**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| СОГЛАСОВАНО  Научный руководитель,  старший преподаватель ДПИ ФКН  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.И. Фомичев  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»,  профессор ДПИ ФКН, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |
| |  |  | | --- | --- | | Подп. и дата |  | | Инв. № дубл. |  | | Взам. инв. № |  | | Подп. и дата |  | | Инв. № подл. |  | | **ПРОГРАММА ДЛЯ ПОСТРОЕНИЯ СЕЧЕНИЙ В ДОПОЛНЕННОЙ РЕАЛЬНОСТИ**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 12 01-1-ЛУ** | | | | | | |
|  | |  | | | | |
| Исполнитель:  студент группы БПИ191  \_\_\_\_\_\_\_\_\_\_\_\_\_\_/В.И. Беловицкий/  «\_\_» \_\_\_\_\_\_\_\_\_ 2020 г. | | | | |
|  | | | | | | |
|  | | | | |  | |

**Москва 2020**

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 12 01-1-ЛУ |  |  |
| |  |  | | --- | --- | | Подп. и дата |  | | Инв. № дубл. |  | | Взам. инв. № |  | | Подп. и дата |  | | Инв. № подл. |  | | **ПРОГРАММА ДЛЯ ПОСТРОЕНИЯ СЕЧЕНИЙ В ДОПОЛНЕННОЙ РЕАЛЬНОСТИ**  **Текст программы**  **RU.17701729.04.01-01 12 01-1**  **Листов 35** | | |
|  | | |

**Москва 2020**

**СОДЕРЖАНИЕ**

[1. ОСНОВНОЙ КОД ПРОГРАММЫ 4](#_Toc40640413)

[1.1. Класс ComputeRayScript 4](#_Toc40640414)

[1.2. Класс RayShape 10](#_Toc40640415)

[1.3. Шейдер ComputeRayShader 12](#_Toc40640416)

[2. КЛАССЫ ДЛЯ РАБОТЫ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА 21](#_Toc40640417)

[2.1. Класс ShapePanelOpener 21](#_Toc40640418)

[2.2. Класс ShapePanelManager 22](#_Toc40640419)

[2.3. Класс ShapeTypeDrop 24](#_Toc40640420)

[2.4. ShapeOperationDrop 26](#_Toc40640421)

[2.5. ColorSlider 27](#_Toc40640422)

[2.6. BlendSlider 29](#_Toc40640423)

[2.7. SizePanelManager 30](#_Toc40640424)

[2.8. SizeInput 31](#_Toc40640425)

[2.9. InfoPanelOpener 33](#_Toc40640426)

[2.10. LinkOpener 34](#_Toc40640427)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 35](#_Toc40640428)

ТЕКСТ ПРОГРАММЫ

Программа состоит из двух частей – классов и шейдера для реализации основного функционала приложения (п. 1), а также классов, отвечающих за работу с пользовательским интерфейсом (п. 2).

1. ОСНОВНОЙ КОД ПРОГРАММЫ
   1. Класс ComputeRayScript

using UnityEngine;

using System.Collections.Generic;

using Vuforia;

/// <summary>

/// Класс, отвечающий за инициализацию шейдера

/// </summary>

[ExecuteInEditMode]

[ImageEffectAllowedInSceneView]

[RequireComponent(typeof(Camera))]

public class ComputeRayScript : MonoBehaviour

{

    /// <summary>

    /// Вычислительный шейдер

    /// </summary>

    public ComputeShader raymarchShader;

    /// <summary>

    /// Число для подсчета количества групп потоков для шейдера

    /// </summary>

    float threadDelimeter = 8.0f;

    /// <summary>

    /// Целевая текстура, отображаемая после работы шейдера

    /// </summary>

    RenderTexture target;

    /// <summary>

    /// Основная камера

    /// </summary>

    public new Camera camera;

    /// <summary>

    /// Источник света

    /// </summary>

    public Light lightSource;

    /// <summary>

    /// Ссылка для хранения и высвобождения вычислительного буфера

    /// </summary>

    /// <typeparam name="ComputeBuffer"></typeparam>

    /// <returns></returns>

    ComputeBuffer disposeBuffer;

    /// <summary>

    /// Список форм, отслеживаемых в текущем кадре

    /// </summary>

    /// <typeparam name="RayShape">Объект формы</typeparam>

    List<RayShape> shapes = new List<RayShape>();

    /// <summary>

    /// Устанавливает разрешение экрана при загрузке скрипта

    /// </summary>

    void Awake()

    {

        Screen.SetResolution(400, 240, true);

    }

    /// <summary>

    /// Отпределяет наблюдаемые формы

    /// </summary>

    void DetectTrackables()

    {

        shapes.Clear();

        IEnumerable<TrackableBehaviour> tbs = TrackerManager.Instance.GetStateManager().GetActiveTrackableBehaviours();

        foreach (TrackableBehaviour tb in tbs)

        {

            shapes.Add(tb.GetComponentInChildren<RayShape>());

        }

    }

    /// <summary>

    /// Выполняется при рендеринге изображения:

    /// если на экране присутствуют наблюдаемые формы,

    /// исходное изображение изменяется шейдером.

    /// </summary>

    /// <param name="source"></param>

    /// <param name="destination"></param>

    void OnRenderImage(RenderTexture source, RenderTexture destination)

    {

        DetectTrackables();

        if (shapes.Count != 0)

        {

            InitTargetTexture();

            SetCameraAndLightParameters();

            FillBufferShapes();

            raymarchShader.SetTexture(0, "Source", source);

            raymarchShader.SetTexture(0, "Destination", target);

            int threadGroupsX = Mathf.CeilToInt(camera.pixelWidth / threadDelimeter);

            int threadGroupsY = Mathf.CeilToInt(camera.pixelHeight / threadDelimeter);

            raymarchShader.Dispatch(0, threadGroupsX, threadGroupsY, 1);

            Graphics.Blit(target, destination);

            disposeBuffer.Dispose();

        }

        else

        {

            Graphics.Blit(source, destination);

        }

    }

    /// <summary>

    /// Наполняет буфер формами

    /// </summary>

    void FillBufferShapes()

    {

        shapes.Sort((a, b) => a.operation.CompareTo(b.operation));

        ShapeData[] shapeData = new ShapeData[shapes.Count];

        for (int i = 0; i < shapes.Count; i++)

        {

            var s = shapes[i];

            Vector3 col = new Vector3(s.color.r, s.color.g, s.color.b);

            shapeData[i] = new ShapeData()

            {

                position = s.Position,

                scale = s.Scale,

                color = col,

                shapeType = (int)s.shapeType,

                operation = (int)s.operation,

                blendStrength = s.blendStrength \* 3,

            };

        }

        ComputeBuffer shapeBuffer = new ComputeBuffer(shapeData.Length, ShapeData.GetSize());

        shapeBuffer.SetData(shapeData);

        raymarchShader.SetBuffer(0, "shapes", shapeBuffer);

        raymarchShader.SetInt("shapesNumber", shapeData.Length);

        disposeBuffer = shapeBuffer;

    }

    /// <summary>

    /// Устанавливает параметры камеры и источника света в шейдер

    /// </summary>

    void SetCameraAndLightParameters()

    {

        raymarchShader.SetMatrix("\_CameraToWorld", camera.cameraToWorldMatrix);

        raymarchShader.SetMatrix("\_CameraInverseProjection", camera.projectionMatrix.inverse);

        raymarchShader.SetVector("\_Light", lightSource.transform.forward);

    }

    /// <summary>

    /// Инициализирует целевую текстуру

    /// </summary>

    void InitTargetTexture()

    {

        if (target != null)

        {

            target.Release();

        }

        target = new RenderTexture(camera.pixelWidth, camera.pixelHeight, 0, RenderTextureFormat.ARGBFloat, RenderTextureReadWrite.Linear);

        target.enableRandomWrite = true;

        target.Create();

    }

    /// <summary>

    /// Структура, хранящая данные формы для шейдера

    /// </summary>

    struct ShapeData

    {

        public Vector3 position;

        public Vector3 scale;

        public Vector3 color;

        public int shapeType;

        public int operation;

        public float blendStrength;

        /// <summary>

        /// Возвращает размер данных формы для шейдера

        /// </summary>

        public static int GetSize()

        {

            return sizeof(float) \* 10 + sizeof(int) \* 2;

        }

    }

}

* 1. Класс RayShape

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public enum ShapeType {Sphere, Cube, Torus, Prism};

public enum Operation {Union, Substract, Intersect, Blend}

public class RayShape : MonoBehaviour

{

    public ShapeType \_shapeType = ShapeType.Sphere;

    public ShapeType shapeType

    {

        set

        {

            scaleWasChangedManually = false;

            \_shapeType = value;

        }

        get

        {

            return \_shapeType;

        }

    }

    public Operation operation;

    public Color color = Color.white;

    [Range(0,1)]

    public float blendStrength;

    public Vector3 Position

    {

        get

        {

            return transform.position;

        }

    }

    bool scaleWasChangedManually;

    public Vector3 scale = new Vector3(0.5f, 0.5f, 0.5f);

    public Vector3 Scale

    {

        set

        {

            scaleWasChangedManually = true;

            scale = value;

        }

        get

        {

            if(!scaleWasChangedManually) return GetScale();

            return scale;

        }

    }

    Vector3 GetScale()

    {

        switch(shapeType)

        {

            case ShapeType.Sphere:

                return new Vector3(0.5f, 0, 0);

            case ShapeType.Cube:

                return new Vector3(0.5f , 0.5f, 0.5f);

            case ShapeType.Torus:

                return new Vector3(0.7f , 0.3f, 0);

            case ShapeType.Prism:

                return new Vector3(0.5f , 0.5f, 0);

        }

        return new Vector3(0.5f , 0.5f, 0.5f);

    }

}

* 1. Шейдер ComputeRayShader

#pragma kernel CSMain

Texture2D<float4> Source;

RWTexture2D<float4> Destination;

float4x4 \_CameraToWorld;

float4x4 \_CameraInverseProjection;

float3 \_Light;

static const float maxDst = 10;

static const float maxSteps = 80;

static const float epsilon = 0.01f;

static const float shadowBias = epsilon \* 50;

struct Shape

{

    float3 position;

    float3 size;

    float3 color;

    int shapeType;

    int operation;

    float blendStrength;

};

struct Ray

{

    float3 origin;

    float3 direction;

};

StructuredBuffer<Shape> shapes;

int shapesNumber;

float SphereSDF(float3 p, float3 center, float radius)

{

    return distance(p, center) - radius;

}

float CubeSDF(float3 p, float3 center, float3 size)

{

    float3 o = abs(p - center) - size;

    float ud = length(max(o, 0));

    float n = max(max(min(o.x, 0), min(o.y, 0)), min(o.z, 0));

    return ud + n;

}

float TorusSDF(float3 p, float3 center, float r1, float r2)

{

    float2 q = float2(length((p-center).xz)-r1, p.y-center.y);

    return length(q)-r2;

}

float PrismSDF(float3 p, float3 center, float2 h)

{

    float3 q = abs(p-center);

    return max(q.z-h.y,max(q.x\*0.866025+p.y\*0.5,-p.y)-h.x\*0.5);

}

float CylinderSDF(float3 p, float3 center, float2 h)

{

    float2 d = abs(float2(length((p).xz), p.y)) - h;

    return length(max(d,0.0)) + max(min(d.x,0),min(d. y,0));

}

Ray CreateRay(float3 origin, float3 direction)

{

    Ray ray;

    ray.origin = origin;

    ray.direction = direction;

    return ray;

}

Ray CreateCameraRay(float2 uv)

{

    float3 origin = mul(\_CameraToWorld, float4(0,0,0,1)).xyz;

    float3 direction = mul(\_CameraInverseProjection, float4(uv,0,1)).xyz;

    direction = mul(\_CameraToWorld, float4(direction,0)).xyz;

    direction = normalize(direction);

    return CreateRay(origin,direction);

}

float4 Blend(float a, float b, float3 colorA, float3 colorB, float k)

{

    float h = clamp(0.5 + 0.5 \* (b-a)/k, 0.0, 1.0);

    float bDst = lerp(b, a, h) - k\*h\*(1.0-h);

    float3 bCol = lerp(colorB, colorA, h);

    return float4(bCol, bDst);

}

float4 Combine(float dstA, float dstB, float3 colorA, float3 colorB, int operation, float blendStrength)

{

    float dst = dstA;

    float3 color = colorA;

    // Union

    if (operation == 0) {

        // min(a, b)

        if (dstB < dst) {

            dst = dstB;

            color = colorB;

        }

    }

    // Substract

    else if (operation == 1) {

        // max(a,-b)

        if (-dstB > dst) {

            dst = -dstB;

            color = colorB;

        }

    }

    // Intersect

    else if (operation == 2) {

        // max(a,b)

        if (dstB > dst) {

            dst = dstB;

            color = colorB;

        }

    }

    // Blend

    else if (operation == 3) {

        float4 blend = Blend(dstA, dstB, colorA, colorB, blendStrength);

        dst = blend.w;

        color = blend.xyz;

    }

    return float4(color,dst);

}

float GetShapeDst(Shape shape, float3 p)

{

    if (shape.shapeType == 0)

    {

        return SphereSDF(p, shape.position, shape.size.x);

    }

    else if (shape.shapeType == 1)

    {

        return CubeSDF(p, shape.position, shape.size);

    }

    else if (shape.shapeType == 2)

    {

        return TorusSDF(p, shape.position, shape.size.x, shape.size.y);

    }

    else if(shape.shapeType == 3)

    {

        return PrismSDF(p, shape.position, shape.size.xy);

    }

    else if(shape.shapeType == 4)

    {

        return CylinderSDF(p, shape.position, shape.size.xy);

    }

    return maxDst;

}

float4 SceneDst(float3 p)

{

    float globalDst = maxDst;

    float3 globalColor = 1;

    for (int i = 0; i < shapesNumber; i ++) {

        Shape shape = shapes[i];

        float localDst = GetShapeDst(shape, p);

        float3 localColor = shape.color;

        float4 globalCombined = Combine(globalDst, localDst, globalColor, localColor, shape.operation, shape.blendStrength);

        globalColor = globalCombined.xyz;

        globalDst = globalCombined.w;

    }

    return float4(globalColor, globalDst);

}

float3 CalculateNormal(float3 p)

{

    float x = SceneDst(float3(p.x+epsilon,p.y,p.z)).w - SceneDst(float3(p.x-epsilon,p.y,p.z)).w;

    float y = SceneDst(float3(p.x,p.y+epsilon,p.z)).w - SceneDst(float3(p.x,p.y-epsilon,p.z)).w;

    float z = SceneDst(float3(p.x,p.y,p.z+epsilon)).w - SceneDst(float3(p.x,p.y,p.z-epsilon)).w;

    return normalize(float3(x,y,z));

}

float CalculateShadow(Ray ray, float dstToShadePoint)

{

    float rayDst = 0;

    int marchSteps = 0;

    float shadowIntensity = 0.2;

    float brightness = 1;

    while (rayDst < dstToShadePoint) {

        marchSteps++;

        float4 sceneDst = SceneDst(ray.origin);

        float dst = sceneDst.w;

        if (dst <= epsilon) {

            return shadowIntensity;

        }

        brightness = min(brightness, dst\*200);

        ray.origin += ray.direction \* dst;

        rayDst += dst;

    }

    return shadowIntensity + (1-shadowIntensity) \* brightness;

}

void RayMarching(uint3 id, uint width, uint height)

{

    float2 uv = id.xy / float2(width, height) \* 2 - 1;

    float rayDst = 0;

    Ray ray = CreateCameraRay(uv);

    int marchSteps = 0;

    while (marchSteps < maxSteps && rayDst < maxDst)

    {

        marchSteps++;

        float4 sceneDst = SceneDst(ray.origin);

        float dst = sceneDst.w;

        if (dst <= epsilon) {

            // Light

            float3 pointOnSurface = ray.origin + ray.direction \* dst;

            float3 normal = CalculateNormal(pointOnSurface - ray.direction \* epsilon);

            float3 lightDir = -\_Light;

            float lighting = saturate(saturate(dot(normal,lightDir)));

            float3 col = sceneDst.xyz;

            // Shadow

            float3 offsetPos = pointOnSurface + normal \* shadowBias;

            float3 dirToLight = -\_Light;

            ray.origin = offsetPos;

            ray.direction = dirToLight;

            float dstToLight = maxDst;

            float shadow = CalculateShadow(ray, dstToLight);

            Destination[id.xy] = float4(col \* lighting \* shadow, 1);

            break;

        }

        ray.origin += ray.direction \* dst;

        rayDst += dst;

    }

}

[numthreads(8, 8, 1)]

void CSMain (uint3 id : SV\_DispatchThreadID)

{

    uint width, height;

    Destination.GetDimensions(width, height);

    Destination[id.xy] = Source[id.xy];

    RayMarching(id, width, height);

}

1. КЛАССЫ ДЛЯ РАБОТЫ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА
   1. Класс ShapePanelOpener

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ShapePanelOpener : MonoBehaviour

{

    public GameObject ShapePanel;

    public int shapeIndex;

    public void TogglePanel()

    {

        if(ShapePanel != null)

        {

            ShapePanelManager spm = ShapePanel.GetComponent<ShapePanelManager>();

            bool isActive = ShapePanel.activeSelf;

            if(!isActive)

            {

                ShapePanel.SetActive(!isActive);

                spm.SetData(shapeIndex);

            }

            else

            {

                if(spm.Index != shapeIndex)

                {

                    spm.SetData(shapeIndex);

                    isActive = false;

                }

                ShapePanel.SetActive(!isActive);

            }

        }

    }

}

* 1. Класс ShapePanelManager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using Vuforia;

public class ShapePanelManager : MonoBehaviour

{

    GameObject Panel;

    public int Index { get; set; }

    List<RayShape> shapes;

    GameObject typeDrop;

    GameObject operationDrop;

    GameObject colorSlider;

    GameObject blendSlider;

    GameObject sizePanel;

    void InitElements()

    {

        Panel = this.gameObject;

        typeDrop = GameObject.FindGameObjectWithTag("TypeDrop");

        operationDrop = GameObject.FindGameObjectWithTag("OperationDrop");

        colorSlider = GameObject.FindGameObjectWithTag("ColorSlider");

        blendSlider = GameObject.FindGameObjectWithTag("BlendSlider");

        sizePanel = GameObject.FindGameObjectWithTag("SizePanel");

        GetAllTrackables();

    }

    public void GetAllTrackables()

    {

        shapes = new List<RayShape>();

        foreach(GameObject obj in GameObject.FindGameObjectsWithTag("Shape"))

        {

            shapes.Add(obj.GetComponent<RayShape>());

        }

    }

    public void SetData(int index)

    {

        if(Panel == null)

        {

            InitElements();

        }

        Index = index;

        ShapeTypeDrop stp = typeDrop.GetComponent<ShapeTypeDrop>();

        stp.SetOptions(shapes[index]);

        operationDrop.GetComponent<ShapeOperationDrop>().SetOptions(shapes[index]);

        colorSlider.GetComponent<ColorSlider>().SetData(shapes[index]);

        blendSlider.GetComponent<BlendSlider>().SetData(shapes[index]);

        SizePanelManager spm = sizePanel.GetComponent<SizePanelManager>();

        spm.SetData(shapes[index]);

        stp.SetSizePanel(spm);

    }

}

* 1. Класс ShapeTypeDrop

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ShapeTypeDrop : MonoBehaviour

{

    Dropdown dropdown;

    RayShape currentShape;

    SizePanelManager sizePanel;

    void Init()

    {

        dropdown = this.gameObject.GetComponent<Dropdown>();

        dropdown.ClearOptions();

        dropdown.onValueChanged.AddListener(delegate{ OnClickHandler(dropdown); });

        string[] names = Enum.GetNames(typeof(ShapeType));

        dropdown.AddOptions(new List<string>(names));

    }

    public void OnClickHandler(Dropdown drop)

    {

        if(currentShape != null)

        {

            Array values = Enum.GetValues(typeof(ShapeType));

            currentShape.shapeType = (ShapeType)values.GetValue(drop.value);

            if(sizePanel != null)

            {

                sizePanel.SetData(currentShape);

            }

        }

    }

    public void SetOptions(RayShape shape)

    {

        if(dropdown == null)

        {

            Init();

        }

        currentShape = shape;

        dropdown.SetValueWithoutNotify((int)shape.shapeType);

    }

    public void SetSizePanel(SizePanelManager spm)

    {

        sizePanel = spm;

    }

}

* 1. ShapeOperationDrop

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ShapeOperationDrop : MonoBehaviour

{

    Dropdown dropdown;

    RayShape currentShape;

    void Init()

    {

        dropdown = this.gameObject.GetComponent<Dropdown>();

        dropdown.ClearOptions();

        dropdown.onValueChanged.AddListener(delegate{ OnClickHandler(dropdown); });

        string[] names = Enum.GetNames(typeof(Operation));

        dropdown.AddOptions(new List<string>(names));

    }

    public void OnClickHandler(Dropdown drop)

    {

        if(currentShape != null){

            Array values = Enum.GetValues(typeof(Operation));

            currentShape.operation = (Operation)values.GetValue(drop.value);

        }

    }

    public void SetOptions(RayShape shape)

    {

        if(dropdown == null)

        {

            Init();

        }

        currentShape = shape;

        dropdown.

        SetValueWithoutNotify((int)shape.operation);

    }

}

* 1. ColorSlider

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ColorSlider : MonoBehaviour

{

    public Slider slider;

    public Image sliderHandle;

    public Image sliderBackground;

    Texture2D colorTexture;

    RayShape currentShape;

    void Awake()

    {

        colorTexture = CreateColorTexture(360);

        Rect rect = new Rect(0, 0, colorTexture.width, colorTexture.height);

        sliderBackground.sprite = Sprite.Create(colorTexture, rect, rect.center);

        slider.onValueChanged.AddListener(OnValueChanged);

    }

    public void SetData(RayShape shape)

    {

        currentShape = shape;

        float hue;

        Color.RGBToHSV(shape.color, out hue, out \_, out \_);

        OnValueChanged(hue);

        slider.value = hue;

    }

    void OnValueChanged(float value)

    {

        Color color = Color.HSVToRGB(value, 1, 1);

        sliderHandle.color = color;

        currentShape.color = color;

    }

    Texture2D CreateColorTexture (int density)

    {

        Texture2D hueTex = new Texture2D (density, 1);

        Color[] colors = new Color[density];

        for (int k = 0; k < density; k++)

        {

            colors[k] = Color.HSVToRGB ((1.0f \* k) / density, 1, 1);

        }

        hueTex.SetPixels (colors);

        hueTex.Apply ();

        return hueTex;

    }

 }

* 1. BlendSlider

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class BlendSlider : MonoBehaviour

{

    public Slider slider;

    RayShape currentShape;

    void Awake()

    {

        slider.onValueChanged.AddListener(OnValueChanged);

    }

    void OnValueChanged(float value)

    {

        currentShape.blendStrength = value;

    }

    public void SetData(RayShape shape)

    {

        currentShape = shape;

        slider.value = shape.blendStrength;

    }

}

* 1. SizePanelManager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class SizePanelManager : MonoBehaviour

{

    GameObject Panel;

    RayShape currentShape;

    void Init()

    {

        Panel = this.gameObject;

    }

    public void SetData(RayShape shape)

    {

        if(Panel == null)

        {

            Init();

        }

        currentShape = shape;

        SizeInput[] inputs = Panel.GetComponentsInChildren<SizeInput>();

        foreach(SizeInput input in inputs)

        {

            input.SetValue(currentShape);

        }

    }

}

* 1. SizeInput

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class SizeInput : MonoBehaviour

{

    public int Index = 0;

    InputField input;

    RayShape currentShape;

    void Init()

    {

        input = this.gameObject.GetComponent<InputField>();

        input.onValueChanged.AddListener(ChangeScale);

    }

    void ChangeScale(string value)

    {

        if(currentShape != null)

        {

            Vector3 scale = currentShape.Scale;

            float num = 0.5f;

            if(!float.TryParse(value, out num))

            {

                input.text = currentShape.Scale[Index].ToString("g4");

            }

            scale[Index] = num;

            currentShape.Scale = scale;

        }

    }

    public void SetValue(RayShape shape)

    {

        if(input == null)

        {

            Init();

        }

        currentShape = shape;

        input.text = shape.Scale[Index].ToString("g4");

    }

}

* 1. InfoPanelOpener

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class InfoPanelOpener : MonoBehaviour

{

    public GameObject InfoPanel;

    public void TogglePanel()

    {

        if(InfoPanel != null)

        {

            InfoPanel.SetActive(!InfoPanel.activeSelf);

        }

    }

}

* 1. LinkOpener

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class LinkOpener : MonoBehaviour

{

    public string link = "https://github.com/vbelovitsky/RaymarchAR/tree/master/QR-codes";

    public void OpenLink()

    {

        if(link != null)

        {

            Application.OpenURL(link);

        }

    }

}

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопрово-дитель-ного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |