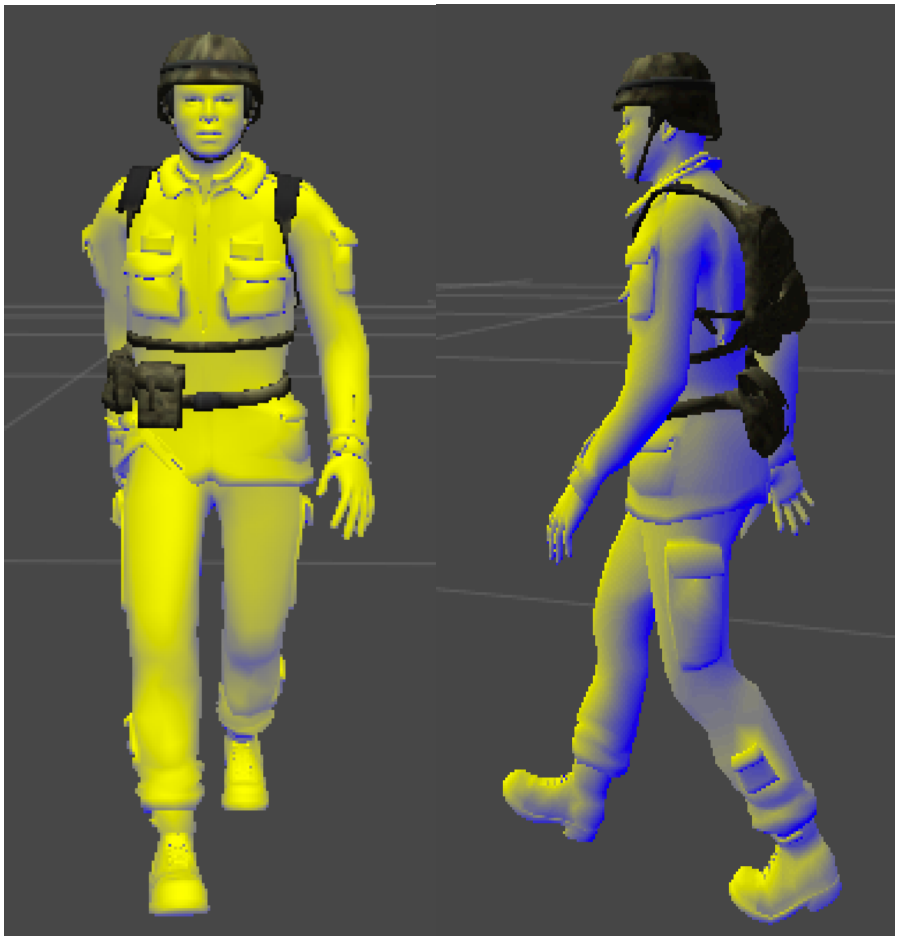
Abhishek Jain
Assignment 2
Gpu 14
ECE
[redacted]

}uestion 1

'art 1)

GPUXXSpecmapPixellit.shader

```
float4 frag_specmappixellit(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 lightDir = normalize(_WorldSpaceLightPos0.xyz - input.vWorldPos * _WorldSpaceLightPos0.w);
       float3 eyeDir = normalize(_WorldSpaceCameraPos.xyz - input.vWorldPos);
         float3 h = normalize(lightDir + eyeDir);
         // renormalizing because the GPU's interpolator doesn't know this is a unit vector
         float3 n = normalize(input.nWorld);
      //float3 diff_almost = 2*unity_LightColor0.rgb * max(0, dot(n, lightDir));
      float w = (1+(dot(n,lightDir)))/2;
      float3 diff_almost=lerp(float3(0,0,1),float3(1,1,0),w);
      float ndoth = max(0, dot(n, h));
      float3 spec_almost = 2*unity_LightColor0.rgb * _SpecColor.rgb * pow(ndoth, _Shininess*128.0);
      //float3 spec_almost =0;
       float4 base = tex2D(_BaseTex, input.tc);
       float3 output = (diff_almost + 2*0*UNITY_LIGHTMODEL_AMBIENT.rgb) * base.rgb
                    + 0*spec_almost.rgb * base.a;
       return(float4(output,1));
    }
```

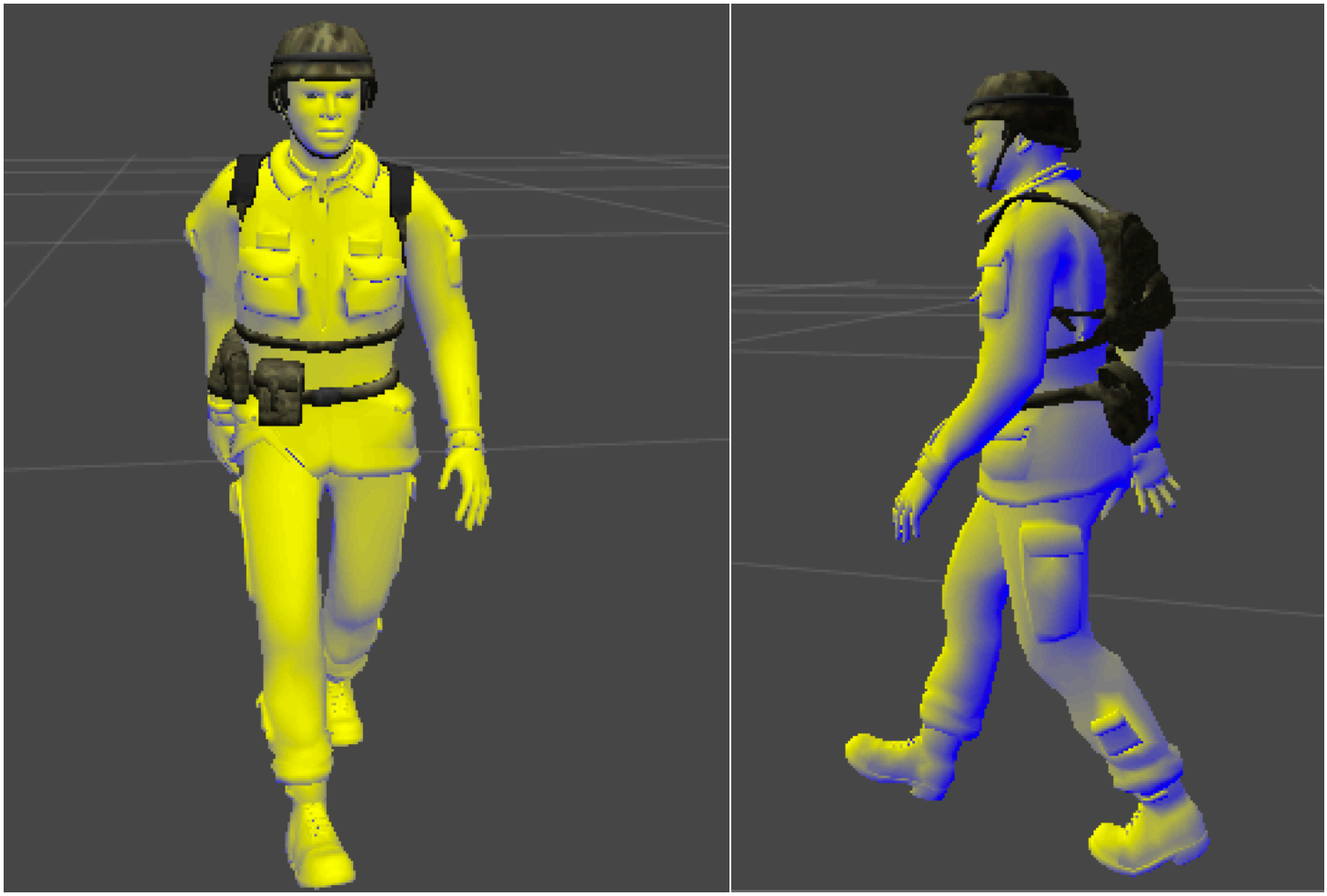


Part 2)

GPUXXSpecmapVertexLit.shader

```
v2f vert_specmapvertexlit(a2v input)  {
        v2f output;
        output.sv = mul (UNITY_MATRIX_MVP, input.v);

        float3 vWorldPos = mul (_Object2World, input.v).xyz;
        // To transform normals, we want to use the inverse transpose of upper left 3x3
        // Putting input.n in first argument is like doing trans((float3x3)_World2Object) * input.n;
        float3 nWorld = normalize(mul(input.n, (float3x3) _World2Object));

        // 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 lightDir = normalize(_WorldSpaceLightPos0.xyz - vWorldPos * _WorldSpaceLightPos0.w);
        float3 eyeDir = normalize(_WorldSpaceCameraPos.xyz - vWorldPos);
          float3 h = normalize(lightDir + eyeDir);
        //output.diff_almost = 2*unity_LightColor0.rgb * max(0, dot(nWorld, lightDir));
        float w = (1+(dot(nWorld,lightDir)))/2;
```

```
    output.diff_almost=lerp(float3(0,0,1),float3(1,1,0),w);
    float ndoth = max(0, dot(nWorld, h));
    output.spec_almost = 2*unity_LightColor0.rgb * _SpecColor.rgb * pow(ndoth, _Shininess*128.0);

    output.tc = TRANSFORM_TEX(input.tc, _BaseTex);
    return output;
}

 float4 frag_specmapvertexlit(v2f input) : COLOR {
     float4 base = tex2D(_BaseTex, input.tc);
     float3 output = (input.diff_almost + 2*0*UNITY_LIGHTMODEL_AMBIENT.rgb) * base.rgb
                   + 0*input.spec_almost.rgb * base.a;
      return(float4(output,1));
}

ENDCG
```



Question 2)

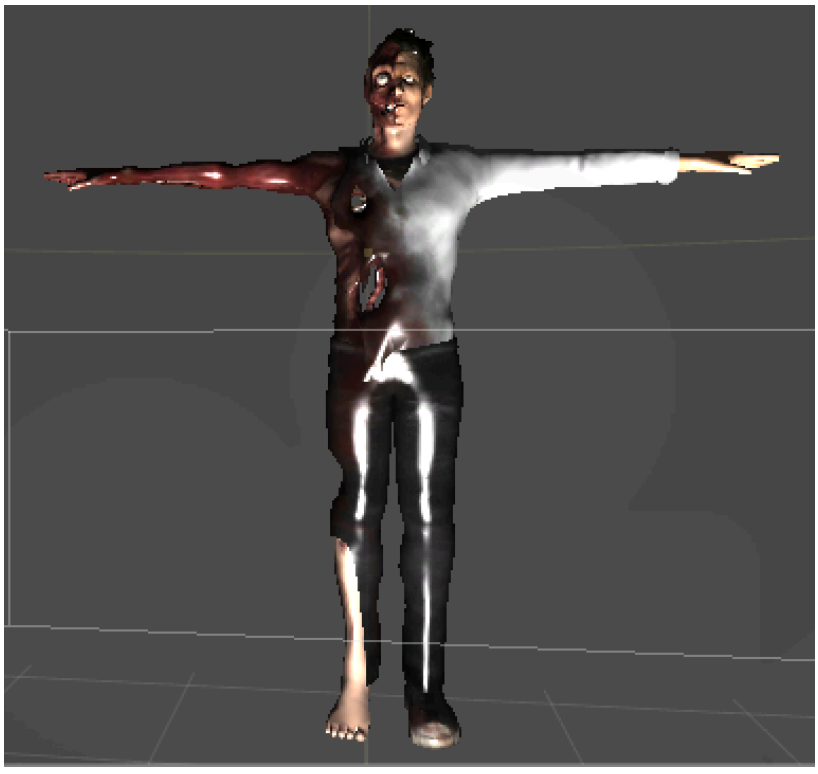
TexturedStuctTileCorrectly.shader

```
struct v2f {              // vertex to fragment
            float4 sv: SV_POSITION;
            float2 tc: TEXCOORD0;   // not same as TEXCOORD0 above
            float2 depthFactor:TEXCOORD1;
          };

        v2f vert_texturedstruct(a2v input) {
          v2f output;
          output.sv = mul(UNITY_MATRIX_MVP, input.v);
          // Make sure you TRANSFORM_TEX the vertex shader, not the fragment shader!
          float e=5;
          float s=-2;
          output.depthFactor = (max(0,min(1,(e-output.sv.y)/(e-s))),max(0,min(1,(e-output.sv.y)/(e-s))));
          output.tc = (TRANSFORM_TEX(input.tc, _BaseTex));
          return output;
        }

         float4 frag_texturedstruct(v2f input) : COLOR {
            return(tex2D(_BaseTex, input.tc)*input.depthFactor.y); }
```

Question 3)

GPUXXSpecNormMap.shader

```
v2f vert_specmappixellit(a2v input)  {
        v2f output;
        output.sv = mul(UNITY_MATRIX_MVP, input.v);
        output.vWorldPos = mul(_Object2World, input.v).xyz;
        // To transform normals, we want to use the inverse transpose of upper left 3x3
        // Putting input.n in first argument is like doing trans((float3x3)_World2Object) * input.n;
        output.nWorld = normalize(mul(input.n, (float3x3) _World2Object));
        output.tWorld = normalize(mul((float3x3) _Object2World, input.t.xyz));
        output.btWorld = normalize(cross(output.nWorld, output.tWorld)
                * input.t.w); // Flip tangents if needed (memory saving trick)
        float A=0.2;
        float B=1;
        float C=3;
        float D=0.02;
        float E=5;
        float F=6;
        input.tc.x+=A*sin(B*input.tc.y)*sin(C*_Time.x);
        input.tc.y+=D*sin(E*input.tc.x)*sin(F*_Time.x);
        output.bmap_tc = TRANSFORM_TEX(input.tc, _BaseTex);
        output.nmap_tc = TRANSFORM_TEX(input.tc, _NormalMap);
        return output;
    }
```



Question 4

GPUXXSpecNormalMap.shader

```
v2f vert_specmappixellit(a2v input)  {
        v2f output;
        float A=0.1;
        float B=50;
        input.v.x += A * input.n.x * (1+sin(B*_Time.x));
        input.v.y += A * input.n.y * (1+sin(B*_Time.x));
        input.v.z += A * input.n.z * (1+sin(B*_Time.x));

        output.sv = mul(UNITY_MATRIX_MVP, input.v);
        output.vWorldPos = mul(_Object2World, input.v).xyz;
        // To transform normals, we want to use the inverse transpose of upper left 3x3
        // Putting input.n in first argument is like doing trans((float3x3)_World2Object) * input.n;
        output.nWorld = normalize(mul(input.n, (float3x3) _World2Object));
        output.tWorld = normalize(mul((float3x3) _Object2World, input.t.xyz));
        output.btWorld = normalize(cross(output.nWorld, output.tWorld)
                * input.t.w); // Flip tangents if needed (memory saving trick)

        output.bmap_tc = TRANSFORM_TEX(input.tc, _BaseTex);
        output.nmap_tc = TRANSFORM_TEX(input.tc, _NormalMap);
        return output;
    }
```