Relatório
Computação Gráfica

Grupo CG13
Daniel Batista – fc52773
Rafael Abrantes – fc52751
Ricardo Gonçalves – fc52765

Índice

1 – Modificações na cena original

Adição de novos elementos à cena

Para criar o cubo e a esfera, criamos um buffer para os vértices, outro para as normais, outro para as cores e outro para os índices, no initBuffers. Para criar a pirâmide temos apenas três buffers em vez de quatro, um para as posições, outro para as normais e outro para as cores. Para cada objeto, todos os buffers desse objeto são guardados numa posição de um array. (linhas 114 – 429).

Tendo os buffers criados, falta desenhar os objetos. Para criar tanto a esfera como o cubo, cria-se uma nova modelViewMatrix e move-se para o lado (translate) para os objetos não ficarem sobrepostos, caso queiramos rotações também é sobre esta nova modelViewMatrix que vai ser aplicada. De seguida cria-se uma nova matriz para a normal e inverte-se e faz-se a transposta da mesma.

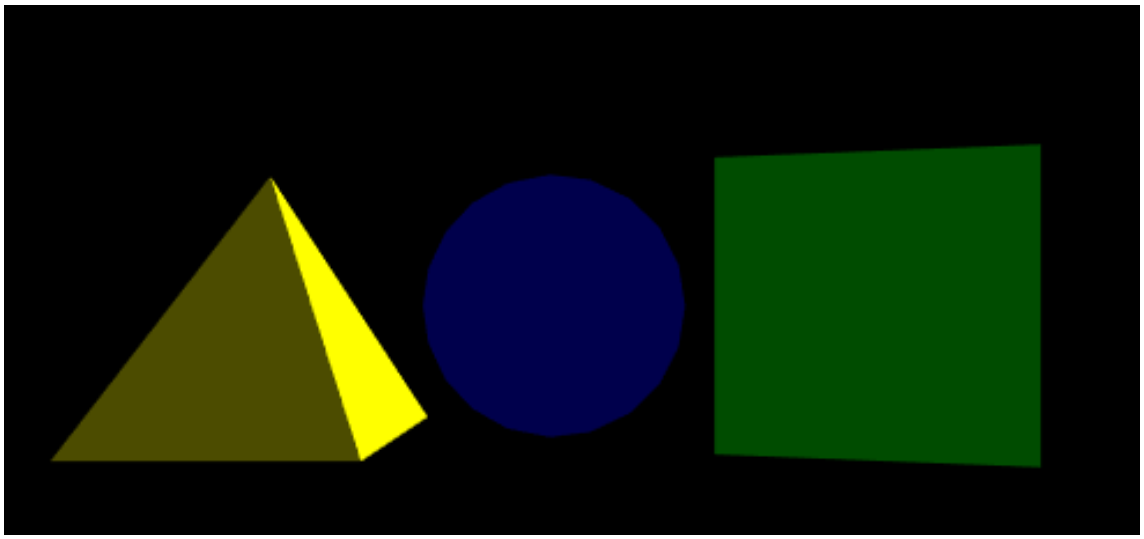
Após isto é necessário dizer ao WebGL como vai buscar as posições dos vértices para conseguir desenhar o objeto (linha 499 - 514). E faz-se o mesmo procedimento para as cores, normais e índices (linhas 518 – 557).

Para finalizar temos de dizer ao WebGL para usar o nosso programa quando tiver a desenhar e definimos os shaders (linhas 561 – 577).

Para desenhar o cubo chama-se a função *drawElements* com 36 indices.

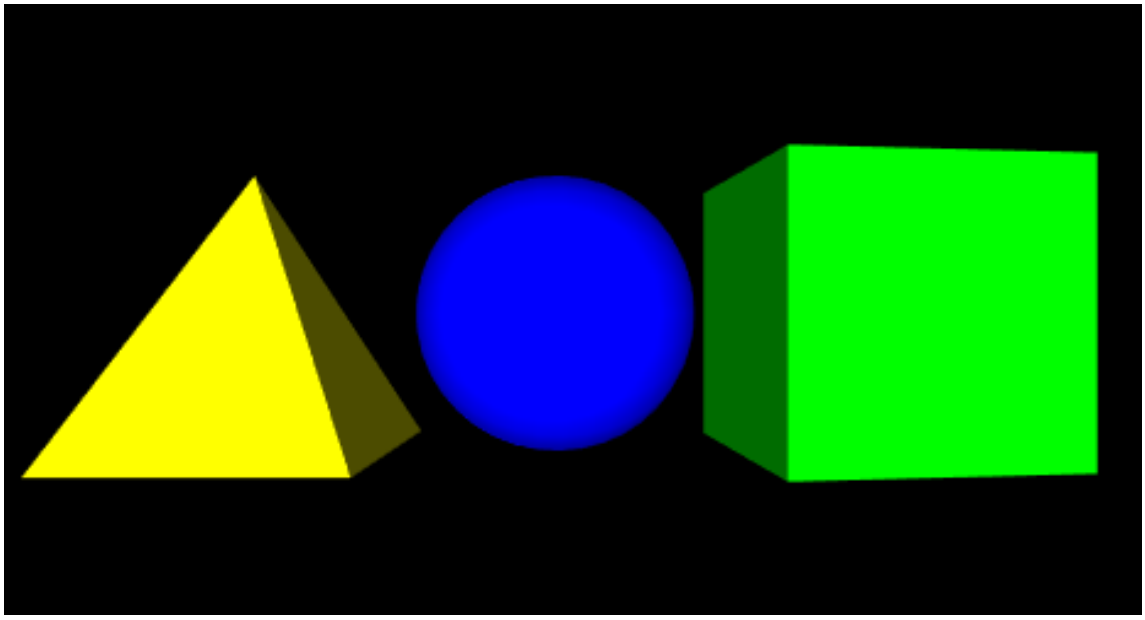
Para desenhar a esfera chama-se a funlão *drawElements* com 2280 indices pois o sectorCount e o stackCount estão a 20. O i é diferente de 0 19 vezes e diferente de stackCount – 1 19 vezes, logo 19*3*20*2.

A maior diferença para a pirâmide é que não se dá set aos índices e para desenhar a pirâmide usa-se a função *drawArrays* em vez de *drawElements*.



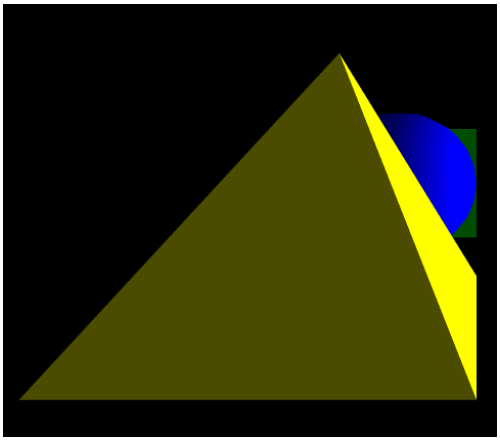
Modificar a posição da fonte de luz de modo a iluminar a parte do cubo de frente para a câmara

Para alterar a posição da luz de maneira a iluminar a parte do cubo de frente para a câmara, colocamos o valor de z no seguinte comando a 1.0 *"highp vec3 directionalVector = normalize(vec3(0.0, 0.0, 1.0))" (*linha 42).



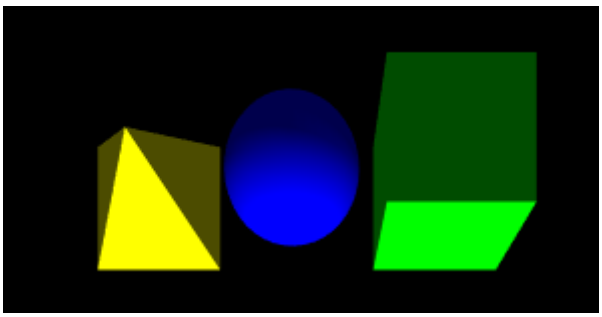
Modificar a posição da câmara de modo a apresentar uma visão da lateral da cena

Para alterarmos a posição da câmara, temos de aplicar translações e rotações sobre a *projectionMatrix*, logo para termos uma visão lateral da cena fazemos uma rotação de 90º sobre o eixo do y *"mat4.rotate(projectionMatrix, projectionMatrix, Math.PI / 2, [0.0, 1.0, 0.0])*(linha 468)*"* e uma translação para colocar a câmara no local certo *"mat4.translate(projectionMatrix, projectionMatrix, [5.0, 0.0, -5.0])*(linha 467)*"*.



Modificar a posição da câmara de forma a apresentar uma visão de topo da cena

Para alterarmos a posição da câmara, temos de aplicar translações e rotações sobre a *projectionMatrix*, logo para termos uma visão de topo da cena fazemos uma translação *"mat4.translate(projectionMatrix, projectionMatrix, [0.0, -10.0, 0.0]* (linha 474)*"* e uma rotação de 90º graus sobre o eixo do x *"mat4.rotate(projectionMatrix, projectionMatrix, Math.PI / 2, [1.0, 0.0, 0.0])*(linha473)*"*.

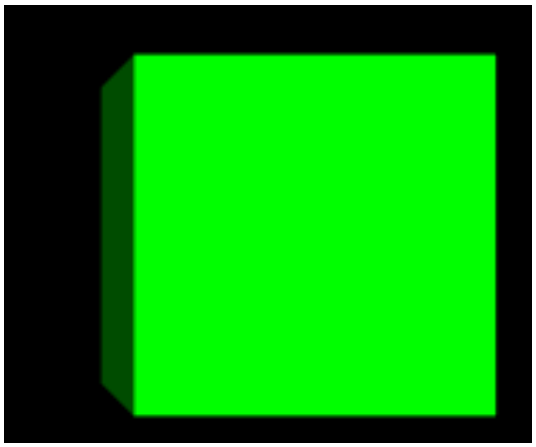


2 - Iluminação

Método de Phong

Tendo em conta que o código que inicialmente usámos tinha o método de Gouraud implementado onde a cor é calculada no Vertex Shader. Para implementar o método de Phong os cálculos passam a ser feitos no Fragment Shader (linhas 45 – 59). Pois o método de Phong usa a interpolação de normais no Fragment Shader para o cáculo das intensidades enquanto que o Gouraud usa a interpolação das intensidades no Vertex Shader

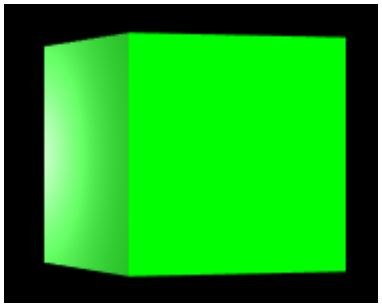
Os resultados são bastante parecidos e é complicado por vezes reparar nas diferenças.



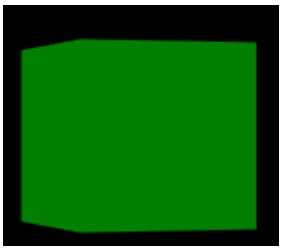
Luz especular

Para calcular a luz especular, começa-se por ir buscar o sítio para o qual a câmara está a olhar que é passado por um uniform e calcula-se a posição da câmara que é uma crossreference entre a modelview e os vértices

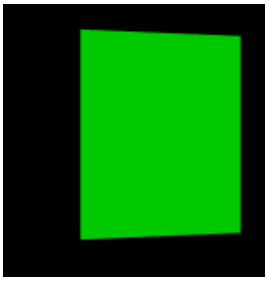
O halfVector será a normal da soma do vetor incidente da luz no objeto mais o vector da camara com o objeto a dividir pela normal dos mesmos



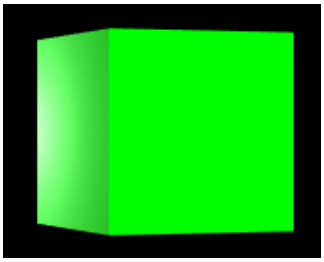
Visualização das diferentes componentes



Luz Ambiente



Luz Difusa

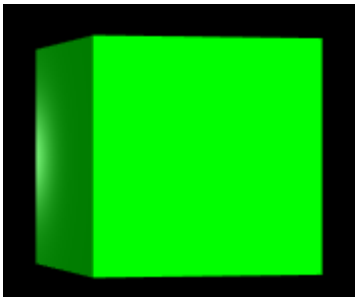


Luz Especular

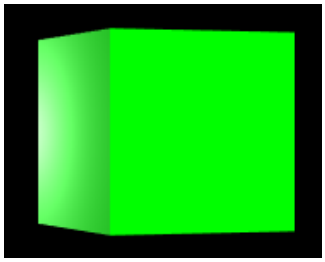
Vários valores diferentes de coeficiente especular

```
      gl_FragColor = vec4(vColor.rgb * vLighting + (pow(especular, 5.0)),
 vColor.a); //0 e 1 0 = MUITO ILUMINADO AKA ILUMINATI
```

Nesta linha de código presente em cima. É o cálculo feito para a luz especular. Quando temos pow(especular, 5.0) esse 5 é o coeficiente de especularidade.



Valor Especular a 5



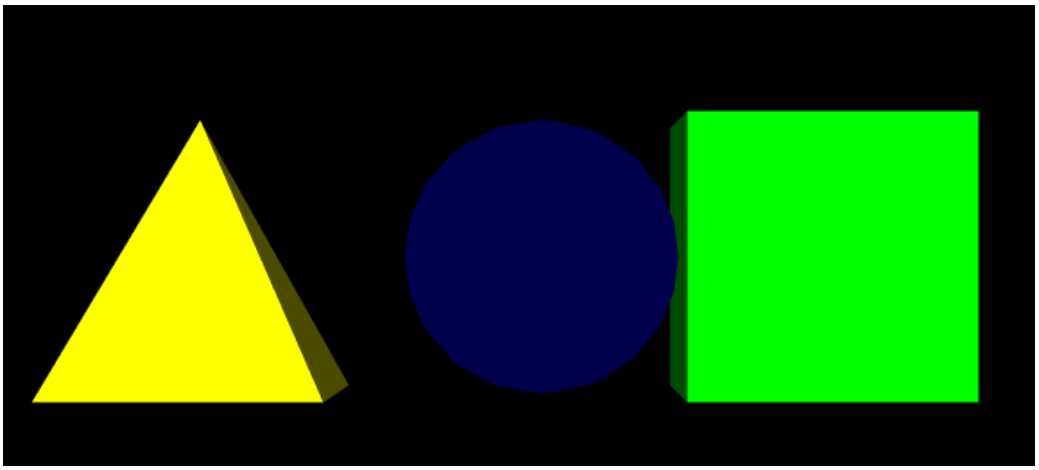
Valor especular a 1

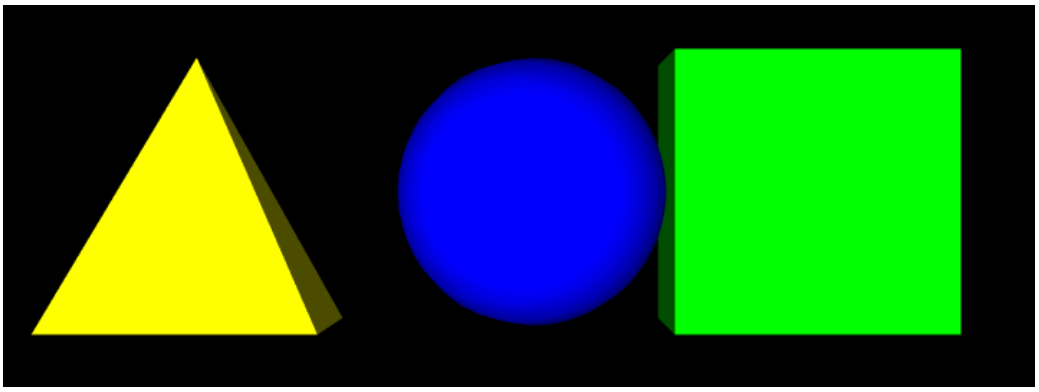
3 – Animação

Produza uma animação da cena iluminada movimentando apenas a fonte de luz direcional em torno da esfera

Para ser possível rodar a luz a cada frame, os valores da direção da luz não podem ser estáticos, mas sim alterar de acordo com o tempo. Portanto para que tal aconteça foram criados dois novos uniforms no vertex Shader que representam a direção da luz (linhas 42 e 43) e esses valores são usados para alterar a direção da luz. (linha 55).

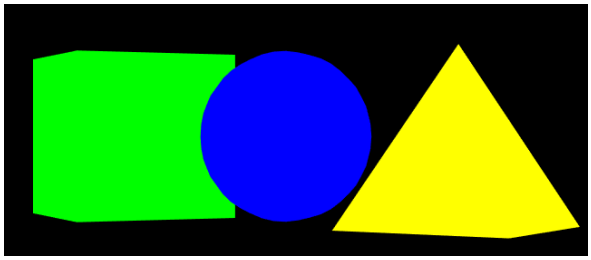
Como queremos apenas a luz em torno da esfera, para o cubo e a pirâmide a direção da luz vai continuar a ser de frente, o que significa que o vetor da luz será (0.0, 0.0, 1.0) (linhas 579 – 580 e 684-685). Para a esfera como queremos que o valor mude passamos o cosseno do cubeRotation e o seno do cubeRotation. É usado o cosseno e o seno para os valores variarem entre -1 e 1 o que faz com que a luz ande em torno da esfera. (linhas 797 – 798).

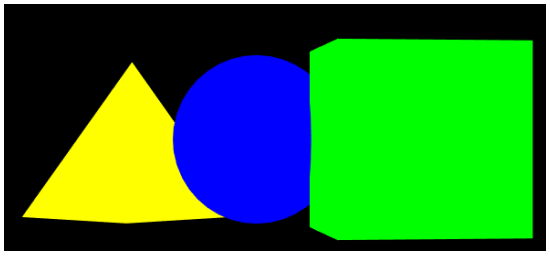




Produza uma animação da cena original movimentando apenas a câmara em torno da cena

Para rodar a câmara tem de se alterar a rotação da mesma a cada drawScene (linha 468) e apenas chegamos a câmara um bocado para trás para se conseguir ver melhor o que está a acontecer.





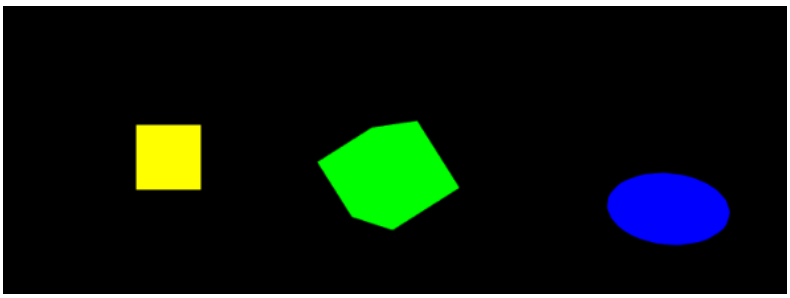
Considere os objetos adicionados no Exercício 1. Considere que a pirâmide é um sol, e que, portanto, permanece imóvel. O cubo é um planeta, que deve orbitar em torno do sol, e a espera é a lua do planeta. Implemente estes movimentos para o cubo e o planeta.

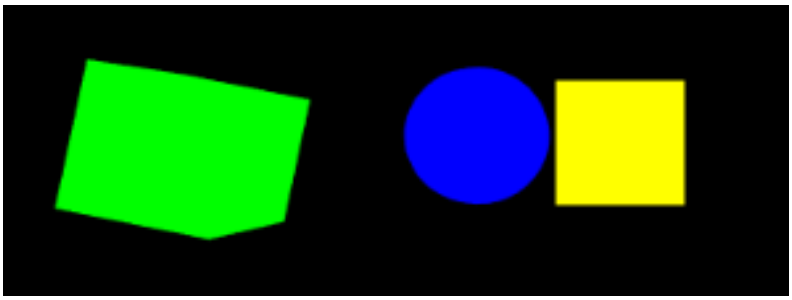
Inicialmente começamos por meter a câmara vista de cima para conseguir ter uma melhor perspetiva do que está a acontecer. (linhas 465 – 466).

No início metemos o cubo a rodar sobre si próprio (linha 489), depois para meter o cubo à volta da pirâmide, afastamos um bocado o cubo e depois rodamos sobre o eixo dos y na origem, pois é onde a pirâmide está situada. (linhas 478-486).

Para ter a certeza que a pirâmide está na origem é feito uma translação para lá (linhas 596 – 599).

Em relação à esfera, inicialmente também a metemos a rodar sobre si própria, depois fazemos uma translação desta para a posição do cubo e é feita uma rotação para rodar sobre o cubo e para conseguirmos ver essa rotação afastamos um pouco a esfera do cubo (linhas 697 – 711).





4. Novos modelos de objetos

No Blender modifiquem e simplifiquem significativamente o modelo da estátua ou do arco e gerem o modelo em formato .JSON

Para simplificar o modelo, o arco neste caso, baixamos o tamanho do arco (scale) e como já tínhamos o arco todo num só objeto não alteramos nada e ao tentar usar o modifier Decimate usando a opção Un-Subdivide estraga o arco logo não fizemos nada em relação à complexidade do mesmo.

Adicionem o objeto à cena de forma a que os objetos adicionados no exercício 1 e o objeto importado do Blender estejam na mesma cena.

Começamos por abrir o ficheiro que o professor disponibilizou no moodle e a partir disso fizemos o seguinte:

Como o ficheiro já estava a desenhar um objeto importado do blender, tivemos em conta esse exemplo e aplicamos o mesmo para o nosso arco (linha 403).

Tendo o arco desenhado só falta adicionar os objetos que desenhamos no exercício 1. Para tal aplicamos o mesmo código que foi feito no exercício 1, ou seja, criação de buffers necessários, (função initBuffers_main -> linhas 393 – 400) uma model matrix para cada um dos objetos, dar toda a informação necessária ao WebGL e por fim desenhá-los. (linhas 456 – 666).