Assignment 3

TDT4195 – Visual Computing Fundamentals

Sigurd Totland — MTTK

October 7, 2019

Task 1

a)

We begin by simply creating a `Mesh` object from `lunarsurface.obj` and sending the vertex, colour, and index vectors into the `createVAO` function. As expected, the lunar surface renders correctly, albeit without any colors at this time, so it is completely white as shown in figure 1 below.

Figure 1: Lunar surface without any coloring

c)

In this task and the previous, we add a third VBO to hold the normals and pass these values through the vertex shader into the fragment shader. Setting the fragment color equal to the x, y, and z components of each normal vector yields a predominantly green surface as shown in figure 2 below.

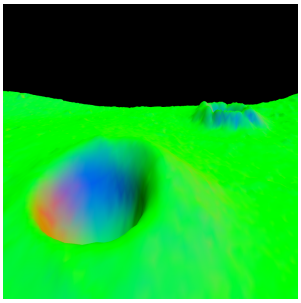

Figure 2: Lunar surface colored by directly inserting the surface normal values

We notice how green color corresponds to upward pointing normals, whereas red and blue are for more sideways surfaces.

d)

Inserting the Lambertian illumination model,

$$\text{color} * \max(0, \hat{\mathbf{n}} \cdot (-\hat{\mathbf{l}})), \tag{1}$$

where $\hat{\mathbf{n}}$ is the surface normal and $\hat{\mathbf{l}}$ is the normalized lighting direction the surface becomes a *Lambertian surface*. Coloring the surface a pure white results in the following (quite convincing) scene, clearly showing that the surface is indeed lit. (Figure 3).

Figure 3: Lunar surface with Lambertian shading

Task 2

c)

We define the scene graph illustrated in figure 4.

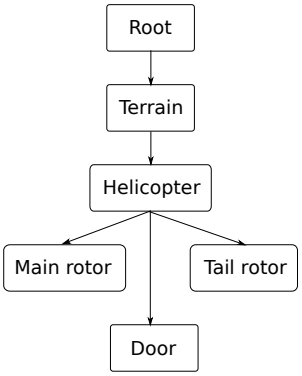

Figure 4: Scene graph

This graph, when drawn using a basic pre-order recursive draw method yields the physics defying helicopter shown in figure 5 below.

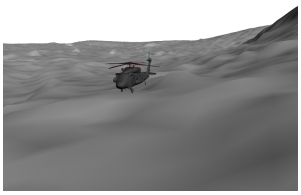

Figure 5: A helicopter on the moon

Task 5

a)

In the following figures, the helicopter has been rotated a quarter turn, but clearly does not obey the laws of light, and the dark side of the helicopter stays dark even though it is supposedly lit from the correct direction.

Figure 6: The bright side of the helicopter

Figure 7: The dark side of the helicopter

The issue becomes even clearer when rendering multiple helicopters simultaneously as faces that point the same way get completely different shading. Observe how the two leftmost helicopters in figure 8 both have surfaces facing the camera, but one is lit and one is dark.

Figure 8: The two leftmost helicopters have very different shading

Task 7

b) Controllable Helicopter

We add another helicopter above the five figure-8 helicopters that we wish to control. The dynamics of this helicopter is a grossly simplified kind of helicopter dynamics, with no special motion whatsoever. You can however control it just fine using wasd.

w	forward
a	rotate left
s	backward
d	rotate right
space	up
shift	down
c	toggle control

c) Chase Camera

Implementing the controllable helicopter without also implementing a chase camera was simply not acceptable, and so it had to be done. As with the helicopter, the chase camera is also done in a simple barely-work fashion. The camera needs some sort of control law to smoothly bring it to its target and the major simplification in this case is that control of each coordinate, x, y and z are decoupled, i.e. x-control depends only on x, y depends only on y and so on. The control laws we then opted for were the following

$$x_{k+1} = x_k - v(x_k - \text{sgn}(x_k - r_k)R - r_k),$$

where x_k and r_k respectively denote the given coordinate of the camera and the reference (the helicopter), v denotes the camera "speed" gain and R denotes the wanted chase radius. This law is used for both the x and the z coordinate, but for y we omit the sign function to ensure that the helicopter is always chased from above. The keen eye will notice that this doesn't really keep the camera in a circle around its target, but rather in a square-like shape, in fact it will move according to figure 9 below.

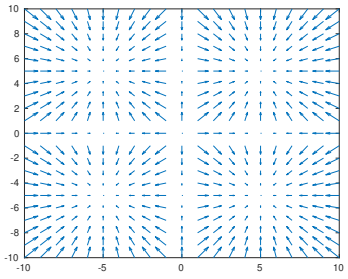

Figure 9: Quiver plot of chase camera dynamics

We see that the camera will tend towards one of four stable points around the helicopter.

Finally, once the camera motion was in place, making the camera look at its target was as simple as using the opengl `lookAt` function.

Use the `c`-key to toggle between chase camera/helicopter control and the regular free-moving camera.

f)

You wanted to be amazed, so here you go. I proudly present *deepfry-shading*, or in other words, what happens when the *uninitialized* output color vector is used to color the scene.

Figure 10: Deepfried Helicopter