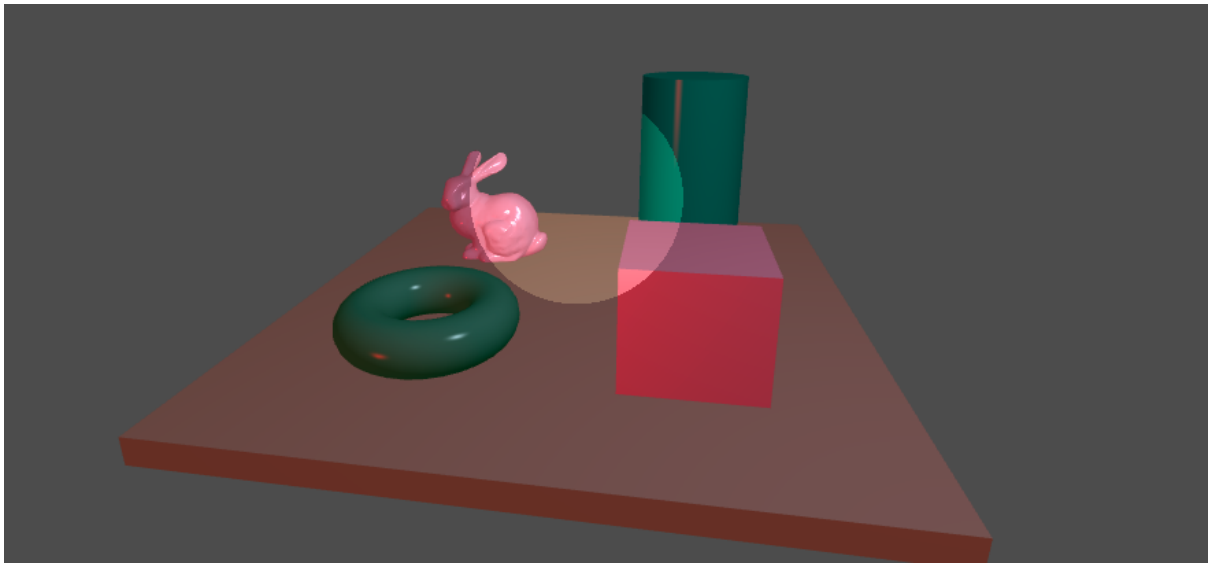

Relatório 3º Projeto

Modelo de iluminação de Phong com sombreamento de Phong

Tipos de dados

Javascript

Para cada objeto da cena, temos um struct que indica os valores **ambient**, **diffuse, specular** e **shininess** do seu material.

Temos um array com as informações de cada luz (no nosso caso só temos 3 luzes, a pontual, a direcional e a spotlight)

Vale realçar que decidimos que as luzes que não são o spotlight, têm o **cutoff = -1.0** e a **aperture = 180.0**.

Temos um objeto GUI onde tem todas as opçoes do GUI que aparece no programa, separado em pastas.

Shaders

Duas estruturas, uma para as informações das luzes *LightInfo* e outra para as informações dos materiais *MaterialInfo*.

A estrutura dos materias é “única” no shader, enquanto para as informações das luzes temos um array de *LightInfo* de tamanho igual ao número de luzes no programa.

Se tivessemos mais luzes

No Javascript

Começamos por criar três botões no dat.GUI que permitem criar cada tipo de luz. Acima do botão estariam opçoes para alterar as propriedades da nova luz. Este botão iria fazer push de um novo objeto para o array *lightInfo*. O método drawLight seria invocado num *for loop* com a dimensão do array. Se o array alcançar uma dimenção = 8, então impedir que se adicionem novas luzes.

Nos shaders

Os nossos shaders já estão prontos para receber um maior número de luzes (Máximo 8 luzes).

Começamos por verificar se o *cutoff* da luz atual é igual a -1. Se **não** fôr então trata-se de uma luz spotlight.

De seguida, calculamos o ângulo formado pelo vetor **L** e -**axis** e verificamos se esse ângulo é menor que **aperture**, para provocar o efeito "cone”**.**

Por fim, calculamos a atenuação da intensidade*,*

cuttoffFactor = $cos(\alpha)^n$

onde α é o tal ângulo anteriormente calculado e n é o **cutoff,** e multiplica-se essa tal atenuação pela componente *diffuse* e *specular* do *FragColor*.

Desafio

O desafio envolvia a rotação da camera. Para resolvermos, vimo-nos obrigados a arranjar uma forma de transformar as coordenadas do rato em angulo, ou, pelo menos, aumentar o angulo através das coordenadas do rato. A solução mais óbvia

e que acabamos por seguir foi transformar o movimento do rato em vetor. Cada vez que se clica (e se mantem a clicar) no rato, recolhemos as coordenadas deste e calculamos a distância às coordenadas de quando o movemos. O resultado desta operação é multiplicado por uma velocidade adaptada por nós e adicionada ao angulo atual de rotação. Logo a seguir a ser adicionada, recolhemos novamente a posição do rato. Caso não fizéssemos este segundo passo, quanto mais afastássemos o rato da primeira posição, mais rápido a camera rodaria, o que não é ideal.

Com o angulo calculado, limitava-nos rodar a camera em si. Porém, depois de alguns testes, reparamos que ao aplicar rotações em X, Y e Z alteravam de acordo com a posição da camera. Por exemplo, quando a camera está posicionada atrás do eixo X, rodava ao contrário, ou quando não estava alinhada com os eixos X e Z, deixava a cena ligeiramente inclinada. Para resolver estes problemas, adaptamos o tipo de rotação (em X ou Z) e o sinal do angulo, de acordo com a posição atual da camera e o angulo que faz com o eixo Y.

Comentários extra

Para distinguir luzes direcionais de pontuais, definimos o ‘w’ da posição da luz diretamente no javacript em vez de o fazer no fragment shader.