# HW4 problem1

## Code

Properties {  
  
 \_Rough1 ("Roughness for Yellow ", Range (0, 1)) = 0.5  
 \_rough2("Roughness for Blue", Range (0, 1)) = 0.5  
}

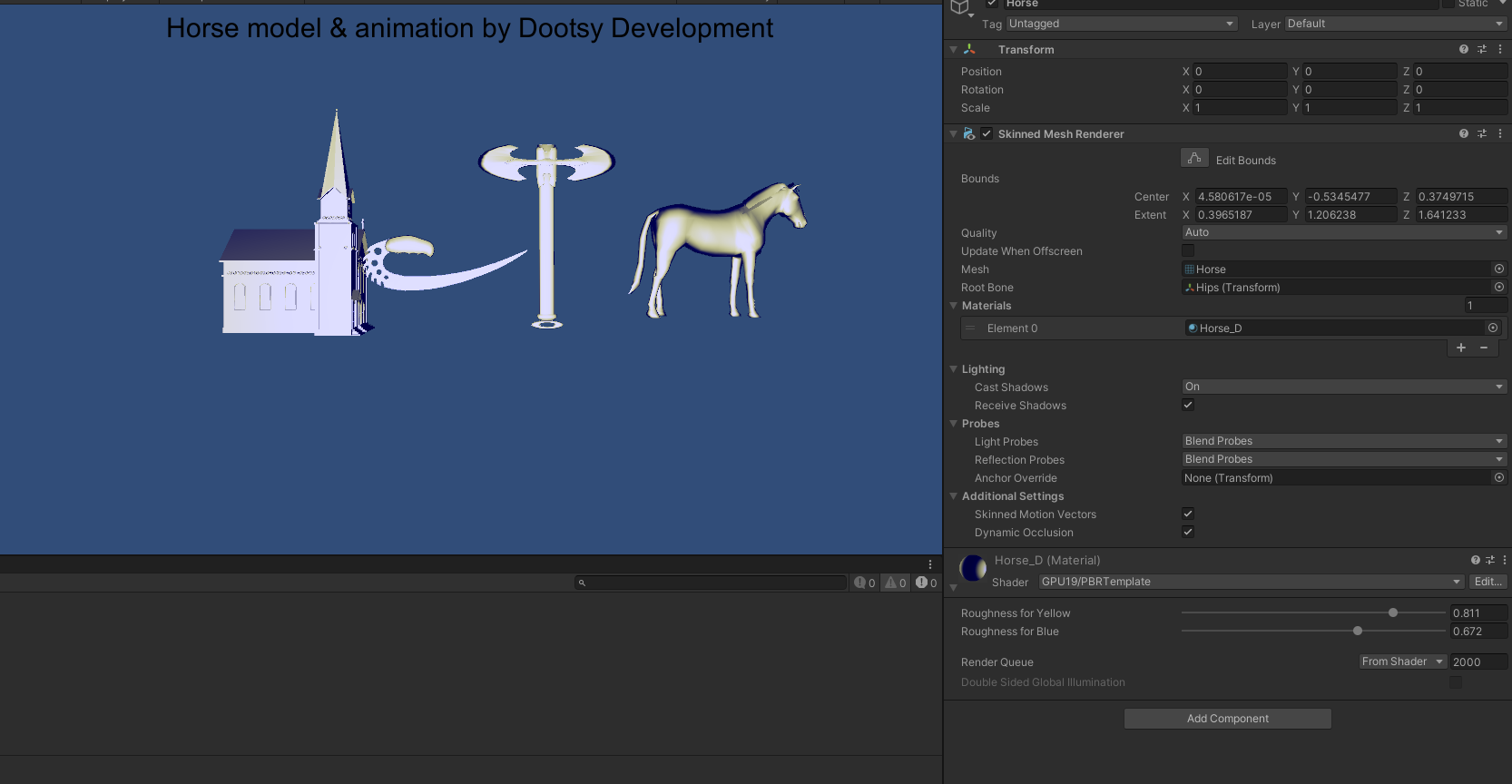
sampler2D \_BaseTex;   
float4 \_BaseTex\_ST;  
  
float4 \_LightColor0; // Unity fills for us  
float \_Rough1;  
float \_rough2;

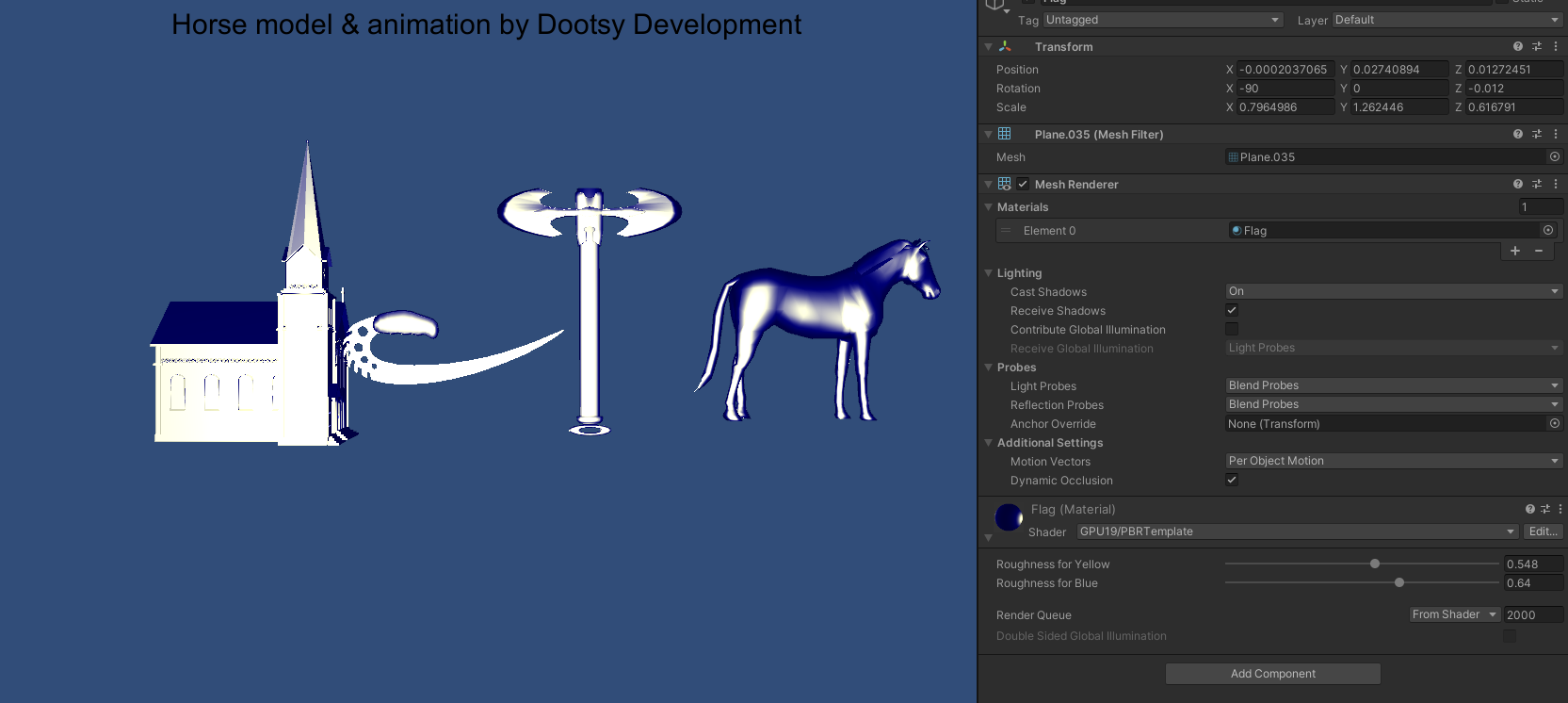
float4 FragPBRTemplate(v2f input) : COLOR {  
 // Unity light position convention is:  
 // w = 0, directional light, with x y z pointing in opposite of light direction   
 // w = 1, point light, with x y z indicating position coordinates  
 float3 unNormLightDir = \_WorldSpaceLightPos0.xyz - input.positionWS \* \_WorldSpaceLightPos0.w;  
 float3 lightDir = normalize(unNormLightDir);  
 // renormalizing because the GPU's interpolator doesn't know this is a unit vector  
 float3 n = normalize(input.normalWS);  
  
 float3 viewDir = normalize(\_WorldSpaceCameraPos - input.positionWS);   
 float3 halfVector = normalize(viewDir + lightDir);  
 float NdotM = dot(n,halfVector);  
 float pi = 3.14159;  
  
 // roughness calculation  
 float alpha1 = \_Rough1 \* \_Rough1;  
 float alphaSquare1 = alpha1 \* alpha1;  
 // beckman normal ditribution  
 float exponenet = ((pow(dot(n,halfVector),2) -1)/(pow(alpha1,2)\*(pow(dot(n,halfVector),2))));  
 float beckMan = (1/((pi\*alphaSquare1)\*(pow(dot(n,halfVector),4))));  
 float Beckmann3 = beckMan\*exp(exponenet);  
  
 // rough ness 2  
 float alpha2 = \_rough2 \* \_rough2;  
 float alphaSquare2 = alpha2 \* alpha2;  
 // Blinn-Phong distributation  
  
 float blingEXP =(2/alphaSquare2) -2 ;  
 float blingDinominator = 1/(pi\*alphaSquare2);  
 float BlinnPhong = blingDinominator\*pow(dot(n,halfVector),blingEXP);  
 //  
 float GGXDenominator = pi\*pow((pow(NdotM,2)\*(alphaSquare2-1) + 1),2);  
 float ggx = alphaSquare2 / GGXDenominator;  
 float3 output = float3(Beckmann3,Beckmann3,ggx) ; // A placeholder, replace with your own code  
 return(float4(output,1));

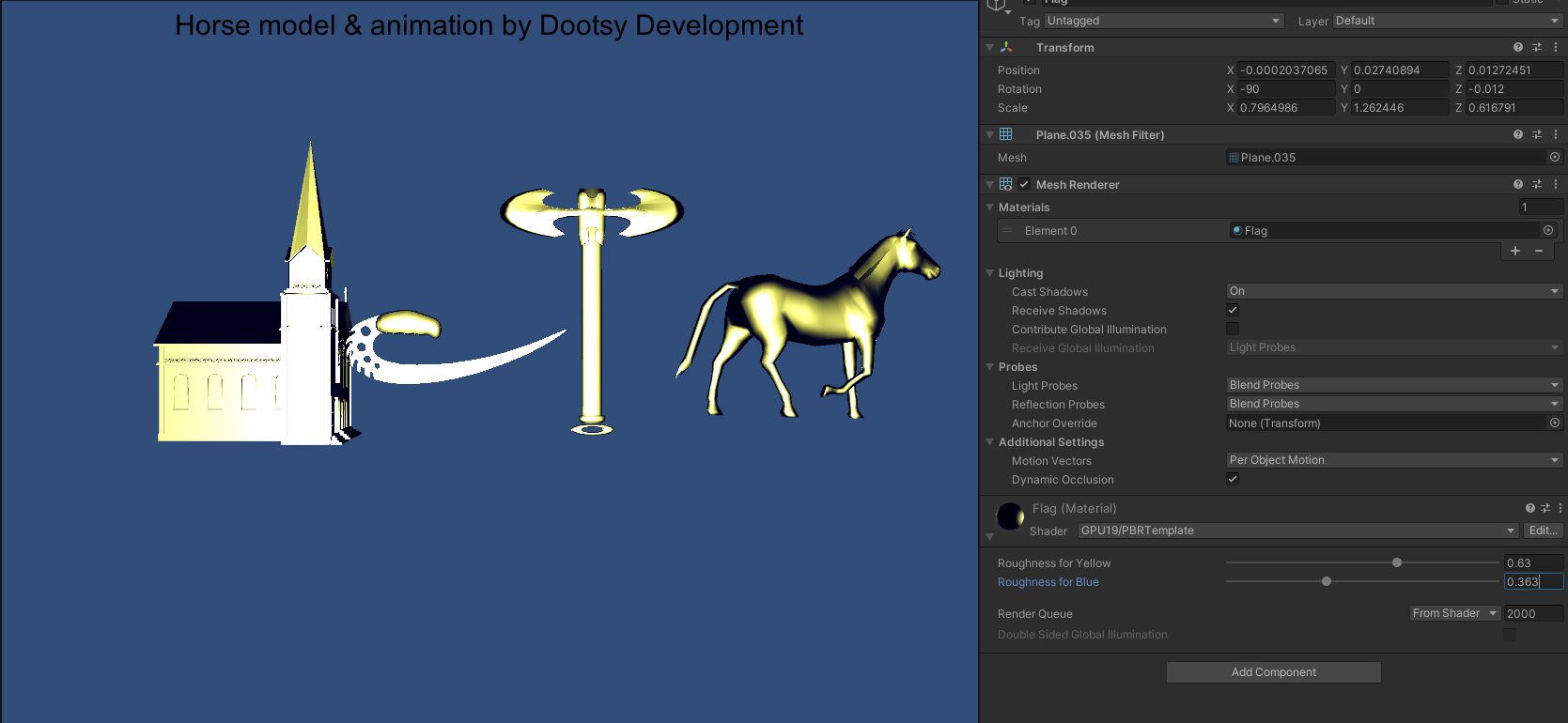
* As you can see I have tried both the model, I am not sure if I was suppose to do that, but out of curiosity I did it anyway.

## Output

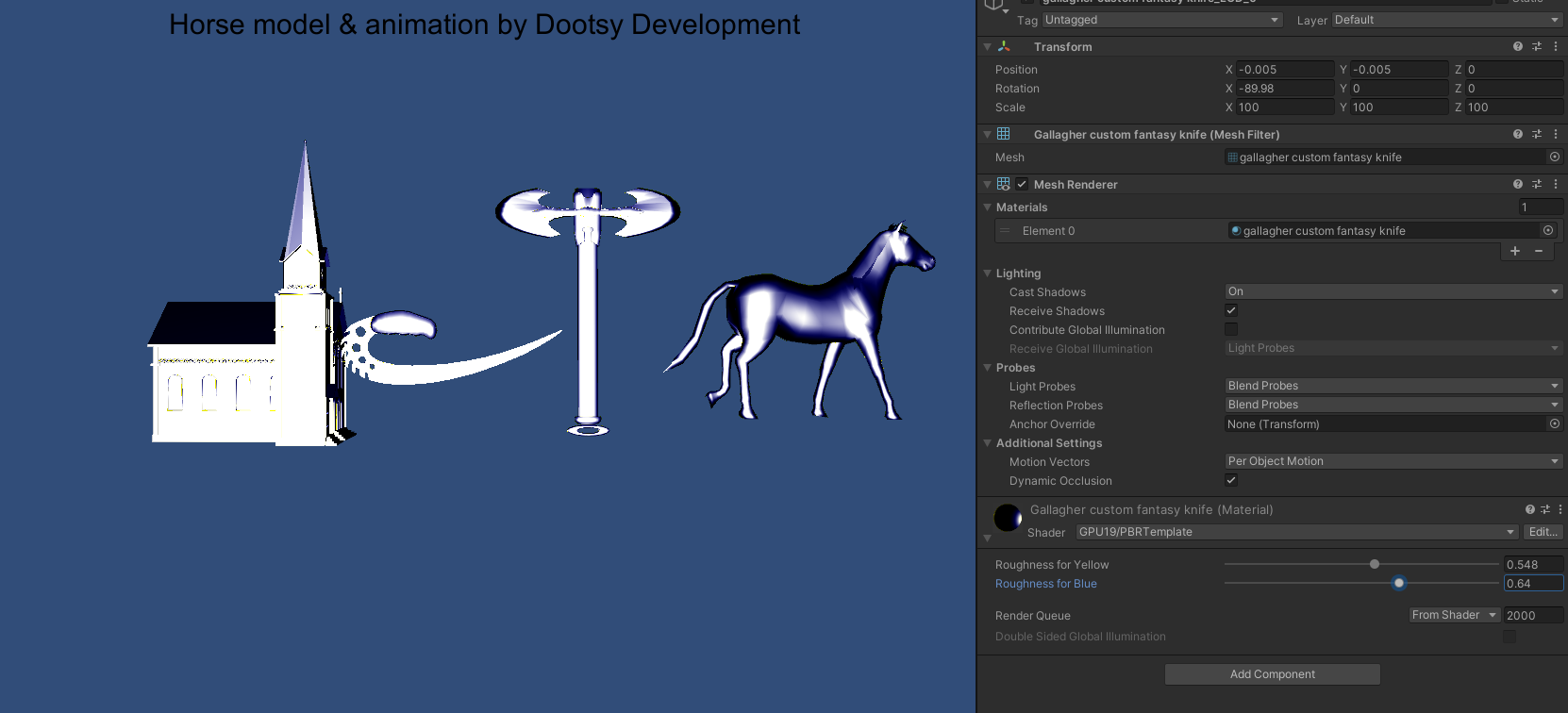
With ggx as the second distribution function.

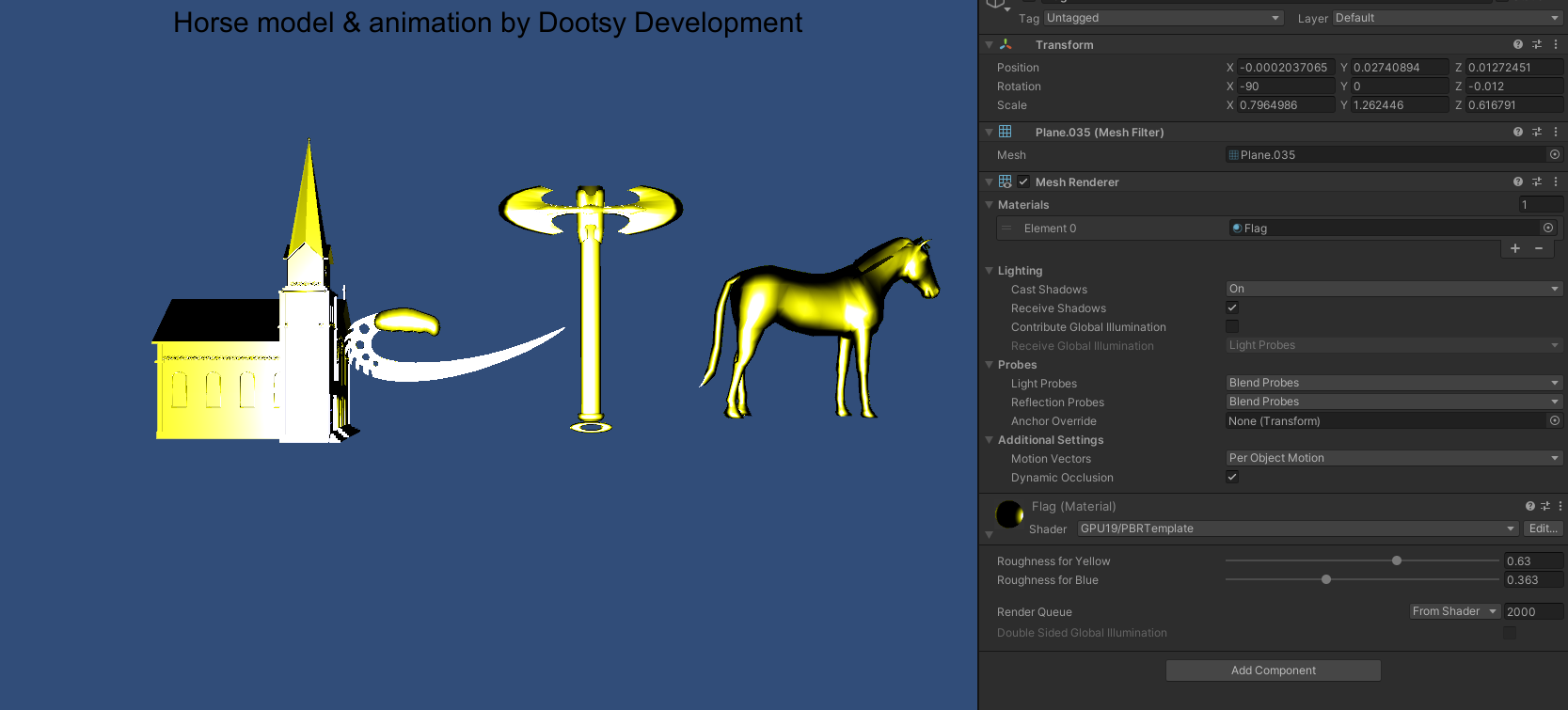


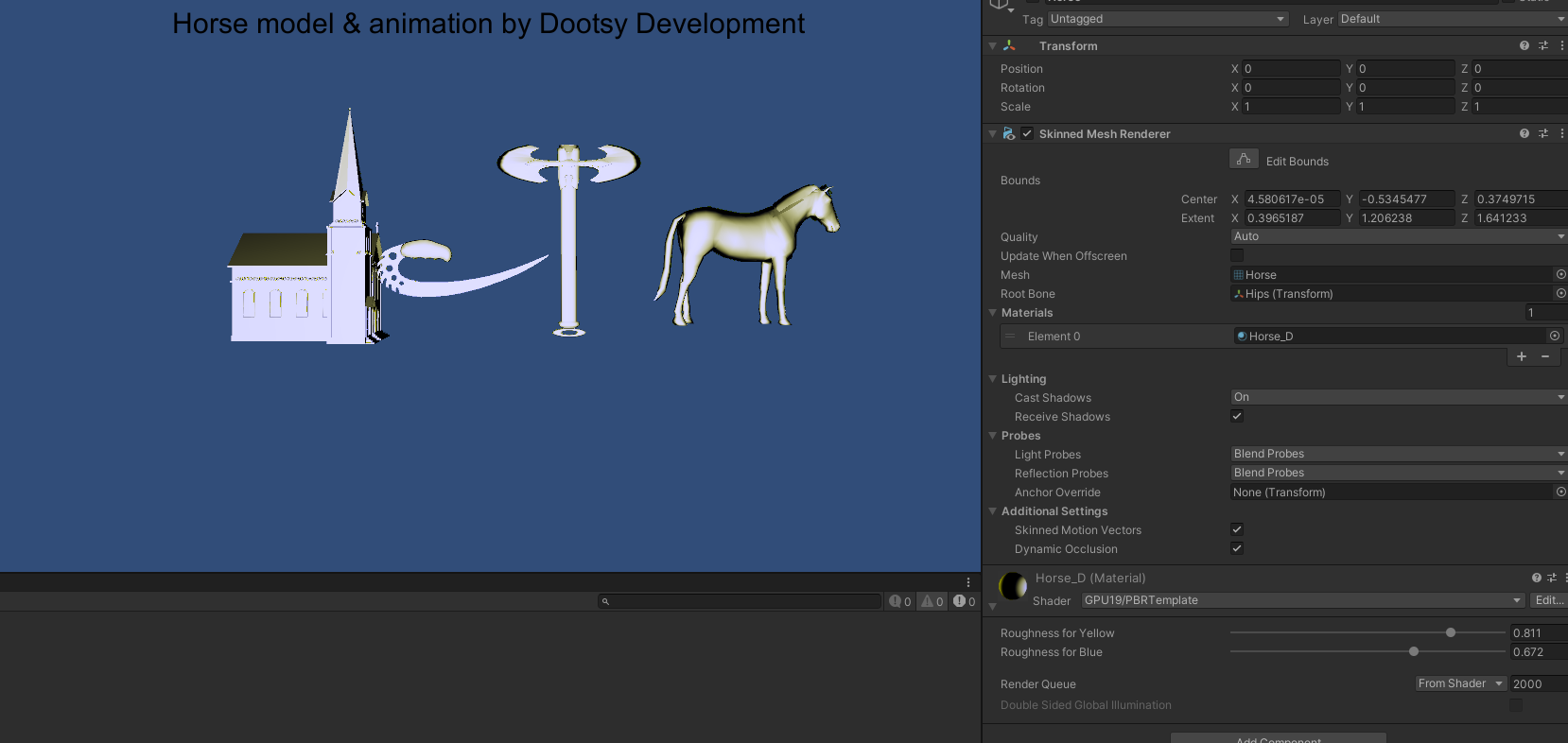




With the Blinn-Phong







# HW4\_problem2

## Code:

clear; close all; clc;

rough = 0.412;

rough2 = 0.5;

rough3 = 0.389;

a = rough \* rough;

a2 = rough2 \* rough2;

a3 = rough3 \* rough3;

x = (0:0.01:1);

beck = 1./(pi \* a.^2.\*(x.^4)) .\* exp((x.^2 - 1)./ (a.^2 \* x.^2));

bling = (1/pi\*a2.^2) .\* (power(x,((2/a2.^2) - 2)));

ggx = (a3^2) ./( pi \* ((x.^2)\*(a3^2 - 1) +1).^2);

plot(x, beck,'DisplayName','Beckmann');

hold on;

plot(x, bling,'DisplayName','Blinn-Phong');

hold on;

plot(x, ggx, 'DisplayName','GGX');

lgd = legend;

## output

Chart

Description automatically generated with medium confidence

# HW4\_Problem3:

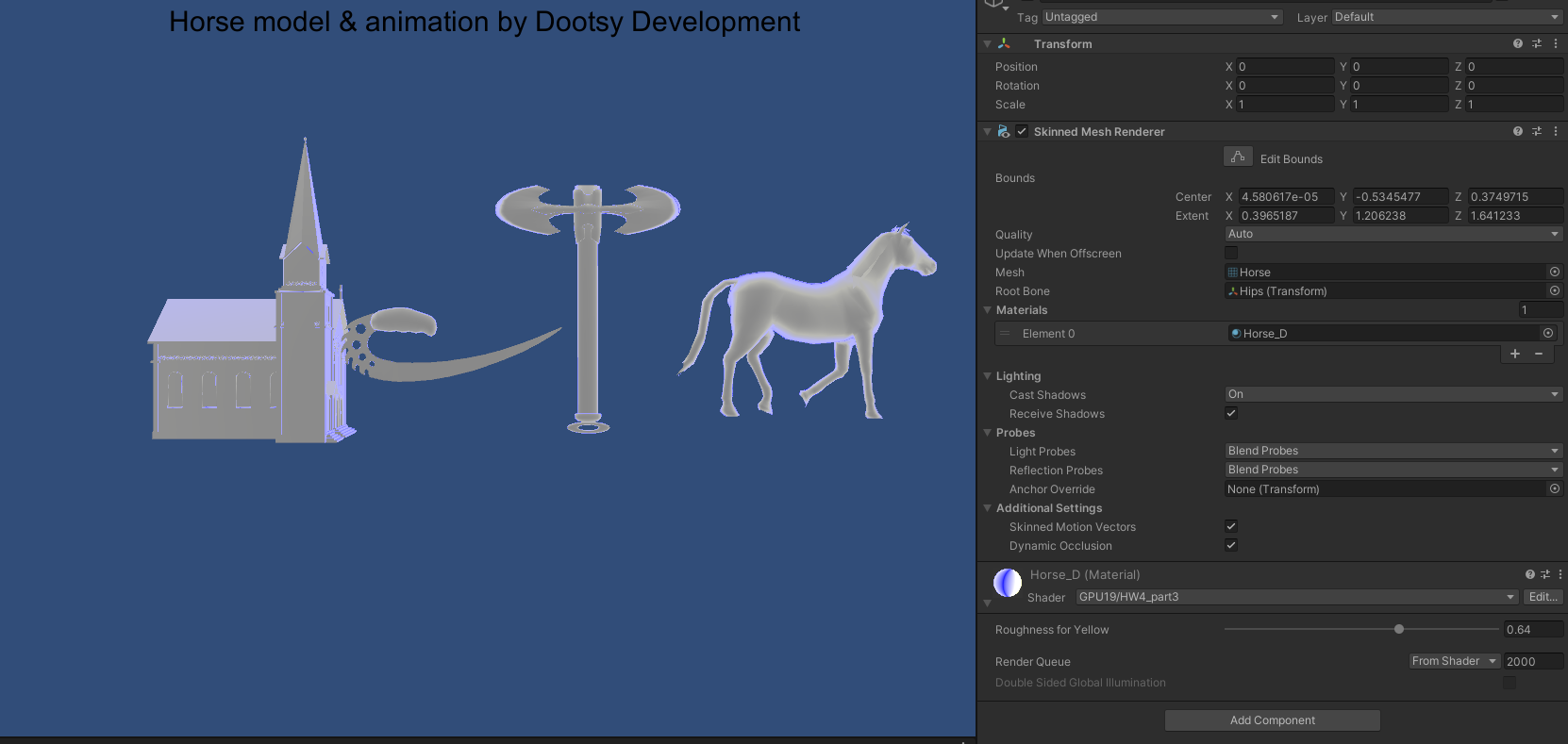
## Code

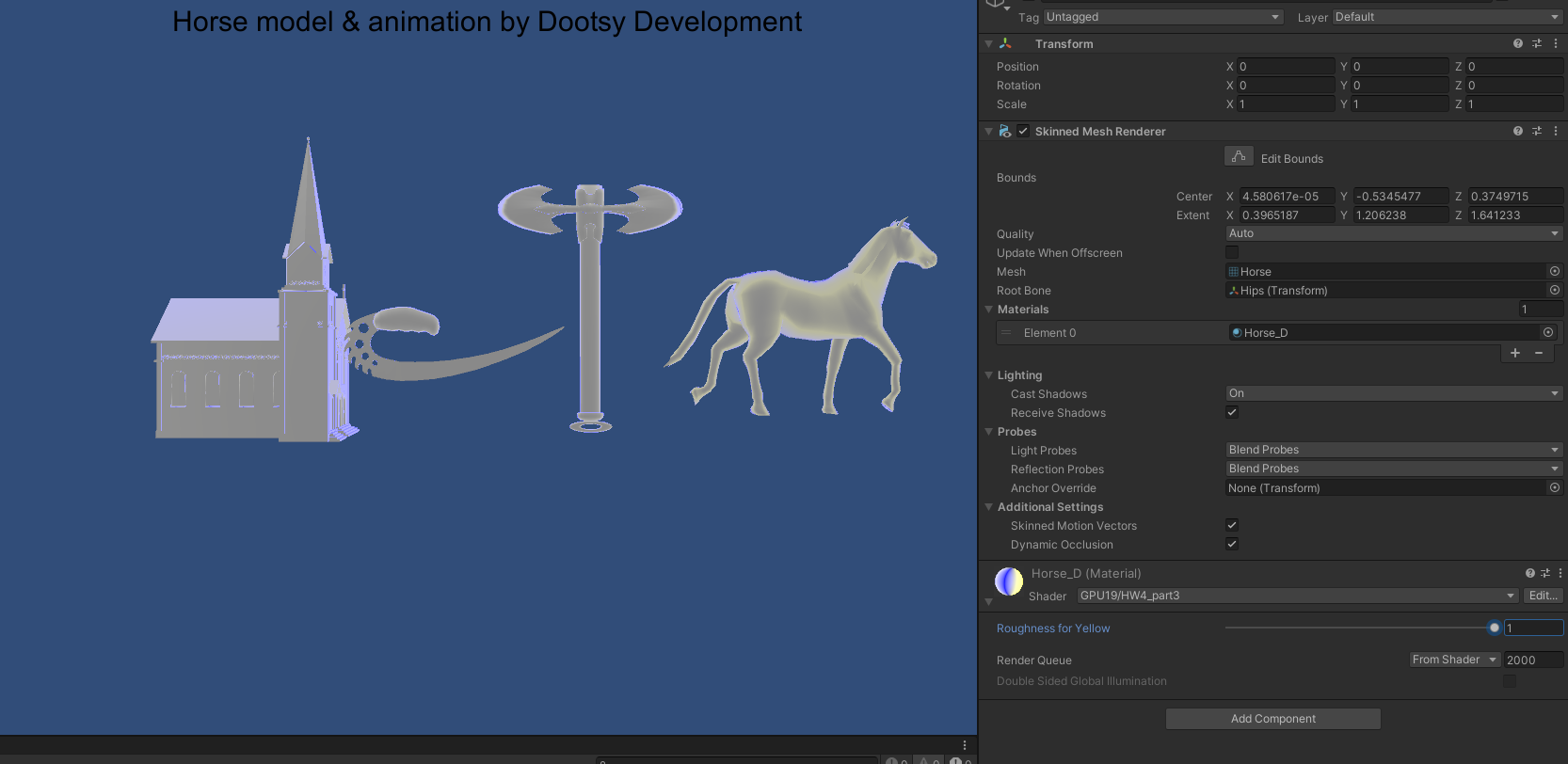
Properties {  
 \_Rough1 ("Roughness for Yellow ", Range (0, 1)) = 0.5  
}

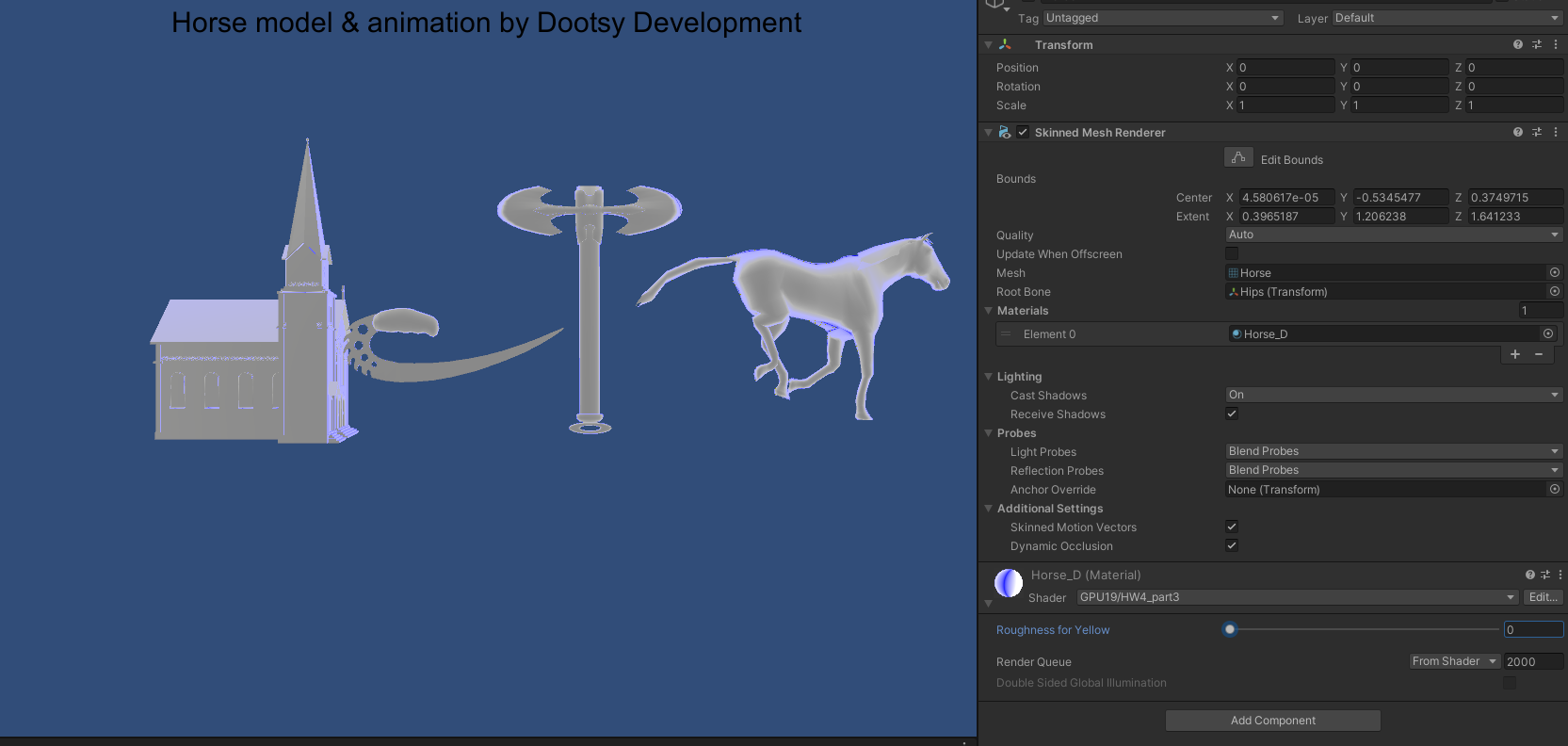
sampler2D \_BaseTex;   
float4 \_BaseTex\_ST;  
  
float4 \_LightColor0; // Unity fills for us  
float \_Rough1;

float4 FragPBRTemplate(v2f input) : COLOR {  
 // Unity light position convention is:  
 // w = 0, directional light, with x y z pointing in opposite of light direction   
 // w = 1, point light, with x y z indicating position coordinates  
 float3 unNormLightDir = \_WorldSpaceLightPos0.xyz - input.positionWS \* \_WorldSpaceLightPos0.w;  
 float3 lightDir = normalize(unNormLightDir);  
  
 // renormalizing because the GPU's interpolator doesn't know this is a unit vector  
 float3 n = normalize(input.normalWS);  
  
 float3 viewDir = normalize(\_WorldSpaceCameraPos - input.positionWS);   
 float3 halfVector = normalize(viewDir + lightDir);  
  
 float pi = 3.14159;  
  
 // roughness calculation  
 float alpha1 = \_Rough1 \* \_Rough1;  
  
 // Cook-torrance  
 float cook = (2 \* (dot(n,halfVector)) \* (dot(n,viewDir))) / (dot(viewDir,halfVector));  
 float torrance = (2 \* (dot(n,halfVector)) \* (dot(n,lightDir))) / (dot(viewDir,halfVector));  
 float CookTorrance =min(1, min(cook, torrance))/ (4\*dot(n,lightDir)\*dot(n,viewDir));  
  
  
 // GGX  
 float ggxV = (2\*dot(n,viewDir))/(dot(n,viewDir)+ sqrt(pow(alpha1,2)+(1-pow(alpha1,2))\*pow(dot(n,viewDir),2)));  
 float ggxL = (2\*dot(n,lightDir))/(dot(n,lightDir)+ sqrt(pow(alpha1,2)+(1-pow(alpha1,2))\*pow(dot(n,lightDir),2)));  
 float ggx = ggxV\*ggxL/(4\*dot(n,lightDir)\*dot(n,viewDir));  
 float3 output = float3(CookTorrance,CookTorrance,ggx) ; // A placeholder, replace with your own code  
 return(float4(output,1));  
}

## Output







# HW4\_problem4:

As you can see there is not much of the difference between the two models. Even with changing roughness all the way there is verry little change happening. if one took more computation then other it might not be worth to use the other one unless we get better hardware that can handle it better then It is a possibility to enhance the game a little further.

# HW4\_part5:

## Code

float4 FragPBRTemplate(v2f input) : COLOR {  
 // Unity light position convention is:  
 // w = 0, directional light, with x y z pointing in opposite of light direction   
 // w = 1, point light, with x y z indicating position coordinates  
 float3 unNormLightDir = \_WorldSpaceLightPos0.xyz - input.positionWS \* \_WorldSpaceLightPos0.w;  
 float3 lightDir = normalize(unNormLightDir);  
  
 // renormalizing because the GPU's interpolator doesn't know this is a unit vector  
 float3 n = normalize(input.normalWS);  
  
 float3 viewDir = normalize(\_WorldSpaceCameraPos - input.positionWS);   
 float3 halfVector = normalize(viewDir + lightDir);  
 float NdotM = dot(n,halfVector);  
 float pi = 3.14159;  
  
 // roughness calculation  
  
 float alpha2 = \_Rough1 \* \_Rough1;  
 float alphaSquare2 = alpha2 \* alpha2;  
  
 // Blinn-Phong distributation D term  
  
 float blingEXP =(2/alphaSquare2) -2 ;  
 float blingDinominator = 1/(pi\*alphaSquare2);  
 float BlinnPhong = blingDinominator\*pow(dot(n,halfVector),blingEXP);  
 // GGX  
 float ggxV = (2\*dot(n,viewDir))/(dot(n,viewDir)+ sqrt(pow(alpha2,2)+(1-pow(alpha2,2))\*pow(dot(n,viewDir),2)));  
 float ggxL = (2\*dot(n,lightDir))/(dot(n,lightDir)+ sqrt(pow(alpha2,2)+(1-pow(alpha2,2))\*pow(dot(n,lightDir),2)));  
 float ggx = ggxV\*ggxL;  
  
 // fersenal  
 float3 f0 = {0.955008f,0.637427f, 0.538163f}; // i used copper  
 float3 fresnel = f0 + (1-f0)\*pow((1-dot(lightDir,halfVector)),5);  
  
 float3 BRDF = (BlinnPhong \* fresnel \* ggx)/(4\*dot(n,lightDir)\*dot(n,viewDir));  
 float3 output = \_LightColor0.rgb \* BRDF \* max(0, dot(n,lightDir)); // A placeholder, replace with your own code  
 return(float4(output,1));  
}

## Outout

