Name: Riley Ballachay

Student Number: 261019324

Answer Questions

i.

Can see from the following code that this is a form of linear interpolation where the step size is dependant upon the difference between the current and target. Note: This does lead to a little bit of asymptotic behavior, as the step size technically approaches zero asymptotically.

```
self.V_current = self.V_current * 0.9 + self.V_target*0.1
self.P_current = self.P_current * 0.9 + self.P_target*0.1
```

Based on a bit of research, this appears to be called camera path interpolation. This is different from the other interpolation forms as we are interpolating the view and projection matrices instead of the model, and we are interpolating two matrices instead of one. Note that we are also interpolating the matrices instead of quaternions.

ii.

If the two viewing projections are on different sides of the objects, you will have to travel directly through them, which isn't ideal. The velocity won't be constant, leading to weird acceleration at certain points in the interpolation. If changing between two rotations that are significantly different, there can be very fast twists.

iii.

Interpolating the matrices can result in Gimbal lock, as I observed when testing this out. Projection transformations are also generally not linear, so interpolating these can lead to wacky distortions. If you are interpolating between two homogenous projection transformations, you risk passing through a point that is non-homogeneous.

iv.

The problem comes when normalizing the quaternions to preserve unit length. This means that the angular velocity does not remain constant during the rotation. This can be confirmed using the following:

```python
import numpy as np
def angular_velocities(q1, q2, dt):
    return (2 / dt) * np.array([
        q1[0]*q2[1] - q1[1]*q2[0] - q1[2]*q2[3] + q1[3]*q2[2],
        q1[0]*q2[2] + q1[1]*q2[3] - q1[2]*q2[0] - q1[3]*q2[1],
        q1[0]*q2[3] - q1[1]*q2[2] + q1[2]*q2[1] - q1[3]*q2[0]])
```

The angular velocity is much faster in the middle of the interpolation, up to 100x, resulting in the fast acceleration in the middle. There is only a slight difference in the angular velocity over the interpolation for 180 degree rotations, so almost no acceleration is observed.

v.

First thing we want to check is if they are valid rotation matrices. This can be verified by checking if R^T == R^(-1) and det R == 1, using the following function:

```
def is_valid_rotation(R):
    R = np.around(R,decimals=4)
    if not (R.T==np.linalg.inv(R)).all():
        print("Invalid R.T ~= R-1")
    if  (np.linalg.det(R)!=1):
         print("Invalid det R~= 1")
```

We can verify that the matrices returned when the quaternion is not unit length are not valid rotation matrices, while all that are unit length are valid.