Computação Gráfica

Phase 4 - Final

GRUPO 6
A95917 - Eduardo Diogo Costa Soares - LCC
A95609 - Duarte Alexandre Oliveira Faria - LEI
A97941 - Diogo Filipe Oliveira da Silva - LEI
A98979 - Pedro Domingues Viana - LCC

Introdução

Nesta quarta fase do projeto foram modificados:

- o gerador para que este passasse a gerar não só os vértices mas também as normais e as coordenadas de textura.
- o engine para poder aplicar textura e simular iluminação nos modelos.

Mudanças no generator

Como já foi mencionado, esta fase tem como objetivo suportar iluminação e texturas. Para isso, foi necessário que os ficheiros .3d armazenassem não só os vértices mas também as normais de cada vértice e coordenadas de textura.

Para permitir a simulação de iluminação no engine, foram adicionados cálculos de normais aos modelos gerados. Cada modelo foi ajustado para calcular a normal de cada vértice com base na sua geometria:

- No caso de superfícies analíticas como por exemplo a *sphere*, as normais foram calculadas com base nas coordenadas esféricas.
- Para modelos com faces planas como é o caso do *plane* e da *box*, as normais foram associadas de forma direta com a orientação das faces.

Adicionalmente, foram incluídas coordenadas de textura (u, v) para permitir o mapeamento de imagens sobre os modelos sendo que tal como no cálculo das normais estas coordenadas foram definidas de acordo com a topologia da figura:

- Para superfícies como é o caso da *sphere*, as coordenadas foram derivadas diretamente dos ângulos polares.

- Para por exemplo *box*, as coordenadas foram atribuídas proporcionalmente à área da face.

Depois de geradas as normais e coordenadas de textura são guardadas em sequência após os vértices no ficheiro .3d, linha por linha, mantendo a correspondência de ordem.

Mudanças no engine

Tal como aconteceu na terceira fase, o engine voltou a ser o maior alvo de mudanças, passando agora a suportar as funcionalidades de **iluminação** e **textura**.

A principal mudança introduzida foi a criação da classe Models, que veio substituir a estrutura anterior usada para representar modelos. Esta alteração tornou-se necessária devido à complexidade acrescida associada à nova fase, onde cada modelo passou a conter não só vértices, mas também normais, coordenadas de textura, textura associada e materiais (iluminação). Assim, a classe Models permite encapsular:

- os identificadores dos VBOs (vértices, normais, coordenadas de textura),
- a textura associada ao modelo (caso exista),
- e os parâmetros de material (via um objeto Color).

Esta abordagem permite que os modelos estejam separados dentro de cada grupo sendo que agora, cada grupo contém uma lista de objetos Models, o que exigiu alterações em várias partes do código, sendo as principais:

- readXMLgroups(): passou a construir instâncias de Models para cada modelo, preenchendo os respetivos campos com os dados do ficheiro .3d, da textura, e dos materiais definidos na tag <color>.
- readFile(): foi modificada para já não devolver apenas vértices, mas também normais e coordenadas de textura, preenchendo os campos apropriados no objeto Models.
- draw(): foi totalmente reestruturada para trabalhar com objetos Models.

Para a iluminação foi criada uma classe **Light** que permite representar os três tipos de luz presentes nos XML com a tag <light>:

- **Point light:** luz que é emitida em todas as direções por um ponto.
- **Directional:** luz que é imitada numa direção específica.
- **Spotlight:** luz emitida de um ponto numa direção específica, com um ângulo de abertura.

Adicionalmente, foi introduzido o conceito de materiais através de uma nova tag <color> . Para suportar esta funcionalidade foi criada uma outra classe **Color**, permitindo definir os seguintes componentes:

- **ambient:** componente ambiente da luz refletida;
- **diffuse:** componente difusa;
- **specular:** reflexos especulares;
- **emissive:** emissão própria de luz do objeto;
- **shininess:** brilho.

Estes valores são utilizados com as funções glMaterialfv() e glMaterialf() para configurar as propriedades dos modelos.

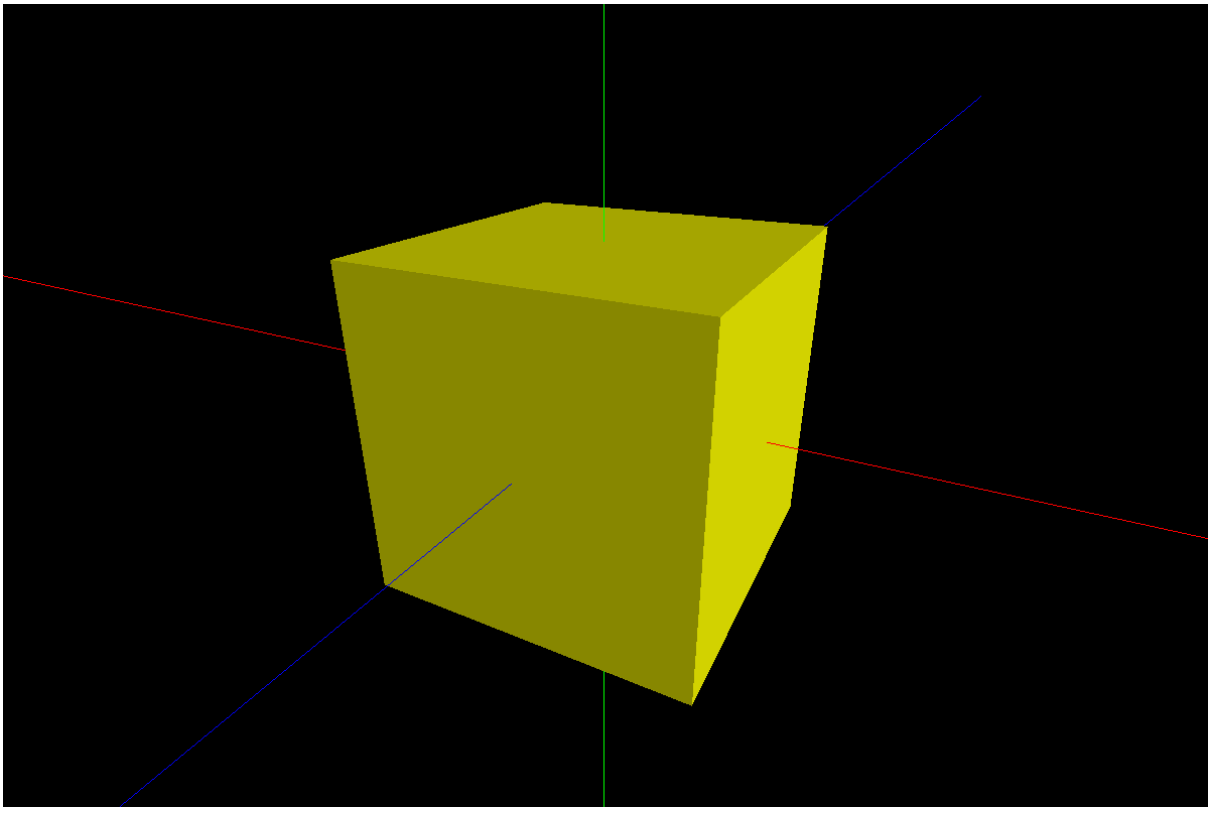
Com esta quarta fase, os ficheiros XML passaram a incluir também a nova tag <texture>, responsável pela aplicação de texturas nos modelos. As imagens especificadas são carregadas utilizando a biblioteca DevIL, sendo que foi implementado um mapa de texturas para evitar vários carregamentos desnecessários da mesma imagem.

A função responsável por carregar as imagens é a loadTexture(), esta gera o identificador da textura e define os parâmetros adequados (filtros, repetição, mipmaps, etc.). Durante a renderização, a textura é ativada com GL_TEXTURE_2D sempre que aplicável, sendo definido o modo de combinação com a iluminação através de GL_MODULATE.

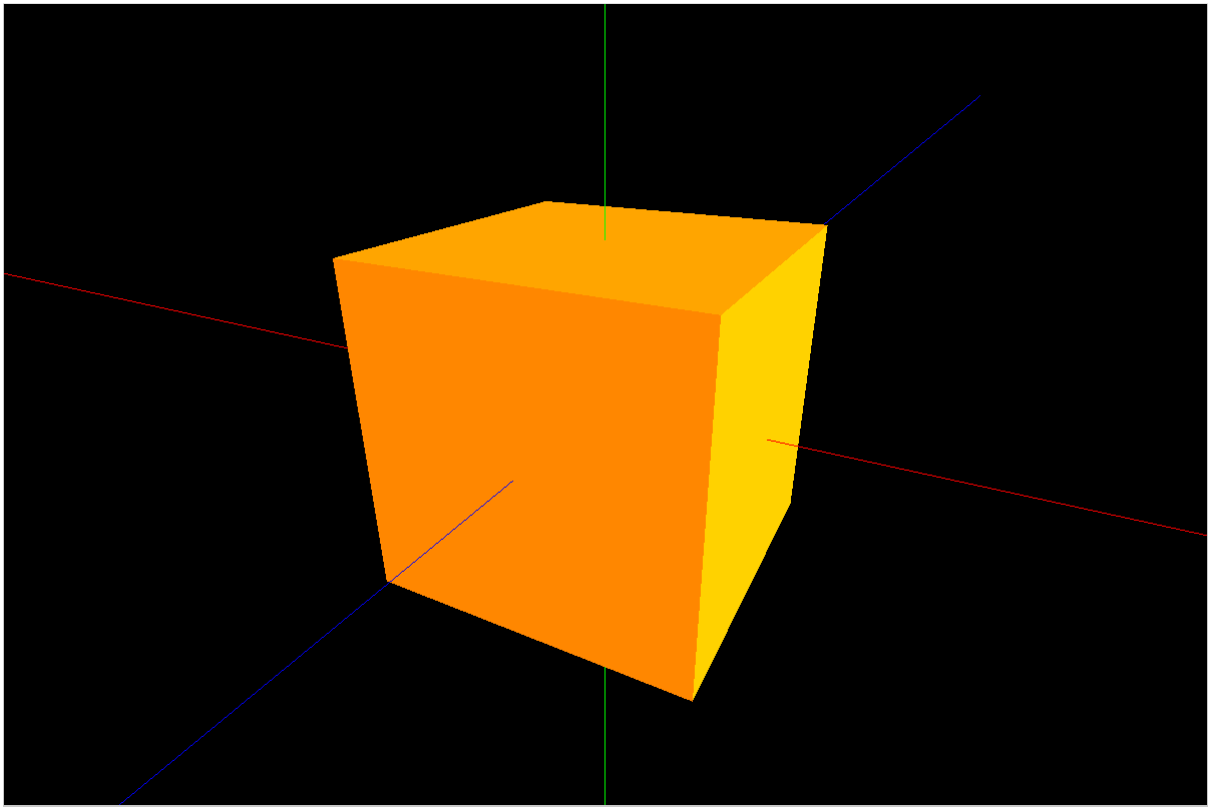
As coordenadas de textura são lidas do VBO anteriormente carregado com as coordenadas do ficheiro .3d e aplicadas com glTexCoordPointer.

Demos

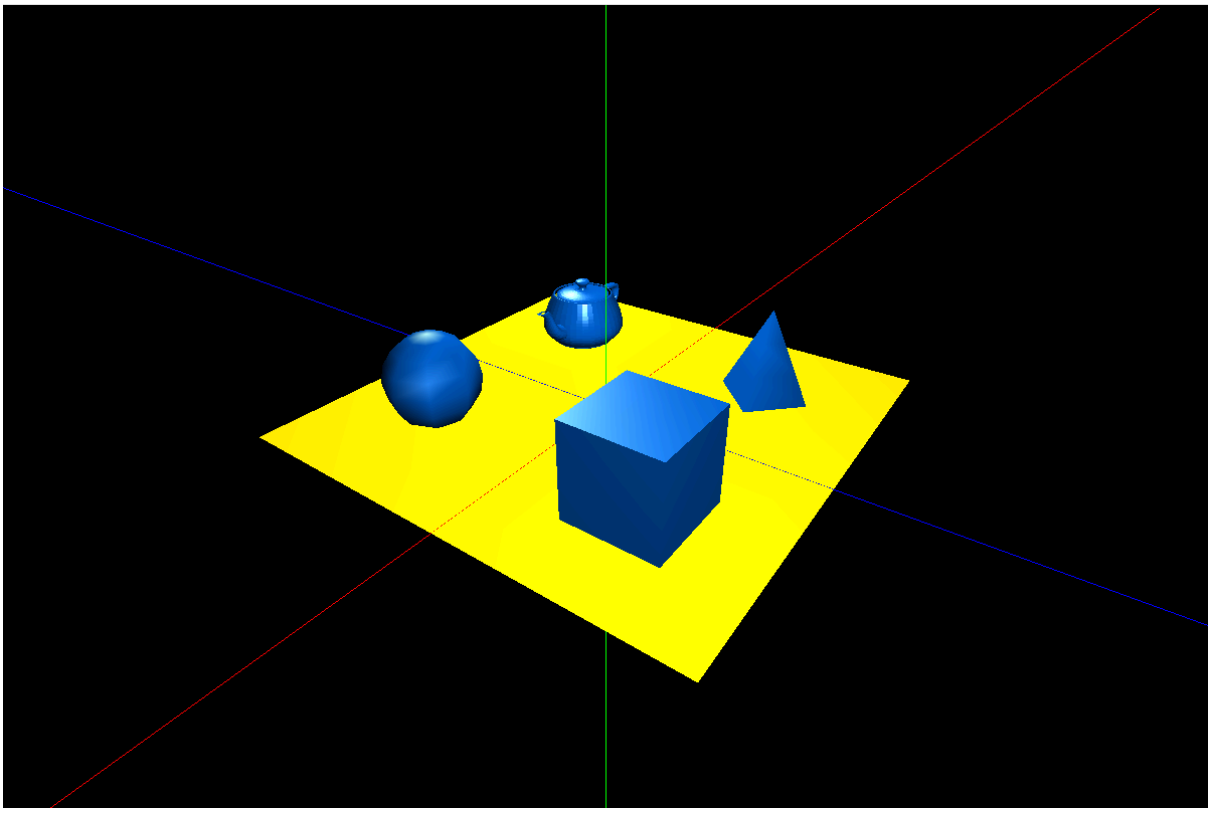
- test_4_1.xml



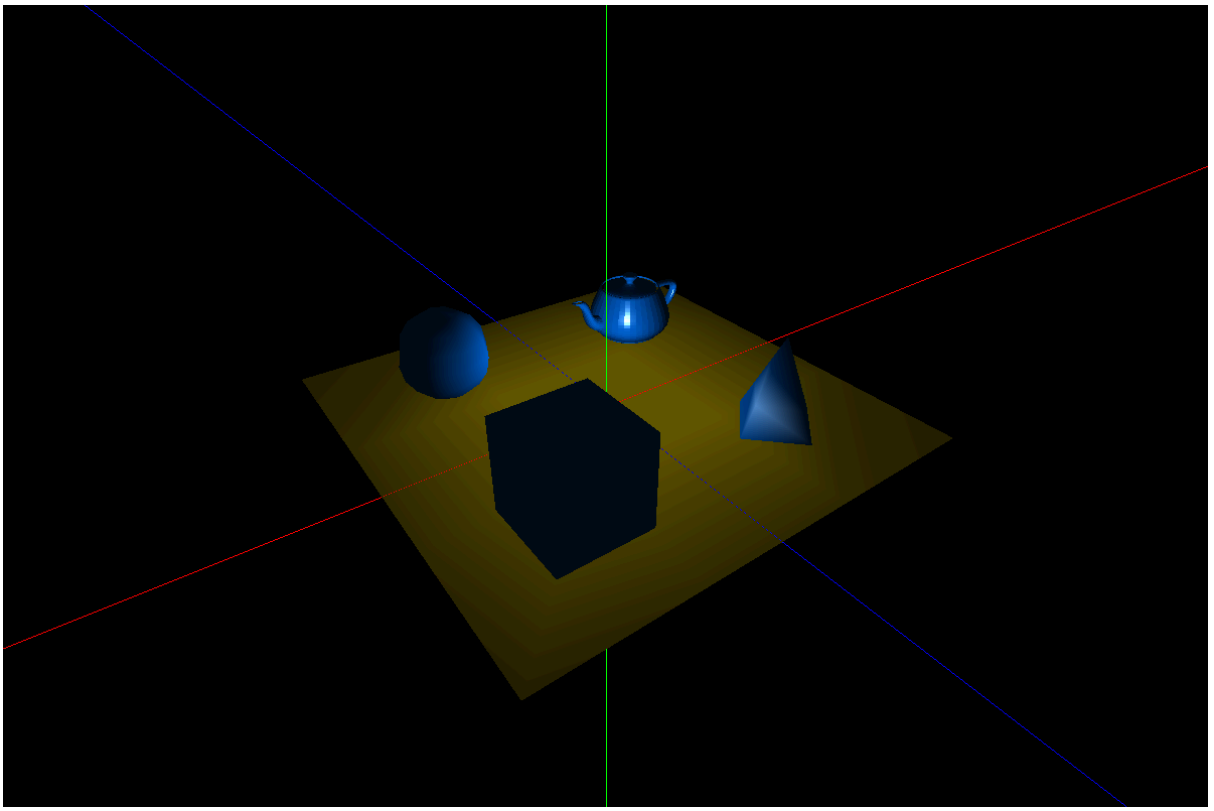
- test_4_2.xml



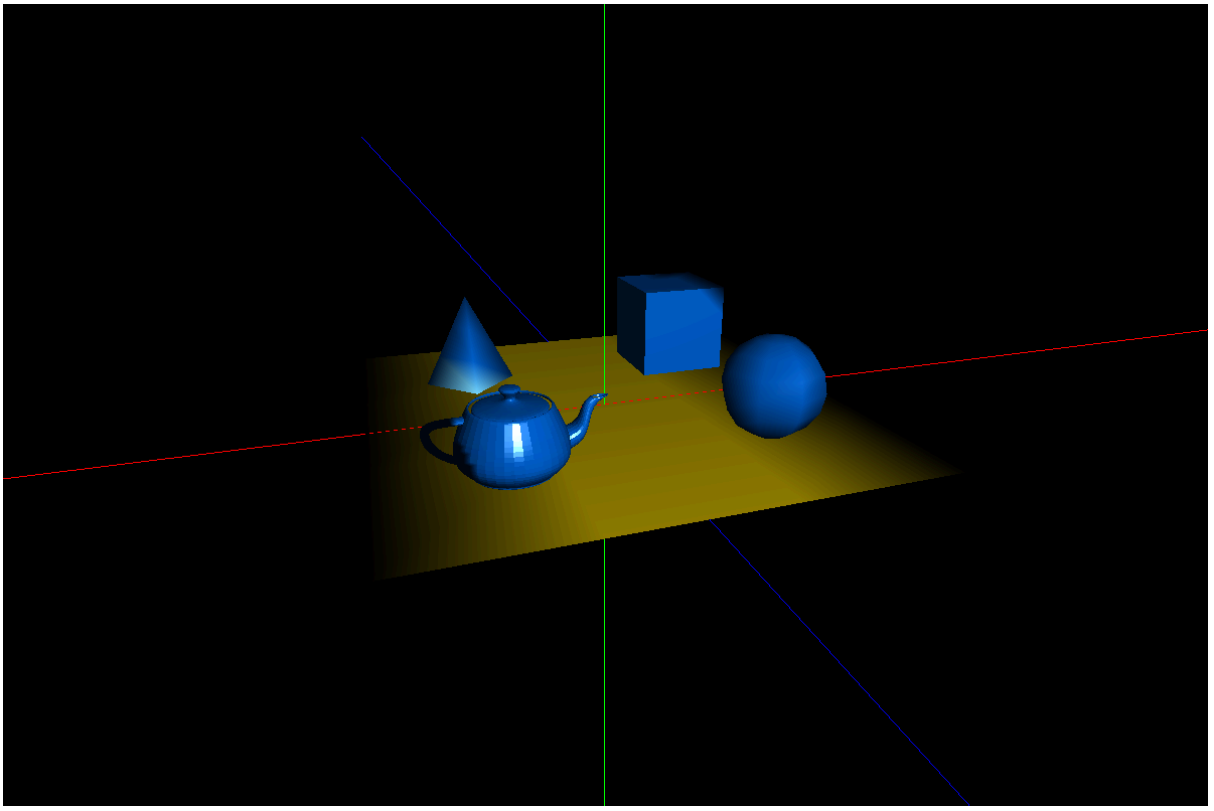
- test_4_3.xml



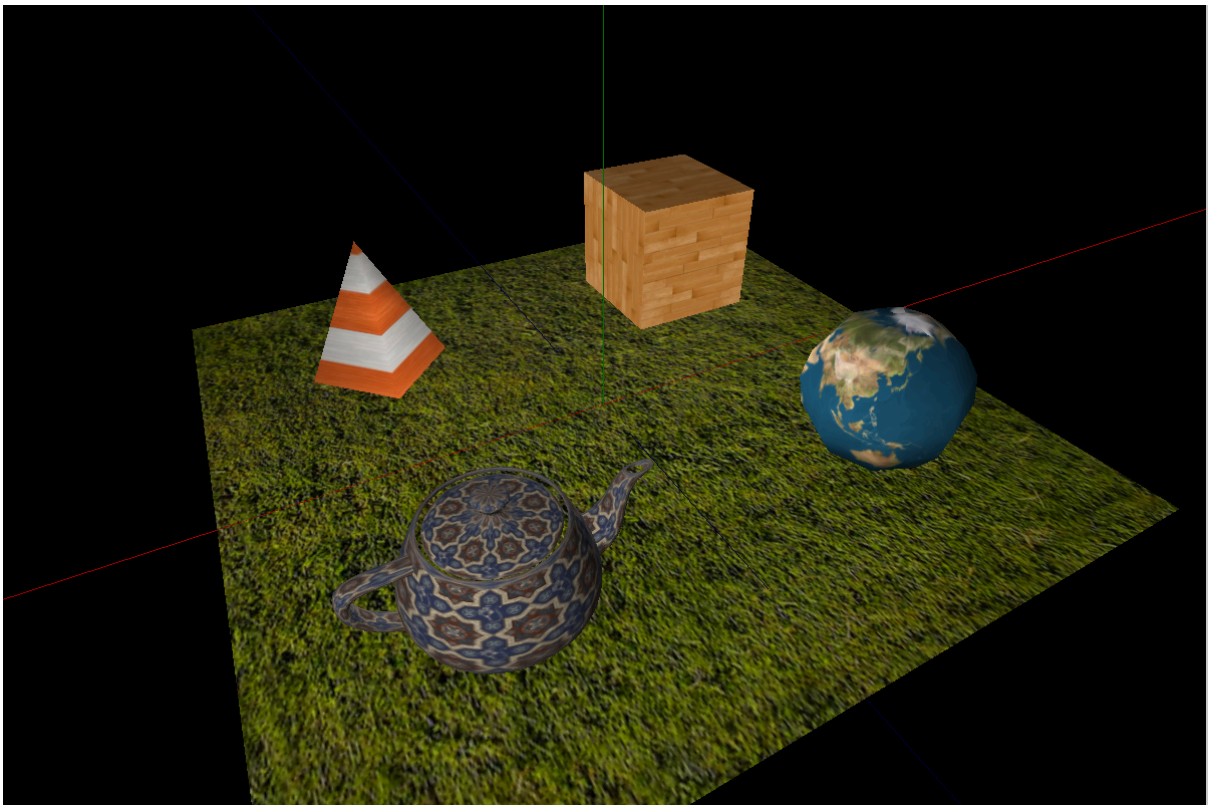
- test_4_4.xml



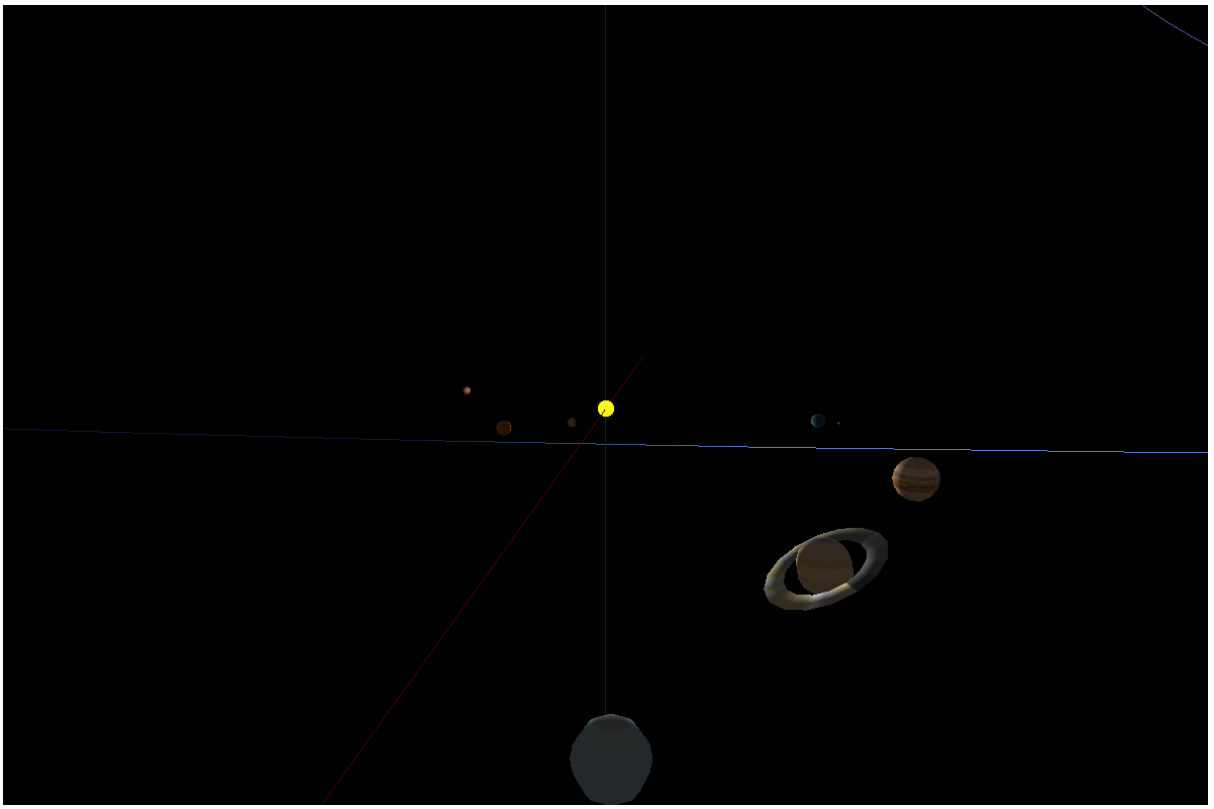
- test_4_5.xml



- test_4_6.xml



- solar_system.xml



Testar

Para correr assumindo que está no diretório do projeto(ou seja no CG-main) basta fazer:

- make
- ./Generator/generator nome [argumentos do nome] nome.3d
- ./Engine/engine teste_desejado