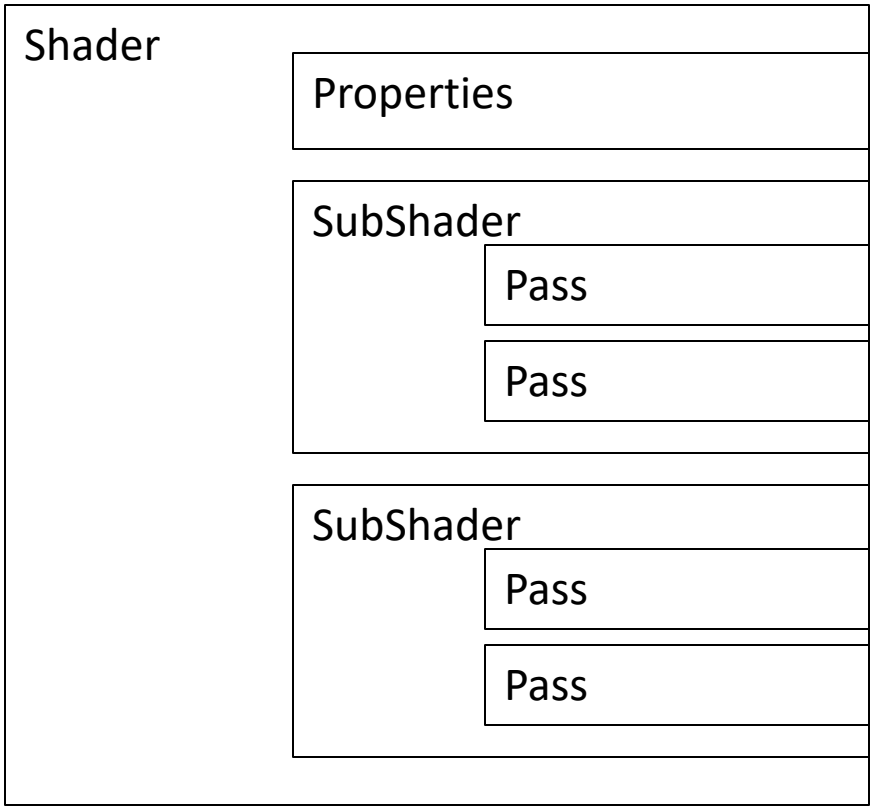
Računarska grafika 2

13M111RG2

5 Unity – programi za senčenje

Shader – struktura programa (podsetnik)



Shader - sintaksa

```
Shader "name" {
   [Properties] Subshaders [Fallback] [CustomEditor] }
```

- https://docs.unity3d.com/Manual/SL-Shader.html

- Properties - podaci (teksture, boja, ...) vidljivi u inspektoru mat.
  - mogu podešavati i fiks. delove rend. pipel. – na primer Blend
  - vrednosti se čuvaju (serijalizuju) od strane Unity Editora
- Subshaders + Fallback
  - lista shader programa
  - najmanje 1
  - primenjuje se prvi koji može da radi na datom hardveru
- CustomEditor
  - naziv klase koju treba koristiti u Unity Editoru za prikazivanje ovog shader-a

Shader – sintaksa
Properties

```
Properties { Property [Property ...] }


name ("display name", Range (min, max)) = number
name ("display name", Float) = number
name ("display name", Int) = number

name ("display name", Color) = (number,number,number,number)
name ("display name", Vector) = (number,number,number,number)

name ("display name", 2D) = "defaulttexture" {}
name ("display name", Cube) = "defaulttexture" {}
name ("display name", 3D) = "defaulttexture" {}
```

```
"" "white" "black"
"gray" "bump" "red"
```

Shader – sintaksa
SubShader

```
Subshader { [Tags] [CommonState] Passdef [Passdef ...] }
```

- Definiše:
  - oznake (tags)
  - zajedničko stanje za sve prolaze (sintaksa ista kao za pojedinačne prolaze)
  - prolaze

Shader – sintaksa
Pass

```
Pass { [Name and Tags] [State] }
```

- Stanja:
  - Cull (Back | Front | Off)
  - ZTest (Less | Greater | LEqual | GEqual | Equal | NotEqual | Always)
  - ZWrite (On | Off)
  - Offset *OffsetFactor*, *OffsetUnits*
  - Blend *srcBlendMode dstBlendMode*, *aSr*cBlendMode *aDstBlendMode*
  - ColorMask (RGB | A | 0 | any combination of R, G, B, A)
- Vrste prolaza
  - "regularni"
  - UsePass – koristi postojeći prolaz nekog drugog prog. za senčenje
  - GrabPass – tekući sadržaj bafera slike čuva u teksturu dostupnu narednim prolazima

Shader – primer sintakse (podsetnik)

```
Shader "RG2/SimpleShader"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Tags { "RenderType"="Opaque" }
        LOD 100
        Pass
        {
            CGPROGRAM
            #pragma ...
            #include ...
                ...
            }
            ENDCG
        }
    }
}
```

Shader – jednostavan program

```
Shader "RG2/SimpleShader"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }

    SubShader
    {
        Tags { "RenderType"="Opaque" }

        LOD 100

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"
```

Shader – jednostavan program

```
struct appdata
{
   float4 vertex : POSITION;
   float2 uv : TEXCOORD0;
};


struct v2f
{
   float2 uv : TEXCOORD0;
   float4 vertex : SV_POSITION;
};


sampler2D _MainTex;
```

```
Properties
{
    _MainTex ("Texture", 2D) = "white" {}
}
```

Shader – jednostavan program

```
        v2f vert (appdata v)
        {
            v2f o;
            o.vertex = UnityObjectToClipPos(v.vertex);
            o.uv = v.uv;
            return o;
        }

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

Semantika ulazno/izlaznih podataka

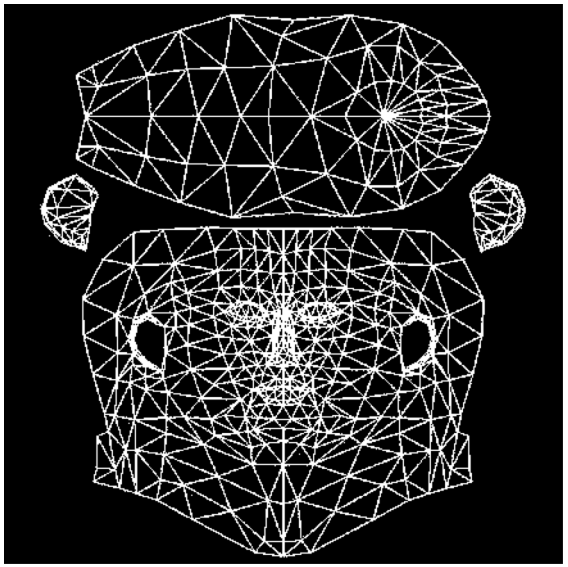
Atributi temena

- `POSITION :` pozicija (najčešće `float3` ili `float4`).
- `NORMAL:` normala (najčešće `float3`).
- `TEXCOORD0..3:` UV koordinate (najčešće `float2, float3` ili `float4`).
- `TANGENT:` tangenta (najčešće `float4`).
- `COLOR:` boja (najčešće `float4`).

```
struct appdata
{
    float4 vertex : POSITION;
    float2 uv : TEXCOORD0;
};
```

`float4 uv : TEXCOORD0;` → **`(uv.xy, 0, 1)`**

Primer UV mape



UnityCG.cginc

- Biblioteka raznih uslužnih funkcija
- Transformacije geometrije ili u prostoru tekstura
- Projekcije, osvetljenje
- Sakriva uniformne podatke koje Unity prosleđuje do shader-a

```
struct appdata_base {
    float4 vertex : POSITION;
    float3 normal : NORMAL;
    float4 texcoord : TEXCOORD0;
    UNITY_INSTANCE_ID
};
```

```
struct appdata_tan {
    float4 vertex : POSITION;
    float4 tangent : TANGENT;
    float3 normal : NORMAL;
    float4 texcoord : TEXCOORD0;
    UNITY_INSTANCE_ID
};

struct appdata_full {
...
};
```

UnityCG.cginc

```
inline float4 UnityObjectToClipPos( in float3 pos ) {
#ifdef UNITY_USE_PREMULTIPLIED_MATRICES
    return mul(UNITY_MATRIX_MVP, float4(pos, 1.0));
#else
    // More efficient than computing M*VP matrix product
    return mul(UNITY_MATRIX_VP, mul(unity_ObjectToWorld, float4(pos, 1.0)));
#endif
}

inline float4 UnityWorldToClipPos( in float3 pos )
{   return mul(UNITY_MATRIX_VP, float4(pos, 1.0));  }

inline float4 UnityViewToClipPos( in float3 pos )
{   return mul(UNITY_MATRIX_P, float4(pos, 1.0));  }

inline float3 UnityObjectToWorldDir( in float3 dir )
{   return normalize(mul((float3x3)unity_ObjectToWorld, dir));  }

inline float3 UnityWorldToObjectDir( in float3 dir )
{   return normalize(mul((float3x3)unity_WorldToObject, dir));  }

inline float3 UnityObjectToViewPos( in float3 pos ) ...
inline float3 UnityWorldToViewPos( in float3 pos ) ...
inline float3 UnityObjectToWorldNormal( in float3 norm ) ...
```

Još neki pomoćni mehanizmi

```
CGPROGRAM
#pragma vertex vert
#pragma fragment frag

// make fog work
#pragma multi_compile_fog

#include "UnityCG.cginc"

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

Još neki pomoćni mehanizmi

```
sampler2D _MainTex;
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);
 return o;
}

fixed4 frag (v2f i) : SV_Target
{
 fixed4 col = tex2D(_MainTex, i.uv);
 // apply fog
 UNITY_APPLY_FOG(i.fogCoord, col);
 return col;
}
ENDCG
```

```
#define TRANSFORM_TEX(tex,name) (tex.xy * name##_ST.xy + name##_ST.zw)
```

Primer jednostavnog Phong-ovog shader-a

```
Shader "RG2/02_SpecularShader"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Tags { "RenderType"="Opaque" "LightMode" = "ForwardBase" }
        LOD 100
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
            #include "Lighting.cginc"
```

Primer jednostavnog Phong-ovog shader-a

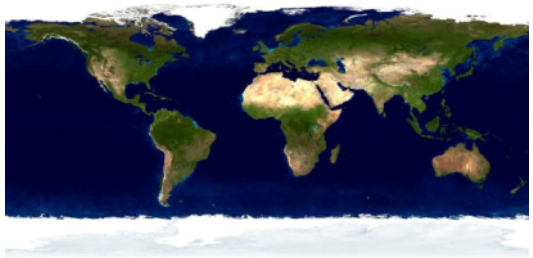
```
struct appdata
{
    float4 vertex : POSITION;
    float3 normal : NORMAL;
    float2 uv : TEXCOORD0;
};
struct v2f
{
    float2 uv : TEXCOORD0;
    float3 normal : NORMAL;
    float4 vertex : SV_POSITION;
    float3 lightDir : TEXCOORD1;
    float3 viewDir : TEXCOORD2;
};

sampler2D _MainTex;
```

Primer jednostavnog Phong-ovog shader-a

```
    v2f vert (appdata v) {
        v2f o;
        o.vertex = UnityObjectToClipPos(v.vertex);
        o.normal = UnityObjectToWorldNormal(v.normal);
        float3 worldPos = mul(unity_ObjectToWorld, v.vertex).xyz;
        o.lightDir = normalize(UnityWorldSpaceLightDir(worldPos));
        o.viewDir = normalize(_WorldSpaceCameraPos.xyz - worldPos.xyz);
        o.uv = v.uv;
        return o;
    }

    fixed4 frag (v2f i) : SV_Target {
        fixed4 col = tex2D(_MainTex, i.uv);
        float3 normal = normalize(i.normal);
        float3 lightDir = normalize(i.lightDir);
        float lambert = max(0,dot(normal, lightDir));
        float3 reflection = reflect(-lightDir, normal);
        float specular = pow(max(0,dot(normalize(i.viewDir), reflection)), 25);
        return fixed4(col.rgb*lambert + fixed3(specular, specular, specular), 1.0);
    }
    ENDCG
}
```

Upotreba više tekstura istovremeno

(scena 03_MultiTexture)



```
Properties
{
    _DayTex ("Day", 2D) = "white" {}
    _NightTex ("Night", 2D) = "white" {}
    _GlossTex ("Gloss", 2D) = "white" {}
}
```

Upotreba više tekstura istovremeno

```
fixed4 frag (v2f i) : SV_Target
{
    fixed4 day = tex2D(_DayTex, i.uv);
    fixed4 night = tex2D(_NightTex, i.uv);
    fixed4 gloss = tex2D(_GlossTex, i.uv);

    float3 normal = normalize(i.normal);
    float3 lightDir = normalize(i.lightDir);

    float lambert = max(0, dot(normal, lightDir));
    float3 reflection = reflect(-lightDir, normal);
    float specular = pow(max(0,dot(normalize(i.viewDir), reflection)), 25) * gloss.r;
    return fixed4(
                        lerp(night.rgb, day.rgb, lambert)
                        +
                        fixed3(specular, specular, specular),
                        1.0
                );
}
```

Alpha blending

```
Properties
{
    DayColor ("Day", Color) = (0,0.7,1)
    SunsetColor ("Sunset", Color) = (1, 0.5, 0)
}
SubShader
{
    Tags { "RenderType"="Opaque" "LightMode"="ForwardBase" }

    Blend SrcAlpha OneMinusSrcAlpha
```

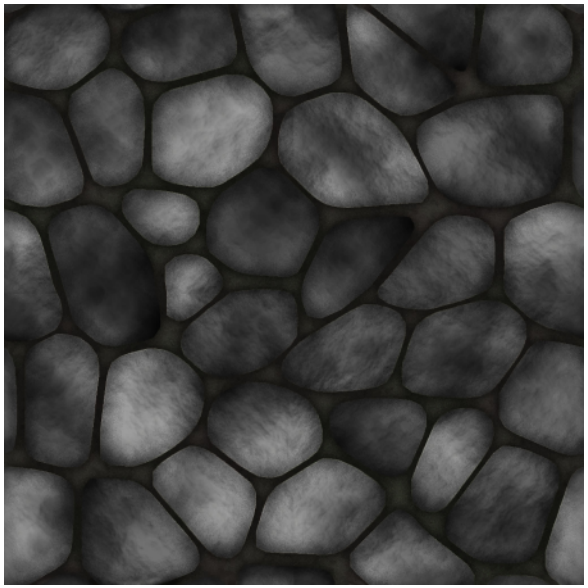
```
fixed4 frag (v2f i) : SV_Target
{
    float3 normal = normalize(i.normal);
    float3 lightDir = normalize(i.lightDir);

    float lambert = max(0, dot(normal, lightDir));
    float alpha = (1 - dot(i.viewDir, normal)) * lambert;
    return fixed4(lerp(SunsetColor.rgb, DayColor.rgb, lambert), alpha);
}
```

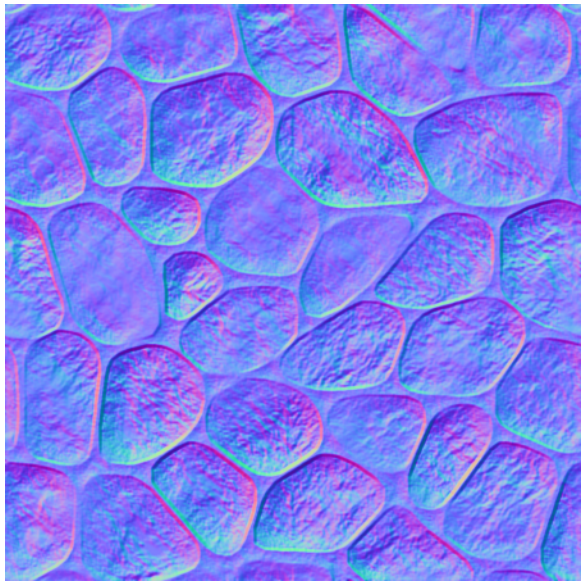
Normal mapping (Bump mapping)

(scena 04_NormalMap)

- Tehnika kojom se povećava vizuelni doživljaj detalja na površi 3D modela
- U realnosti, retko koja površ je glatka
- Neravnine se vizuelno manifestuju lokalnim promenama intenziteta reflektovanog svetla (ispupčenja i udubljenja)
- Modeliranje neravnina geometrijom (temenima) bi bilo neefikasno
- Vizuelni doživljaj: suština je u promeni intenziteta svetla
- Nije neophodno teme, dovoljna je lokalna informacija o normali
- Ideja: sadržaj teksture → informacija o normalama

Normal - mapa





- RGB komponente svakog teksela preslikavaju se u XYZ komponente normale
- Opseg RGB komponenti je [0, 1]
- Opseg XYZ komponenti je [-1,1]
- Najčešće se primenjuje transformacija: (RGB-0.5)·2
- Posledica:
  - "neutralna" normala je: (0,0,1).
  - odgovarajuća RGB je: (0.5, 0.5, 1)

Tangentni prostor

- Problem:
  - normale u normal mapi nisu zadate u k.s. objekta
  - lokalne su za svaki poligon – tzv. *tangentni prostor*
- Kako definisati tangentni prostor? Potrebna 3 vektora:
    - normala (atribut temena)
    - tangenta (atribut temena)
    - bi-tangenta (bi-normala) – vektorski proizvod normale i tangente
- U vertex shader-u se izračuna vektor ka svetlu u tangentnom prostoru svakog verteksa i automatski se interpolira

Shader program

```
Shader "RG2/03_BumpShader" {
    Properties {
        _MainTex ("Base (RGB)", 2D) = "white" {}
        _Bump ("Bump", 2D) = "bump" {}
    }
    SubShader {
        Tags { "RenderType"="Opaque" "LightMode"="ForwardBase" }
        LOD 200
        Pass {
            Cull Back

            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"

                sampler2D _MainTex;
                sampler2D _Bump;
                struct app2v {
                    float4 vertex : POSITION;
                    float3 normal : NORMAL;
                    float4 texcoord : TEXCOORD0;
                    float4 tangent : TANGENT;
                };
                struct v2f {
                    float4 pos : POSITION;
                    float2 uv : TEXCOORD0;
                    float3 lightDirection : TEXCOORD1;
                };
```

Shader program

```
    v2f vert (app2v v) {
      v2f o;
      TANGENT_SPACE_ROTATION;
      o.lightDirection = mul(rotation, ObjSpaceLightDir(v.vertex));
      o.pos = UnityObjectToClipPos(v.vertex);
      o.uv = v.texcoord;
      return o;
    }

    fixed4 frag(v2f i) : COLOR {
      float4 c = tex2D (_MainTex, i.uv);
      float3 n =  UnpackNormal(tex2D (_Bump, i.uv));
      float diff = saturate (dot (n, normalize(i.lightDirection)));
      c.rgb = diff * c.rgb;
      return c;
    }
    ENDCG
  }
}
```

Shader program

Iz UnityCG.cginc:

```
#define TANGENT_SPACE_ROTATION \
  float3 binormal = cross(normalize(v.normal), normalize(v.tangent.xyz)) * v.tangent.w; \
  float3x3 rotation = float3x3( v.tangent.xyz, binormal, v.normal )


float3 ObjSpaceLightDir( in float4 v ) {
    float3 objSpaceLightPos = mul(unity_WorldToObject, _WorldSpaceLightPos0).xyz;
    return objSpaceLightPos.xyz - v.xyz * _WorldSpaceLightPos0.w;
}


fixed3 UnpackNormal(fixed4 packednormal)
{
    return packednormal.xyz * 2 - 1;
}
```

Interaktivno bojenje 3D modela

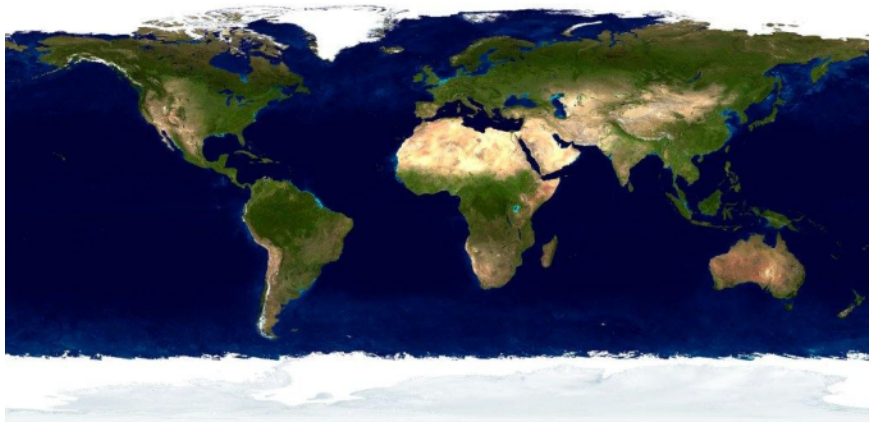
(scena 05_Paint)

- Raspored trouglova u prostoru teksture može biti (i često jeste) nepravilan
- Bojenje modela složeno – zahteva crtanje po teksturi u veoma nepravilnim oblastima
- Jednostavnije: "nanošenje boje" direktno na modelu nalik na bojenje skulpture

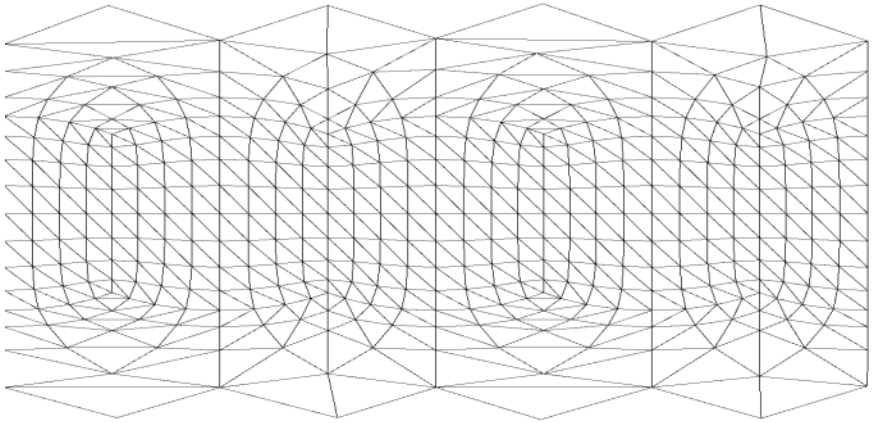
Interaktivno bojenje 3D modela

- Algoritam u dva prolaza
  1. Pravljenje obojene teksture
     i. Za selektovanu tačku na ekranu odrediti da li pripada objektu
     ii. Novu boju upisati u obojenu teksturu
  2. Obojenu teksturu kombinovati sa originalnom teksturom
- Kako odrediti da li tačka na ekranu pripada objektu?
  - Tačka sa ekrana i tačka objekta treba da budu u istom k.s., recimo NDC
  - Transformacija za tačku sa ekrana: k.s. ekrana -> NDC
  - Transformacija za tačku sa objekta: k.s. modela -> NDC
- Kako upisati boju u obojenu teksturu?
  - Izlaz fragment funkcije će biti boja za upis
  - Ta boja odgovara fragmentu koji bi se prikazao na ekranu
  - Koord. fragmenta je u NDC
  - Fragment međutim *neće* završiti na ekranu, već u obojenoj teksturi
  - Prema tome, NDC koord. ne treba da dolazi od tačke sa objekta, već *od uv tačke teksture koja odgovara tački sa objekta*
  - Potrebno je da uv koord. prebacimo u NDC k.s.

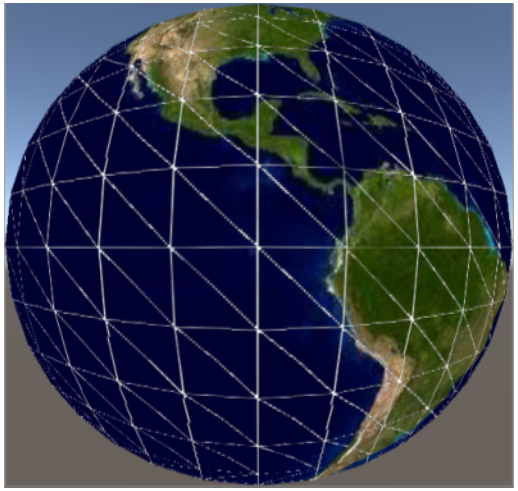
Interaktivno bojenje 3D modela



Tekstura

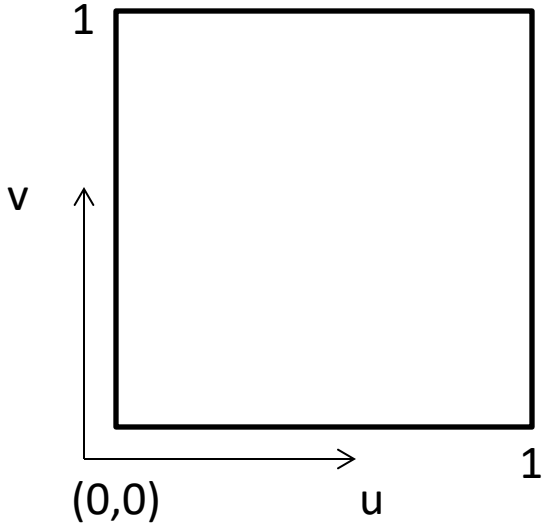


UV mapa

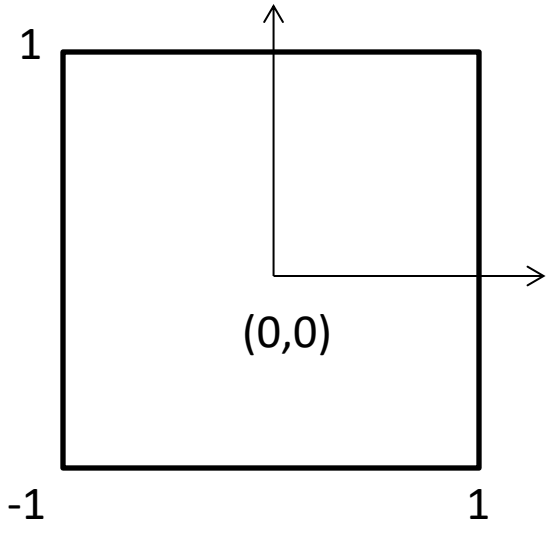


Interaktivno bojenje 3D modela

- Obratiti pažnju:
  - UV koordinate po pravilu zadate su u opsegu [0, 1]
  - Koordinate u NDC su u opsegu [-1, 1]
  - Preslikavanje: NDC = (UV – 0.5) · 2



Prostor teksture



NDC

Interaktivno bojenje 3D modela

```
Shader "RG2/05_PaintShader" {
    SubShader     {
        Tags { "RenderType"="Opaque" }
        LOD 100
        Blend SrcAlpha OneMinusSrcAlpha
        BlendOp Add, Max
        Cull Off
        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
            struct appdata
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };
            struct v2f
            {
                float4 vertex : SV_POSITION;
                float2 screenCoord : TEXCOORD0;
            };
```

Program za senčenje
koji crta po teksturi

Interaktivno bojenje 3D modela

```
        float4 center;
        float radius;

        v2f vert (appdata v)  {
            v2f o;
            float4 vertexInHCS = UnityObjectToClipPos(v.vertex);

            o.vertex = float4((v.uv-0.5)*2, 0, 1);
            o.screenCoord = vertexInHCS.xy / vertexInHCS.w;
            return o;
        }

        fixed4 frag (v2f i) : SV_Target  {
            float2 dist = i.screenCoord - center.xy;
            float dist2 = dot(dist, dist);
            float alpha = 1 - dist2 / (radius*radius);
            fixed4 col = fixed4(1, 1, 1, alpha);
            return col;
        }
        ENDCG
    }
  }
}
```

Parametri četkice – vrh je kružnog oblika

Preslikavanje UV→NDC

Interpolacija pozicije

Udaljenost piksela od centra četkice

Vrednosti boje automatski odsečene u opsegu [0,1]

Interaktivno bojenje 3D modela

Program za senčenje koji kombinuje osnovnu teksturu i teksturu po kojoj je korisnik crtao

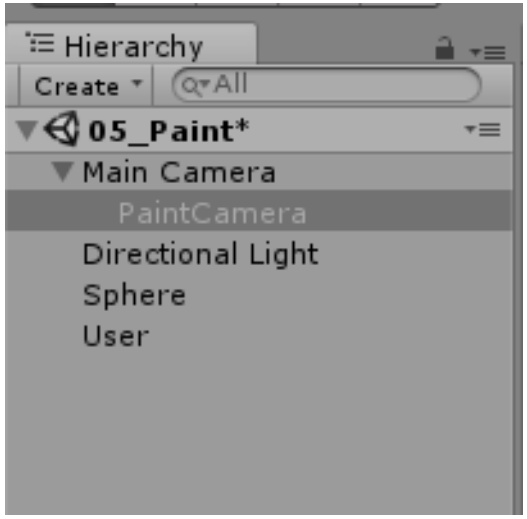
```
Shader "RG2/05_ViewPaint" {
    Properties     {
        _MainTex ("Main", 2D) = "white" {}
        _PaintTex ("Paint", 2D) = "white" {}
    }
    SubShader     {
        Tags { "RenderType"="Opaque" "LightMode"="ForwardBase" }
        LOD 100
        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
            struct appdata
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };
            struct v2f
            {
                float4 vertex : SV_POSITION;
                float2 uv : TEXCOORD0;
            };
```

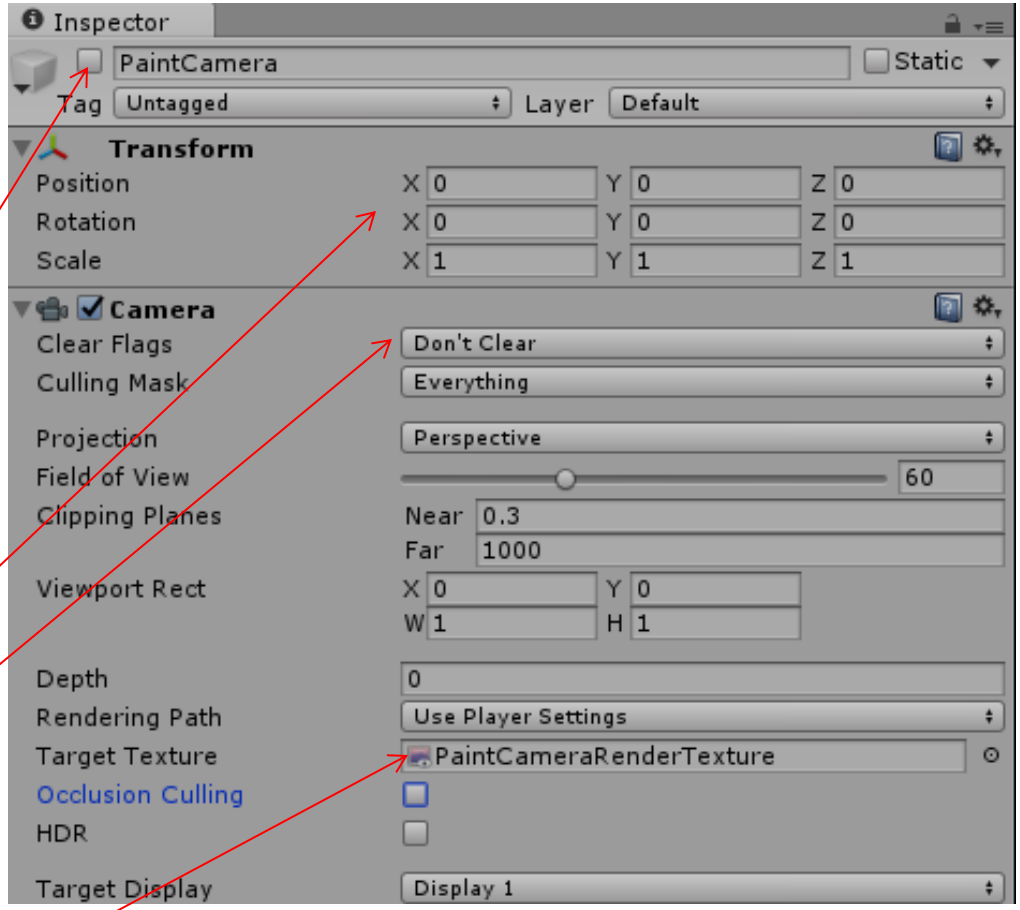
Interaktivno bojenje 3D modela

```
            sampler2D _MainTex;
            sampler2D _PaintTex;

            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                o.uv = v.uv;
                return o;
            }

            fixed4 frag (v2f i) : SV_Target
            {
                fixed4 main = tex2D(_MainTex, i.uv);
                fixed4 paint = tex2D(_PaintTex, i.uv);

                return fixed4(lerp(main.rgb, paint.rgb, paint.a), 1.0);
            }
            ENDCG
        }
    }
}
```

Koristi se prozirnost teksture sa crtežom da bi se kontrolisala modulacija osnovne teksture i teksture sa crtežom

Interaktivno bojenje 3D modela





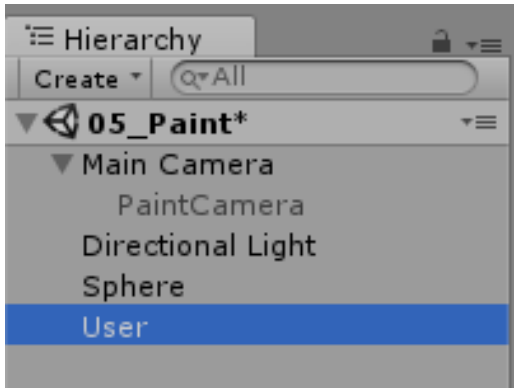
Posebna kamera koja će služiti za bojenje. NEAKTIVNA!

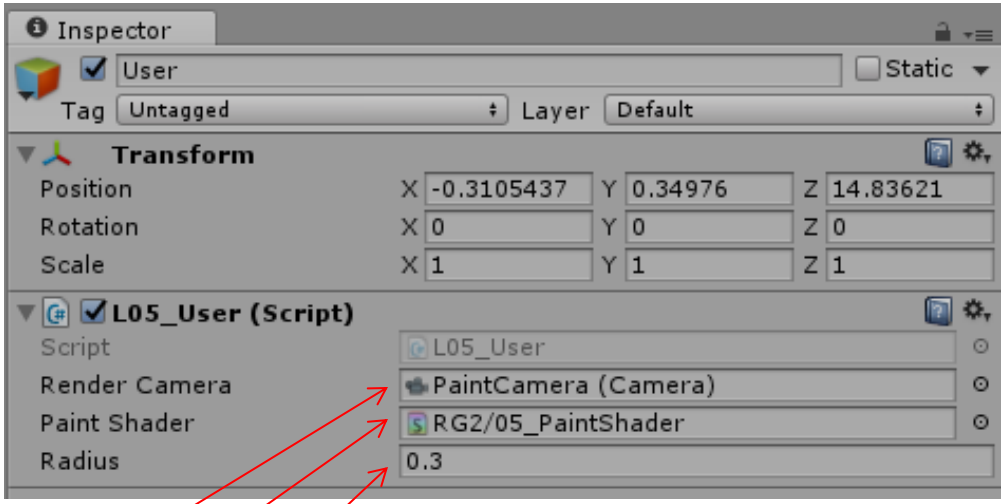
Ima istu poziciju i orijentaciju kao glavna kamera.

Ne treba da briše sliku po kojoj crta (*Don't clear*).

Crta po posebnoj "Render" teksturi.

Interaktivno bojenje 3D modela





Poseban objekat sa skriptom kojom korisnik kontroliše rad.

Eksplicitno se zadaje kamera koja "crta" po teksturi

Eksplicitno se zadaje *shader* koji će kamera koristiti

Debljina četkice

Interaktivno bojenje 3D modela

```
using UnityEngine;
using System.Collections;

public class L05_User : MonoBehaviour {

        [SerializeField]
        private Camera renderCamera;

        [SerializeField]
        private Shader paintShader;

        [SerializeField]
        private float Radius = 0.1f;

        private Vector4 center = new Vector4();
```

Kamera bi mogla da se dohvati programskim putem, ali je ovako jednostavnije.

Slično važi i za program za senčenje.

Interaktivno bojenje 3D modela

```
void Start () {
    if (renderCamera == null || paintShader == null )
        return;

    Graphics.SetRenderTarget(renderCamera.targetTexture);
    GL.Clear(false, true, Color.clear);



    renderCamera.aspect = 1;
    Camera.main.aspect = 1;



    renderCamera.SetReplacementShader(paintShader, null);
}
```

Brisanje sadržaja teksture, samo jednom, na početku rada.

Neophodno poštovati odnos širine i visine teksture.

Kamera će pri crtanju koristiti samo ovaj program za senčenje.

Interaktivno bojenje 3D modela

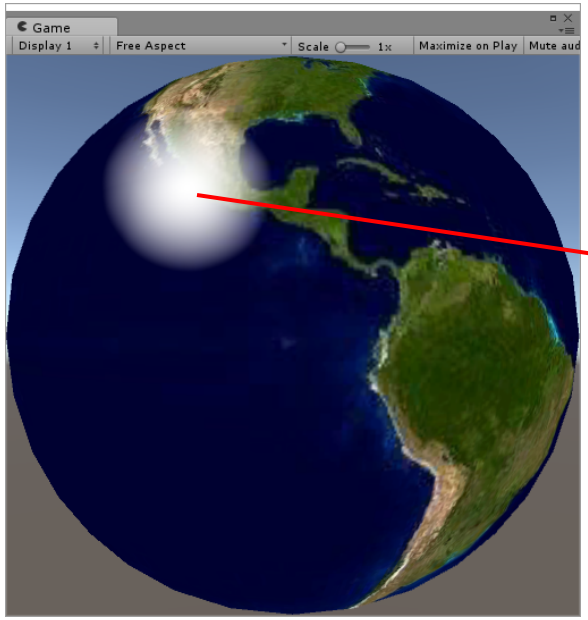
```
    void Update () {
        if (renderCamera == null || paintShader == null)
            return;

        if( Input.GetMouseButton(0) )  {
            Vector3 screenMousePos = Input.mousePosition;
            center.x = (screenMousePos.x / Screen.width - 0.5f) * 2;
            center.y = (screenMousePos.y / Screen.height - 0.5f) * 2;

            Shader.SetGlobalFloat("radius", Radius);
            Shader.SetGlobalVector("center", center);

            renderCamera.Render();
        }
    }
}
```

Postavljanje vrednosti uniformnih promenljivih

Eksplicitno pozivanje crtanja

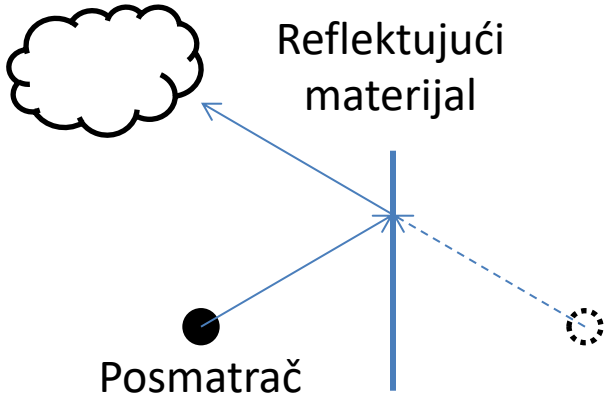
Interaktivno bojenje 3D modela

- Oprez!
- Ovo rešenje obuhvata sve trouglove čije XY koordinate u HPO
- To uključuje i trouglove koji su okrenuti naličijem prema posmatraču
- Posledica: boje se i "prednja" i "zadnja" strana objekta.



Reflektujući i proziran materijal

- Prave refleksije (ogledanja) su veoma zahtevne u tradicionalnom načinu sinteze slike (rasterizacija poligona)
- Generalni pristup senčenja ogledajućeg poligona:
  - kameru postaviti tako da odgovara poziciji i orijentaciji posmatrača u ogledalu
  - formirati sliku (= slika koju bi posmatrač video u ogledalu)
  - koristiti formiranu sliku kao teksturu za senčenje poligona
- Moralo bi da se primeni za svaku ogledajuću ravan
- Neefikasno

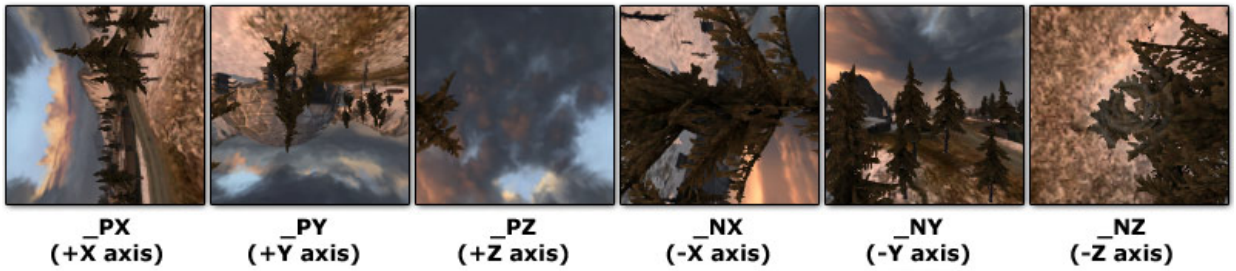


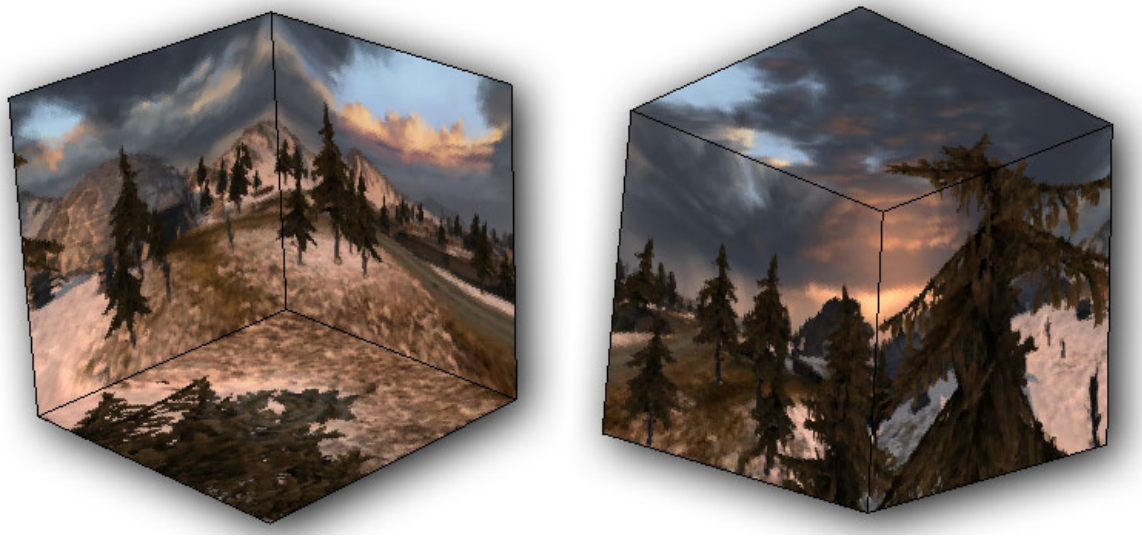
Reflektujući i proziran materijal

- Aproksimativno rešenje: *Cube map*
- Kocka koja "okružuje" posmatrani deo scene
- Presvučena slikama (6) koje "vidi" kamera u centru kocke
- Pretpostavka: kocka je veoma (beskonačno) velika
  - dobar rezultat samo za velike scene gde se "ogleda nebo"
- Pozitivno: efikasno, hardverska podrška
- Negativno: za dinamične scene, slike moraju da se generišu pri svakoj promeni
- Kompromis: lokalizovane mape, za različite delove scene

Reflektujući i proziran materijal

- Primer za *cube map*





Zadatak za vežbu: napraviti Unity program koji formira svih 6 slika
za *cube map* trenutne scene, snimi ih u .png i formira jednu *cube map* od toga.

Reflektujući i proziran materijal

- Pravljenje *cube map*-e u Unity
  - 1 tekstura (Type: Advanced, Mapping: 6 Frames Layout)
  - 6 tekstura koje se objedine u jedan objekat (Assets/Legacy/CubeMap)

- Postavljanje *cube map*-e u Unity:
  - Globalno: Window/Lighting/Scene/Skybox
  - Po kameri: Dodati komponentu SkyBox

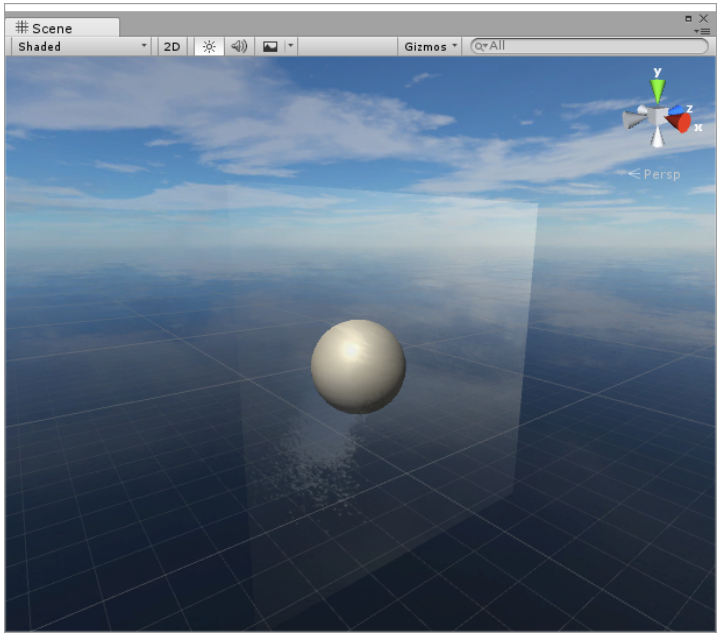
- Podrazumeva se:
  - strane kocke su paralelne osama k.s.s.
  - kocka je centrirana u (0, 0) k.s.s.

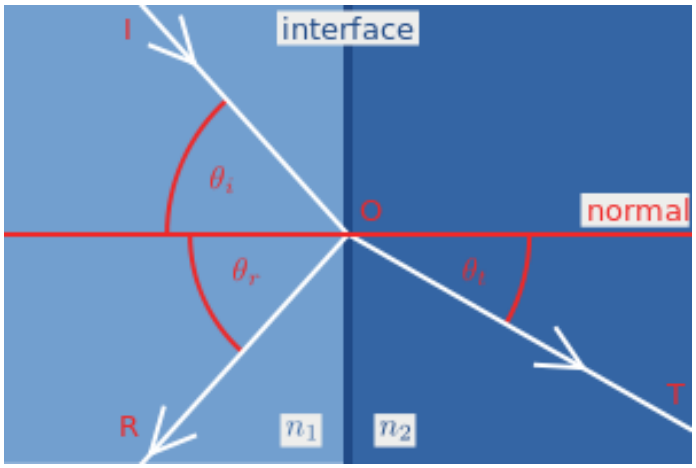
Reflektujući i proziran materijal

- Kako izabrati koja od 6 tekstura se koristi?
  - potrebna trodimenziona koordinata (dvodimenziona UV nije dovoljna)
  - strana kocke se bira na osnovu komponente najvecećg intenziteta
  - +X ili –X se bira ako je X komponenta dominantna
  - ostale 2 se koriste kao "UV"

Reflektujući i proziran materijal

- Bitno svojstvo materijala: intenzitet refleksije zavisi od ugla posmatranja
- Frenelova (Fresnel) reflektansa
  - realna formula
  - Šlikova aproksimacija
  - Tabelirana funkcija (tekstura)





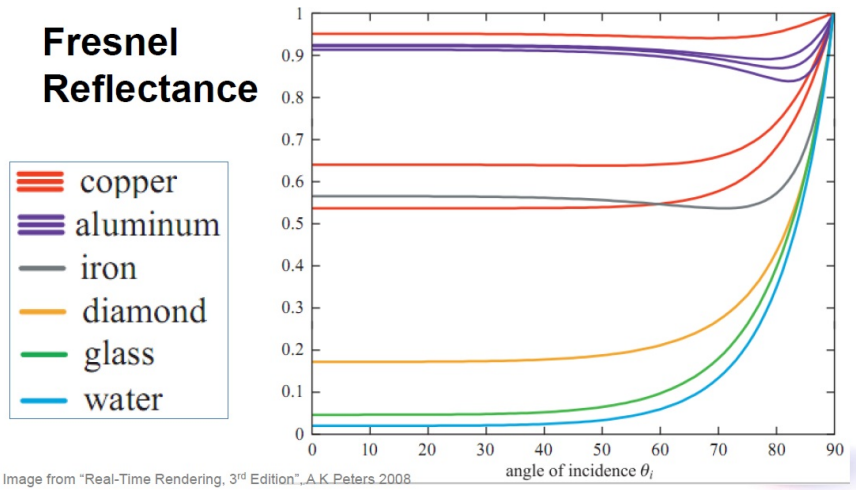


Image from "Real-Time Rendering, 3rd Edition", A K Peters 2008

Reflektujući i proziran materijal

```
Shader "RG2/06_GlassShader"
{
    Properties
    {
        _CubeMap ("Cube map", CUBE) = "white" {}
        _Transparency("Transparency", Range(0, 1)) = 1.0
    }
    SubShader
    {
        Tags {  "Queue" = "Transparent" "LightMode"="ForwardBase" }

        Blend SrcAlpha OneMinusSrcAlpha

        ZWrite off

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
```

Reflektujući i proziran materijal

```
struct appdata {
    float4 vertex : POSITION;
    float3 normal : NORMAL;
};

struct v2f {
    float3 normal : NORMAL;
    float3 vertexPosInObjSpace : TEXCOORD0;
    float4 vertex : SV_POSITION;
};

samplerCUBE _CubeMap;
float _Transparency;

v2f vert (appdata v) {
    v2f o;
    o.vertex = UnityObjectToClipPos(v.vertex);
    o.normal = v.normal;
    o.vertexPosInObjSpace = v.vertex;
    return o;
}
```

Reflektujući i proziran materijal

Ako se viewDir računa u Vertex shader-u, uočljive vizuelne deformacije

```
            fixed4 frag (v2f i) : SV_Target
            {
                float3 viewDir =
                   normalize(ObjSpaceViewDir(float4(i.vertexPosInObjSpace, 1))).xyz;

                float3 uv = reflect( -viewDir, normalize(i.normal));

                uv = mul(UNITY_MATRIX_M, float4(uv, 0));

                fixed4 col = texCUBE(_CubeMap, uv);
                col.a = (1-dot(viewDir, i.normal))*(1-_Transparency);
                return col;
            }
            ENDCG
        }
    }
}
```

Ovaj program za senčenje ne razmatra Frenelovu reflektansu.
Vežba: dodati Frenelovu reflektansu!

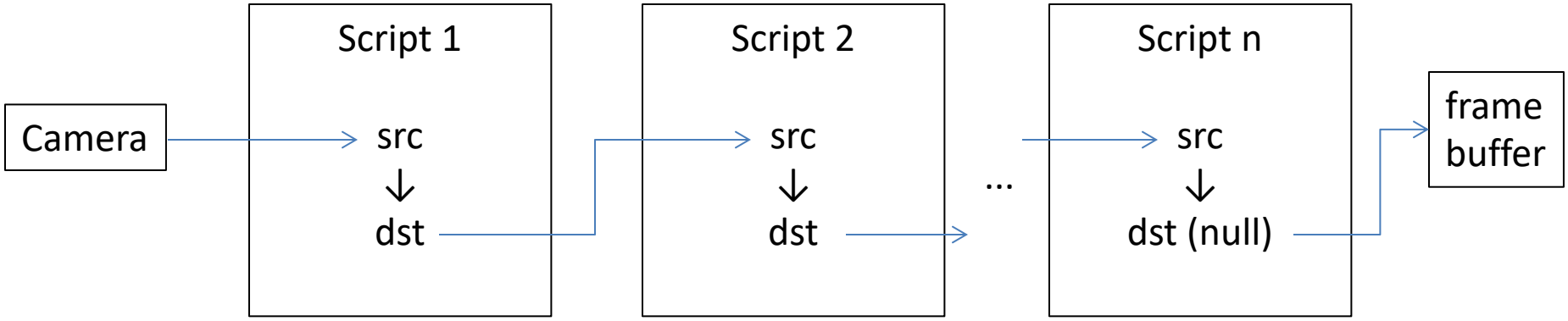
Naknadna obrada slike

- Eng.: postprocessing
- Obrada se vrši nad prethodno sintetizovanom slikom
- Obično se vrši nad celom slikom, ali nije neophodno
- Postizanje efekata:
  - promena nijanse boja (desaturacija, hue shifting, ...)
  - promena rezolucije
  - zamućenje (blur)
  - lens flare
  - ...

Samostalna vežba: promena H komponente u HSV sistemu

Naknadna obrada slike

- Kameri se doda `MonoBehaviour` komponenta
  - proizvoljan broj
  - primenjuju se u redosledu dodavanja
- `void OnRenderImage(RenderTexture src, RenderTexture dst)`
- Poziva se nakon što kamera formira sliku

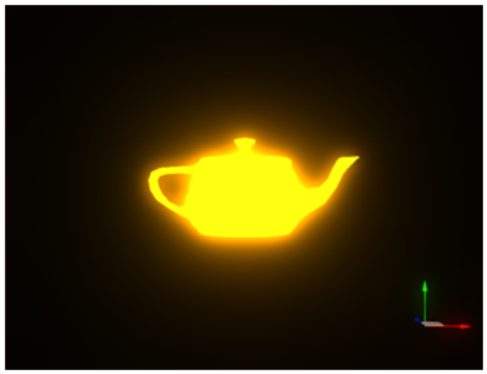


Naknadna obrada slike

- `Graphics.Blit`
  - `public static void Blit(Texture source, RenderTexture dest);`
  - `public static void Blit(Texture source, RenderTexture dest, Material mat, int pass = -1);`
  - `public static void Blit(Texture source, Material mat, int pass = -1);`
- Primenjuje zadati materijal (program za senčenje) i prolaz
- Vrši kopiranje izvorišne slike u odredišnu (bit block transfer), odnosno crta pravougaonik preko odredišne slike
- Postavlja **`dest`** kao *render target*
- Postavlja **`source`** kao `_MainTex` materijala

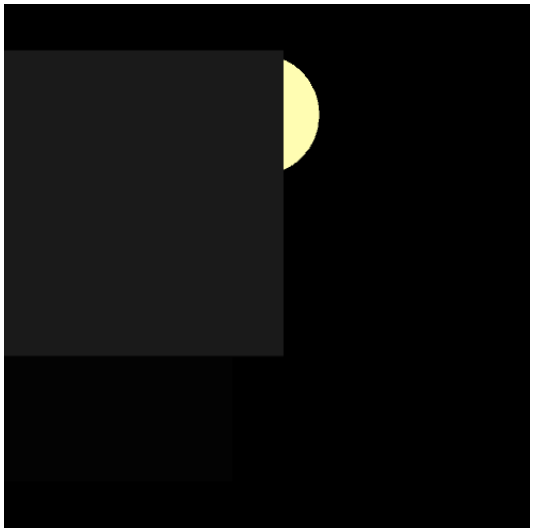
Bloom efekat

- "Isijavanje" (bukvalno: cvetanje)
- Svetli delovi formirane slike se delimično prelivaju preko susednih tamnih delova
- Problem: **prelivanje** nije u skladu sa rasterizacijom poligona



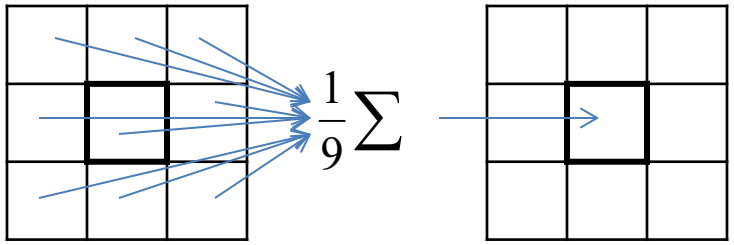
Bloom efekat

- Ideja: formirati filtriranu sliku originala
  - niskopropusni filtar (zamućenje, prelivanje, nema oštrih ivica)
  - optimizacija: smanjiti veličinu filtrirane slike
- Konačnu sliku formirati kombinovanjem



Bloom efekat

- U ovom rešenju (nije optimizovano)
  - Posebna kamera formira sliku niže rezolucije (BloomRenderTexture)
  - Skripta vrši filtriranje (L07_Blur.cs)



- Glavna kamera
  - Formira sliku
  - Kombinuje filtriranu i formiranu (L07_Bloom.cs)

- Nisu neophodne 2 kamere
  - Može da se izvede pomoću jedne kamere
  - Videti Unity Bloom efekat i prateće klase/programe za senčenje

Bloom efekat

```
using UnityEngine;
using System.Collections;

public class L07_Blur : MonoBehaviour {

    [SerializeField]
    private  Shader  BlurShader;

    Material  m_Material = null;
    protected  Material  material {
        get {
            if (m_Material == null) {
                m_Material = new Material(BlurShader);
                m_Material.hideFlags = HideFlags.DontSave;
            }
            return  m_Material;
        }
    }
```

L07_Blur.cs

Bloom efekat

```
    protected void OnDisable() {
        if (m_Material) {
            DestroyImmediate(m_Material);
        }
    }

    void OnRenderImage(RenderTexture src, RenderTexture dst){
        int rtW = src.width / 4;
        int rtH = src.height / 4;
        RenderTexture buffer = RenderTexture.GetTemporary(rtW, rtH, 0);

        material.SetTexture("_MainTexture", src);
        Graphics.Blit(src, buffer, material);

        material.SetTexture("_MainTexture", buffer);
        Graphics.Blit(buffer, dst, material);

        RenderTexture.ReleaseTemporary(buffer);
    }
}
```

L07_Blur.cs

Bloom efekat

```
using UnityEngine;
using System.Collections;

public class L07_Bloom : MonoBehaviour {

    [SerializeField]
    private RenderTexture BlurTexture;

    [SerializeField]
    private Shader BloomShader;

    static Material m_Material = null;
    protected Material material {
        get {
            if (m_Material == null)    {
                m_Material = new Material(BloomShader);
                m_Material.hideFlags = HideFlags.DontSave;
            }
            return m_Material;
        }
    }
```

L07_Bloom.cs

Bloom efekat

```
    protected void OnDisable() {
        if (m_Material) {
            DestroyImmediate(m_Material);
        }
    }

    void OnRenderImage(RenderTexture src, RenderTexture dst) {
        material.SetTexture("_MainTexture", src);
        material.SetTexture("_BlurTexture", BlurTexture);
        Graphics.Blit(src, dst, material);
    }
}
```

L07_Bloom.cs

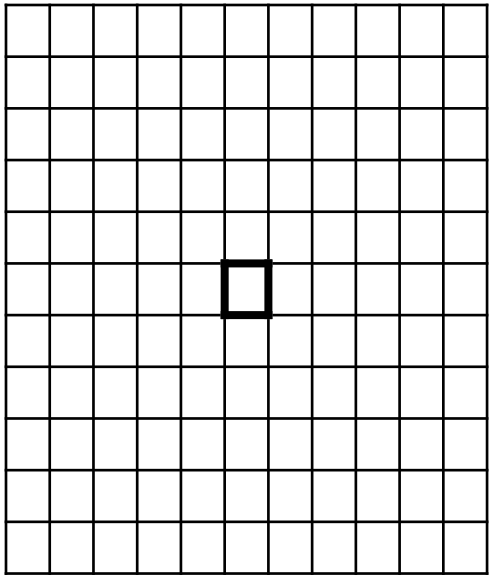
Bloom efekat

```
Shader "RG2/07_BlurShader" {
    SubShader     {
        Tags { "RenderType"="Opaque" }

        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
            struct appdata
            {
                float4 vertex : POSITION;
            };
            struct v2f
            {
                float4 vertex : SV_POSITION;
                float2 screenPos : TEXCOORD0;
            };
            sampler2D _MainTexture;
            // Unity popunjava: x=1/width, y=1/height, z=width, w=height
            float4 _MainTexture_TexelSize;
```

07_BlurShader

https://docs.unity3d.com/Manual/SL-UnityShaderVariables.html

Bloom efekat



07_BlurShader

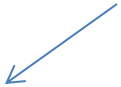
```
    v2f vert (appdata v)     {
      v2f o;
      o.vertex = UnityObjectToClipPos(v.vertex);
      o.screenPos = (o.vertex.xy/o.vertex.w)*0.5+0.5;
      return o;
    }

    fixed4 frag (v2f i) : SV_Target {
      int filterSize = 5;  int maxSteps = 11;  float divisor = 121;
      float2 uv = i.screenPos - filterSize*_MainTexture_TexelSize.xy;
      float4 color = float4(0,0,0,0);
      for(int i = 0; i < maxSteps; i++) {
        for(int j = 0; j < maxSteps; j++) {
          float4 tmp = tex2D(_MainTexture, uv +
                             float2(_MainTexture_TexelSize.x*j,
                                    _MainTexture_TexelSize.y*i)) + 0.35;
          color += tmp*tmp*tmp;
        }
      }
      return fixed4(color.rgb/divisor, 1);
    }
    ENDCG
  }
  }
}
```

Eksperimentalno utvrđeno

Bloom efekat

```
...
sampler2D _MainTexture;
sampler2D _BlurTexture;

v2f vert (appdata v) {
v2f o;
  o.vertex = UnityObjectToClipPos(v.vertex);
  o.screenPos = (o.vertex.xy/o.vertex.w)*0.5+0.5;
  return o;
}
```

```
#if UNITY_UV_STARTS_AT_TOP
```



```
fixed4 frag (v2f i) : SV_Target {
  float2 uvMain = float2(i.screenPos.x, 1-i.screenPos.y);
  float2 uvBlur = i.screenPos;
  fixed4 mainColor = tex2D(_MainTexture, uvMain);
  fixed4 bloomColor = tex2D(_BlurTexture, uvBlur);

  float factor = dot(fixed3(0.2126, 0.7152, 0.0722), bloomColor.rgb);
  return lerp(mainColor, bloomColor, factor);
}
```

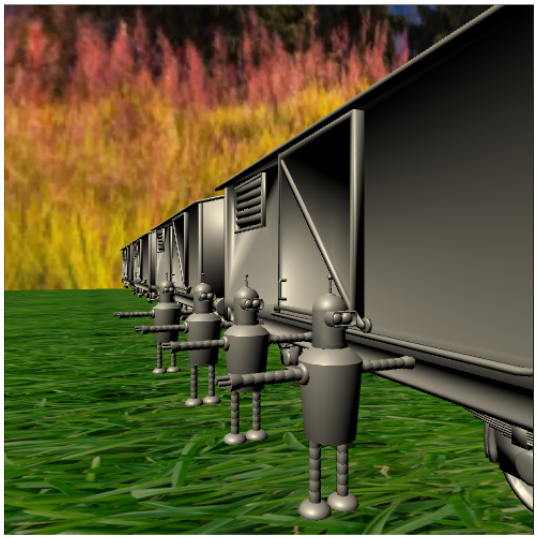
07_BloomShader

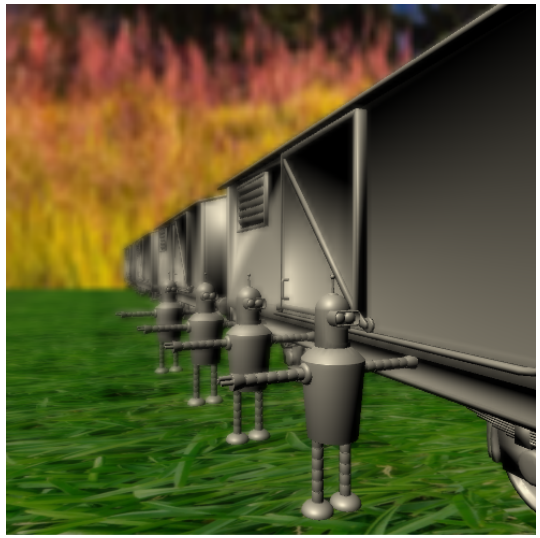
Bloom efekat

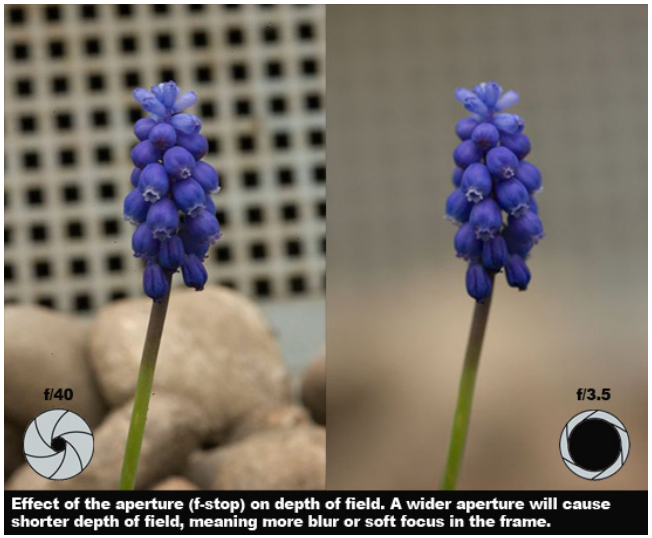
- Samostalan rad
  - proučiti kako je implementiran Bloom efekat u paketu koji se distribuira uz Unity
  - implementirati efikasnije filtriranje teksture
  - pronaći bolji način da se istaknu svetli delovi spram ostatka (umesto dodavanja neke konstante i računanja trećeg stepena)

Depth of Field (DoF) efekat

- Simulacija posmatranja scene kamerom
  koja fokusira objekte na određenoj razdaljini
- Ideja zasnovana na zakonima optike,
  ali često realizovana trikovima
- Često dovoljno dobro da se vizuelno dočara efekat
  u dinamičnim scenama







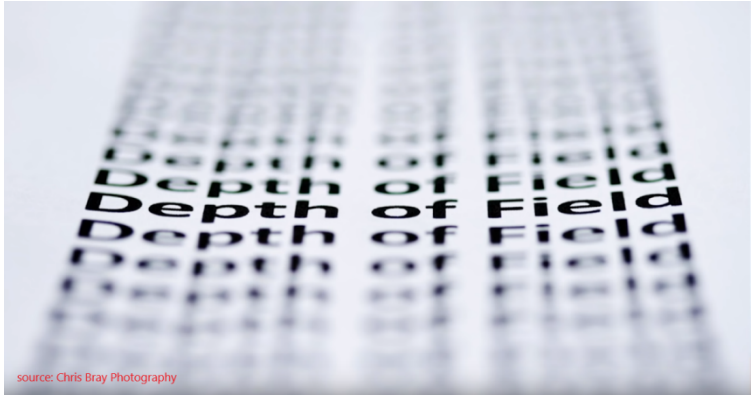
DoF efekat

- Moguća realizacija:
  - Kamera "posmatra" istu tačku dok trpi relativno male pomeraje
  - Dobijene slike se akumuliraju i odredi se prosek
  - Fokusirani elementi su oštri, nefokusirani mutni
  - Loša osobina: zahteva veći broj crtanja cele scene

- Realizacija obradom slike:
  - Potrebna je "zamućena" slika (optimizacija kao kod Bloom efekta)
  - Potrebno poznavati razdaljinu svakog piksela od kamere (Z-buffer)
  - Kombinovati polaznu i "zamućenu" sliku
    prema razdaljini od fokusirane razdaljine

DoF efekat

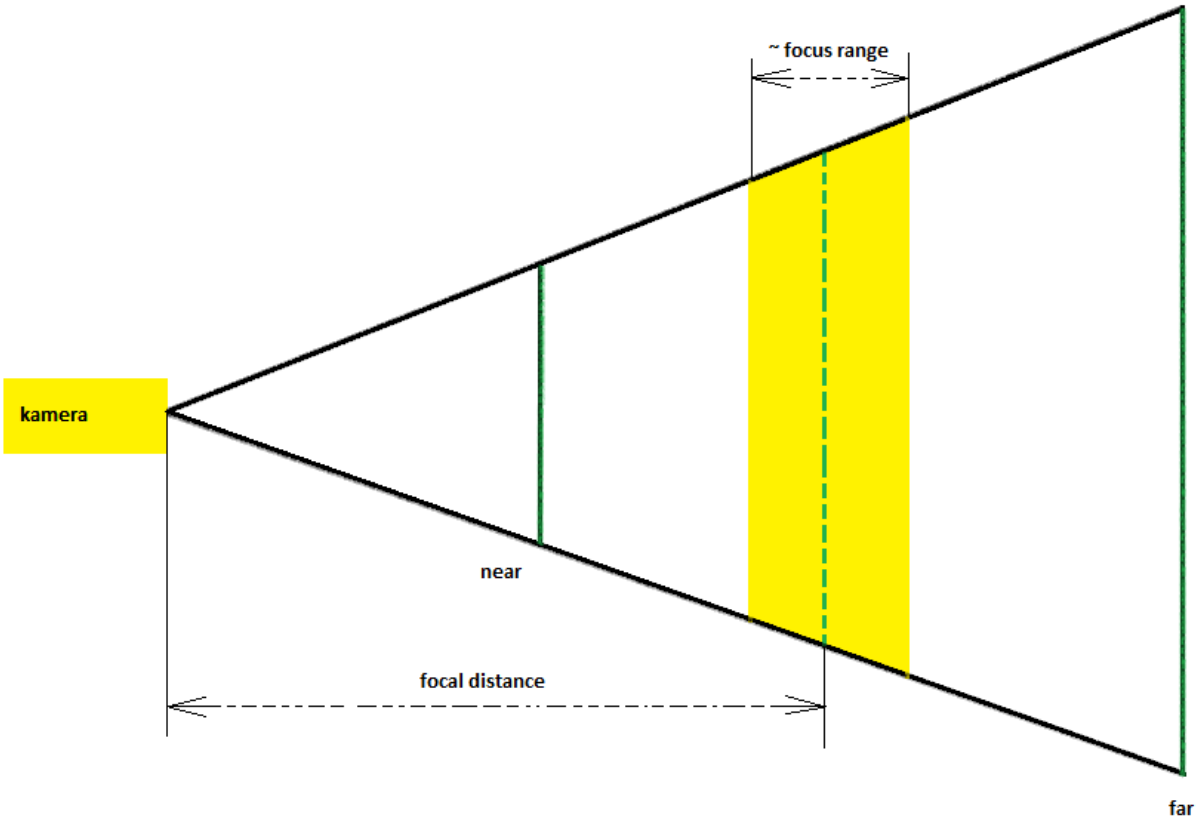

source: Chris Bray Photography


source: Chris Bray Photography


source: Chris Bray Photography

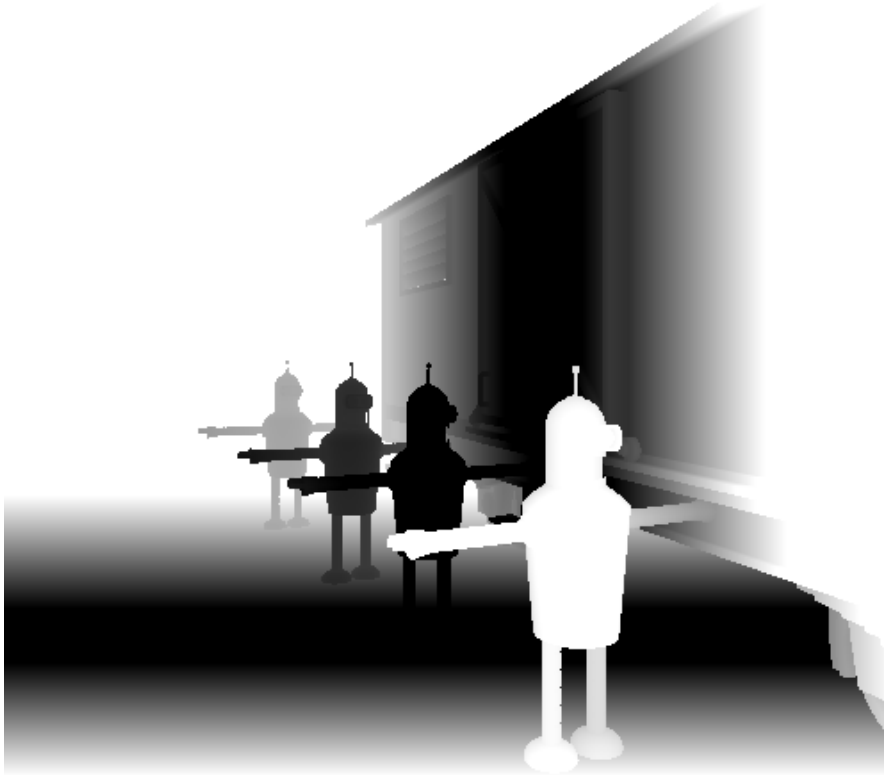

source: Chris Bray Photography

DoF efekat



DoF efekat

Bafer dubine



Zona fokusa (crno)

DoF efekat

```
void OnRenderImage(RenderTexture src, RenderTexture dst)
{
    int rtW = src.width / 4;
    int rtH = src.height / 4;
    RenderTexture buffer = RenderTexture.GetTemporary(rtW, rtH, 0);

    DownsampleMaterial.SetTexture("_MainTex", src);
    Graphics.Blit(src, buffer, DownsampleMaterial);

    DoFMaterial.SetFloat("FocusRange", focusRange);
    DoFMaterial.SetFloat("FocalDistance", focalDistance);
    DoFMaterial.SetTexture("_DownsampledTex", buffer);
    DoFMaterial.SetTexture("_MainTex", src);
    Graphics.Blit(src, dst, DoFMaterial);
}
```

L09_DoF.cs

DoF efekat

```
sampler2D _MainTex;
// x=1/width, y=1/height, z=width, w=height
float4 _MainTex_TexelSize;

v2f vert (appdata v) {
  v2f o;
  o.vertex = UnityObjectToClipPos(v.vertex);
  o.screenPos = (o.vertex.xy/o.vertex.w)*0.5+0.5;
  return o;
}

fixed4 frag (v2f i) : SV_Target {
  float2 delta = _MainTex_TexelSize.xy;
  float2 uvMain = float2(i.screenPos.x, 1-i.screenPos.y);
  half4 sample1 = tex2D(_MainTex, uvMain + delta*float2(0.5,0.5));
  half4 sample2 = tex2D(_MainTex, uvMain + delta*float2(-0.5,0.5));
  half4 sample3 = tex2D(_MainTex, uvMain + delta*float2(0.5,-0.5));
  half4 sample4 = tex2D(_MainTex, uvMain + delta*float2(-0.5,-0.5));

  return (sample1+sample2+sample3+sample4)*0.25;
}
```

09_DownsampleShader

DoF efekat

```
sampler2D _MainTex;
sampler2D _DownsampledTex;
float4 _DownsampledTex_TexelSize;
sampler2D _CameraDepthTexture;
float FocalDistance;
float FocusRange;

v2f vert (appdata v) {
v2f o;

   o.vertex = UnityObjectToClipPos(v.vertex);
   o.screenPos = (o.vertex.xy/o.vertex.w)*0.5+0.5;
   return o;
}
```

09_DoFShader

DoF efekat

```
fixed4 frag (v2f i) : SV_Target  {
  float2 uv = i.screenPos.xy;
  float2 uvInverted = float2(i.screenPos.x, 1-i.screenPos.y);
  fixed4 depth = tex2D(_CameraDepthTexture, uv);
  fixed4 mainColor = tex2D(_MainTex, uvInverted);
  float4 blurColor = tex2D(_DownsampledTex, uv);

  float zDist =
    (FocalDistance-_ProjectionParams.y)/(_ProjectionParams.z-_ProjectionParams.y);
  float offFocus = pow(abs(depth.r-zDist), FocusRange);

  return lerp(mainColor, blurColor, offFocus);
}
```

09_DoFShader

https://docs.unity3d.com/Manual/SL-UnityShaderVariables.html
_ProjectionParams.y = near clipping plane
_ProjectionParams.z = far clipping plane