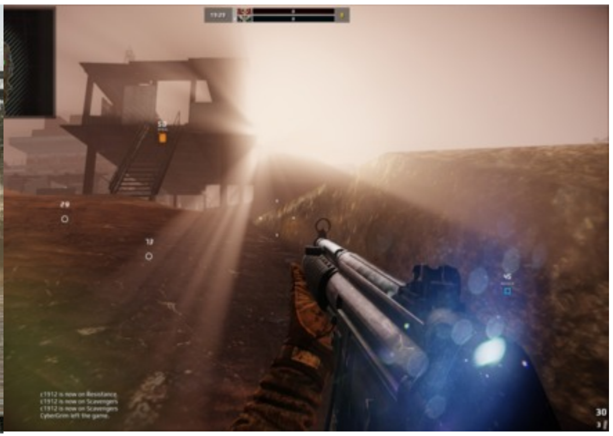
Homework 3C ~ Matthew Loyola (mjloyola@ucsc.edu)

Topic: Crepuscular Rays

Part 1:

Crepuscular rays, or more commonly known as sun rays or “god rays,” is a real-world lighting effect that makes it appear that rays of light are emanating from the origin of the sun. The effect is most noticeable in real life, video games, and other media when there is something obfuscating a portion of the light so that the light beams end up passing over or around the object(s).

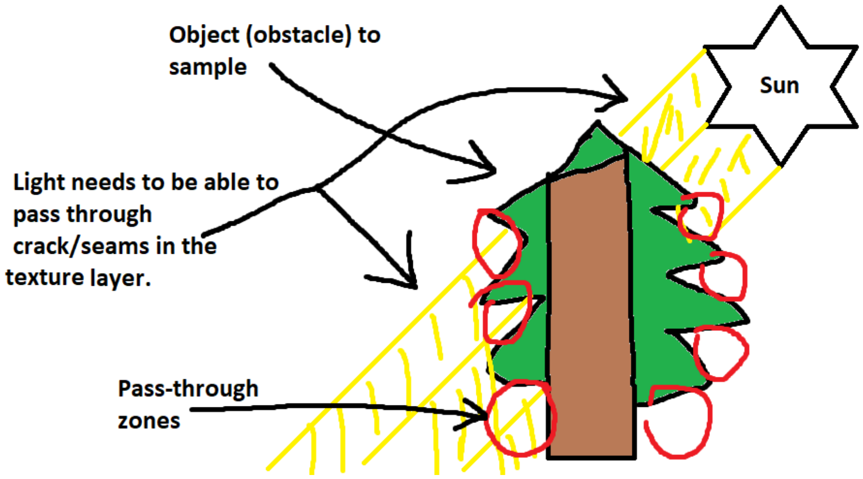


The image to the left is an example of this phenomenon in the real world and the right image is from a game attempting to replicate the effect using post-processing shaders.

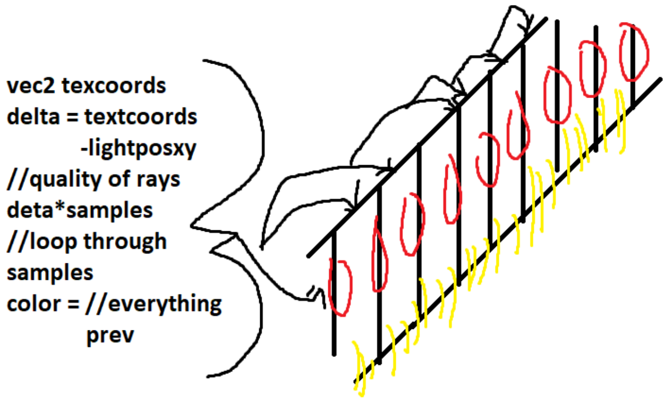
Similar to other lighting implementations we’ve done, the work on the god rays is done in the fragment shader which is then applied to a screen quad. This is similar to the implementation in homework 3c in that a plane would be the quad and used as the focal point of the shader effects. We’d need a way to calculate the obstacles the light would be interacting with since that is a major part of the ray effect. This would involve sampling the texture and storing the scattering pass texture data in two sampler2D functions; one for the user map and one for the color map. The light source(s) position would also need to be stored in a vec2 – most scenes, especially in video games, will typically only have one light source which will have the god ray effect attached to it, however it’s possible to compound more into the scene. Next, we would have fields for decay, exposure, density, and weight that can be tweaked to get the desired effect of the rays. The overall quality of the rays can be set as an integer controlling the number of samples. This is very important in games and process-intensive programs in order to provide the person running the program with the ability to lower the quality to improve performance and still enjoy the scene. Then we’d need to get the texture coordinates, delta texture coordinates, and divide by the number of samples (mentioned previously). Finally, before assigning the gl_FragColor, you’d have a simple for-loop to increment the textcoord array and grab out the samples of the texture. The look would also need to take the sample and add it to the color variable which will in the end be set to the frag color.

Below is another example from a web tutorial which discusses implementing the effect in GLSL code.

I also drew an (extremely crude) representation of what needs to be done to apply this effect on a random object – in this case a tree since forests with god rays tend to produce the most amazing effects. Our group is also looking into using landscape in the project so casting god rays through tree leaves would one desired effect.


Object (obstacle) to sample
Sun
Light needs to be able to pass through crack/seams in the texture layer.
Pass-through zones

Lastly, a form of pseudo code based on my research into the effect. This isn't completely everything that is necessary but is the basic idea behind what needs to be done in the fragment/pixel shader.



```
vec2 texcoords
delta = textcoords
        -lightposxy
//quality of rays
deta*samples
//loop through
samples
color = //everything
        prev
```

Part 2:

Group Members: Matt Loyola, Jonathan Chuang, Eisaku Imura, Arvind Vallabha

We plan to coordinate online/after class in the next week to solidify our project plans. One initial idea our group had was to utilize data provided online on the specifications of buildings and landmarks of the university campus in order to generate heighmaps and create a 3D scene of the school similar to Google Street View. We'd incorporate things like the water from the nearby beach and lighting of trees, landscapes, and buildings. On the surface this seems very ambitious given the time constraints so if we choose to work on this idea it would definitely need to be scaled down to only sample a bit of everything.