Parker Eischen

23, May 2019

Computer Graphics

Uppsala University

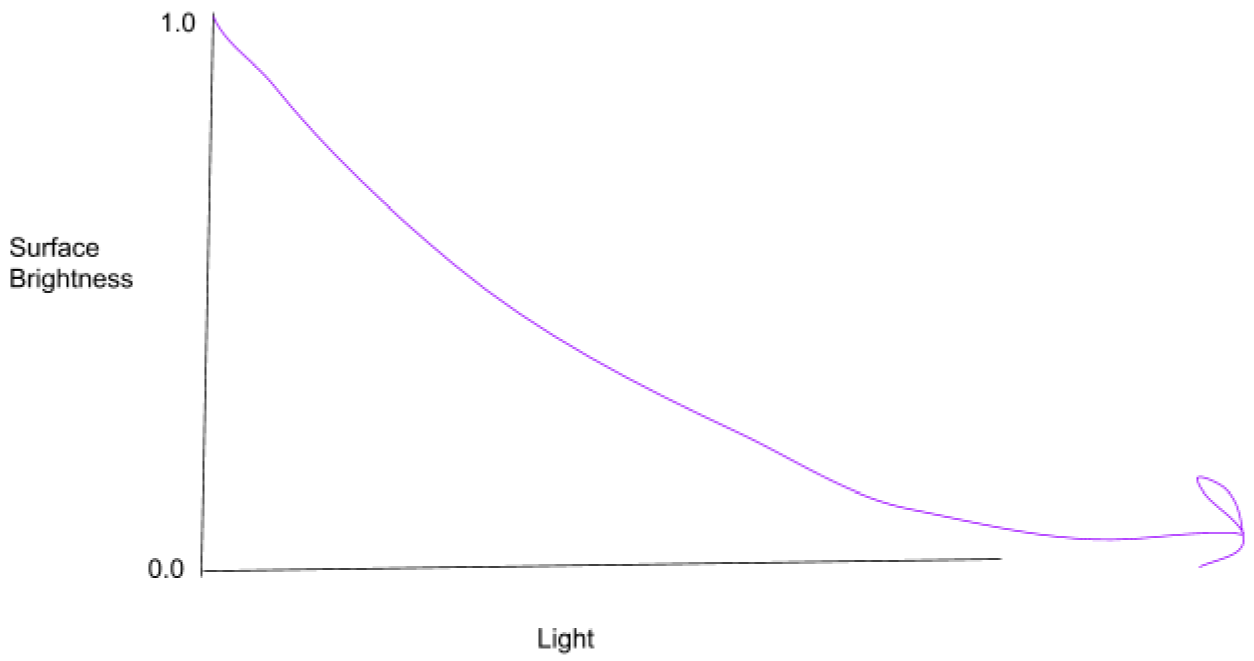
Computer Graphics Final Project Report

For my final project, I wanted to explore an implementation of Toon/Cel shading. This style of shading always interested me since I played games like *Borderlands* and the Telltale Games. I used to assume that this style was attached to all the models by being drawn on, similar to comic books. After spending some time researching how the effect is done, I created a successful implementation.

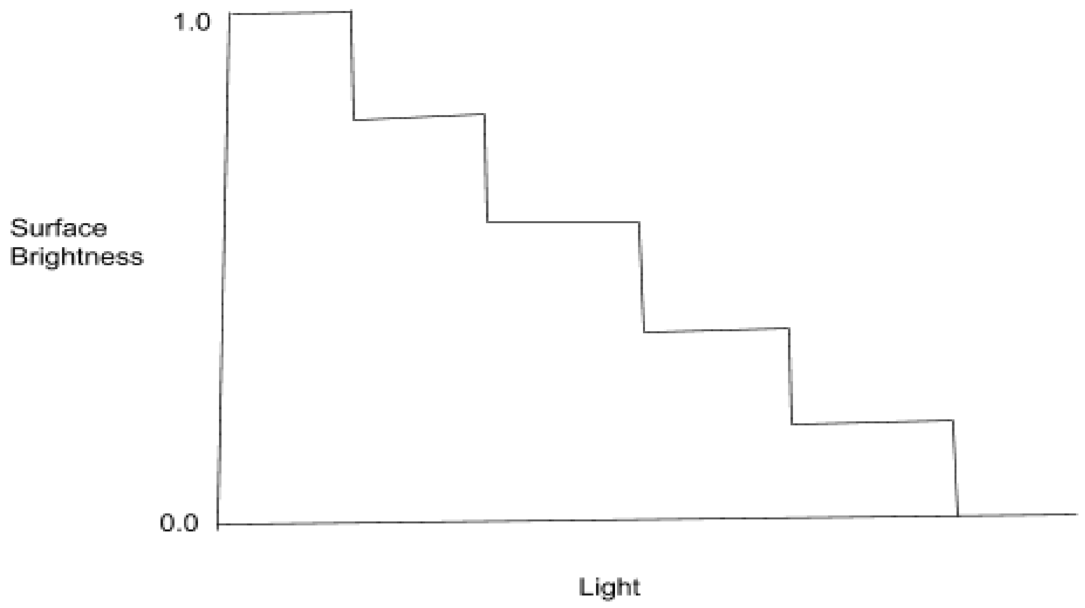
In order to implement Toon Shading, we have to first find a way to detect the edges of our model. This is done through Lambert's cosine law. Lambert's cosine law says that the amount of reflected light is proportional with the cosine(dot product) of the angle between the normal and incident vector (angle of incidence)[1]. To simplify, light reflected directly to the viewer should be brighter than light reflected at other angles. This is helpful to detect edges by removing all light to areas of the shape that have an angle greater than 90*. This will blacken all the edges by removing diffuse light and keep the face of the model reflective. I also added a slider to allow adjustments to the angle in which the light is omitted, to demonstrate how the edges can increase/shrink. Now, the final step is to actually add the Toon/Cel effect. For most

[1] "Lighting in Vertex and Fragment Shader." *in2gpu*, 8 Mar. 2016, in2gpu.com/2014/06/19/lighting-vertex-fragment-shader/.

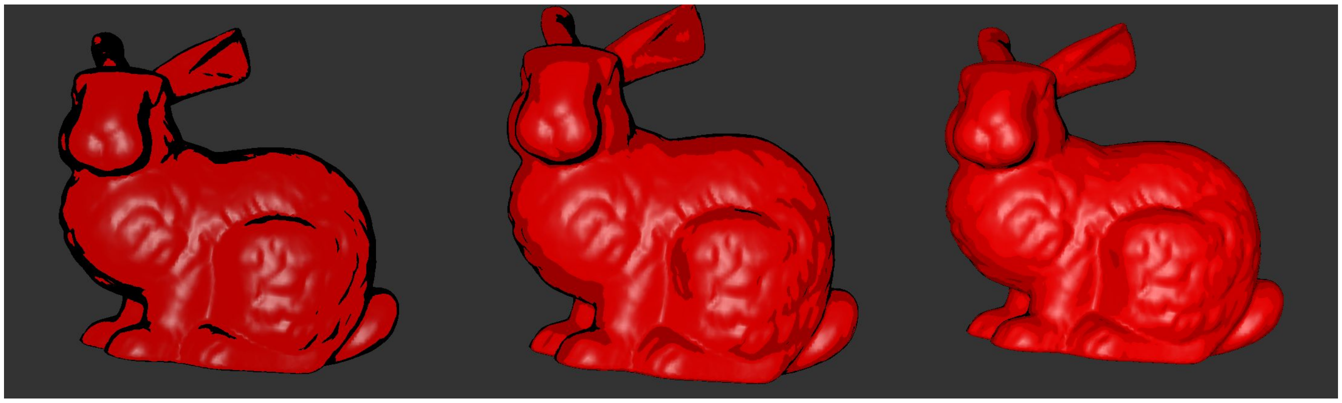
shadering, the model smoothly becomes darker with the less light it receives. This can be demonstrated by a downward gradient.



For Toon Shading, we need to implement levels to this function, where there is a sharp transition of light between each level. Similar to this

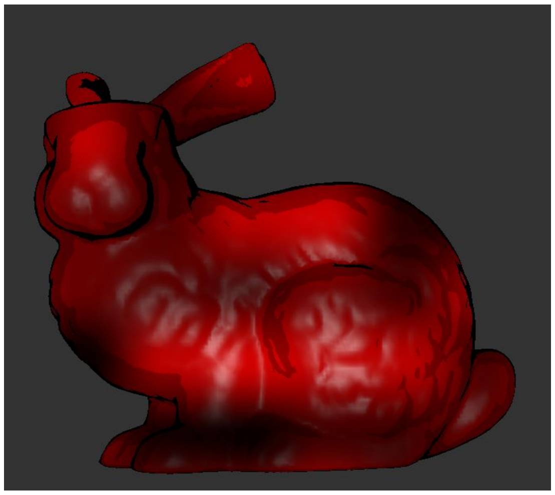


This effect is done by calculating the brightness of each point and capping it to the brightness of the lower level. After we receive the brightness for each of the points in the fragment shader, we can multiply this variable to the front of our Blinn-Phong calculation to achieve the final effect.

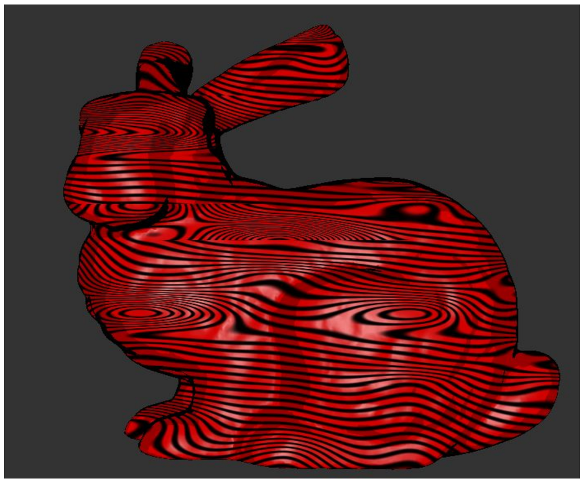


1 Levels 3 Levels 5 Levels

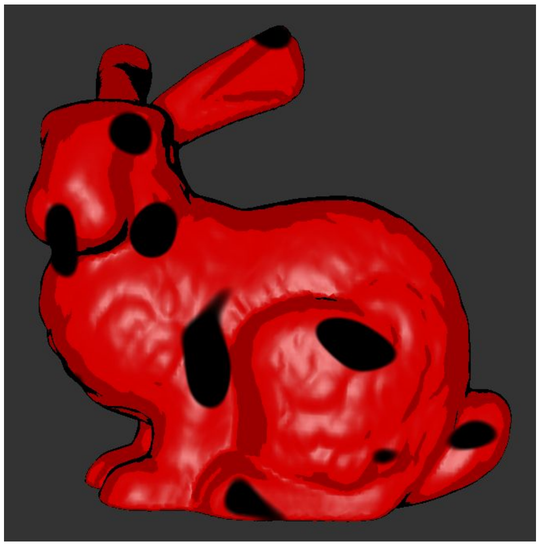
To texture my model, I wanted to stray away from cube mapping like in the previous assignment. I dug around the lecture slides and was interested in implementing some form of perlin noise texturing. Perlin noise works by generating a random number over time that is closely related to its neighbors. This creates an effect that is random but smooth. It resembles a more natural way of generation compared to perfectly random numbers. This solves generation in a 1D scale, yet for a 2D scale, like our window, we have to interpolate between the four corners of a plane. A random value is chosen for these 4 points and generates a vec3 with that value. The noise algorithm creates a texture of different squares of varying shades of white and black for each point in the vertex shader. To add this texture with cel shading, I added the float value to the final result of the color that was cel shaded. Producing a model that looks like this.



I wanted to experiment more with the perlin noise and looked at other implementations of it online. Most were slight variations in which random parts of the model were put into the noise algorithm. For example, if we randomly rotate the area where straight lines were rendered, a swirly effect would be produced.



Or an effect that moves around randomly based on the time, similar to a lava lamp



I never realized how complex generated graphics could be. Before this class I had always assumed it was all by created by hand or animation, never based on algorithms and matrices. If I had more time on this project, I would explore world generation and shading environments, rather than individual models. I am currently satisfied with my result, however I would like to do more with it in the future.