Goldsmiths, University of London

CM3045 – 3D Graphics & Animation

Peer-Graded Assignment: Texture Shader

Report by Hristo Stantchev

This report covers my submission for the “Texture Shader” Peer-Graded Assignment. This shader implements Normal Maps, Perlin Noise Effect, and Height Maps.

Code:

Vertex-to-fragment:

```
Interpolators vert(MeshData v)
{
    Interpolators o;
    o.uv = TRANSFORM_TEX(v.uv, _MainTex);
    //sample height map
    float height = tex2Dlod(_HeightMap, float4(v.uv,0,0)).x*2-1;

    v.vertex.xyz += (v.normal * (height * _HeightStr));

    o.vertex = UnityObjectToClipPos(v.vertex);
    o.normal = UnityObjectToWorldNormal(v.normal);
    o.wPos = mul(unity_ObjectToWorld, v.vertex);
    o.tangentWS = UnityObjectToWorldDir(v.tangentOS.xyz);
    o.bitangentWS = cross(o.normal, o.tangentWS) * (v.tangentOS.w * unity_WorldTransformParams.w);
    return o;
}
```

Fragment:

```
fixed4 frag(Interpolators i) : SV_Target
{
    //get normal map
    float3 tsNormal = UnpackNormal(tex2D(_NormalTex, i.uv));
    float3x3 mtxTangToWorld = {
        i.tangentWS.x, i.bitangentWS.x, i.normal.x,
        i.tangentWS.y, i.bitangentWS.y, i.normal.y,
        i.tangentWS.z, i.bitangentWS.z, i.normal.z
        };
    //get normal
    float3 norm = mul(mtxTangToWorld,tsNormal * _NormalStr);
    //Diffuse Lighting

    //get light dir
    float3 L = _WorldSpaceLightPos0.xyz;
    //get dot product
    float3 lambert = saturate(dot(norm, L));
    float3 DiffuseLight = lambert * _LightColor0.xyz;

    //Specular Lighting
    //get dir to eye
    float3 V = normalize(_WorldSpaceCameraPos - i.wPos);
    //get H
    float3 bisect = normalize(L + V);
    //get specular + Blinn Phong
    float3 specularLight = saturate(dot(bisect, norm)) * (lambert > 0);
    //get m
    float3 specularExp = exp2(_Gloss * 11) + 2;
    //get dot power of m
    specularLight = pow(specularLight, specularExp);
    //get specular light times intensity
    specularLight *= _LightColor0.xyz;
    specularLight *= _SpecularColor.xyz;
    // sample the texture
    fixed4 col = tex2D(_MainTex, i.uv);
    //implement color switch and lighting
    col *= float4(DiffuseLight * _BaseColor + specularLight,1.0);

    half4 noise = tex2D(_NoiseTex, i.uv);
    // animate the threshold noise value for transparency
    // using a sin function makes it oscillate
    float threshold = (sin(_Frequency*_Time.w) + 1)/2.0;
    if(noise.r < threshold) col = _BorderColor;

    return col ;
}
```

Task 1: Normal Map

Normal maps begin in the vertex shader. As normal maps dictate varied lighting for object relief, they require the vertex surface normal, UV, a transformed tangent to world space, as well as the bitangent in world space – a cross product of the world space normal and tangent vectors multiplied by the product of the object space tangent and the unity world transform parameters (range (-1,1); includes where the normal is facing).

In this code, heavily borrowed from Freya Holmér [1], the fragment shader implements the normal maps by unpacking it with the input UVs in mind, then by introducing a tangent to world space 3x3 matrix it creates the normal directions used in the Lambert lighting model set up by multiplying the matrix by the normal map times its intensity.

Task 2: Perlin Noise

This is very much taken from the code provided in this course. It Implements Perlin Noise in the fragment shader by sampling the Perlin Noise texture from WikiPedia [2]. This is then multiplied by the sine of frequency times the w portion of the elapsed time, added by 1, and divided by 2, which gives a smoother speed of animation. Some form of scaled or unscaled time is always used for animation.

Extension Task: Height Map

Heavily sampled from Freya Holmér as well, displacement happens in the vertex shader where the texture is sampled through tex2DLod (which takes a vector 4 input, however only the x value is needed, it is written to sidestep tex2D as non-applicable to vertex shaders). It is multiplied by 2 and subtracted 1 in order to remap it from (0, 1) to (-1, 1) as it allows for negative vertex displacement. Only its X value is taken into the "height" variable. The product of the input normals, the height, and its intensity, is added to the X, Y, Z values of the input vertex, which displaces the pixel alongside the x values of the height map, leading to vertex displacement. This is all handled by the GPU and is surprisingly inexpensive performance-wise.

References:

1. **Holmér, Freya, "Normal Maps, Tangent Space & IBL • Shaders for Game Devs [Part 3]", 26, 2021, YouTube:** https://www.youtube.com/watch?v=E4PHFnvMzFc
2. **WikiPedia Perlin Noise 1000x1000 texture, CC License:**
   By Lord Belbury - Own work, CC0,
   https://commons.wikimedia.org/w/index.php?curid=121585558
3. **Sphere textures – Tuytel, Rob, PolyHaven, CC0:**
   https://polyhaven.com/a/patterned_cobblestone_02