CS 457 Final Project Proposal

Skin Shader

Annette Tongsak

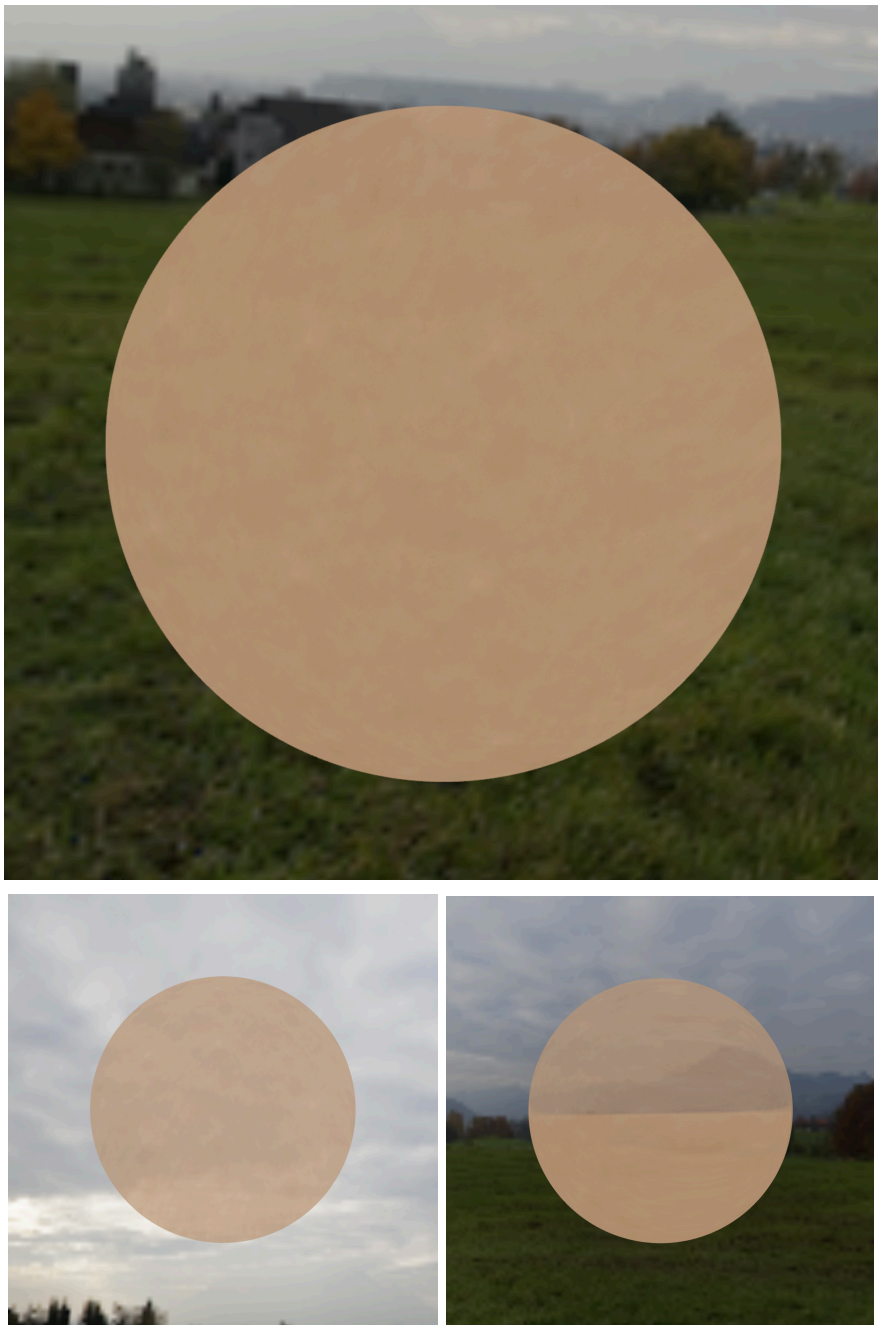
tongsaan@oregonstate.edu

Inspired by PDI/DreamWorks' novel skin shaders and groundbreaking implementation of subsurface scattering on Fiona's skin, I want to propose the implementation of a skin shader with OpenGL and GLSL. I plan to reference Curtis Beeson and Kevin Bjorke's documentation of their Skin in the "Dawn" Demo from Chapter 3 of *GPU Gems* and potentially the course notes from the SIGGRAPH 2017 course Physically Based Shading in Theory and Practice.

From a quick read of the *GPU Gems* chapter, this project will implement familiar topics like cube mapping and bump mapping alongside techniques like subsurface scattering and edge-effect lighting.

Alternatively, if my skin shader proves too challenging, I want to propose the rendering of water and its caustics based on Chapter 2 of *GPU Gems*. For this idea, I would like to animate falling leaves causing overlapping ripples on the water's surface using the equations from Project #3.

Project Execution - Video link



Additional resources used:

- HDRI to Cube Map
- Referred to this HDRShop guide in order to make diffuse and specular maps on the HDRI
- Cloud Convert to convert cube map files from PNG to BMP
- HDRI Panorama of a grassy plain

To create this display, I worked in this order:

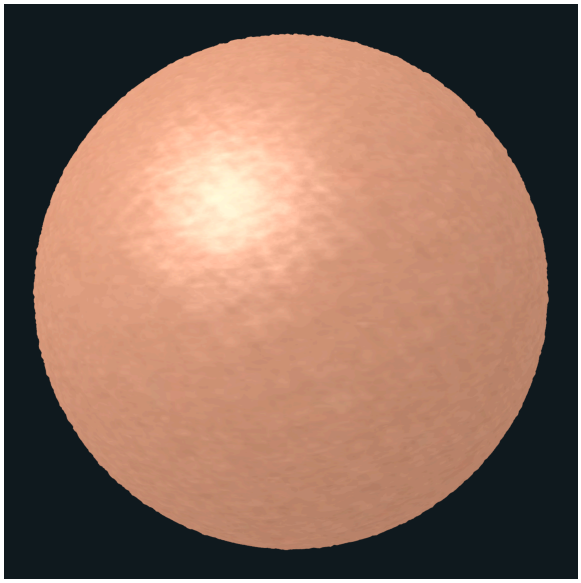
1. Found an HDRI panorama of a grassy plain and obtained the following cube maps:
    a. Environment Cube Map (unaltered)
        i. I passed the panorama into this HDRI to CubeMap Converter to obtain the cube map. I then converted the resulting PNGs to BMP using Cloud Convert.
    b. Diffuse Cube Map
        i. I used HDR Shop to perform a diffuse convolution on the HDRI (phong exponent = 1) and then passed the result into the HDRI to CubeMap Converter before converting the PNGs to BMP with Cloud Convert.
    c. Specular Cube Map
        i. I used HDR Shop to perform a specular convolution on the HDRI (phong exponent = 100) and then passed the result into the HDRI to CubeMap Converter before converting the PNGs to BMP with Cloud Convert.
2. Called the different cube maps, skin texture, and sphere in finalproj.glib
3. In pattern.vert, I calculate and output the texture coordinates, normal vector, vector from the point to the eye position, and model coordinates to be passed into pattern.frag.
4. In pattern.frag, I essentially:
    a. Normalize the normal vector and eye vector
    b. Sample skin texture with noise
    c. Sample noise textures for rotation angles
    d. Calculate rotation angles for the x- and y-axis
    e. Rotate and transform the normal vector
    f. Calculate the dot product between the eye vector and the transformed normal vector
    g. Calculate diffuse lighting by sampling the diffuse cube map and multiply it by skin color
    h. Retrieve the environment color behind the object to compute specular highlights
    i. Compute the reflection vector based on the eye direction, dot product between the eye direction and surface normal (calculated in f), and surface normal
    j. Calculate the final color by adding the diffuse and specular components
    k. Compute the haze factor based on the alpha channel of the reflected color
    l. Output the final color with haze as the fragment color

    This attempts to simulate diffuse reflection and specular highlights.

There was a lot of Dawn-specific code in the *GPU Gems* chapter that I spent a lot of time figuring out if I needed, like computing skinning for an animated model. In retrospect, I should have followed an implementation like this instead that renders human skin by having a combination of three layers of skin: diffuse/dermal, epidermal, and subdermal, though that would have involved more texture painting than I would have liked.

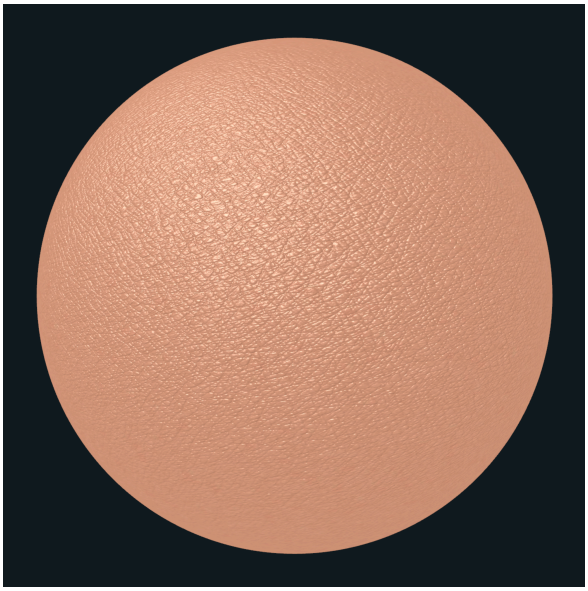
Out of frustration at my implementation looking like a translucent skin bubble, I tried out two simpler methods:

1. **Texture mapping and displacement mapping with a height map under per-fragment lighting**



I somewhat felt like texture mapping a solid skin texture was kind of cheating, but it already looked better than my first attempt. Using a height map to add displacement mapping to the sphere added a bit of texture, which was a small improvement.

2. **Texture mapping and normal mapping**



Using the code from the notes and the normal map that came with this skin texture gave the sphere a more detailed and realistic look.