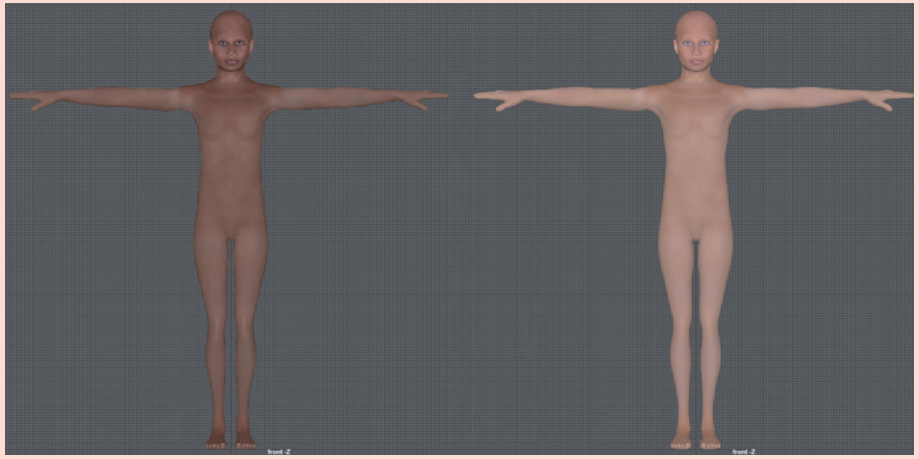

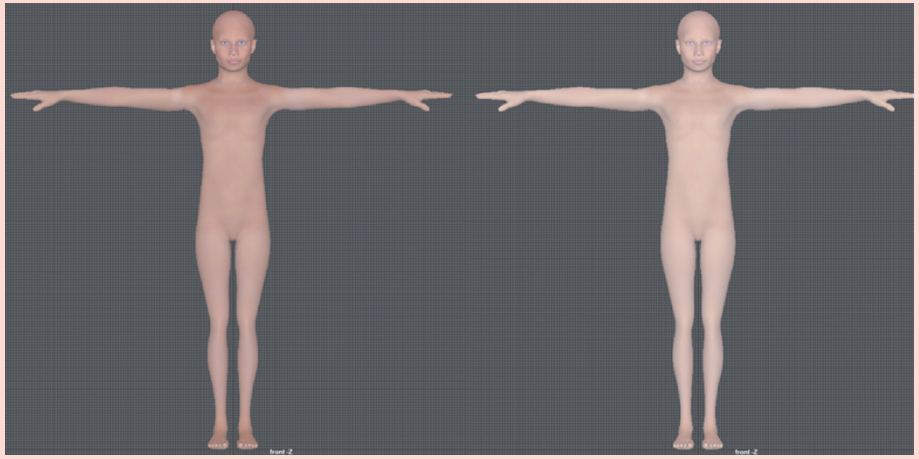

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(MeshFilter))]
public class MeshDeformer : MonoBehaviour {
    Mesh deformingMesh;
    Vector3[] originalVertices, displacedVertices;
    Vector3[] vertexVelocities;
    float uniformScale = 1f;

    public float springForce = 20f;

    public float damping = 5f;
    // Use this for initialization
    void Start () {
        deformingMesh = GetComponent<MeshFilter>().mesh;
        originalVertices = deformingMesh.vertices;
        displacedVertices = new Vector3[originalVertices.Length];
        for (int i = 0; i < originalVertices.Length; i++)
        {
            displacedVertices[i] = originalVertices[i];
        }
        vertexVelocities = new Vector3[originalVertices.Length];
    }

                                // Update is called once per frame
                                void Update () {
        uniformScale = transform.localScale.x;
        for (int i = 0; i < displacedVertices.Length; i++)
        {
            UpdateVertex(i);
        }
        deformingMesh.vertices = displacedVertices;
        deformingMesh.RecalculateNormals();

    }
    public void AddDeformingForce(Vector3 point, float force)
    {
        Debug.DrawLine(Camera.main.transform.position, point);
        point = transform.InverseTransformPoint(point);
        for (int i = 0; i < displacedVertices.Length; i++)
        {
            AddForceToVertex(i, point, force);
        }
    }


    void AddForceToVertex(int i, Vector3 point, float force)
    {
        Vector3 pointToVertex = displacedVertices[i] - point;
        pointToVertex *= uniformScale;
        float attenuatedForce = force / (1f + pointToVertex.sqrMagnitude);
        float velocity = attenuatedForce * Time.deltaTime;
        vertexVelocities[i] += pointToVertex.normalized * velocity;

    }
    void UpdateVertex(int i)
    {
        Vector3 velocity = vertexVelocities[i];
        Vector3 displacement = displacedVertices[i] - originalVertices[i];

        displacement *= uniformScale;
        velocity -= displacement * springForce * Time.deltaTime;
        velocity *= 1f - damping * Time.deltaTime;
        vertexVelocities[i] = velocity;
        displacedVertices[i] += velocity * (Time.deltaTime/uniformScale);

    }

}
```