Computer Graphics Individual Assignment #1

—

Constantine Pallas
Student Number 100822644

Outline

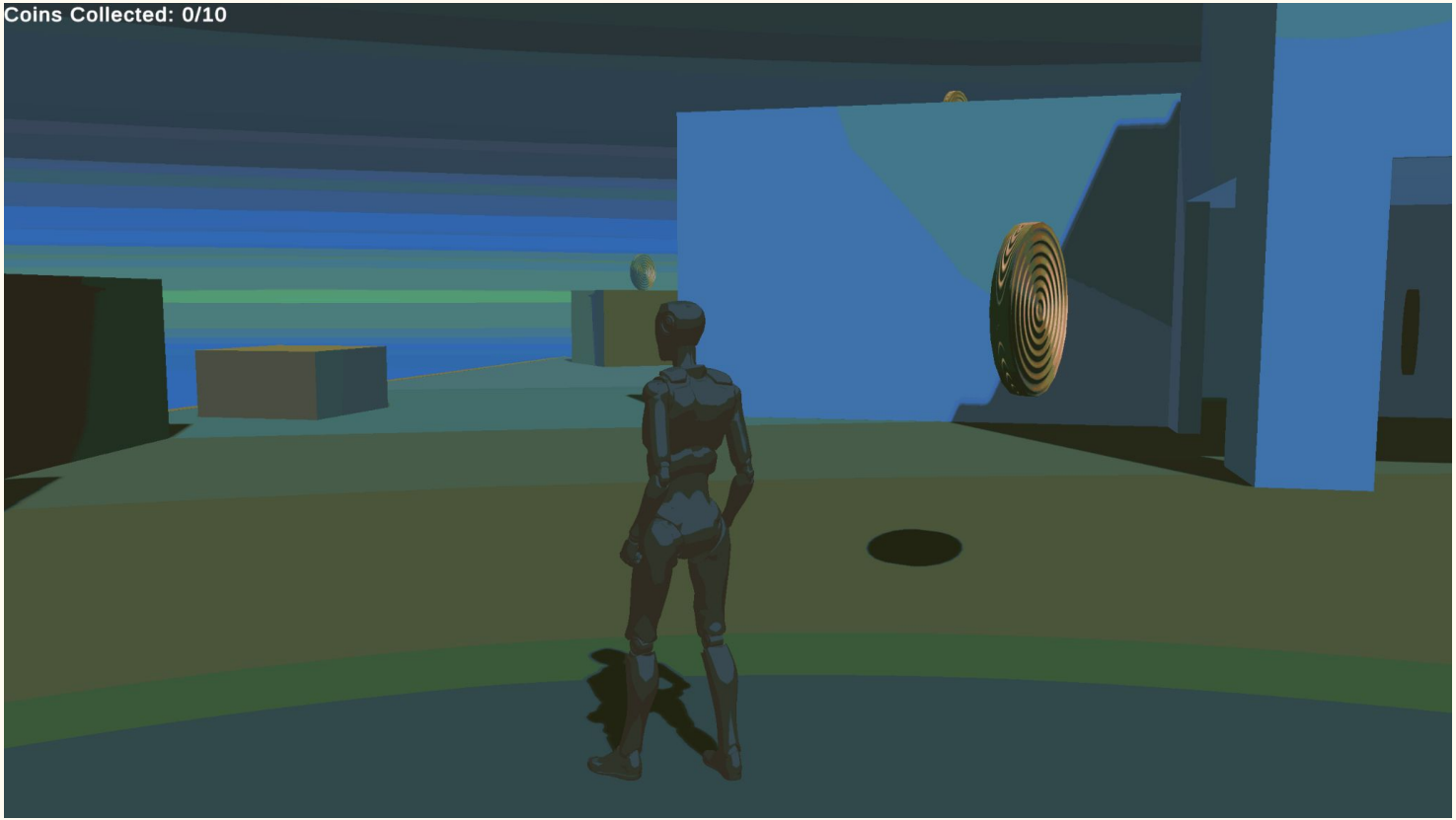
Part 1 > **Game breakdown**

Part 2 > **Illumination**

Part 3 > **Colour Grading**

Part 4 > **Additional Shaders**

Game Breakdown



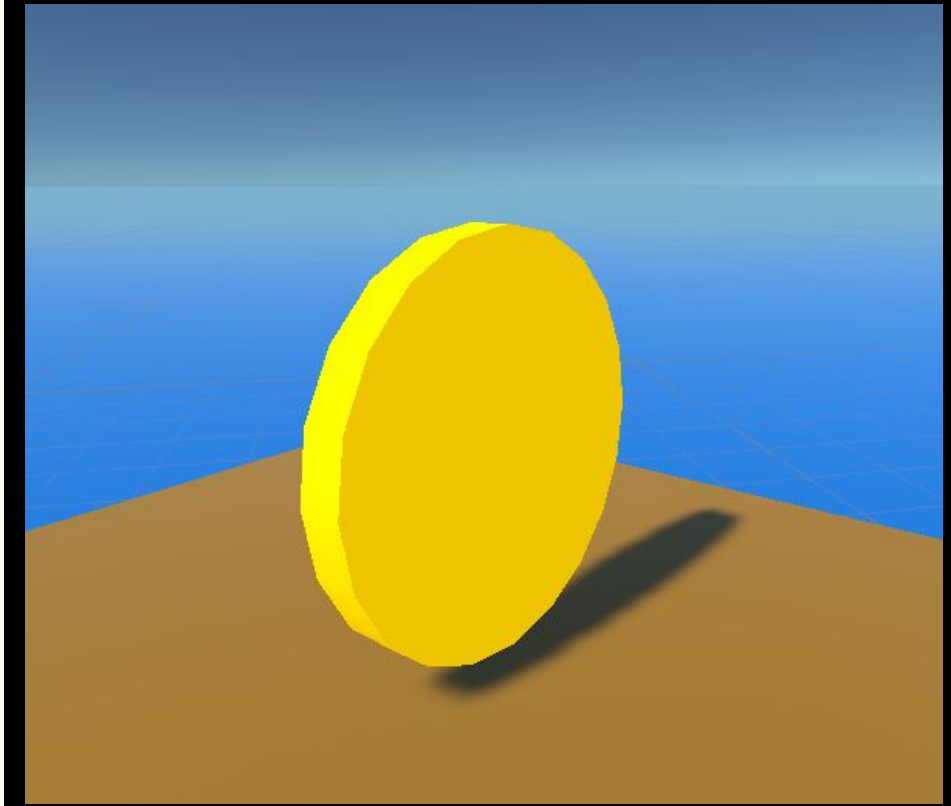
Game based on Unity 3rd person Template - Code is added for additional features
To win, collect all the coins.

Illumination

Features:

- Diffuse Shading can scatter light
- Ambient Shading
- Pretty much what you'd expect

Diffuse + Ambient



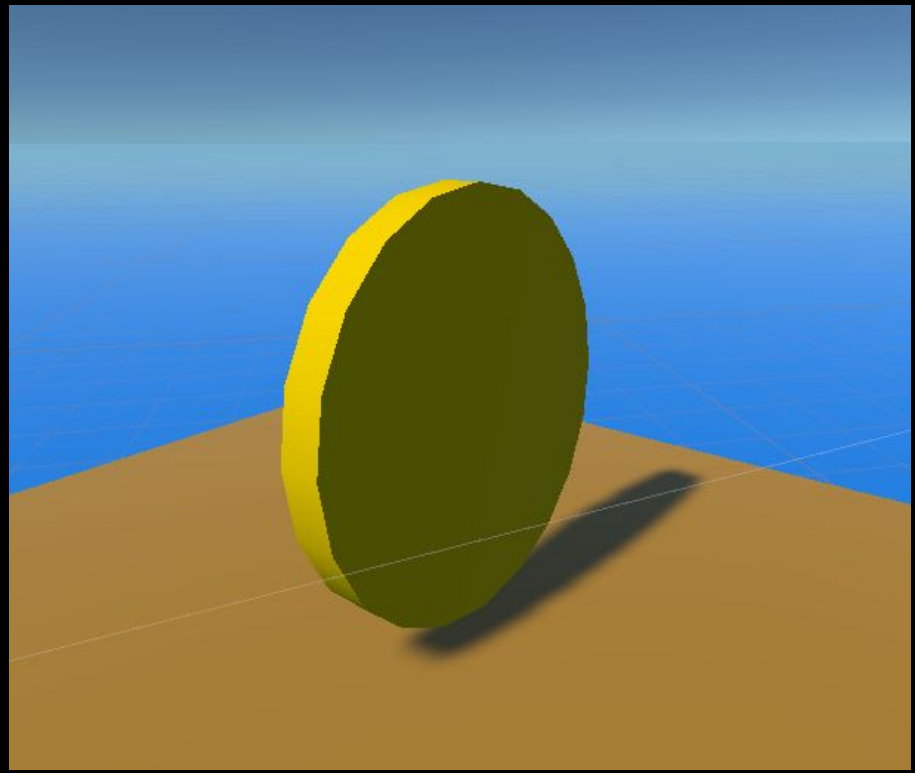
Implementation

```
Shader "Custom/AmbientDiffuse"
{
    Properties
    {
        _Color("Color", Color) = (1,1,1,1)
        _Color2("Color2", Color) = (1,1,1,1)
    }
        SubShader
    {
        Tags { "LightMode" = "ForwardBase" }

        //fixed4 _Color;

        Pass{
            CGPROGRAM
             #pragma vertex vert
             #pragma fragment frag

            uniform float4 _Color;
            uniform float4 _LightColor0;
        struct vertexInput {
            float4 vertex: POSITION;
            float3 normal: NORMAL;
        };
        struct vertexOutput {
            float4 pos: SV_POSITION;
            float4 col: COLOR;
         };

        vertexOutput vert(vertexInput v) {
            vertexOutput o;
            float3 normalDirection = normalize(mul(float4(v.normal, 0.0), unity_WorldToObject).xyz);
            float3 lightDirection;
            float atten = 1.0;

            lightDirection = normalize(_WorldSpaceLightPos0.xyz);
            //float3 diffuseReflection = atten * _LightColor0.xyz * _Color.rgb * max(0.0, dot(normalDirection, lightDirection));
            //o.col = float4(diffuseReflection, 1.0);

            float3 diffuseReflection = atten * _LightColor0.xyz * max(0.0, dot(normalDirection, lightDirection));
            float3 lightFinal = diffuseReflection + UNITY_LIGHTMODEL_AMBIENT.xyz;
            o.col = float4(lightFinal * _Color.rgb, 1.0);
            o.pos = UnityObjectToClipPos(v.vertex);
            return o;
        }



        float4 frag(vertexOutput i) : COLOR
        {
            return i.col;
        }
            ENDCG
        }

    }
        FallBack "Diffuse"
}
```

Specular

Features:

- Great for shiny surfaces (such as this coin)
- Includes specular highlights (not seen here, facing away from the light)



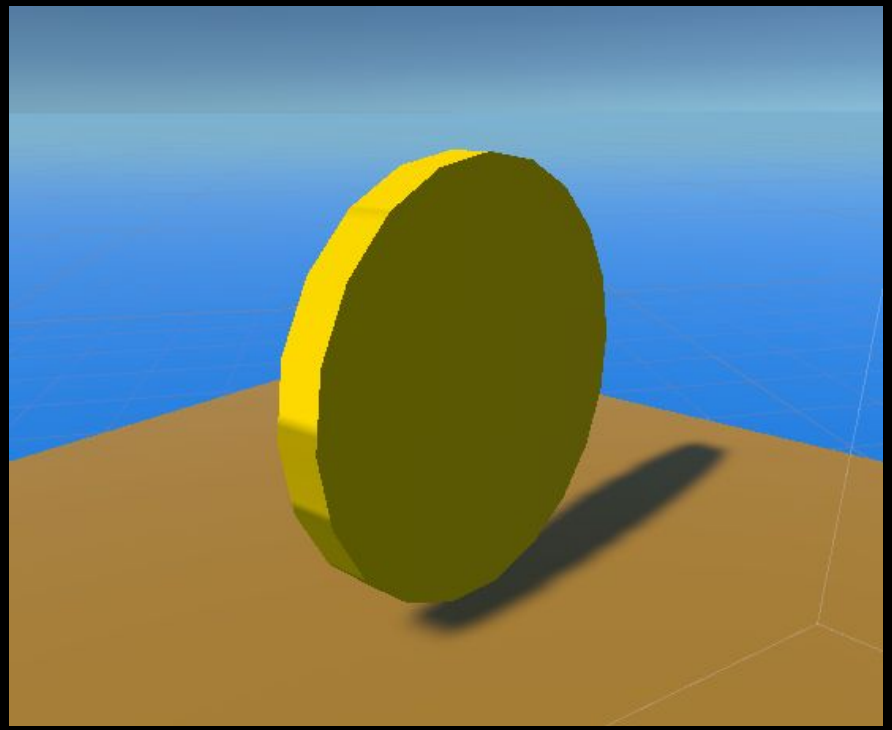
Implementation

```
Shader "Custom/Specular"
{
    Properties
    {
        _Color("Color", Color) = (1,1,1,1)
        _SpecColor("Color", Color) = (1.0,1.0,1.0,1.0)
        _Shininess("Shininess", Float) = 10
    }
        SubShader
    {
        Pass
        {
            Tags{"LightMode" = "ForwardBase"}

            CGPROGRAM
        #pragma vertex vert
        #pragma fragment frag
        uniform float4 Color;
        uniform float4 SpecColor;
        uniform float Shininess;

        uniform float4 LightColor0;

        struct vertexInput
        {
            float4 vertex : POSITION;
            float3 normal : NORMAL;
        };

        struct vertexOutput
        {
            float4 pos : SV_POSITION;
            float4 posWorld : TEXCOORD0;
            float4 normalDir : TEXCOORD1;
        };

        vertexOutput vert(vertexInput v)
        {
            vertexOutput o;
            o.posWorld = mul(unity_ObjectToWorld, v.vertex);
            o.normalDir = normalize(mul(float4(v.normal, 0.0), unity_WorldToObject));
            o.pos = UnityObjectToClipPos(v.vertex);
            return o;
        }

        float4 frag(vertexOutput i) : COLOR
        {
            float3 normalDirection = i.normalDir;
            float atten = 0;

            float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);
            float3 diffuseReflection = atten * LightColor0.xyz * max(0.0, dot(normalDirection, lightDirection));
            float3 lightReflectionDirection = reflect(-lightDirection, normalDirection);
            float3 viewDirection = normalize(float3(float4(_WorldSpaceCameraPos.xyz, 1.0) - i.posWorld.xyz));
            float3 lightSeeDirecton = max(0.0, dot(lightReflectionDirection, viewDirection));
            float3 shininessPower = pow(lightSeeDirecton, Shininess);
            float3 specularReflection = atten * SpecColor.rgb * shininessPower;
            float3 lightFinal = diffuseReflection + specularReflection + UNITY_LIGHTMODEL_AMBIENT;
            return float4(lightFinal * Color.rgb, 1.0);
        }

            ENDCG
        }
    }
        //FallBack "Diffuse"
}
```

Features:

- Uses a ramp texture to shade in defined sections instead of smoothly
- ‘Cartoonish’ look
- Matches quite well with my LUT profiles

Toon Ramp

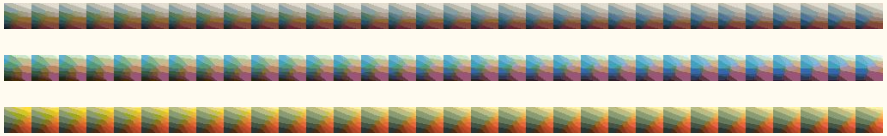


Implementation

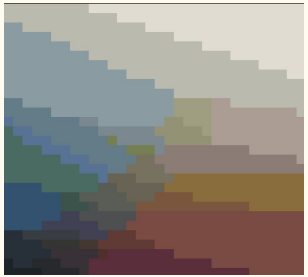
```
Shader "Custom/ToonRamp"
{
    Properties
    {
        _Color("Color", Color) = (1,1,1,1)
        _RampTex("Ramp Texture", 2D) = "white"{}
    }
        SubShader
    {

        CGPROGRAM
        // Physically based Standard lighting model, and enable shadows on all light types
        #pragma surface surf ToonRamp

        float4 _Color;
        sampler2D _RampTex;

        float4 LightingToonRamp(SurfaceOutput s, fixed3 lightDir, fixed atten)
        {
            float3 diff = dot(s.Normal, lightDir);
            float h = diff * 0.5 + 0.5;
            float2 rh = h;
            float3 ramp = tex2D(_RampTex, rh).rgb;

            float4 c;
            c.rgb = s.Albedo * _LightColor0.rgb * (ramp);
            c.a = s.Alpha;
            return c;
        }

        struct Input
        {
            float2 uv_MainTex;
        };

        void surf(Input IN, inout SurfaceOutput o)
        {
            o.Albedo = _Color.rgb;
        }
        ENDCG
    }
        FallBack "Diffuse"
}
```

Colour Grading

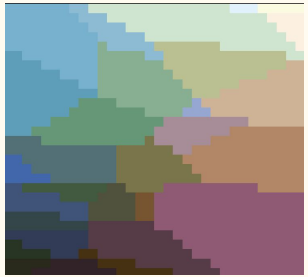
Look Up Tables (LUTs)



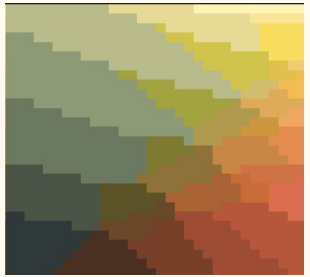
- Colour palettes from Lospec
- Reducing standard LUT to palette using Aseprite



Custom
‘waldgeist’



Cool
‘franzton’



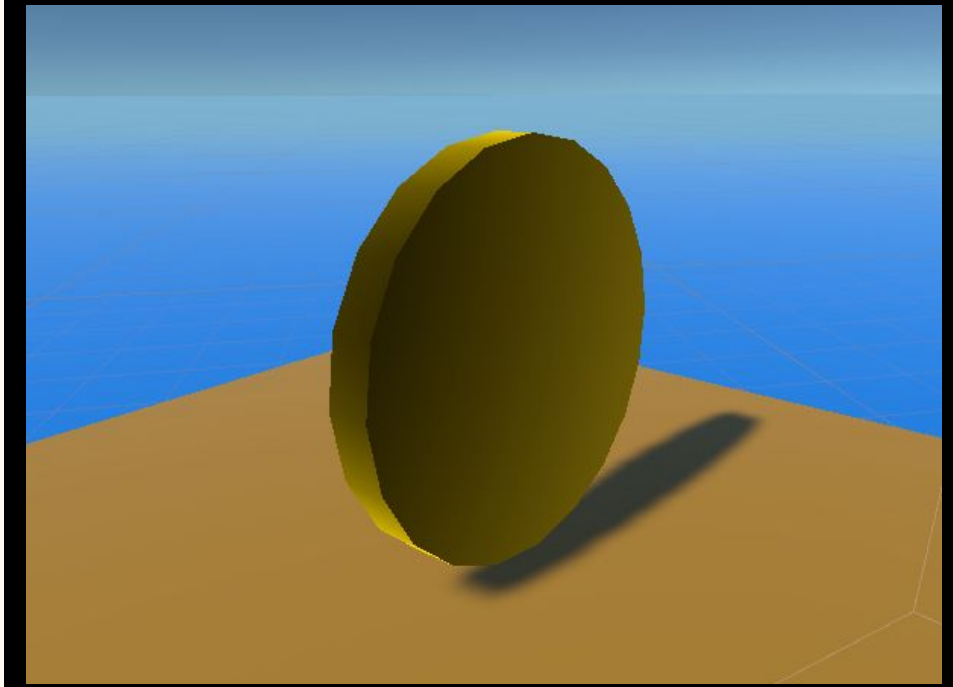
Warm
‘Autumn Harvest’

Additional Shaders

Features:

- Well defined outlines
- With transparency, can create a ‘hologram’ effect
- A cheap way to do outlines

Rim Lighting



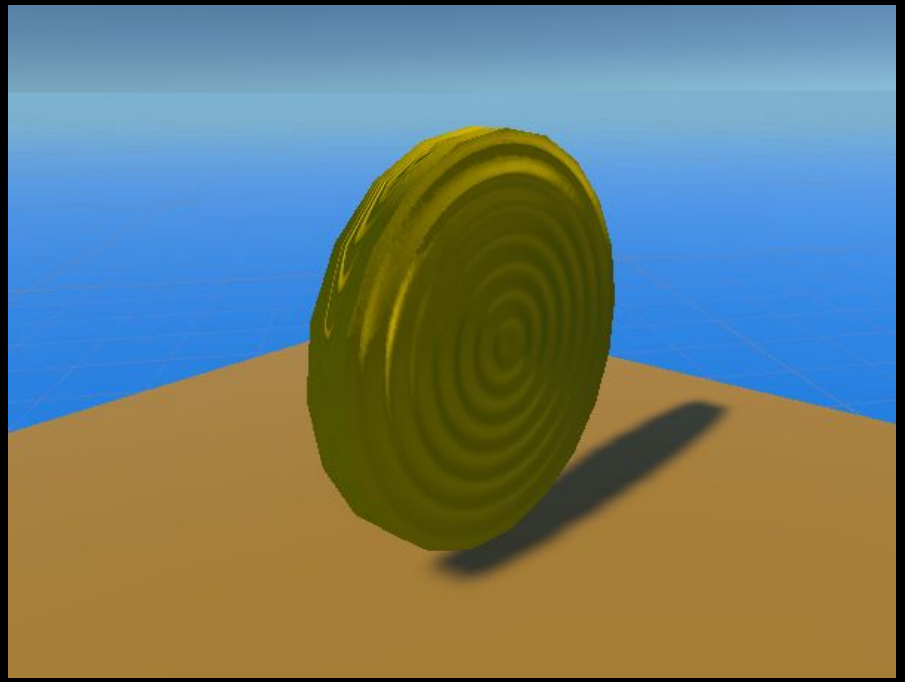
Implementation

```
Shader "Custom/RimLighting"
{
    Properties
    {
        _RimColor("Rim Color", Color) = (0.0,1.0,1.0,0.0)
        _RimPower("Rim Power", Range(0.5,8.0)) = 3.0
    }
        SubShader
    {
        CGPROGRAM
        #pragma surface surf Lambert
        struct Input
        {
            float3 viewDir;
        };

    float4 _RimColor;
    float _RimPower;
    void surf(Input IN, inout SurfaceOutput o)
    {
        //half rim = dot(normalize(IN.viewDir), o.Normal);
        half rim = 1.0 - saturate(dot(normalize(IN.viewDir),o.Normal));
        //o.Emisssion = _RimColor.rgb * rim;
        o.Emission = _RimColor.rgb * pow(rim, _RimPower);
    }
        ENDCG
    }
        FallBack "Diffuse"
}
```

Features:

- Adds extra detail to the mesh with very little performance cost
- Specifically, it adds depth information.
- Looks the most like an actual coin.

Normal Mapping



Implementation

```
Shader "Custom/NormalMap"
{
    Properties
    {
        _Diffuse("Diffuse Texture", 2D) = "white" {}
        _Bump("Bump Texture", 2D) = "bump" {}
        _Slider("Bump Amount", Range(0,10)) = 1
    }
        SubShader
        {
            CGPROGRAM
            #pragma surface surf Lambert
            sampler2D _Diffuse;
            sampler2D _Bump;
            half _Slider;

            struct Input
            {
                float2 uv_Diffuse;
                float2 uv_Bump;
            };

            void surf(Input IN, inout SurfaceOutput o)
            {
                o.Albedo = tex2D(_Diffuse, IN.uv_Diffuse).rgb;
                o.Normal = UnpackNormal(tex2D(_Bump, IN.uv_Bump));
                o.Normal *= float3(_Slider, _Slider, 1);
            }
        ENDCG
        }
            FallBack "Diffuse"
}
```

Thanks for Watching