プログラム
ワークショップIV

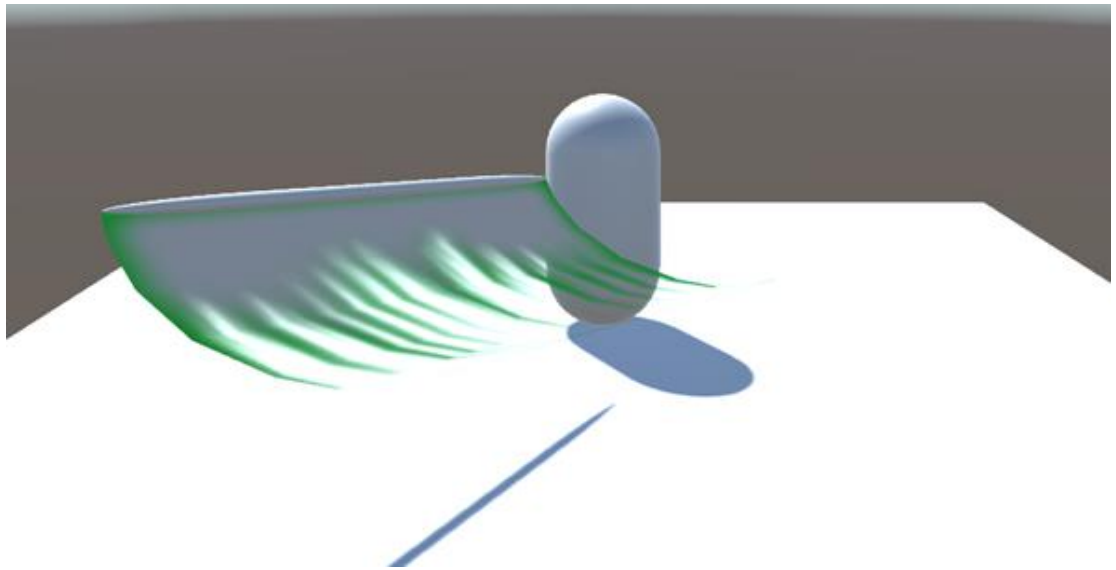
(9) 輪郭抽出・トレイル

軌跡の表現

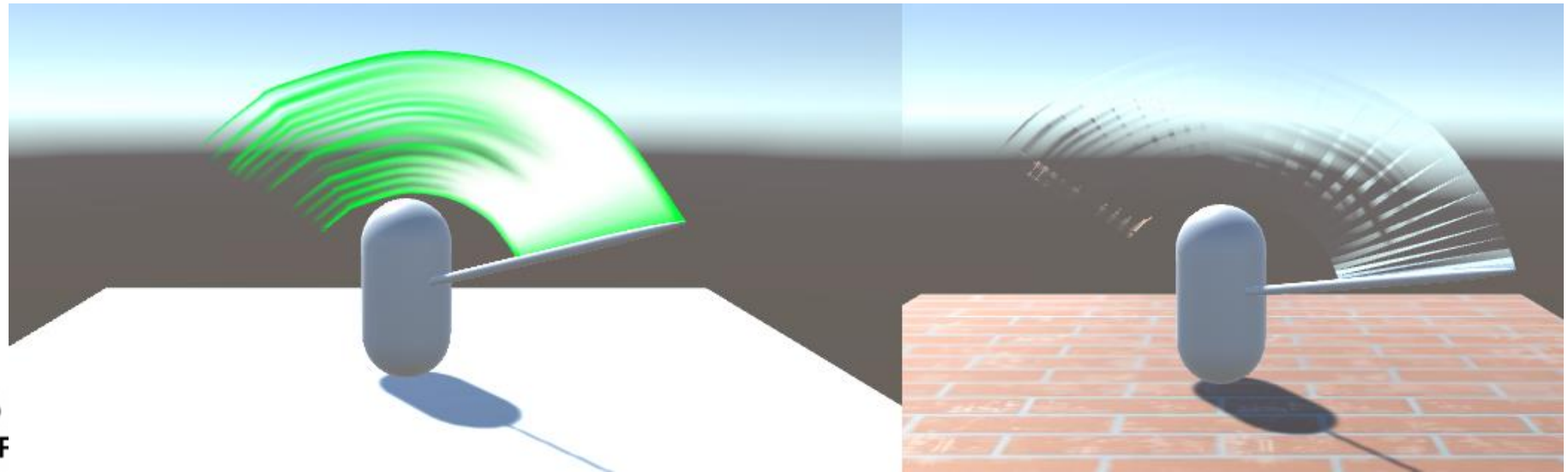
- 物体の動きの履歴を追う

今日やること

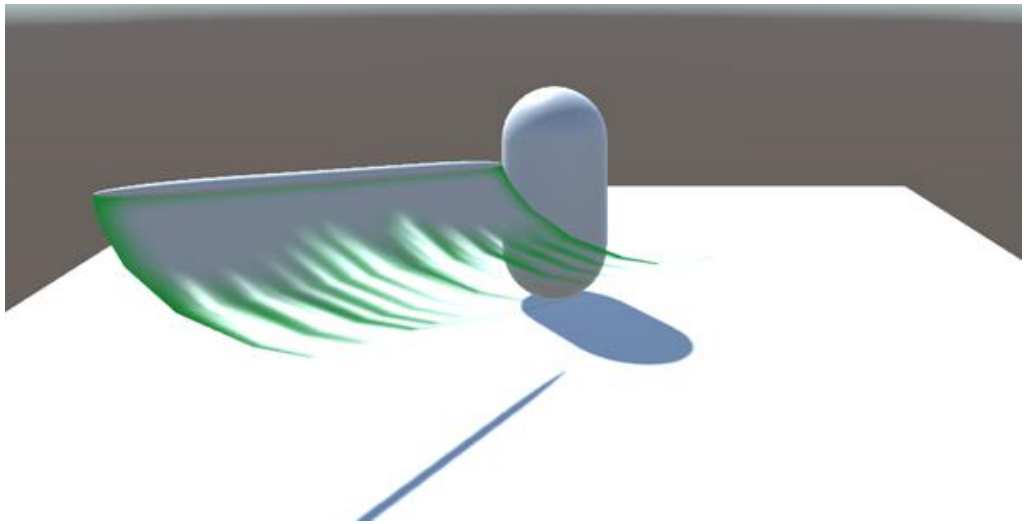
- Trail Renderer

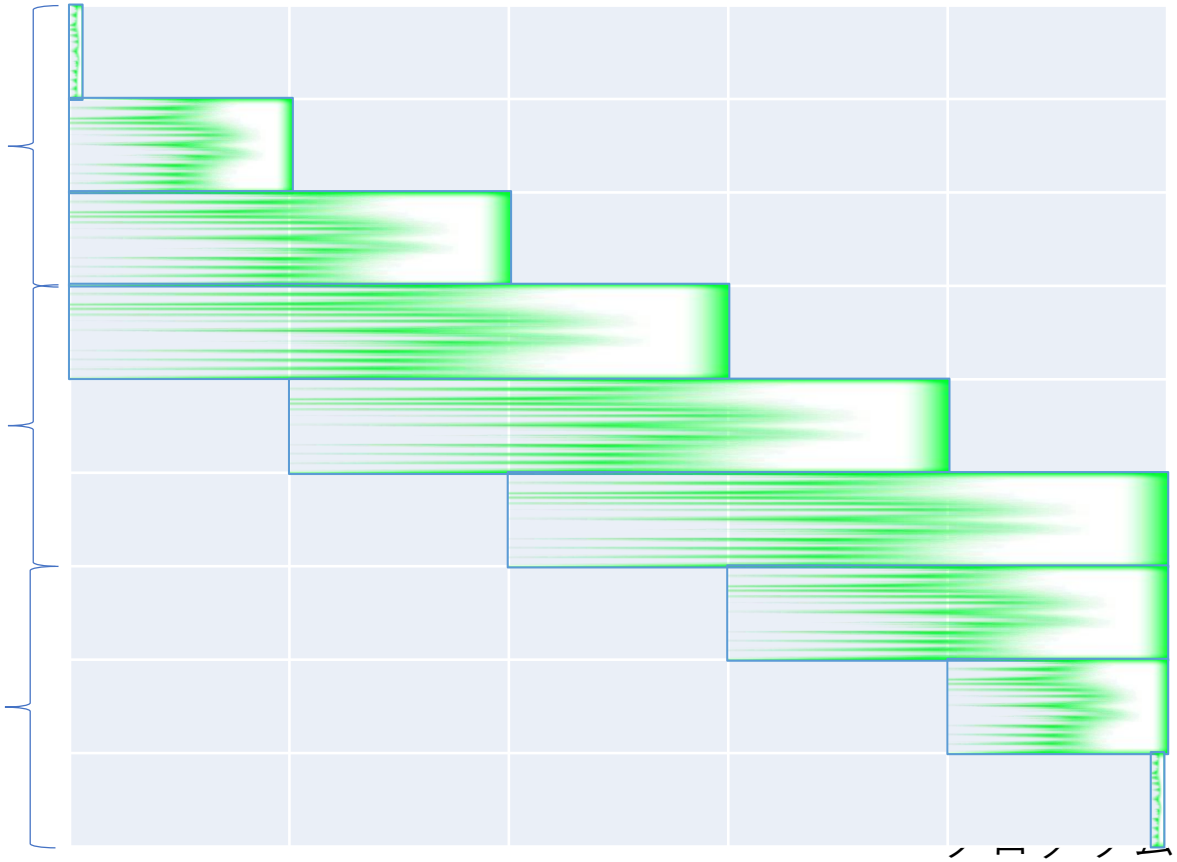


- 自作



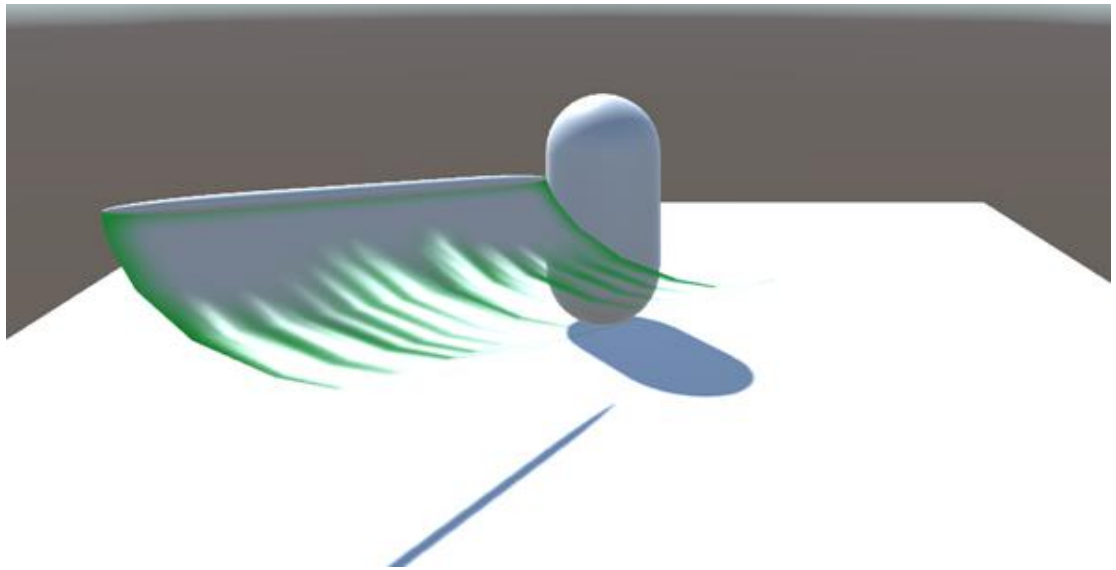
トレイル



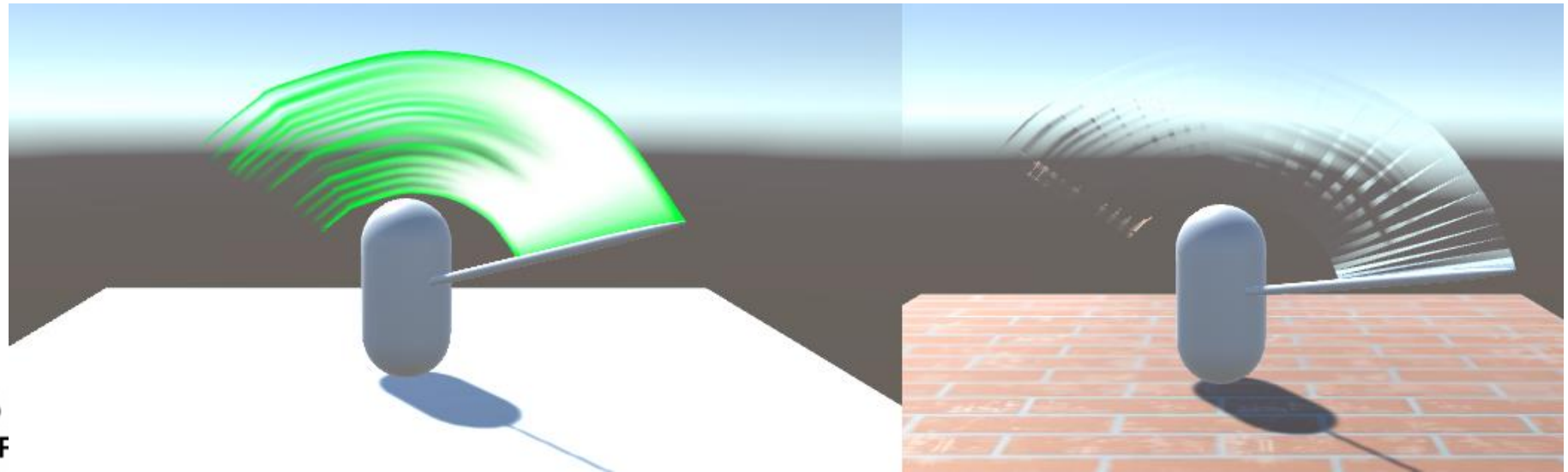


今日やること

- Trail Renderer

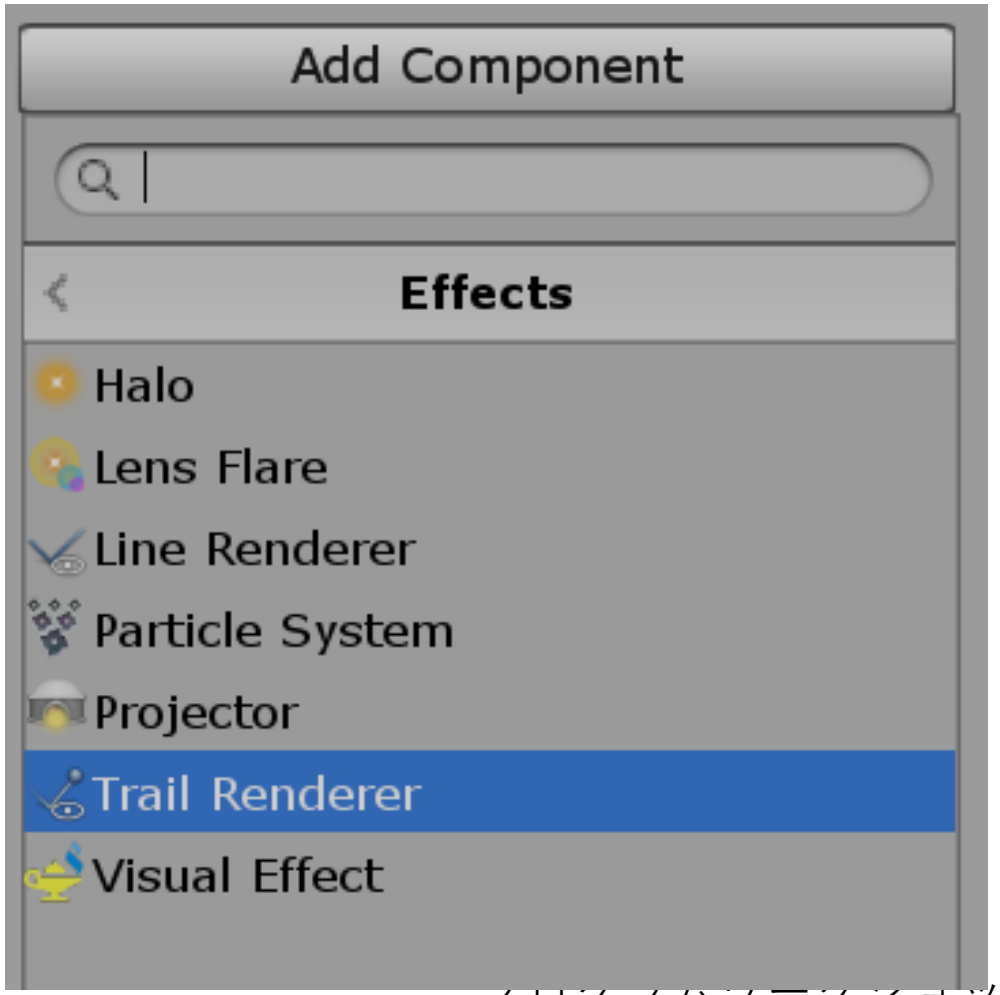


- 自作

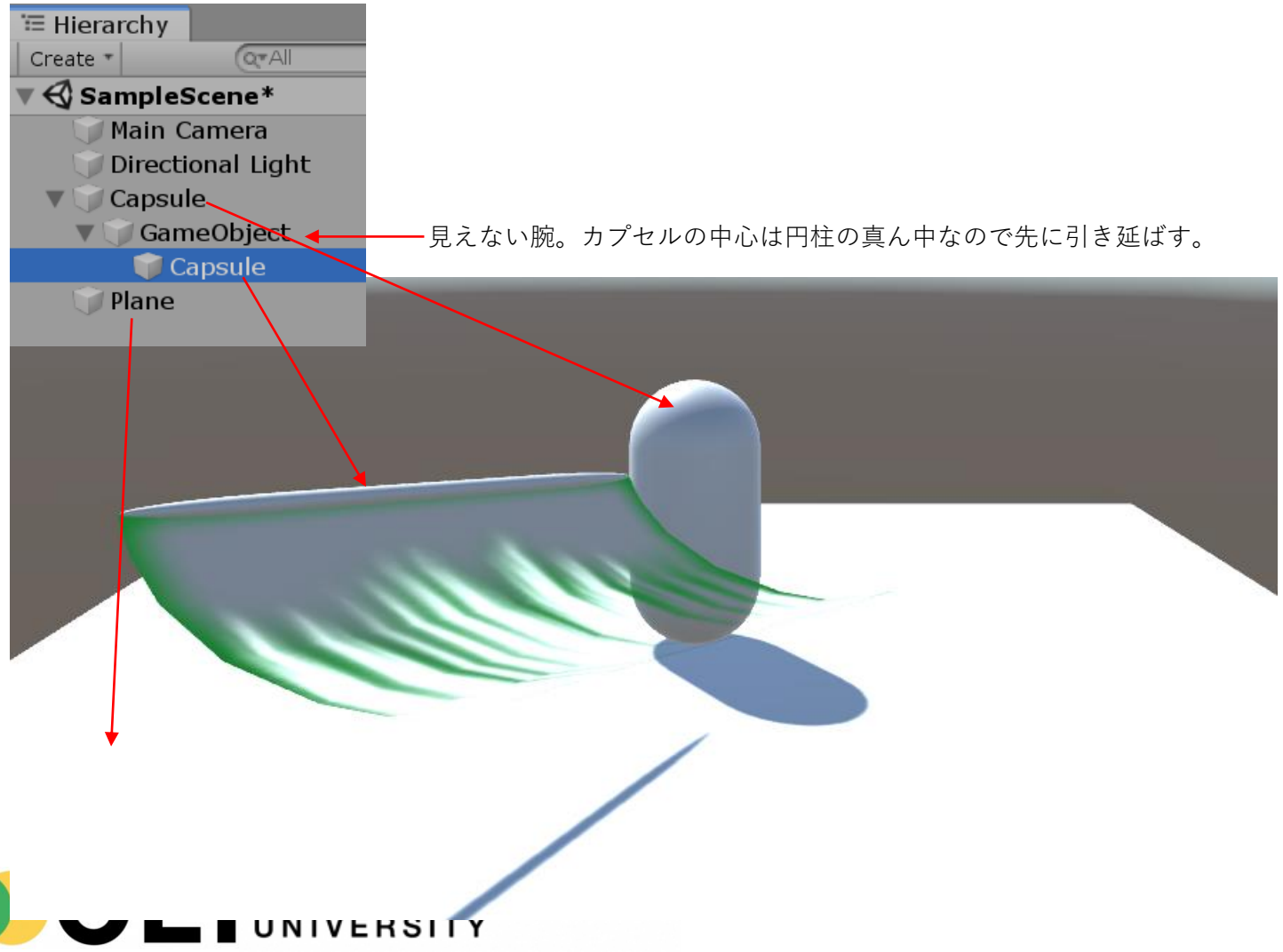


Trail Renderer

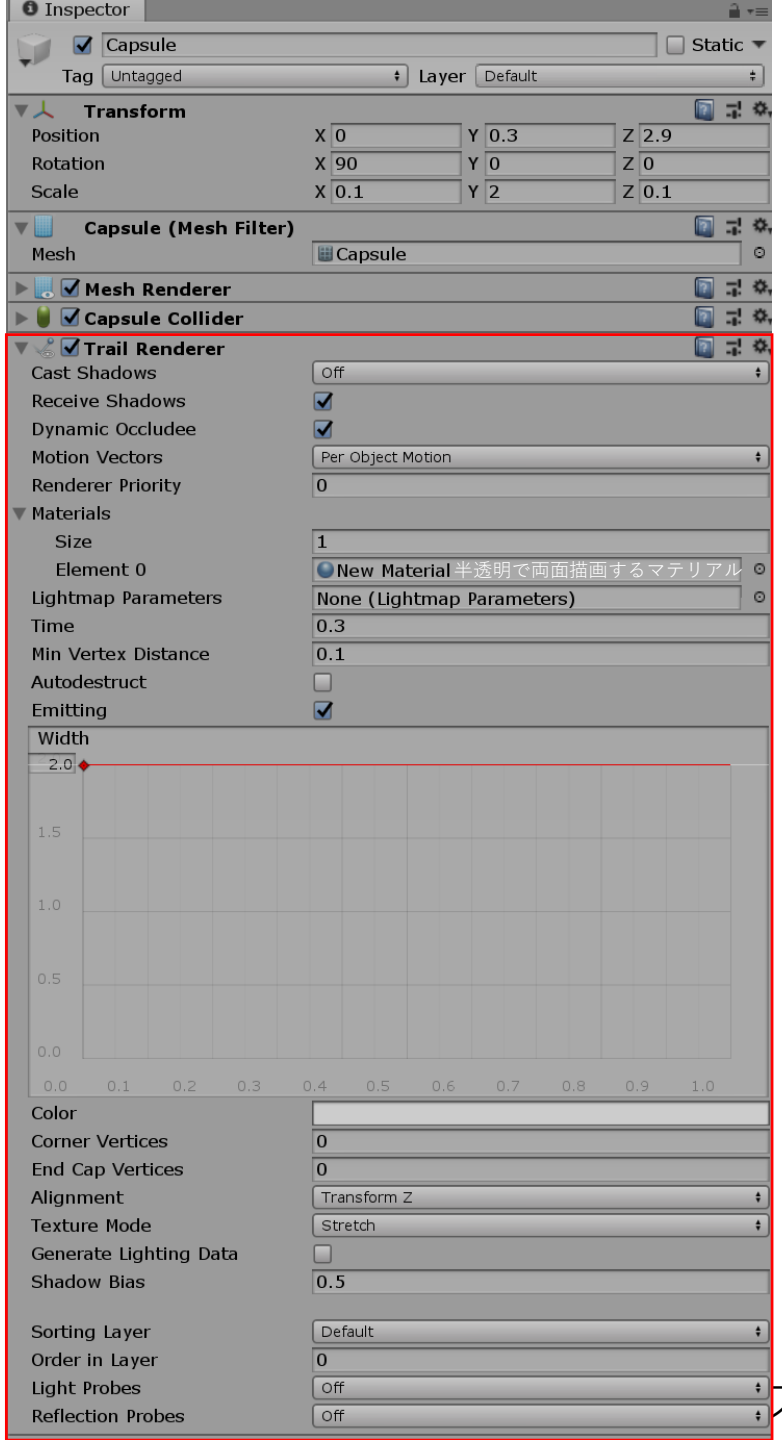
- エフェクトのコンポーネント
- 物体に張り付けると、その動きを追うエフェクトを発生させることができる



使い方



見えない腕。カプセルの中心は円柱の真ん中なので先に引き延ばす。



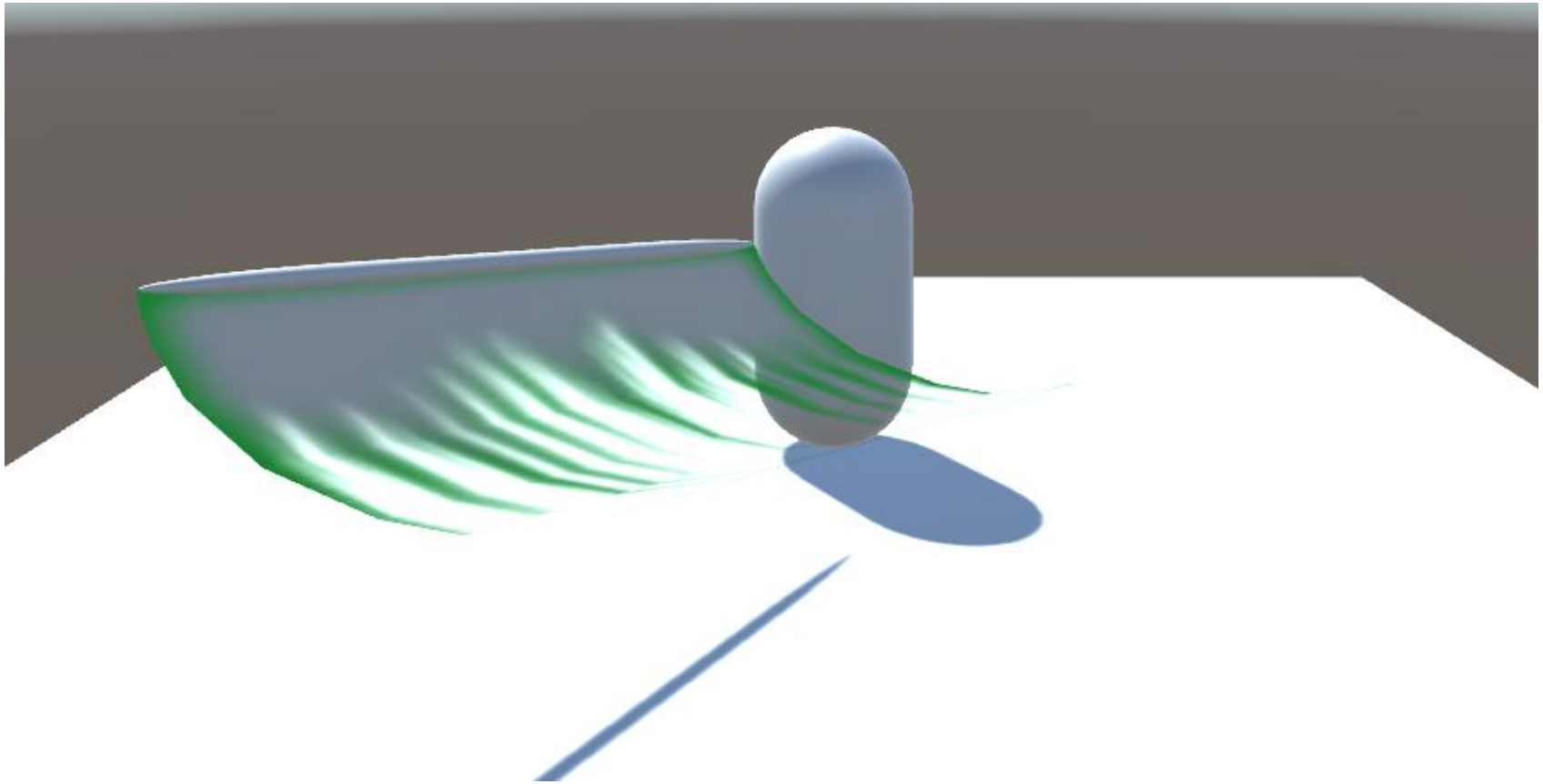
剣の振り方

- 今回はドラッグした点をカメラから5.0離れた距離に置いてそちらを剣先が向くように回転

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

0 references
public class PlayerCOntroller : MonoBehaviour
{
    public float speed = 3.0f;
    GameObject sword;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        sword = transform.GetChild(0).gameObject;
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        // 前後左右
        if (Input.GetKey("up"))...
        if (Input.GetKey("down"))...
        if (Input.GetKey("right"))...
        if (Input.GetKey("left"))...

        Swipe();
    }

    1 reference
    void Swipe()
    {
        if (Input.GetMouseButton(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            Vector3 target_pos = ray.GetPoint(5.0f);

            sword.transform.LookAt(target_pos);
        }
    }
}
```

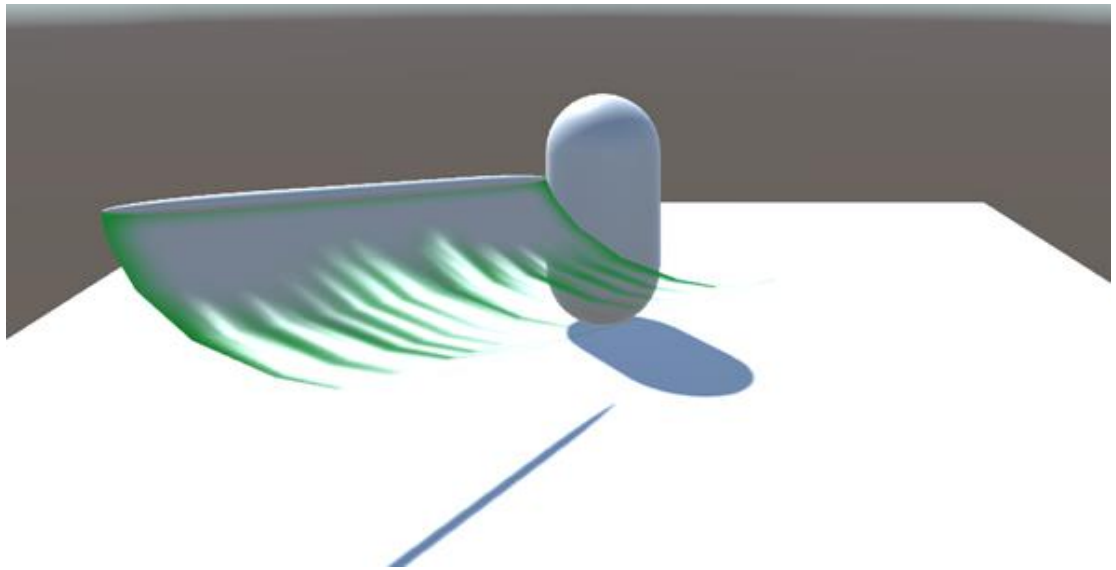
やってみよう

- なるべくかっこよく剣を振ろう

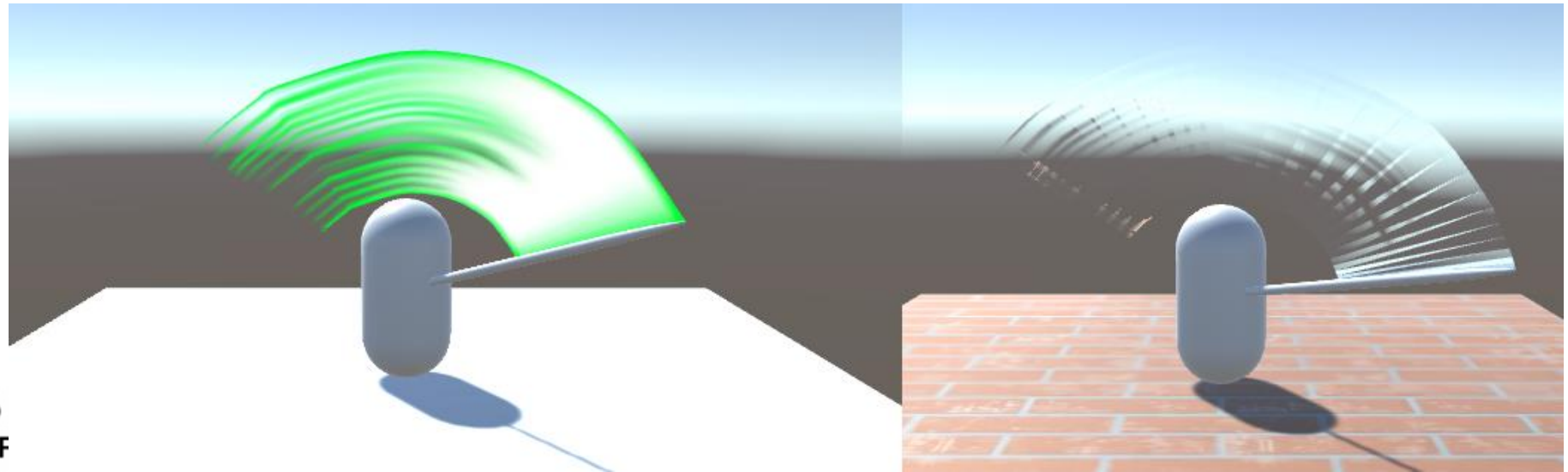


今日やること

- Trail Renderer

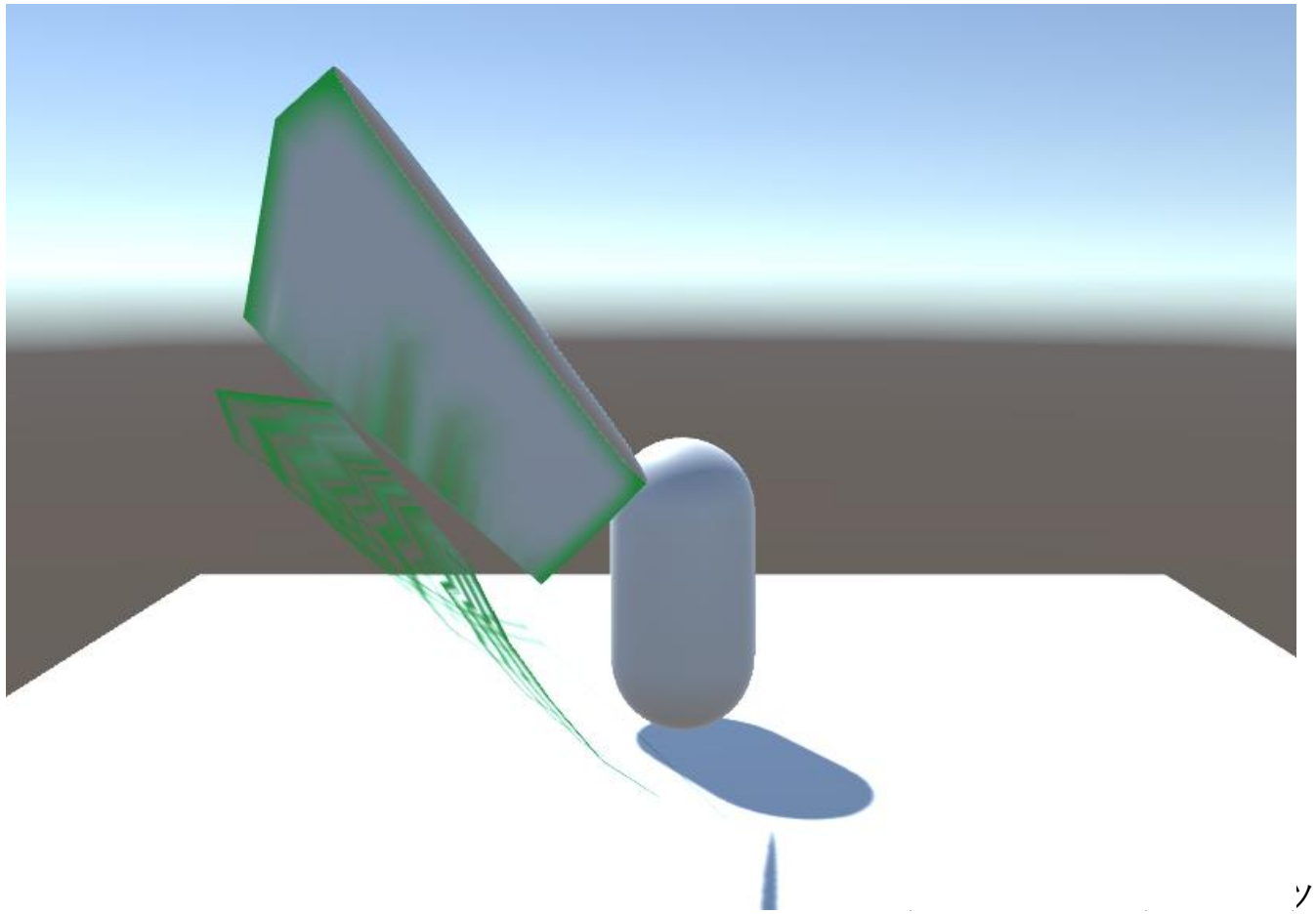


- 自作



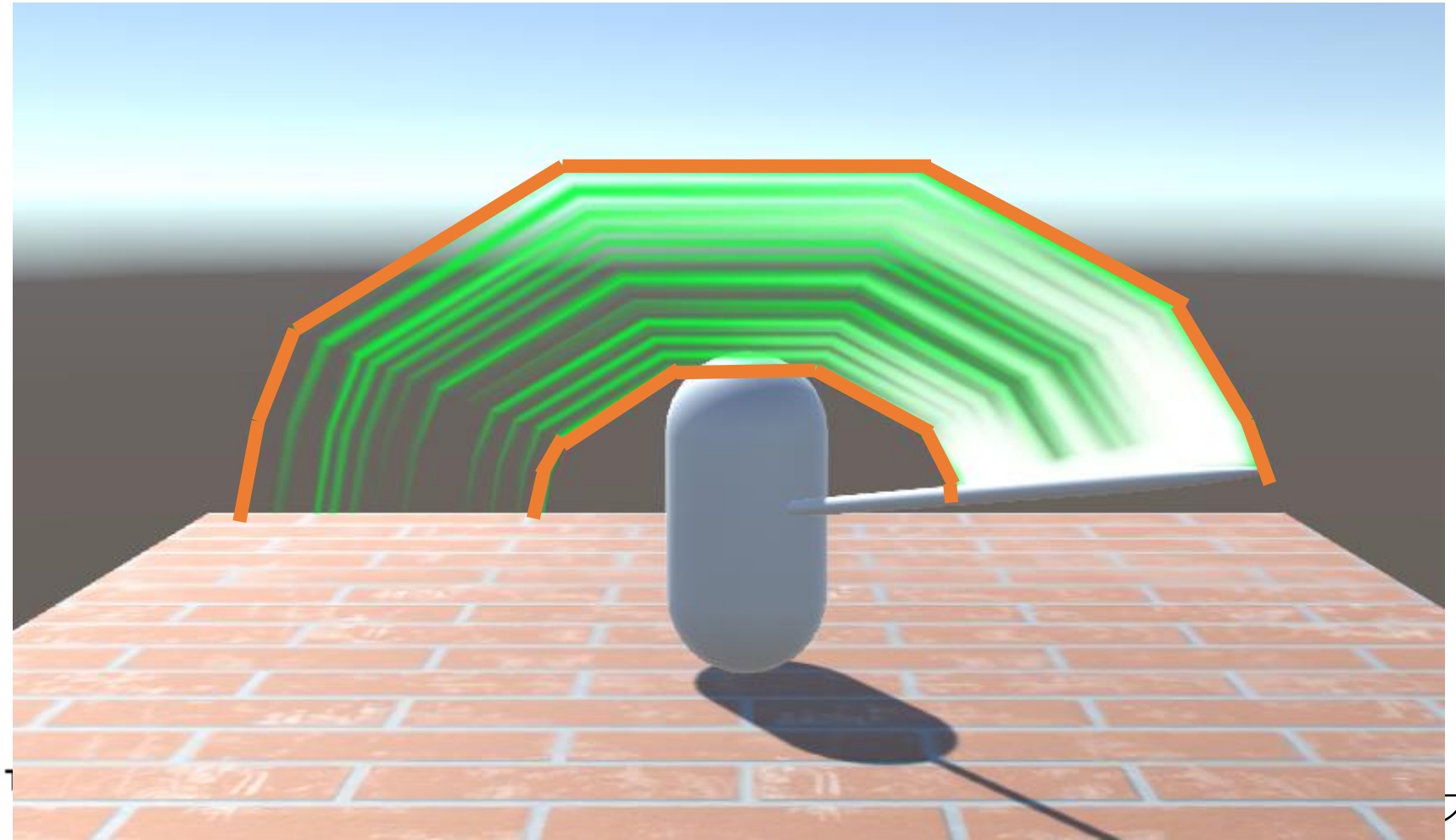
欠点

- 表示が怪しいときがある



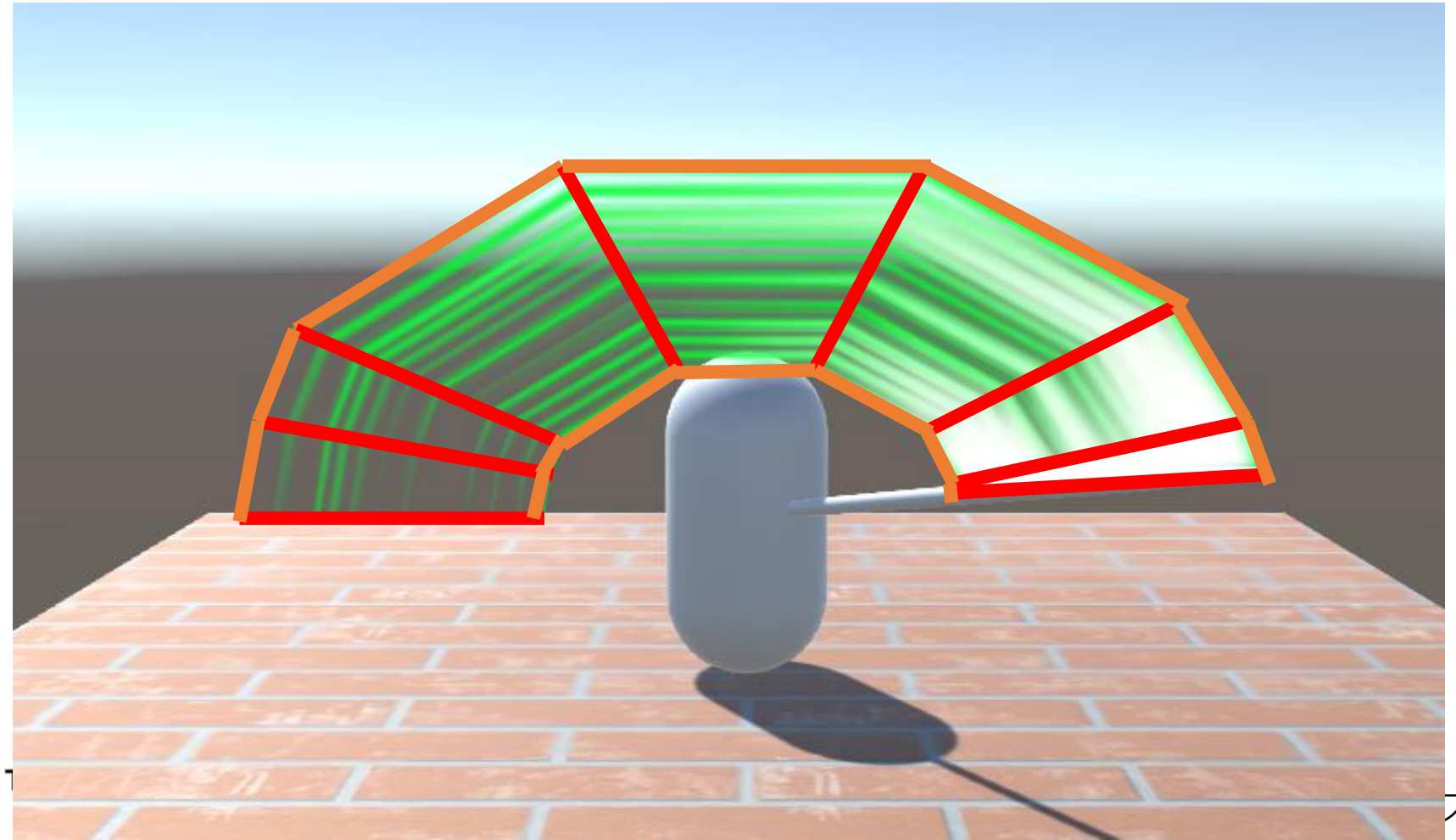
自分でポリゴンを描画してみる

- 先端と末端の位置を追跡する



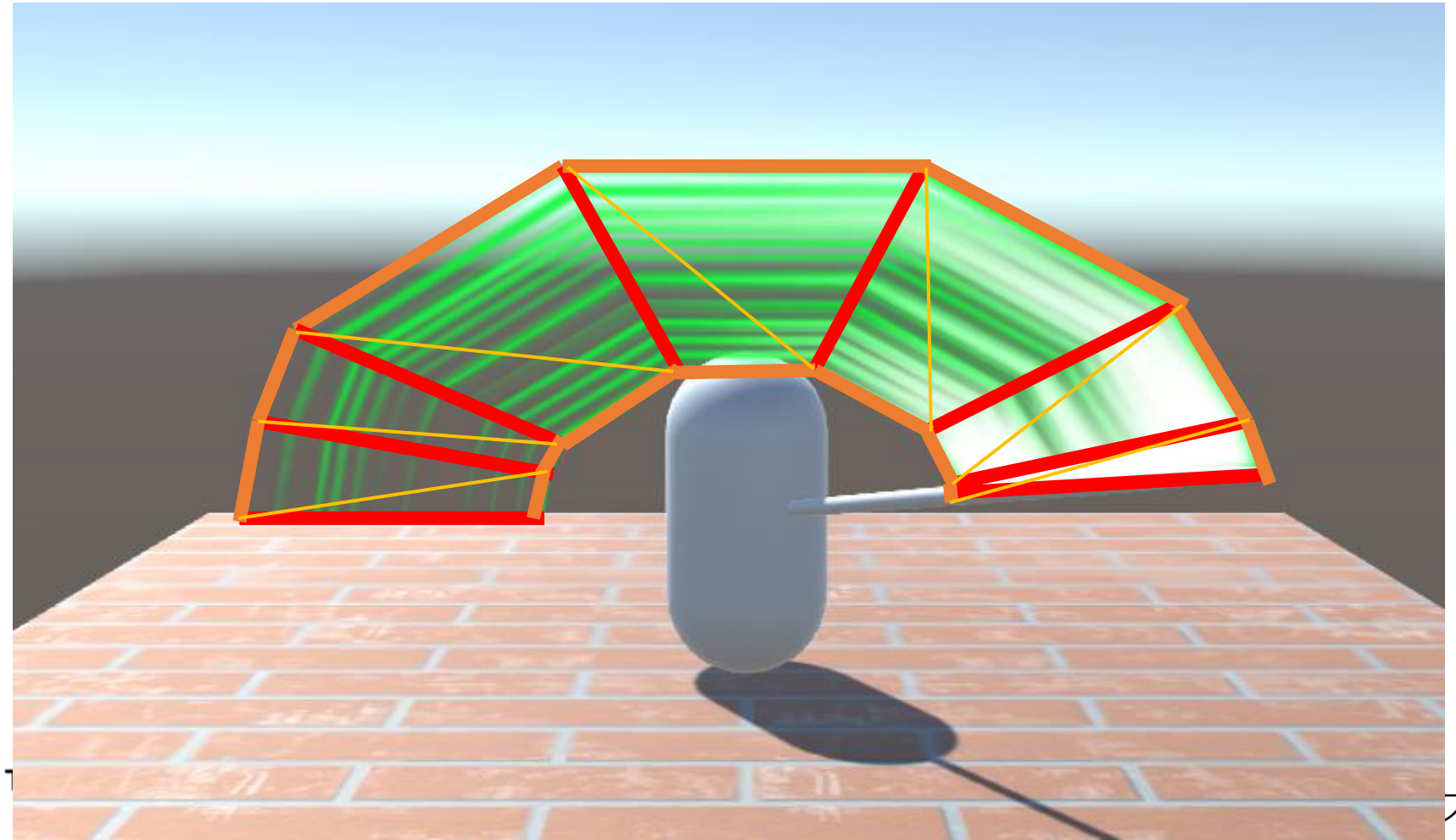
自分でポリゴンを描画してみる

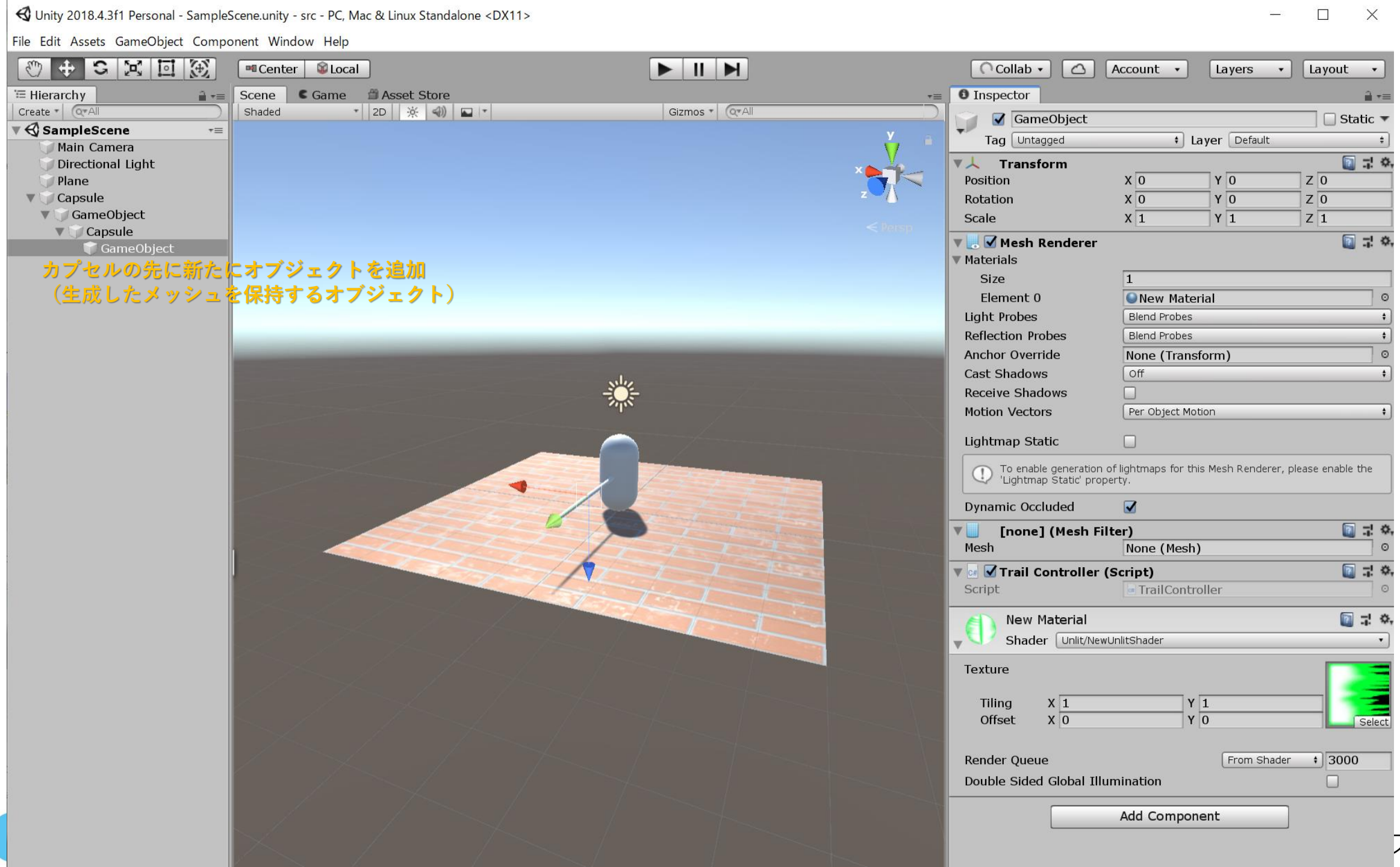
- つなげて四角形を作る



自分でポリゴンを描画してみる

- 三角形で描画




Unity 2018.4.3f1 Personal - SampleScene.unity - src - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Hierarchy
Create
SampleScene
Main Camera
Directional Light
Plane
Capsule
GameObject
Capsule
GameObject
カプセルの先に新たにオブジェクトを追加
（生成したメッシュを保持するオブジェクト）
Scene Game Asset Store
Shaded 2D Gizmos
Persp
Inspector
GameObject
Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Mesh Renderer
Materials
Size 1
Element 0 New Material
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Cast Shadows Off
Receive Shadows
Motion Vectors Per Object Motion
Lightmap Static
To enable generation of lightmaps for this Mesh Renderer, please enable the 'Lightmap Static' property.
Dynamic Occluded
[none] (Mesh Filter)
Mesh None (Mesh)
Trail Controller (Script)
Script TrailController
New Material
Shader Unlit/NewUnlitShader
Texture
Tiling X 1 Y 1
Offset X 0 Y 0
Select
Render Queue From Shader 3000
Double Sided Global Illumination
Add Component

TrailController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class TrailController : MonoBehaviour
{
    private const int FRAME_MAX = 10;
    private List<Vector3> Points0 = new List<Vector3>();// 根本
    private List<Vector3> Points1 = new List<Vector3>();// 先端

    private Mesh mesh;

    // Start is called before the first frame update
    void Start()
    {
        mesh = GetComponent<MeshFilter>().mesh;
    }

    // Update is called once per frame
    void Update()
    {
        if (FRAME_MAX <= Points0.Count)
        {
            Points0.RemoveAt(0);
            Points1.RemoveAt(0);
        }

        Points0.Add(transform.position);
        Points1.Add(transform.TransformPoint(new Vector3(0.0f, 1.0f, 0.0f)));

        if (2 <= Points0.Count)// エッジ一つでは帯は作れない
        {
            CreateMesh();
        }
    }
    private void CreateMesh()
    {
        mesh.Clear();  //  メッシュのクリア

        // メモリ確保
        int n = Points0.Count;
        Vector3[] vertexArray = new Vector3[2 * n];
        Vector2[] uvArray = new Vector2[2 * n];
        int[] indexArray = new int[6 * (n - 1)];

        int idV = 0, idI = 0;
        float dUv = 1.0f / ((float)n - 1.0f);
        for (int i = 0; i < n; i++)
        {
            // 頂点座標
            vertexArray[idV + 0] = transform.InverseTransformPoint(Points0[i]);
            vertexArray[idV + 1] = transform.InverseTransformPoint(Points1[i]);
            // UV座標
            uvArray[idV + 0].x =
            uvArray[idV + 1].x = 1.0f - dUv * (float)i;
            uvArray[idV + 0].y = 0.0f;
            uvArray[idV + 1].y = 1.0f;

            // インデックス
            if (i == n - 1) break;// 最後の辺では帯を作らない
            indexArray[idI + 0] = idV;
            indexArray[idI + 1] = idV + 1;
            indexArray[idI + 2] = idV + 2;
            indexArray[idI + 3] = idV + 2;
            indexArray[idI + 4] = idV + 1;
            indexArray[idI + 5] = idV + 3;

            idV += 2;
            idI += 6;
        }

        mesh.vertices = vertexArray;
        mesh.uv = uvArray;
        mesh.triangles = indexArray;

        mesh.RecalculateBounds();
        mesh.RecalculateNormals();
    }
```

位置を保持

一番古い位置を削除

位置を更新

TrailController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class TrailController : MonoBehaviour
{
    private const int FRAME_MAX = 10;
    private List<Vector3> Points0 = new List<Vector3>();// 根本
    private List<Vector3> Points1 = new List<Vector3>();// 先端

    private Mesh mesh;

    // Start is called before the first frame update
    void Start()
    {
        mesh = GetComponent<MeshFilter>().mesh;
    }

    // Update is called once per frame
    void Update()
    {
        if (FRAME_MAX <= Points0.Count)
        {
            Points0.RemoveAt(0);
            Points1.RemoveAt(0);
        }

        Points0.Add(transform.position);
        Points1.Add(transform.TransformPoint(new Vector3(0.0f, 1.0f, 0.0f)));

        if (2 <= Points0.Count)// エッジ一つでは帯は作れない
        {
            CreateMesh();
        }
    }
```

メッシュを生成

先端と末端を
古いものから追加

```
    private void CreateMesh()
    {
        mesh.Clear();  //  メッシュのクリア

        // メモリ確保
        int n = Points0.Count;
        Vector3[] vertexArray = new Vector3[2 * n];
        Vector2[] uvArray = new Vector2[2 * n];
        int[] indexArray = new int[6 * (n - 1)];

        int idV = 0, idI = 0;
        float dUv = 1.0f / ((float)n - 1.0f);
        for (int i = 0; i < n; i++)
        {
            // 頂点座標
            vertexArray[idV + 0] = transform.InverseTransformPoint(Points0[i]);
            vertexArray[idV + 1] = transform.InverseTransformPoint(Points1[i]);
            // UV座標
            uvArray[idV + 0].x =
            uvArray[idV + 1].x = 1.0f - dUv * (float)i;
            uvArray[idV + 0].y = 0.0f;
            uvArray[idV + 1].y = 1.0f;

            // インデックス
            if (i == n - 1) break;// 最後の辺では帯を作らない
            indexArray[idI + 0] = idV;
            indexArray[idI + 1] = idV + 1;
            indexArray[idI + 2] = idV + 2;
            indexArray[idI + 3] = idV + 2;
            indexArray[idI + 4] = idV + 1;
            indexArray[idI + 5] = idV + 3;

            idV += 2;
            idI += 6;
        }

        mesh.vertices = vertexArray;
        mesh.uv = uvArray;
        mesh.triangles = indexArray;

        mesh.RecalculateBounds();
        mesh.RecalculateNormals();
    }
}
```

配列を確保

ローカル座標に変換

U座標：0:新しい～1:古い
(Trail Rendererに合わせた)

データをメッシュに渡す

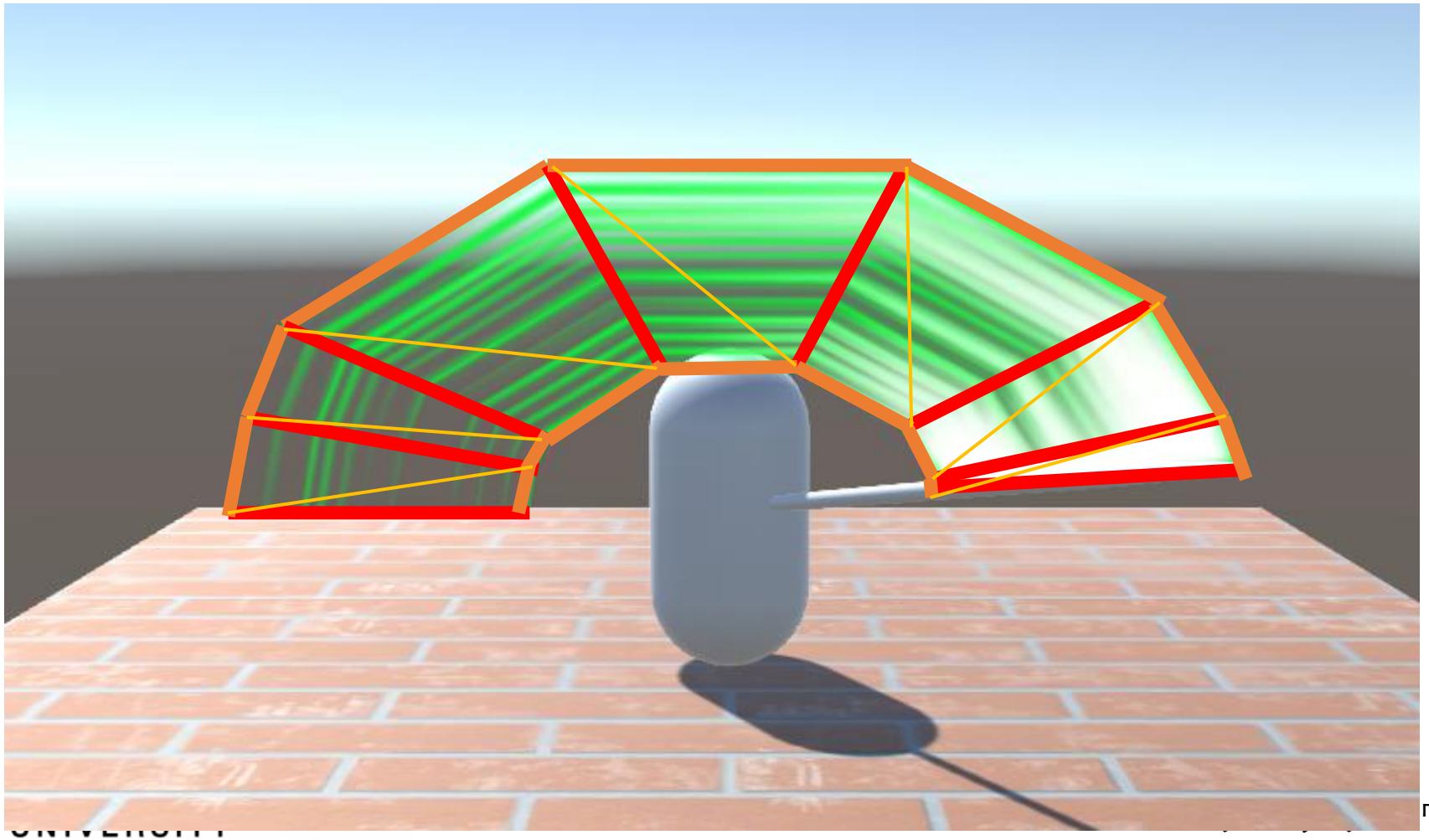
TrailController.cs

```cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class TrailController : MonoBehaviour
{
    private const int FRAME_MAX = 10;
    private List<Vector3> Points0 = new List<Vector3>();// 根本
    private List<Vector3> Points1 = new List<Vector3>();// 先端

    private Mesh mesh;

    // Start is called before the first frame update
    void Start()
    {
        mesh = GetComponent<MeshFilter>().mesh;
    }

    // Update is called once per frame
    void Update()
    {
        if (FRAME_MAX <= Points0.Count)
        {
            Points0.RemoveAt(0);
            Points1.RemoveAt(0);
        }

        Points0.Add(transform.position);
        Points1.Add(transform.TransformPoint(new Vector3(0.0f, 1.0f, 0.0f)));

        if (2 <= Points0.Count)// エッジ一つでは帯は作れない
        {
            CreateMesh();
        }
    }
    private void CreateMesh()
    {
        mesh.Clear();  //  メッシュのクリア

        // メモリ確保
        int n = Points0.Count;
        Vector3[] vertexArray = new Vector3[2 * n];
        Vector2[] uvArray = new Vector2[2 * n];
        int[] indexArray = new int[6 * (n - 1)];

        int idV = 0, idI = 0;
        float dUv = 1.0f / ((float)n - 1.0f);
        for (int i = 0; i < n; i++)
        {
            // 頂点座標
            vertexArray[idV + 0] = transform.InverseTransformPoint(Points0[i]);
            vertexArray[idV + 1] = transform.InverseTransformPoint(Points1[i]);
            // UV座標
            uvArray[idV + 0].x =
            uvArray[idV + 1].x = 1.0f - dUv * (float)i;
            uvArray[idV + 0].y = 0.0f;
            uvArray[idV + 1].y = 1.0f;

            // インデックス
            if (i == n - 1) break;// 最後の辺では帯を作らない
            indexArray[idI + 0] = idV;
            indexArray[idI + 1] = idV + 1;
            indexArray[idI + 2] = idV + 2;
            indexArray[idI + 3] = idV + 2;
            indexArray[idI + 4] = idV + 1;
            indexArray[idI + 5] = idV + 3;

            idV += 2;
            idI += 6;
        }

        mesh.vertices = vertexArray;
        mesh.uv = uvArray;
        mesh.triangles = indexArray;

        mesh.RecalculateBounds();
        mesh.RecalculateNormals();
    }
}
```

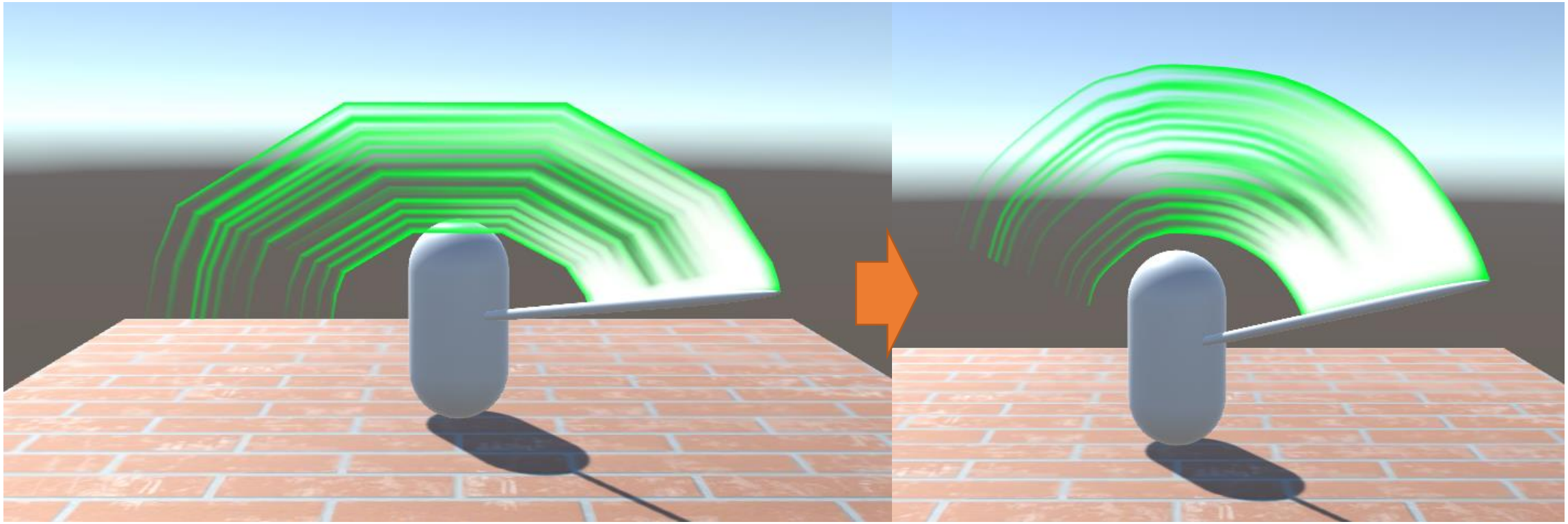
メッシュを生成

二つの三角形の集合で
ポリゴンを構築

結果

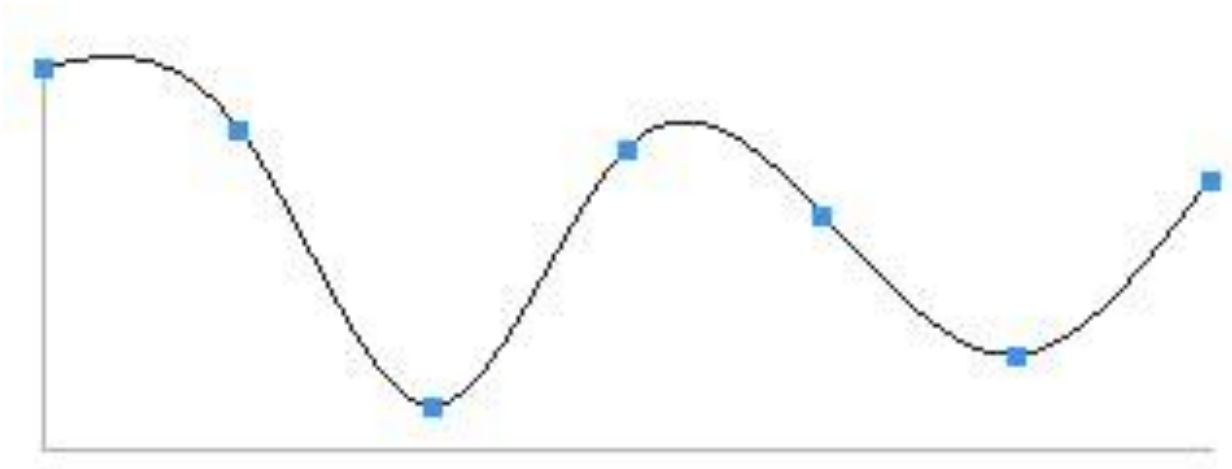


なめらかにしよう



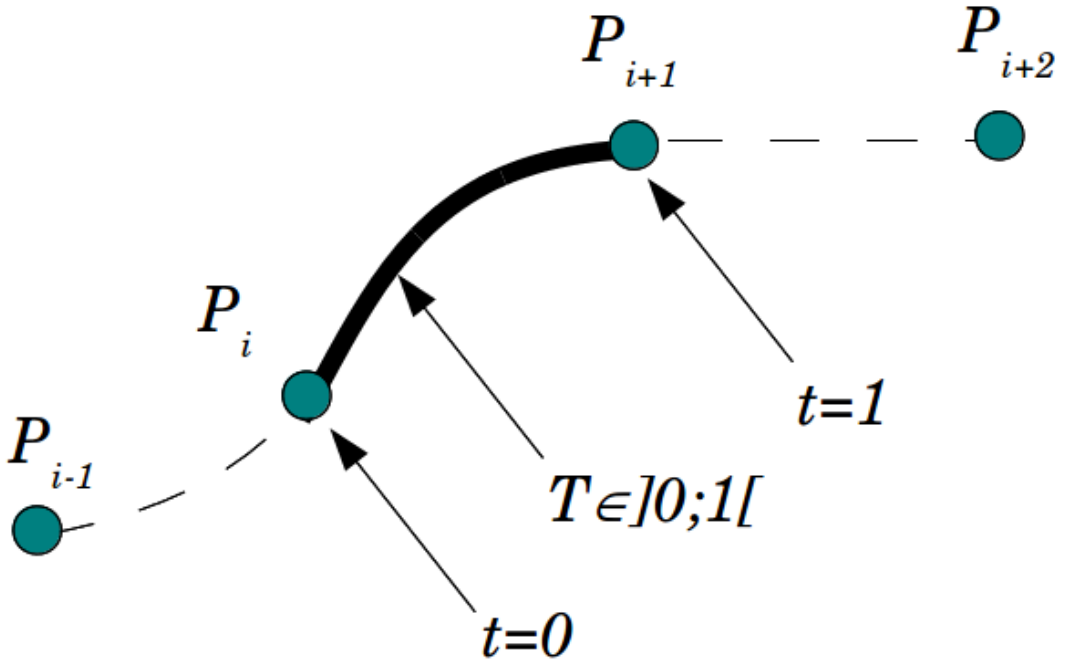
スプライン曲線

- 点列から、その間の曲線を推定する



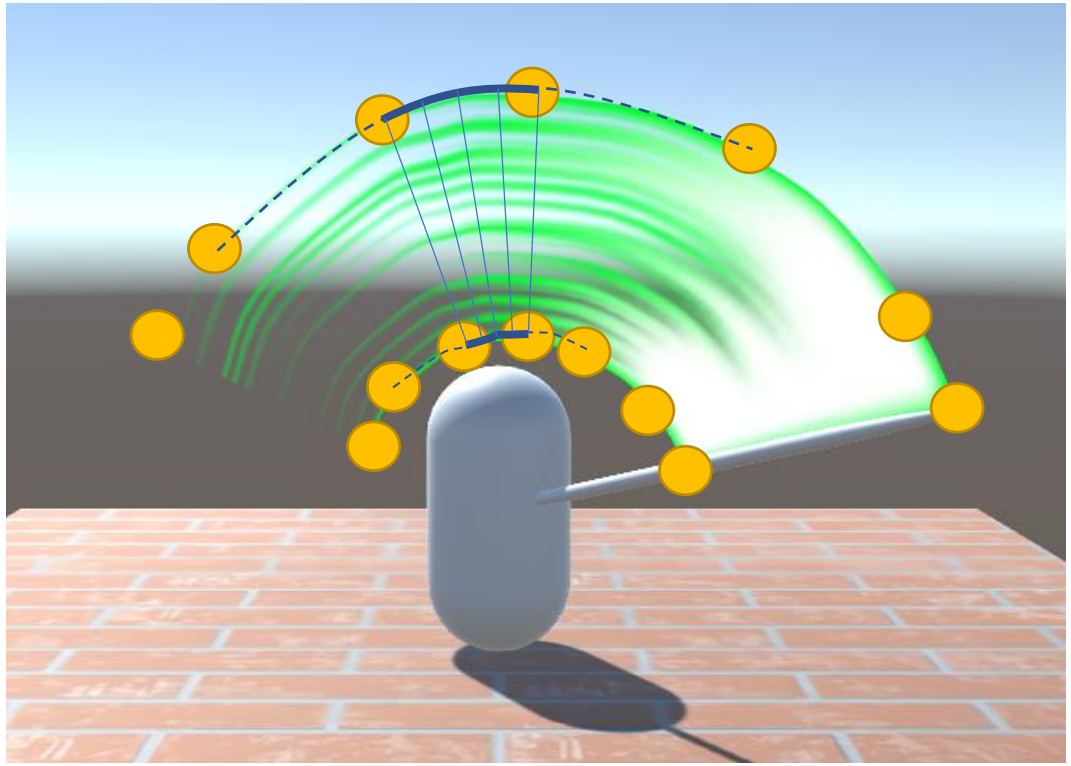
Catmull-Rom曲線

- 4つの点列を指定すると、パラメータが0から1に動くにしたがって、2番目の位置から３番目の位置まで滑らかに動く
- 一つずつずらして表示することで、なめらかな連続曲線が作れる



Catmull-Rom曲線の使い方

- 軌跡の四角形の頂点をCatmull-Rom曲線の制御点にして曲線上の点で四角形を分割



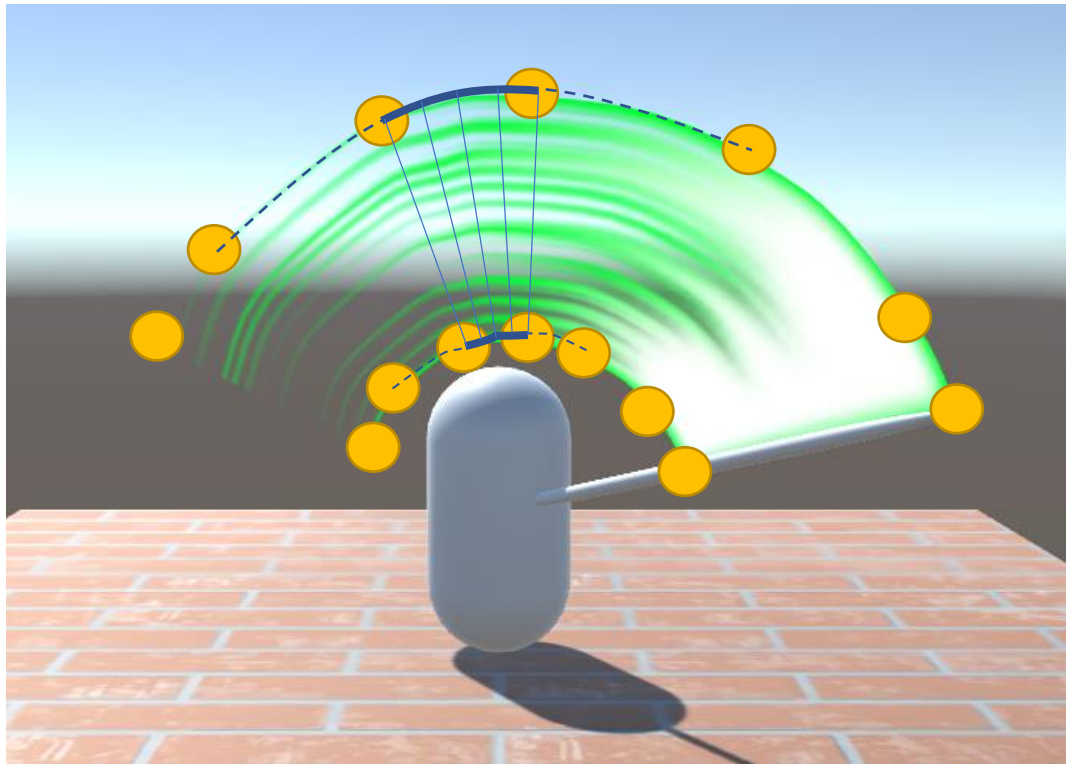
```
95      private void CreateMeshSubDevided(int s/* 分割数 */)
96      {
97          mesh.Clear();  //  メッシュのクリア
98
99          // メモリ確保
100         int n = Points0.Count;
101         Vector3[] vertexArray = new Vector3[2 * s * (n - 1) + 2];
102         Vector2[] uvArray     = new Vector2[2 * s * (n - 1) + 2];
103         int[] indexArray = new int[6 * s * (n - 1)];
104
105         int idV = 0, idI = 0;
106         float dUv = 1.0f / ((float)s * (float)(n - 1.0f));
107         int id0 = 0, id1 = 0, id2 = 1, id3 = 2;
108         if (n - 1 < id3) id3 = n - 1;
109
110         for (int i = 0; i < n - 1; i++)// N.B. 最後の一個手前まで
111         {
112             for (int j = 0; j < s; j++)
113             {
114                 // 頂点座標
115                 float t = (float)j / (float)s;
116                 Vector3 p0 = CatmullRom(Points0[id0], Points0[id1], Points0[id2], Points0[id3], t);
117                 Vector3 p1 = CatmullRom(Points1[id0], Points1[id1], Points1[id2], Points1[id3], t);
118                 vertexArray[idV + 0] = transform.InverseTransformPoint(p0);
119                 vertexArray[idV + 1] = transform.InverseTransformPoint(p1);
120
121                 // UV座標
122                 uvArray[idV + 0].x =
123                 uvArray[idV + 1].x = 1.0f - dUv * (float)(i*s+j);
124                 uvArray[idV + 0].y = 0.0f;
125                 uvArray[idV + 1].y = 1.0f;
126
127                 // インデックス
128                 indexArray[idI + 0] = idV;
129                 indexArray[idI + 1] = idV + 1;
130                 indexArray[idI + 2] = idV + 2;
131                 indexArray[idI + 3] = idV + 2;
132                 indexArray[idI + 4] = idV + 1;
133                 indexArray[idI + 5] = idV + 3;
134
135                 idI += 6;
136                 idV += 2;
137             }
138             if(i != 0) id0++;
139             if(n - 1 < ++id1) id1 = n - 1;
140             if(n - 1 < ++id2) id2 = n - 1;
141             if(n - 1 < ++id3) id3 = n - 1;
142         }
```

細分割の数だけ繰り返す

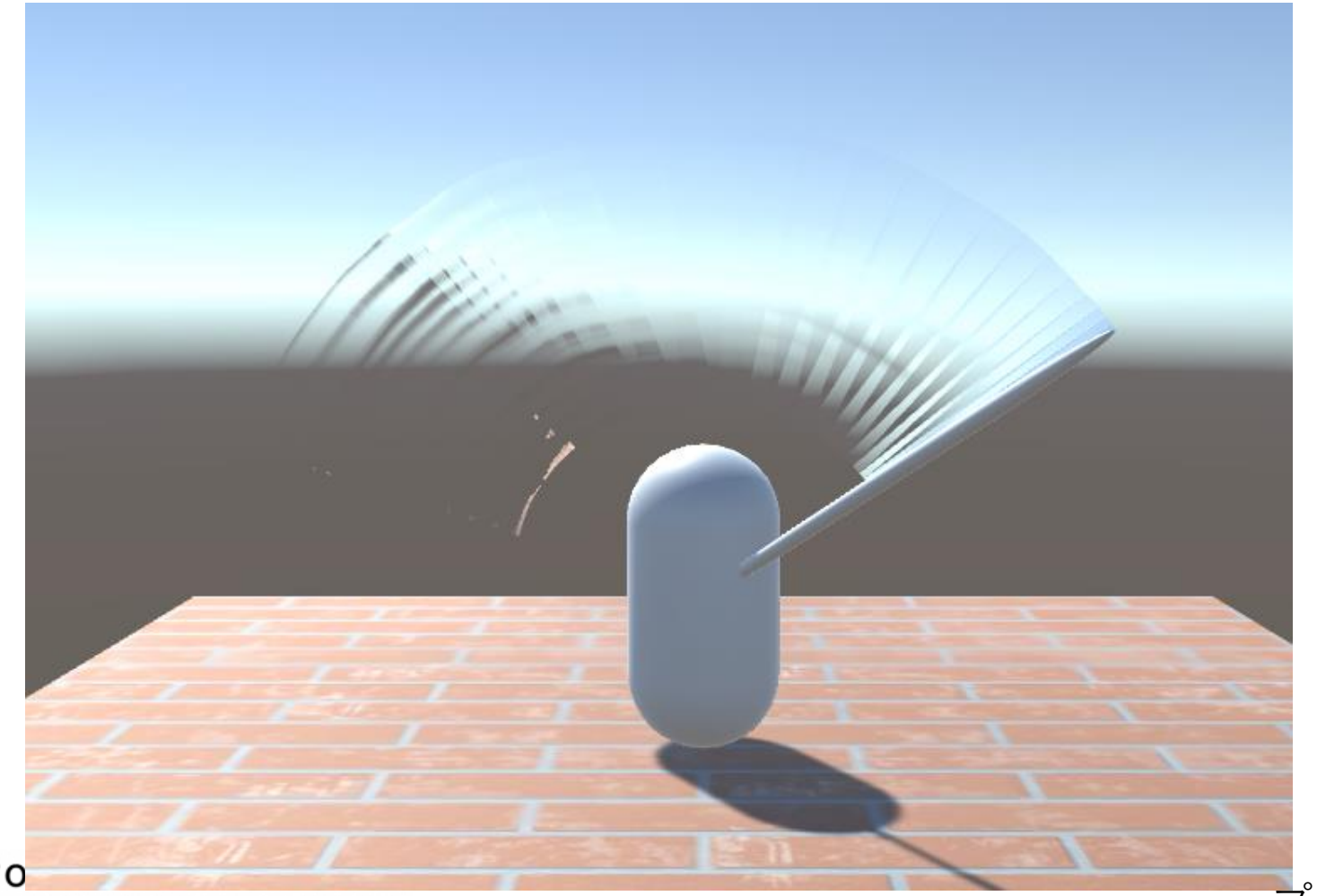
```
144         // 最後の辺
145         vertexArray[idV + 0] = transform.InverseTransformPoint(Points0[n - 1]);
146         vertexArray[idV + 1] = transform.InverseTransformPoint(Points1[n - 1]);
147         uvArray[idV + 0].x = uvArray[idV + 1].x = 0.0f;
148         uvArray[idV + 0].y = 0.0f;
149         uvArray[idV + 1].y = 1.0f;
150
151         mesh.vertices = vertexArray;
152         mesh.uv = uvArray;
153         mesh.triangles = indexArray;
154
155         mesh.RecalculateBounds();
156         mesh.RecalculateNormals();
157     }
```

t=0, 0.25, 0.5, 0.75 での曲線の値で補間

```
39      private Vector3 CatmullRom(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3, float t)
40      {
41          float t2 = t * t;
42          float t3 = t * t2;
43          float c0 = -0.5f * t3 + t2 - 0.5f * t;
44          float c1 = 1.5f * t3 -2.5f * t2 + 1.0f;
45          float c2 = -1.5f * t3 + 2.0f * t2 + 0.5f * t;
46          float c3 = 0.5f * t3 - 0.5f * t2;
47
48          return p0 * c0 + p1 * c1 + p2 * c2 + p3 * c3;
49      }
```

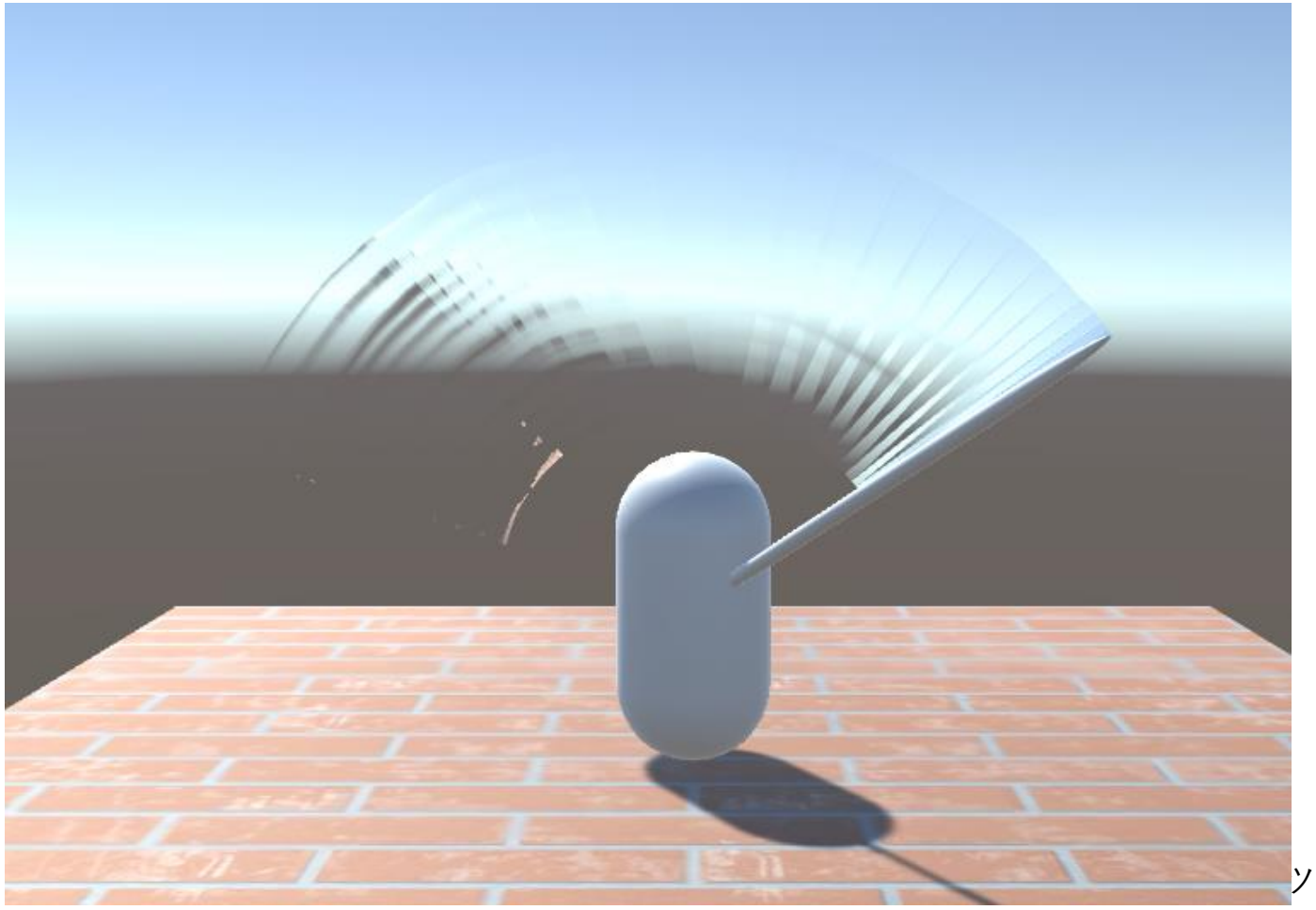


空間をゆがませる



空間をゆがませる

- その時までに描画された結果をテクスチャとして使う
  - 重いのでほどほどに



GrabPass

- テクスチャに結果を格納して読み込む

```
Shader "Unlit/GrabUnlitShader"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Tags { "RenderType" = "Transparent" "Queue" = "Transparent" }
        LOD 100

        Cull off// 両面描画
        Blend SrcAlpha OneMinusSrcAlpha// 半透明合成式

        GrabPass { "_BackgroundTexture" }

        Pass
        {
            struct v2f
            {
                float2 uv : TEXCOORD0;
                float4 grabPos : TEXCOORD1;
                UNITY_FOG_COORDS(1)
                float4 vertex : SV_POSITION;
            };

            sampler2D _MainTex;
            sampler2D _BackgroundTexture;
            float4 _MainTex_ST;

            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                o.uv = TRANSFORM_TEX(v.uv, _MainTex);
                UNITY_TRANSFER_FOG(o,o.vertex);

                o.grabPos = ComputeGrabScreenPos(o.vertex);

                return o;
            }

            fixed4 frag (v2f i) : SV_Target
            {
                // sample the texture
                fixed4 col = tex2D(_MainTex, i.uv);

                fixed3 grab = tex2Dproj(_BackgroundTexture, i.grabPos
                + 100.0 // ずらす強さ
                *  (col.a + 0.5) // αが0でも少しはゆがめる
                * float4(ddx(i.uv.x), ddy(i.uv.x), 0.0, 0.0)// u座標の向きでずらす
                ).rgb;
                col.rgb = grab;

                // apply fog
                UNITY_APPLY_FOG(i.fogCoord, col);
                return col;
            }
            ENDCG
        }
```

格納するテクスチャの指定

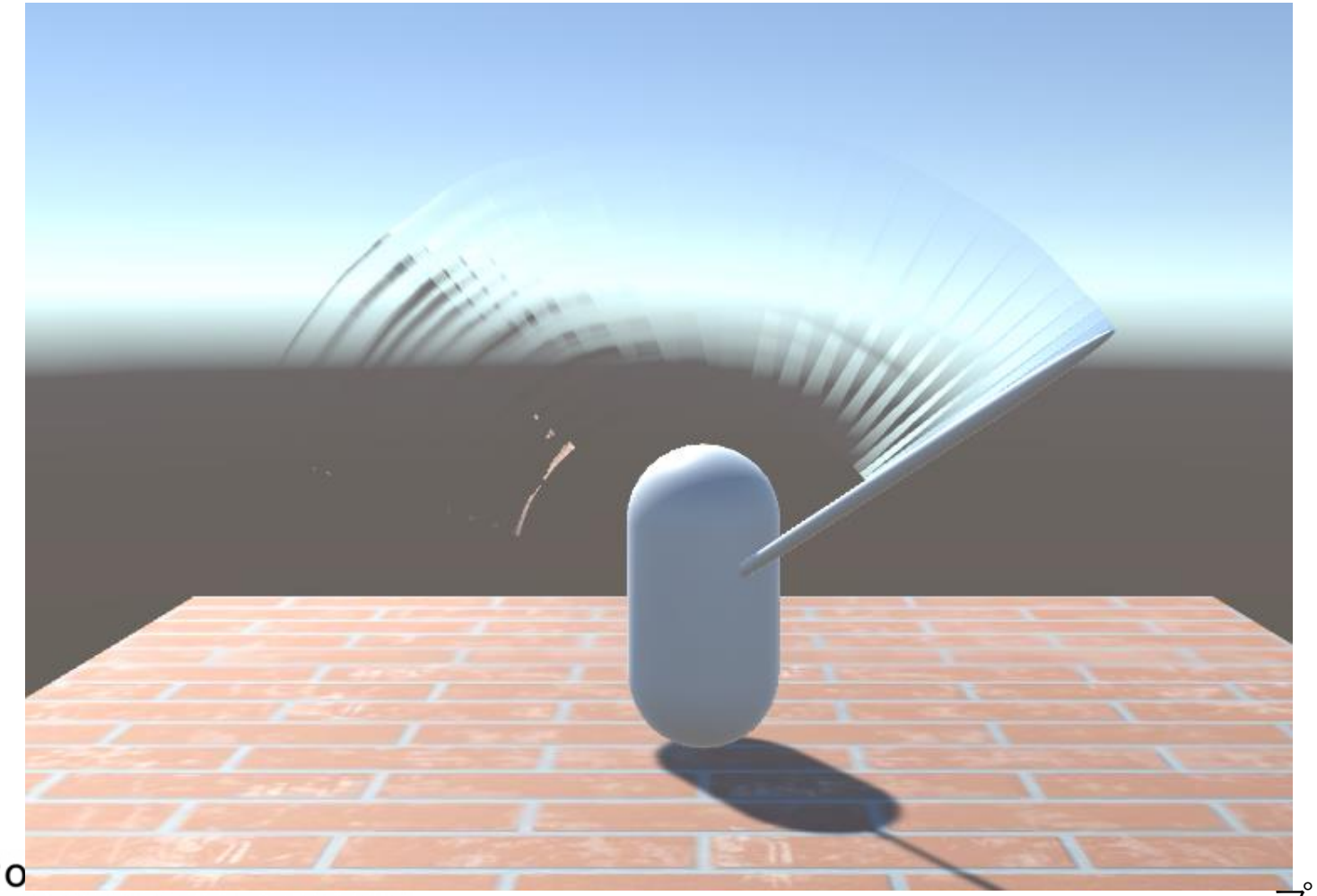
この歪めがないと
現在の場所のテクスチャを描画
（何もないように見える）

頂点の位置からスクリーンの
テクスチャ座標を渡す

格納するテクスチャ

頂点の位置からスクリーンの
テクスチャ座標を計算

結果



やってみよう

- result2.png
  - 軌跡を出してみよう
- result3.png
  - スプライン曲線で補間しよう
- result4.png
  - 空間をゆがませよう
- result5.png
  - いろいろなパラメータを変えて、おれだけの軌跡を描こう