The change of color of the plane depending on its direction of translation

The topic I will be choosing is part a. I am using python and the bpy library to change the color of the plane depending on its translation in the 3D map. Essentially the plane is like a chameleon. The color coding used will be in the RGB diffuse scale (for more information see appendix B). The x axis represents R, the y axis represents G, and the z axis represents B. For example, if the plane were to translate in the x axis, it would turn more red. The color of the plane will only change as long as it translates in the positive axis. If it goes in the negative direction, it will automatically be set to 0 as negative numbers are not read. For example, if the plane went straight up it would turn blue, if it went straight down it would be black. Furthermore, the color white is generated on the RGB scale if there is an equal ratio or value in RGB and it is higher or equal to 1. In short, this would happen to a plane but very temporarily if it was translating in all axes positive at an equal amount.

During the script, the plane is first selected as the active object to make sure nothing else gets colored at the same time as in the initial trial the plane and the runway were colored at the same time. The plane must also be selected for the entire function to work.

In the python bpy code, I would first start at (0,0,0) for the color ‘k’ so it can refer to the fact that it is stationary. The plotting values of the next color would be the difference between the finishing point and the starting point of the position of the plane in the x,y, and z direction. Ex: if position A = (0,0,5) and B = (2,3,6), then the color from A to B would be (2,3,1) which is around light green. It would continue on every single point except the last where it eventually goes back to the beginning. If the value of each axis was higher, so was the intensity of the color. For example, if and only if the y value was positive but at a low value the color would be dark green. If it became significantly higher (hundreds) it would turn lighter green.

All the different diffusions of each color in the material were created under a list assigned to a variable (see appendix A for all the RGB inputs for each position of the plane). First, I would set the plane as the active object using its variable and then I would make a new material and add the newly created one directly to the plane so I could set it as the active material. Then, I would make an empty list followed by a for loop to fill up all the frames in the specific timeline onto that list. This was to make sure I didn’t have to list out ALL the frames in the list as it would take too much unnecessary space. In the loop range, the low function was 0 as it marked the starting point and the end point was 1536 instead of 1512 as the end of the range doesn’t include the number displayed itself. The third number would be the amount i increases every time the inside loop finishes. It is also important to make sure there will be as many keyframes as colors in the colors list because I want the color of the material to change at these specific locations on the timeline.

Finally, I would make the last for loop where I would insert the keyframes with each respective color. Due to this, there are two loop variables (f for frames and c for colors). It is also required to zip the two lists using the zip function. I would first set the newly made material's diffuse color to the current color from the color list. Then I would insert a diffuse color at its current frame. I used a zip function because I believed it was the appropriate algorithm to turn two lists of equal length into a dictionary where frames represent the key and the colors represent its value within the dictionary.

REFLECTION:

There were definitely multiple errors and difficulties I encountered during the process.

I accidently created the material within the for loop, appended to the plane and set it as the active material, so it created multiple materials with the same name instead of just one. Furthermore, negative numbers in RGB reflected on why there were frames in which the plane was black. However, I did not know this earlier. Next time, I should try making every frame absolute value.

I experienced a challenge with getting the camera angle right with sample 3 because of an optical illusion. At the beginning, it was rotating counterclockwise, but after the first half it would rotate clockwise. I had to fix many rotation axes because of this.

My initial animation was WAY too long as the original video was over 4000 frames. I also realized that I didn't need to set the plane as the active object in order to just color the plane, but it does ensure nothing else unintentionally happens.

Next time, I should add more graphics characteristics for my project such as more lights, as well as windows on my plane to enhance transparency, translucency, reflection, and refraction. However, that would lead to an increase in render time. I was also informed that max for RGB is 100, but I can actually go more since one point went 200.

I also noticed particles of certain colors radiating near the ground. This is due to the effect of my coding. If the intensity was past 1 in any of the RGB, it would start glowing.

Next time, I should try to keep every value below or equal to 1. I could do a test where I divide every point by the maximum value in each of the axes. (See appendix C).

Also, due to my programmed colors, it appeared that since I was providing color through code, the automatic algorithm that responds to light did not work. My plane was generating color, not reflecting color. Therefore, reflection wasn't visible.

During my shots for the sample mimesis, it was pretty difficult to program because I had to make sure every camera shot and angle was precise to the video imitation. There were some frames where the camera wasn't fully in the picture, but that was to make sure the camera translates for the plane's next big actions. Plus, if I set too many locations and rotations, it would make it look too dizzy as the camera moves way too quickly.

As seen in appendix D, this was my initial attempt to color a cube and change the color when it translates. However, it did not work as I wasn't inserting any color. I also did not set the material to the object that I wanted to color (i.e, plane.active_material = variablename).

See appendix E for full bpy code used in Blender.

Appendix:

A (coding):

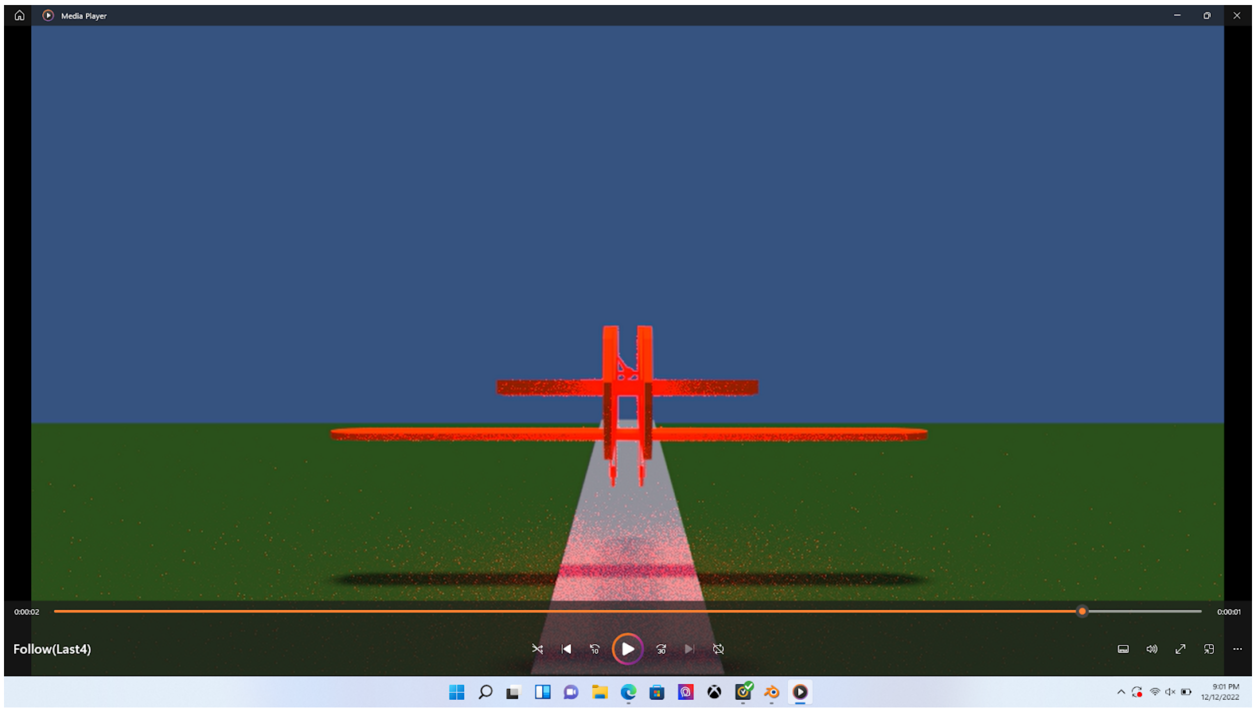
```
color = [(0,0,0), (0, 15, 0), (0, 50, 0), (0, 70, 30), (0, 100,
30), (0, 100, 60), (0, 50, 130), (0, 50, 150), (0, 0, 100), (0,
0, 200), (0, 50, 100), (0, 50, -100), (0, 0, -400), (0, 50,
-200), (0, 50, -50), (50, 50, 0), (150, 0, 50),(50, 0, 0), (25,
-75, 0), (0, -375, -50), (25, -25, 0),(25, 25, 25), (-25, 25,
25), (-50, 0, 0), (-50, 0, 0), (-50, 0, 0), (-50, 0, 0),(-25,
-25, 0), (0, -25, 0), (0, -25, 0), (0, -50, 50),(0, -50, 50),
(0, -50, 50), (0, 0, 50), (0,0,50), (0,0,50), (0,0,50), (0, 0,
50), (0,0,50), (-10,-10,50), (-10,-10,50), (-10, -10, 0),
(-5,-5,0), (-5,-5,25), (-2,-2,0), (-2,-2,0),
(-2,-2,0),(-2,-2,0), (-7,-7,0),(-5, -5, 0), (-5, -5, 0), (-30,
-30, 0), (-10,-10,0), (-10, -10, 0), (5, 5, -25), (5, 5, -25),
(0, 0, -50), (0, 0, -50), (0, 0, -425), (25, 25, -25), (25, 25,
-25), (50, 50, -25), (50, 10, -20), (0,0,0)]
```

```
def select_color():
    mat = bpy.data.materials.new(name="E") #set new material to variable
    plane.data.materials.append(mat) #add the material to the object
    plane.active_material = mat #set it as the active material
    frames = [] #empty array list that will later be denoted as all frames within
    for i in range(0, 1536, 24):
      frames.append(i)
    for f, c in zip(frames, color):
       # Set the material's diffuse color to the current color from the colors list
       mat.diffuse_color = c
       # Insert a diffuse color keyframe at the current frame
       mat.keyframe_insert(data_path='diffuse_color', frame=f, index=-1)
```

B (credit):

Source:
[Color Animation in Blender with Python - Prospero Coder]()

C (glowing):



D (initial trial):

```
import bpy
import math
import bmesh

# Initialize lists of tuples

positions = (0,0,0), (1,0,0), (0,1,0), (0,0,1), (0,0,0)

# Initialize object references
plane = bpy.data.objects["Cube"]

# Declare function
def set_keyframes():
    frame_num = 0

    activeObject = bpy.context.active_object #Set active object to variable
    mat = bpy.data.materials.new(name="M") #set new material to variable
    activeObject.data.materials.append(mat) #add the material to the object

    for index,position in enumerate(positions):
        bpy.context.scene.frame_set(frame_num)

        # Record location coords
        plane.location = position

        # Record keyframes
        plane.keyframe_insert(data_path='location', index=-1)

        bpy.context.object.active_material.diffuse_color = ((positions[index][0])
        , (positions[index][1]), (positions[index][2])) #change color




        frame_num += 5


    pass

def main():
    set_keyframes()
    pass
```

```
main()
```

Complex camera:

Frames = [792, 888, 936, 984, 1080, 1128, 1176, 1202, 1224, 1272, 1365, 1392]

Locations = (22.5, -36, 335), (66.97601, -39.5, 493.67), (87, -25.66, 547.65), (32.2, -23.44, 576.95), (6.88, -44, 618.8), (-0.16956, -53.6, 684.72), (-50.86, -52, 720.8), (-78.7, -49.73608, 710.53217), (-88.3, -46.8, 717.2), (-105.88, -112.53450, 730.4), (-109.43, -106.78, 653.34), (-109.43, -106.78, 618.96)

Rotations = (30, 0, 70), (60, 0, 70), (95, 0, 80), (145, 0, 90), (138.886, 0, 84.418), (80, 0, 80.972), (0, 0, 60), (-45, 17, 90), (-45, 0, 60), (0,0,0), (0,0,0), (0,0,0)

Appendix E:

```
import bpy
import math
import bmesh

# Initialize lists of tuples

positions = (0,-35,3.7), (0, -20, 3.7), (0, 30, 3.7), (0, 100, 30), (0, 200, 60), (0, 300, 120), (0, 350,
250), (0, 400, 400), (0, 400, 500), (0, 400, 700), (0, 450, 800), (0, 500, 700), (0, 500, 300), (0,
550, 100), (0, 600, 50), (50, 650, 50), (200, 650, 100), (250, 650, 100),(275, 575, 100), (275,
200, 50), (300, 175, 50), (325, 200, 75),(300, 225, 100), (250, 225, 100), (200, 225, 100), (150,
225, 100), (100, 225, 100), (50, 225, 100),(25, 200, 100), (0, 175, 100), (0, 150, 100), (0, 100,
150), (0, 50, 200), (0, 0, 250), (0, 0, 300), (0,0,350), (0,0,400), (0,0,450), (0, 0, 500), (0,0,550),
(-10,-10,600), (-20,-20,650), (-30, -30, 650), (-35,-35,650), (-40,-40,675), (-42,-42,675),
(-42,-42,675), (-44,-44,675),(-46,-46,675), (-48,-48,675),(-55, -55, 675), (-60, -60, 675), (-90,
-90, 675), (-100,-100,675), (-110, -110, 675), (-105, -105, 650), (-100, -100, 625), (-100, -100,
575), (-100, -100, 525), (-100, -100, 100), (-75, -75, 75), (-50, -50, 50), (0, -40, 30), (0, -35, 3.7)


positions_cam = (0,-35,3.7), (0, -20, 3.7), (0, 30, 3.7), (0, 100, 30), (0, 200, 60), (0, 300, 120), (0,
350, 250), (0, 400, 400), (0, 400, 500), (0, 400, 700), (0, 450, 800), (0, 500, 700), (0, 500, 300),
(0, 550, 100), (0, 600, 50), (50, 650, 50), (200, 650, 100), (250, 650, 100),(275, 575, 100), (275,
200, 50), (300, 175, 50), (325, 200, 75),(300, 225, 100), (250, 225, 100), (200, 225, 100), (150,
225, 100), (100, 225, 100), (50, 225, 100),(25, 200, 100), (0, 175, 100), (0, 150, 100), (0, 100,
150), (0, 50, 200), (0, 0, 250), (0, 0, 300), (0,0,350), (0,0,400), (0,0,450), (0, 0, 500), (0,0,550),
(-10,-10,600), (-20,-20,650), (-30, -30, 650), (-35,-35,650), (-40,-40,675), (-42,-42,675),
(-42,-42,675), (-44,-44,675),(-46,-46,675), (-48,-48,675),(-55, -55, 675), (-60, -60, 675), (-90,
-90, 675), (-100,-100,675), (-110, -110, 675), (-105, -105, 650), (-100, -100, 625), (-100, -100,
575), (-100, -100, 525), (-100, -100, 100), (-75, -75, 75), (-50, -50, 50), (0, -40, 30), (0, -35, 3.7)

rotations = (90,0,90), (90,0,90), (90,0,90), (90, -15, 90), (90, -30, 90), (90, -45, 90), (90, -60, 90),
(90, -90, 90), (90, -90, 90), (90,-90,90), (90, 0, 90), (90, 90, 90), (90, 90, 90), (90, 45, 90), (90, 0,
90), (135, 0, 0), (90, 0, 0), (90, 0, 0), (135, 0, -90), (90, 0, -90), (45, 0, 0), (45, -45,90), (45, 0,
180), (90, 0, 180), (90, 0, 180), (90, 0, 180), (90, 0, 180), (90, 0, 180),(90,0, 225), (90, 0, 270),
(90, 0, 270),(90, -30, 270), (90, -60, 270), (90, -90, 270),(180, -90, 270), (270, -90, 270), (360,
-90, 270), (450, -90, 270), (540, -90, 270), (630, -90, 270), (720, -90, 270), (810, -90, 270),(900,
-75, 225), (990, -75, 225), (1080, -75, 225), (1098, -75, 225), (1116, -75, 225), (1134, -75, 225),
(1152, -75, 225), (1170, -75, 225),(1170, -75, 225), (1080, -40, 225), (990, 0, 225), (900, 45,
225), (810, 90, 225),(720, 90, 225), (630, 90, 225), (450, 90, 225),(270, 90, 225),(270, 90, 225),
(270, 120, 225), (270, 150, 225), (270, 170, 270), (270, 180, 270)

color = [(0,0,0), (0, 15, 0), (0, 50, 0), (0, 70, 30), (0, 100, 30), (0, 100, 60), (0, 50, 130), (0, 50,
150), (0, 0, 100), (0, 0, 200), (0, 50, 100), (0, 50, -100), (0, 0, -400), (0, 50, -200), (0, 50, -50),
(50, 50, 0), (150, 0, 50),(50, 0, 0), (25, -75, 0), (0, -375, -50), (25, -25, 0),(25, 25, 25), (-25, 25,
25), (-50, 0, 0), (-50, 0, 0), (-50, 0, 0), (-50, 0, 0),(-25, -25, 0), (0, -25, 0), (0, -25, 0), (0, -50,
50),(0, -50, 50), (0, -50, 50), (0, 0, 50), (0,0,50), (0,0,50), (0,0,50), (0, 0, 50), (0,0,50),
(-10,-10,50), (-10,-10,50), (-10, -10, 0), (-5,-5,0), (-5,-5,25), (-2,-2,0), (-2,-2,0), (-2,-2,0),(-2,-2,0),
(-7,-7,0),(-5, -5, 0), (-5, -5, 0), (-30, -30, 0), (-10,-10,0), (-10, -10, 0), (5, 5, -25), (5, 5, -25), (0, 0,
-50), (0, 0, -50), (0, 0, -425), (25, 25, -25), (25, 25, -25), (50, 50, -25), (50, 10, -20), (0,0,0)]
#all colors in the 64 frames

# Initialize object references
plane = bpy.data.objects["Plane_Cylinder"]
camera = bpy.data.objects["PlaneCamera"]

# Declare function
def set_keyframes():

    frame_num = 0


    for index,position in enumerate(positions):
        # gives me back index and value
        bpy.context.scene.frame_set(frame_num)

        # Record location coords
        plane.location = position
        camera.location = positions_cam[index]

        # Record rotation angles in XYZ Euler
        plane.rotation_euler = (math.radians(rotations[index][0]), math.radians(rotations[index][1]),
math.radians(rotations[index][2]))

        # Record keyframes
        plane.keyframe_insert(data_path='location', index=-1)
        plane.keyframe_insert(data_path='rotation_euler', index=-1)
        camera.keyframe_insert(data_path='rotation_euler', index=-1)

        frame_num += 24

    pass

def select_color():
    mat = bpy.data.materials.new(name="ColorfulChange") #set new material to variable
    plane.data.materials.append(mat) #add the material to the object
    plane.active_material = mat #set it as the active material
    frames = [] #empty array list that will later be denoted as all frames within
    for i in range(0, 1536, 24):
      frames.append(i)
    for f, c in zip(frames, color):
        # Set the material's diffuse color to the current color from the colors list
        mat.diffuse_color = c
        # Insert a diffuse color keyframe at the current frame
        mat.keyframe_insert(data_path='diffuse_color', frame=f, index=-1)



def setsamplecamera():
    simon = bpy.data.objects["Camera"] #assign camera to variable

    oneposition = (285, 222, 110), (230, 195, 110), (185, 225, 115), (155.433, 231.77, 115), (120,
225, 108), (110, 210, 117.5)
    onerotation = (70, -10, 90), (70, -45, 60), (70, 0, 90), (70, 4.836, 99.215), (80, 0, 90), (70, -20,
90)

    #oneposition and onerotation denote all locations and rotations of camera

    frameone = [552, 576, 600, 616, 632, 648]

    #all frames

    for i in range(len(frameone)):

        current_frame = frameone[i] #sets current frame to one in array list
        bpy.context.scene.frame_set(current_frame)
        simon.location = oneposition[i]#sets current location to one in array list
        simon.rotation_euler =  (math.radians(onerotation[i][0]), math.radians(onerotation[i][1]),
math.radians(onerotation[i][2])) # Record rotation angles in XYZ Euler
        simon.keyframe_insert(data_path='location', frame = current_frame , index=-1)
        simon.keyframe_insert(data_path='rotation_euler', frame = current_frame , index=-1)
        # Record keyframes

def complexity():
    complex = bpy.data.objects["ComplexCamera"] #assign camera to variable

    locations = (22.5, -36, 335), (66.97601, -39.5, 493.67), (87, -25.66, 547.65), (32.2, -23.44,
576.95), (6.88, -44, 618.8), (-0.16956, -53.6, 684.72), (-50.86, -52, 720.8), (-78.7, -49.73608,
710.53217), (-88.3, -46.8, 717.2), (-105.88, -112.53450, 730.4), (-109.43, -106.78, 653.34),
(-109.43, -106.78, 618.96)

    rotations = (30, 0, 70), (60, 0, 70), (95, 0, 80), (145, 0, 90), (138.886, 0, 84.418), (80, 0,
80.972), (0, 0, 60), (-45, 17, 90), (-45, 0, 60), (0,0,0), (0,0,0), (0,0,0)


    #Location and Rotation denote all locations and rotations of camera

    frames = [792, 888, 936, 984, 1080, 1128, 1176, 1202, 1224, 1272, 1365, 1392]

    #all frames

    for i in range(len(frames)):

        current_frame = frames[i] #sets current frame to one in array list
        bpy.context.scene.frame_set(current_frame)
        complex.location = locations[i]#sets current location to one in array list
        complex.rotation_euler =  (math.radians(rotations[i][0]), math.radians(rotations[i][1]),
math.radians(rotations[i][2])) # Record rotation angles in XYZ Euler
        complex.keyframe_insert(data_path='location', frame = current_frame , index=-1)
        complex.keyframe_insert(data_path='rotation_euler', frame = current_frame , index=-1)
        # Record keyframes





def main():
    set_keyframes()
    select_color()
    setsamplecamera()
    complexity()
    pass

main()
```