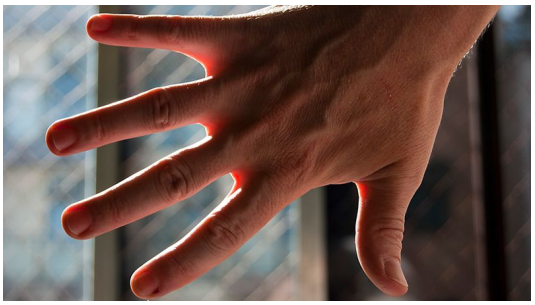
Bryan Arvizu

CMPM 163

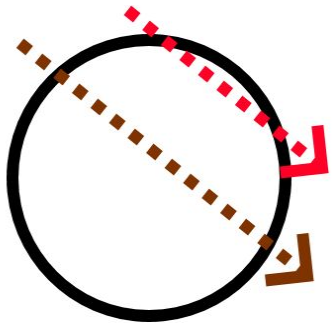
Part B: Subsurface Scattering

Subsurface scattering is the phenomena that describes the way light is absorbed by certain materials and reemitted to create a translucent effect. From what I understand, in the real world, when light rays enter a material that is not fully opaque, like human skin, they bounce around internally under the surface until they emerge at a random point outside of the object.



This creates an effect like the one shown above, where the parts of the object that are the thinnest glow much more brightly and appear more translucent than the parts of the object that have more depth.

Because a simulation of subsurface scattering is dependent on simulating light rays bouncing within an object, it is not likely possible to implement using Unity and HLSL shaders. Instead I imagine a crude version would have to be made that is able to be efficiently run in realtime. Perhaps this effect could simply color a pixel based on the distance to the other side of the object? But how would be measure depth in Unity?



After a bit of research, it appears that the simplest way to calculate depth in realtime is to create a depth map texture that stores the distance of each point in the object to the light. Using the texture, you can just sample the depth value via a simple texture lookup. I then imagine this value can be used to calculate how bright each pixel processed by the fragment shader should be.

Since light permeates through translucent objects, we would have to color the object's side opposite of the light, unlike phong shading, which merely colors it with an ambient color. The sides opposite of the light should be very easy to calculate by merely taking the dot product of the the light vector and the surface normal. The strength of the light added to the indirectly

lighted side should be dependent on the depth value, so that the thinner parts are much more bright than the deeper parts.

In retrospect, subsurface scattering seems to be an effect that cannot be recreated accurately in realtime rendering, and as such we have to use clever cheats to create effects efficient enough to look realtime and real enough to fool our eyes. This way of calculating realtime SSS does not look too complex, and looks to be fairly straightforward to implement. If any part would be trouble, I imagine it would be the depth texture.