```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class MatrixOrbit : MonoBehaviour
6  {
7      //Unity Objects
8      TrailRenderer trailrender;                                    //Marks the
         path of the planet
9      public float t = 0;                                           //Simple time
          variable
10     public GameObject MainBody;                                   //Unity game
         object that marks the planet body the script is attached to
11     public MeshFilter MF;                                         //Mesh filter
          tied to planet object
12     public Vector3[] ModelSpaceVertices;                          //Array that
         stores the Vertex points of the mesh
13     //Matrix attributes
14     public float RotX, RotY, RotZ = 0;
15     public float ScaleX, ScaleY, ScaleZ = 1;
16     public float TranslateX, TranslateY, TranslateZ = 0;
17     //Planet Rotation Variables
18     public float PlanetOrbitRadius = 60.0f;
19     public float speed = 10.0f;
20     //Boolean Variables
21     public bool AxisXEnabled, AxisYEnabled, AxisZEnabled = true;
22     public bool IsSun, IsMercury = false;
23     public static MyVector3 SunPos = new MyVector3(0, 0, 0);
24     public static MyVector3 MercuryPos = new MyVector3(0, 0, 0);
25     // Start is called before the first frame update
26     void Start()
27     {
28         //Sets the mesh filter variable equal to the game object mesh filter
29         MF = MainBody.GetComponent<MeshFilter>();
30         //Gets a copy of all vertices from mesh and stores in array
31         ModelSpaceVertices = MF.mesh.vertices;
32     }
33
34     // Update is called once per frame
35     void Update()
36     {
37
38         t += Time.deltaTime * speed;
39         //Checks to see if each axis is enabled and then translates
             accordingly
40         if (AxisXEnabled == true) { TranslateX = PlanetOrbitRadius *
             Mathf.Cos(t);}
41         if (AxisYEnabled == true) { TranslateY = PlanetOrbitRadius *
             Mathf.Sin(t);}
42         if (AxisZEnabled == true) { TranslateZ = PlanetOrbitRadius *
             Mathf.Sin(t);}
43
44         Vector3[] TransformedVertices = new Vector3
```

```
            [ModelSpaceVertices.Length];
45          Matrix4By4 T = MyTransform.Translate(TranslateX, TranslateY,
              TranslateZ);
46          Matrix4By4 R = MyTransform.Rotation(RotX, RotY, RotZ); //Rotation is
              in radians
47          Matrix4By4 S = MyTransform.Scale(ScaleX, ScaleY, ScaleZ);
48          Matrix4By4 M = MyTransform.TRS(T, R, S);
49          for (int i = 0; i < TransformedVertices.Length; i++)
              { TransformedVertices[i] = M * ModelSpaceVertices[i]; }
50          MF.mesh.vertices = TransformedVertices;
51          MF.mesh.RecalculateNormals();
52          MF.mesh.RecalculateBounds();
53
54          //Line drawn to show axial tilt
55          //Debug.DrawLine((MercuryPos * 100.0f).ToUnityVector() + new
              MyVector3(0, 100, 0).ToUnityVector(), (MercuryPos *
              100.0f).ToUnityVector() + new MyVector3(0, -100, 0).ToUnityVector
              (), Color.red);
56
57          //Bool check is created to be able to toggle planet rotations in the
              menu
58          if (IsSun == true)
59          {
60              SunPos = new MyVector3(TranslateX, TranslateY, TranslateZ);
61            //  Debug.Log(SunPos.ToUnityVector());
62          }
63          if (IsMercury == true)
64          {
65              MercuryPos = new MyVector3(TranslateX, TranslateY, TranslateZ);
66            //  Debug.Log(MercuryPos.ToUnityVector());
67          }
68      }
69
70
71
72
73
74  }
75
```