CMPT 766

Computer Animation

Final Project

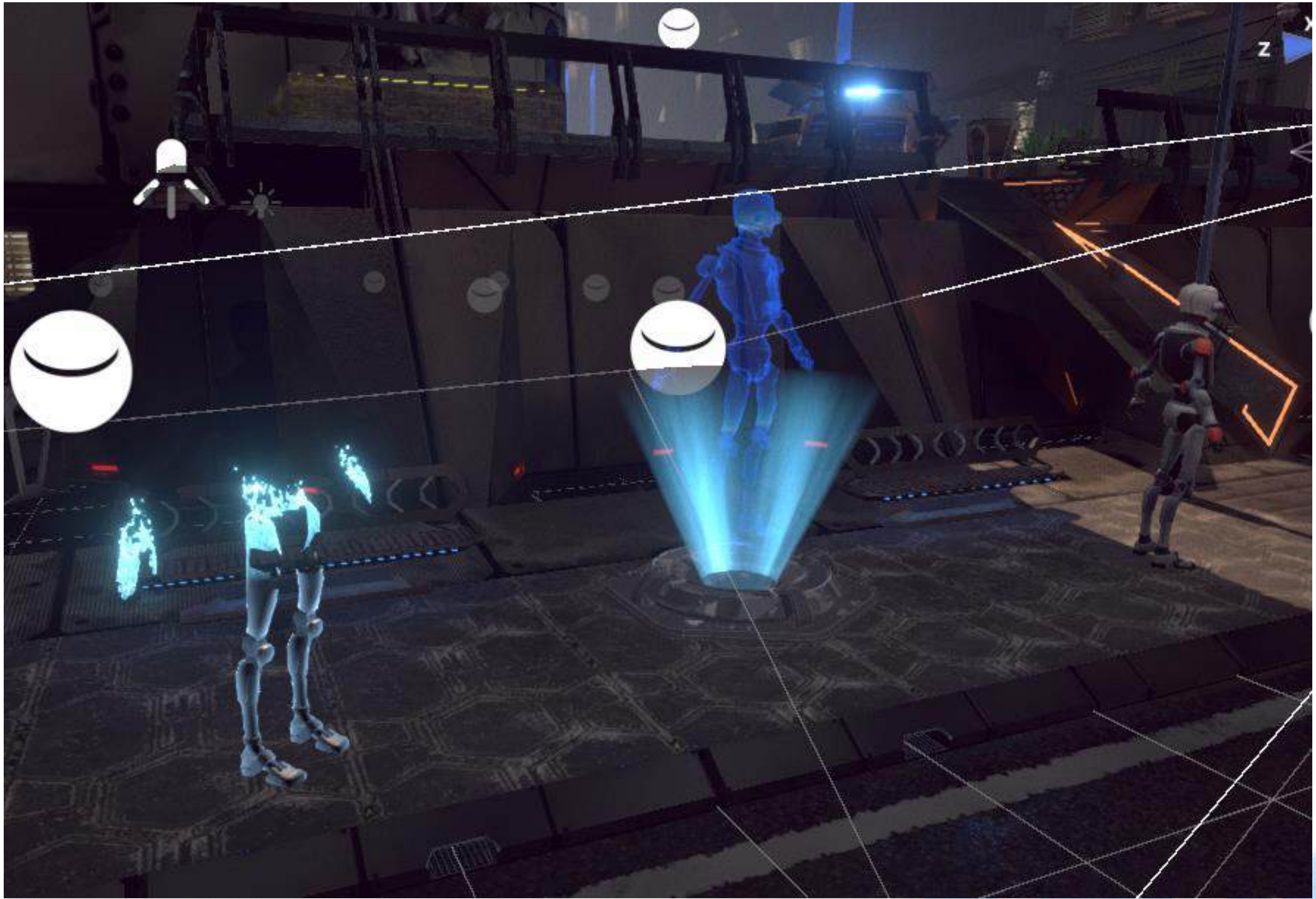
Unity Cyberpunk Hologram and Teleport effects

Reporter:

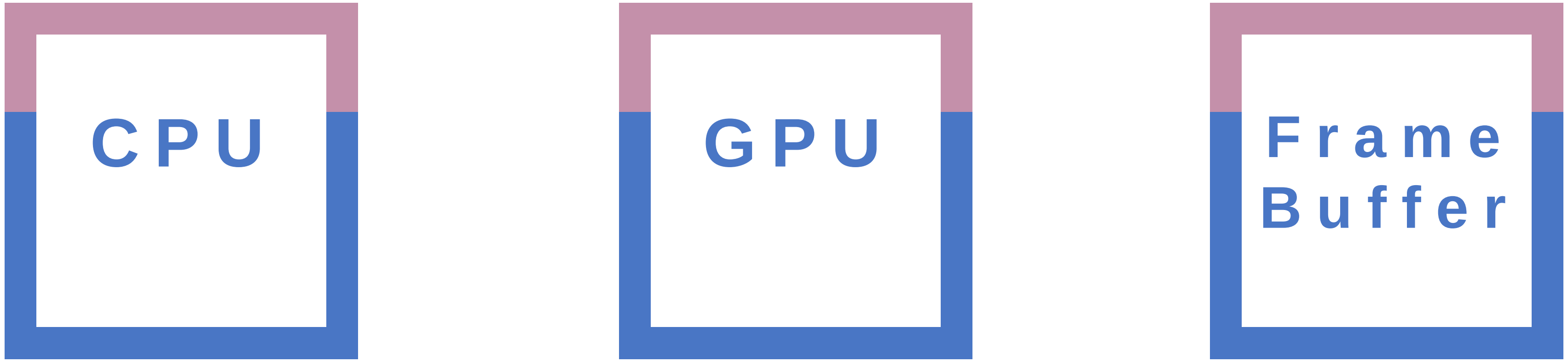
Wenhe Wang

Gouttham Nambirajan

Final Effect



Rendering Pipeline



CPU

GPU

Frame Buffer

Data preparation

Culling, Sort, Batch, SetPassCall, DrawCall

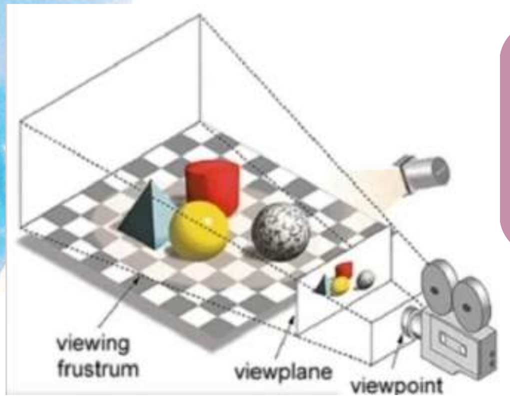
Rendering

Vertex Shader, Primitive Assembly, Rasterization, Fragment Shader, and Output Merge

Post-processing

Applying more visual effects

Unity Build-in Rendering Pipeline



Call Render()

1. Culling: Vertebral Culling, Occlusion Culling, laminar, etc.

2. Sort: Distance, Queue, etc.

3. Data Packaging: data, parameters

4. Call Shader: SetPassCall, DrawCall

5. GPU Rendering Pipeline

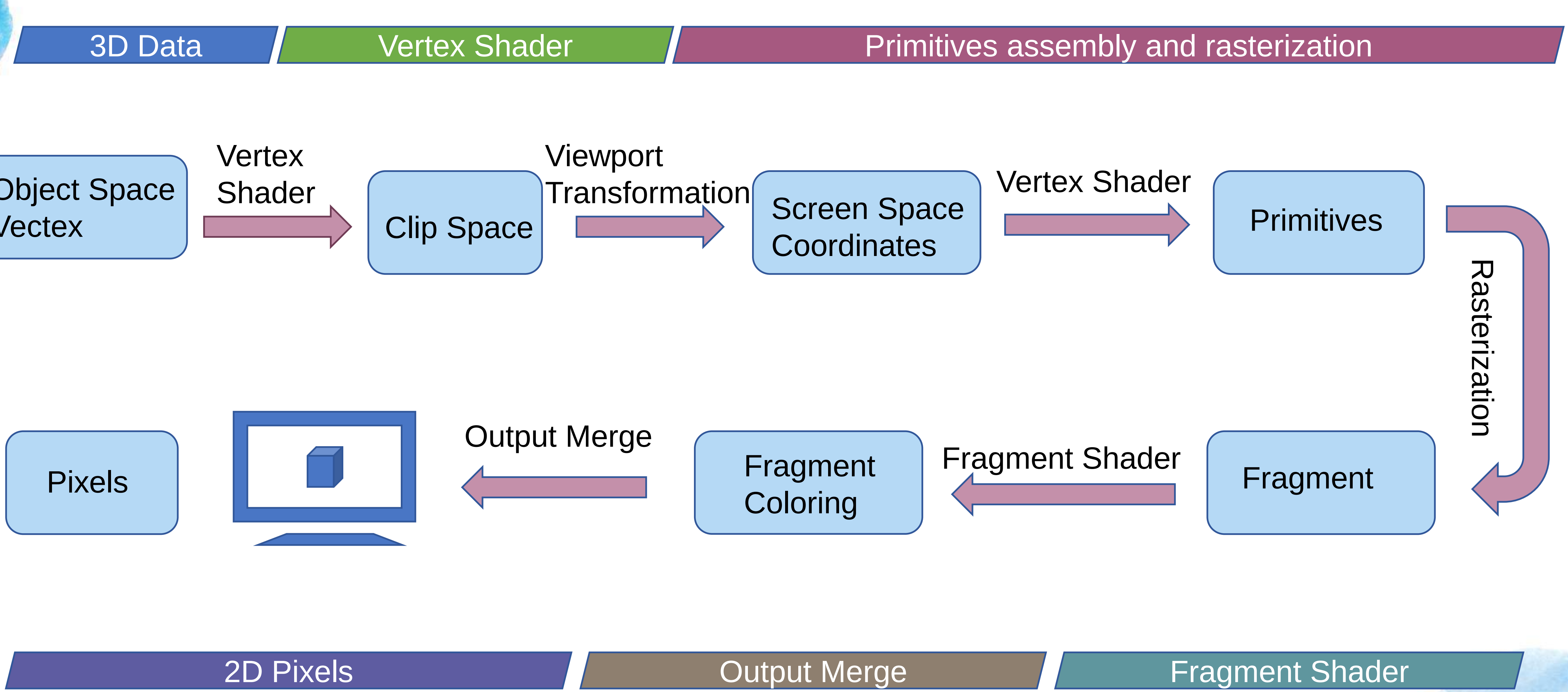
Frame Buffer

Post-processing logics

GPU Rendering Pipeline

Rendering Object

GPU Rendering Pipeline



3D Data

Vertex Shader

Primitives assembly and rasterization

Object Space Vectex

Vertex Shader

Clip Space

Viewport Transformation

Screen Space Coordinates

Vertex Shader

Primitives

Rasterization

Fragment

Fragment Shader

Fragment Coloring

Output Merge

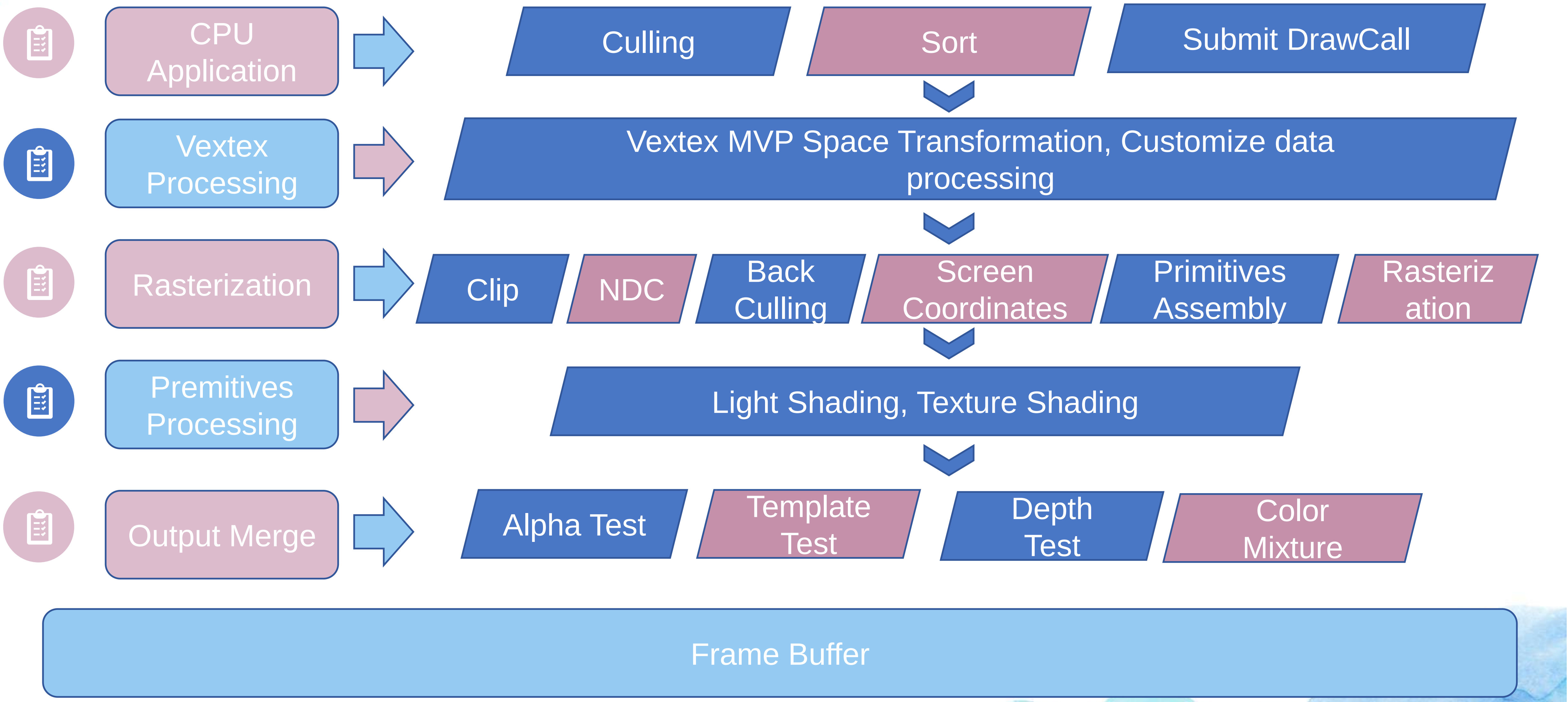
Pixels

2D Pixels

Output Merge

Fragment Shader

General Rendering Pipeline



CPU Application → Culling, Sort, Submit DrawCall

Vextex Processing → Vextex MVP Space Transformation, Customize data processing

Rasterization → Clip, NDC, Back Culling, Screen Coordinates, Primitives Assembly, Rasterization

Premitives Processing → Light Shading, Texture Shading

Output Merge → Alpha Test, Template Test, Depth Test, Color Mixture

Frame Buffer

Hologram Effect

01

Transparency Effect

02

Scanline Effect

03

Rim Lighting Effect

04

Glitch Effect
(Shape Disturb, Flicker Effect, Emission Effect)

01

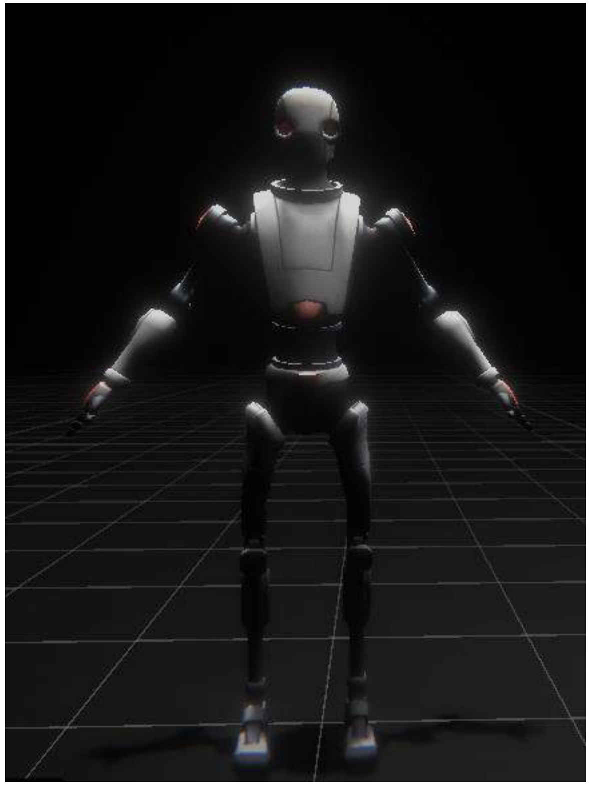
Hologram Effect

Wenhe Wang

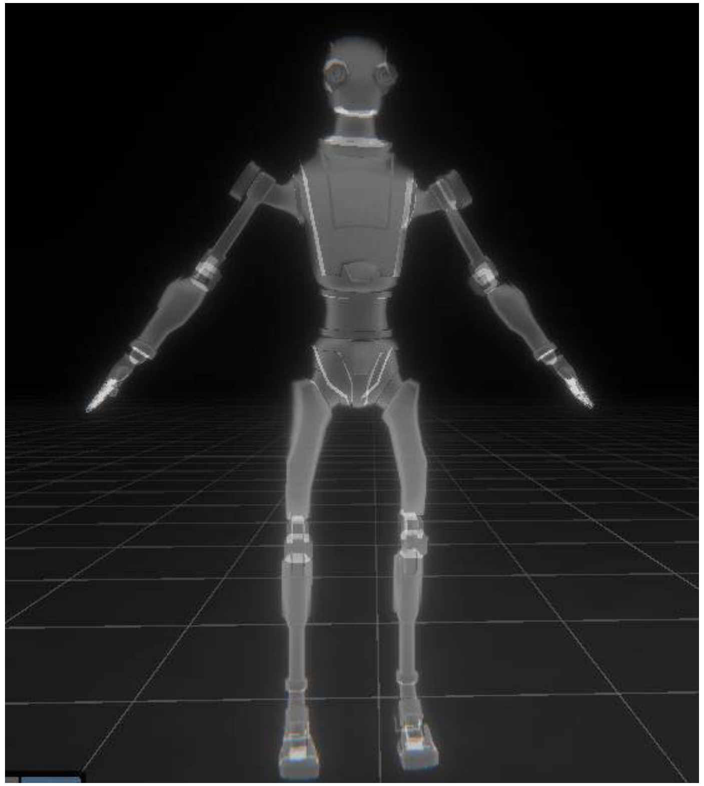
Concept

Transparency Effect

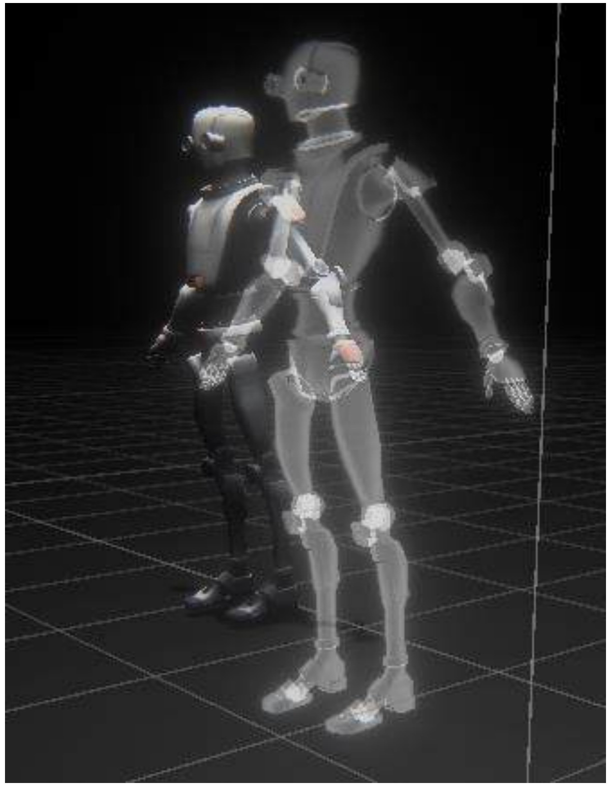
Original



Transparent

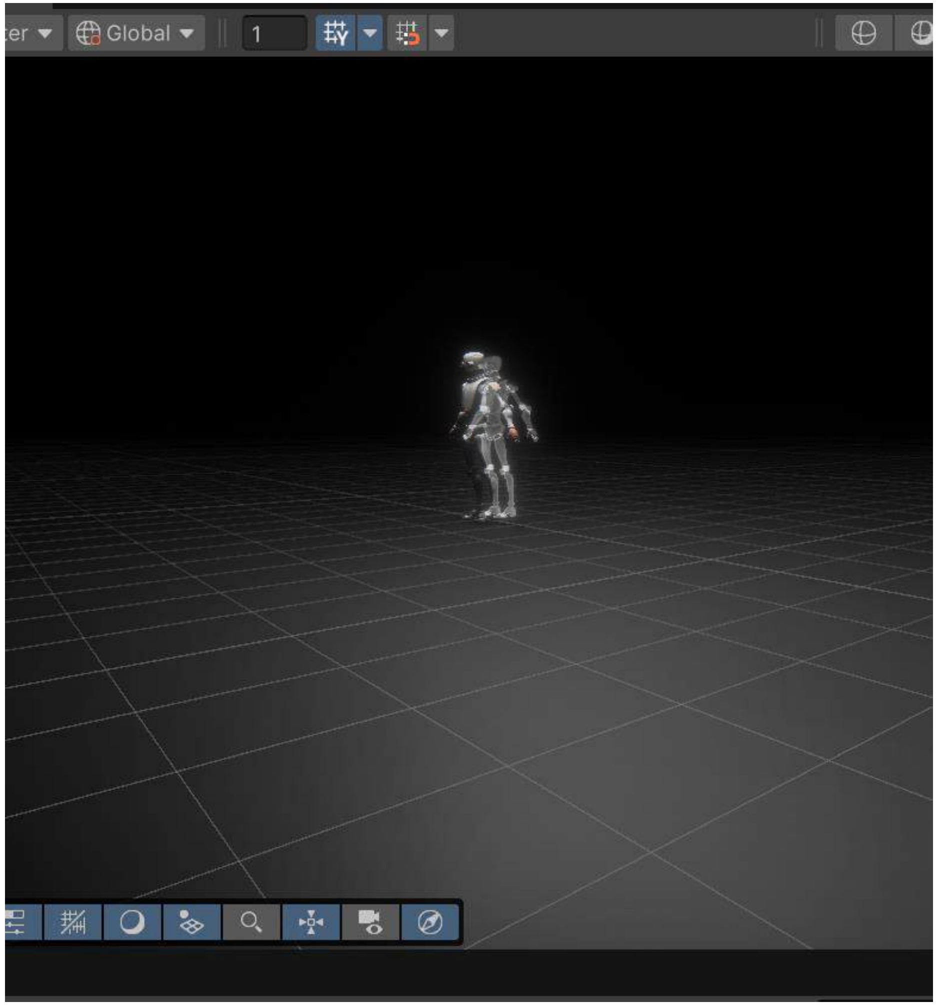


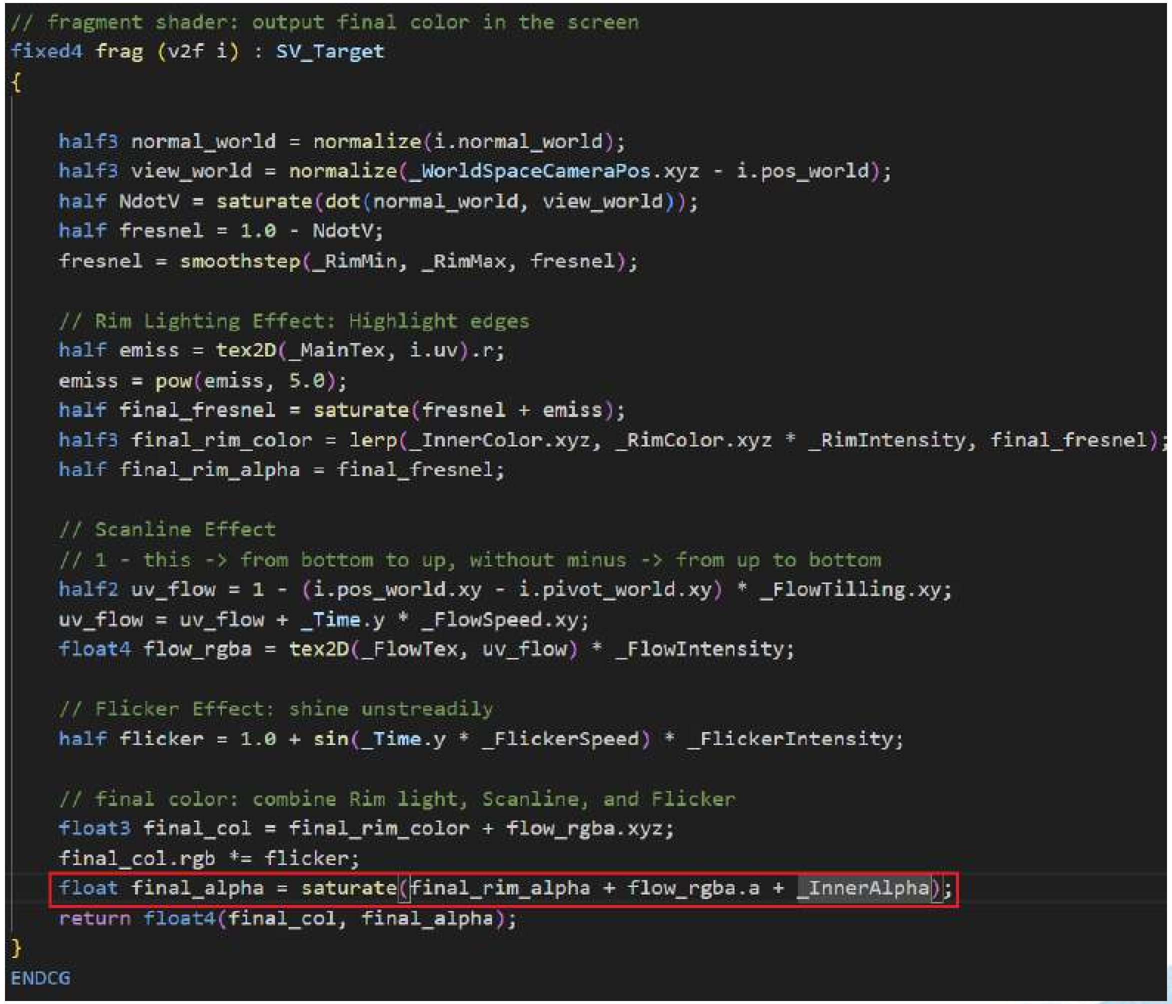
Transparent



Transparency Effect

Fresnel Effect and Transparency level

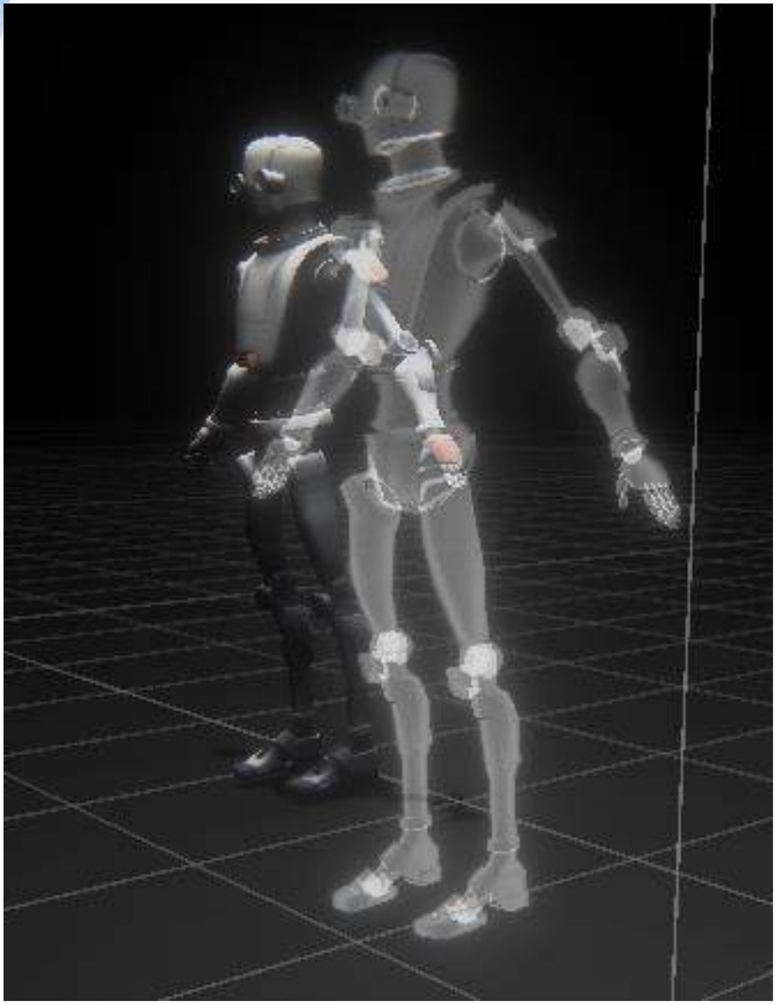


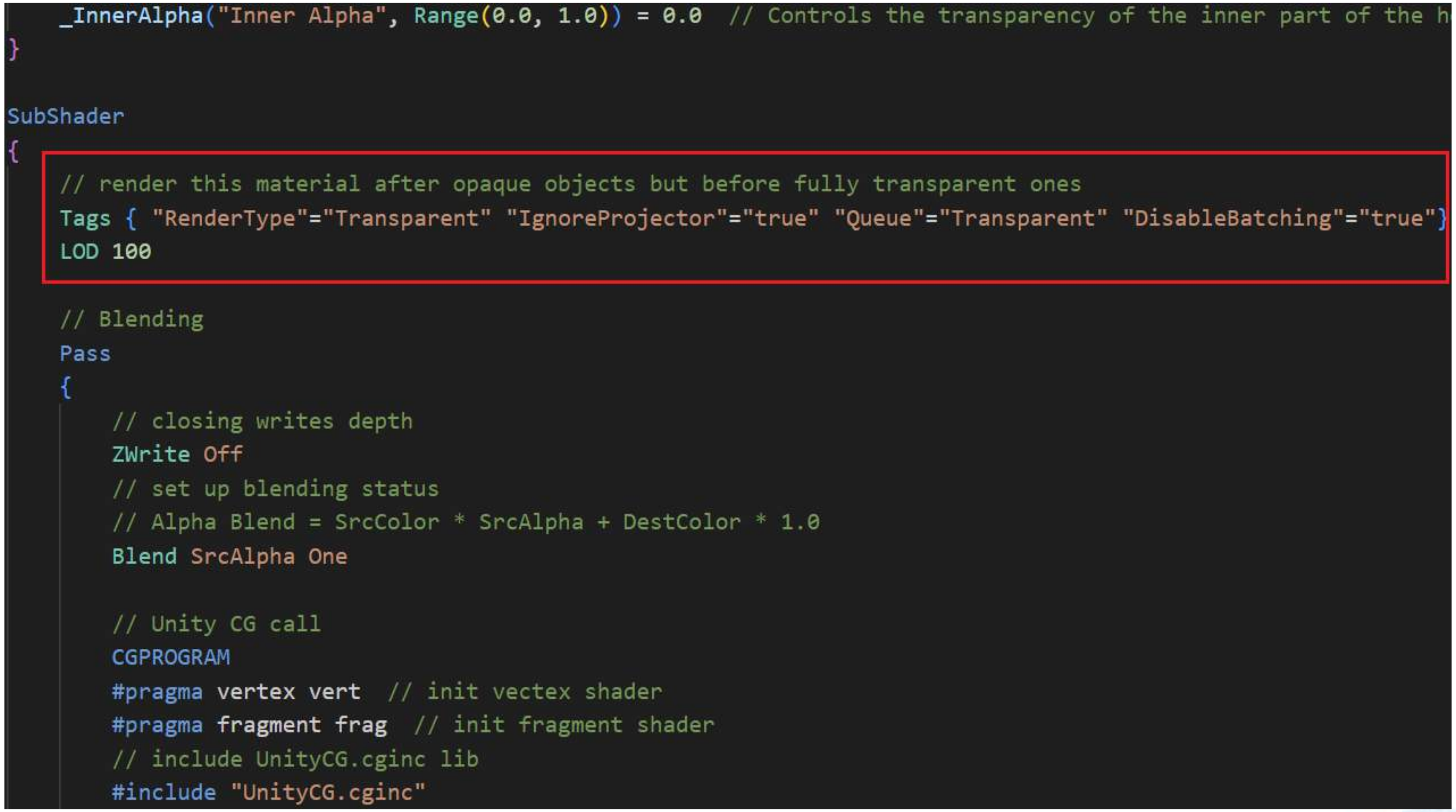


```
// fragment shader: output final color in the screen
fixed4 frag (v2f i) : SV_Target
{

    half3 normal_world = normalize(i.normal_world);
    half3 view_world = normalize(_WorldSpaceCameraPos.xyz - i.pos_world);
    half NdotV = saturate(dot(normal_world, view_world));
    half fresnel = 1.0 - NdotV;
    fresnel = smoothstep(_RimMin, _RimMax, fresnel);

    // Rim Lighting Effect: Highlight edges
    half emiss = tex2D(_MainTex, i.uv).r;
    emiss = pow(emiss, 5.0);
    half final_fresnel = saturate(fresnel + emiss);
    half3 final_rim_color = lerp(_InnerColor.xyz, _RimColor.xyz * _RimIntensity, final_fresnel);
    half final_rim_alpha = final_fresnel;

    // Scanline Effect
    // 1 - this -> from bottom to up, without minus -> from up to bottom
    half2 uv_flow = 1 - (i.pos_world.xy - i.pivot_world.xy) * _FlowTilling.xy;
    uv_flow = uv_flow + _Time.y * _FlowSpeed.xy;
    float4 flow_rgba = tex2D(_FlowTex, uv_flow) * _FlowIntensity;

    // Flicker Effect: shine unstreadily
    half flicker = 1.0 + sin(_Time.y * _FlickerSpeed) * _FlickerIntensity;

    // final color: combine Rim light, Scanline, and Flicker
    float3 final_col = final_rim_color + flow_rgba.xyz;
    final_col.rgb *= flicker;
    float final_alpha = saturate(final_rim_alpha + flow_rgba.a + _InnerAlpha);
    return float4(final_col, final_alpha);
}
ENDCG
```

Transparency Effect

Self-masking of transparent objects



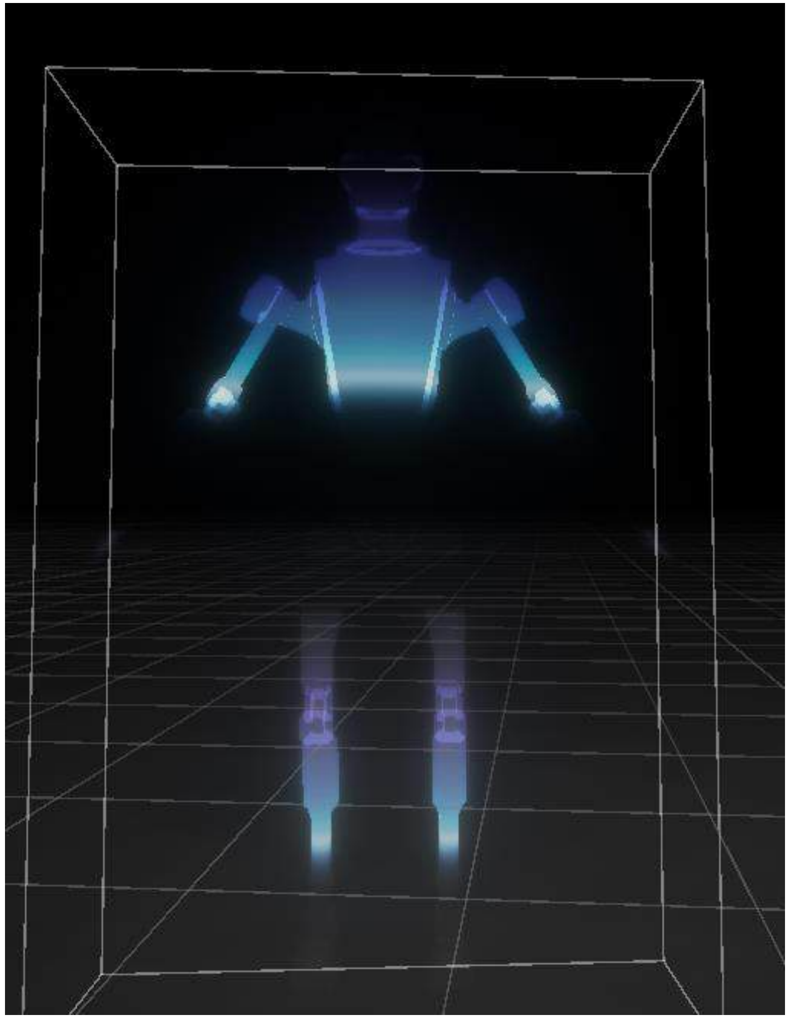


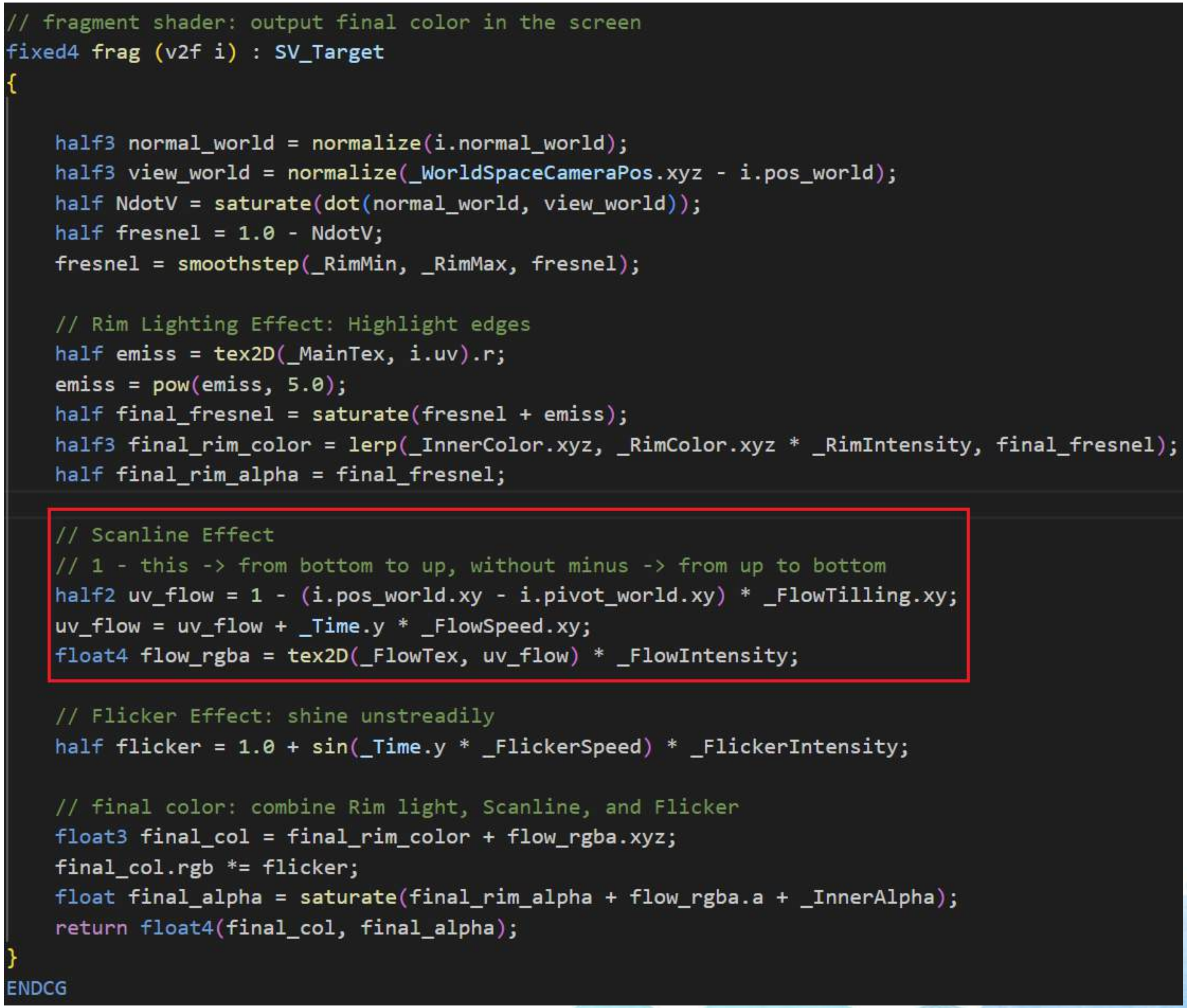
```
    _InnerAlpha("Inner Alpha", Range(0.0, 1.0)) = 0.0  // Controls the transparency of the inner part of the h
}

SubShader
{
    // render this material after opaque objects but before fully transparent ones
    Tags { "RenderType"="Transparent" "IgnoreProjector"="true" "Queue"="Transparent" "DisableBatching"="true"}
    LOD 100

    // Blending
    Pass
    {
        // closing writes depth
        ZWrite Off
        // set up blending status
        // Alpha Blend = SrcColor * SrcAlpha + DestColor * 1.0
        Blend SrcAlpha One

        // Unity CG call
        CGPROGRAM
        #pragma vertex vert  // init vectex shader
        #pragma fragment frag  // init fragment shader
        // include UnityCG.cginc lib
        #include "UnityCG.cginc"
```

Scanline Effect

Texture map

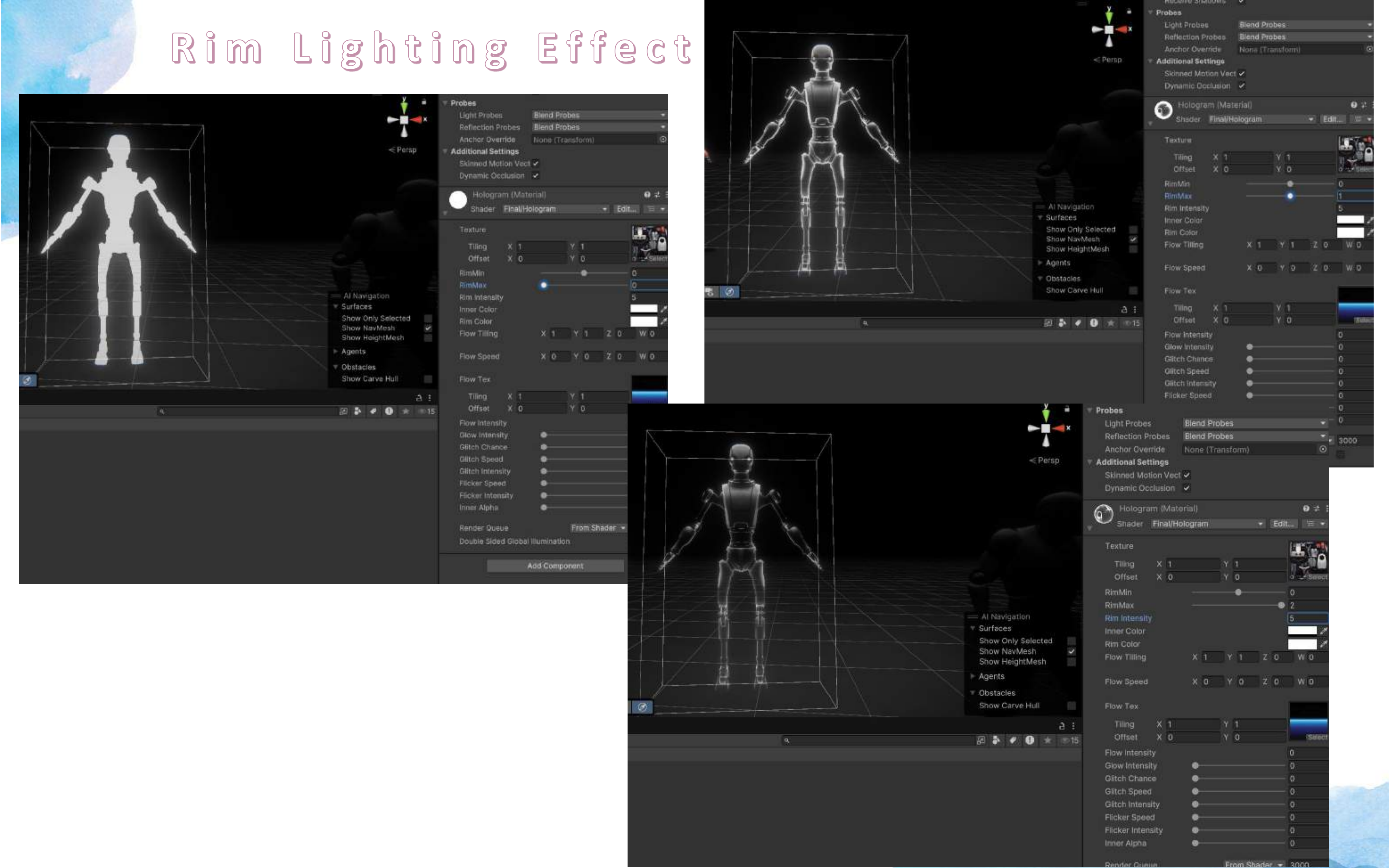
Scanline Effect





```
// fragment shader: output final color in the screen
fixed4 frag (v2f i) : SV_Target
{

    half3 normal_world = normalize(i.normal_world);
    half3 view_world = normalize(_WorldSpaceCameraPos.xyz - i.pos_world);
    half NdotV = saturate(dot(normal_world, view_world));
    half fresnel = 1.0 - NdotV;
    fresnel = smoothstep(_RimMin, _RimMax, fresnel);

    // Rim Lighting Effect: Highlight edges
    half emiss = tex2D(_MainTex, i.uv).r;
    emiss = pow(emiss, 5.0);
    half final_fresnel = saturate(fresnel + emiss);
    half3 final_rim_color = lerp(_InnerColor.xyz, _RimColor.xyz * _RimIntensity, final_fresnel);
    half final_rim_alpha = final_fresnel;

    // Scanline Effect
    // 1 - this -> from bottom to up, without minus -> from up to bottom
    half2 uv_flow = 1 - (i.pos_world.xy - i.pivot_world.xy) * _FlowTilling.xy;
    uv_flow = uv_flow + _Time.y * _FlowSpeed.xy;
    float4 flow_rgba = tex2D(_FlowTex, uv_flow) * _FlowIntensity;

    // Flicker Effect: shine unstreadily
    half flicker = 1.0 + sin(_Time.y * _FlickerSpeed) * _FlickerIntensity;

    // final color: combine Rim light, Scanline, and Flicker
    float3 final_col = final_rim_color + flow_rgba.xyz;
    final_col.rgb *= flicker;
    float final_alpha = saturate(final_rim_alpha + flow_rgba.a + _InnerAlpha);
    return float4(final_col, final_alpha);
}
ENDCG
```

Rim Lighting Effect

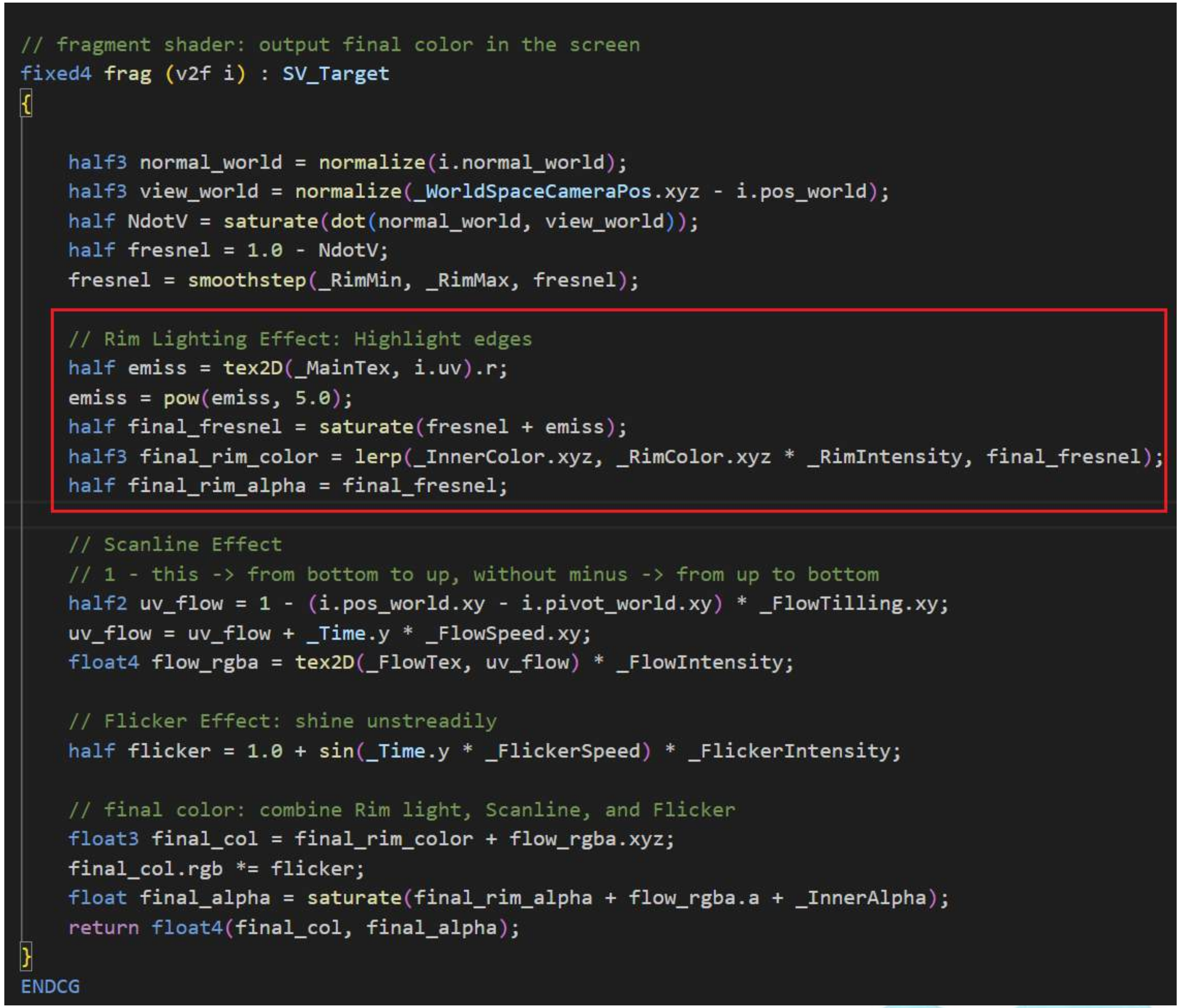


Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Skinned Motion Vect ✓
Dynamic Occlusion ✓
Hologram (Material)
Shader: Final/Hologram
Texture
Tiling X 1 Y 1
Offset X 0 Y 0
RimMin 0
RimMax 0
Rim Intensity 5
Inner Color
Rim Color
Flow Tiling X 1 Y 1 Z 0 W 0
Flow Speed X 0 Y 0 Z 0 W 0
Flow Tex
Tiling X 1 Y 1
Offset X 0 Y 0
Flow Intensity
Glow Intensity
Glitch Chance
Glitch Speed
Glitch Intensity
Flicker Speed
Flicker Intensity
Inner Alpha
Render Queue: From Shader
Double Sided Global Illumination

Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Skinned Motion Vect ✓
Dynamic Occlusion ✓
Hologram (Material)
Shader: Final/Hologram
Texture
Tiling X 1 Y 1
Offset X 0 Y 0
RimMin 0
RimMax 1
Rim Intensity 5
Inner Color
Rim Color
Flow Tiling X 1 Y 1 Z 0 W 0
Flow Speed X 0 Y 0 Z 0 W 0
Flow Tex
Tiling X 1 Y 1
Offset X 0 Y 0
Flow Intensity 0
Glow Intensity 0
Glitch Chance 0
Glitch Speed 0
Glitch Intensity 0
Flicker Speed 0
0
0
3000

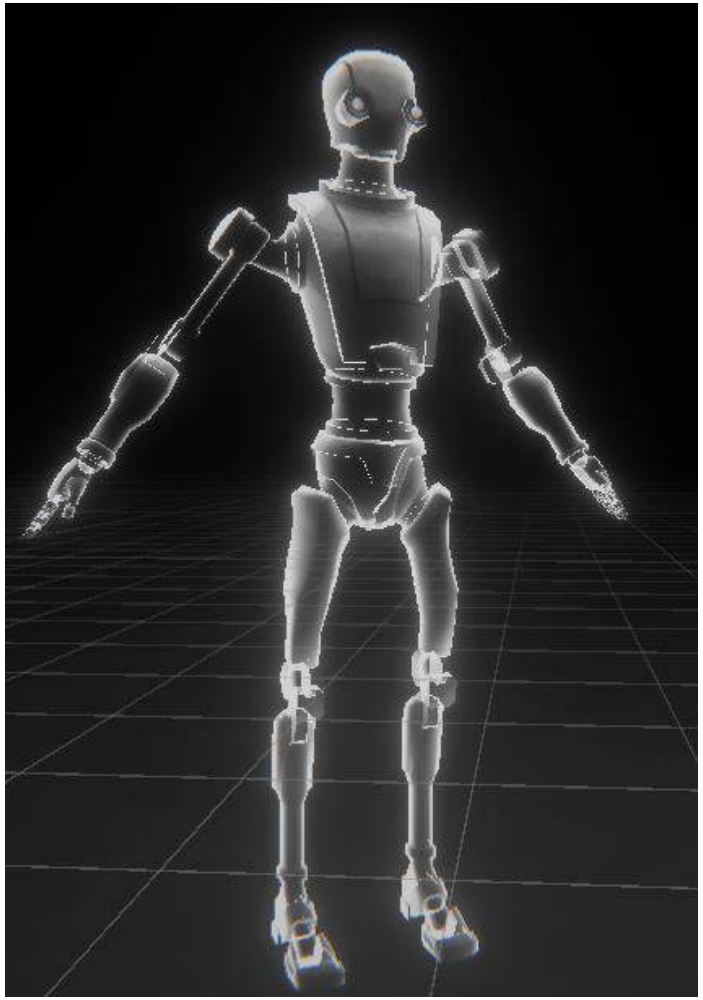
Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Skinned Motion Vect ✓
Dynamic Occlusion ✓
Hologram (Material)
Shader: Final/Hologram
Texture
Tiling X 1 Y 1
Offset X 0 Y 0
RimMin 0
RimMax 2
Rim Intensity 5
Inner Color
Rim Color
Flow Tiling X 1 Y 1 Z 0 W 0
Flow Speed X 0 Y 0 Z 0 W 0
Flow Tex
Tiling X 1 Y 1
Offset X 0 Y 0
Flow Intensity 0
Glow Intensity 0
Glitch Chance 0
Glitch Speed 0
Glitch Intensity 0
Flicker Speed 0
Flicker Intensity 0
Inner Alpha 0
Render Queue: From Shader 3000

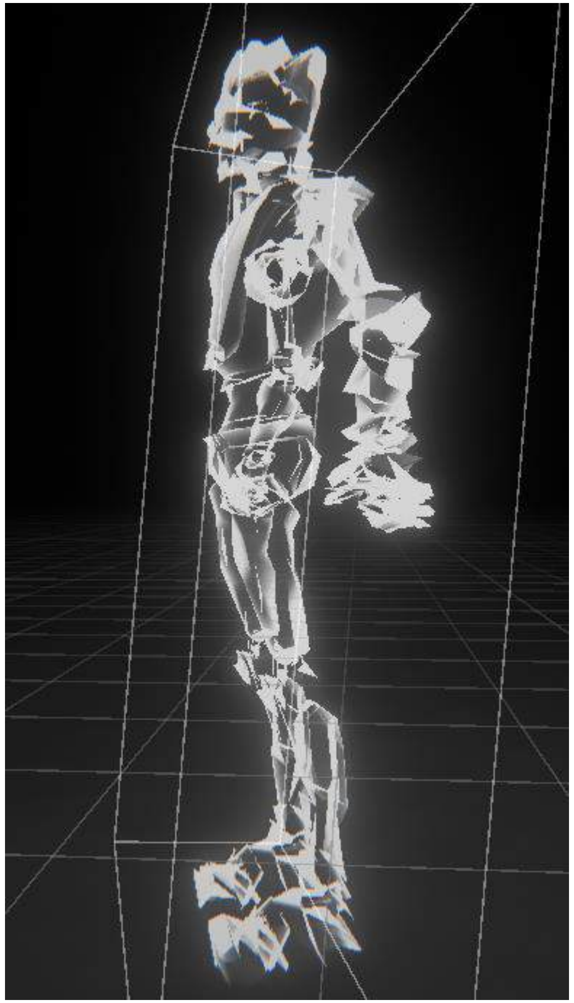
Rim Lighting Effect

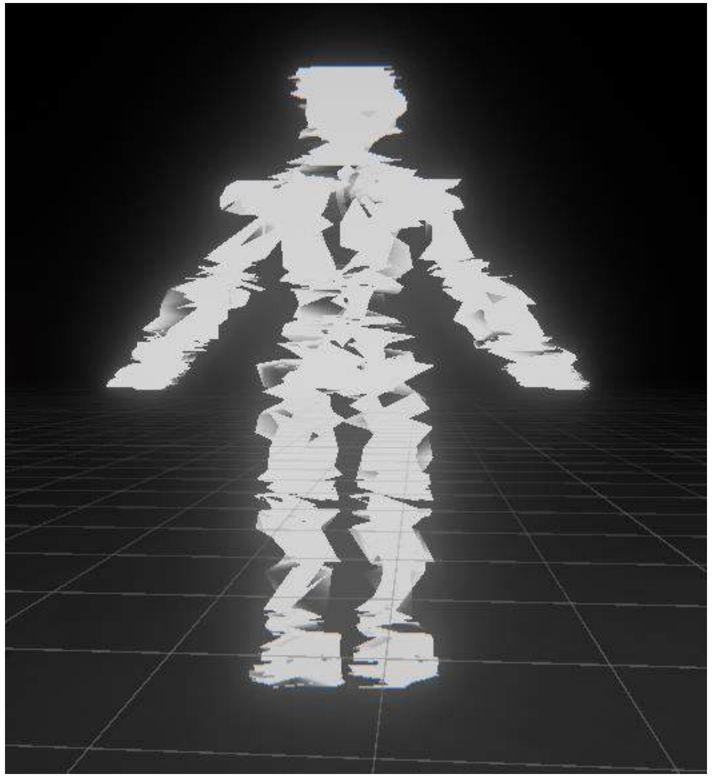


```
// fragment shader: output final color in the screen
fixed4 frag (v2f i) : SV_Target
{

    half3 normal_world = normalize(i.normal_world);
    half3 view_world = normalize(_WorldSpaceCameraPos.xyz - i.pos_world);
    half NdotV = saturate(dot(normal_world, view_world));
    half fresnel = 1.0 - NdotV;
    fresnel = smoothstep(_RimMin, _RimMax, fresnel);

    // Rim Lighting Effect: Highlight edges
    half emiss = tex2D(_MainTex, i.uv).r;
    emiss = pow(emiss, 5.0);
    half final_fresnel = saturate(fresnel + emiss);
    half3 final_rim_color = lerp(_InnerColor.xyz, _RimColor.xyz * _RimIntensity, final_fresnel);
    half final_rim_alpha = final_fresnel;

    // Scanline Effect
    // 1 - this -> from bottom to up, without minus -> from up to bottom
    half2 uv_flow = 1 - (i.pos_world.xy - i.pivot_world.xy) * _FlowTilling.xy;
    uv_flow = uv_flow + _Time.y * _FlowSpeed.xy;
    float4 flow_rgba = tex2D(_FlowTex, uv_flow) * _FlowIntensity;

    // Flicker Effect: shine unstreadily
    half flicker = 1.0 + sin(_Time.y * _FlickerSpeed) * _FlickerIntensity;

    // final color: combine Rim light, Scanline, and Flicker
    float3 final_col = final_rim_color + flow_rgba.xyz;
    final_col.rgb *= flicker;
    float final_alpha = saturate(final_rim_alpha + flow_rgba.a + _InnerAlpha);
    return float4(final_col, final_alpha);
}
ENDCG
```

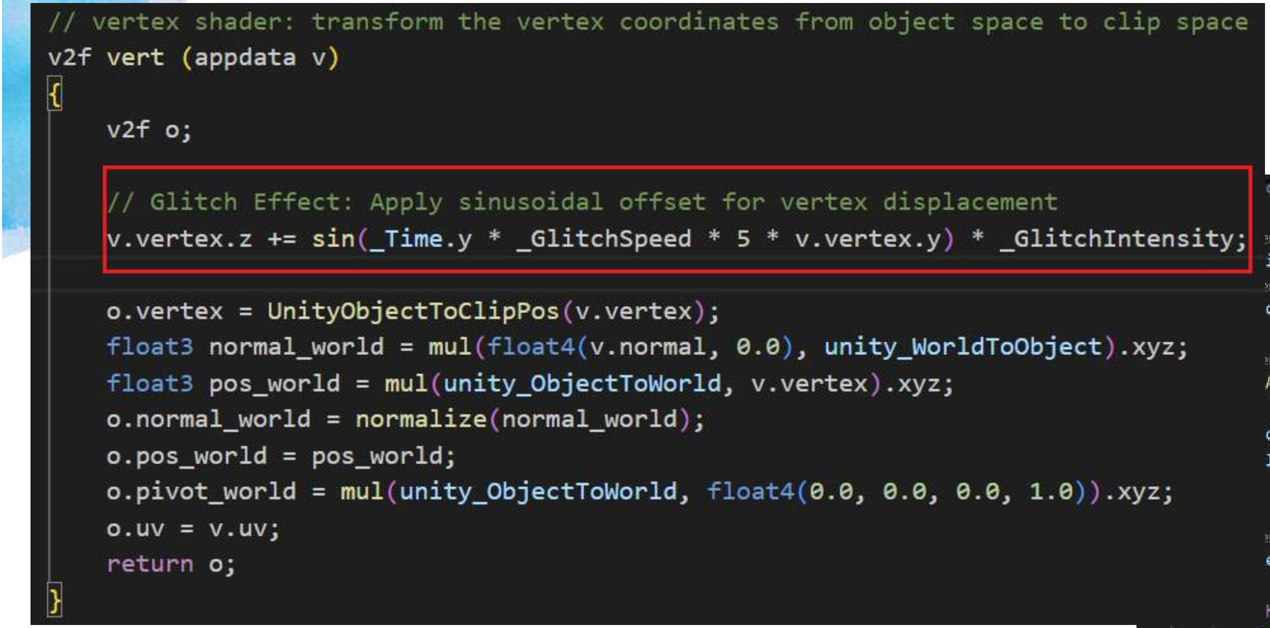
Glitch Effect - Random Shape Disturb



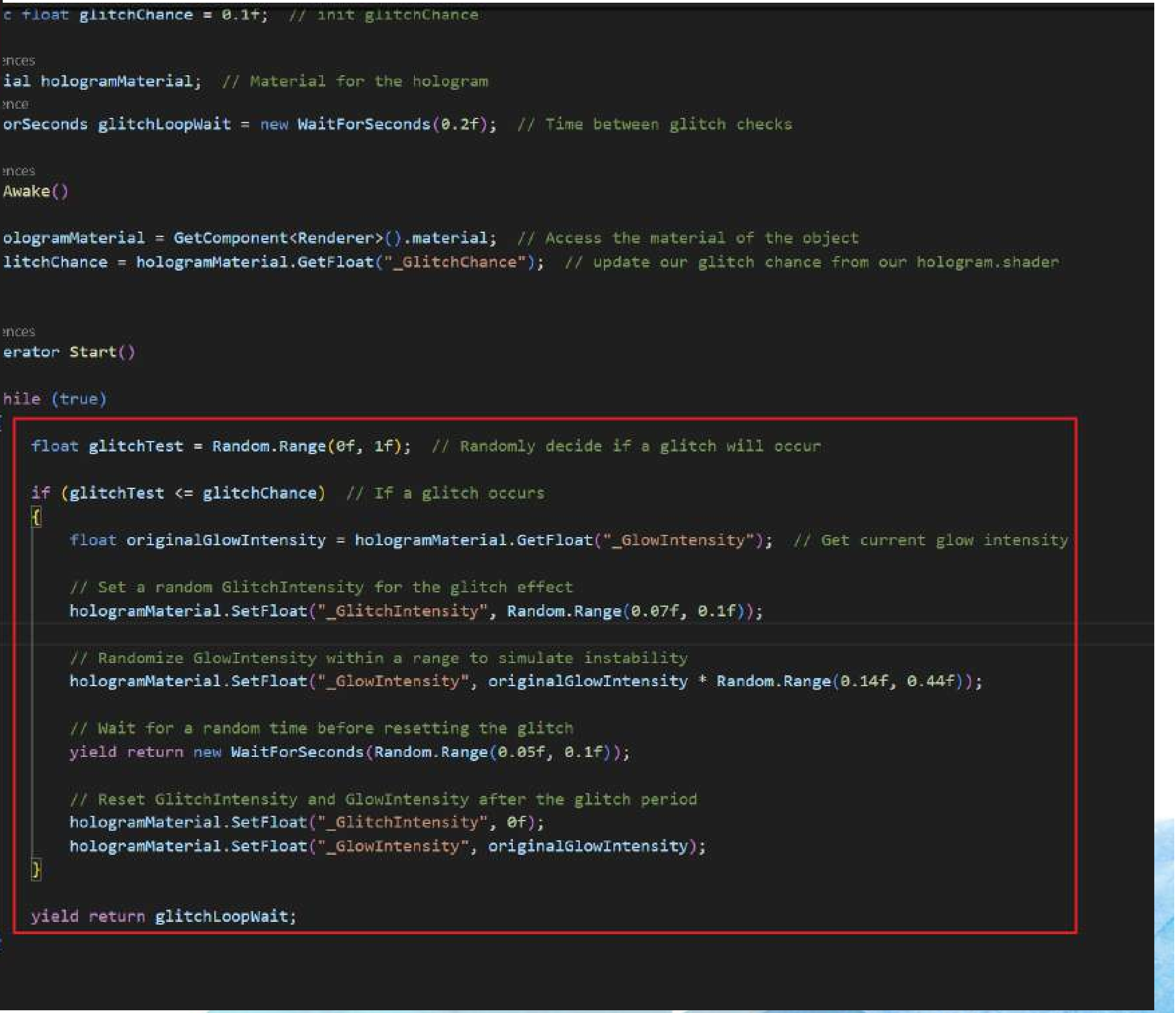




Glitch Effect - Random Shape Disturb

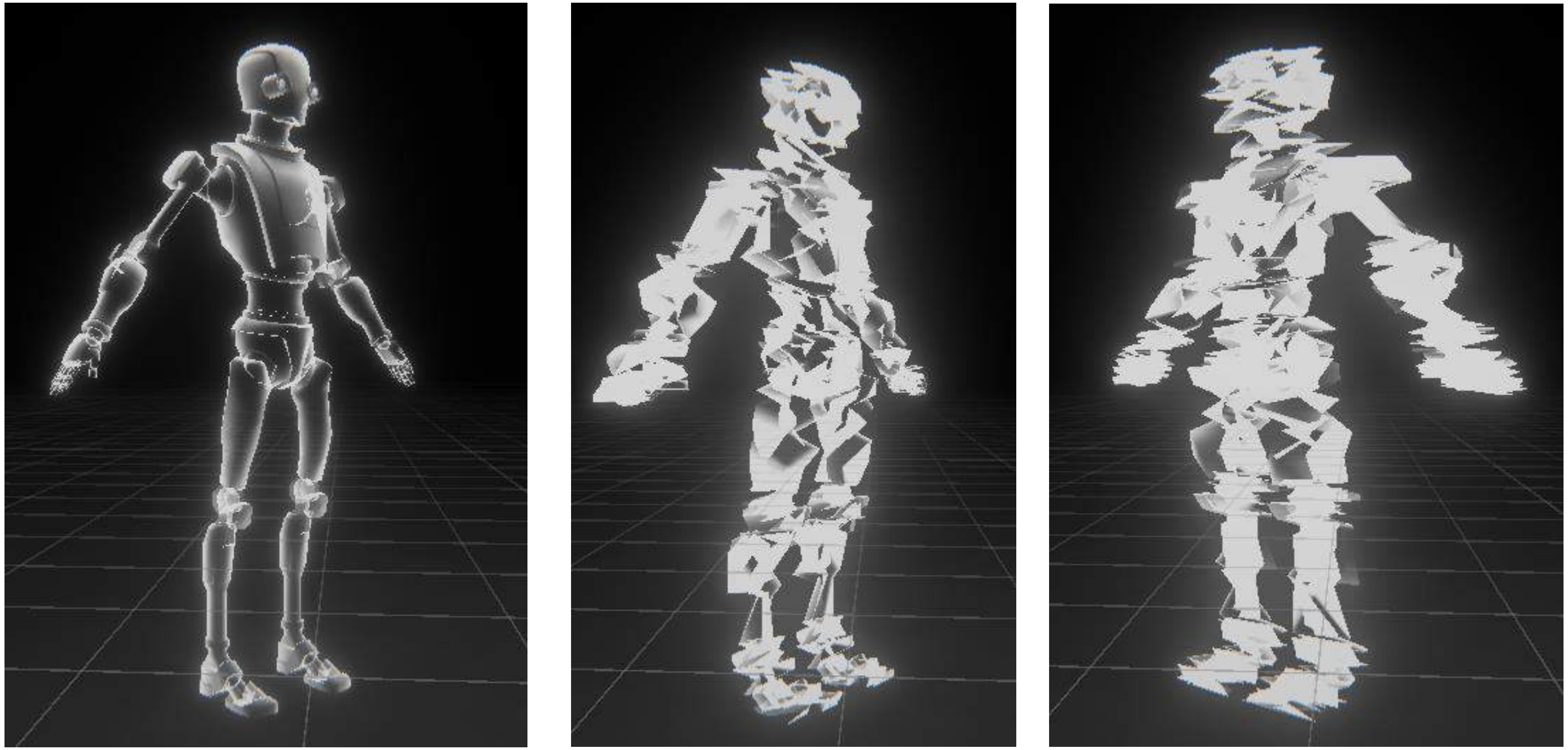


```
// vertex shader: transform the vertex coordinates from object space to clip space
v2f vert (appdata v)
{
    v2f o;

    // Glitch Effect: Apply sinusoidal offset for vertex displacement
    v.vertex.z += sin(_Time.y * _GlitchSpeed * 5 * v.vertex.y) * _GlitchIntensity;

    o.vertex = UnityObjectToClipPos(v.vertex);
    float3 normal_world = mul(float4(v.normal, 0.0), unity_WorldToObject).xyz;
    float3 pos_world = mul(unity_ObjectToWorld, v.vertex).xyz;
    o.normal_world = normalize(normal_world);
    o.pos_world = pos_world;
    o.pivot_world = mul(unity_ObjectToWorld, float4(0.0, 0.0, 0.0, 1.0)).xyz;
    o.uv = v.uv;
    return o;
}
```

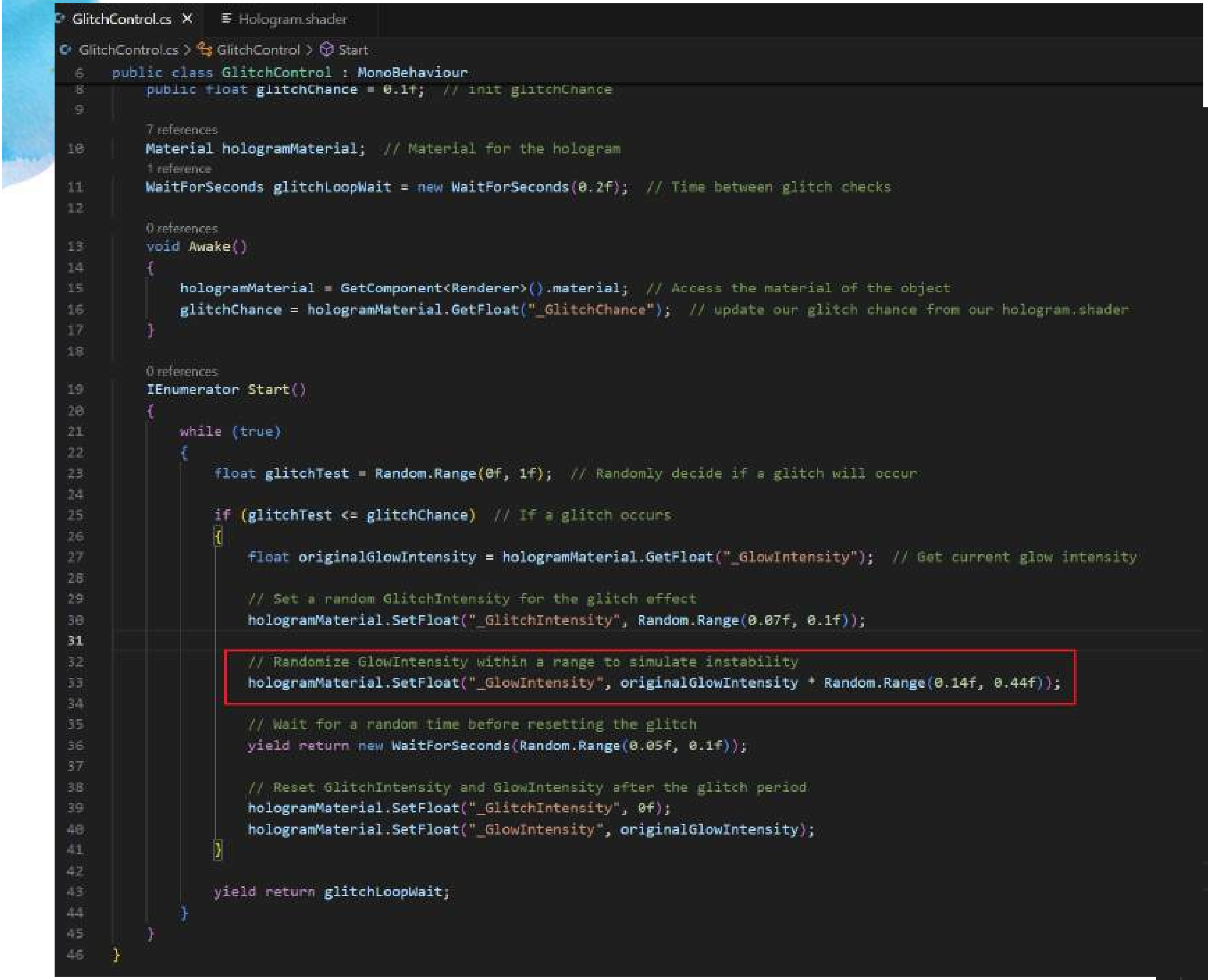


```
c float glitchChance = 0.1f;  // init glitchChance

ences
ial hologramMaterial;  // Material for the hologram
ence
orSeconds glitchLoopWait = new WaitForSeconds(0.2f);  // Time between glitch checks

ences
Awake()

ologramMaterial = GetComponent<Renderer>().material;  // Access the material of the object
litchChance = hologramMaterial.GetFloat("_GlitchChance");  // update our glitch chance from our hologram.shader


ences
erator Start()

hile (true)
{
    float glitchTest = Random.Range(0f, 1f);  // Randomly decide if a glitch will occur

    if (glitchTest <= glitchChance)  // If a glitch occurs
    {
        float originalGlowIntensity = hologramMaterial.GetFloat("_GlowIntensity");  // Get current glow intensity

        // Set a random GlitchIntensity for the glitch effect
        hologramMaterial.SetFloat("_GlitchIntensity", Random.Range(0.07f, 0.1f));

        // Randomize GlowIntensity within a range to simulate instability
        hologramMaterial.SetFloat("_GlowIntensity", originalGlowIntensity * Random.Range(0.14f, 0.44f));

        // Wait for a random time before resetting the glitch
        yield return new WaitForSeconds(Random.Range(0.05f, 0.1f));

        // Reset GlitchIntensity and GlowIntensity after the glitch period
        hologramMaterial.SetFloat("_GlitchIntensity", 0f);
        hologramMaterial.SetFloat("_GlowIntensity", originalGlowIntensity);
    }

    yield return glitchLoopWait;
}
    }
}
```

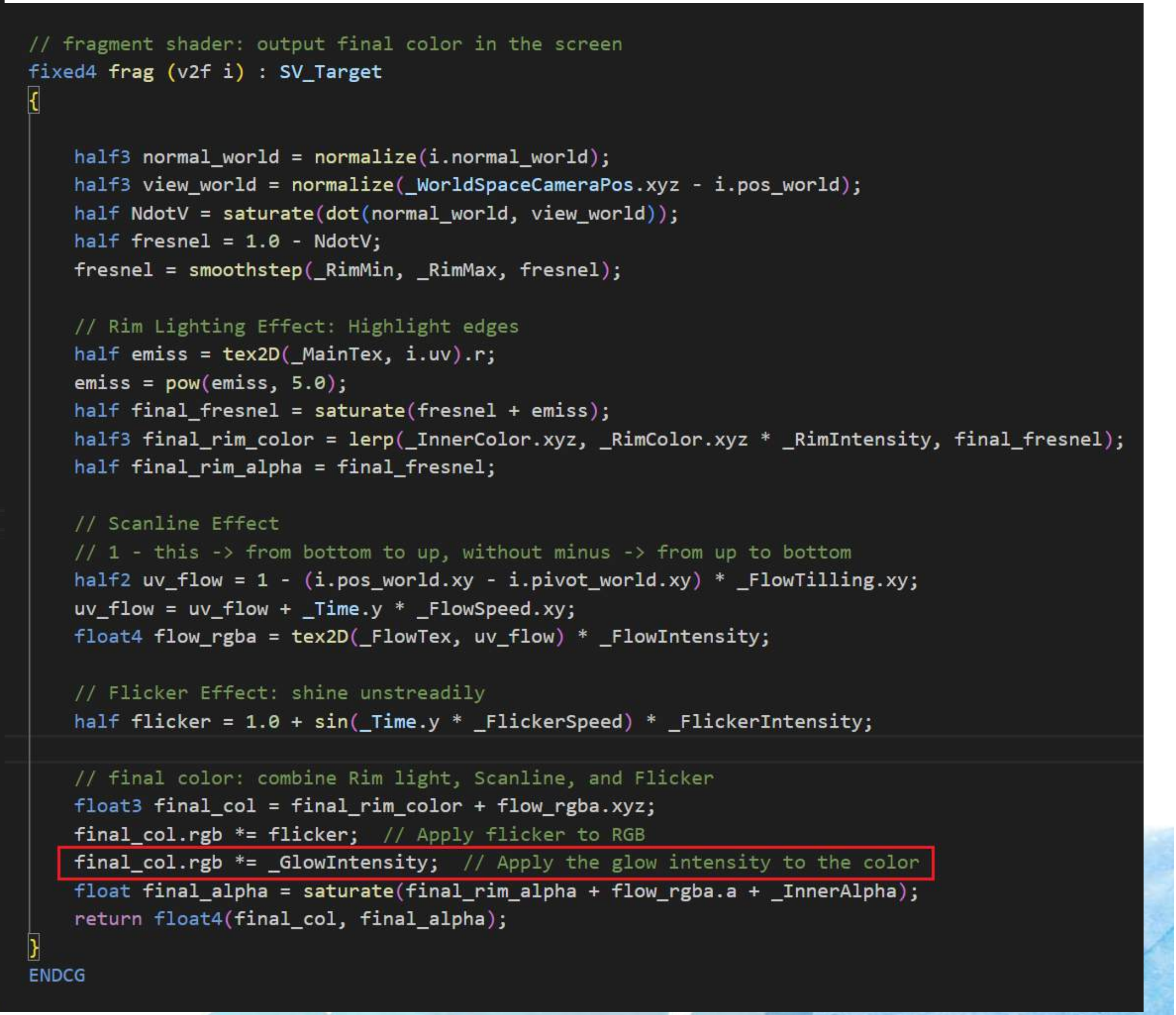
Glitch Effect - Random Emission Effect



Glitch Effect - Random Emission Effect

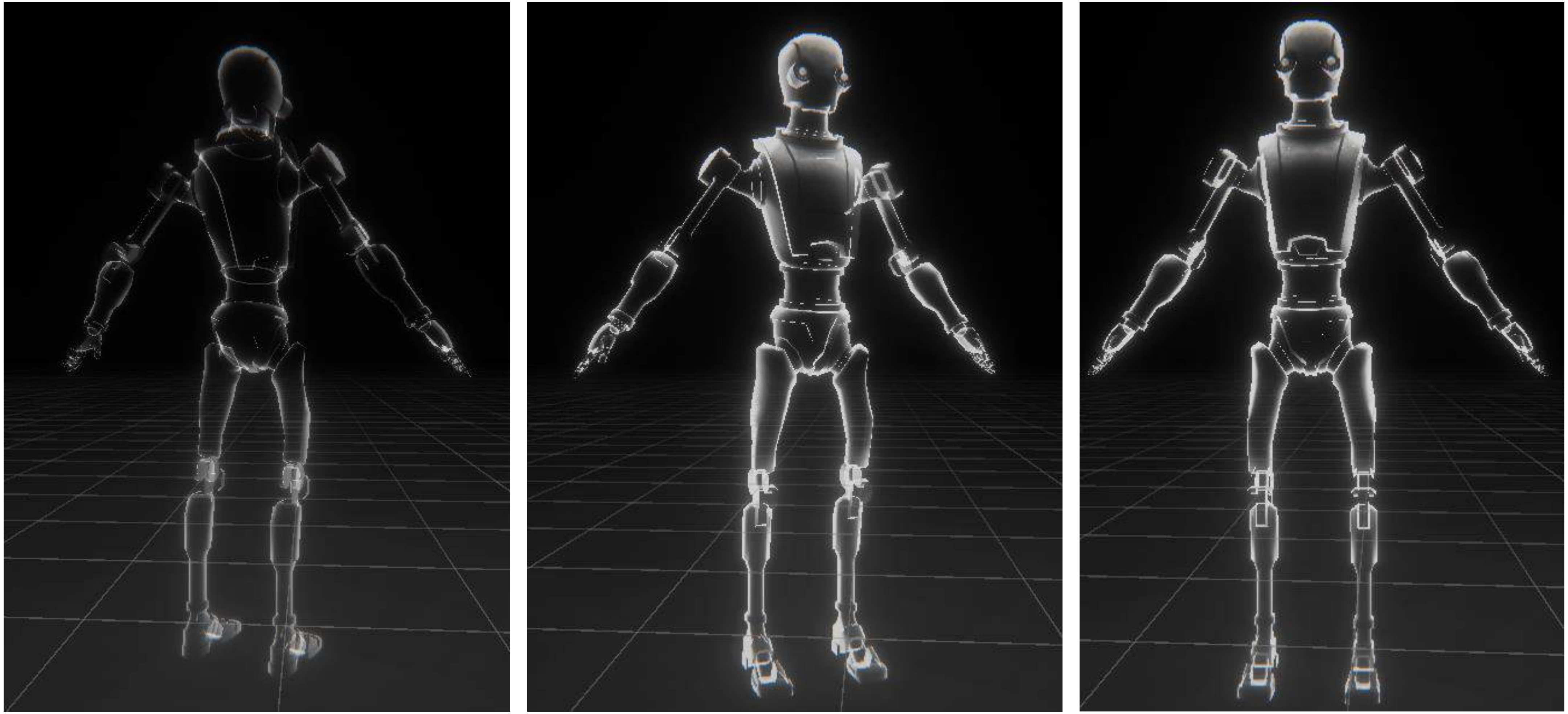


```csharp
public class GlitchControl : MonoBehaviour
    public float glitchChance = 0.1f;  // init glitchChance

    7 references
    Material hologramMaterial;  // Material for the hologram
    1 reference
    WaitForSeconds glitchLoopWait = new WaitForSeconds(0.2f);  // Time between glitch checks

    0 references
    void Awake()
    {
        hologramMaterial = GetComponent<Renderer>().material;  // Access the material of the object
        glitchChance = hologramMaterial.GetFloat("_GlitchChance");  // update our glitch chance from our hologram.shader
    }

    0 references
    IEnumerator Start()
    {
        while (true)
        {
            float glitchTest = Random.Range(0f, 1f);  // Randomly decide if a glitch will occur

            if (glitchTest <= glitchChance)  // If a glitch occurs
            {
                float originalGlowIntensity = hologramMaterial.GetFloat("_GlowIntensity");  // Get current glow intensity

                // Set a random GlitchIntensity for the glitch effect
                hologramMaterial.SetFloat("_GlitchIntensity", Random.Range(0.07f, 0.1f));

                // Randomize GlowIntensity within a range to simulate instability
                hologramMaterial.SetFloat("_GlowIntensity", originalGlowIntensity * Random.Range(0.14f, 0.44f));

                // Wait for a random time before resetting the glitch
                yield return new WaitForSeconds(Random.Range(0.05f, 0.1f));

                // Reset GlitchIntensity and GlowIntensity after the glitch period
                hologramMaterial.SetFloat("_GlitchIntensity", 0f);
                hologramMaterial.SetFloat("_GlowIntensity", originalGlowIntensity);
            }

            yield return glitchLoopWait;
        }
    }
}
```

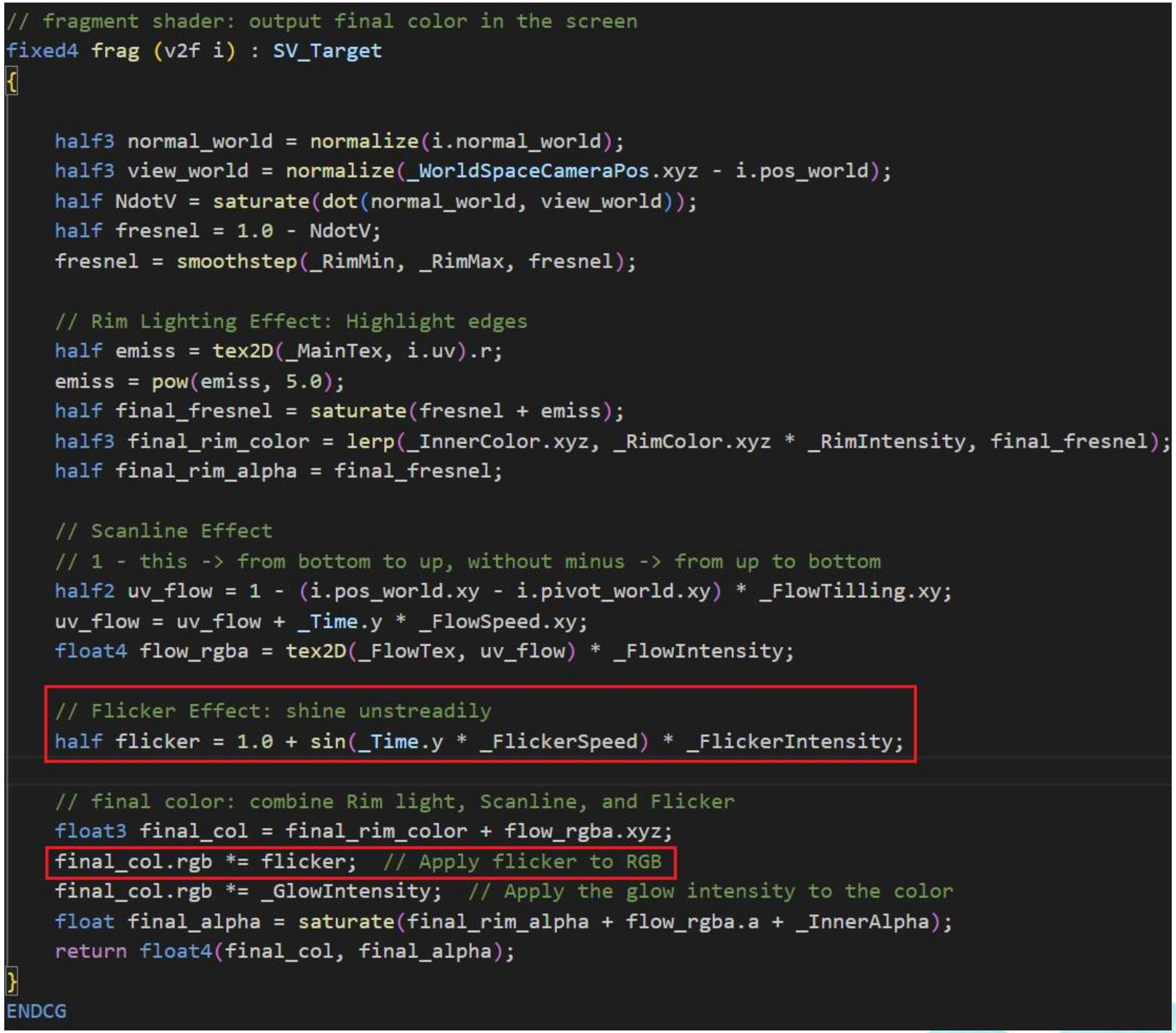


```hlsl
// fragment shader: output final color in the screen
fixed4 frag (v2f i) : SV_Target
{

    half3 normal_world = normalize(i.normal_world);
    half3 view_world = normalize(_WorldSpaceCameraPos.xyz - i.pos_world);
    half NdotV = saturate(dot(normal_world, view_world));
    half fresnel = 1.0 - NdotV;
    fresnel = smoothstep(_RimMin, _RimMax, fresnel);

    // Rim Lighting Effect: Highlight edges
    half emiss = tex2D(_MainTex, i.uv).r;
    emiss = pow(emiss, 5.0);
    half final_fresnel = saturate(fresnel + emiss);
    half3 final_rim_color = lerp(_InnerColor.xyz, _RimColor.xyz * _RimIntensity, final_fresnel);
    half final_rim_alpha = final_fresnel;

    // Scanline Effect
    // 1 - this -> from bottom to up, without minus -> from up to bottom
    half2 uv_flow = 1 - (i.pos_world.xy - i.pivot_world.xy) * _FlowTilling.xy;
    uv_flow = uv_flow + _Time.y * _FlowSpeed.xy;
    float4 flow_rgba = tex2D(_FlowTex, uv_flow) * _FlowIntensity;

    // Flicker Effect: shine unsteadily
    half flicker = 1.0 + sin(_Time.y * _FlickerSpeed) * _FlickerIntensity;

    // final color: combine Rim light, Scanline, and Flicker
    float3 final_col = final_rim_color + flow_rgba.xyz;
    final_col.rgb *= flicker;  // Apply flicker to RGB
    final_col.rgb *= _GlowIntensity;  // Apply the glow intensity to the color
    float final_alpha = saturate(final_rim_alpha + flow_rgba.a + _InnerAlpha);
    return float4(final_col, final_alpha);
}
ENDCG
}
```

Glitch Effect - Periodic Flicker Effect

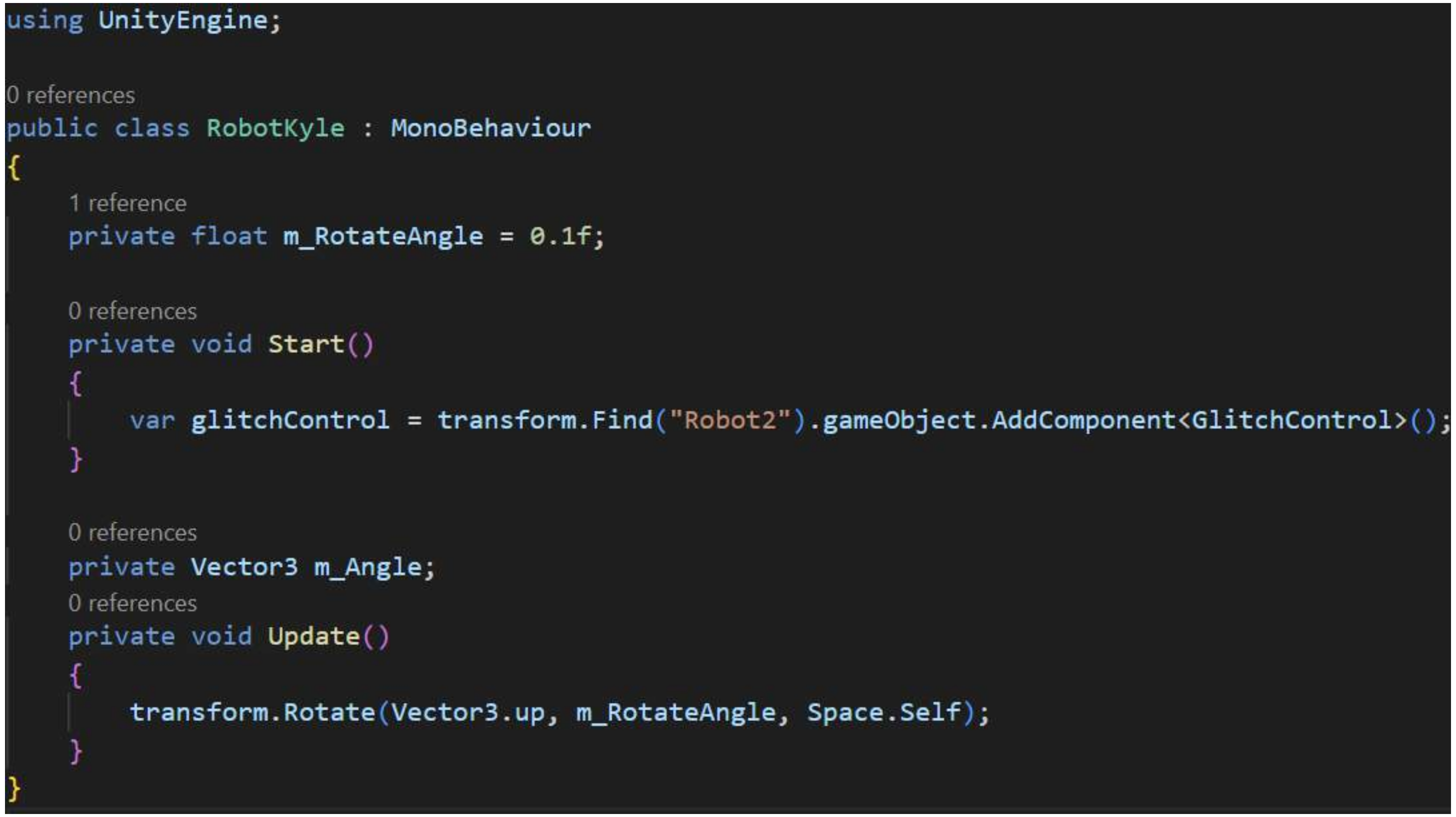


Glitch Effect - Periodic Flicker Effect



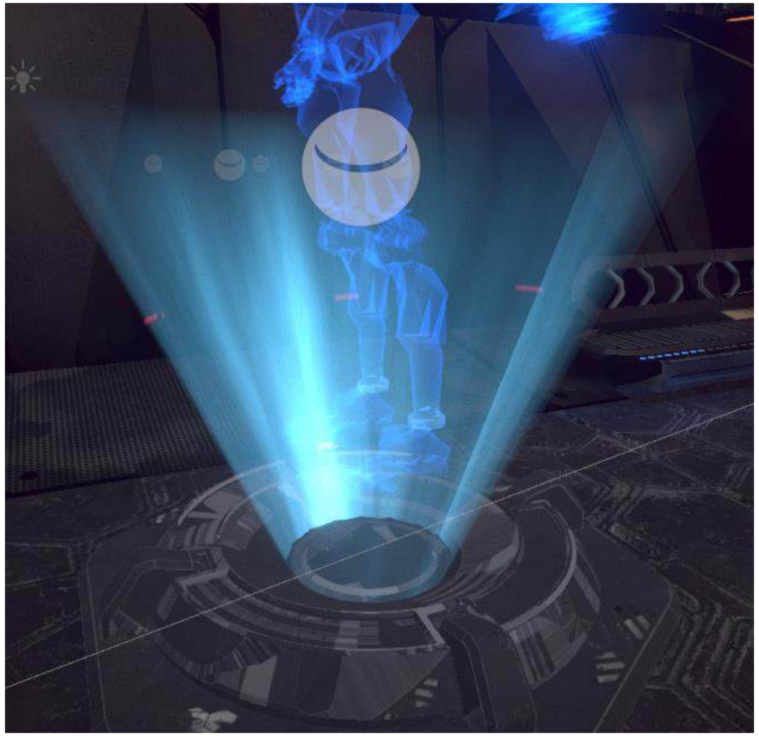
```
// fragment shader: output final color in the screen
fixed4 frag (v2f i) : SV_Target
{

    half3 normal_world = normalize(i.normal_world);
    half3 view_world = normalize(_WorldSpaceCameraPos.xyz - i.pos_world);
    half NdotV = saturate(dot(normal_world, view_world));
    half fresnel = 1.0 - NdotV;
    fresnel = smoothstep(_RimMin, _RimMax, fresnel);

    // Rim Lighting Effect: Highlight edges
    half emiss = tex2D(_MainTex, i.uv).r;
    emiss = pow(emiss, 5.0);
    half final_fresnel = saturate(fresnel + emiss);
    half3 final_rim_color = lerp(_InnerColor.xyz, _RimColor.xyz * _RimIntensity, final_fresnel);
    half final_rim_alpha = final_fresnel;

    // Scanline Effect
    // 1 - this -> from bottom to up, without minus -> from up to bottom
    half2 uv_flow = 1 - (i.pos_world.xy - i.pivot_world.xy) * _FlowTilling.xy;
    uv_flow = uv_flow + _Time.y * _FlowSpeed.xy;
    float4 flow_rgba = tex2D(_FlowTex, uv_flow) * _FlowIntensity;

    // Flicker Effect: shine unstreadily
    half flicker = 1.0 + sin(_Time.y * _FlickerSpeed) * _FlickerIntensity;

    // final color: combine Rim light, Scanline, and Flicker
    float3 final_col = final_rim_color + flow_rgba.xyz;
    final_col.rgb *= flicker;  // Apply flicker to RGB
    final_col.rgb *= _GlowIntensity;  // Apply the glow intensity to the color
    float final_alpha = saturate(final_rim_alpha + flow_rgba.a + _InnerAlpha);
    return float4(final_col, final_alpha);
}
ENDCG
```

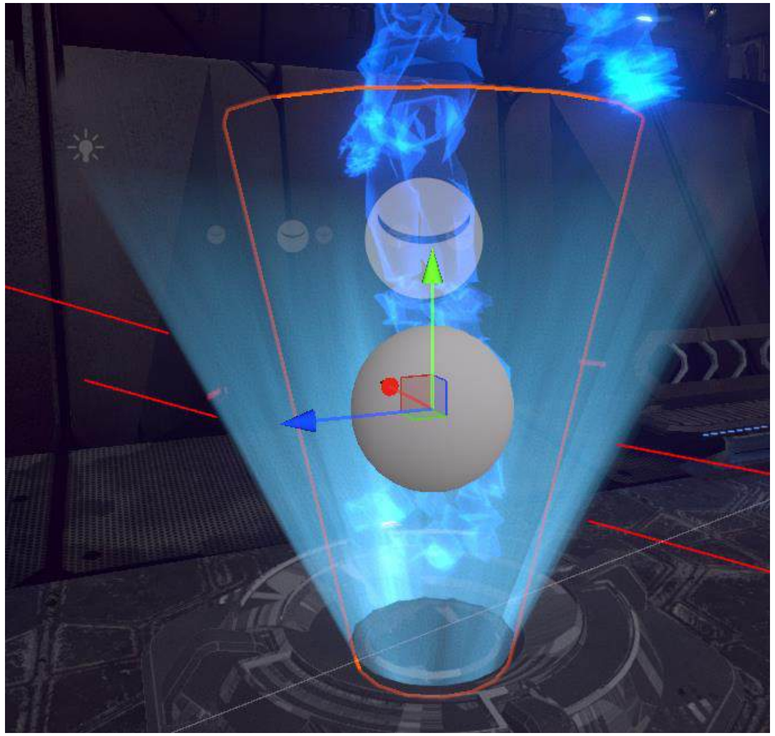
Character Rotation

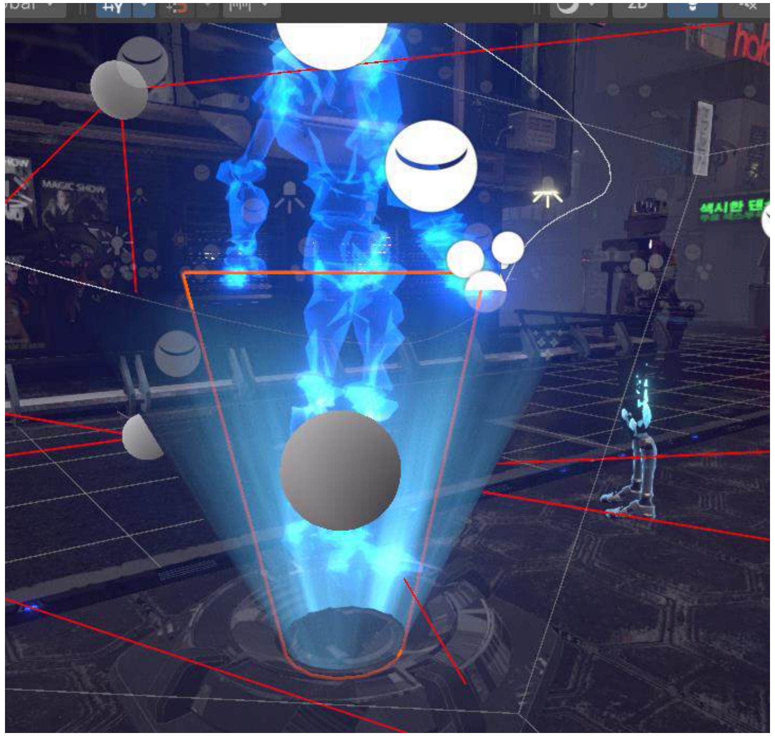


```
using UnityEngine;

0 references
public class RobotKyle : MonoBehaviour
{
    1 reference
    private float m_RotateAngle = 0.1f;

    0 references
    private void Start()
    {
        var glitchControl = transform.Find("Robot2").gameObject.AddComponent<GlitchControl>();
    }

    0 references
    private Vector3 m_Angle;
    0 references
    private void Update()
    {
        transform.Rotate(Vector3.up, m_RotateAngle, Space.Self);
    }
}
```

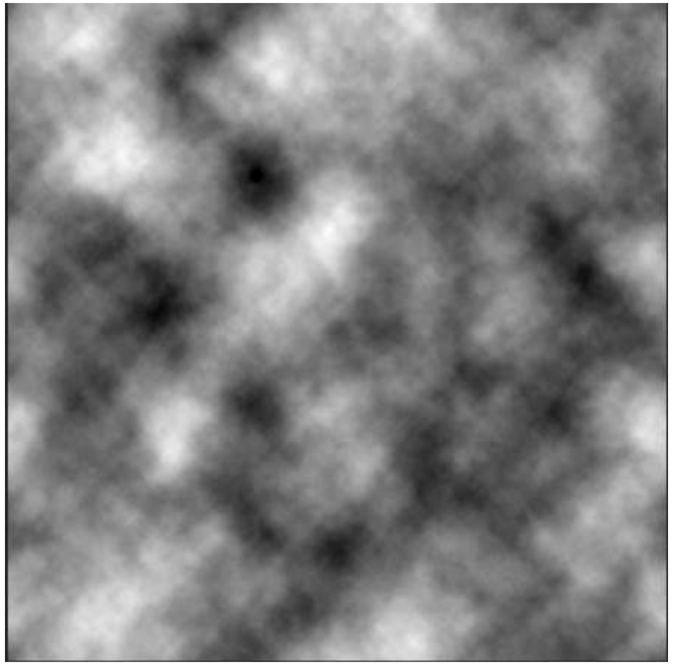
Lamp Effect =
Emissison Map + Vertex Normal Offset

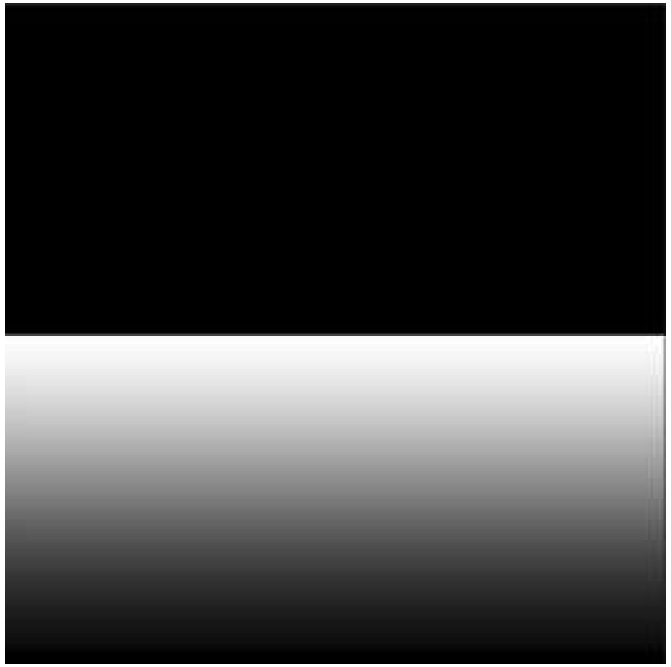


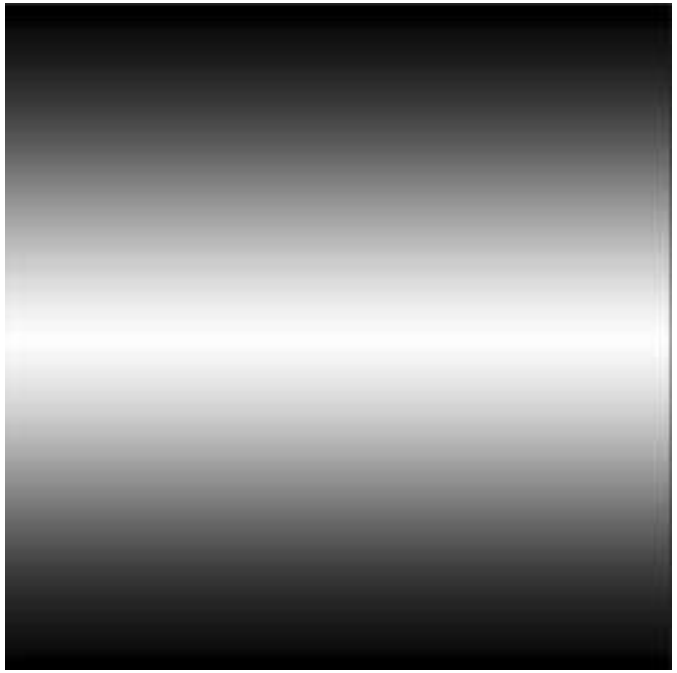




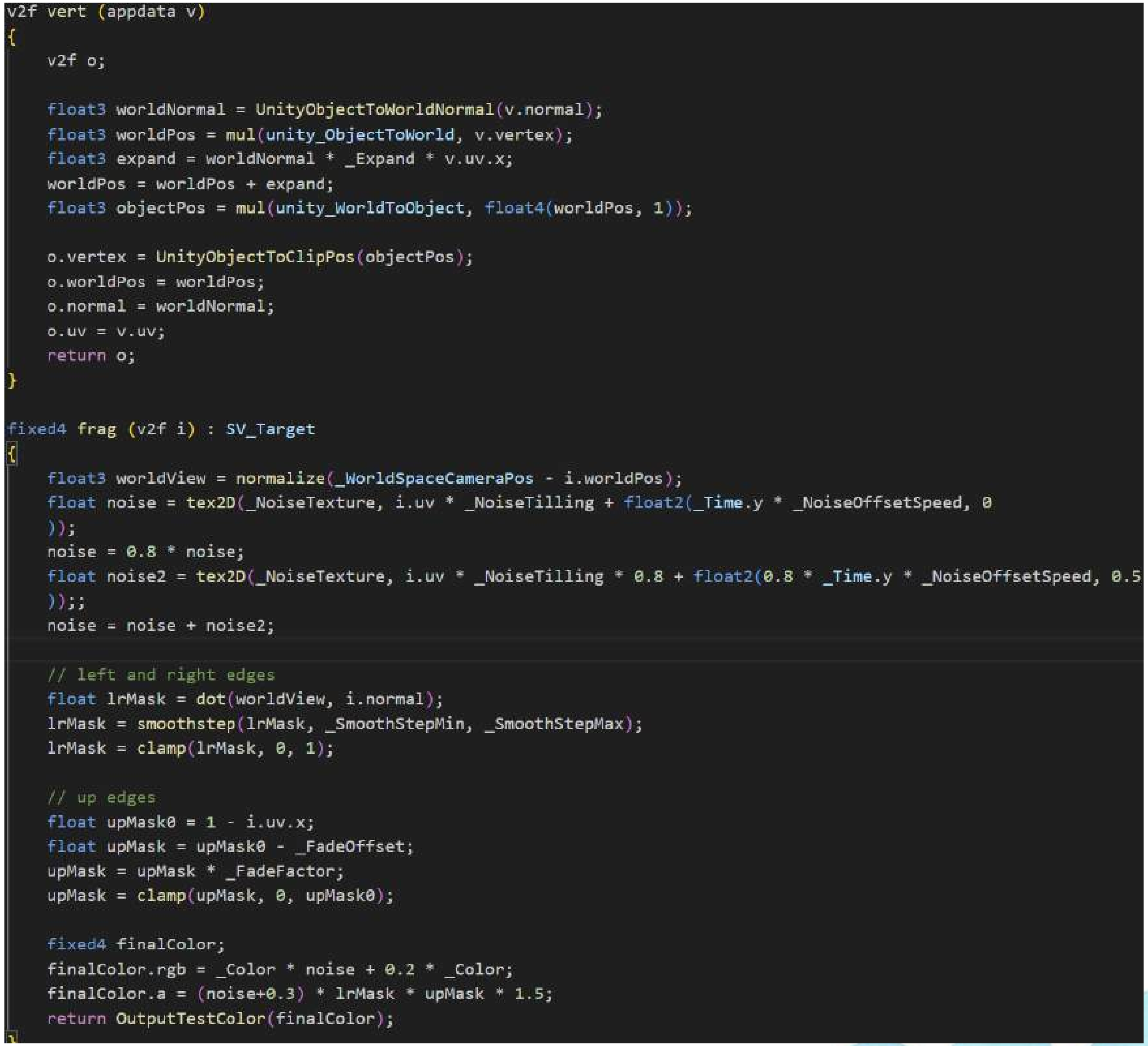
Lamp Effect







Lamp Effect



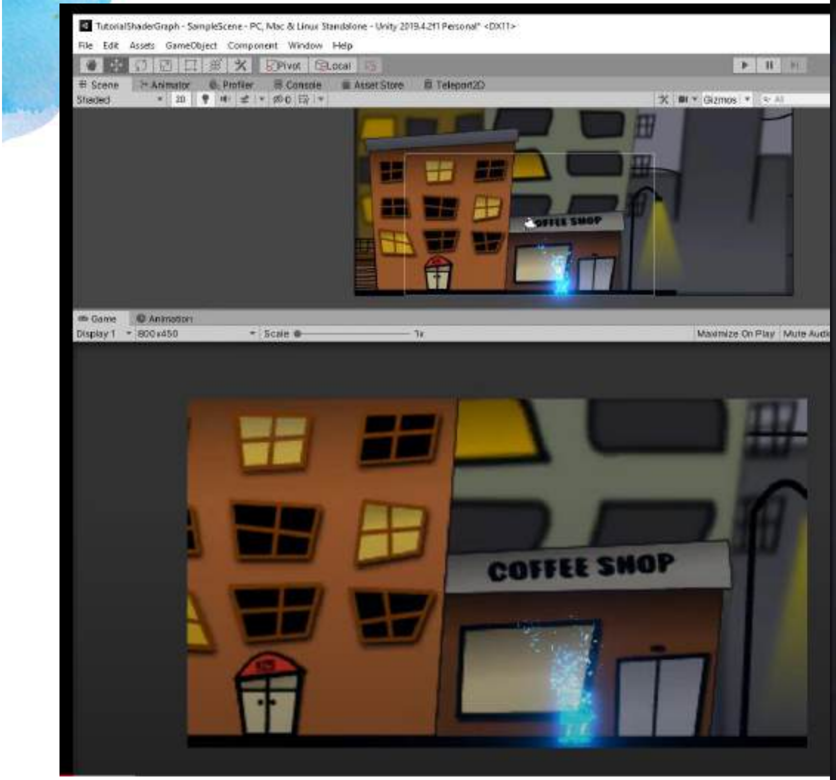
```
v2f vert (appdata v)
{
    v2f o;

    float3 worldNormal = UnityObjectToWorldNormal(v.normal);
    float3 worldPos = mul(unity_ObjectToWorld, v.vertex);
    float3 expand = worldNormal * _Expand * v.uv.x;
    worldPos = worldPos + expand;
    float3 objectPos = mul(unity_WorldToObject, float4(worldPos, 1));

    o.vertex = UnityObjectToClipPos(objectPos);
    o.worldPos = worldPos;
    o.normal = worldNormal;
    o.uv = v.uv;
    return o;
}

fixed4 frag (v2f i) : SV_Target
{
    float3 worldView = normalize(_WorldSpaceCameraPos - i.worldPos);
    float noise = tex2D(_NoiseTexture, i.uv * _NoiseTilling + float2(_Time.y * _NoiseOffsetSpeed, 0
    ));
    noise = 0.8 * noise;
    float noise2 = tex2D(_NoiseTexture, i.uv * _NoiseTilling * 0.8 + float2(0.8 * _Time.y * _NoiseOffsetSpeed, 0.5
    ));;
    noise = noise + noise2;

    // left and right edges
    float lrMask = dot(worldView, i.normal);
    lrMask = smoothstep(lrMask, _SmoothStepMin, _SmoothStepMax);
    lrMask = clamp(lrMask, 0, 1);

    // up edges
    float upMask0 = 1 - i.uv.x;
    float upMask = upMask0 - _FadeOffset;
    upMask = upMask * _FadeFactor;
    upMask = clamp(upMask, 0, upMask0);

    fixed4 finalColor;
    finalColor.rgb = _Color * noise + 0.2 * _Color;
    finalColor.a = (noise+0.3) * lrMask * upMask * 1.5;
    return OutputTestColor(finalColor);
}
```

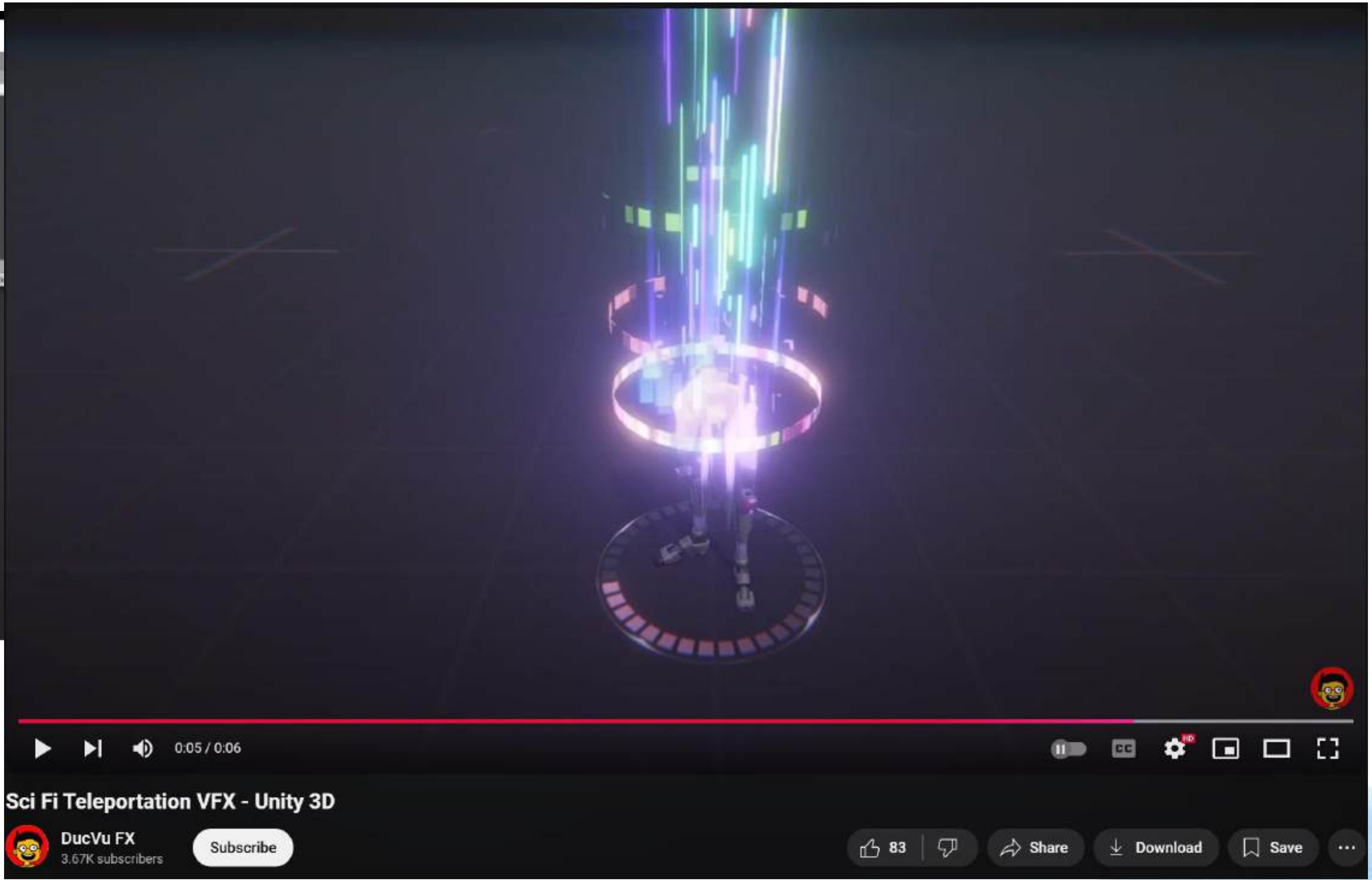
02

Teleport Effect

Gouttham Nambirajan

Concept


TutorialShaderGraph - SampleScene - PC, Mac & Linux Standalone - Unity 2019.4.2f1 Personal* <DX11>
File
Edit
Assets
GameObject
Component
Window
Help
Pivot
Local
Scene
Animator
Profiler
Console
Asset Store
Teleport2D
Shaded
2D
Gizmos
All
COFFEE SHOP
Game
Animation
Display 1
800x450
Scale
1x
Maximize On Play
Mute Audi
COFFEE SHOP


0:05 / 0:06
Sci Fi Teleportation VFX - Unity 3D
DucVu FX
3.67K subscribers
Subscribe
83
Share
Download
Save

Teleport Effect

Shading and Lighting

○○○○○○○○○○

Lambertian and Blinn-Phong lighting models (like PBR)

Dissolve Effect

○○○○○○○○○○

Noise-based Distortion, Emission Glow, and Rim Lighting

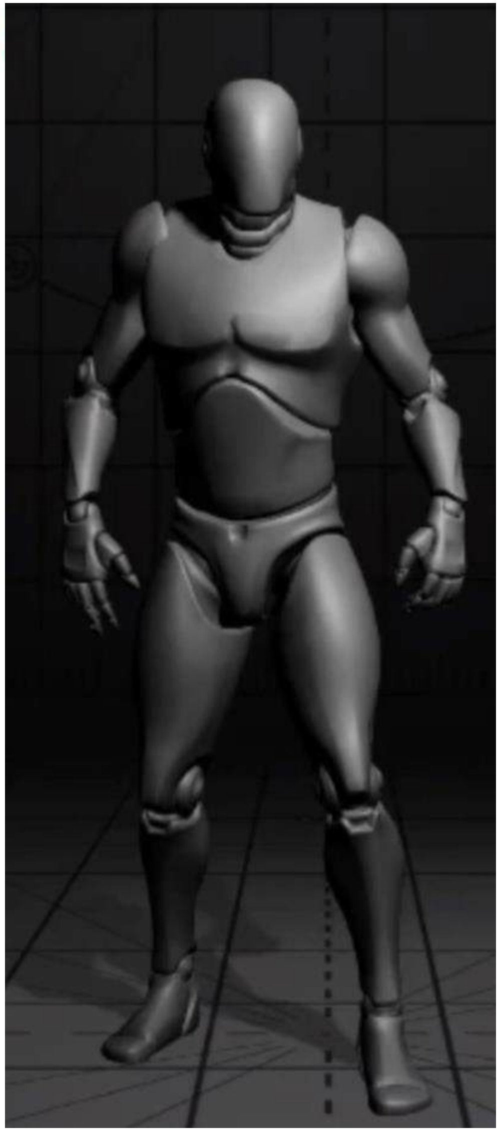
Vertex Stretching

○○○○○○○○○○

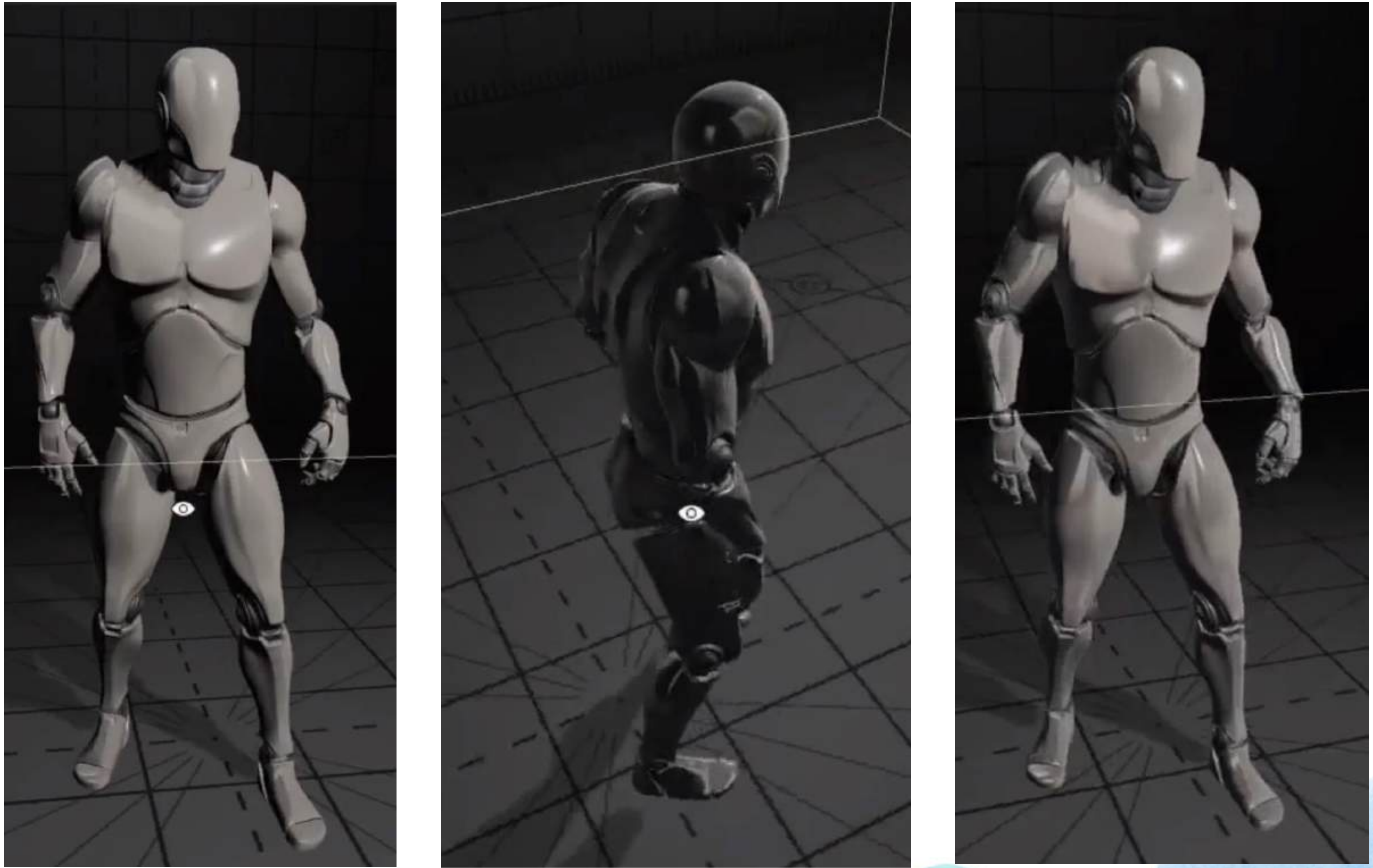
Dynamically stretches vertices

Shading and Lighting

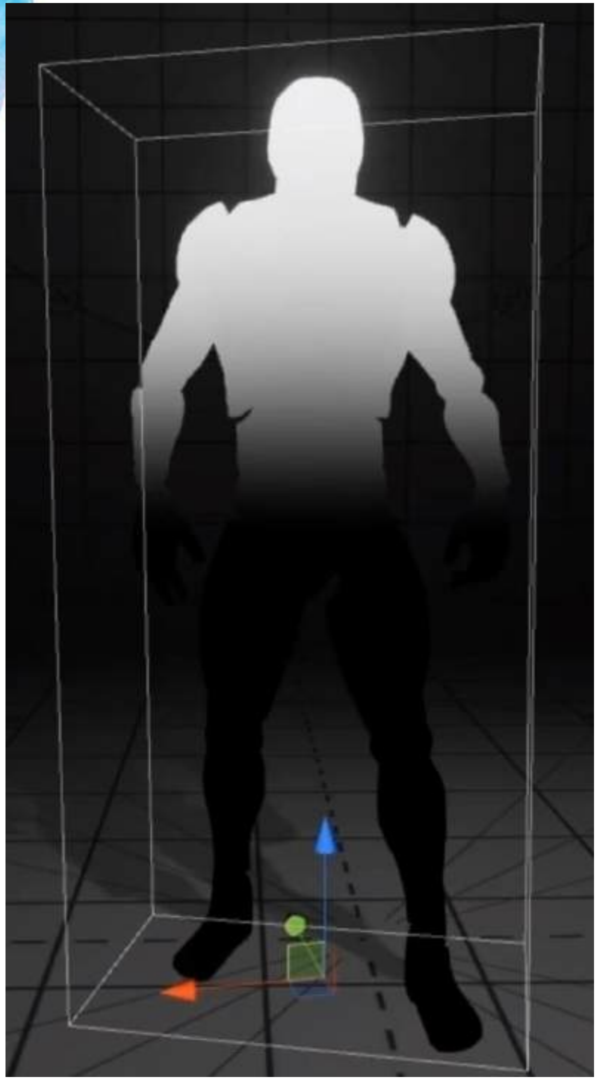
Original



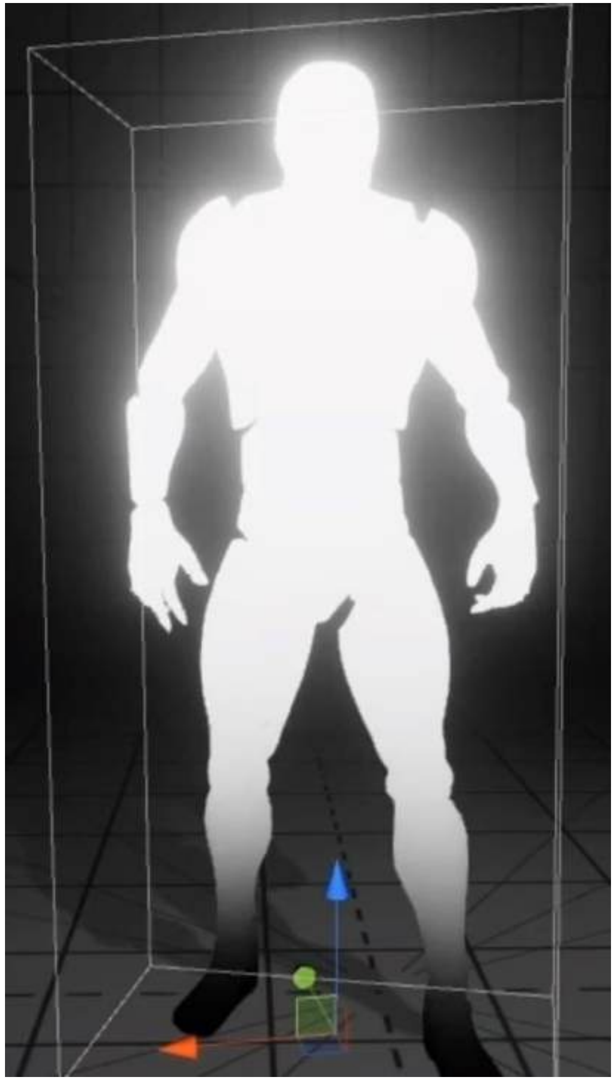
After adding Blinn-Phong and Lambert lighting models



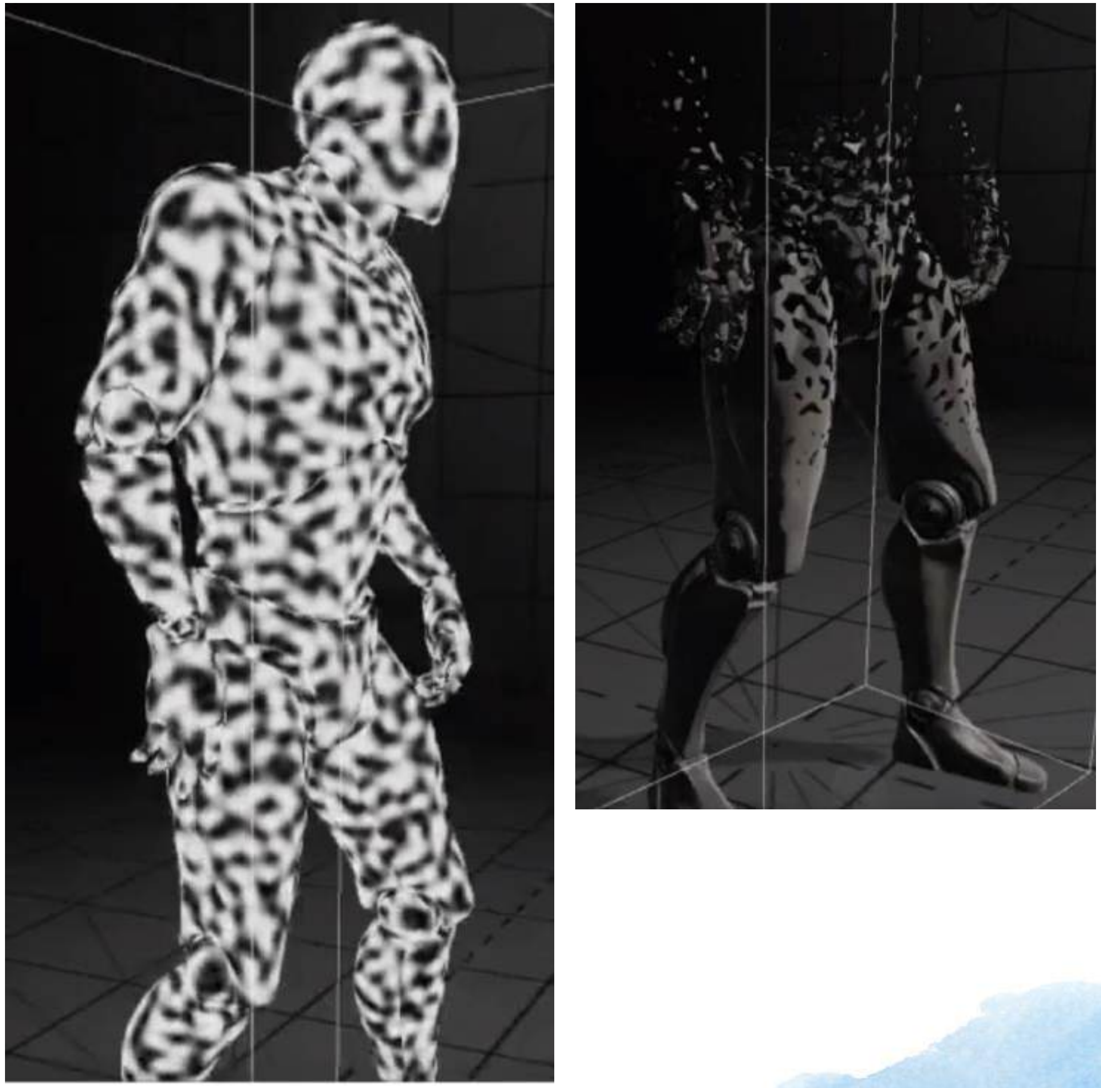
Dessolve Effect



Manupulate Object world Position

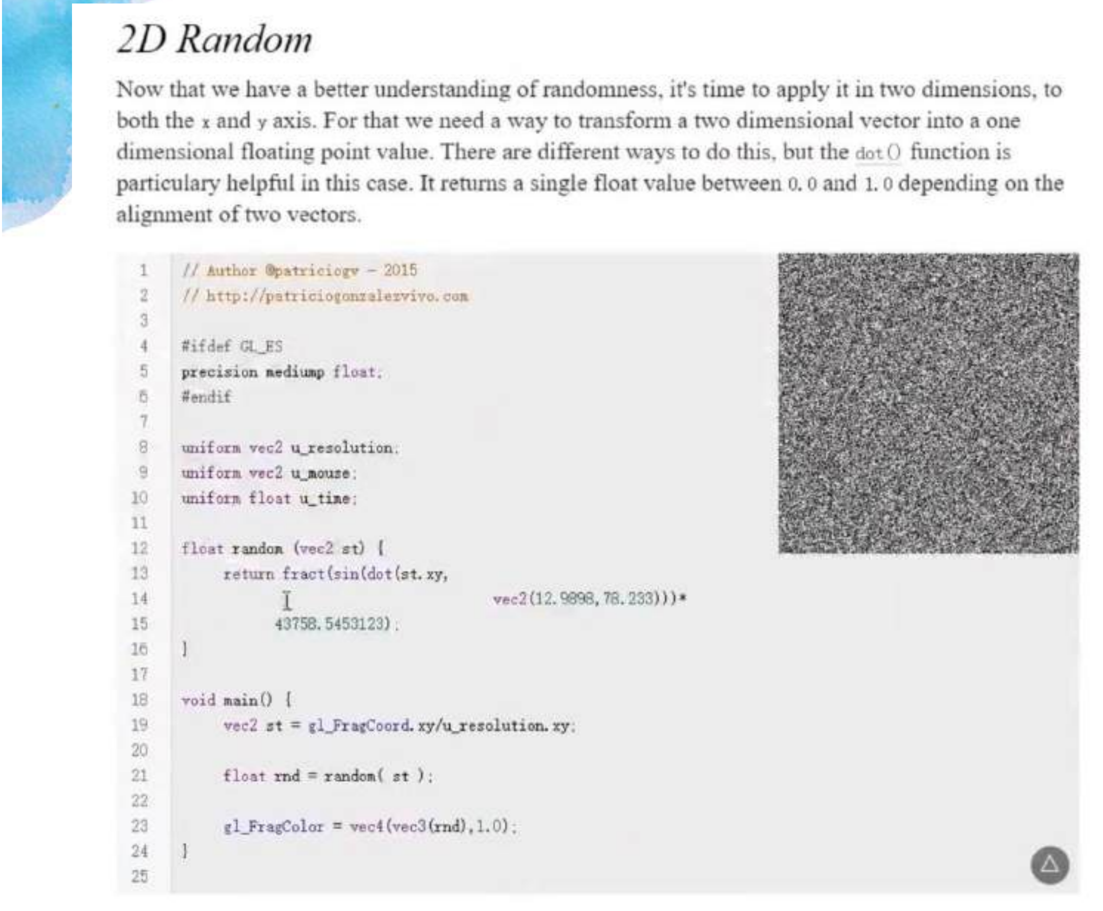


Adding rim lighting effect w.r.t. to position



Adding 3D noises map

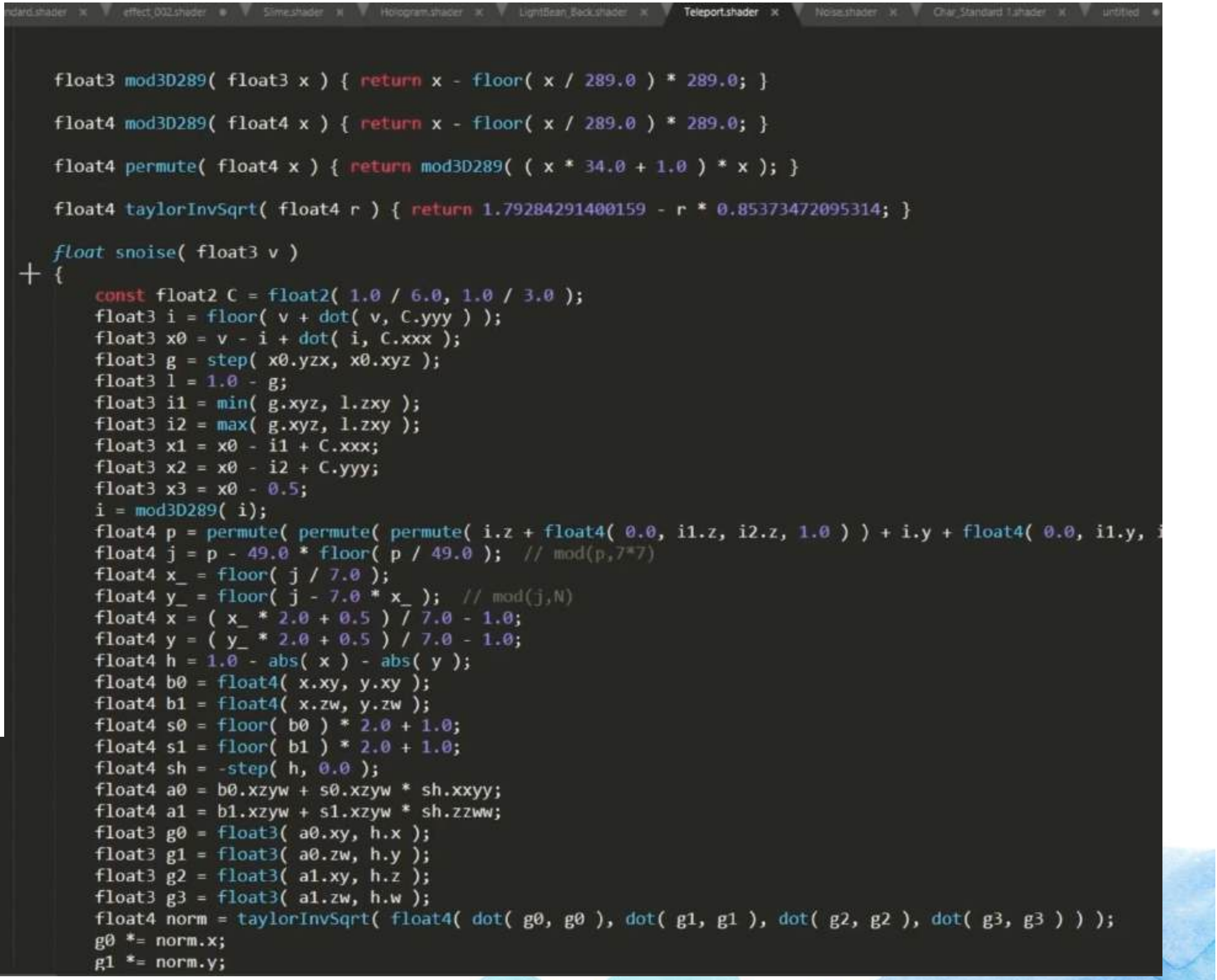
Dessolve Effect - 3D noises



2D Random

Now that we have a better understanding of randomness, it's time to apply it in two dimensions, to both the x and y axis. For that we need a way to transform a two dimensional vector into a one dimensional floating point value. There are different ways to do this, but the dot() function is particulary helpful in this case. It returns a single float value between 0.0 and 1.0 depending on the alignment of two vectors.

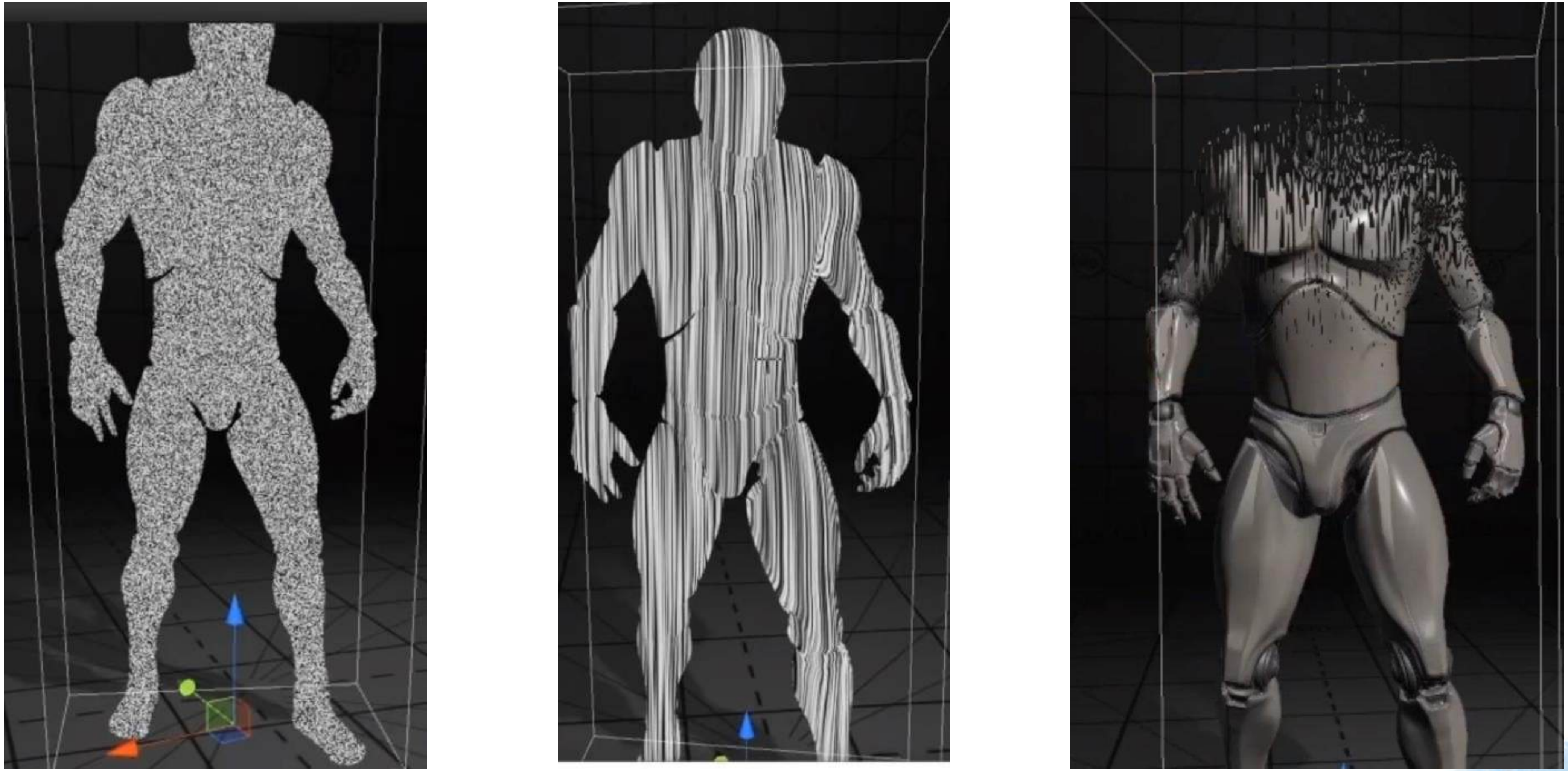
```
// Author @patriciogv - 2015
// http://patriciogonzalezvivo.com

#ifdef GL_ES
precision mediump float;
#endif

uniform vec2 u_resolution;
uniform vec2 u_mouse;
uniform float u_time;

float random (vec2 st) {
    return fract(sin(dot(st.xy,
                                    vec2(12.9898,78.233)))*
        43758.5453123);
}

void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;

    float rnd = random( st );

    gl_FragColor = vec4(vec3(rnd),1.0);
}
```



```
float3 mod3D289( float3 x ) { return x - floor( x / 289.0 ) * 289.0; }

float4 mod3D289( float4 x ) { return x - floor( x / 289.0 ) * 289.0; }

float4 permute( float4 x ) { return mod3D289( ( x * 34.0 + 1.0 ) * x ); }

float4 taylorInvSqrt( float4 r ) { return 1.79284291400159 - r * 0.85373472095314; }

float snoise( float3 v )
{
    const float2 C = float2( 1.0 / 6.0, 1.0 / 3.0 );
    float3 i = floor( v + dot( v, C.yyy ) );
    float3 x0 = v - i + dot( i, C.xxx );
    float3 g = step( x0.yzx, x0.xyz );
    float3 l = 1.0 - g;
    float3 i1 = min( g.xyz, l.zxy );
    float3 i2 = max( g.xyz, l.zxy );
    float3 x1 = x0 - i1 + C.xxx;
    float3 x2 = x0 - i2 + C.yyy;
    float3 x3 = x0 - 0.5;
    i = mod3D289( i);
    float4 p = permute( permute( permute( i.z + float4( 0.0, i1.z, i2.z, 1.0 ) ) + i.y + float4( 0.0, i1.y, i
    float4 j = p - 49.0 * floor( p / 49.0 );  // mod(p,7*7)
    float4 x_ = floor( j / 7.0 );
    float4 y_ = floor( j - 7.0 * x_ );  // mod(j,N)
    float4 x = ( x_ * 2.0 + 0.5 ) / 7.0 - 1.0;
    float4 y = ( y_ * 2.0 + 0.5 ) / 7.0 - 1.0;
    float4 h = 1.0 - abs( x ) - abs( y );
    float4 b0 = float4( x.xy, y.xy );
    float4 b1 = float4( x.zw, y.zw );
    float4 s0 = floor( b0 ) * 2.0 + 1.0;
    float4 s1 = floor( b1 ) * 2.0 + 1.0;
    float4 sh = -step( h, 0.0 );
    float4 a0 = b0.xzyw + s0.xzyw * sh.xxyy;
    float4 a1 = b1.xzyw + s1.xzyw * sh.zzww;
    float3 g0 = float3( a0.xy, h.x );
    float3 g1 = float3( a0.zw, h.y );
    float3 g2 = float3( a1.xy, h.z );
    float3 g3 = float3( a1.zw, h.w );
    float4 norm = taylorInvSqrt( float4( dot( g0, g0 ), dot( g1, g1 ), dot( g2, g2 ), dot( g3, g3 ) ) );
    g0 *= norm.x;
    g1 *= norm.y;
```

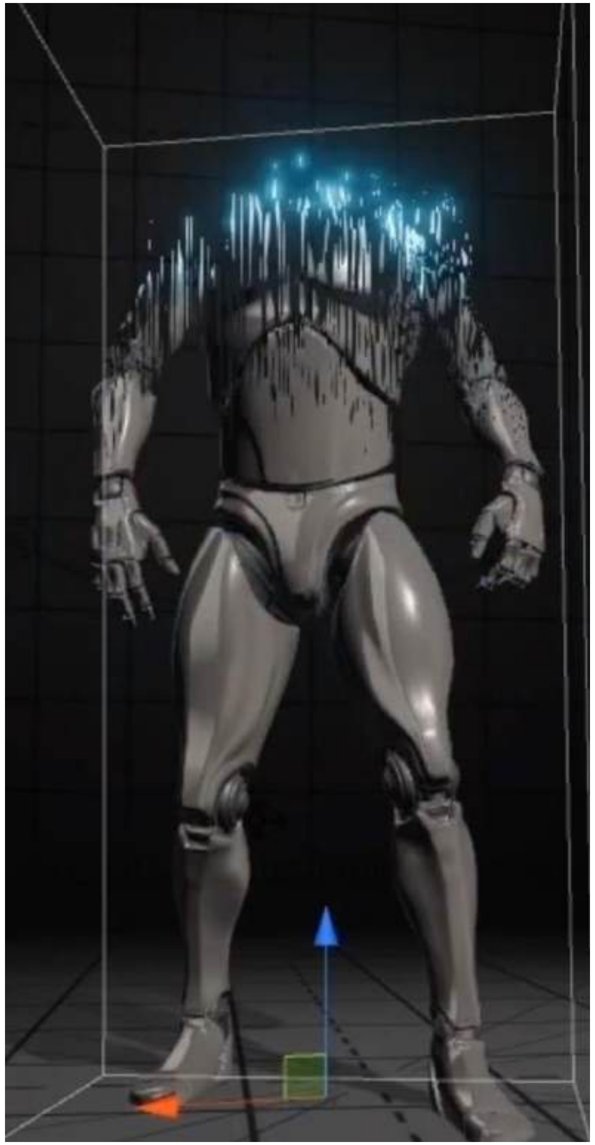
Dessolve Effect - 3D noises

Scale the noise tilling along X-axis

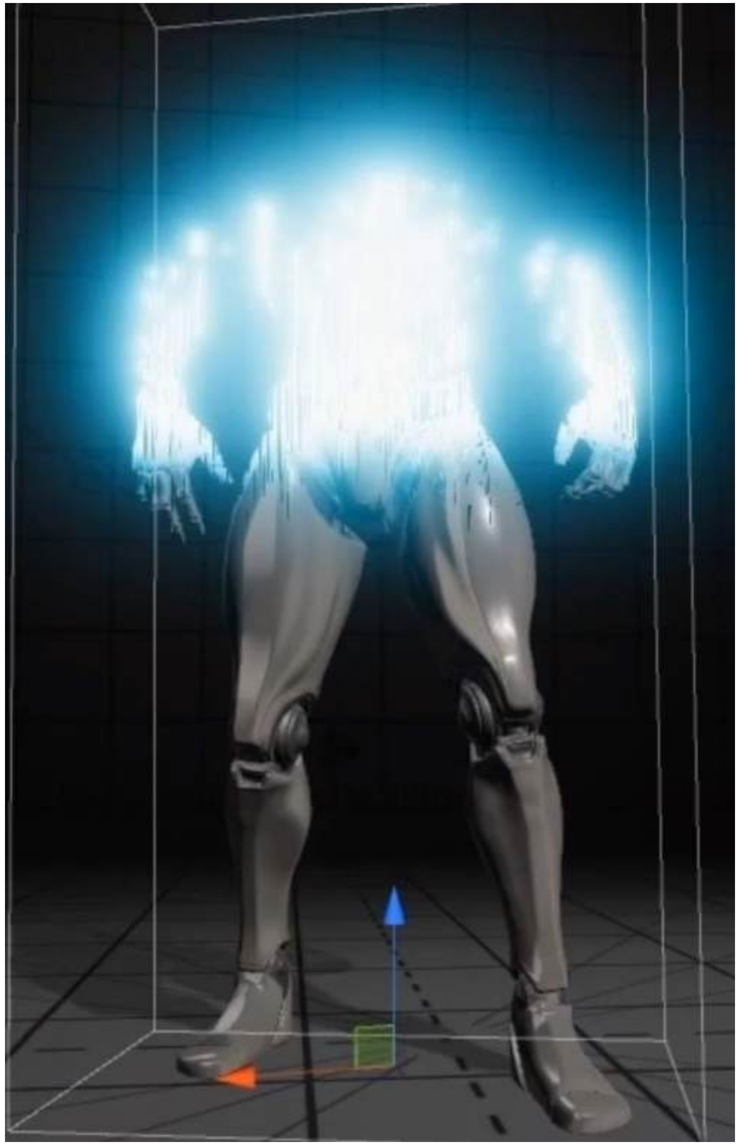


Dessolve Effect - Edge Emission

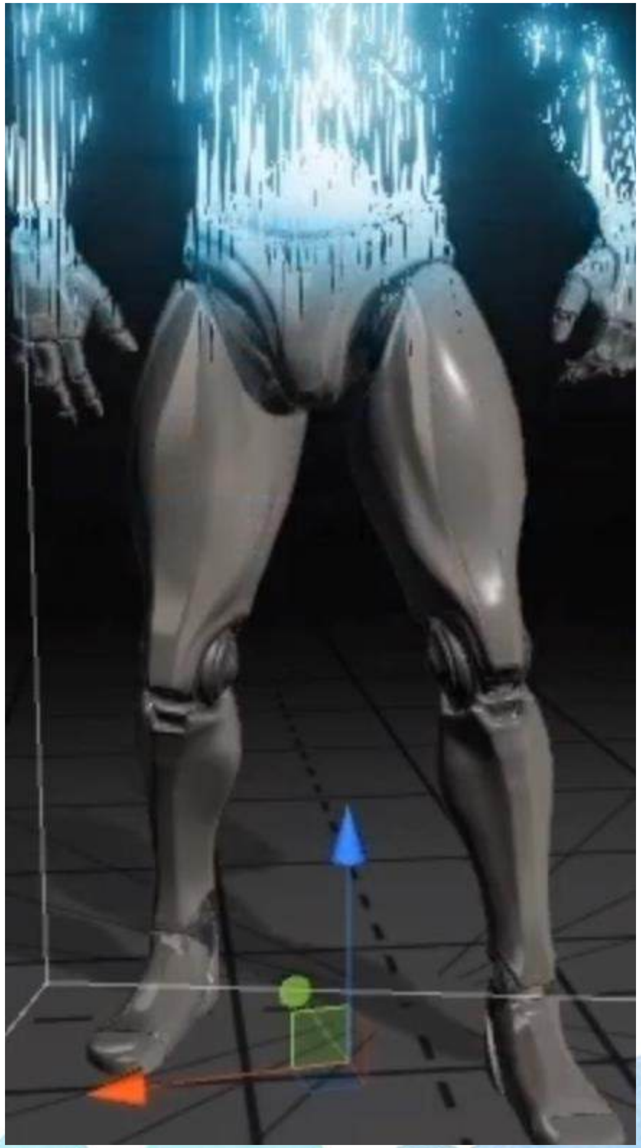
Adjust color of edge area



Add more lighting area

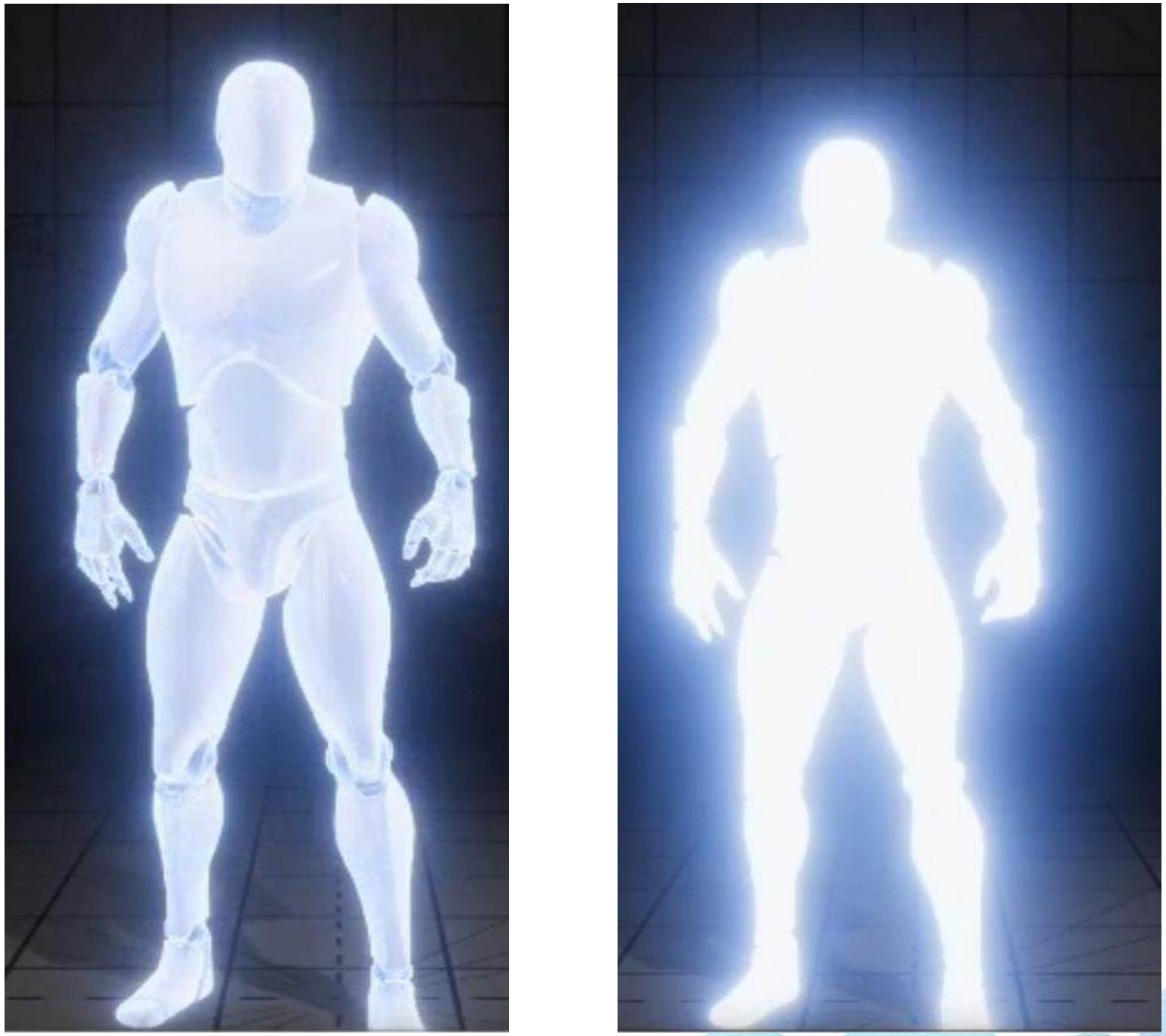


Adjust the color



Dessolve Effect - Rim Lighting

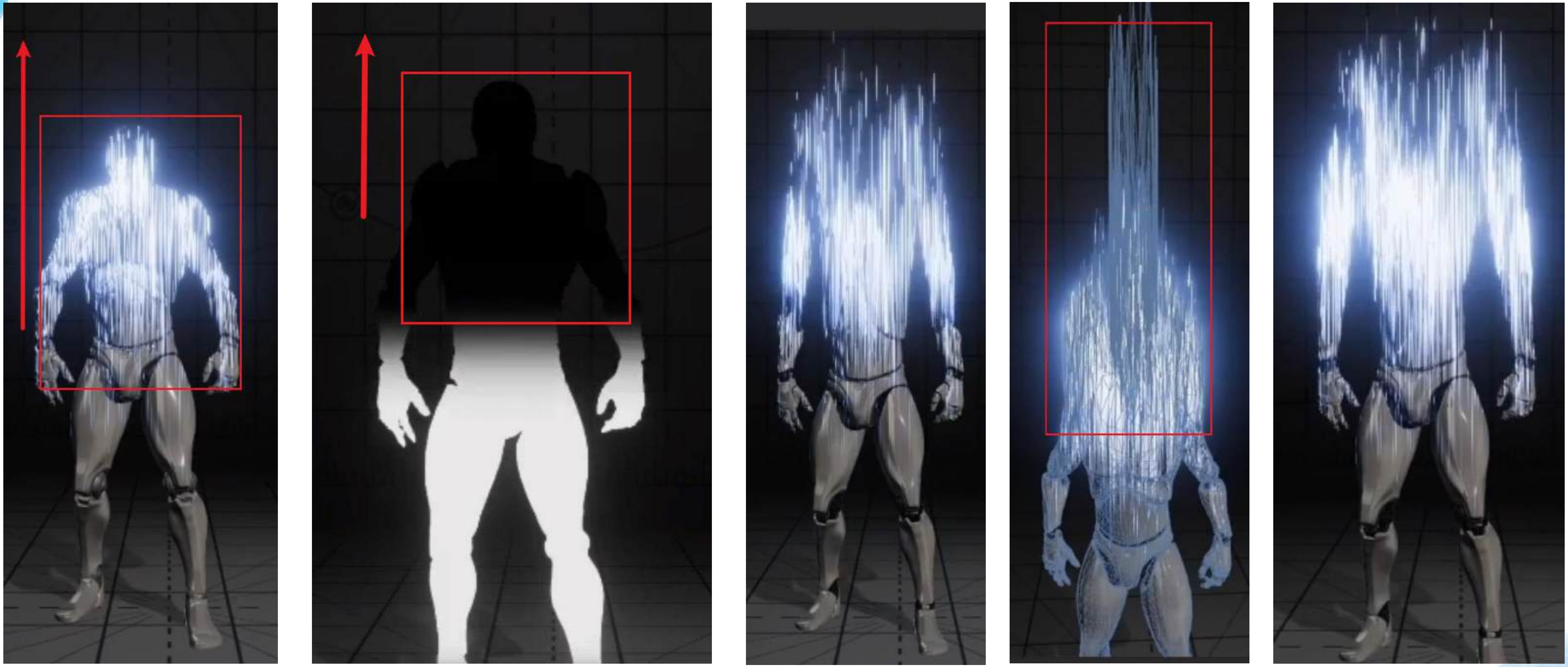
Original



Adjust the rim color of the world normal map before ressolve

Vertex Stretching

Multiply the world position of mask by some arbitrary vector and noise again



Reference

Concept

Hologram: [Angry Cheese Bread from bilibili]

Teleport: [Shack Man from YouTube;]
[DucVu FX from YouTube]

Code

Hologram: [andydbc from GitHub]

Teleport: [The Book of Shaders;]
[Noisy nodes from JimmyCushnie GitHub]

Unity Shader

[Unity Manual from Unity Documentation]

Scene

[SciFi Neon City 1.2.1 from Unity AssetStore]

Character

[Robot Kyle from Unity AssetStore]

Texture

Substance Designer and Stable Diffusion

Animation

Unity Timeline Director, Animation and Recorder

Technical Support

[Keer An]

Soundtrack

V from Cyberpunk 2077 Soundtrack

Unity

2022.3.46f1 <DX11>

Cloud image from Pinterest