

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5
6 //[ExecuteInEditMode]
7
8 struct Dvec3
9 {
10     public double x;
11     public double y;
12     public double z;
13 }
14
15
16 public class Line : MonoBehaviour
17 {
18     public Color baseColor;
19     public GameObject target;
20     public Material material;
21
22     [SerializeField] private Button Resetbutton;
23     [SerializeField] private Slider slider;
24
25     //[SerializeField] private Button Resetbutton;
26
27     public List<Vector3> points = new List<Vector3>();
28     public List<float> coeffAs = new List<float>();
29     public List<float> coeffBs = new List<float>();
30     public List<float> KnotSequence = new List<float>();
31     int Degree = 0;
32     int N = 0;
33     float t_value = 0;
34     float J = 0;
35     Vector3[, ] res;
36
37     [SerializeField] TemplatePointPlacer pointplaceController;
38     public Vector3 mousePos;
39
40     private void Start()
41     {
42         KnotSequence.Add(0);
43         KnotSequence.Add(1);
44         KnotSequence.Add(2);
45         KnotSequence.Add(3);
46         Degree = 3;
47
48     }
49
50
51     void Update()
52     {
53         N = KnotSequence.Count;
```

```
54
55     Resetbutton.onClick.AddListener(ResetActions);
56
57
58     if (pointplaceController != null)
59     {
60         if (pointplaceController.degree < 31)
61         {
62             res = new Vector3[pointplaceController.degree + 1,           ↗
63                 pointplaceController.degree + 1];
64             for (int a = 0; a <= pointplaceController.degree; a++)
65             {
66                 res[0, a] = pointplaceController.pointPool           ↗
67                     [a].transform.position;
68             }
69         }
70         if (points.Count < 31)
71         {
72             if (Input.GetMouseButtonDown(1))
73             {
74                 mousePos = Camera.main.ScreenToWorldPoint           ↗
75                     (Input.mousePosition);
76                 mousePos.z = 0;
77
78                 if (pointplaceController != null)
79                 {
80                     if (pointplaceController.degree < 31)
81                     {
82                         points.Add(pointplaceController.pointPool           ↗
83                             [pointplaceController.degree].transform.position);
84                         KnotSequence.Add(pointplaceController.degree +           ↗
85                             4);//TODO
86                         t_value = KnotSequence.Count / 2f; //TODO
87                     }
88                 }
89             }
90         }
91         if (pointplaceController != null)
92         {
93             if (pointplaceController.degree < 31)
94             {
95                 if (pointplaceController.degree > -1)
96                 {
97                     for (int i = 0; i <= pointplaceController.degree; i++)
98                     {
99                         points[i] = pointplaceController.pointPool           ↗
100                             [i].transform.position;
```

```
101
102     }
103
104     void OnRenderObject()
105     {
106         if (points.Count > 0)
107         {
108             RenderLines(points, baseColor);
109             RenderLines1(points, baseColor);
110         }
111     }
112
113     public float FindJ(List<float> Knot, float tVal)
114     {
115         for (int i = 0; i < Knot.Count - 1; i++)
116         {
117             if (Knot[i] <= tVal && tVal < Knot[i + 1])
118                 return i;
119         }
120         return 0;
121     }
122     virtual public void RenderLines(List<Vector3> points, Color color)
123     {
124         GL.Begin(GL.LINES);
125         material.SetPass(0);
126         for (int i = 0; i < points.Count - 1; i++)
127         {
128             GL.Color(Color.white);
129             GL.Vertex(points[i]);
130
131             //GL.Color(material.color);
132             GL.Vertex(points[i + 1]);
133         }
134         GL.End();
135     }
136
137     virtual public void RenderLines1(List<Vector3> points, Color color)
138     {
139         int size_ = points.Count;
140         List<Vector3> result = new List<Vector3>();
141
142         float a = KnotSequence[Degree];
143         float b = KnotSequence[N - Degree - 1];
144
145         if (Degree < size_ && size_ < (N - Degree))
146             for (float i = a; i < b; i += 0.1f)
147             {
148                 J = FindJ(KnotSequence, i);
149                 result.Add(P_d_J(i, KnotSequence, (int)J, Degree, Degree));
150             }
151
152         GL.Begin(GL.LINES);
```

```
153     material.SetPass(0);
154
155     for (int i = 0; i < result.Count - 1; i++)
156     {
157         GL.Color(Color.red);
158         GL.Vertex(result[i]);
159         GL.Vertex(result[i + 1]);
160     }
161     GL.End();
162 }
163
164 public Vector3 P_d_J(float t, List<float> t_i, int index_, int d, int k)
165 {
166     if (k == 0)
167     {
168         if (index_ < 0 || index_ >= points.Count)
169             return new Vector3(0, 0, 0);
170         return points[index_];
171     }
172
173     Vector3 left = ((t - t_i[index_]) / (t_i[index_ + d - (k - 1)] - t_i[index_])) * P_d_J(t, t_i, index_, d, k - 1);
174     Vector3 right = ((t_i[index_ + d - (k - 1)] - t) / (t_i[index_ + d - (k - 1)] - t_i[index_])) * P_d_J(t, t_i, index_ - 1, d, k - 1);
175
176     return left + right;
177 }
178
179 public void ResetActions()
180 {
181     points.Clear();
182
183
184
185     if (pointplaceController != null)
186         for (int i = 0; i < 30; i++)
187         {
188             pointplaceController.pointPool[i].gameObject.SetActive(false);
189             pointplaceController.pointPool[i].transform.position = new Vector3(0, 0, 0);
190         }
191     if (pointplaceController != null)
192         pointplaceController.degree = -1;
193 }
194
195
196 }
197
```