Ambient Occlusion and Various Visual Effects

Team Psyko Jr.

Jacob Daniels-Flechtner
UCSC
jatdanie@ucsc.edu

Junhao Su
UCSC
jusu@ucsc.edu

Oskar Alfaro
UCSC
ogalfaro@ucsc.edu

Jason Chen
UCSC
jadchen@ucsc.edu

Taylor Infuso
UCSC
tinfuso@ucsc.edu

ABSTRACT

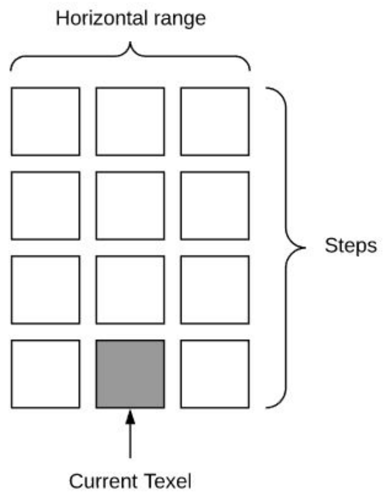
Our creative vision for this project was to create a visual effect that would allow for more realistic-looking lighting, similar to indirect lighting but with greater efficiency. After quickly learning the number of computations and raycasts required for realistic-looking indirect lighting, we decided to turn to ambient occlusion for an efficient method of ambient lighting. We wanted to set up a scene populated with objects in order to better showcase the lighting effect. We also included other particle/visual effects which will be discussed later on, such as a mirage effect, heat waves, and fire/wind/dust particles that react to music.

Ambient occlusion, as we understand it, is a cheap method of calculating ambient lighting by detecting which areas of a scene are less likely to receive light from the environment. In our example, the light source is assumed to be the sky, so any surfaces that are facing downwards are assumed to be darker. A simpler interpretation is to assume that if there is a noticeable overhang above a specific point in the world space, that point will a noticeable amount of shading. This also applies to areas that are inside corners or wedges where it is difficult for light to reach.

We wanted to avoid creating an individual shader for each separate object in the scene, so only one shader simulates the ambient occlusion effect. This is particularly useful for maintaining the default lighting of the scene, and no extra lighting calculations are involved. Since the effect is only applied to the camera, it can be applied to various different scenes without much tweaking.

SCREEN SPACE AMBIENT OCCLUSION

Our camera shader consists of a screen texture and a depth buffer. For each texel on the screen, the depth of the current texel is compared to the texels above it and around it within a certain range. If the depth of a queried texel is below that of the root texel, we know that the queried texel is closer to the camera, and may indicate an overhang.



Each time a texel is queried above the current texel that seems to create an overhang, we increase a shadow variable. The shadow variable is used to subtract the tint of the texel color so it appears darker.

However, if the queried texel is too far away from the root texel, we assume that it is either sufficiently in the foreground or background that it is irrelevant to the root texel. Consider the following line of pseudocode.

For each queried texel that is closer to the camera than the root texel, do:

Darkeness += constant / (1 + difference of depth);

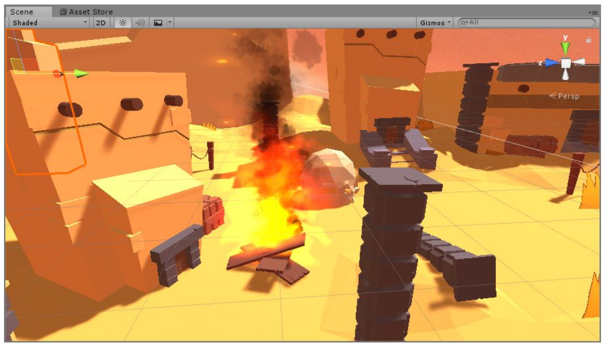
We also included a "step skip" variable so that more texels can be queried across the screen, making the process more efficient. This simply queries every X texels so that we sample a much larger space while keeping our code efficient.

For a more in depth explanation about screen space ambient occlusion and its implementation visit the URL https://en.wikipedia.org/wiki/Screen_space_ambient_occlusion. It contains said information for sampling depth calculation per pixel which is useful for understanding the visual effect. For information about implementing camera depth textures: https://docs.unity3d.com/Manual/SL-CameraDepthTexture.html.

Fire Effect

We made a fire effect with several particle systems. Four particle systems were used to generate foundation fire effect, glow fire effect, bloom fire effect and fire sparks. To represent the proper flame layers of a real fire, we had to correctly layer each particle systems. We also changed the values and implemented a texture for each particle system.

Image of the fire effect:



We made this effect based on this tutorial on YouTube: https://www.youtube.com/watch?v=5Mw6NpSEb2o&t=799s

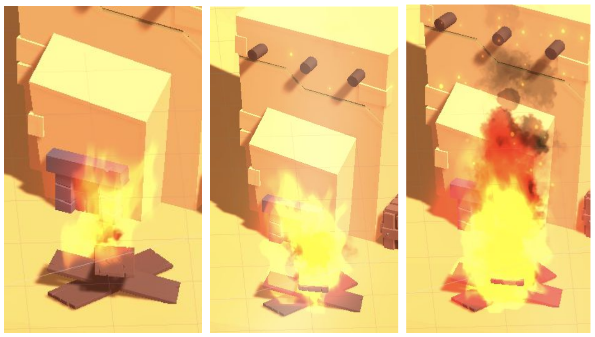
Dancing Fire Effect

We made all the fire in the scene dance with background music. We first collected music's spectrum data using the peerAudio script provided in previous lab, then calculated magnitude based on that data per frame.

Magnitude is between 0 and 1, and fire effect emission has to around 10 to be obvious. Therefore, we set newMagnitude to:

newMagnitude = (int)((magnitude - 0.5f) * 200);

Then we tweak this variable for each part of the shadow, so that the fire goes out when the music is quite, and dark smoke comes out when heavy base appears.



A similar effect was applied to the wind and sand particles surrounding the scene.

Mirage Effect

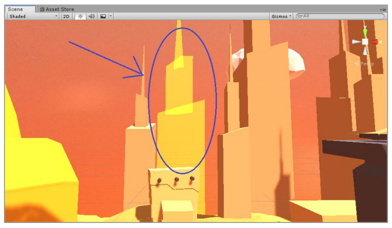
There is a mirage effect for two background buildings which help the scene look like a desert where mirages appear.

We add several properties in the shader code such as "Transparency", "Amplitude", "Speed", etc. These input values are based on a sine wave over time.

The following is pseudocode that used in the vertex function to create this effect:

v.vertex.x += sin(_Time.y * _Speed + v.vertex.y * _Amplitude) * _Distance * _Amount;

This is an image of the mirage which is constantly changing its shape without using animation. Its properties can be found and changed in "Unlit_Mirage" material.



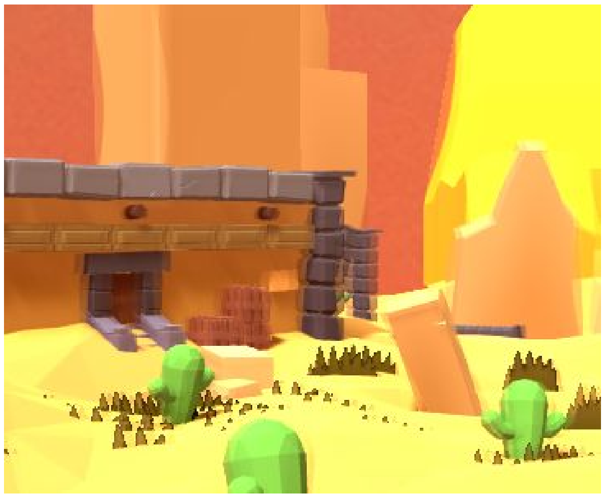
Heat Wave Effect

An additional visual effect was added to the camera shader that allows for the appearance of heat waves. This effect is done by finding an offset uv coordinate from the input uv based on a cosine wave and combining it with the default value. This effect is made to be more obvious the further away an object is.

The following is pseudocode that can be used to accomplish this effect:

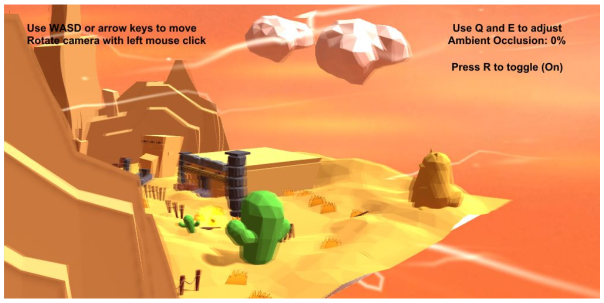
```
Shifted Color = tex2D(_MainTex, uv + cos(time + uv.y));

Return Lerp(Base Color, Shifted Color, Distance);
```

This is an image of the heat haze visual effect, showcasing how it visibly bends the screen texture. The effect changes over time, causing the screen to wobble.



Wind Effect

There is a wind effect in the scene. Includes noise, fading in and out, random size and speed, and noise that increases with the volume in the scene.


Use WASD or arrow keys to move
Rotate camera with left mouse click
Use Q and E to adjust
Ambient Occlusion: 0%
Press R to toggle (On)

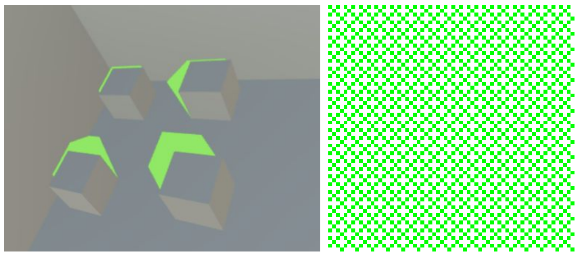
Other Attempted Effects (not in build)

Box Blur

During the development of Ambient Occlusion, we encountered that shadows were being layered in a rough manner. We wanted a gradual change of the density of shadow, so we attempted to blur the shadows in order to make them appear more natural. However, since we were passing the entire camera screen as a texture, the shader could not accurately detect shadows. We tried to blur all of the seemingly dark areas, but it also detected areas with darker texture patterns. We spent a lot of time developing the next method to better detect where each shadow is located.

Custom Shadows

This method utilizes a green screen technique commonly used in the movie industry. We calculated shadows by casting multiple rays from the light source passing every vertex of each object that casts a shadow to generate a hit on every platform that receives a shadow. Then we repositioned and enclosed hit points to generate shadow objects, and applied a green grid texture to each generated shadow.

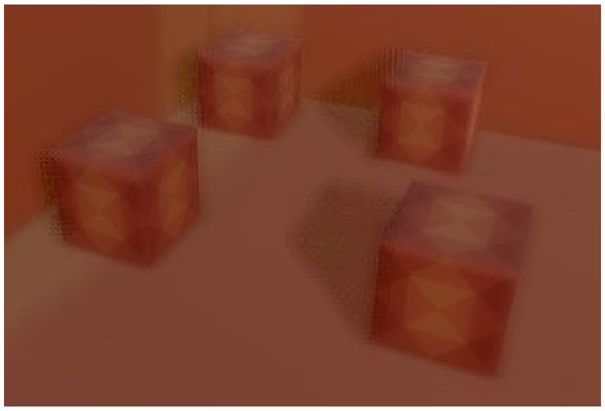


Unfortunately, we discarded this idea because it caused our ambient occlusion code to become inefficient and seemed to severely dampen our initial visual effects.

(Shadows are solid green in this image)

This method seemed to work at first, but we later realized that

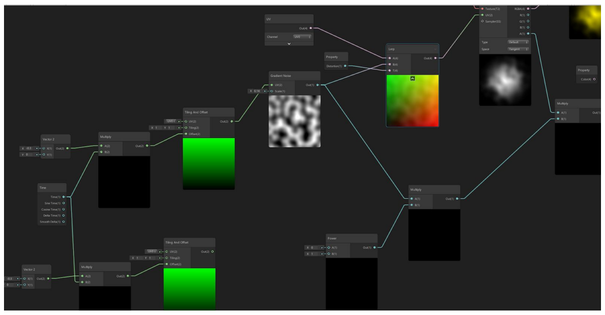
we could not completely remove the green grid. We ended up with very blurred, low resolution shadows that had a noticeable amount of green grid left on them, which was not the intended result. Our means of removing this grid was to recolor each pixel that had the same green shade as the green grid. Precise color checking was difficult so we used smoothstep. At the very best, we made the image below which still retains some artifacts.

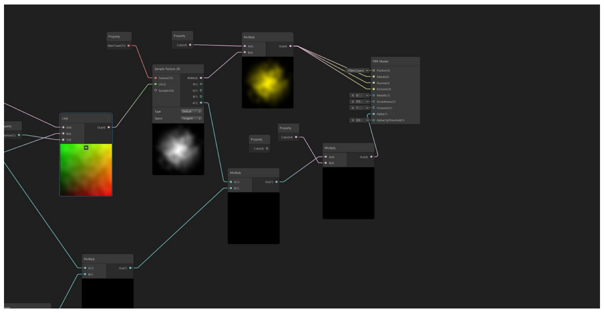


Shader Graph Dust Effect

We currently have a dust effect that is pretty standard. It decreases in opacity over time, has noise, and uses Unity's legacy shaders. However, we initially were going to use an effect made from shader graphs. Shader graphs, however, required the use of the lightweight render pipeline which was incompatible with our ambient occlusion effect.

The effect was made with offset that would increase over time, be run through a gradient noise filter, and then multiplied to create an effect that appears like yellow fog.





Use WASD or arrow keys to move
Rotate camera with left mouse click
Use Q and E to adjust Ambient Occlusion: 0%
Press R to toggle SSAO

ACKNOWLEDGMENTS

Most of the models were made by Oskar Alfaro others were made by Jacob Daniels-Flechtner. Desert music was created by Daniel Qi, an external collaborator.

REFERENCES

[1]
https://en.wikipedia.org/wiki/Screen_space_ambient_occlusion

[2]
https://docs.unity3d.com/Manual/SL-CameraDepthTexture.html

[3]
https://www.youtube.com/watch?v=5Mw6NpSEb2o&t=799s