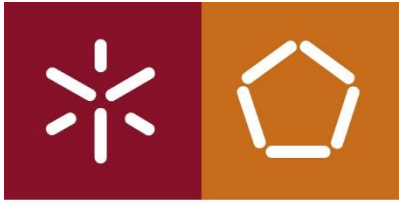

Universidade do Minho
Escola de Engenharia

Computação Gráfica

Relatório do projeto prático - Fase IV

Normals and Texture Coordinates

Grupo de Trabalho

Ana Esmeralda Fernandes

A74321

Bárbara Nadine Oliveira

A75614

João Paulo Ribeiro Alves

A73542

Miguel Dias Miranda

A74726

Mestrado Integrado em Engenharia Informática

maio de 17

Conteúdo

Índice figuras

1. Introdução

Neste relatório serão descritas todas as etapas que levaram à resolução da fase *IV* do trabalho prático, relativo à unidade curricular de Computação Gráfica, procurando assim, explicitar e justificar todas as decisões tomadas pelo grupo para a resolução dos novos requisitos.

Nesta fase surge a implementação de texturas, a partir de um determinado ficheiro especificado no xml de input, assim como a aplicação de iluminação e propriedades do material das superfícies dos objetos. Relativamente à aplicação da luz, foram criados no ficheiro *xml* de input campos e elementos que contem informações relativas às fontes de luz que se desejam implementar, bem como a enumeração das suas componentes (difusa, ambiente, especular).

Serão ainda descritas as alterações efetuadas no gerador, onde passa a ser necessário determinar quais as normais dos vértices que compõe um determinado modelo, assim como as respetivas coordenadas de textura. Sem este novo tipo de informação, não seria possível aplicar de forma correta as diferentes componentes da luz nem a texturização de um modelo, posteriormente, na aplicação *engine* sistemaSolar.

Toda a modulação do problema foi feita com recurso à ferramenta de programação Visual Studio e à linguagem C++, focando as componentes e técnicas abordadas nas aulas práticas da UC.

2. Aplicação sistemaSolar

2.1. Iluminação em OpenGL

Em OpenGL, a técnica de iluminação é dividida em quatro componentes individuais que procuram representar as características da luz e as propriedades do material sobre o qual a luz incide. Estas componentes (difusa, ambiente, especular e emissiva) podem ser definidas individualmente e posteriormente combinadas, criando assim o efeito de iluminação de um objeto.

A componente difusa, representa a iluminação geral do objeto, quando este se encontra na direção de uma fonte de luz especifica e bem definida. A luz difusa pode assim ser descrita como uma fonte de luz que vem apenas de uma única direção e na qual a cor da luz define a cor do objeto sobre o qual esta incide.

Quando a luz embate numa superfície, esta espalha-se igualmente em todos os sentidos.

A componente ambiente representa uma fração desta luz que foi recursivamente fragmentada pelo meio, em sucessivas reflexões, sendo praticamente impossível determinar a sua direção inicial. As luzes de fundo e as sombras têm a sua componente de luz ambiente maior que as restantes, exatamente pelo facto de apesar de nenhuma fonte de luz incidir diretamente nestas zonas, estas ainda recebem alguma luz que foi refletida indefinidamente até chegar ao objeto. De forma geral, a componente ambiente é a cor de um objeto quando este se encontra à sombra, mesmo sem que este tenha uma fonte de luz diretamente direcionada para ele.

A iluminação especular, ou *specular highlight,* representa o "brilho" de um objeto, causado pela reflexão de uma determinada fonte de luz sobre si. Tal como na vida real, esta componente da luz apresenta a cor branca.

Quando se pretende representar objetos com uma superfície brilhante e polida, como no caso de metais ou alguns tipos de plástico, estas objetos devem ser incididos com uma fonte de luz cuja componente especular seja bastante alta. Para materiais mais baços, como giz ou tecidos, a sua reflexão espelhada e brilho são praticamente nulos, pelo que a luz que nele incidir deve ter uma componente especular baixa.

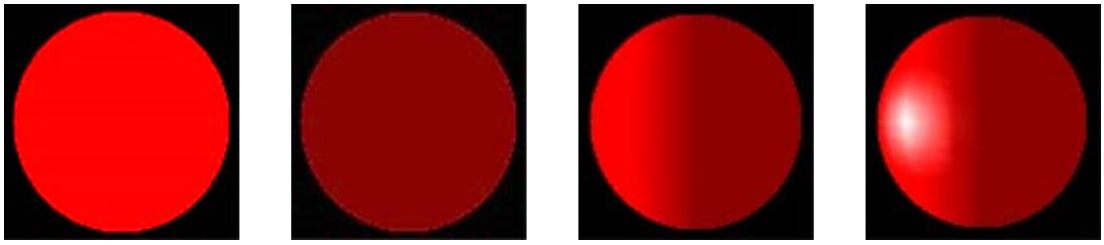

Figura 1 – Combinações das componentes Difusa, Ambiente e Especular

O conjunto de imagens apresentado exemplifica os diferentes aspetos de uma superfície esférica quando exposta ás componentes da luz referidos acima.

No primeiro exemplo temos a esfera iluminada apenas com a componente difusa, que define a sua cor geral. No segundo exemplo, temos a esfera incidida apenas com a componente ambiente, como se toda a superfície estivesse à sombra.

Na terceira imagem, obtemos o resultado da combinação das componentes difusa com a ambiente, criando o efeito de iluminação e sombra, conforme o corpo do objeto. No último exemplo, é possível identificar a aplicação de luz especular, que ao incidir na esfera cria a "mancha branca" que representa a luz refletida pelo corpo, quando este se encontra na direção de uma fonte de luz com componente especular elevada.

A componente emissiva é ligeiramente diferente das outras componentes da luz previamente referidas. Esta componente é responsável pelas características refletoras da superfície/material dos objetos, ou seja, representa a quantidade de luz que um objeto reflete ou absorve para si mesmo.

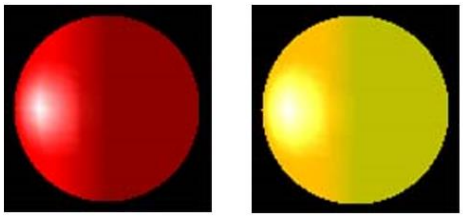

Figura 2 – Componente Emissiva

A imagem anterior apresenta como exemplo uma esfera inicialmente incidida por uma luz difusa vermelha, luz ambiente vermelha escura e luz especular branca.

A esta esfera é aplicada então uma luz emissiva verde. Se esta luz não existisse a esfera permaneceria com o aspeto inicial, contudo quando os raios atingem a superfície da esfera as cores, fundem-se com as componentes vermelhas, produzindo um acabamento amarelado como cor da superfície.

Na vida real, as inúmeras ocorrências de reflexões e refrações da luz são demasiado complexas e dependentes de vários fatores externos. A tentativa de simulação computacional destas situações, tal como acontecem com a luz na vida real, seriam assim praticamente impossíveis de determinar devido ao elevado poder de processamento e tempo necessário para computar todos os parâmetros.

Desta forma, em OpenGL, a técnica de iluminação utilizada baseia-se em aproximações da luz real, utilizando assim um modelo (formula) simplificada. No entanto, não existem diferenças significativas quando se compara visualmente um objeto sobre luz natural e um objeto renderizado, iluminado virtualmente.

Um dos modelos mais usados nestes processos de iluminação simulada, sendo também o escolhido para utilizar neste projeto, é o de modelo de iluminação de Phong. Os principais parâmetros da formula de Phong são as componentes ambiente, difusa e especular da luz e o resultado obtido num objeto é equivalente ao exemplo apresentado no ultimo quadro da **figura 1**.

2.1.1. Estrutura do XML

Para a inserção de luzes no contexto do nosso projeto, foi estabelecido um novo elemento no formato do xml, designado por ***luzes***.

Este campo poderá ser inserido quer dentro do elemento ***scene*** como do elemento ***group***, e deve conter nele todas as fontes de luz que se desejam implementar. Por cada fonte de luz que se pretenda definir, deve ser criado um elemento xml filho deste elemento ***luzes***, e designado como ***luz***, onde se irão fazer as especificações da luz a inserir.

Cada elemento **luz** terá por sua vez um conjunto de elementos filho, designados por **tipo.**

Cada um destes elementos ***tipo***, representará cada uma das componentes da luz referidas na secção anterior, sendo que não é obrigatória a introdução explicita de todas as diferentes componentes.

Este elemento **tipo** irá ter um atributo designado por ***componente,*** para identificar o tipo de luz a que os dados do elemento se referem.

Dependendo do valor do atributo ***componente***, o elemento ***tipo*** irá ter os seguintes atributos:

- Se a componente for *"VECTOR"* ou "*PONTO*" o elemento ***tipo*** contem ainda os atributos ***posX, posY, posZ,*** que representam as coordenadas do vetor que indica a direção da luz ou do ponto de luz, respetivamente;

- Se a componente for "*DIFUSA*" o elemento ***tipo*** contem ainda os atributos ***diffR, diffG, diffB,*** que representam os valores a atribuir aos parâmetros RGB da componente difusa da luz;

- Se a componentes for "*AMBIENTE*" o elemento ***tipo*** contem ainda os atributos ***ambR, ambG, ambB,*** que representam os valores a atribuir aos parâmetros RGB da componente ambiente da luz;

Para os requisitos necessário na abordagem do presente projeto, o elemento ***luzes*** é apenas utilizado dentro do elemento ***scene*** e tem apenas uma única fonte de luz.

O excerto de XML abaixo representa a luz utilizada na cena no projeto e permite identificar tudo aquilo explicado acima:

```
<luzes>
      <luz>
          <tipo componente="VECTOR" posX="-10.0" posY="10.0" posZ="6.0" />
          <tipo componente="DIFUSA" diffR="1.0" diffG="1.0" diffB="1.0" />
          <tipo componente="AMBIENTE" ambR="0.0" ambG="0.0" ambB="0.0" />
      </luz>
</luzes>
```

*Figura 3 – Exemplo do elemento **Luzes** no xml de input*

2.1.2. Implementação das luzes

Tal como referido na secção anterior, na introdução de iluminação no projeto sistema solar apenas se permite que o elemento ***luzes*** seja utilizado dentro do elemento ***scene*** do xml.

Com esta decisão, a animação apresentada com todos os elementos do sistema solar desenvolvidos, pode ter várias fontes de luz, mas estas serão comuns a todos os objetos processados, não existindo assim luzes exclusivas para um determinado grupo.

Para realizar esta alteração foi necessária a criação de uma nova função de parsing do ficheiro xml de input, de forma a retirar os dados relevantes relativos a luzes. A implementação desta função não acrescenta nenhuma funcionalidade nova ao projeto, que já não fosse aplicada nas restantes funções de parsing do xml associadas aos outros elementos do xml.

De forma genérica, a função determina se existe um grupo ***luzes*** no ficheiro de entrada. Caso exista, irá percorrer cada um dos eventuais elementos ***luz*** descendentes. Para cada um destes, chama a função de parsing, designada por *parseLuzes*, que iterar os eventuais elementos ***tipo*** filhos do elemento ***Luz***.

Para cada elemento ***tipo*** encontrado, é analisada o tipo de informação que este representa, olhando para o atributo ***componente*** já citado no anterior tópico. Feito este processo é criada uma variável da classe **Luz**, que irá assim guardar as informações retiradas dos elementos xml ***tipo*** encontrados.

Esta variável é ainda acrescentada a um vector de Luzes, porque apesar de no exemplo produzido só existir uma luz para tudo o sistema solar, seria possível especificar várias fontes de luz (novamente, estas seriam comuns a todos os objetos do sistema solar produzido).

Foi ainda criada uma nova classe, designada por Luz. Esta classe contem como variáveis um conjunto de vetores, que representam a posição e os parâmetros RGB para as componentes da luz ambiente e difusa. Durante o processo de parsing, caso o formato do xml esteja incorreto ou caso alguma das componentes não seja fornecida (por exemplo indicando apenas os valores RGB da componente difusa e a direção da luz, mas não indicando os valores RGB da componente ambiente), serão utilizado valores pré definidos, evitando assim possíveis erros e facilitando o processo de "ligar" as luzes, com todas as suas componentes, na função *renderscene*.

Como referido, pelo facto das luzes especificadas no ficheiro xml só poderem estar associadas ao elemento scene, na função *renderscene* será apenas necessário ligar as luzes, antes de desenhar todos os objetos do sistema solar, e desligar as mesmas luzes posteriormente ao render destes objetos. Não existe assim a possibilidade, na implementação atual, de ligar luzes associadas exclusivamente a um grupo ou a um modelo.

Para esta tarefa foi criada a função *setLighting,* que ativa ou desativa as luzes conforme o parâmetro de entrada recebido.

Caso o parâmetro seja *true*, e caso existam luzes na variável global com a informação sobre a cena a desenhar, ativa cada uma dessas luzes considerando as suas componentes *POSITION AMBIENT e DIFFUSE.* Para a definição de cada uma das componentes da luz, é utilizada a função *glLightfv* e para "ligar" a luz é utilizada a função *glEnable*, inicialmente com a primitiva GL_LIGHT**i**, onde **i** representa a luz a ligar, e depois com a primitiva GL_LIGHTING.

Caso o parâmetro de entrada seja *false,* é realizado o processo de desligar as luzes. O seguinte excerto de código procura modelar o processo de ativação das luzes.

```
void setLighting(bool escolha) {
       if (ligar) {
           int i = 0;
           for (auto luz : parsingXML->getLuzesCena()) {
              glLightfv(GL_LIGHT0 + i, GL_POSITION, luz.getPosicao().data());
              glLightfv(GL_LIGHT0 + i, GL_AMBIENT, luz.getAmbiente().data());
              glLightfv(GL_LIGHT0 + i, GL_DIFFUSE, luz.getDifusa().data());
              glEnable(GL_LIGHT0 + i);
              i++;
           }
           glEnable(GL_LIGHTING);
       }
       if (desligar) {
            for (int i = 0; i < parsingXML->getLuzesCena().size(); i++) {
              glDisable(GL_LIGHT0 + i);
            }
            glDisable(GL_LIGHTING);
       }
```

Figura 4 – Método para ativar e desligar as luzes

Em Anexo, na seção designada "Iluminação do sistema solar", estarão disponíveis algumas imagens que representam o sistema solar produzido composto apenas com processos de iluminação.

2.2. Materiais em OpenGL

Tal como na vida real, a superfície de cada objeto reage de forma diferente quando exposta a um determinado foco de luz. Em OpenGL, estas propriedades do material de uma determinada superfície podem também ser simuladas, recorrendo às componentes de cor ambiente, difusa, especular e ainda a um fator de "brilho" do material do objeto.

De forma geral, a tarefa e contexto das componentes ambiente, difusa e especular, têm o mesmo significado e referem-se ás mesmas características já enunciadas na secção anterior referente à iluminação. Contudo, os valores destas componentes especificadas para as luzes têm um significado diferente quando aplicadas aos materiais de um dado objeto.

No contexto da iluminação, os valores de uma determinada componente correspondem a uma percentagem de intensidade total para cada cor. Por exemplo, se as componentes R, G e B estiverem todas a 1, a cor apresentada será o branco puro e caso as mesmas componentes estejam todas a 0.5, a cor será igualmente branca, mas com metade da intensidade, parecendo cinzento. Para o contexto dos materiais, estes valores referentes aos elementos RGB de cada componente, correspondem à parte refletida dessas cores. Assim, se o R=1, G=0.5 e B=0, numa determinada componente, o material irá refletir toda a luz vermelha que receber, refletir metade da luz verde e absorver toda a fração de luz azul.

Por exemplo, uma bola que seja completamente azul reflete toda a quantidade de azul recebida e absorve todas as outras componentes vermelha e verde.

- A referida bola azul, incidida com uma luz branca (composta com as mesmas quantidades que vermelho, verde e azul) irá refletir toda a componente azul desta luz, e absorver as restantes. A bola será assim azul;
- Se a bola for incidida com um foco de luz puramente azul, a bola é novamente azul, porque este foco de luz tem uma componente azul que a bola reflete, e por isso a vemos com cor azul.
- Se por outro lado, esta bola for visualizada com uma luz verde, ou qualquer outro foco de luz cuja cor não apresente componente azul, veremos a bola preta.
  Isto ocorre porque como não é recebida nenhuma componente de luz azul, a bola só reflete esta componente, nada é refletido e o foco de luz é "absorvido" pela bola.

2.2.1. Estrutura do XML

De forma semelhante ao processo seguido para a implementação de iluminação, para se definirem as componentes do material de um objeto, foi criado o elemento xml designado por **material**. Este elemento será descendente do elemento ***model*** e, conforme o valor do seu atributo ***tipo,*** o elemento irá conter informações relativas a uma determinada componente da cor do material.

Dependendo do valor do atributo ***componente***, o elemento ***material*** irá ter os seguintes atributos:

- Se a componente for *"EMISSIVA"* o elemento ***material*** contem ainda os atributos ***emisR, emisG*** *e* ***emisB****,* que representam os valores a atribuir aos parâmetros RGB da componente emissiva do material do objeto;

- Se a componente for "*DIFUSA*" o elemento ***material*** contem ainda os atributos ***diffR, diffG, diffB****,* que representam os valores a atribuir aos parâmetros RGB da componente difusa do material do objeto;

- Se a componentes for "*AMBIENTE*" o elemento ***material*** contem ainda os atributos ***ambR, ambG, ambB****,* que representam os valores a atribuir aos parâmetros RGB da componente ambiente do material do objeto;

Na seguinte imagem será possível ver um exemplo deste formato do xml, em que são atribuídos valores ás componentes difusa, ambiente e emissiva do material do objeto, que no caso apresentado representa o sol do sistema sola. Neste modelo, assim como para os restantes, a luz especular usada é a luz branca.

```
<models>
    <model>
        <ficheiros file="esfera.3d" textura="sun.jpg" />
        <material componente="DIFUSA" diffR="1.0" diffG="0.8" diffB="0.0" />
        <material componente="AMBIENTE" ambR="1.0" ambG="0.647" ambB="0.0" />
        <material componente="EMISSIVA" emisR="0.1" emisG="0.1" emisB="0.0" />
    </model>
</models>
```

*Figura 5 – Exemplo do elemento **material** no xml de input*

2.2.2. Implementação dos materiais

Para determinar as eventuais informações relativas às propriedades do material de um determinado modelo, foi alterada a função que realiza o parsing do campo ***model*** do xml, designada por parseModelo. A função passa assim por também analisar os possíveis elementos ***tipo*** e conforme o valor do seu atributo ***componente*** determina qual a informação a retirar dos seus campos, seguindo um procedimento bastante semelhante ao utilizado para realizar o parsing das luzes em xml.

Para guardar esta informação, foi adiciona à classe Modelo quatro novas variáveis que representam as quatro componentes (ambiente, difusa, especular e emissiva) da cor que se podem atribuir ao material de um dado modelo. Tal como ocorria no parsing das componentes da luz, durante o processo de leitura, caso o formato do xml esteja incorreto ou caso alguma das componentes não seja fornecida, serão utilizados valores pré-definidos.

A atribuição das propriedades de um determinado material a um objeto, é realizada na função *desenhaGrupos* chamada pelo método *renderscene*. Assim, para cada modelo que exista num determinado grupo, depois de se carregarem os dados dos vértices e das normais do modelo para os respetivos VBOs, retira-se do modelo os valores das variáveis que corresponde ás componentes da cor do material. De seguida, com recurso á função *glMaterialfv,* estes valores são atribuídos à respetiva componente difusa, ambiente, especular ou emissiva.

Na implementação realizada, não existe a possibilidade de definir o grau de brilho do material de um objeto. Este valor, que poderia variar entre 0 e 128, é atribuído à primitiva GL_SHININESS sempre com o valor de 80, para qualquer superfície de qualquer modelo.

O seguinte excerto de código procura modelar o processo de ativação das diferentes componentes do material de um dado modelo a desenha.

```
for (Modelo modelo : vecModelos) {
	(…)
	GLfloat dif[4], amb[4], emis[4], spec[4];
	for (int j = 0; j < 4; j++) {
		dif[j] = modelo.getDifusa().at(j);
		amb[j] = modelo.getAmbiente().at(j);
		spec[j] = modelo.getEspecular().at(j);
		emis[j] = modelo.getEmissiva().at(j);
	}
	glMaterialfv(GL_FRONT, GL_DIFFUSE, dif);
	glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
	glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
	glMaterialfv(GL_FRONT, GL_EMISSION, emis);
	glMaterialf(GL_FRONT, GL_SHININESS, 80.0);

	(…)

}
```

Figura 6 – Método para atribuir as propriedades do materiais

2.3. Texturas em OpenGL

As texturas em OpenGL, à semelhança dos VBO's, são objetos que precisam primeiro ser gerados através de uma função.

```
GLuint tex;
glGenTextures(1, &tex);
```

Figura 7 - Função que gera um objeto de textura

As texturas geralmente são utilizadas em modelos 3D, porém é possível utiliza-las noutros contextos, como por exemplo, guardar informações relativas ao relevo topográfico de um terreno. Apesar destas funcionalidades, neste relatório vamos será apenas focado o uso das texturas como imagens a aplicar sobre objetos 3D, sendo os princípios utilizados os mesmos.

Como nos outros objetos, as texturas têm de ser vinculadas para que se possa operar sobre as mesmas, uma vez que as imagens são arrays 2D de pixéis, vão ser vinculadas ao alvo da primitiva GL_TEXTURE_2D.

```
glBindTexture(GL_TEXTURE_2D, tex);
```

Os pixéis da textura podem ser acedidos utilizando as coordenadas de textura durante as opções de desenho. Estas coordenadas têm um alcance entre 0.0 1.0 onde convencionalmente (0,0) é o canto inferior esquerdo e a coordenada (1,1) o canto superior direito.

À operação que usa as coordenadas de textura para recolher a informação para o pixel é denominada *sampling.* Existem várias maneiras diferentes de executar esta operação, cada uma com os seus méritos. O OpenGL disponibiliza várias opções para controlar o modo como esta operação é executada, dos quais destacamos:

Técnica de Wrapping:

O Wrapping consiste em "dizer" como a textura é *sampled* quando as coordenadas ficam fora do alcance de 0.0 a 1.0. Porém, como no desenvolvimento do projeto gerador3d é garantido que a determinação das coordenadas de textura, a utilizar para cada modelo gerado, estão sempre dentro do alcance mencionado, não vamos elaborar este ponto.

Filtering:

Uma vez que as coordenadas de textura são independentes da resolução, elas nem sempre correspondem exatamente ao pixel, isto acontece quando as dimensões da imagem são modificadas em relação ao seu tamanho normal. No OpenGL temos vários métodos para decidir qual vai ser a cor recolhida. Este processo é denominado *filtering* e tem disponíveis as seguintes opções:

- GL_NEAREST: retorna o pixel mais próximo das coordenadas.
- GL_LINEAR: retorna a média ponderada da cor dos 4 pixéis que envolvem as coordenadas.
- GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR: realiza o sampling com recurso a *mipmaps.*

A seguinte imagem procura mostrar as diferenças entre as primitivas GL_NEAREST e GL_LINEAR.

Figura 8 - GL_NEAREST vs GL_LINEAR

A escolha entre um destes métodos dependerá um pouco do contexto sobre o qual será aplicada a textura. Enquanto que a técnica de interpolação linear dá um resultado mais suave (e desfocado) à imagem, se a intenção for imitar um visual mais pixelado, por exemplo para um videojogo, a interpolação que usa o pixel mais próximo é mais apropriada.

Como citado acima, existe outra maneira de filtrar as texturas, através da técnica de *Mipmaps*. Mipmaps são copias mais pequenas da imagem inicial, que pretendemos utilizar. Estas replicações foram recursivamente reduzidas em tamanho e filtradas antecipadamente, sendo assim uma técnica recomendada por resultar numa superior qualidade de imagem com uma penalização muito baixa na performance.

```
glGenerateMipmap(GL_TEXTURE_2D);
```

Para a aplicação de *Mipmaps* existem as seguintes opções:

- GL_NEAREST_MIPMAP_NEAREST: utiliza o mimap que mais se aproxima do tamanho do pixel a ser usado e utiliza a interpolação da coordenada mais próxima;
- GL_LINEAR_MIPMAP_NEAREST: Utiliza o mipmap mais com interpolação linear;
- GL_NEAREST_MIPMAP_LINEAR: Utiliza os 2 mipmaps que mais se aproximam do pixel a ser usado e utiliza a interpolação da coordenada mais próxima;
- GL_LINEAR_MIPMAP_LINEAR: Utiliza os 2 mipmaps mais próximos mas com interpolação linear:

Neste projeto foi utilizada a biblioteca *devil*, para carregar as imagens e utiliza-las como texturas.

2.3.1. Estrutura do XML

A estrutura do *XML* para indicar as texturas é bastante simples, uma vez que o processo para gerar as coordenadas de textura correspondentes aos vértices que vão ser utilizados já foram geradas na aplicação *gerador3d*.

Na estrutura do *XML*, no atributo ***file*** do elemento ***ficheiros*** é apenas referido o ficheiro onde se encontram as informações do modelo a desenhar (coordenadas dos vértices, das normais e das coordenas de textura) e no atributo opcional ***textura*** é indicada designação do ficheiro que vai ser utilizado como textura do modelo.

```
<!-- SOL -->
<group>
   <scale X=23 Y=23 Z=23 />
   <models>
      <model>
         <ficheiros file="esfera.3d" textura="sun.jpg" />
          <material componente="DIFUSA" diffR="1.0" diffG="0.8" diffB="0.0" />
       (…)
```

*Figura 9 – Exemplo do atributo Textura no **xml***

No exemplo apresentado, para o Sol, teremos no ficheiro "esfera.3d" a informação relativa ao modelo da esfera e o ficheiro "sun.jpg" indica a textura a ser utilizada para o a esfera que representa o sol.

2.3.2. Implementação das Texturas

Para implementar as texturas, utilizando os procedimentos das funções da biblioteca *Devil*, inicialmente temos é necessário carregar a imagem, passando-a para memória na forma de um objeto sobre o qual podemos trabalhar.

Depois deste carregamento inicial de todas as texturas que sejam necessárias, quando for necessário utilizar a imagem, para a aplicar a um objeto, este processo é realizado com a função *glBindTexture*. Uma vez indicada qual a textura a utilizar, por cada vértice inserido para ser desenhado, deve ser também indicada a respetiva coordenada de textura associada ao vértice na imagem carregada, de forma a que a textura seja disposta corretamente pelo objeto a desenhar.

```
glBindTexture(GL_TEXTURE_2D, texIDFloor);
glBegin(GL_QUADS);
    glNormal3f(0, 1, 0);
    glTexCoord2f(1, 0);
    glVertex3f(xmax, 0, zmin);
    glTexCoord2f(0, 0);
    glVertex3f(xmin, 0, zmin);
    glTexCoord2f(0, 1);
    glVertex3f(xmin, 0, zmax);
    glTexCoord2f(1, 1);
    glVertex3f(xmax, 0, zmax);
    glBindTexture(GL_TEXTURE_2D, 0);
glEnd();
```

Figura 10- Exemplo de código utilizando textura

Como é possível analisar no exemplo de código acima, após fazer a vinculação do objeto de textura previamente carregado, para cada vértice a ser desenhado, é indicado qual a coordenada de textura correspondente ao mesmo (tal como sucede com as normais, mas que não são objeto de estudo deste capítulo).

Outra forma de utilizar as texturas, e que representa a técnica utilizada no trabalho prático desenvolvido, consiste em utilizar VBOs para guardar a informação relativas às coordenadas de textura que vão ser utilizados.

O método utilizado para preencher estes VBOs pode ser descrito da seguinte forma: à medida que preenchemos o VBO para guardar os vértices de um determinado modelo, preenchemos na mesma ordem um outro VBO que irá ter na mesma posição, as coordenadas de textura a serem utilizada para cada um dos vértices. Por exemplo, se na posição N do VBO dos modelos tivermos os vértices para uma esfera, então na posição N do VBO das normais teremos as normais para cada vértice da esfera e no VBOs das coordenadas de textura, também na posição N, teremos as coordenadas para cada vértice da esfera, tudo isto seguindo a mesma ordem de vértices. Como em qualquer contexto semelhante, estes VBOs têm de ser inicializados antes de começarem a ser preenchidos.

De forma semelhante ao modo como são preenchidos, quando for pretendido desenhar um determinado modelo:

- são ativados os VBOs relativos aos vértices do modelo e ás normais do modelo, na posição associada com o modelo a desenhar;
- são ativadas as coordenadas de textura a aplicar ao modelo, previamente inseridas no respetivo VBO, e indica-se o tipo de dados desse VBO, com a função *gltexCoorPointer;*
- Ativa-se a imagem, com referencia ao id que a representa;
- Desenham-se os dados citados.

O seguinte código tenta exemplificar este processo.

```
//Coordenadas dos vértices
glBindBuffer(GL_ARRAY_BUFFER, VBO_Modelos[indice_VBO_Vertices]);
glVertexPointer(3, GL_FLOAT, 0, 0);

//normais dos vértices
glBindBuffer(GL_ARRAY_BUFFER, VBO_Normais[indice_VBO_Vertices]);
glNormalPointer(GL_FLOAT, 0, 0);

if ((modelo tem textura) {
	//apontador para o objeto em memória que representa a imagem carregada
inicialmente
	glBindTexture(GL_TEXTURE_2D, vecIdsTextura[indice_IdTextura]);

	//coordenadas de textura a utilizar no modelo para a imagem carregada
	glBindBuffer(GL_ARRAY_BUFFER, VBO_Texturas[indice_VBO_Texturas]);
	glTexCoordPointer(2, GL_FLOAT, 0, 0);

	//Desenhar os dados processados
	glDrawArrays(GL_TRIANGLES, 0, modelo.getNrCoordText() );
	glBindTexture(GL_TEXTURE_2D, 0);

	indice_VBO_Texturas++;
	indice_IdTextura++;
}
```

Figura 11 – Desenho de um objeto, utilizando VBOs para a textura

Mockup do código utilizado na implementação:

Gerador:

- Para cada modelo a gerar, são calculadas as coordenadas de textura e guardadas no ficheiro gerado, depois da escrita dos vértice e do número de normais;

Motor:

- Fase de parsing:
  As coordenadas de textura escritas no ficheiro .3d são lidas, e guardadas numa variável do tipo vetor, dentro da classe Modelo.
  Caso no xml de input exista a referência a um ficheiro de textura, a sua designação é guardada como string, também dentro da classe Modelo.
- Fase de inicialização:

```
Índice = 0;
for(grupos carregados)
  for (modelos em cada grupo)
        If(tem textura)
             VBO_CoordTextura [índice++] =
                  parseCoordTextura(Texture_fileName);

for(grupos carregados)
    for(modelos em cada grupo)
       inicializa e preenche VBO's de vértices e normais,
       if(tem textura)
            inicializa e preenche VBO's das texturas.
```

- Fase de desenho:

```
Índice = 0, ind = 0;
for(grupos carregados)
      for(modelos em cada crupo)
           If(tem textura){
                Glbind(VBO_Vertices[ind++])
                Glbind(VBO_Normais[ind++]
                Glbind(VBO_CoordTextura[índice++])
                Desenha utilizando a textura;
           }
```

3. Aplicação gerador3d

3.1. Formato dos ficheiros gerados

Para esta última fase do projeto foi necessário remodelar a aplicação designada como gerador3d, partindo da sua implementação para as fases anteriores.

Com a necessidade de conhecer as normais e das coordenadas de textura dos vértices do modelo, estes parâmetros passam a ser definidos no gerador, aquando a determinação dos seus vértices do modelo. Por este motivo, foi alterada a estrutura dos ficheiros criados no sentido de incluir o registo destes novos parâmetros associados com modelo gerado.

A abordagem ao calculo das normais de um vértice e da determinação das respetivas coordenadas de textura serão temas abordados nas secções seguintes deste capitulo.

Assim, o formato do ficheiro *.3d* gerado, que nas fases anteriores continha apenas informações relativas aos vértices do modelo, passa agora a ter a seguinte estrutura, inspirada no formato dos Patch de *Bezier* já trabalhos:

- Na primeira linha é escrito o número de triângulos do modelo;
- Nas linhas seguintes são escritas as informações de cada triângulo do modelo. Cada linha contem informação relativa a um triangulo, estando as coordenadas de cada vértice separadas por espaço e os vértices separados por virgulas;
- Depois dos vértices é escrito o número de normais determinadas para o modelo. Este número de normais será 3 vezes o número de vértices do modelo, escrito na primeira linha. Por cada linha é escrito uma normal, cujas coordenadas estão separadas por espaço;
- Depois das normais, é escrito o número de coordenadas de textura do modelo;
- Por cada linha, são escritos os valores de *u* e *v*, separados por espaço. Estes valores representam as coordenadas de textura para cada vértice.

Como no processo de geração de um modelo, sempre que se determinada um vértice é na mesma altura determinada a sua normal e a sua coordenada de textura, quando estes dados forem escritos no ficheiro, vão estar ordenados. Por exemplo, o primeiro vértice escrito terá as informações da sua normal na primeira normal escrita e as suas coordenadas de textura na primeira coordenada de textura lida do ficheiro.

De forma análoga e recorrente, o vértice que esteja na posição *N*, terá a sua normal e a sua coordenada de textura também na posição *N* das respetivas variáveis. A seguinte imagem procura exemplificar o formato do ficheiro gerado para um determinado modelo.

```
NT     Nr de Triângulos
Tri1 v1.x v1.y v1.z, v2.x v2.y v2.z, v3.x v3.y v3.z
Tri2 v1.x v1.y v1.z, v2.x v2.y v2.z, v3.x v3.y v3.z
. . .
TriNT v1.x v1.y v1.z, v2.x v2.y v2.z, v3.x v3.y v3.z
NN     Nr de Normais
Norm1 n.x n.y n.z
Norm2 n.x n.y n.z
. . .
NormNN n.x n.y n.z
NT     Nr de coordenadas de textura
Coord1  u v
Coord2  u v
. . .
CoordNT u v
```

*Figura 12 – Formato dos ficheiros **.3d** gerados*

3.2. Calculo das Normais de um modelo

Para a implementação do processo de iluminação em OpenGL, passou a ser necessário determinar quais as normais dos vértices que compõe um determinado modelo. Sem este novo tipo de informação, não era possível aplicar de forma correta as diferentes componentes da luz, referidas no capitulo anterior onde se aborda a aplicação sistemaSolar.

Este calculo de determinação das normais foi aplicado com detalhe aos seguintes modelos:

- Esfera:
  Por ser a forma que representa a maioria dos objetos do sistema solar criado foi o primeiro modelo sobre o qual se desenvolveu o calculo das normais. Pelo facto da esfera ser desenhada centrada na origem, segundo coordenadas esféricas, a normal de cada vértice é igual ao vetor com as coordenadas do próprio vértice.
- Cubo:
  As normais variam conforme a face a desenhar.
  Todos os vértices da face superior irão ter como normal o vetor *up* = (0,1,0) e os da face inferior o vetor -*up* = (0,-1,0);
  Os vértices da face frontal terão como normal o vetor ***i*** = (1,0,0) e os da face traseira o vetor -***i*** = (-1,0,0);
  Os vértices da face direita terão como normal o vetor ***j*** *= (0,0,1) e por fim, os da face esquerda o vetor -****j*** *= (0,0,-1).*
- Plano:
  Como é regra que o plano esteja contido no plano xOz, garante-se assim que as normais de cada vértice do plano será sempre o vetor *up* = (0,1,0);
- Cilindro:
  Como o algoritmo para determinar os vértices do cilindro itera stack a stack, considerando o modelo sempre centrado na origem, a normal dos vértices que compões a superfície lateral do cilindro serão sempre iguais às coordenadas do próprio vértice. Todos os vértices contidos no circulo da face superior irão ter como normal o vetor *up* = (0,1,0) e os do circulo da face inferior o vetor -*up* = (0,-1,0);
- Cone:
  Por motivos semelhantes aos citados para determinar as normais do cilindo, a normal dos vértices do plano lateral serão iguais às coordenadas do próprio vértice. No caso dos vértices que pertencem ao circulo da face inferior, a normal destes será o vetor -*up* = (0,-1,0).

Infelizmente, para o modelo gerado através de Patchs de *Bezier*, as normais de cada vértice que se determine irão ter as mesmas coordenadas que o próprio vértice. Este procedimento, que para objetos como a esfera está correto, não serão o mais apropriado para outros modelos. Contudo, pela impossibilidade de determinar que modelo está representado num determinado patch, esta foi a solução encontrada.

Para demonstrar que o processo de calculo das normais implementado está consideravelmente correto, estará em anexo, na secção “Normais e iluminação de Objetos”, um conjunto de imagens que representam cada um dos modelos geométricos citados e criados, expostos sobre um foco de luz.

3.3. Calculo das Coordenadas de Textura

Para ser possível a aplicação de texturas aos objetos, na aplicação *engine* sistemaSolar, é necessário conhecer as coordenadas de textura de cada vértice que compõe o modelo a desenhar. Assim, mesmo que um determinado objeto não venha a ser posteriormente "coberto" por uma textura, este processo de calculo das coordenadas de textura de cada vértice passa a ser processado também na aplicação gerador3d, sendo posteriormente escrito no ficheiro .3d, tal como explicado na primeira secção deste capitulo.

No processo de calcular as coordenadas de textura de um determinado vértice, considerou-se que a imagem da textura seria um quadrado, de lado 1. Com estas considerações, e conforme o modelo a processar, foram feitos os cálculos necessário para uma correta aplicação da textura no modelo.

O calculo das coordenadas de textura foram apenas implementadas para praticamente todos os modelos possíveis de criar no *gerador3d*, à excepção do modelo gerado pelo patch de *Bezier* e ao modelo do cone, que por estar implementado segundo coordenadas cilíndricas exigia um tempo, não existente, para refazer o seu algoritmo.

Dos modelos sobre os quais se implementou esta funcionalidade, estará em anexo, na secção "Texturas em modelos gerados" alguns exemplos dos resultados obtidos.

3.3.1. Coordenadas textura plano

A geração dos vértices de um plano está implementada assumindo que o plano será um quadrado assente sobre o eixo xOz, bastando apenas dois triângulos para o definir, não havendo assim a possibilidade do plano ser composto por N divisões.

Deste modo, sendo o plano um quadrado e sendo a textura vista também como um quadrado de lado 1, para determinar as coordenadas de textura dos quatro vértices do plano basta apenas atribuir a cada um deles, o equivalente vértice da textura.

A seguinte imagem procura descrever este procedimento.

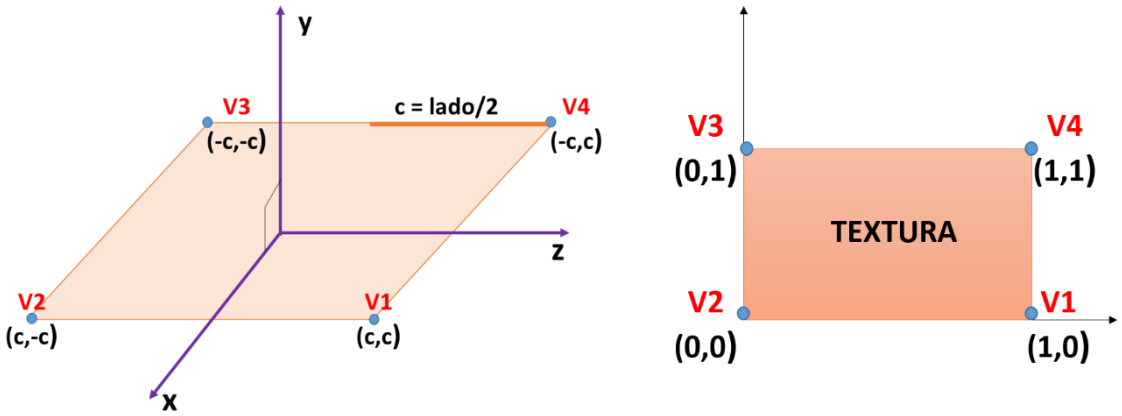

Figura 13 – Coordenadas Textura Plano

3.3.2. Coordenadas textura cubo

Numa abordagem inicial, as coordenadas de textura para cada vértice do cubo foram calculadas seguindo um procedimento semelhante ao utilizado para o plano.

Desta forma, se o cubo tivesse apenas uma divisão, ou seja, as suas faces fossem compostas apenas por dois triângulos, cada face iria repetir a textura a aplicar.

Se cubo tivesse N divisões, essas N divisões iriam repetir a textura, ficando assim com um aspeto de papel de parede repetido. A seguinte imagem procura representar este procedimento, que para algumas texturas e para determinados contextos, poderia ser o processo recomendado.

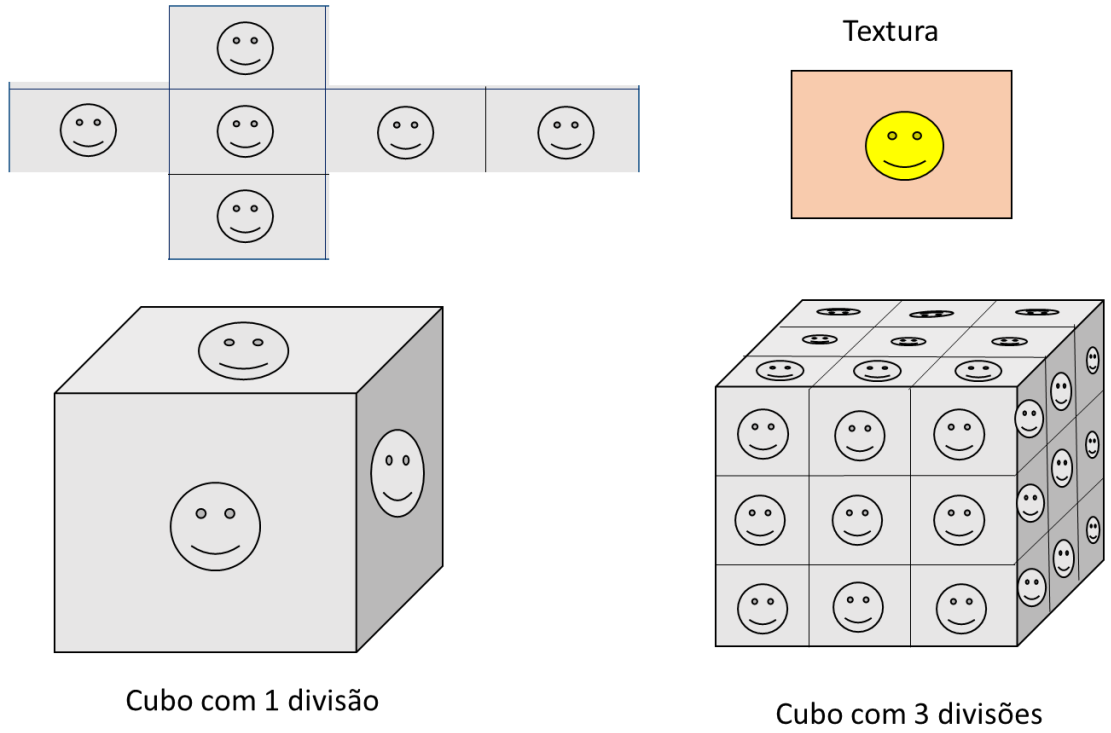

Figura 14 – Replicação de uma textura pelas divisões do Cubo

Posteriormente, foi tomada a decisão de não aplicar a textura igualmente a cada face/subdivisão do cubo, mas sim aplicar uma textura como um "cobertor", interpolando cada coordenada de textura conforme a face que se pretende aplicar parte da textura. O seguinte esquema procura representar este procedimento, para o caso de um cubo com apenas uma divisão. As imagens que representam a textura aplicada a um cubo, utilizando cada um dos processos referidos, estarão colocadas em Anexos.

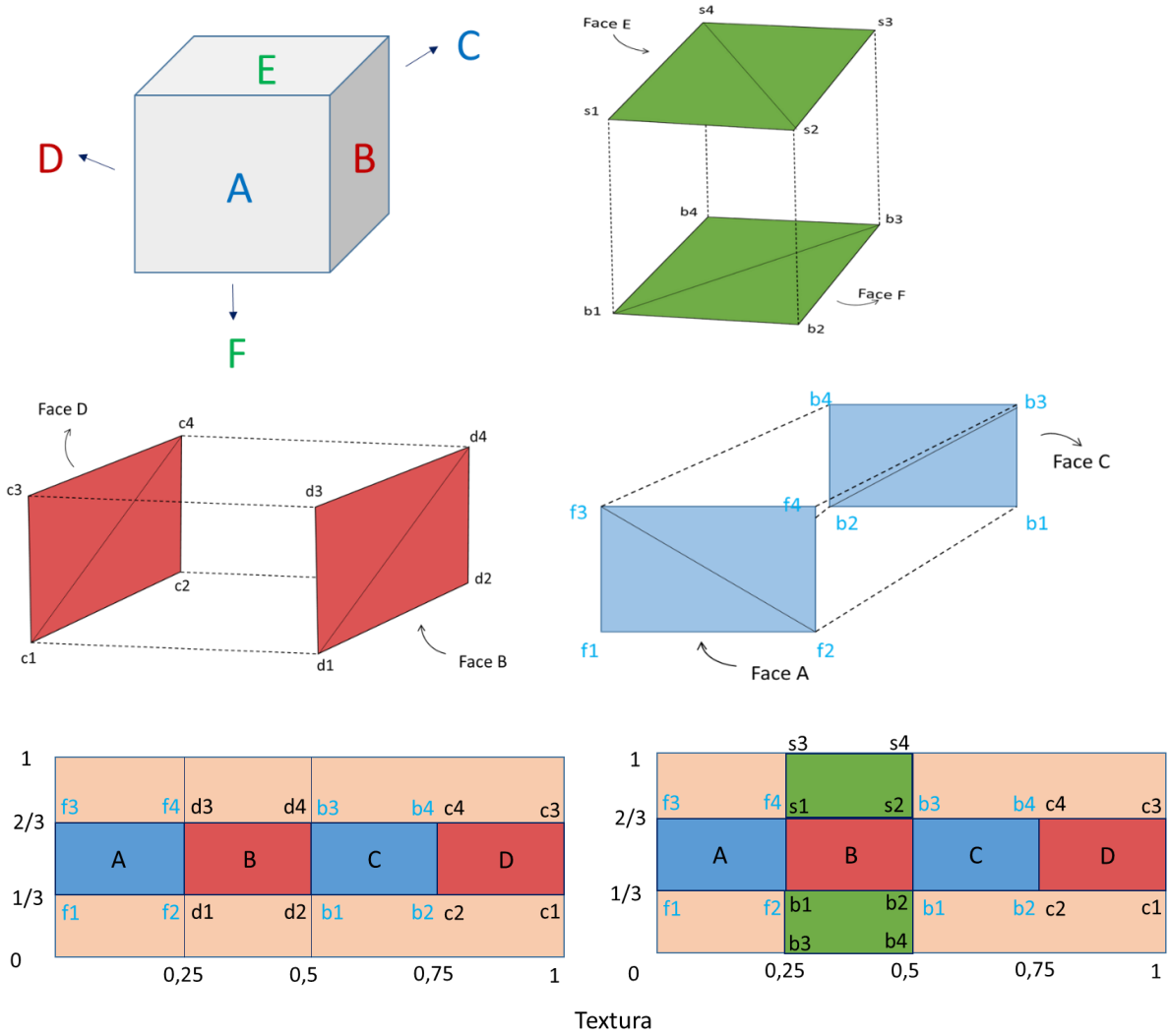

Figura 15 – Interpolação das coordenadas de textura para um cubo

Para aplicar a textura "em forma de dado" foi necessário ter em conta o modo como o cubo é desenhado e a forma como pretendemos selecionar as coordenadas na imagem da textura.

Por exemplo, no desenho da face **B**, o gerador3d processa esta face de trás para à frente e de baixo para cima, ou seja, de d1 para d2 e de d2 para d4. Deste modo, as coordenadas de textura são inversas, ou seja, vamos iterar a imagem no sentido contrário, começando em (0.5, $\frac{1}{3}$) e diminuindo os valores para o eixo das abcissas até 0.25 e subindo os valores no eixo das ordenadas até $\frac{2}{3}$, garantindo assim que vamos buscar à imagem os pontos de textura pretendidos.

Na face A, a iteração para obter as coordenadas de textura é semelhante a que é efetuada para obter os pontos a serem desenhados, de notar que as coordenadas de textura são inicializadas em (0, $\frac{1}{3}$) e a iteração para as coordenadas de textura tem um passo de $\frac{1}{nr\ divisões\ cubo}$.

Usando estes princípios e aplicando-os para as restantes faces do cubo conseguimos então obter as coordenadas de textura pretendidas.

3.3.3. Coordenadas textura esfera

Para o processo de determinação dos vértices de uma esfera, são consideradas coordenadas esféricas, onde **β** representa o angulo sobre o eixo do Y e **α** representa o angulo sobre o eixo do X (os do Z, conforme o ponto de vista).

O algoritmo construído começa assim por iterar o angulo **β**, iniciado com o valor de $\frac{\pi}{2}$ e iterado até $-\frac{\pi}{2}$, para construir as stacks, onde a construção de uma nova stack acrescenta a **β** um "passo" de $\frac{\pi}{stacks}$. Por cada stack, o angulo **α** é percorrido desde 0 até 2π, em "passos" de $\frac{2*\pi}{fatias}$, calculando assim os vértices dos triângulos que compõe as fatias daquela stack.

Como nesta implementação a esfera pode ser gerada com a noção de stacks e fatias, para lhe ser aplicada uma textura é também necessário particionar a imagem da textura.

Considerando a coordenas de textura na forma (**u**,**v**), e pelo facto da textura ser vista como um quadrado de lado 1, então, por cada passo no angulo **β**, o passo equivalente na coordenada **v** da textura será $\frac{1}{stacks}$, e, por cada passo no angulo **α**, o respetivo passo na coordenada de textura **u** será $\frac{1}{fatias}$.

Como na esfera o eixo dos Y é iterado do sentido positivo para o negativo mas nas coordenadas de textura **v** é iterado desde 0 até 1 (sentido inverso), para evitar que a textura aplicada ficasse invertida (sendo um erro visível em texturas especificas), as coordenadas de textura **v** foram invertidas, utilizando-se (1-**v**).

Desta forma, a imagem da textura aplicada à esfera é feita de forma correta e elegante. A seguinte imagem procura descrever este procedimento:

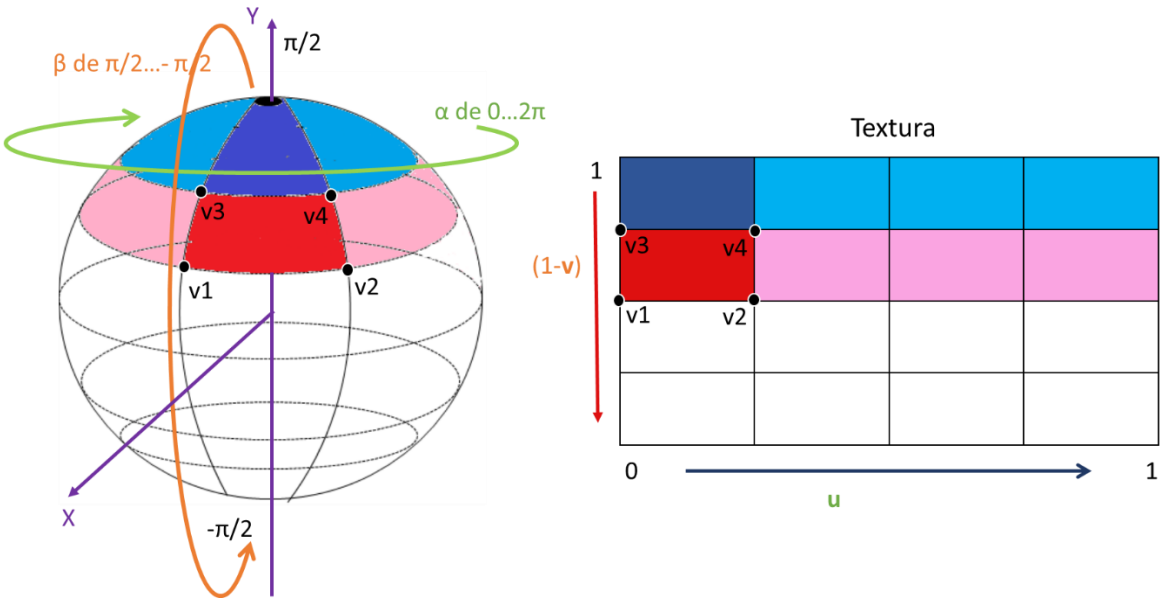

Figura 16 – Interpolação coordenadas textura esfera

3.3.4. Coordenadas textura cilindro

Para determinar as coordenas de textura dos vértices de um cilindro, foi assumido que a textura a aplicar ao modelo gerado usaria a mesma forma que a figura utilizada nas aulas práticas para o mesmo efeito.

Deste modo, a parte lateral do cilindro pode ser vista como um plano, onde para cada um dos seus vértices, a coordenada de textura **u** estará entre [0,1] e a coordenada **v** entre [0.375,1]. O processo de calculo da coordenada de textura para cada vértice segue assim o mesmo conceito aplicado para o plano, para para intervalos de **u** e **v** diferentes.

Para as duas faces circulares do cilindro, e considerando β o angulo que define o vértice, foi assumido que:

- O vértice central da face superior, Vsup = (0, altura/2, 0), terá como coordenada de textura (u,v) = (0.4375, 0.1825);
- Para os restantes vértices que definem a face circular superior, as suas coordenadas de textura serão:
  u = 0.4375 + 0.1875*sin(β) e v = 0.1875 + 0.1875*cos(β);

- O vértice central da face inferior, Vinf = (0, -altura/2, 0), terá como coordenada de textura (u,v) = (0.8125, 0.1825);
- Para os restantes vértices que definem a face circular inferior, as suas coordenadas de textura serão:
  u = 0.81255 + 0.1875*sin(β) e v = 0.1875 + 0.1875*cos(β).

Tal como na esfera, pelo facto do algoritmo criado para calcular os vértices do cilindro iterar desde 0 a 2*π, mas a coordenava de textura **u** crescer desde 0 até 1 (no sentido inverso), para evitar que a imagem ficasse invertida utilizou-se para a coordenada de textura **u,** nos vértices da face lateral, o valor (**1-u**).

A seguinte imagem procura descrever o processo de determinação de texturas:

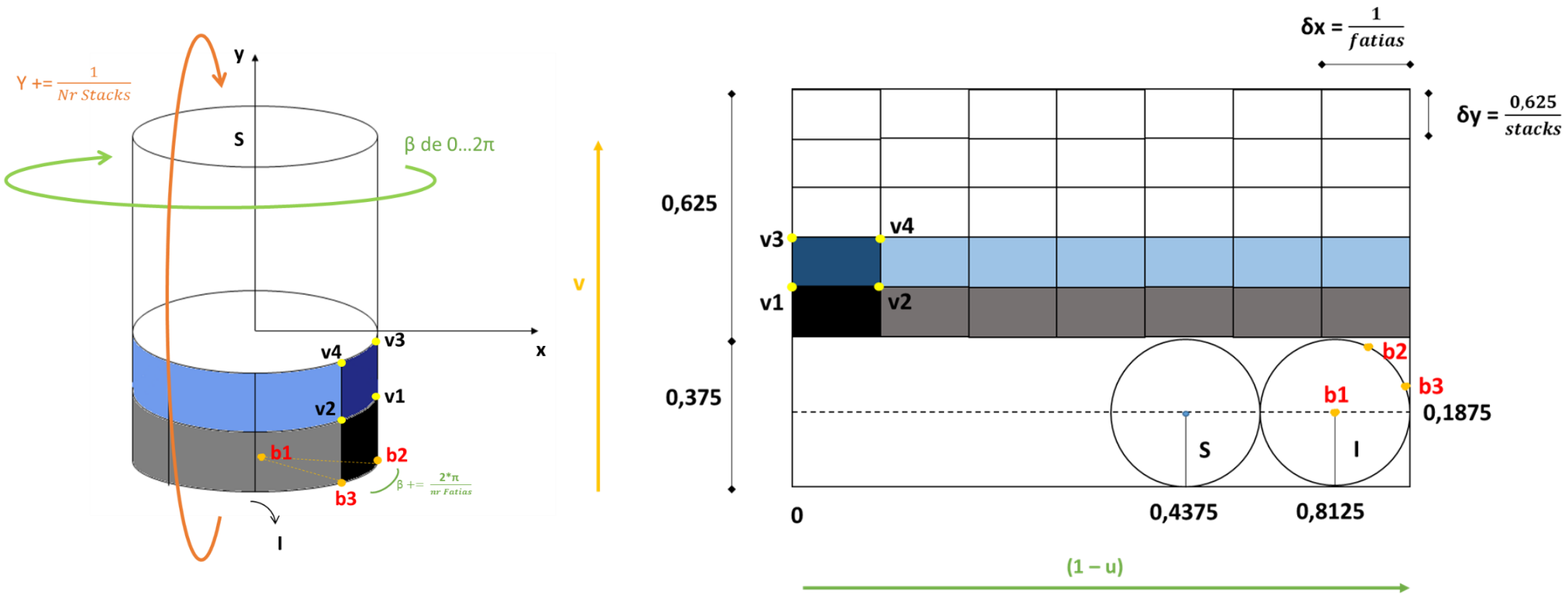

Figura 17 – Interpolação coordenadas textura cilindro

4. Conclusões

Durante a realização deste trabalho, foi possível colocar em pratica todo o conhecimento adquirido nas aulas, realizando os requisitos propostos para esta última fase.

As principais dificuldades encontradas durante a realização do trabalho estarão associadas com o desenvolvimento e implementação das texturas aos diferentes objetos do sistema solar.

No nosso entender, todas as eventuais dificuldades foram totalmente ultrapassadas e compreendidas em grupo, tendo conseguido utilizar todos os conceitos aprendidos, cumprindo assim todos os requisitos básicos do enunciado do projeto.

Para demonstrar as funcionalidades, apresentamos uma maquete do sistema solar completa, enriquecida com elementos como luas; cores dos planetas através da implementação das propriedades do material da sua superfície; processos de iluminação e aplicação de texturas.

Mantemos ainda na apresentação final o modelo criado através de um Patch de *Bezier* e as rotações dos objetos do sistema solar, segundo uma órbita criada com uma curva de Catmull-Rom, em torno do sol.

No final desta fase, referimos ainda a boa reutilização das estruturas e classes já utilizadas nas fases anteriores.

Num sentido de valorização, focamos a implementação do cálculo das normais para todos os modelos previamente implementados na aplicação gerador3d, e o calculo das coordenadas de textura para o modelo do plano, cubo e esfera.

Sobre este assunto das normais, salientamos ainda os testes de iluminação realizados a cada um dos modelos possíveis de gerar, de forma a demonstrar o calculo das normais do modelo e a correta aplicação das técnicas de iluminação. Estes resultados estão disponíveis em Anexos, na seção *"Normais e iluminação de Objetos".*

5. Referências

Lighthouse3d. (2017). GLUT Tutorial. [online] Available at:
http://www.lighthouse3d.com/tutorials/glut-tutorial/
[Accessed 14 May 2017].

Learnopengl.com. (2017). *Learn OpenGL, extensive tutorial resource for learning Modern OpenGL*. [online] Available at:
https://learnopengl.com/#!Lighting/Basic-Lighting
[Accessed 10 May 2017].

Learnopengl.com. (2017). *Learn OpenGL, extensive tutorial resource for learning Modern OpenGL*. [online] Available at:
https://learnopengl.com/#!Lighting/Materials
[Accessed 12 May 2017].

Sidelnikov, G. (2017). *OpenGL Light Tutorial For Beginners*. [online] OpenGL Tutorials. Available at:
http://www.falloutsoftware.com/tutorials/gl/gl8.htm
[Accessed 12 May 2017].

Open.GL (2017). Textures objects and parameters. [online] Available at:

https://open.gl/textures
[Accessed 14 May 2017].

6. Anexos

6.1. Iluminação do sistema solar

As seguintes imagens procuram exemplificar o resultado obtido através da iluminação do sistema solar. Para a cena apresentada, é utilizada apenas um único foco de luz, comum a todos os objetos do sistema solar. Além disto, foram definidas todas as componentes da luz assim como todas as componentes do material dos objetos.

Para criar um efeito de sombras mais ténue, foram utilizadas componentes da luz ambiente que correspondessem a uma gradação mais escura da cor atribuída à luz difusa do mesmo objeto.

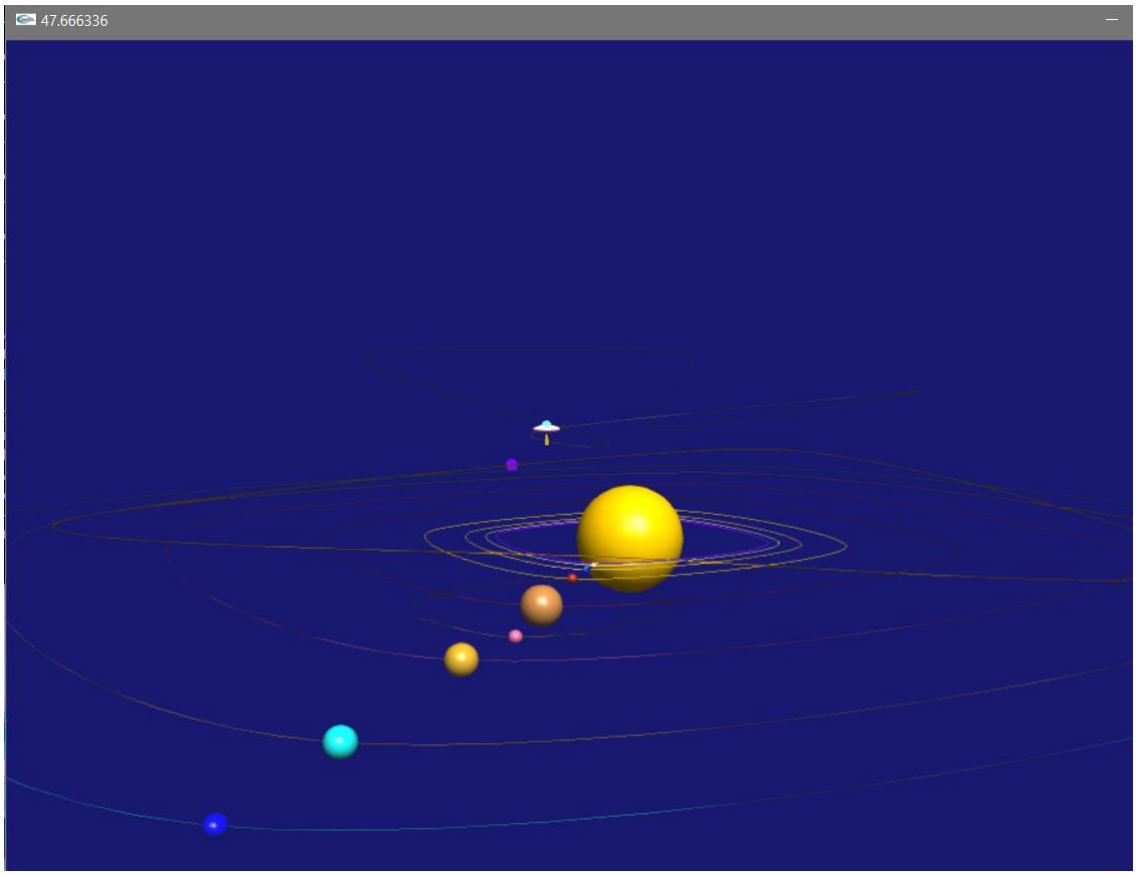

Figura 18 – Sistema solar com iluminação direcional – I

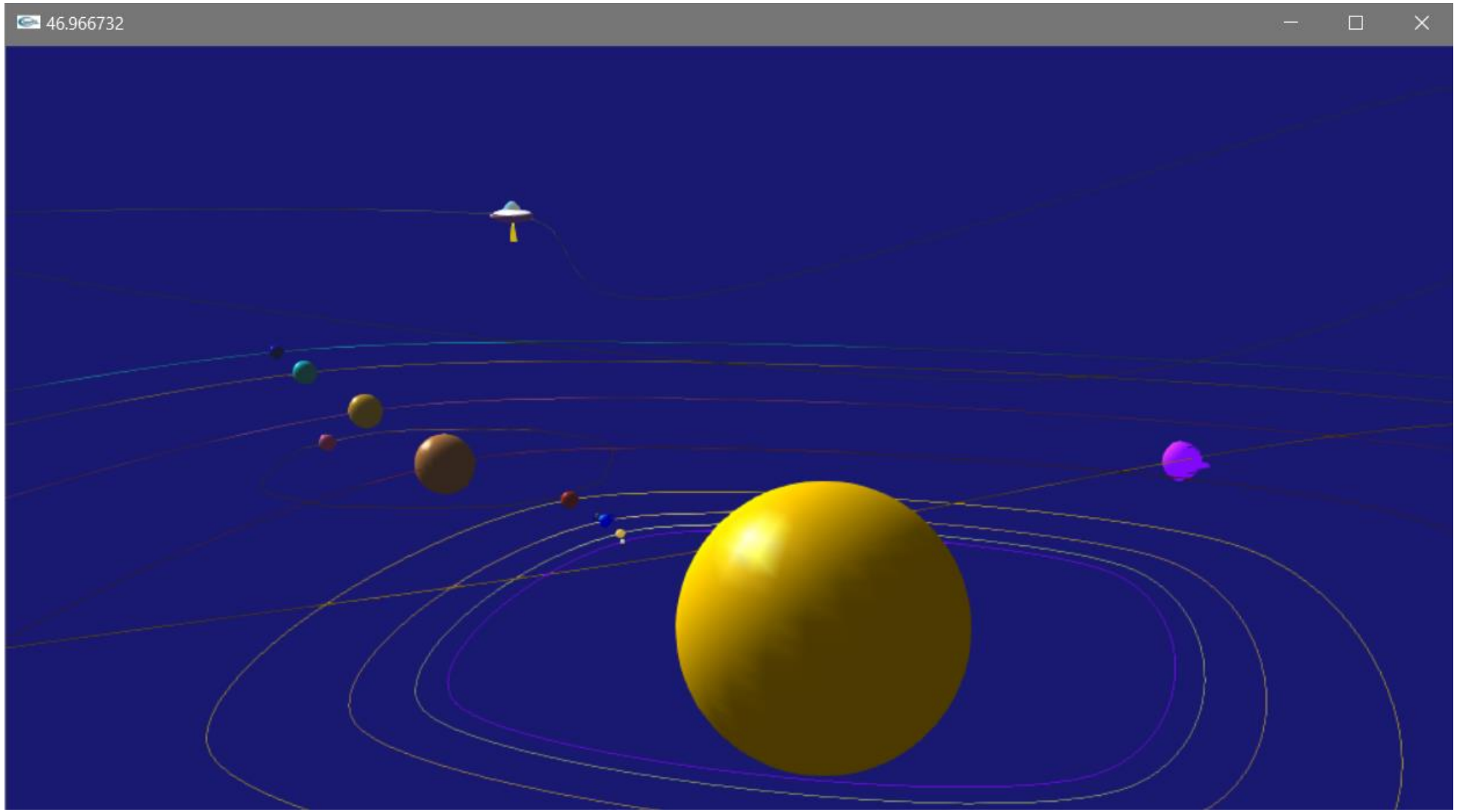

Figura 19 – Sistema solar com iluminação direcional - II

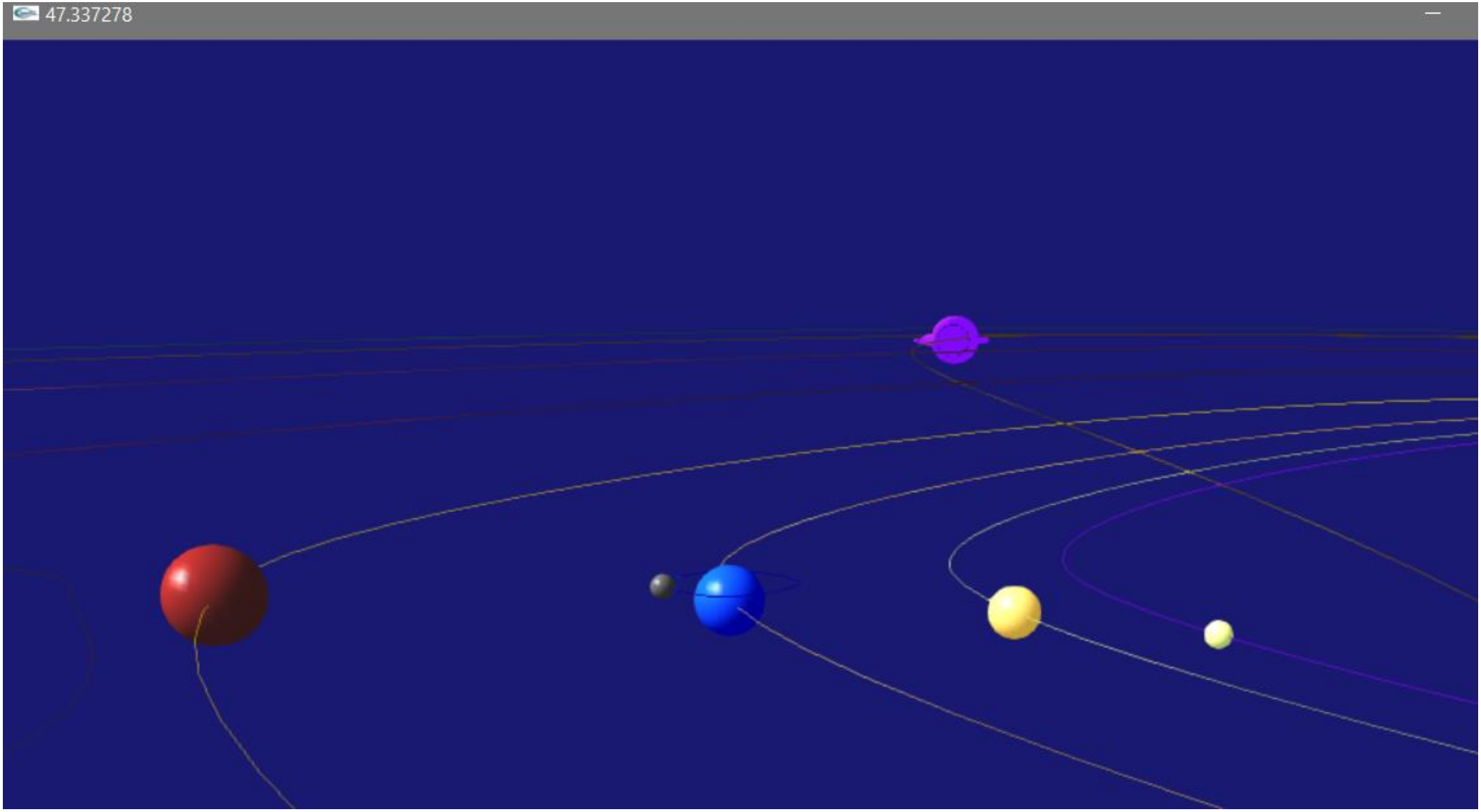

Figura 20 – Sistema solar com iluminação direcional - III

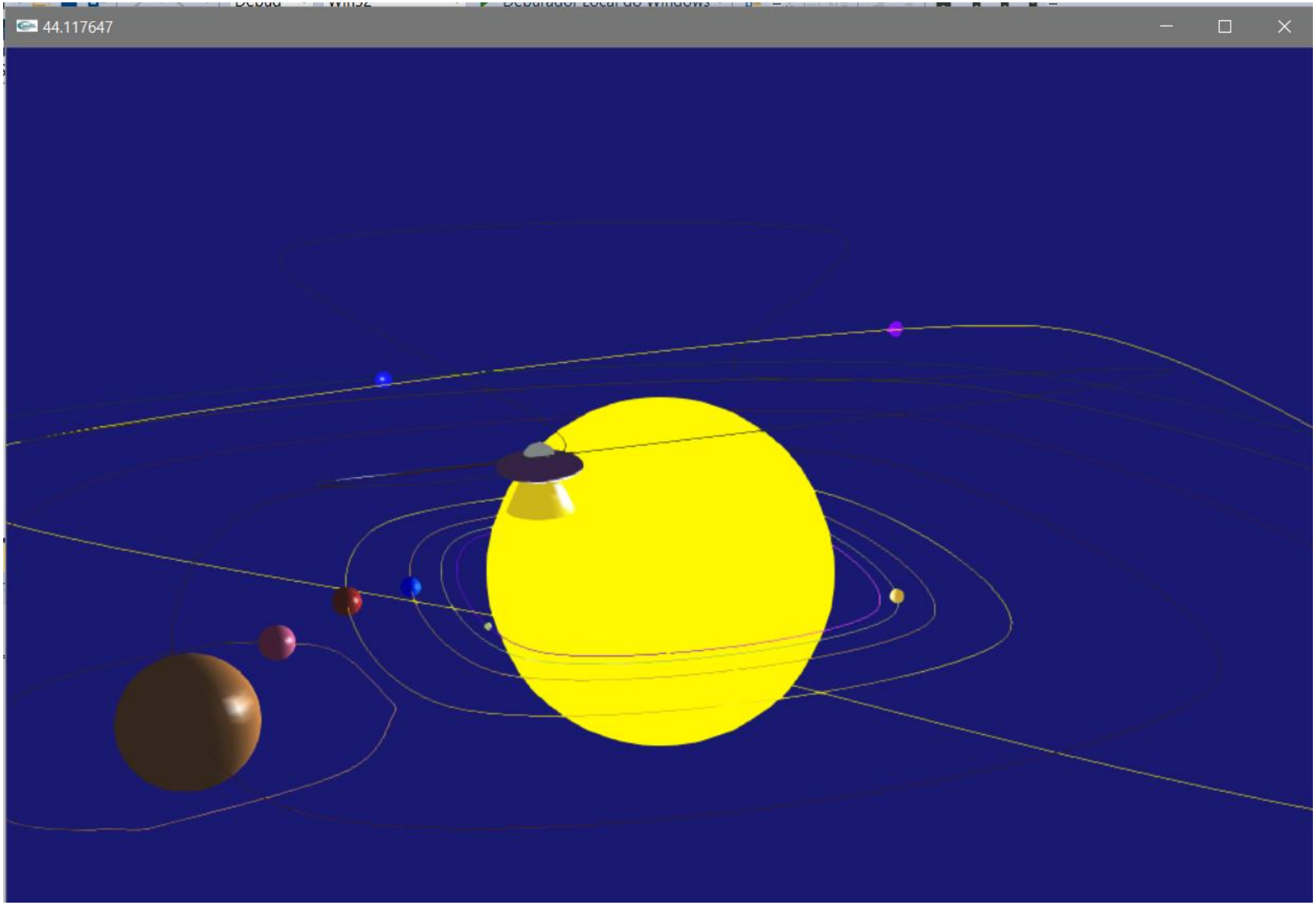

Figura 21 – Sol como ponto de luz - I

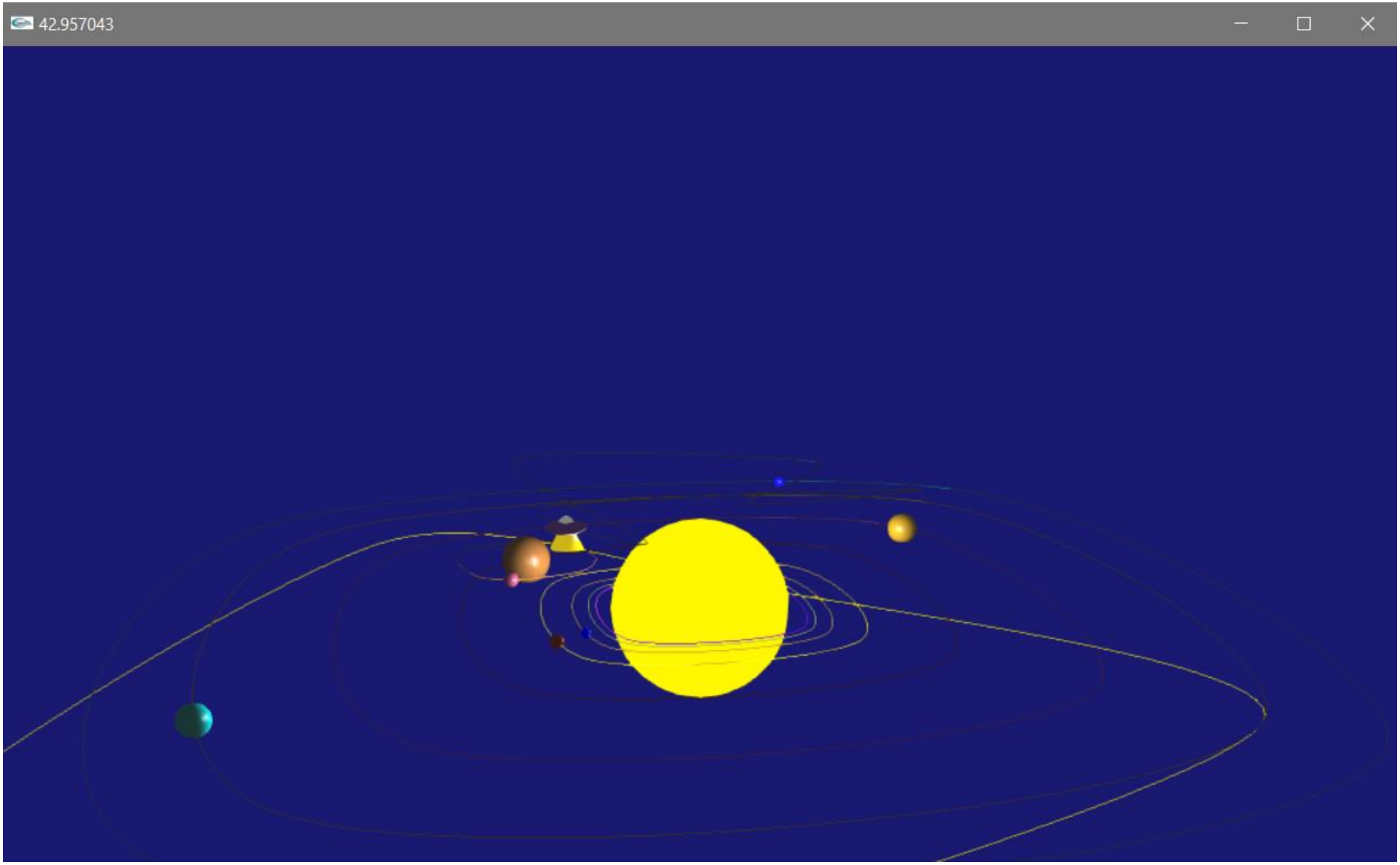

Figura 22 - Sol como ponto de luz - II

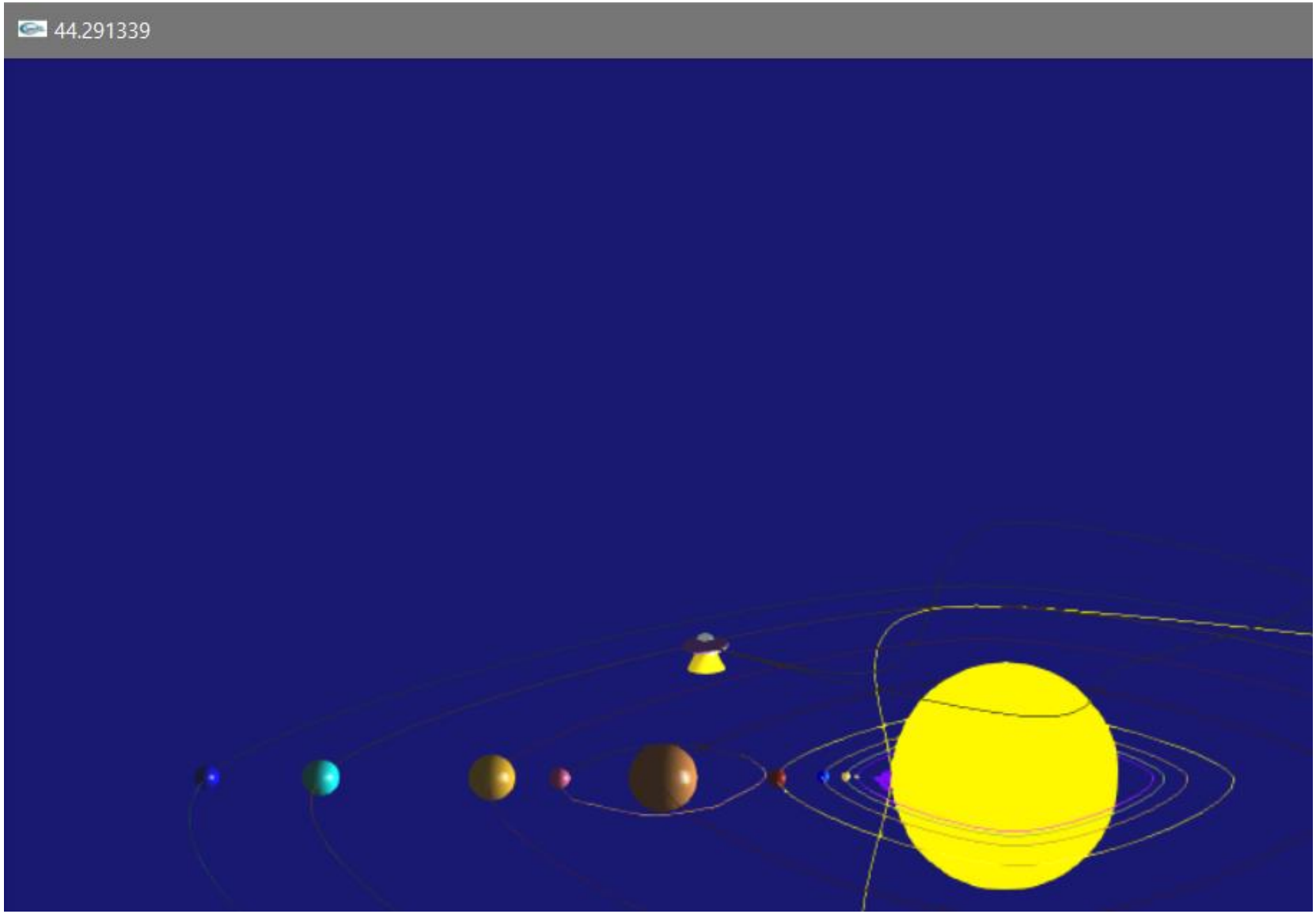

Figura 23 - Sol como ponto de luz - III

6.2. Texturização do sistema solar

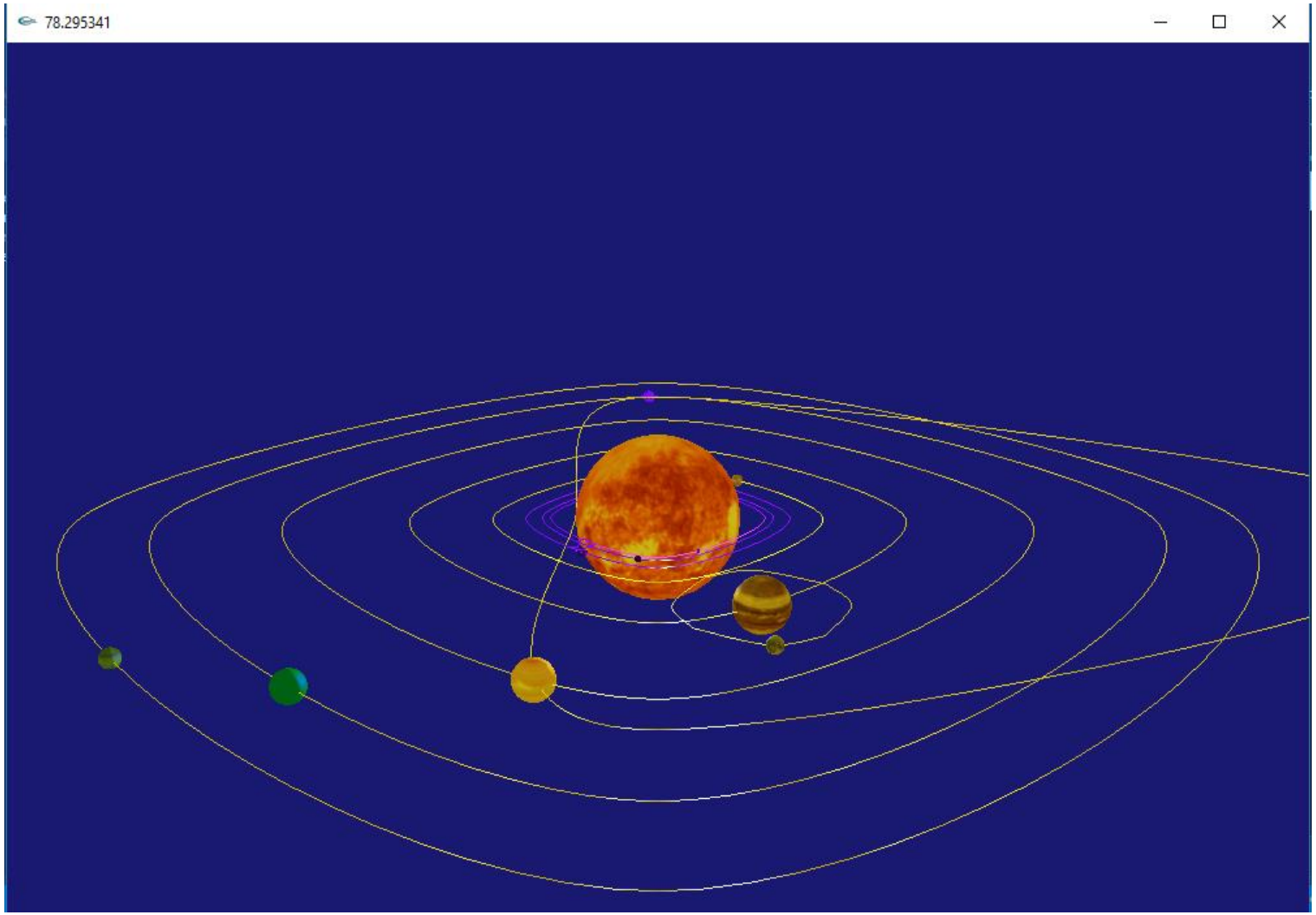

Figura 24 – sistema solar com texturas - I

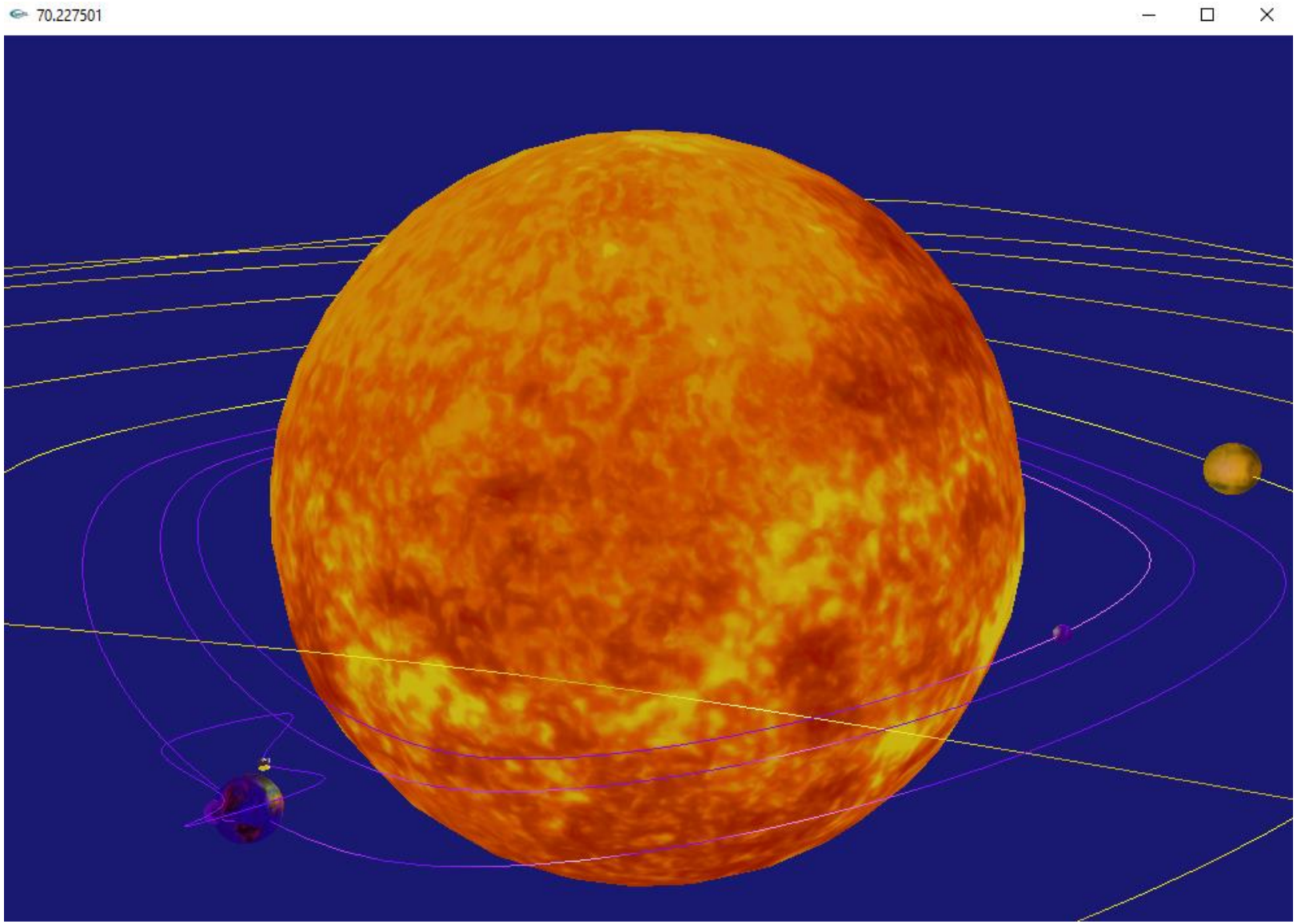

Figura 25 - – sistema solar com texturas - II

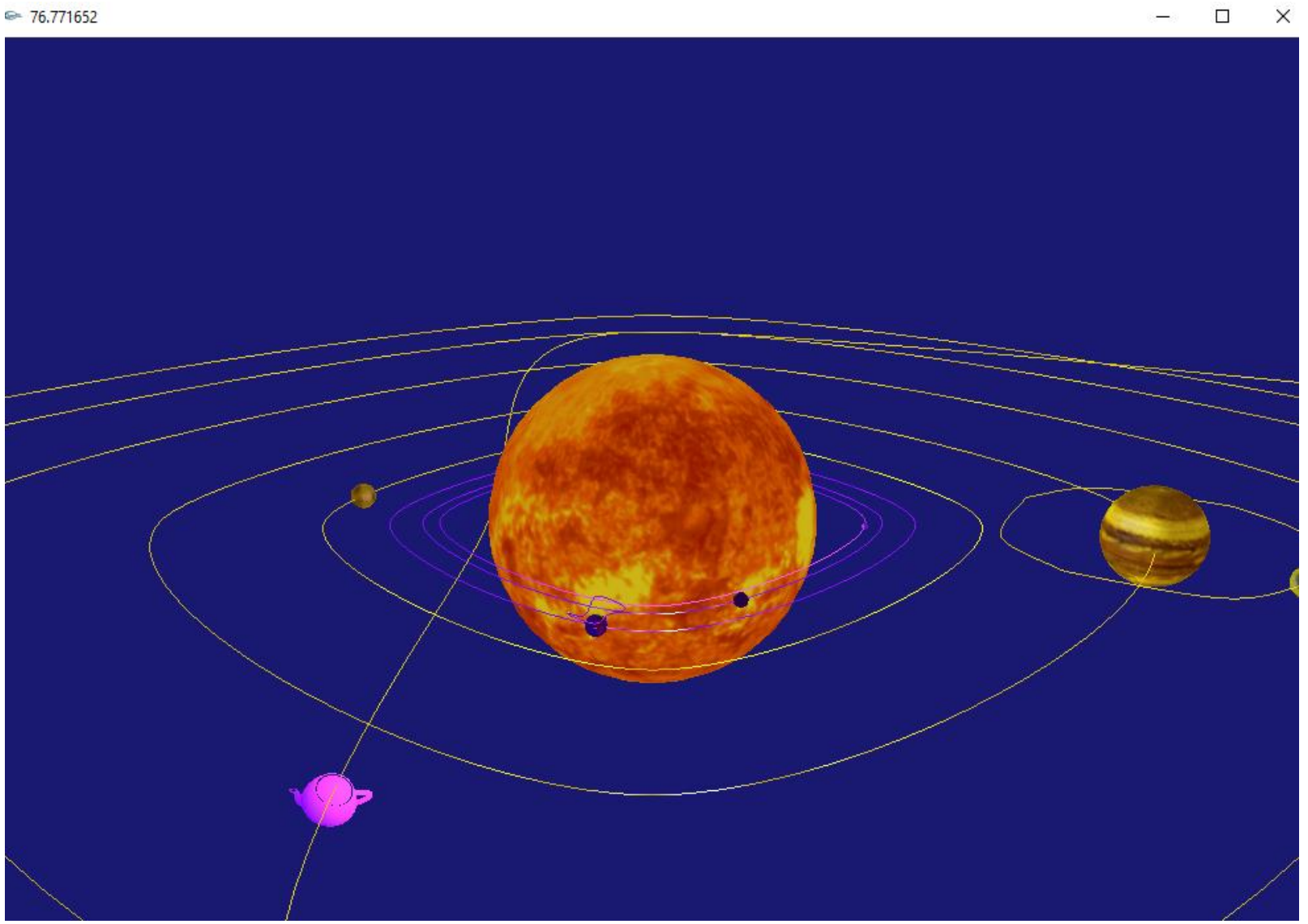

Figura 26 - – sistema solar com texturas - III

6.3. Normais e iluminação de Objetos

Como forma de validar a implementação do calculo das normais de um modelo, determinadas aquando o processo de calculo dos seus vértices na aplicação gerador3d, foi gerado um novo ficheiro para cada um dos modelos possíveis de calcular e, posteriormente, cada um destes ficheiros/modelos foi testados usando o *engine* sistemaSolar.

Para isto, utilizou-se um foco de luz acima do objeto, ligeiramente inclinada. Para cada modelo, foi tirado um conjunto de imagens que permita ver o aspeto do modelo quando voltado diretamente para a luz e quando visto de um ponto oposto à direção da luz, ou seja, uma zona do modelo que esteja à sombra. As seguintes imagens foram assim produzidas no decorrer deste trabalho, e não são exemplos retirados de um outro local.

```
<?xml version="1.0" encoding="UTF-8"?>
<scene>
   <diretoria dir="C:\Users\migue\OneDrive\Trabalho CG\Fase
IV\sistemaSolar\Ficheiros\" />
   <luzes>
      <luz>
         <tipo componente="VECTOR" posX="-5.0" posY="15.0" posZ="5.0" />
         <tipo componente="DIFUSA" diffR="1.0" diffG="1.0" diffB="1.0" />
         <tipo componente="AMBIENTE" ambR="0.0" ambG="0.0" ambB="0.0" />
      </luz>
   </luzes>
   <group>
   <!-- modelo -->
      <models>
         <model>
            <ficheiros file="modelo.3d" />
            <material componente="DIFUSA" diffR="1.0" diffG="0.8" diffB="0.0"
/>
            <material componente="AMBIENTE" ambR="1.0" ambG="0.647" ambB="0.0"
/>
            <material componente="EMISSIVA" emisR="0.1" emisG="0.1" emisB="0.0"
/>
         </model>
      </models>
   </group>
</scene>
```

Figura 27 – Exemplo do xml usado para os testes de iluminação

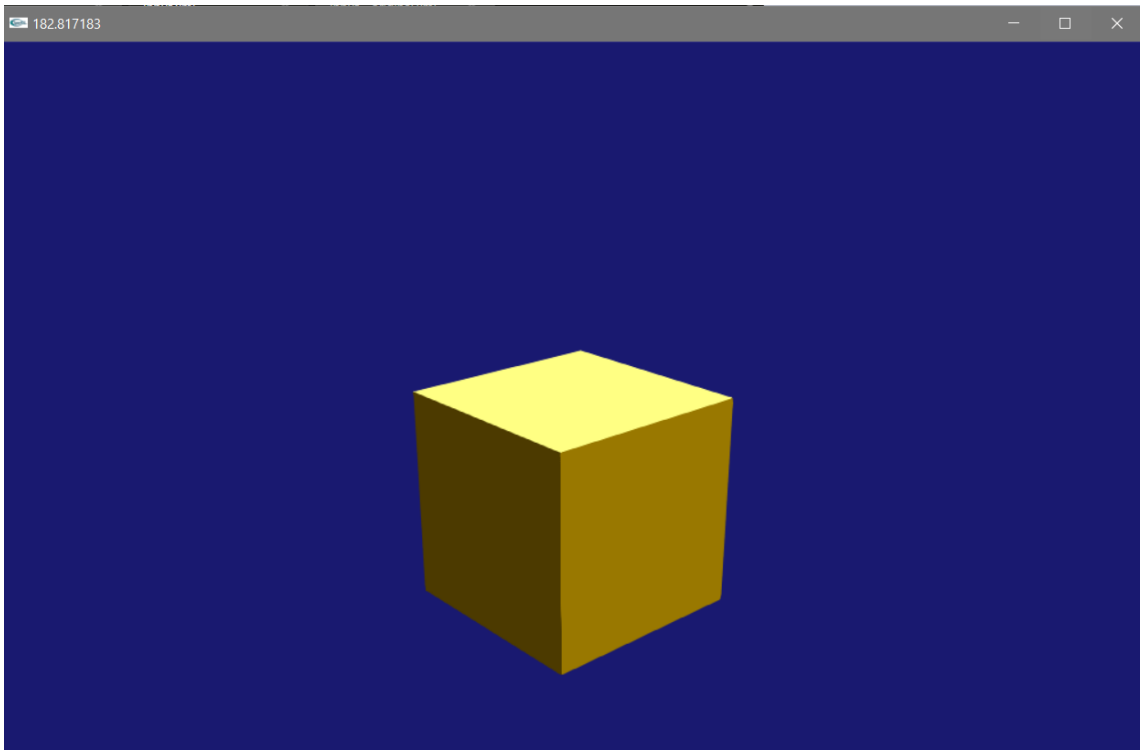

Figura 28 – Iluminação sobre um cubo - *I*

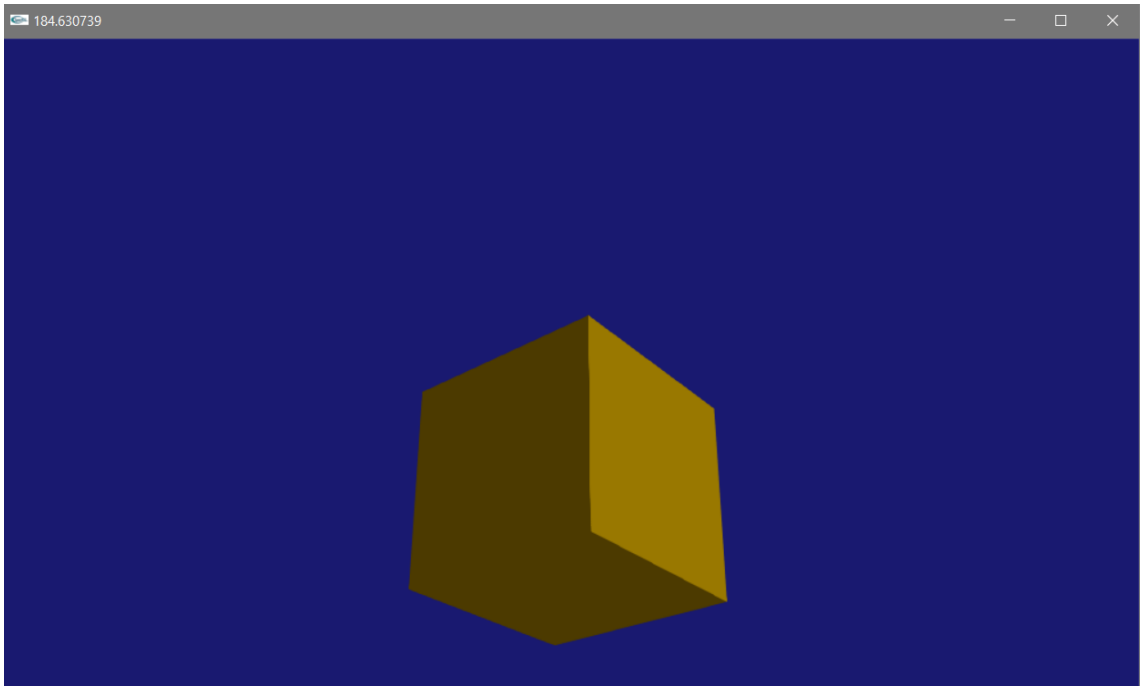

Figura 29 – Iluminação sobre um cubo – *II*

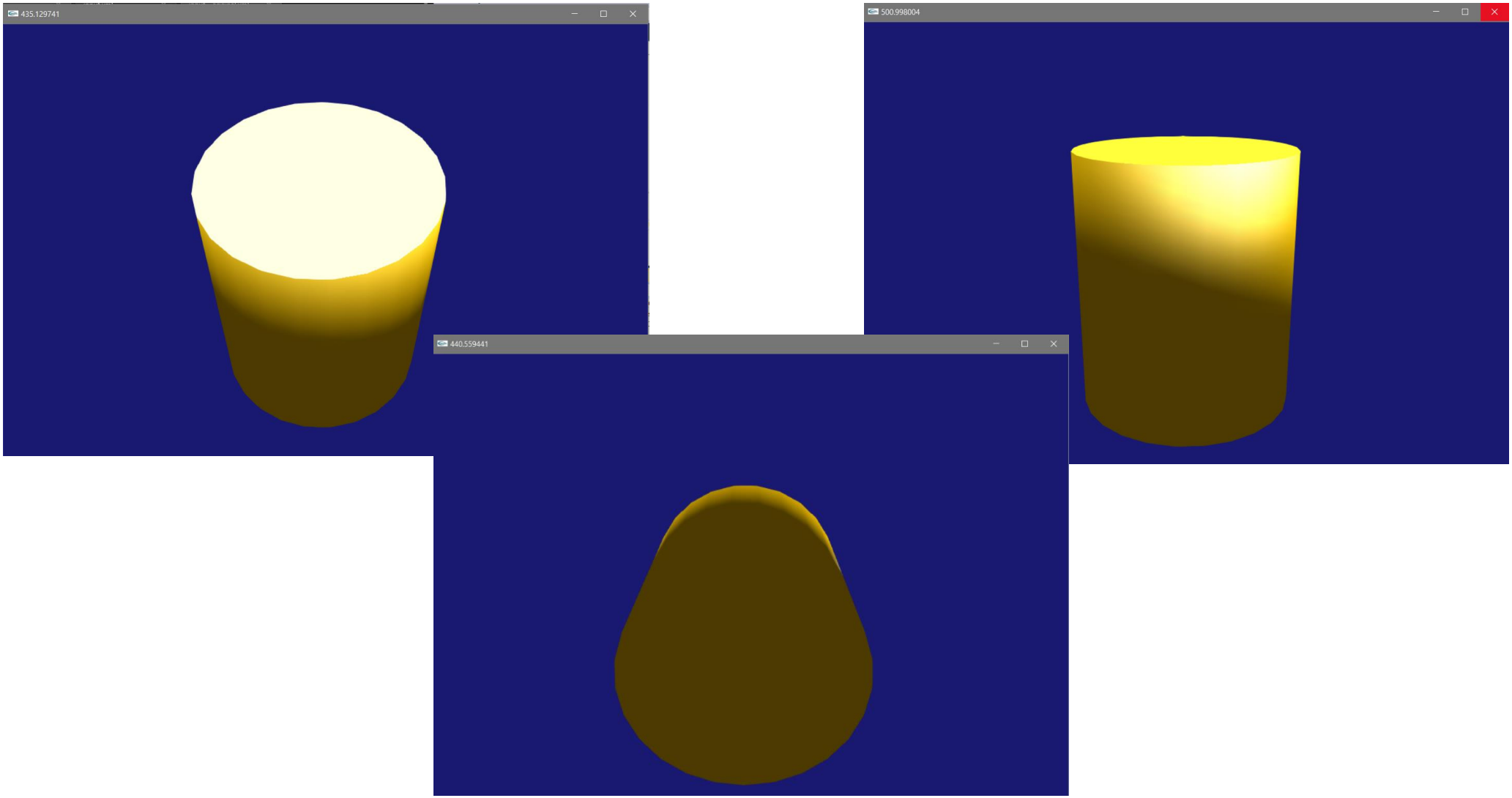

Figura 30 – Iluminação sobre um cilindro

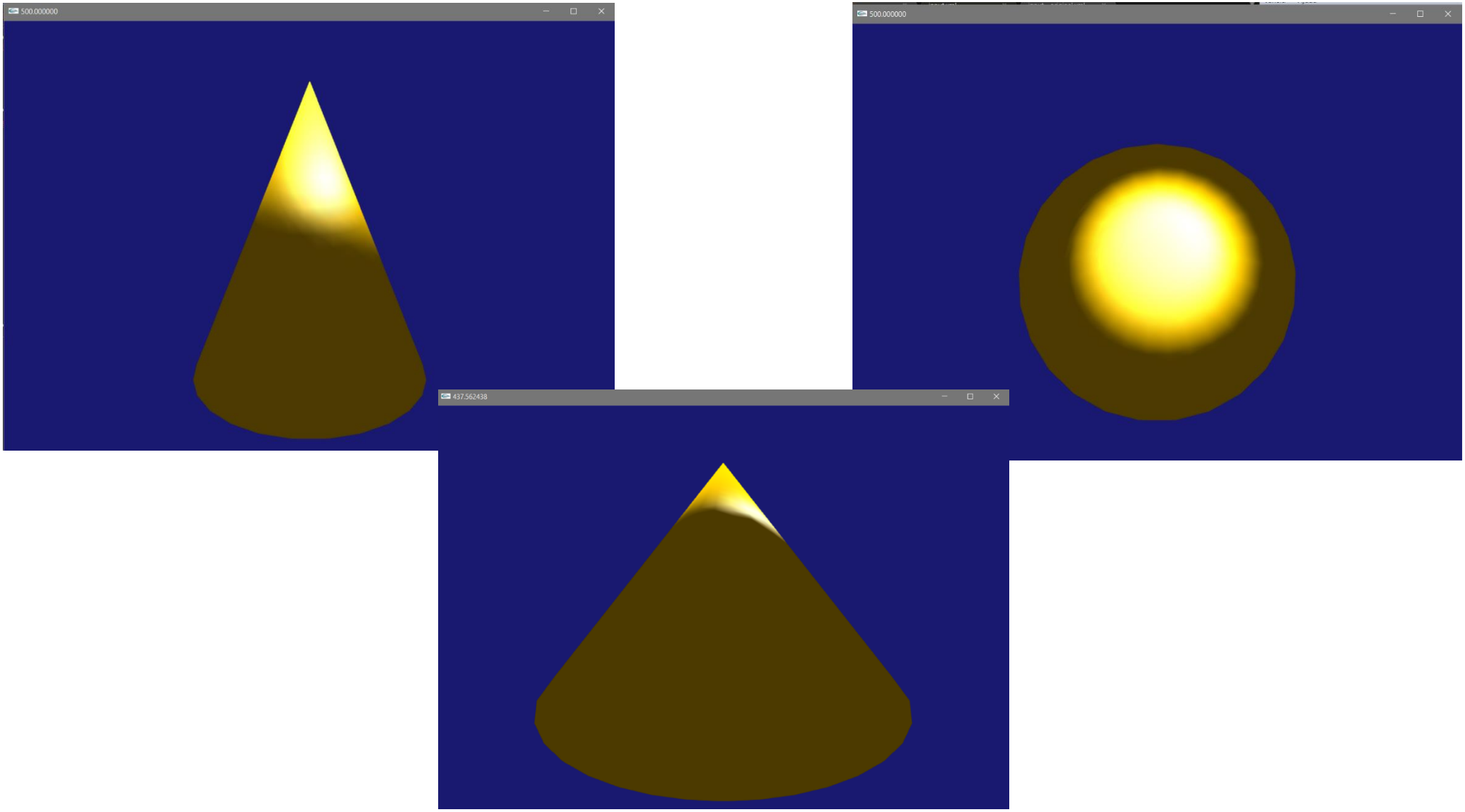

Figura 31 – Iluminação sobre um cone

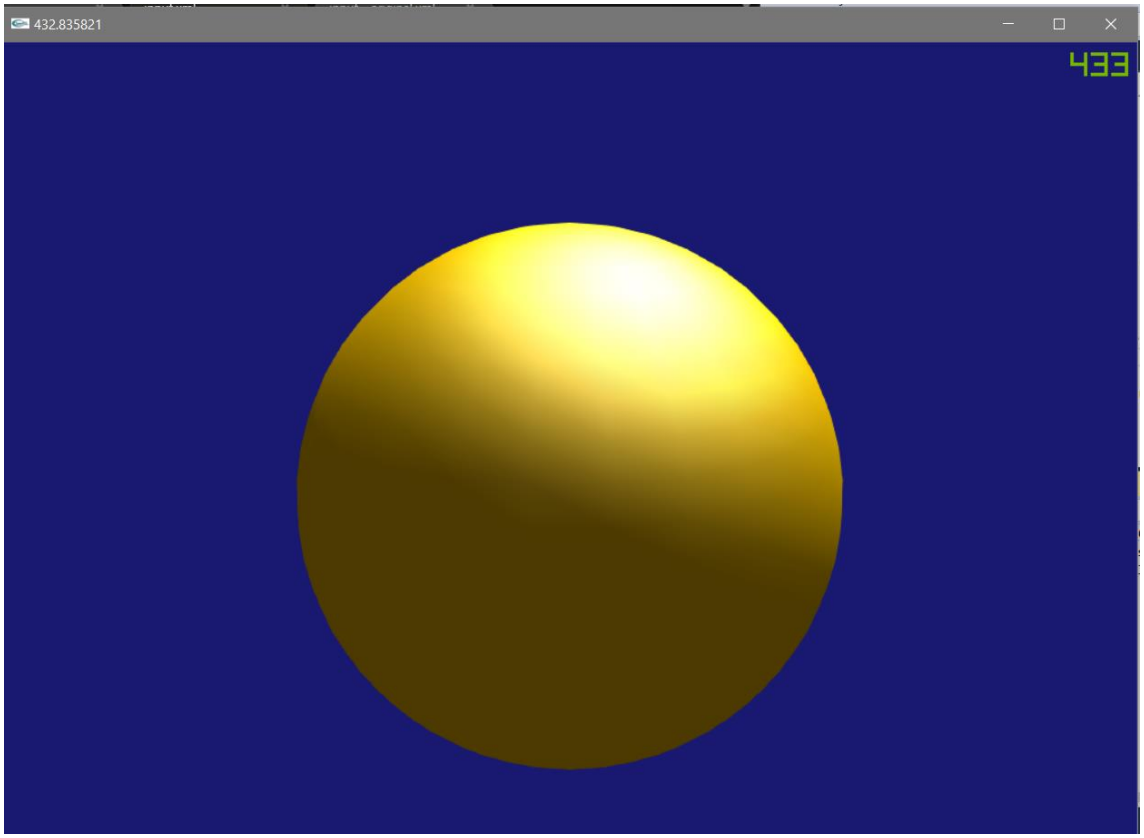

Figura 32 – Iluminação sobre uma esfera - *I*

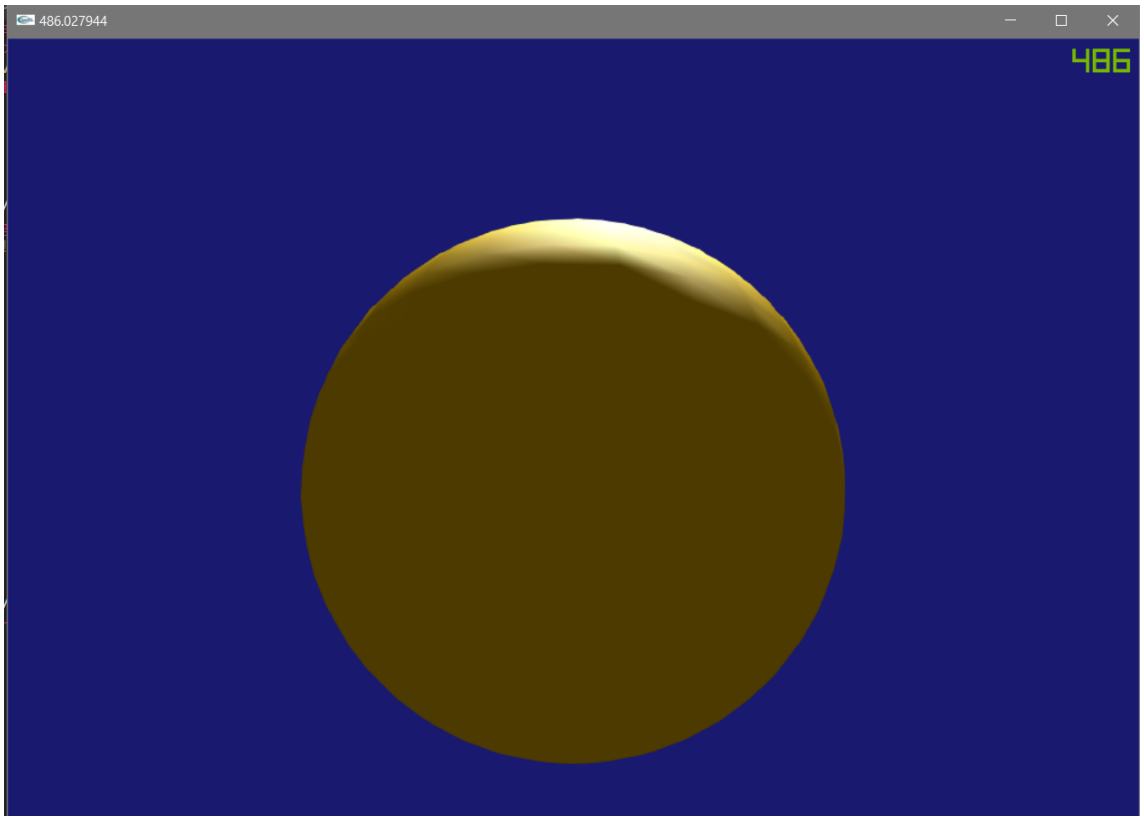

Figura 33 - Iluminação sobre uma esfera - *II*

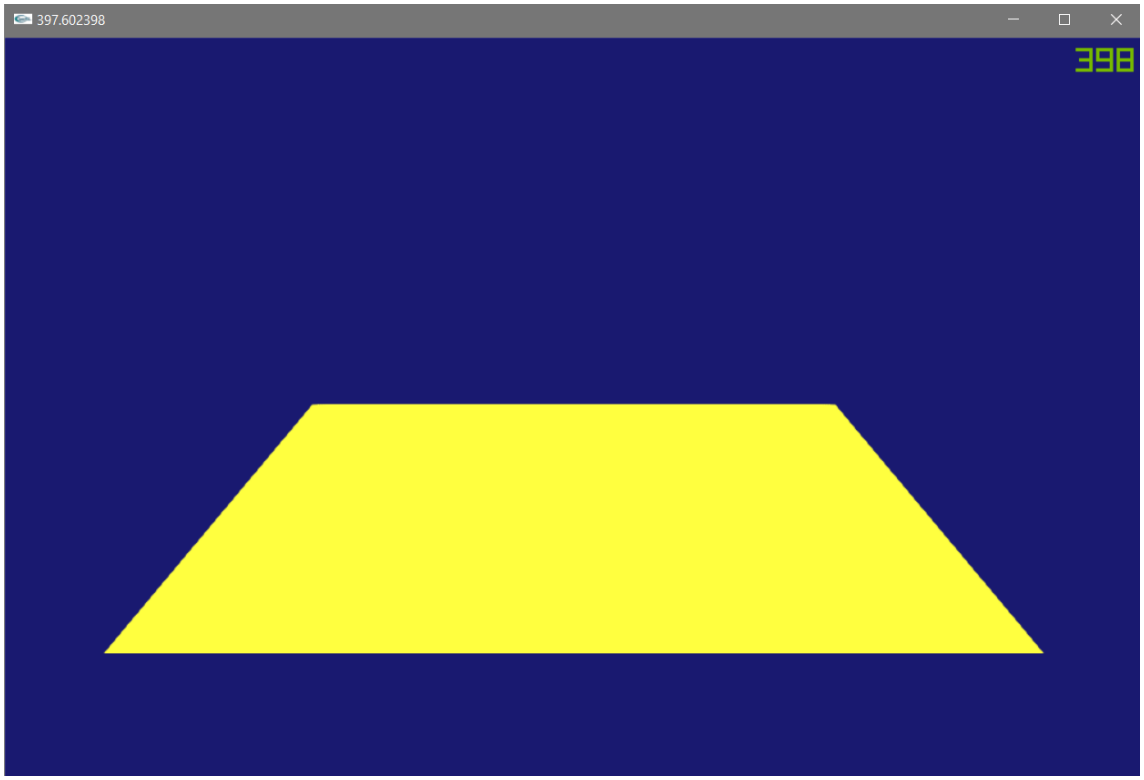

Figura 34 – Iluminação sobre um plano - *I*

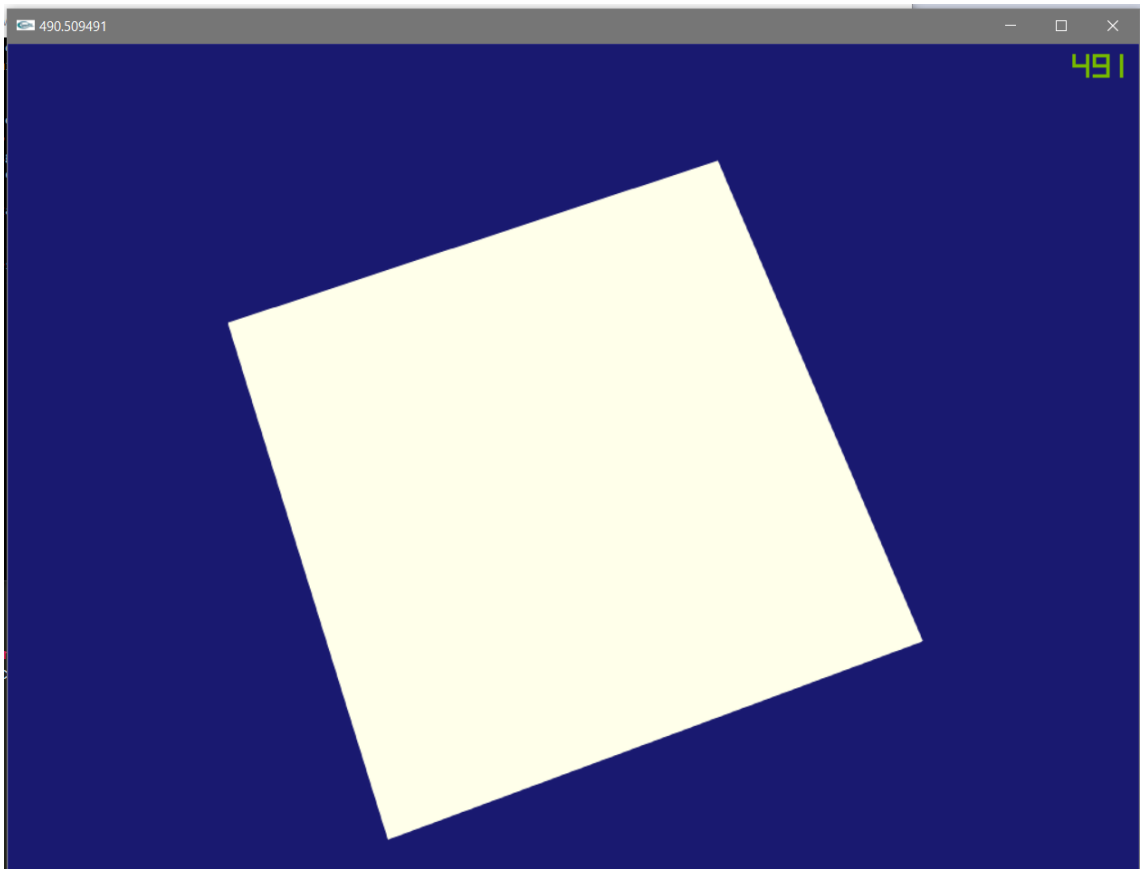

Figura 35 – Iluminação sobre um plano - *II*

6.4. Texturas em modelos gerados

6.4.1. Plano

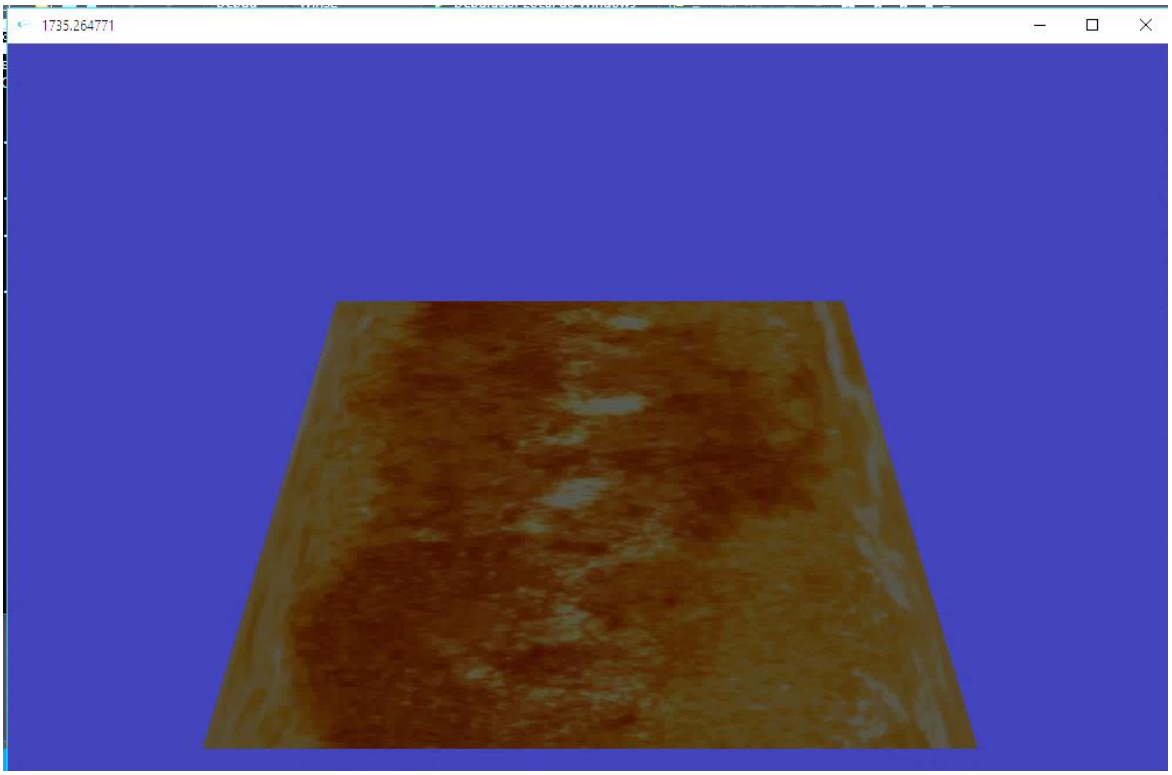

Figura 36 – Textura aplicada a um plano - I

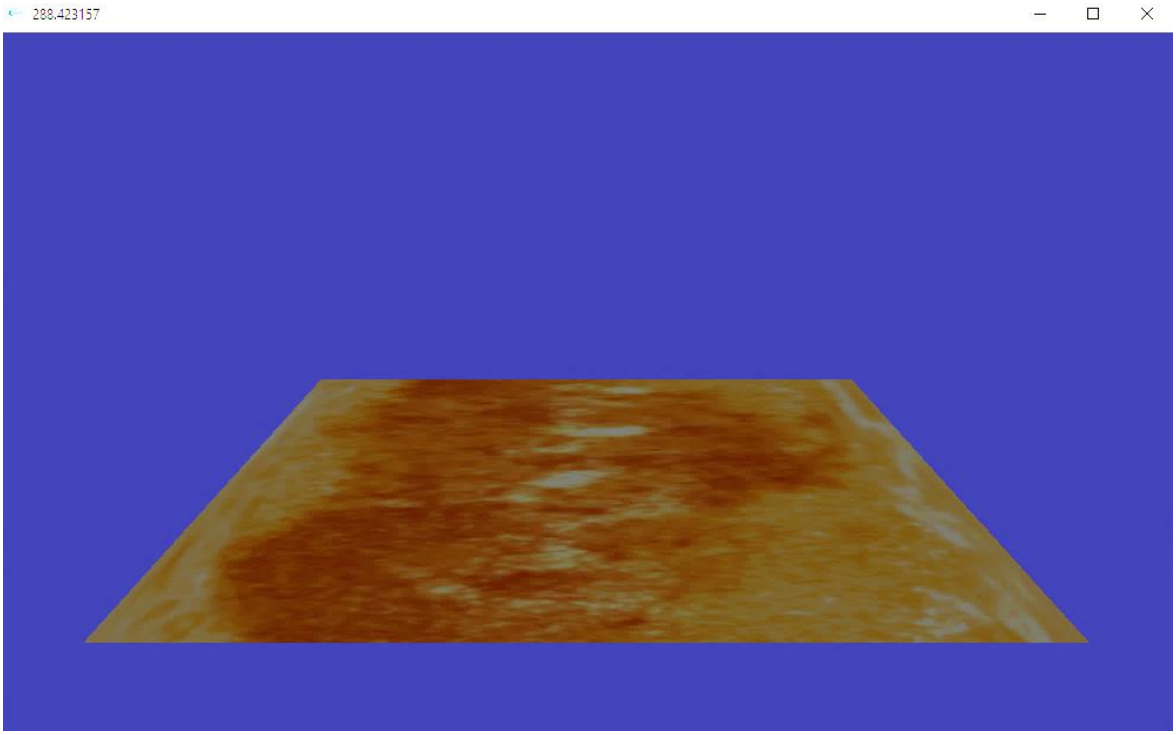

Figura 37 - Textura aplicada a um plano - II

6.4.2. Cubo

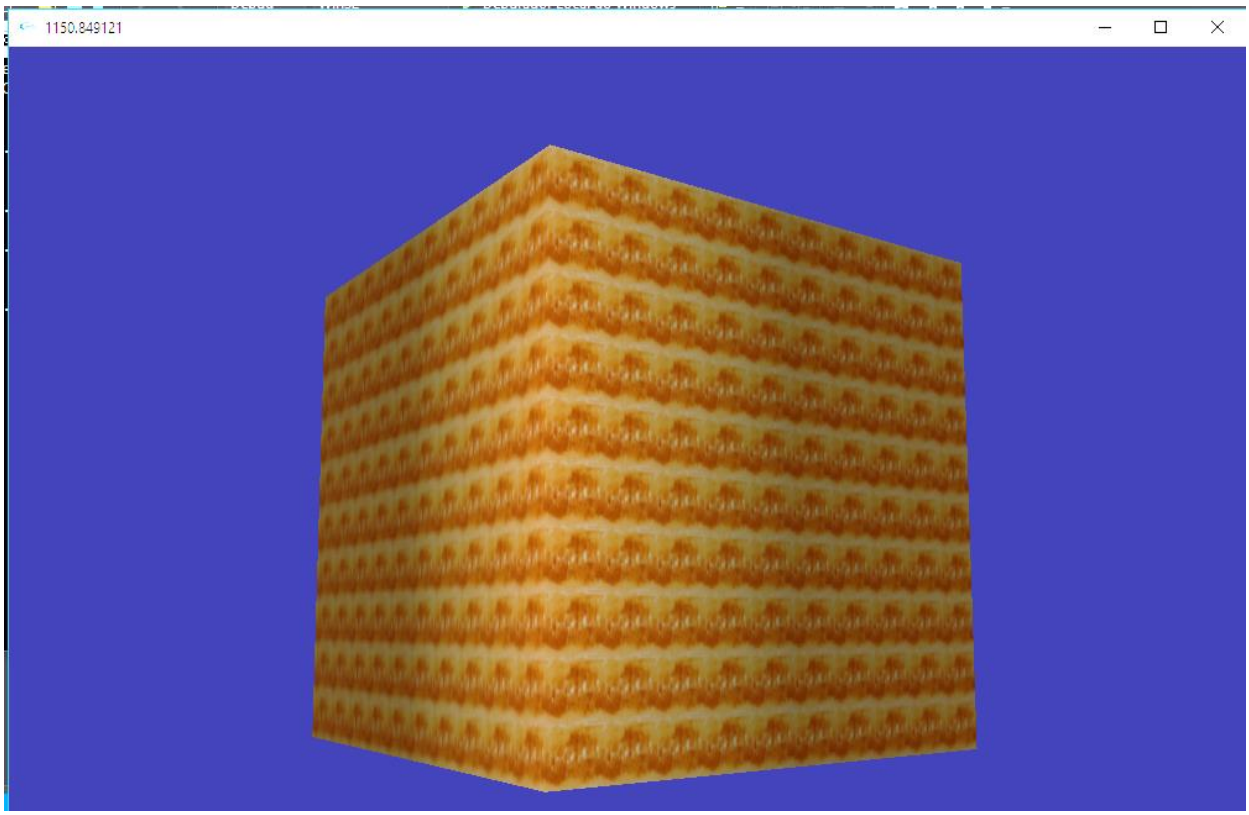

Figura 38 – Replicação de uma textura a um cubo - I

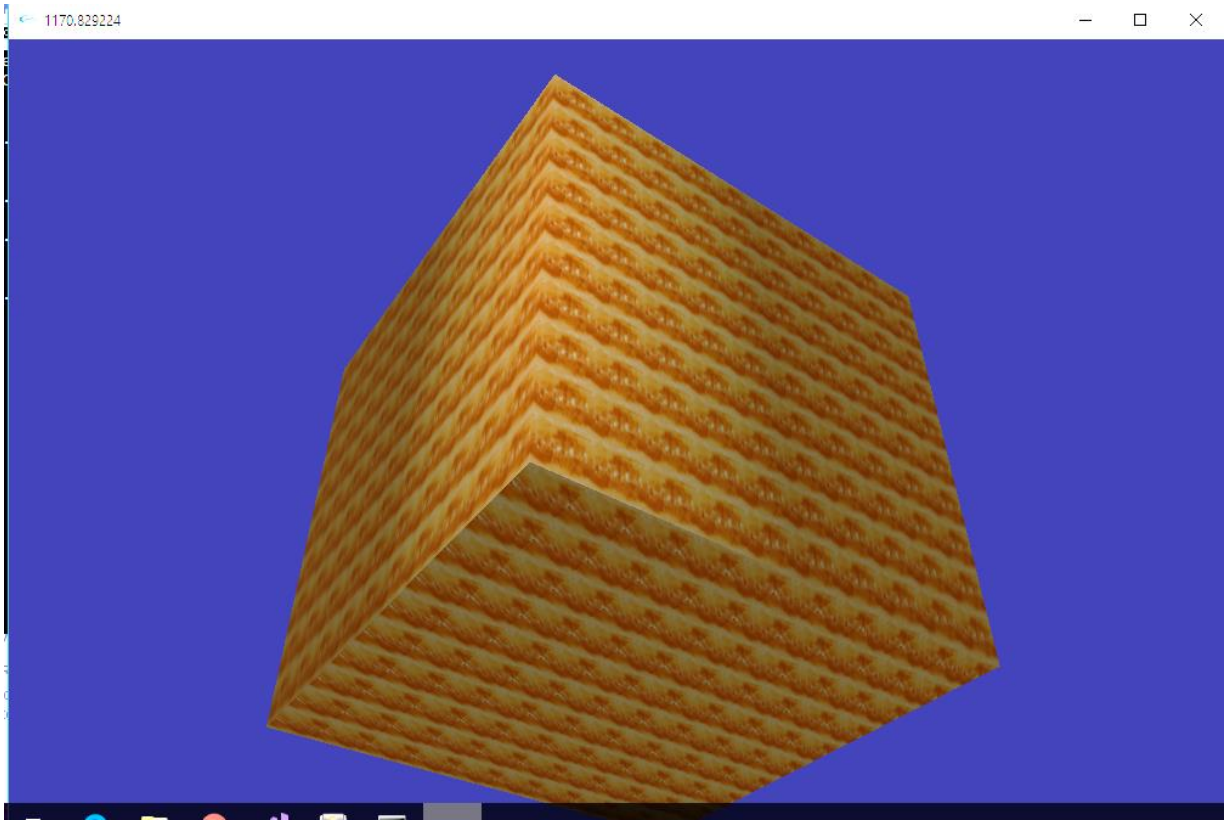

Figura 39 – Replicação de uma textura a um cubo - II

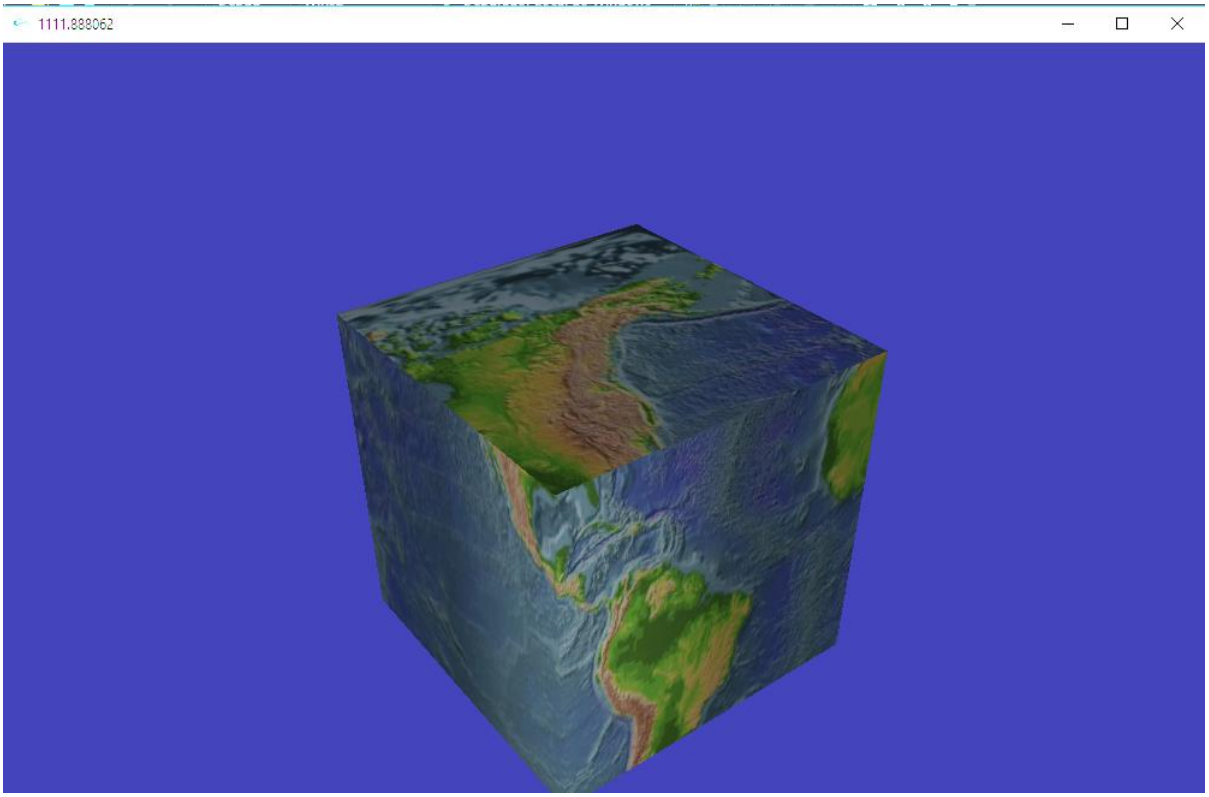

Figura 40 – Aplicação de textura a um cubo - I

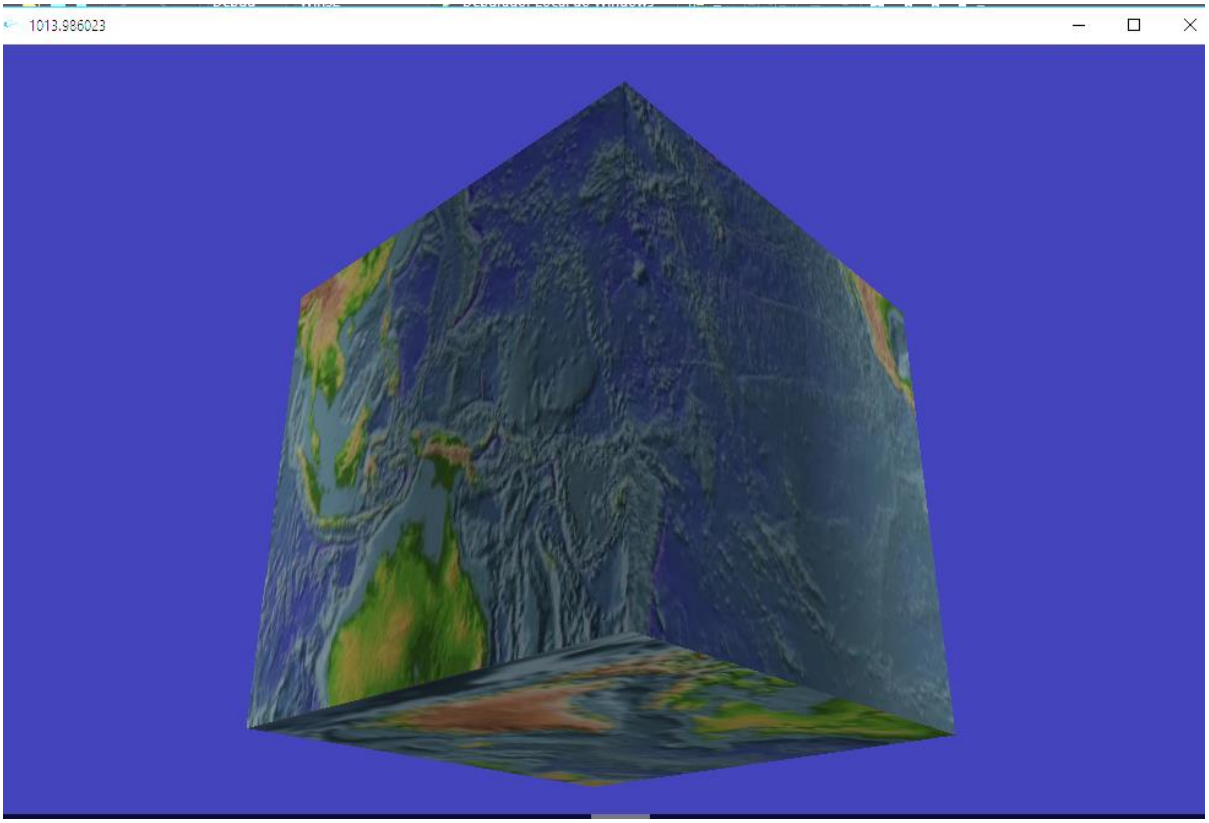

Figura 41 – Aplicação de textura a um cubo - II

6.4.3. Esfera

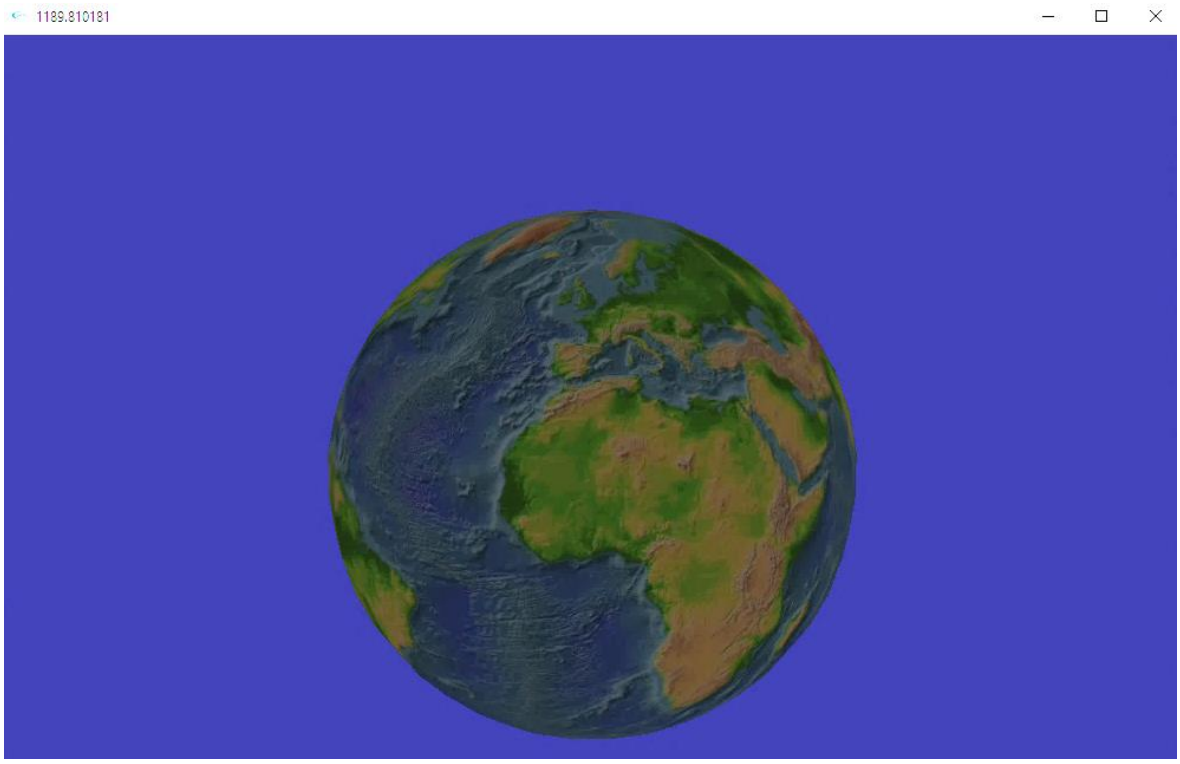

Figura 42 – Aplicação textura terra a uma esfera

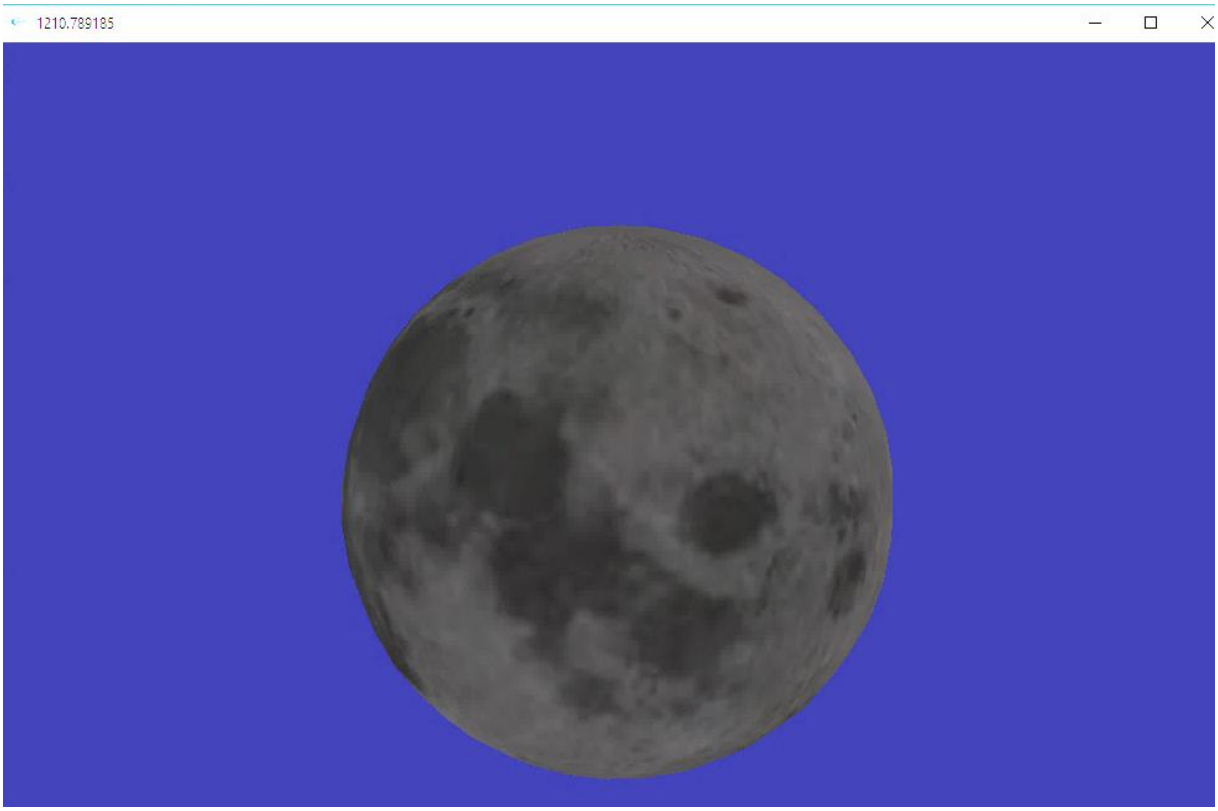

Figura 43 – Aplicação textura da lua a uma esfera - I

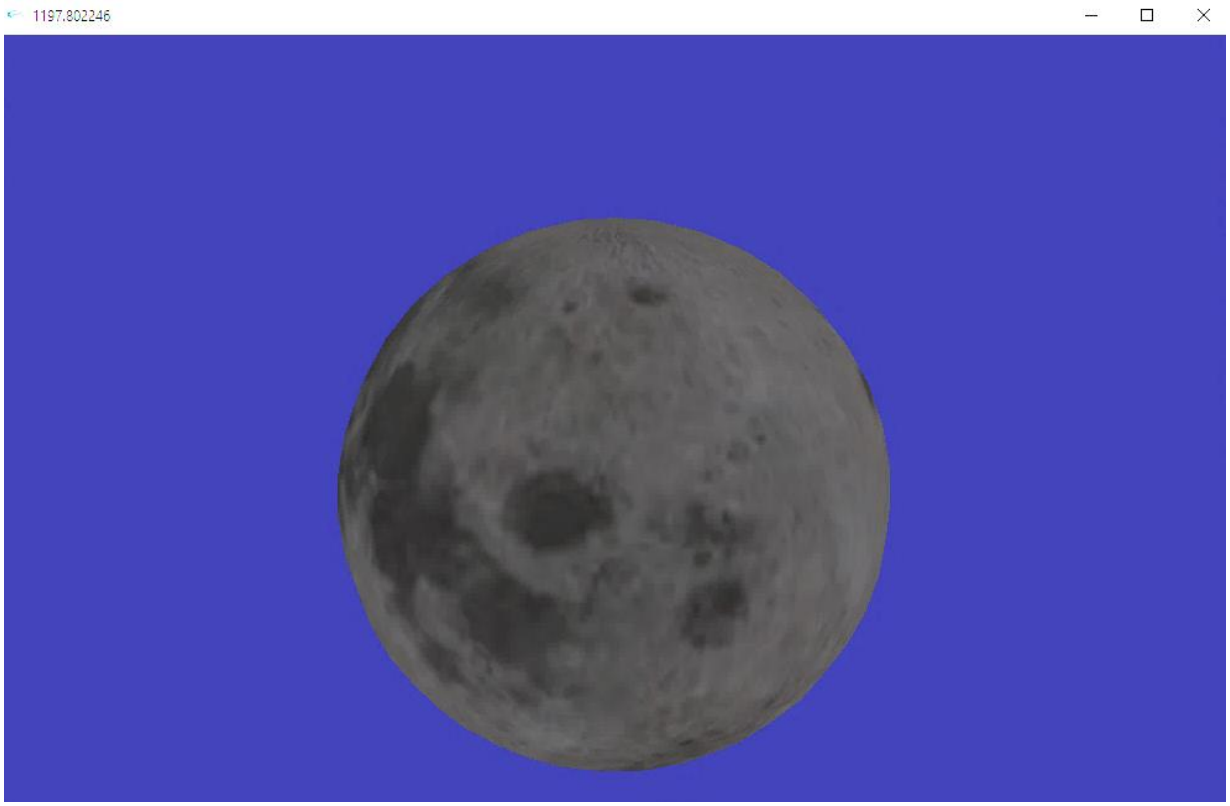

Figura 44 – Aplicação da textura da lua a uma esfera - II

6.4.4. Cilindro

Figura 45 -Textura do barril em cilindro – I

Figura 46 Textura do barril em cilindro - II

Figura 47 - Textura do barril em cilindro - III