Universidade do Minho
Escola de Ciências

Computação Gráfica

Ano letivo 21/22

- Fase 4 – Normais e coordenadas de textura

Grupo 10:

- Hugo Costeira - A87976
- João Silva - A87939
- João Goulart - A82643
- Pedro Martins - A87964

Índice

1. Introdução

Para esta fase do projeto, foi-nos pedido a continuação do desenvolvimento do sistema solar com a aplicação de iluminação e texturas.

O Generator deveria assim gerar coordenadas de textura e normais para cada vértice.

A aplicação de texturas e iluminação aos modelos apresentados no sistema solar foi,à semelhança das fases anteriores, feita através da leitura de um ficheiro XML onde deveria ser possível definir as fontes de iluminação. O presente relatório visa esclarecer, elucidar e demonstrar as soluções que o grupo encontrou para desenvolver a aplicação pedida.

2. Generator

Nesta fase o generator tem de gerar também as coordenadas das normais e as coordenadas de textura de cada ponto. Assim, para as primitivas *plane*, *box*, *sphere*, *cone* e *torus* são geradas as coordenadas dos pontos que as constituem, as coordenadas da normal e as coordenadas de textura de cada um desses pontos. Já na primitiva *teapot* (que representa o cometa) não apresentamos estas mudanças, gerando tal como na última fase apenas os pontos que farão parte do teapot a partir do ficheiro patch e imprimindo-os pela ordem correta. Cada um destes conjuntos é escrito num ficheiro 3D, onde a informação das coordenadas dos vértices, das normais e dos pontos de textura está separada, isto é, na linha antes das coordenadas dos vértices temos o número de coordenadas que estarão presentes, antes das normais temos um “n” a separar esta secção e, por fim, antes das coordenadas de textura temos um “t” seguido do número de coordenadas de textura que serão apresentadas.

2.1 TeaPot

Tal como na última fase, para gerar o teapot definimos a função void teapot (char* patchname, int tessellation, char* filename) que recebe como argumentos o ficheiro patch que possui os patches de Bezier para o Teapot, o nível de tesselação e o ficheiro destino que conterá os pontos geométricos do Teapot. Na função começamos por fazer parsing da informação do ficheiro patch, guardando o número de patches na variável *number_patches* e os patches em si numa matriz *índices* onde, por exemplo, índices[0][3] está presente o índice presente na posição 3 do patch 0 do ficheiro. Fazemos também parsing dos pontos de controlo em si, guardando na variável *n_ctrlpoints* o número de pontos de controlo e no array de arrays *ctrlpoints* os pontos de controlo onde, por exemplo, n_ctrlpoints[0][1] está presente o valor da variável *y* do primeiro ponto de controlo. Após termos toda esta informação guardada corretamente nestas variáveis temos de calcular os pontos do teapot aplicando a fórmula:

$$B(t) = (1 - t)^3 * P0 + 3 * t * (1 - t)^2 * P1 + 3 * t^2 * (1 - t) * P2 + t^3 * P3$$

Que é representada na forma matricial por:

$$p(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Onde t é uma variável que varia entre [0,1] e P0,P1,P2 e P3 são os 4 pontos de controlo mencionados anteriormente. O resultado da equação para qualquer t corresponde a uma determinada posição na curva, se resolvermos a equação para todos os t´s possíveis obtemos a curva completa.

Assim, começamos por imprimir o número de coordenadas que o teapot terá, aplicando a fórmula *number_patches * (tessellation) * 2 * (tessellation) * 3 * 3*. Após isto, para cada patch guardamos na variável *pontos,* definida da forma float pontos[16][3] as coordenadas de cada índice, isto é, se um patch for da forma [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], pontos[3] terá os valores das coordenadas *x,y* e *z* do ponto de controlo com índice 3. Após termos esta informação guardada em *pontos,* calculamos para cada conjunto de 4 pontos de controlo do patch o seu ponto correspondente na curva de bézier através da fórmula mencionada anteriormente (usamos a fórmula na sua forma matricial) e guardamos essa informação na variável *pontosTeapot* definida da forma float pontosTeapot[4][3], onde em pontosTeapot[0][1] temos o valor da coordenada y do primeiro ponto de teapot, resultante da aplicação da fórmula de bézier nos 4 primeiros pontos de controlo do patch.

Temos neste momento então para cada patch um conjunto de coordenadas de 4 pontos e para termos as coordenadas finais de um ponto do teapot temos de aplicar para esse conjunto de 4 pontos mais uma vez a fórmula de bézier, obtendo assim um ponto do teapot [1]. Todo este código de cálculo de pontos aplicando a fórmula de bézier é feito dentro de dois ciclos, onde o ciclo exterior itera sob a variável *v* e o interior sob a variável *u*, variando estas variáveis entre 0 e 1 (quanto maior a tecelagem mais vezes vão ser feitas estas iterações). Estes ciclos servem para definirmos a cada iteração quatro variáveis *x1*, *y1*, *x2* e *y2,* variáveis essas que usamos como a variável *t* na fórmula, passando-as como input na função que aplica a fórmula de bézier, deste modo ao fazermos o passo [1] fazemo-lo com cada uma destas variáveis obtendo assim 4 pontos finais do teapot para cada patch, pontos esses que estão guardados na variável *resultPoints*. Por fim, basta imprimir no ficheiro final as coordenadas de cada ponto pela ordem correta (seguindo logicamente a regra da mão direita) e obtemos assim as coordenadas dos pontos que formarão o nosso teapot.

2.2 Plano

Para permitir a inserção de uma textura no plano e para o iluminar, este, para além das coordenadas dos vértices, passou a ter também coordenadas para as normais e para as texturas. As coordenadas do vetor normal a cada ponto do plano vão ser sempre as mesmas, ou seja, serão (0,1,0) na face que fica voltada para cima.

Por outro lado, as coordenadas de textura serão, iterando para os vários pontos do plano, (1/divisions+x ,1/divisions+y) .

2.3 Box

No caso do cubo, as coordenadas normais serão (0,1,0) relativamente à face de cima, (0,-1,0) relativamente à face de baixo, (1,0,0) em relação à face da direita, (-1,0,0) em relação à face da esquerda, (0,0,-1) em relação à face de trás e (0,0,1) em relação à face da frente.

Já para as coordenadas de textura, as coordenadas serão as mesmas para todas as faces, isto é, (1/divisions+x ,1/divisions+y), para cada face do cubo.

2.4 Sphere

As coordenadas para as normais da esfera são iguais às dos pontos para desenhar esta primitiva, tirando o facto que para as coordenadas das normais dividimos cada ponto pelo raio da esfera.

As coordenadas de textura funcionam da seguinte forma: percorrendo cada stack e cada slice da esfera passamos 4 pontos em que cada um deles corresponde a um vértice da respetiva slice, assim cada coordenada corresponde aos vértices (0,0), (0,1), (1,0) e (1,1) de cada slice da esfera. Para efetuar este cálculo declaramos duas novas variáveis *float texts = 1.0 / slices e float textst = 1.0 / stacks* que vão corresponder ao tamanho de uma slice e de uma stack, respetivamente.

2.5 Cone

As coordenadas das normais para o cone dividem-se em duas partes: para a base e para a parte de cima do cone. As normais para a base têm sempre as mesmas coordenadas que são (0,-1,0). Já para a parte de cima do cone, as coordenadas são iguais às coordenadas dos vértices para desenhar a primitiva tirando o facto que agora se divide pelo raio.

Quanto às coordenadas de textura também seguimos o mesmo modelo, pelo que se divide em duas partes. Para a base do cone as coordenadas são iguais às dos vértices que usamos para desenhar a primitiva com a exceção que se divide pelo raio. Para a parte de cima do cone as coordenadas de textura são iguais à da esfera, ou seja, percorre-se cada slice e cada stack e desenha-se 4 pontos.

3. Engine

O Engine refere-se à parte do projeto que constrói a cena descrita no ficheiro XML, reproduzindo a iluminação, os objetos, as suas texturas e as suas transformações. Vamos abordar de seguida o código pertencente à engine que foi alterado relativamente à última fase, pelo que não será mencionado nada acerca do que permaneceu igual.

3.1 Estruturas de dados

Em primeiro lugar, na estrutura Características, foram adicionadas as seguintes variáveis:

- **int ignore**, que toma o valor 1 caso o objeto não tenha coordenadas normais, nem de textura, ou 0 caso as tenha;

- **Glfloat emiss/diff/spec/amb/shiny**, que se referem ao tipo de luz emitida pelo objeto a desenhar, podendo este ter 4 tipos diferentes: especular (que reflete a luz num local especifico), difusa (que reflete a luz que recebe, emitindo-a), emissiva (que emite a sua luz própria) e ambiente (que emite uma quantidade de luz igual à volta do objeto);

- **Buffer b**, sendo Buffer igual à estrutura definida na fase anterior (necessário para os VBO´s);

Também foi definida uma estrutura chamada Light que contém as informações do tipo de luz que será utilizado para iluminar o sistema solar, podendo essa luz variar entre posicional, direcional ou do tipo spot. O array pos é preenchido independentemente do tipo de iluminação, enquanto o array spot é preenchido apenas se o tipo de iluminação for spot a estrutura, sendo cut o ângulo dessa iluminação spotlight.

3.2 ReadXML e readXMLaux

A função readXML foi separada em duas funções: readXML e readXMLaux. A função ReadXML faz a leitura inicial da informação sobre o tipo de iluminação utilizado no ficheiro XML, conta o número de planetas (sol incluído) no mesmo ficheiro e ainda lê cada planeta, usando para isso a função readXMLaux. Assim, a função readXMLaux lê a informação sobre as translações, rotações e escalas de uma forma semelhante à função readXML da fase anterior, no entanto agora é adicionalmente feita a leitura da informação relacionada à iluminação dos planetas/luas, distinguindo o seu tipo (emissiva, difusa, ambiente ou especular) e guardando a informação nos arrays emiss, dif, amb e spec, respetivamente. No final da função, também verificamos se existem luas associadas ao planeta/lua que estamos a analisar e se existirem, as informações delas serão lidas recursivamente usando esta mesma função.

3.3 Transform

A função Transform, de uma forma bastante semelhante às fases anteriores, vai aplicar as transformações pretendidas para cada objeto a ser desenhado, no entanto, recebe agora uma flag como parâmetro extra, que distingue as transformações do próprio planeta/lua das transformações do planeta/lua ao qual ele está associado. Isto foi necessário porque as luas, além das suas próprias transformações também irão ter todas as transformações dos planetas, exceto a escala.

Em primeiro lugar efetuamos as transformações geométricas tais como translações, rotações e escalas. Para calcular o tempo de uma translação e de uma rotação utilizamos a função glutGet(GLUT_ELAPSED_TIME) que nos fornece o número de milissegundos desde a chamada do glutInit. Na rotação, calculamos o ângulo de rotação através do número de segundos que é necessário para realizar uma rotação completa de 360 graus em torno do eixo especificado. Na translação, primeiro calculamos o resto da divisão entre o glutGet(GLUT_ELAPSED_TIME) e o tempo que foi fornecido no ficheiro xml (em milissegundos), para o tempo ser menor nos casos em que o resto da divisão não é exatamente igual ao valor do glutGet(GLUT_ELAPSED_TIME) e atribuímos o valor a uma variável que chamamos "time". De seguida, utilizamos essa variável time e dividimos pelo tempo fornecido no ficheiro xml (em milissegundos), que será necessário para calcular uma curva. Por fim, verificamos que foram passados argumentos, para iluminação, no ficheiro xml para o planeta ou lua em questão. Se existirem, iluminamos o planeta ou lua com a luz respetiva fazendo, por exemplo, glMaterialfv(GL_FRONT, GL_DIFFUSE, c->diff), quando se trata de um tipo de luz difusa.

3.4 RenderScene e renderSceneaux

A função anteriormente chamada renderScene foi também dividida em duas funções, de modo a ser possível a sua implementação recursivamente. A renderScene começa por definir a iluminação e, em seguida, percorre o array *planetas* onde com a ajuda da função renderSceneaux desenha todos os objetos pretendidos.

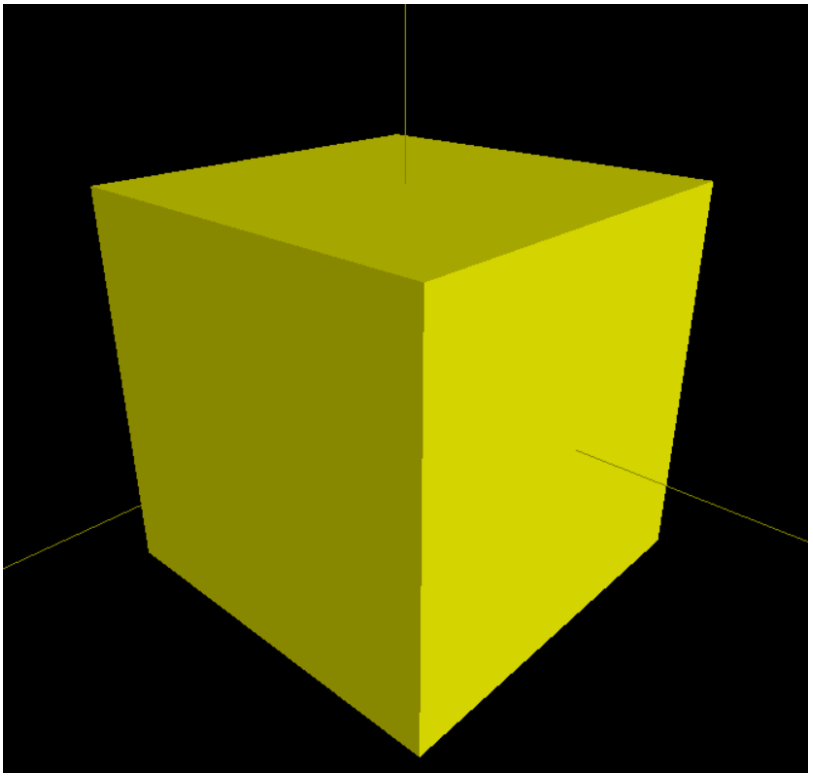
A função renderSceneaux recebe como input o objeto a ser desenhado e o planeta a qual ele pertence (caso não exista, será entregue o valor NULL). Inicialmente, aplica todas as transformações do planeta associado (se for o caso) e as transformações do próprio objeto. Após isto, usa os valores do buffer para gerar o VBO, os valores das normais e os valores das texturas. Caso o objeto analisado tenha luas associadas a ele, irá recursivamente desenhar as luas.

3.5 ReadFile

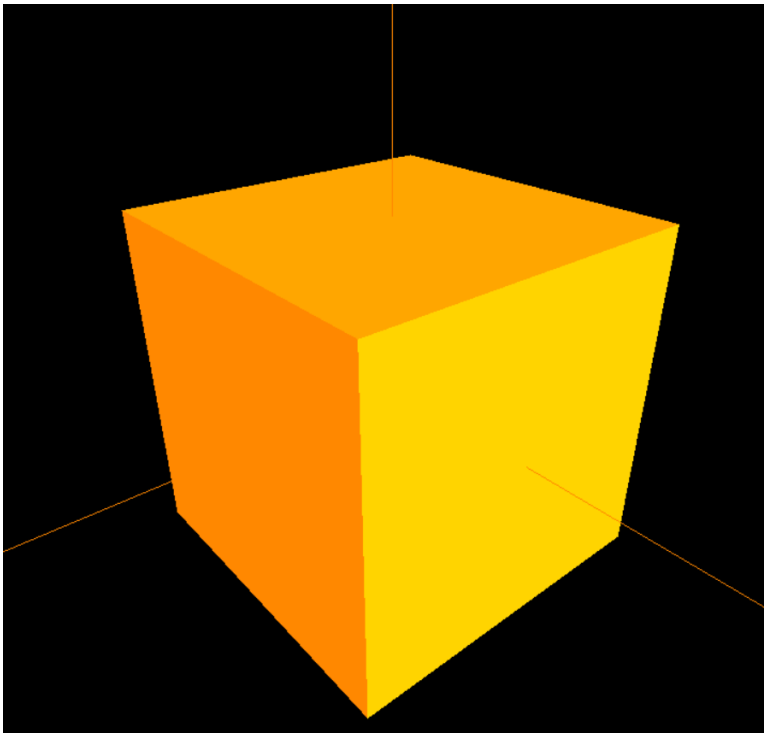
Esta função lê os ficheiros 3d tendo em conta a sua estrutura específica: número de coordenadas de vértices seguidos dos vértices em si, coordenadas normais, número de coordenadas de textura e, no fim dos ficheiros 3d, coordenadas de textura. A função usa o número de coordenadas para alocar memória para os arrays *vertices*, *normais* e *texturas*, tendo os arrays *vertices* e *normais* o mesmo número de coordenadas. Para além disto, a função irá também guardar nesses arrays as coordenadas lidas no ficheiro e caso um ficheiro não tenha coordenadas normais e coordenadas de textura, a função simplesmente não as lê, não gerando problemas. Definimos ainda a variável *ignore,* variável esta que fica com valor 1 se ao ler o ficheiro 3d virmos que é uma primitiva que não tem coordenadas de textura nem normais. Tivemos necessidade de definir esta variável pois quando essa variável está a 1 não aplicamos textura nem iluminação na função renderSceneaux, visto que estamos com uma primitiva que não tem essas propriedades.

4. Demonstração do resultado final

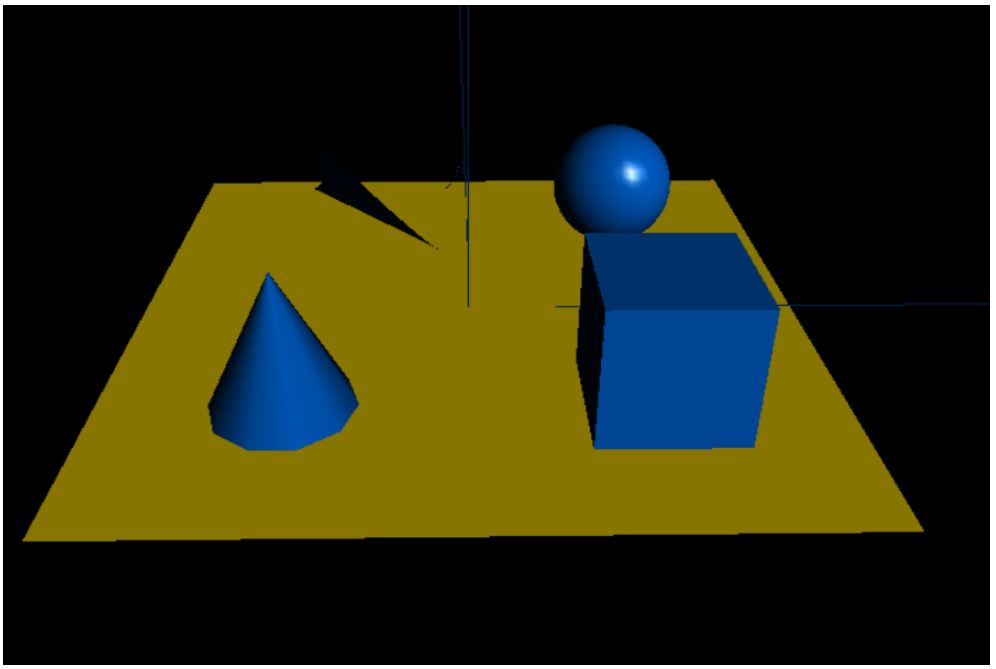
teste 4_1



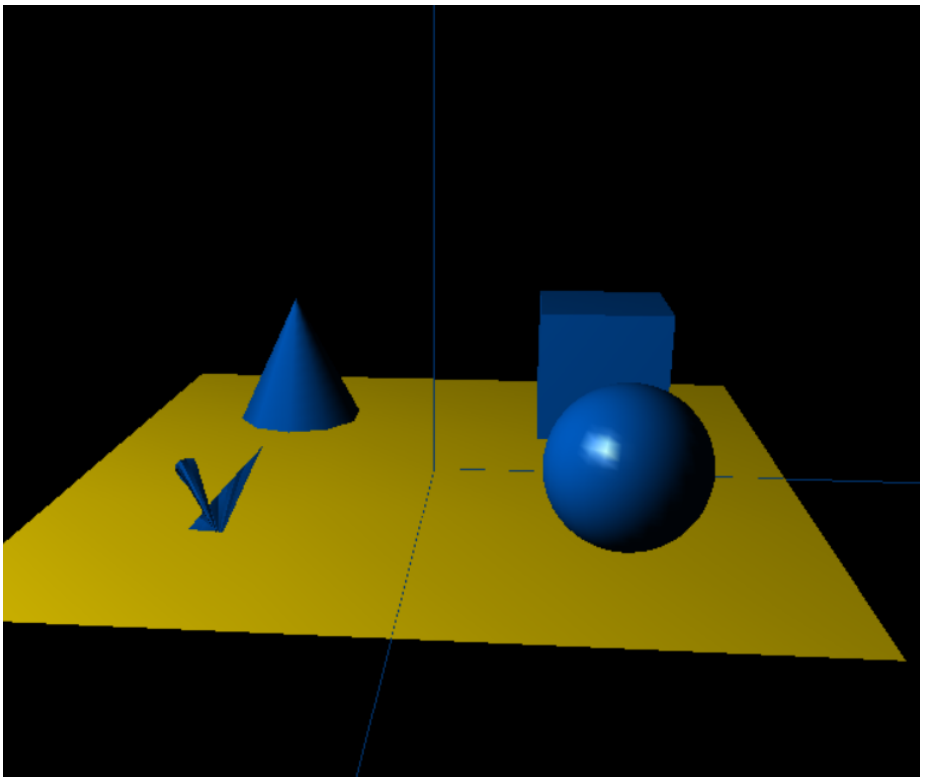
teste 4_2



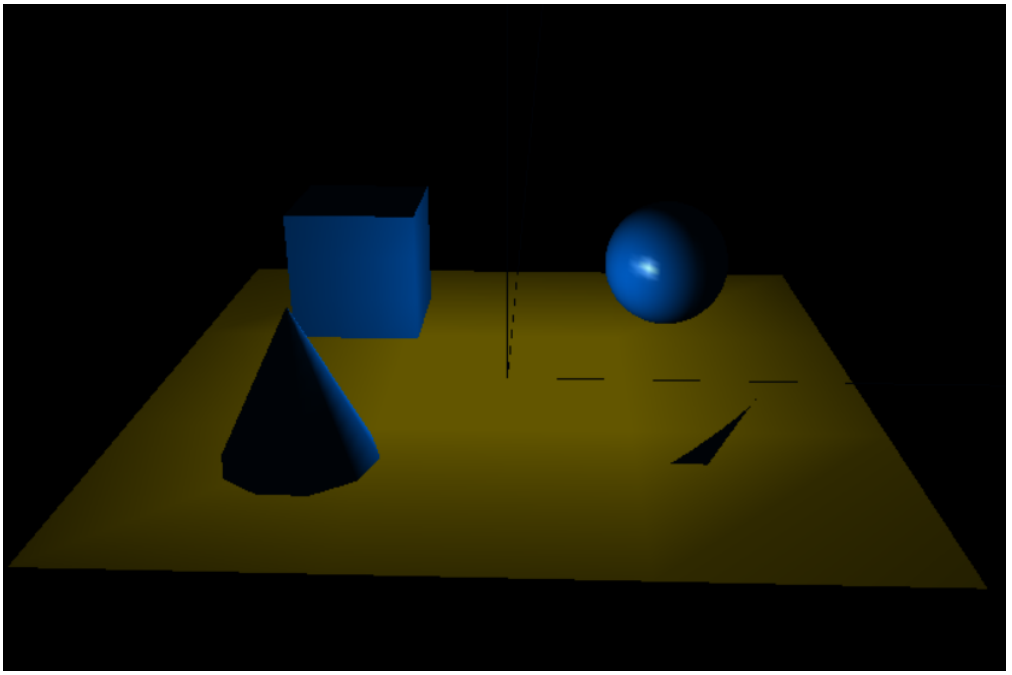
teste 4_3



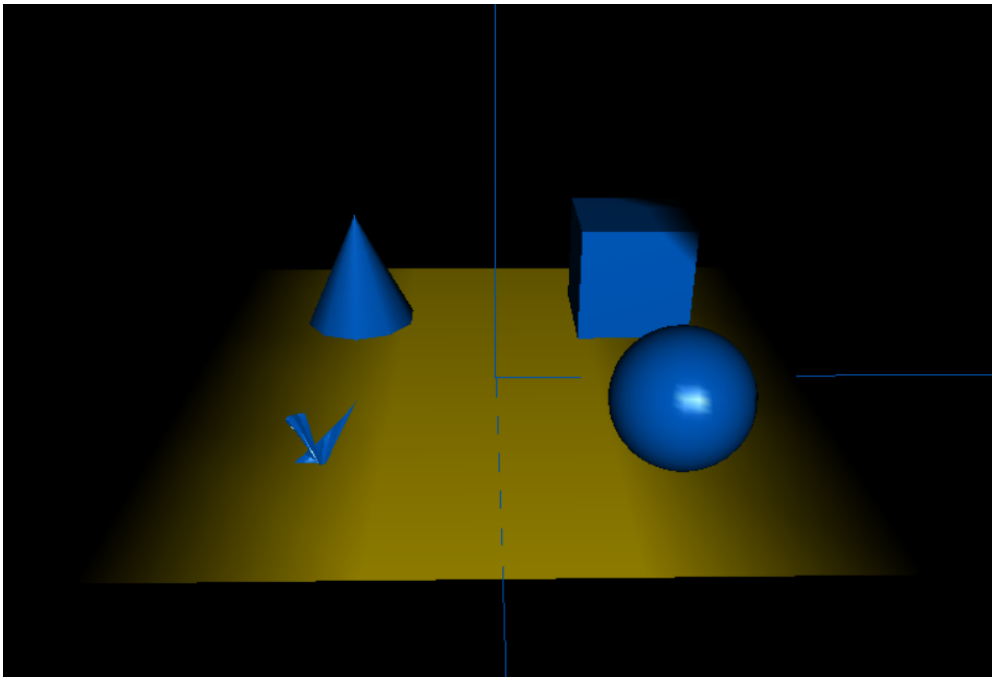
teste 4_4



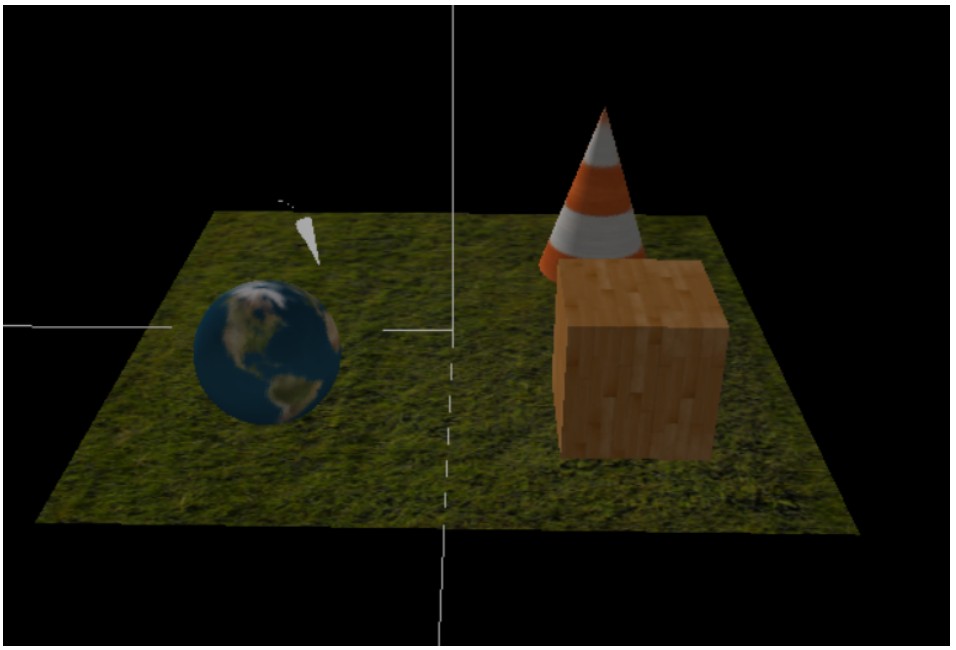
teste 4_5



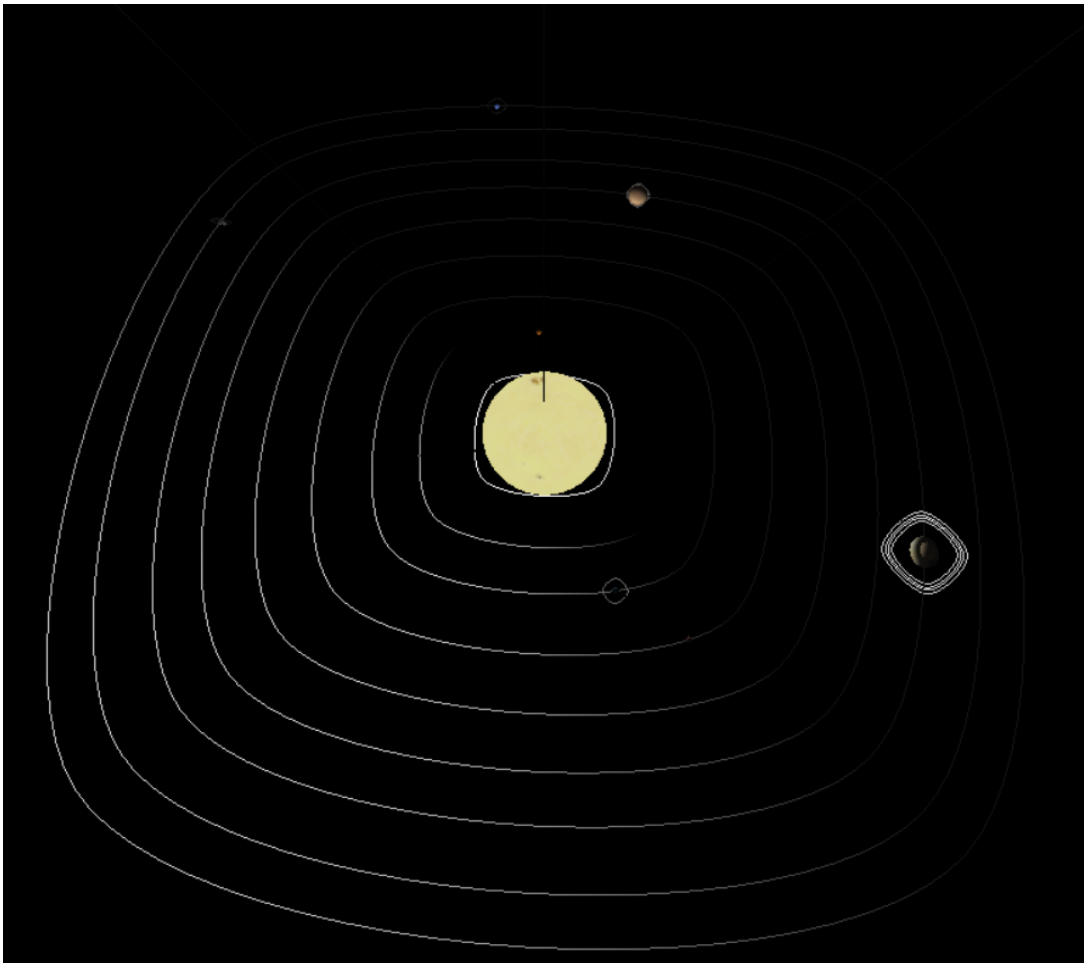
teste 4_6



teste 4_7



Sistema Solar



4. Conclusão

A quarta e última fase do projeto propõe-nos a geração adicional das normais dos modelos e de coordenadas para as texturas, sendo possível aplicar vários tipos de luzes para a iluminação da cena e também aplicar texturas aos modelos. Através de uma reflexão geral sobre a quarta fase do projeto o nosso grupo concluiu que satisfez os requisitos do projeto, com necessidade de menção a alguns aspectos importantes:

- O TeaPot foi algo que nos causou alguma dificuldade/transtorno e acabou por não ficar como pretendido.
- Os planetas não efetuam rotações (em torno de si próprios).
- As transformações apenas funcionam pela seguinte ordem: translate, rotate e scale.

De uma forma geral, concluímos o projeto com a ideia de que assimilamos bem o que a unidade curricular nos ensinou ao longo do semestre, o que nos deixa satisfeitos com o resultado final obtido na componente prática da unidade curricular de Computação Gráfica.