sqrt((point2[0]-point1[0])**2 + (point2[1]-point1[1])**2 + (point2[2]-point1[2])**2)


def get_midpoint(point1, point2):
    x = (point1[0]+point2[0])/2.0
    y = (point1[1]+point2[1])/2.0
    z = (point1[2]+point2[2])/2.0
    return (x, y, z)

#need to find axis AND must be normalized (sqrt(x^2+y^2+z^2)
def get_axis(point1, point2):
    x_diff = point2[0] - point1[0]
    y_diff = point2[1] - point1[1]
    z_diff = point2[2] - point1[2]
    magnitude = math.sqrt(x_diff**2 + y_diff**2 + z_diff**2)
    if magnitude == 0:
        return (0, 0, 0)
    x_norm = x_diff / magnitude
    y_norm = y_diff / magnitude
    z_norm = z_diff / magnitude
    return (x_norm, y_norm, z_norm)

#Rodrigues' rotation formula
def rotate_point(point, axis, angle):
    x, y, z = point
    u, v, w = axis
    cos = math.cos(angle)
    sin = math.sin(angle)
    matrix = [
        [cos + (u**2)*(1 - cos), u*v*(1 - cos)-w*sin, u*w*(1 - cos)+v*sin],
        [u*v*(1 - cos)+w*sin, cos+(v**2)*(1 - cos), v*w*(1 - cos)-u*sin],
        [u*w*(1 - cos)-v*sin, v*w*(1 - cos)+u*sin, cos+(w**2)*(1-cos)]
    ]
    x_new = x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0]
    y_new = x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1]
    z_new = x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2]
    return (x_new, y_new, z_new)

def subdivide_curve(startPoint, midPoint, endPoint, subdivisions,curveDeg):
    points = [startPoint, midPoint, endPoint]
    print("Initial list of points: ", points)
    for i in range(subdivisions):
        newPoints = []
        for j in range(len(points) - 1):
            midpoint = get_midpoint(points[j], points[j+1])
            newPoints.append(points[j])
            newPoints.append(midpoint)
        newPoints.append(points[-1])
        points = newPoints
        print("List of points after subdivision", i+1, ": ", points)
    curveName = cmds.curve(d=curveDeg, p=points)
    return points

def curve_lightning_main(curvePoints, startPoint, endPoint, subdivisions, breakingDistance, maxAngle, minAngle):
    curveLightningPoints = []
    for i in range(len(curvePoints)):
        curvePoint = curvePoints[i]
        if get_distance(curvePoint, startPoint) < get_distance(curvePoint, endPoint):
            # start point closer
            rotationRadius = get_distance(curvePoint, startPoint) / breakingDistance
        else:
            # end point closer
            rotationRadius = get_distance(curvePoint, endPoint) / breakingDistance
        angle = random.uniform(minAngle, maxAngle)
        axis = get_axis(startPoint,endPoint)
        rotatedPoint = rotate_point(curvePoint, axis, angle)
        curveLightningPoints.append(rotatedPoint)
    return curveLightningPoints

def create_tubular_mesh(curves, radius, expand, meshName):
    meshGroup = []

    for current in curves:
        curveEx = cmds.curve(d=1, p=current)
        circleEx = cmds.circle(n="circle", r=radius, ch=False)

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

        mesh = cmds.extrude(circleEx, curveEx, 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))
        #end and start sphere caps the line segment
        cSphere = cmds.sphere(n="start_sphere", r=radius, ch=False)[0]
        cmds.move(current[0][0], current[0][1], current[0][2], cSphere)
        cSphere = cmds.sphere(n="end_sphere", r=radius, ch=False)[0]
        cmds.move(current[-1][0], current[-1][1], current[-1][2], cSphere)
        meshGroup.append([tubular_mesh,cSphere, cSphere])
        cmds.delete(curveEx, circleEx)
        #mesh created, each element in the tube AND both spheres
    return meshGroup

#-----------------------------------------------------------------------------------------------------------------------


startPoint = (1,1,1)
#midPoint controlls the max val  radius and random generation spacing
midPoint = (3,3,5)
endPoint = (8,8,8)

#curveDeg is the d value of the curve constantly used, in oroginal and main bolt
curveDeg=1
#subdivisions controlls how many random points can be controlled

breakingDistance=2
subdivisions=2
width=0.3
#convert degrees to radiants
#90=plane surface in the way, 180=bolt
#use for restriction of the generation, for instance prevention of touching a surface
maxAngle=180 * (math.pi / 180)
minAngle=-180 * (math.pi / 180)

curvePoints = subdivide_curve(startPoint, midPoint, endPoint, subdivisions,curveDeg)
curveBolt = curve_lightning_main(curvePoints, startPoint, endPoint, subdivisions, breakingDistance, maxAngle,minAngle)

curve_name = "mainBolt"
curves = [curveBolt]
radius = 0.1
expand = 1

meshes = create_tubular_mesh(curves, radius, expand,curve_name)
```

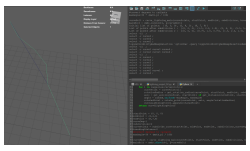
```
----------------------------------------------------------------------------------------------------------------------
import maya.cmds as cmds
import random
import math

class LightningGenerator:
    def __init__(self, startPoint, midPoint, endPoint):
        self.startPoint = startPoint
        self.midPoint = midPoint
        self.endPoint = endPoint

    def get_distance(self, point1, point2):
        return math.sqrt((point2[0]-point1[0])**2 + (point2[1]-point1[1])**2 + (point2[2]-point1[2])**2)

    def get_midpoint(self, point1, point2):
        x = (point1[0]+point2[0])/2.0
        y = (point1[1]+point2[1])/2.0
        z = (point1[2]+point2[2])/2.0
        return (x, y, z)

    def get_axis(self, point1, point2):
        x_diff = point2[0] - point1[0]
        y_diff = point2[1] - point1[1]
        z_diff = point2[2] - point1[2]
        magnitude = math.sqrt(x_diff**2 + y_diff**2 + z_diff**2)
        if magnitude == 0:
            return (0, 0, 0)
        x_norm = x_diff / magnitude
        y_norm = y_diff / magnitude
        z_norm = z_diff / magnitude
        return (x_norm, y_norm, z_norm)
```

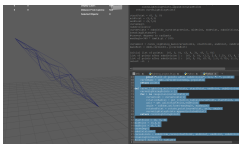
LIGHTTNING GENERATION IN PYTHON IN MAYA

```
#finding axis for rotation of the point
def get_axis(point1, point2):
    x_diff = point2[0] - point1[0]
    y_diff = point2[1] - point1[1]
    z_diff = point2[2] - point1[2]
    magnitude = math.sqrt(x_diff**2 + y_diff**2 + z_diff**2)
    if magnitude == 0:
        return (0, 0, 0)
    x_norm = x_diff / magnitude
    y_norm = y_diff / magnitude
    z_norm = z_diff / magnitude
    return (x_norm, y_norm, z_norm)

point1 = (1,1,1)
point2 = (2,3,2)
axis = get_axis(point1, point2)
print(axis)
cmds.curve(p=[point1, point2])
cmds.curve(p=[point1,axis, point2])
```

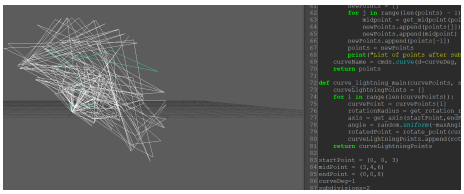


```
import maya.cmds as cmds
import random
import math

#distance formula square
root((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)
def get_distance(point1, point2):
    return math.sqrt((point2[0]-point1[0])**2 +
(point2[1]-point1[1])**2 + (point2[2]-point1[2])**2)


def get_midpoint(point1, point2):
    x = (point1[0]+point2[0])/2.0
    y = (point1[1]+point2[1])/2.0
    z = (point1[2]+point2[2])/2.0
    return (x, y, z)
```

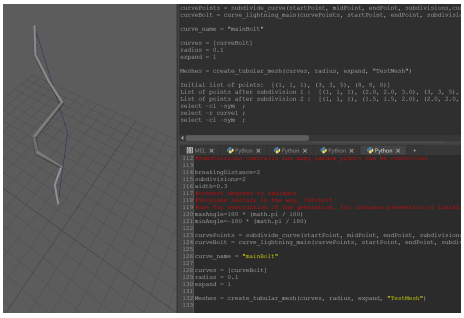




```
    #need to find axis AND must be normalized (sqrt(x^2+y^2+z^2)
def get_axis(point1, point2):
    x_diff = point2[0] - point1[0]
    y_diff = point2[1] - point1[1]
    z_diff = point2[2] - point1[2]
    magnitude = math.sqrt(x_diff**2 + y_diff**2 + z_diff**2)
    if magnitude == 0:
        return (0, 0, 0)
    x_norm = x_diff / magnitude
    y_norm = y_diff / magnitude
    z_norm = z_diff / magnitude
    return (x_norm, y_norm, z_norm)

#chosing which rotation radius based on the breakpoint and distance
from point and start and end
def get_rotation_radius(curvePoint, startPoint, endPoint, divisions,
breakingDistance):
    if get_distance(curvePoint, startPoint) <  get_distance(curvePoint,
endPoint):
        #start point closer, modified returning the radius to be used
        return  (get_distance(curvePoint, startPoint)/ (divisions
*breakingDistance))
    else:
        #end point closer
        return (get_distance(curvePoint, endPoint)/ (divisions *breakingDistance))
```



```
#Rodrigues' rotation formula, inspired by https://www.cs.rpi.edu/~trink/Courses/RobotManipulation/lectures/lecture6.pdf  and https://songho.ca/opengl/gl_rotate.html
#axsis and point DIFFRENT
def rotate_point(point, axis, angle):
    x, y, z = point
    u, v, w = axis
    cos = math.cos(angle)
    sin = math.sin(angle)
    matrix = [
        [cos + (u**2)*(1 - cos), u*v*(1 - cos)-w*sin, u*w*(1 -
cos)+v*sin],
        [u*v*(1 - cos)+w*sin, cos+(v**2)*(1 - cos), v*w*(1 -
cos)-u*sin],
        [u*w*(1 - cos)-v*sin, v*w*(1 - cos)+u*sin, cos+(w**2)*(1-cos)]
    ]
    x_new = x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0]
    y_new = x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1]
    z_new = x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2]
    return (x_new, y_new, z_new)
```



```
def subdivide_curve(startPoint, midPoint, endPoint,
subdivisions,curveDeg):
    points = [startPoint, midPoint, endPoint]
    print("Initial list of points: ", points)
    for i in range(subdivisions):
        newPoints = []
        for j in range(len(points) - 1):
            midpoint = get_midpoint(points[j], points[j+1])
            newPoints.append(points[j])
            newPoints.append(midpoint)
        newPoints.append(points[-1])
        points = newPoints
        print("List of points after subdivision", i+1, ": ", points)
    curveName = cmds.curve(d=curveDeg, p=points)
    return points

def curve_lightning_main(curvePoints, startPoint, endPoint, subdivisions, breakingDistance, maxAngle):
    curveLightningPoints = []
    for i in range(len(curvePoints)):
        curvePoint = curvePoints[i]
        rotationRadius = get_rotation_radius(curvePoint, startPoint, endPoint, subdivisions, breakingDistance)
        axis = get_axis(curvePoint, startPoint) if get_distance(curvePoint, startPoint) < get_distance(curvePoint, endPoint) else get_axis(curvePoint, endPoint)
        angle = random.uniform(-maxAngle, maxAngle)
        rotatedPoint = rotate_point(curvePoint, axis, angle*rotationRadius)
        curveLightningPoints.append(rotatedPoint)
    return curveLightningPoints


startPoint = (0, 0, 0)
midPoint = (3,4,3)
endPoint = (8,7,8)
curveDeg=1
subdivisions=2
curvePoints = subdivide_curve(startPoint, midPoint, endPoint, subdivisions,curveDeg)
breakingDistance=2
#convert degrees to radiants
maxAngle=30 * (math.pi / 180)

curveBolt = curve_lightning_main(curvePoints, startPoint, endPoint, subdivisions, breakingDistance, maxAngle)
mainBolt = cmds.curve(d=1, p=curveBolt)


import maya.cmds as cmds

def create_tubular_mesh(curves, radius, expand, meshName=None):
    meshGroup = []

    for current in curves:
        curveEx = cmds.curve(d=1, p=current)
        circleEx = cmds.circle(n="circle", r=radius, ch=False)

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

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

```
    def rotate_point(self, point, axis, angle):
        x, y, z = point
        u, v, w = axis
        cos = math.cos(angle)
        sin = math.sin(angle)
        matrix = [
            [cos + (u**2)*(1 - cos), u*v*(1 - cos)-w*sin, u*w*(1 - cos)+v*sin],
            [u*v*(1 - cos)+w*sin, cos+(v**2)*(1 - cos), v*w*(1 - cos)-u*sin],
            [u*w*(1 - cos)-v*sin, v*w*(1 - cos)+u*sin, cos+(w**2)*(1-cos)]
        ]
        x_new = x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0]
        y_new = x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1]
        z_new = x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2]
        return (x_new, y_new, z_new)

    def subdivide_curve(self, startPoint, midPoint, endPoint, subdivisions, curveDeg):
        points = [startPoint, midPoint, endPoint]
        print("Initial list of points: ", points)
        for i in range(subdivisions):
            newPoints = []
            for j in range(len(points) - 1):
                midpoint = self.get_midpoint(points[j], points[j+1])
                newPoints.append(points[j])
                newPoints.append(midpoint)
            newPoints.append(points[-1])
            points = newPoints
            print("List of points after subdivision", i+1, ": ", points)
        curveName = cmds.curve(d=curveDeg, p=points)
        return points

    def curve_lightning_main(self, curvePoints, startPoint, endPoint, subdivisions, breakingDistance, maxAngle, minAngle):
        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):
    meshGroup = []

    for current in curves:
        curveEx = cmds.curve(d=1, p=current)
        circleEx = cmds.circle(n="circle", r=radius, ch=False)

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

        mesh = cmds.extrude(
            circleEx, curveEx, 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))

        # end and start sphere caps the line segment
        sSphere = cmds.sphere(n="start_sphere", r=radius, ch=False)[0]
        cmds.move(current[0][0], current[0][1], current[0][2], sSphere)
        eSphere = cmds.sphere(n="end_sphere", r=radius, ch=False)[0]
        cmds.move(current[-1][0], current[-1][1], current[-1][2], eSphere)

        meshGroup.append([tubular_mesh, sSphere, eSphere])
        cmds.delete(curveEx, circleEx)

    # mesh created, each element is the tube AND both spheres
    return meshGroup

def apply_glow(meshes):
    surfaceShader = cmds.shadingNode('surfaceShader', asShader=True)
    surfaceShaderSG = cmds.sets(renderable=True, noSurfaceShader=True, empty=True, name=surfaceShader+'SG')
    cmds.connectAttr(surfaceShader+'.outColor', surfaceShaderSG+'.surfaceShader', force=True)

    cmds.setAttr(surfaceShader+'.outColor', 0.105276, 0.456424, 0.566, type='double3')
    cmds.setAttr(surfaceShader+'.outGlowColor', 0.130508, 0.04525, 0.181, type='double3')

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


# Usage:
startPoint = (1, 1, 1)
midPoint = (3, 3, 5)
endPoint = (8, 8, 8)
subdivisions = 2
breakingDistance = 1
curveDeg=1
maxAngle = 180 * (math.pi / 180)
minAngle = -180 * (math.pi / 180)

generator = LightningGenerator(startPoint, midPoint, endPoint)
curvePoints = generator.subdivide_curve(startPoint, midPoint, endPoint, subdivisions, curveDeg)
curveBolt = generator.curve_lightning_main(curvePoints, startPoint, endPoint, subdivisions, breakingDistance, maxAngle, minAngle)


curves = [curveBolt]
curve_name = "mainBolt"
radius = 0.1
expand = 1
meshes = create_tubular_mesh(curves, radius, expand, curve_name)
apply_glow(meshes)
```

LIGHTTNING GENERATION IN PYTHON IN MAYA

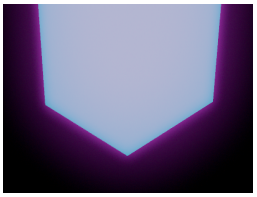
```
    if mesh_name:#rename if one is picked
        tubular_mesh = cmds.rename(mesh[0], meshName + str(curves.index(current) + 1))
    #end and start sphere caps the line segment
    sSphere = cmds.sphere(n="start_sphere", r=radius, ch=False)[0]
    cmds.move(current[0][0], current[0][1], current[0][2], sSphere)
    eSphere = cmds.sphere(n="end_sphere", r=radius, ch=False)[0]
    cmds.move(current[-1][0], current[-1][1], current[-1][2], eSphere)
    meshGroup.append([tubular_mesh,sSphere, eSphere])
    cmds.delete(curveEx, circleEx)
    #mesh created, each element is the tube AND both spheres
  return meshGroup

curve_points_1 = [(1, 0, 0), (2, 1, 0), (3, -1, 0), (4, 0, 0)]
curve_points_2 = [(5, 0, 0), (6, 1, 0), (7, -1, 0), (8, 0, 0)]
curves = [curve_points_1, curve_points_2]
radius = 0.1
expand = 1

tubular_meshes = create_tubular_mesh(curves, radius, expand, "TestMesh")
```

Failed 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=Evbp6kuuJg4

https://forums.autodesk.com/t5/maya-modeling/build-cylinder-between-two-points-with-mel-or-python/td-p/10855803

```
def create duplicate curve(curve_name, width):
  curve_1 = cmds.duplicate(curve_name)[0]
  cmds.move(width, 0, 0, curve_1, relative=True)
  return curve_1

cmds.rename(mainMesh, "mainMeshCurve")
#with mainMesh, I want to generate a mesh that follows its points with a new function called curve_Mesh
# Create a duplicate of the main bolt curve shifted over by 1 unit

width = 0.1
shifted_curves = create_shifted_curves(curve_name, width)

def cylinderBetweenPoints(p1 = [], p2 = [], r = None, name = None):
  #getting height
  x1, y1, z1 = p1
  x2, y2, z2 = p2
  h = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
  c = mc.polyCylinder(name= name, r = r, h = h)[0]
  mc.xform(c, t= p1, wo = True)
  aimAt = mc.group(name = name + "aimGrp", em = True)
  mc.xform(aimAt, t= p2, wo = True)
  mc.delete(mc.aimConstraint(aimAt, c, aimVector = (0,1,0)))
  mc.move(0,h/2.0, c, wo= True, r = True)
  mc.delete(aimAt)

cylinderBetweenPoints(p1 = [-3.061, 18.228, 17.122], p2= [-0.823, 15.562, 15.568], r = 0.9, name = "cyl1")

#modified from https://andrewtubelli.com/portfolio/curves-to-geometry/
#method by Andrew Tubelli's curveToGeom.py
def curveToGeom(curve_list, inputText, mesh_spacing, blobby_radius, smoothing_radius):
  # Rebuild selected curves, if there is none then prints out an error
  if not curve_list:
      print("No curves have been selected. Please select curves before calling this function.")
      return

  cmds.makeIdentity(apply=True, t=1, r=1, s=1, n=0)
  numCurves = len(curve_list)
  for x in curve_list:
      arcLen = cmds.arclen(x)
      numSpan = int(math.ceil(arcLen)*5
      cmds.rebuildCurve(x, rt=0, s=numSpan)

  #get coordinates of the CVs for each curve in an array
  CVcoords=[]
  for x in curve_list:
      numCV = cmds.getAttr(x+'.degree') + cmds.getAttr(x+'.spans')
      CVcoord = cmds.getAttr(x+'.cv[0:'+str(numCV)+']')
      for i in CVcoord:
          CVcoords.append(i)

  # Create nParticle at each point
  nPartObj = cmds.nParticle(p=CVcoords, n=inputText+'_nParticle')
  cmds.setAttr('nucleus1.gravity', 0)

  #nParticle to poly
  cmds.select(nPartObj)
  nPartSurf = mel.eval('particleToPoly;')
  cmds.rename(inputText)
  cmds.setAttr(nPartObj[1]+'.meshMethod', 3)

  # alter the geometry
  cmds.setAttr(inputText+'.nParticleShape1'+'.radius', radius)
  cmds.setAttr(inputText+'.nParticleShape1'+'.meshTriangleSize', mesh_spacing)
  cmds.setAttr(inputText+'.nParticleShape1'+'.blobbyRadiusScale', blobby_radius)
  cmds.setAttr(inputText+'.nParticleShape1'+'.meshSmoothingIterations', smoothing)
  cmds.currentTime(0)
#end of the modified method by Andrew Tuvelli
#------------------------------ Implementing modified method by Andrew Tuvelli-------------------------------------
curve_list = ['mainMesh']#would need to have what is selected the curves added to the list
inputText = 'output_geom'
mesh_spacing = 0.27
blobby_radius = 0.96
smoothing = 4
radius = 0.45

curveToGeom(curve_list, inputText, mesh_spacing, blobby_radius, smoothing, radius)
#-------------------------------------------------------------------------------------------------------------
```



```
import maya.cmds as cmds

# Create surfaceShader
surfaceShader = cmds.shadingNode('surfaceShader', asShader=True)
surfaceShaderSG = cmds.sets(renderable=True, noSurfaceShader=True, empty=True, name=surfaceShader+'SG')
cmds.connectAttr(surfaceShader+'.outColor', surfaceShaderSG+'.surfaceShader', force=True)

cmds.setAttr(surfaceShader+'.outColor', 0.105276, 0.456424, 0.566, type='double3')
cmds.setAttr(surfaceShader+'.outColor', 0.105842, 0.456559, 0.566, type='double3')
cmds.setAttr(surfaceShader+'.outColor', 0.105842, 0.456636, 0.566, type='double3')
cmds.setAttr(surfaceShader+'.outGlowColor', 0.130508, 0.04525, 0.181, type='double3')
cmds.setAttr(surfaceShader+'.outGlowColor', 0.130441, 0.045069, 0.181, type='double3')

# Create a cube, REPLACE with lightning
cube = cmds.polyCube()[0]

# Assign the surfaceShader to the cube
cmds.select(cube)
cmds.hyperShade(assign=surfaceShaderSG)
#-------------------------------------------------------------------------------------------------------------
import maya.cmds as cmds

def apply_glow(meshes):
    surfaceShader = cmds.shadingNode('surfaceShader', asShader=True)
    surfaceShaderSG = cmds.sets(renderable=True, noSurfaceShader=True, empty=True, name=surfaceShader+'SG')
    cmds.connectAttr(surfaceShader+'.outColor', surfaceShaderSG+'.surfaceShader', force=True)

    cmds.setAttr(surfaceShader+'.outColor', 0.105276, 0.456424, 0.566, type='double3')
    cmds.setAttr(surfaceShader+'.outColor', 0.105842, 0.456559, 0.566, type='double3')
    cmds.setAttr(surfaceShader+'.outColor', 0.105842, 0.456636, 0.566, type='double3')
    cmds.setAttr(surfaceShader+'.outGlowColor', 0.130508, 0.04525, 0.181, type='double3')
    cmds.setAttr(surfaceShader+'.outGlowColor', 0.130441, 0.045069, 0.181, type='double3')

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

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