



Universidade do Minho
Escola de Engenharia

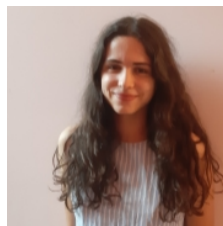
Computação Gráfica

Trabalho Prático - Fase 2

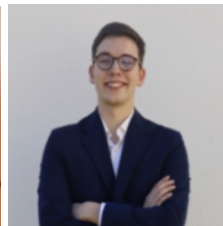
Grupo 02



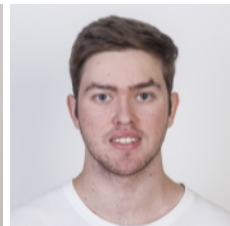
André Campos
a104618



Beatriz Peixoto
a104170



Diogo Neto
a98197



Luís Freitas
a104000

Índice

1. Introdução.....	2
2. Rose Tree.....	2
3. Parsing do XML na Rose Tree.....	2
4. Iterador.....	3
4.1. Como funciona.....	3
4.1.1 startIter.....	4
4.1.2 next.....	4
4.1.3 shouldPush.....	4
4.1.4 shouldPop.....	5
4.1.5 PopsRestantes.....	5
4.2. Como produz o cenário.....	5
5. Demos.....	13
5.1. Demos do Enunciado.....	13
5.1.1. Ficheiro test_2_1.xml.....	13
5.1.2. Ficheiro test_2_2.xml.....	14
5.1.3. Ficheiro test_2_3.xml.....	15
5.1.4. Ficheiro test_2_4.xml.....	16
5.2. As nossas Demos.....	17
5.2.1. Ficheiro demo.xml.....	17
5.2.2. Ficheiro sistema_solar.xml.....	18
6. Conclusão.....	19

1. Introdução

Esta fase consistiu em criar cenas hierárquicas utilizando transformações geométricas. Cada cenário é representado através de uma árvore hierárquica onde cada nodo pode conter tanto transformações geométricas como a translação, a rotação ou a escala, bem como modelos para desenhar. Além disso, cada nodo pode conter subgrupos, em que os seus filhos herdam as suas transformações. Cada cenário referido é descrito num ficheiro XML onde se encontra a configuração de cada grupo, as suas transformações e modelos.

Neste sentido, nos tópicos abaixo explicamos o nosso raciocínio para a implementação desta fase.

2. Rose Tree

Para representar o conteúdo do **group** no XML, decidimos usar uma RoseTree. Cada nodo contém o ID do nodo, a sua última transformação, o conjunto de modelos, a sequência de transformações e uma lista com os seus nodos filhos.

- **ID**: é único e corresponde ao identificador do nodo.
- **Última transformação**: corresponde ao ID do nodo que contém a última transformação que será aplicada no nodo atual, isto é, corresponde às transformações herdadas pelo nodo atual e que lhe serão aplicadas.
- **Conjunto de transformações**: corresponde à sequência de transformações aplicadas a esse nodo.
- **Conjunto de modelos**: corresponde ao conjunto de modelos nesse nodo a desenhar.
- **Lista com os nós filhos**: corresponde aos filhos do nodo em questão.

3. Parsing do XML na Rose Tree

Para implementar esta fase, fizemos algumas alterações no parser dos ficheiros XML, mais precisamente na tag **group**, e na engine. Na primeira fase, obtivemos uma lista de modelos a partir da tag **group**. Nesta fase os **groups** são mais complexos, isto é, existem transformações e **groups** dentro de **groups**.

Deste modo, através da análise dos ficheiros XML representamos a sua estrutura (tag **group**) numa Rose Tree, tal como mencionado acima. Além disso, criamos um objeto chamado **Groups** que contém esta árvore e um iterador, que itera sobre a mesma. O iterador é explicado no ponto 4.

Neste sentido, o parser é essencialmente recursivo, na medida em que é aplicado ao nodo e aos seus filhos, isto é, **groups** dentro do **group** em consideração. Este processo é feito pela função **processaXML_grupo_rec**.

Dentro desta função atribuímos um ID único ao nodo, isto é, ao **group** atual. Verificamos quais as suas transformações, representadas pela tag **transform** nesse **group**, e os modelos a desenhar simbolizados pela tag **models** nesse **group**. Esta função é chamada recursivamente para ser aplicada aos filhos do nodo, isto é, **groups** dentro desse **group**, e retorna esse nodo com todos os seus filhos.

Para verificar quais as transformações do nodo foi feita a função **processaXML_grupo_getTransformacao** que itera sobre todas as transformações (rotação, translação e escala), adiciona-as numa lista de transformações e retorna essa lista. Caso o nodo não tenha transformações, essa lista será nula.

A última transformação que foi atribuída a esse nodo é o ID do último nodo que contém as transformações herdadas. Se o nodo atual tiver transformações, a última transformação dos seus filhos é o nodo atual. Caso contrário, a última transformação dos seus filhos será igual à sua última transformação.

Para verificar quais os modelos do nodo foi feita a função **processaXML_grupo_getModels** que itera sobre todos os modelos (esfera, cone, cubo e plano), carrega-os a partir de um ficheiro .3d indicado pelo **model**, adiciona esse modelo a uma lista de modelos e retorna-a. Caso o nodo não tenha modelos, essa lista será nula.

Assim, em cada nodo temos dois números que indicam o ID e a última transformação do nodo, temos também uma lista de transformações (lista nula caso não haja transformações), uma lista de modelos (lista nula caso não haja modelos) e uma lista com todos os seus filhos. Verificamos também que este modelo é compatível com os ficheiros XML apresentados na primeira fase, algo que era desejado.

4. Iterador

Após o XML ser representado numa Rose Tree, o objeto **Groups** também tem uma secção para o Iterador. Este iterador será ideal para percorrer a Rose Tree de forma eficiente e indicar à **engine** o que deve fazer conforme o nodo que recebe.

Este iterador tem:

- **Próximo nodo do Iterador** (Indica qual o próximo nodo que a engine deve considerar)
- **Stack para o Iterador** (Indica quais os nodos serão os próximos escolhidos para o iterador)
- **Stack de Transformações para o Iterador** (Indica quais os nodos estão a aplicar transformações no cenário)
- **Número de Pops** (Indica quantas vezes a engine deve fazer Pop da matriz de OpenGL)
- **Número de Pushs** (Indica se a engine deve fazer Push da matriz de OpenGL)

4.1. Como funciona

O Iterador tem um conjunto de métodos com as seguintes funcionalidades:

- **startIter** : método que inicia o iterador
- **next** : Obtém o próximo elemento do iterador (se a Rose Tree acabar, obtemos um nullptr)
- **shouldPop** : Indica quantas vezes devemos dar pop da Matriz de OpenGL
- **shouldPush** : Indica se devemos dar push da Matriz de OpenGL
- **PopsRestantes** : No fim do iterador, indica quantos pop da Matriz de OpenGL devemos dar para que a Matriz volte ao seu estado inicial

No caso das transformações, decidimos usar os métodos de pop ou push da matriz de openGL para reverter transformações aplicadas à matriz de openGL. O processo de pop e push explica-se da seguinte forma:

- **PUSH** : Guardar a matriz de openGL atual no topo da stack de openGL
- **POP** : Substituir a matriz atual de openGL pelo topo da stack de openGL, removendo-o da stack

*Exemplo: Caso eu faça **push** da matriz de openGL e aplique uma transformação X, ao dar **pop** da matriz, a matriz atual de openGL será substituída pela matriz que existia antes dessa transformação X (a matriz que foi guardada na stack de openGL graças ao **push**). Isto é bom pois é uma ótima maneira de reverter transformações.*

4.1.1 startIter

Este método é responsável por preparar o iterador para que ele seja utilizado. O que ele faz é:

- Limpa a stack de transformações do iterador
- Limpa a stack do iterador
- Coloca o número de pops a zero
- Coloca o número de pushes a zero
- Próximo nodo será nulo
- É adicionado a raiz da RoseTree à stack do iterador

4.1.2 next

Este método é responsável por indicar à engine qual o nodo a considerar e obter o próximo elemento que o iterador fornecerá à engine. Os seus procedimentos são:

- Se a stack do iterador está **vazia**, o nodo a considerar é **nulo**
- Se a stack do iterador não está **vazia**, então:
 - 1) Obtém o elemento que está no topo da stack do iterador, removendo-o
 - 2) Adiciona todos os filhos desse elemento na stack do iterador
 - 3) Dá pops na stack de transformações do iterador até que:
 - Stack esteja vazia
 - Elemento do topo da stack seja a **última transformação** do elemento

Nota: a quantidade de pops é contabilizada

 - 4) Se o elemento tiver uma transformação, adiciona-o à stack de transformações do iterador
 - 5) Retornar esse elemento

4.1.3 shouldPush

Este método é responsável por indicar à engine se ela precisa de dar push da matriz de openGL, considerando se o nodo retornado pelo método **next** contém transformações ou não.

4.1.4 shouldPop

Este método é responsável por indicar à engine quantas vezes ela precisa de dar pop da matriz de openGL, considerando quantos pops da stack de transformações do iterador foram executados no elemento retornado pelo método **next**.

4.1.5 PopsRestantes

Este método é responsável por indicar à engine quantas vezes ela precisa de dar pop da matriz de openGL para que ela volte ao estado inicial, considerando qual o tamanho da stack de transformações do iterador.

4.2. Como produz o cenário

Para produzirmos o cenário na **engine**, devemos seguir os seguintes passos:

1. Iniciar o Iterador (**startIter**)
2. Obter o próximo nodo da RoseTree (**next**)
3. Em cada nodo, devemos:
 - 1) Dar Pops da Matriz de openGL (indicado pelo número de **shouldPop**)
 - 2) Dar Push da Matriz de openGL (se **shouldPush** o indicar)
 - 3) Se houver transformações, iterar sobre elas e aplicar cada uma em sequência
 - 4) Se houver modelos, iterar sobre eles e desenhá-los
4. Após o Iterador terminar, devemos efetuar Pops da matriz de openGL (indicado pelo **PopsRestantes**)

Para fundamentar o processo descrito apresentamos um conjunto de imagens (*Figuras 1 a 10*) e, para cada imagem, segue-se a seguinte descrição:

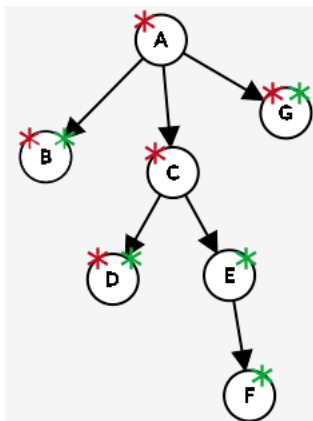
Na **RoseTree**:

- Estão representados os nodos do ficheiro XML
- Se um nodo contém uma estrela (*) vermelha significa que existe pelo menos uma transformação
- Se um nodo contém uma estrela (*) verde significa que existe pelo menos um modelo

Stack do Iterador: Indica quais os nodos que estão na stack do Iterador, stack que ditará quais os próximos nodos a serem escolhidos

Stack das Transformações do Iterador: Indica quais os nodos que contêm transformações (vai ser útil para determinar quantos pops devemos dar na matriz de openGL)

Stack das Transformações em openGL: Indica quais as transformações estão guardadas, sendo que a transformação do topo é a transformação atual

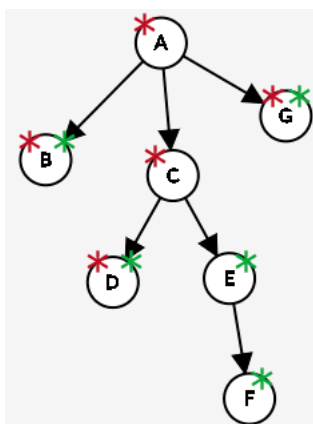


Stack do Iterador

Stack das Transformações do Iterador

Stack das Transformações em OpenGL

Figura 1. Antes de iniciar o iterador



Stack do Iterador

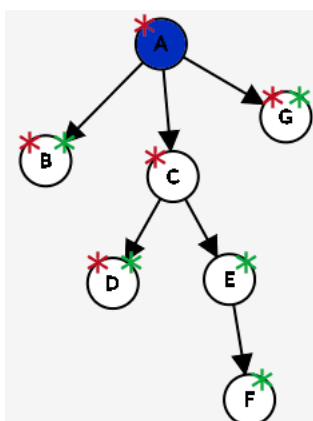
A

Stack das Transformações do Iterador

Stack das Transformações em OpenGL

Figura 2. Iniciar o Iterador

- As stacks do Iterador são limpas
- É adicionado a raiz da árvore na Stack



Stack do Iterador

G C B

Stack das Transformações do Iterador

A

Stack das Transformações em OpenGL

A

Figura 3. Obter o nodo A

- É tirado o nodo A da stack
- **Quais os filhos adicionados à stack? G, C e B**

- Qual a última transformação de A? Nenhuma
- Quantos pops da stack das transformações do iterador temos que dar até que o topo da stack seja igual à última transformação de A? 0 porque a stack estava vazia
- Nodo A contém transformação? Sim, então insere-o na stack das transformações do iterador
- Quantos pops da matriz em OpenGL devemos dar? 0
- Devemos dar push da matriz em OpenGL? Sim
- Qual a transformação atual em OpenGL? A
- Modelos desenhados: nenhum

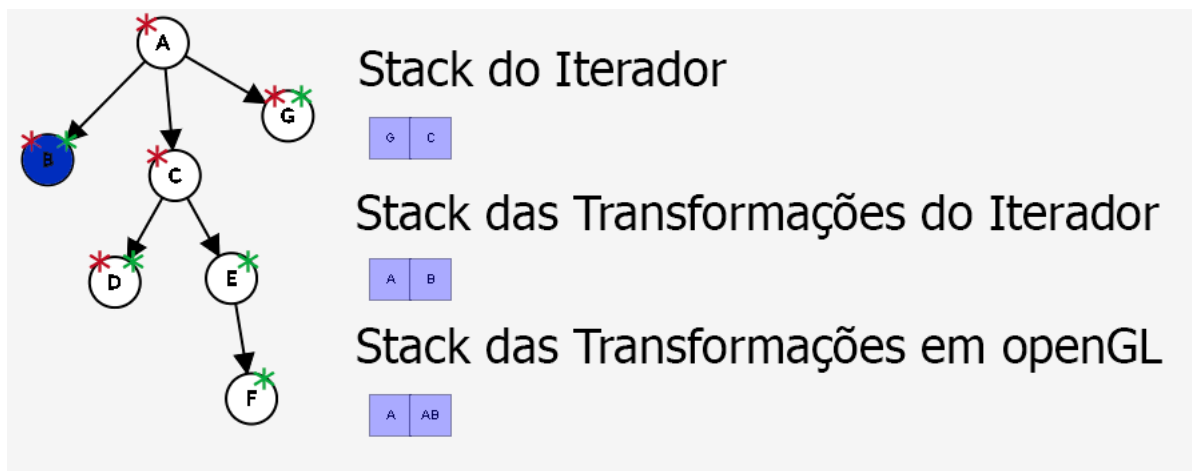


Figura 4. Obter o nodo B

- É tirado o nodo B da stack
- Quais os filhos adicionados à stack? Nenhum
- Qual a última transformação de B? A
- Quantos pops da stack das transformações do iterador temos que dar até que o topo da stack seja igual à última transformação de B? 0
- Nodo B contém transformação? Sim, então insere-o na stack das transformações do iterador
- Quantos pops da matriz em OpenGL devemos dar? 0
- Devemos dar push da matriz em OpenGL? Sim
- Qual a transformação atual em OpenGL? A+B
- Modelos desenhados:
 1. Modelo B (Transformações A e B aplicadas)

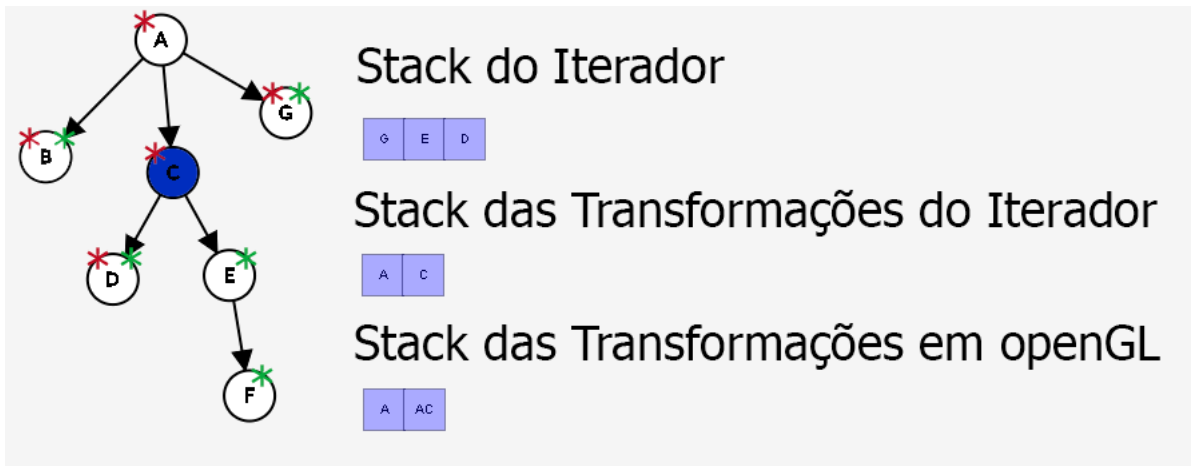


Figura 5. Obter o nodo C

- É tirado o nodo C da stack
- **Quais os filhos adicionados à stack?** D e E
- **Qual a última transformação de C?** A
- **Quantos pops da stack das transformações do iterador temos que dar até que o topo da stack seja igual à última transformação de C?** 1 (Nodo B)
- **Nodo C contém transformação?** Sim, então insere-o na stack das transformações do Iterador
- **Quantos pops da matriz em OpenGL devemos dar?** 1
- **Devemos dar push da matriz em OpenGL?** Sim
- **Qual a transformação atual em OpenGL?** A+C
- **Modelos desenhados:** nenhum

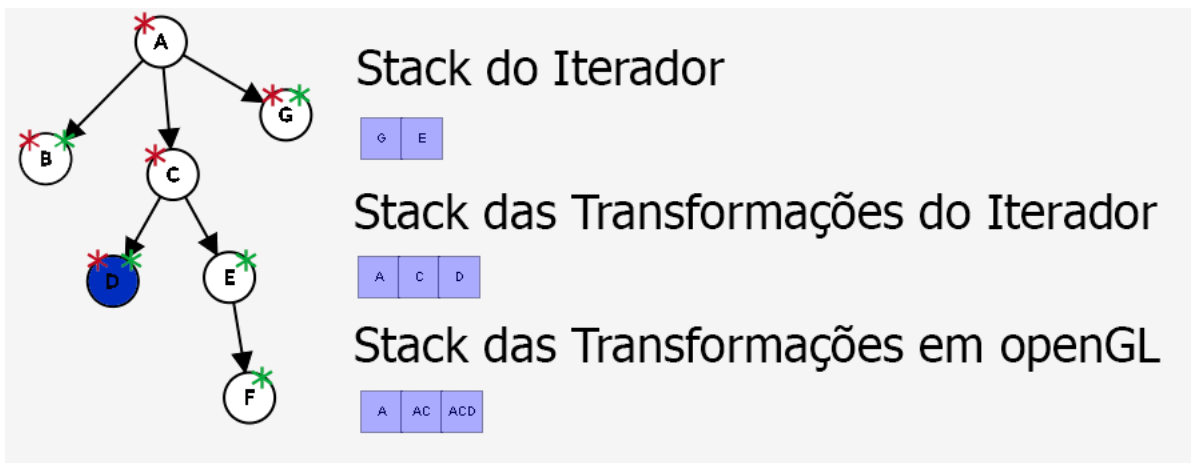


Figura 6. Obter o nodo D

- É tirado o nodo D da stack
- **Quais os filhos adicionados à stack?** Nenhum
- **Qual a última transformação de D?** C
- **Quantos pops da stack das transformações do iterador temos que dar até que o topo da stack seja igual à última transformação de D?** 0
- **Nodo D contém transformação?** Sim, então insere-o na stack das transformações do Iterador

- Quantos pops da matriz em OpenGL devemos dar? 0
- Devemos dar push da matriz em OpenGL? Sim
- Qual a transformação atual em OpenGL? $A+C+D$
- Modelos desenhados:
 1. Modelo D (*Transformações A, C e D aplicadas*)

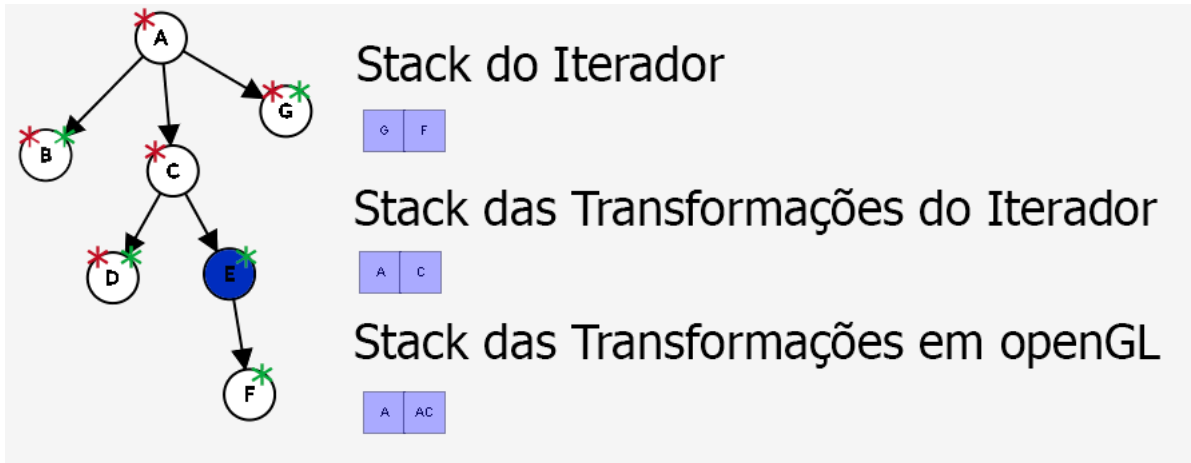


Figura 7. Obter o nodo E

- É tirado o nodo E da stack
- Quais os filhos adicionados à stack? F
- Qual a última transformação de E? C
- Quantos pops da stack das transformações do iterador temos que dar até que o topo da stack seja igual à última transformação de E? 1 (Nodo D)
- Nodo E contém transformação? Não
- Quantos pops da matriz em OpenGL devemos dar? 1
- Devemos dar push da matriz em OpenGL? Não
- Qual a transformação atual em OpenGL? $A+C$
- Modelos desenhados:
 1. Modelo E (*Transformações A e C aplicadas*)

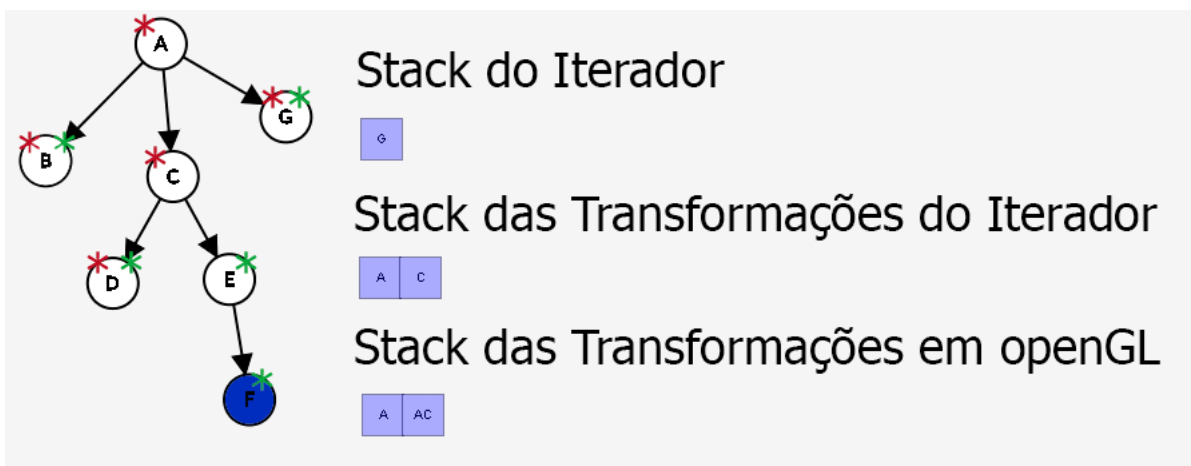


Figura 8. Obter o nodo F

- É tirado o nodo F da stack

- Quais os filhos adicionados à stack? Nenhum
- Qual a última transformação de F? C
- Quantos pops da stack das transformações do iterador temos que dar até que o topo da stack seja igual à última transformação de F? 0
- Nodo F contém transformação? Não
- Quantos pops da matriz em OpenGL devemos dar? 0
- Devemos dar push da matriz em OpenGL? Não
- Qual a transformação atual em OpenGL? A+C
- Modelos desenhados:
 1. Modelo F (*Transformações A e C aplicadas*)

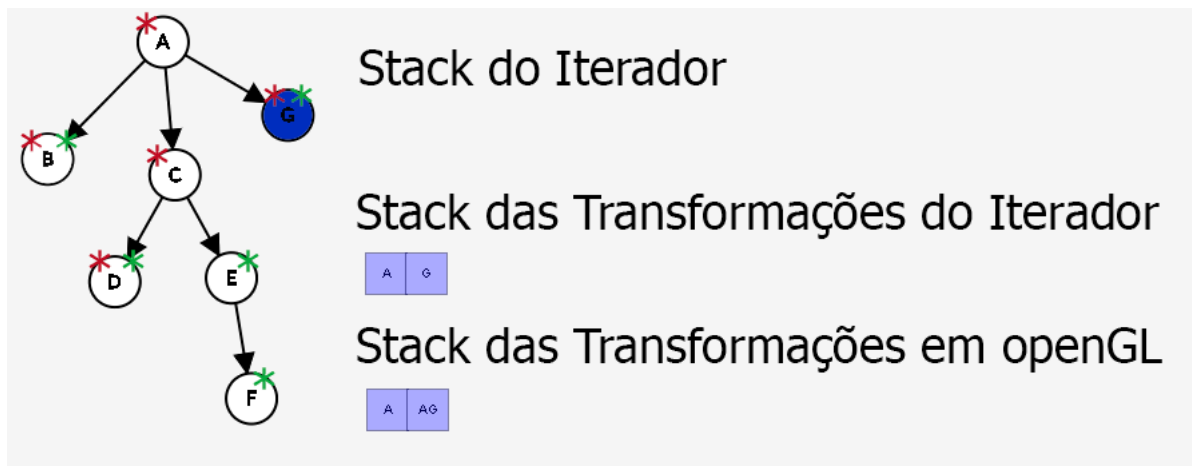


Figura 9. Obter o nodo G

- É tirado o nodo G da stack
- Quais os filhos adicionados à stack? Nenhum
- Qual a última transformação de G? A
- Quantos pops da stack das transformações do iterador temos que dar até que o topo da stack seja igual à última transformação de G? 1 (Nodo C)
- Nodo G contém transformação? Sim, então insere-o na stack das transformações do iterador
- Quantos pops da matriz em OpenGL devemos dar? 1
- Devemos dar push da matriz em OpenGL? Sim
- Qual a transformação atual em OpenGL? A+G
- Modelos desenhados:
 1. Modelo G (*Transformações A e G aplicadas*)

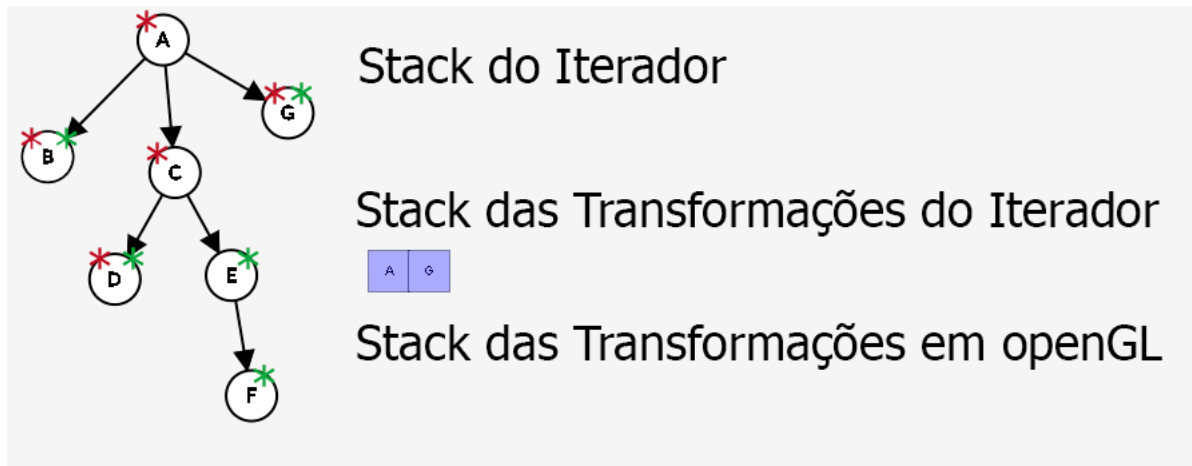


Figura 10. Fim do Iterador

- Para fazermos com que a matriz de openGL volte ao estado inicial, devemos dar pops, sendo que essa quantidade é o tamanho da **Stack das Transformações do Iterador** (neste caso, 2)

Após todo este processo, a matriz de openGL volta ao estado inicial e os modelos desenhados são:

1. Modelo B (*Transformações A e B aplicadas*)
2. Modelo D (*Transformações A, C e D aplicadas*)
3. Modelo E (*Transformações A e C aplicadas*)
4. Modelo F (*Transformações A e C aplicadas*)
5. Modelo G (*Transformações A e G aplicadas*)

Para ilustrar este exemplo, usamos um ficheiro XML, **demo.xml**, onde:

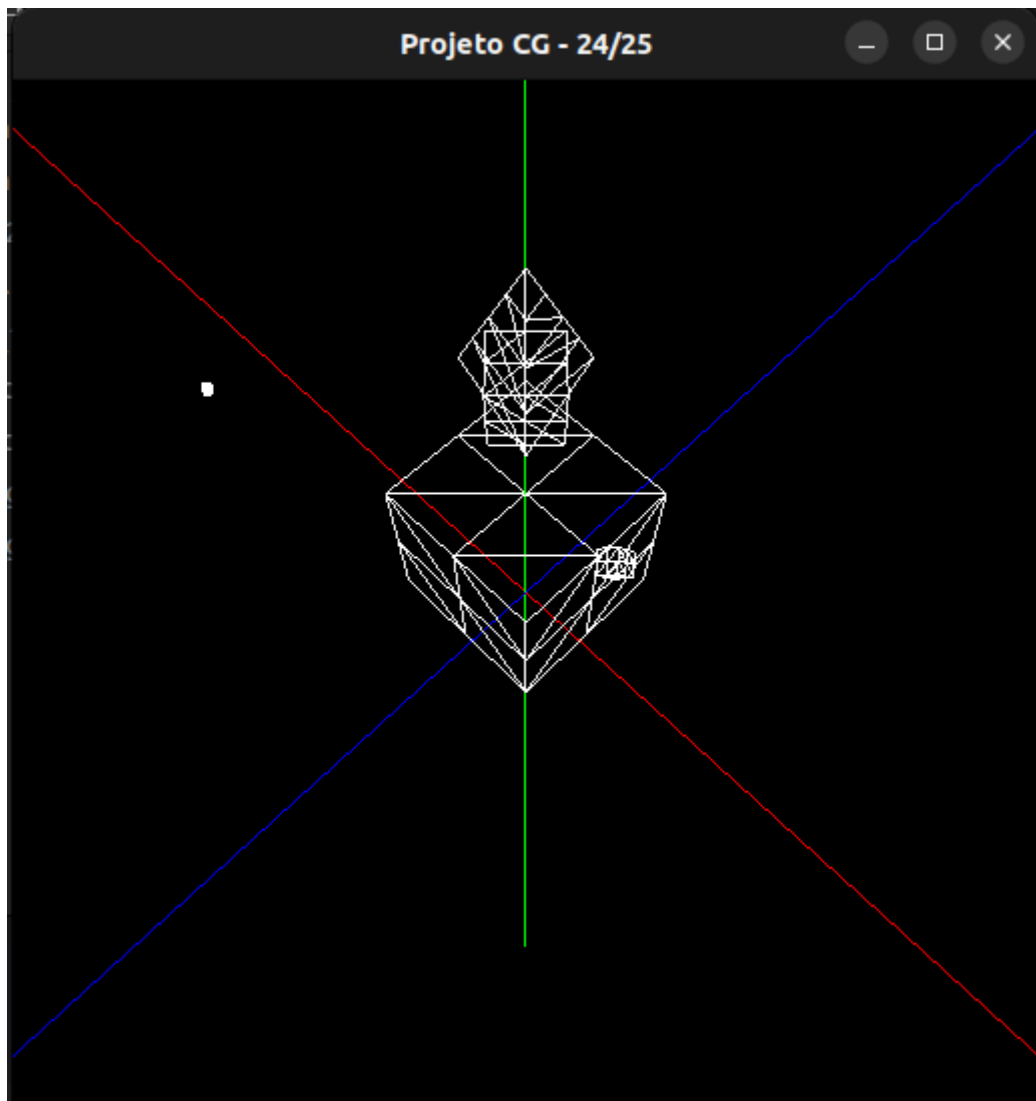


Figura 11. Demo que representa o exemplo apresentado

No nodo:

1. **Nodo A:**
 - **Transformações** : Translação (0,2,0)
 - **Modelos** : não há
2. **Nodo B:**
 - **Transformações** : Escala (3,3,3)
 - **Modelos** : Caixa
3. **Nodo C:**
 - **Transformações** : Rotação (45° ,0,1,0) , Translação (0,4,0)
 - **Modelos** : não há
4. **Nodo D:**
 - **Transformações** : Translação (-4,0,0) , Escala(0.1,0.1,0.1)
 - **Modelos** : Caixa
5. **Nodo E:**
 - **Transformações** : não há
 - **Modelos** : Caixa
6. **Nodo F:**

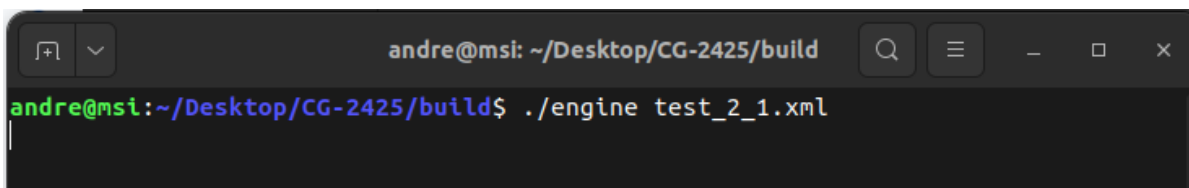
- **Transformações** : não há
 - **Modelos** : Cone
7. **Nodo G:**
- **Transformações** : Rotação ($55^\circ, 1, 1, 1$) , Escala (0.5,0.5,0.5) , Translação (0,-5,0)
 - **Modelos** : Caixa

5. Demos

Aqui estão representadas as demos que fizemos (tanto as que foram fornecidas pelo enunciado, bem como as nossas) para esta fase do trabalho prático da UC Computação Gráfica.

5.1. Demos do Enunciado

5.1.1. Ficheiro test_2_1.xml

A terminal window with a dark background. The title bar shows 'andre@msi: ~/Desktop/CG-2425/build'. The prompt is 'andre@msi:~/Desktop/CG-2425/build\$' and the command entered is './engine test_2_1.xml'.

```
andre@msi: ~/Desktop/CG-2425/build
andre@msi:~/Desktop/CG-2425/build$ ./engine test_2_1.xml
```

Figura 12. Execução da Engine do Teste 1 da Fase 2

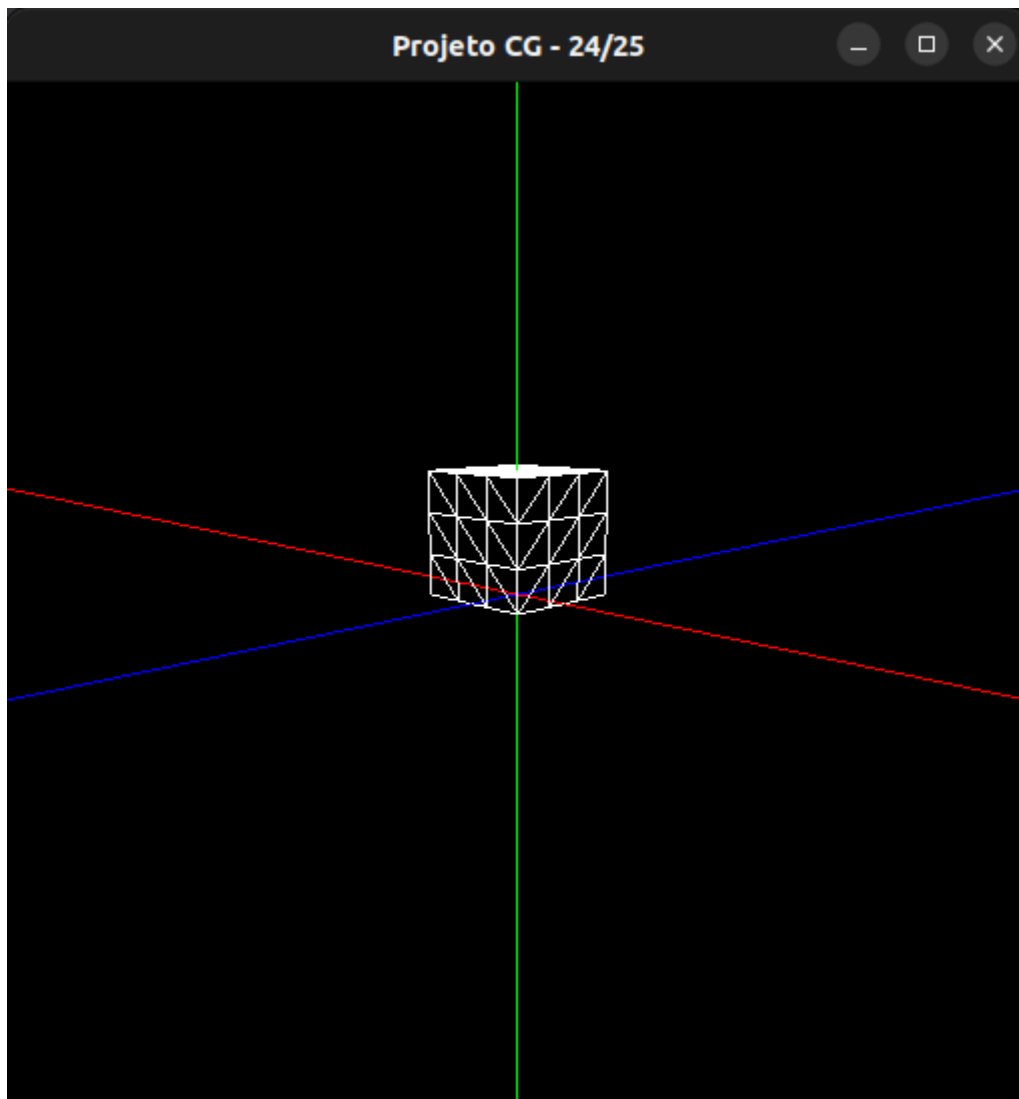


Figura 13. Cenário desenhado pela Engine descrito pelo ficheiro XML Teste 1 da Fase 2

5.1.2. Ficheiro test_2_2.xml

```
andre@msi: ~/Desktop/CG-2425/build
andre@msi:~/Desktop/CG-2425/build$ ./engine test_2_2.xml
```

Figura 14. Execução da Engine do Teste 2 da Fase 2

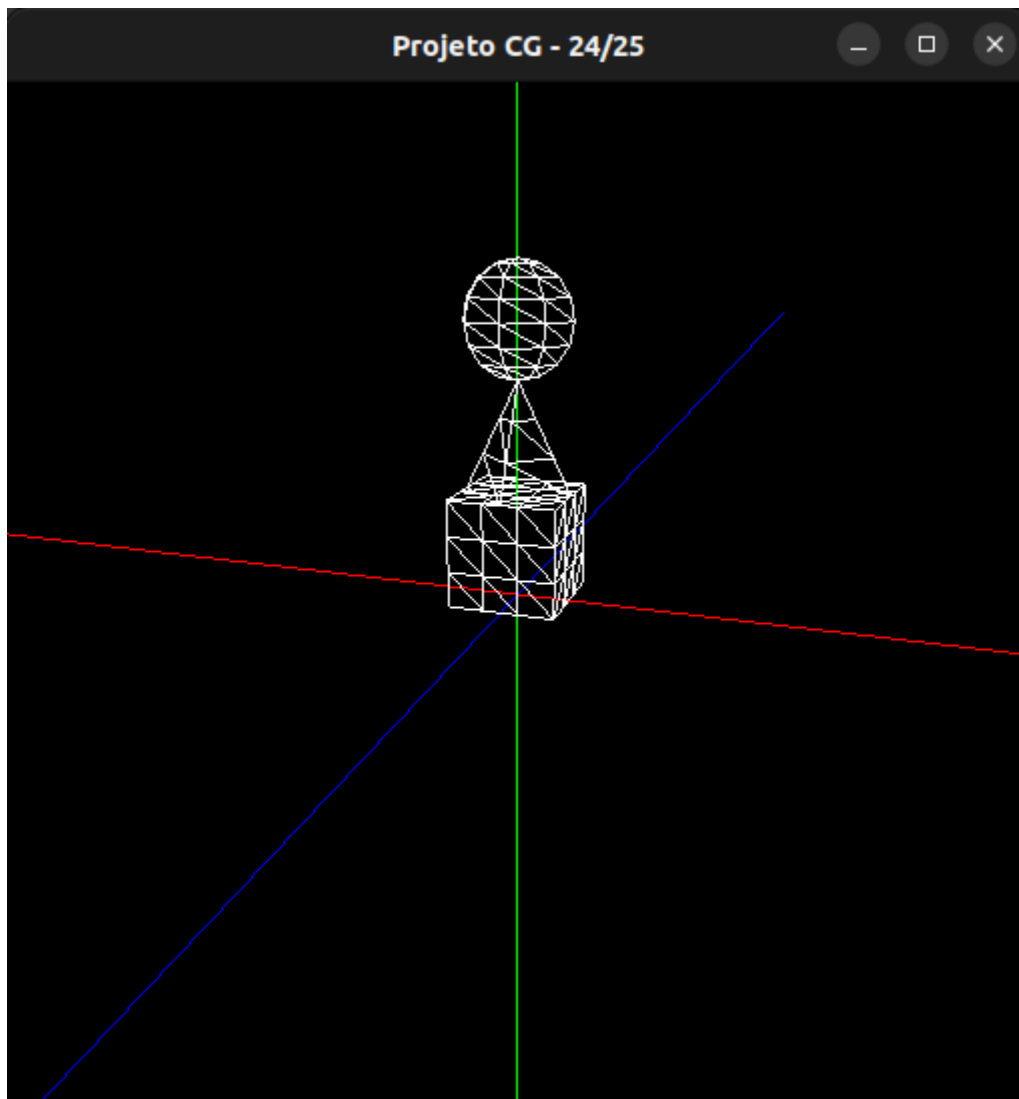


Figura 15. Cenário desenhado pela Engine descrito pelo ficheiro XML Teste 2 da Fase 2

5.1.3. Ficheiro test_2_3.xml

```
andre@msi: ~/Desktop/CG-2425/build
andre@msi:~/Desktop/CG-2425/build$ ./engine test_2_3.xml
```

Figura 16. Execução da Engine do Teste 3 da Fase 2

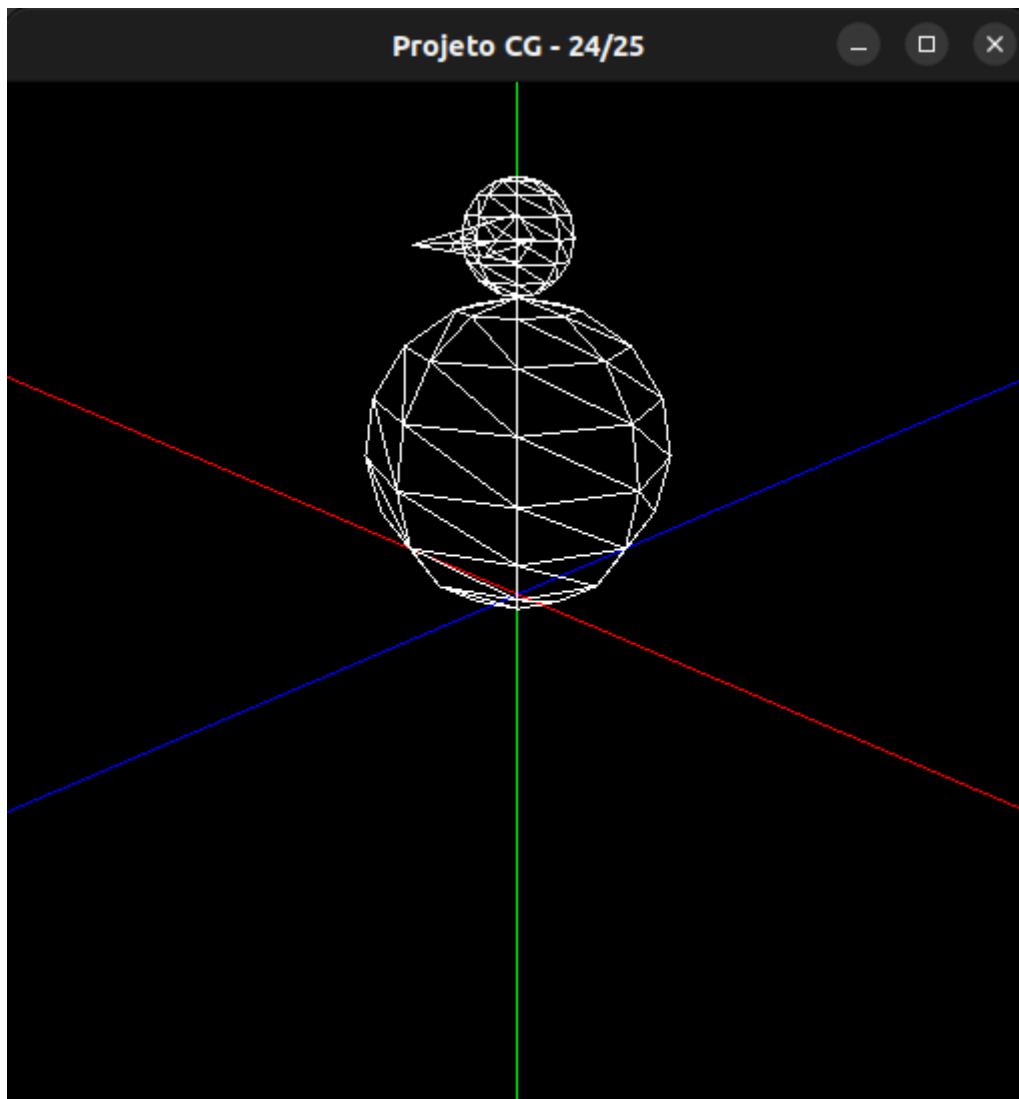


Figura 17. Cenário desenhado pela Engine descrito pelo ficheiro XML Teste 3 da Fase 2

5.1.4. Ficheiro test_2_4.xml

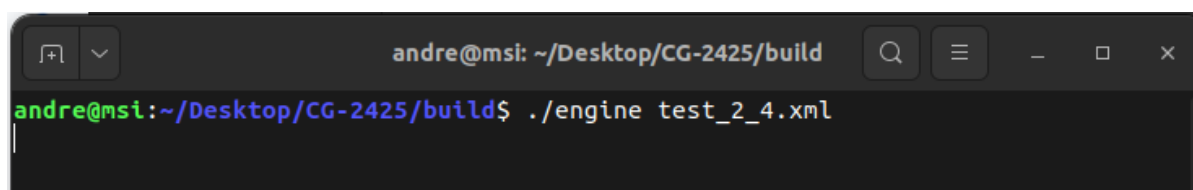


Figura 18. Execução da Engine do Teste 4 da Fase 2

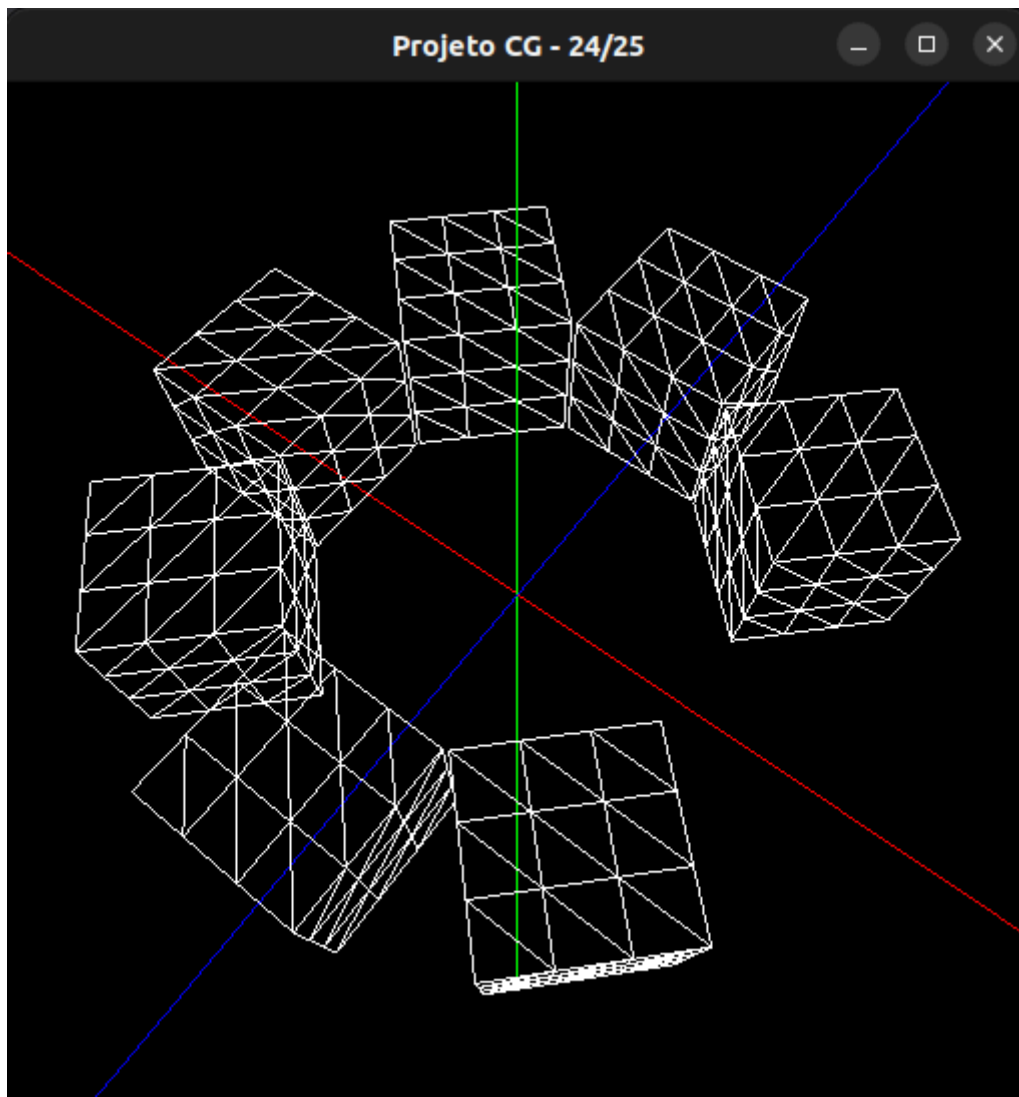


Figura 19. Cenário desenhado pela Engine descrito pelo ficheiro XML Teste 4 da Fase 2

5.2. As nossas Demos

5.2.1. Ficheiro demo.xml

```
andre@msi: ~/Desktop/CG-Grupo02-Fase2/Demo - Fase2
andre@msi:~/Desktop/CG-Grupo02-Fase2/Demo - Fase2$ ./engine demo.xml
```

Figura 20. Execução da Engine da Demo 1 da Fase 2

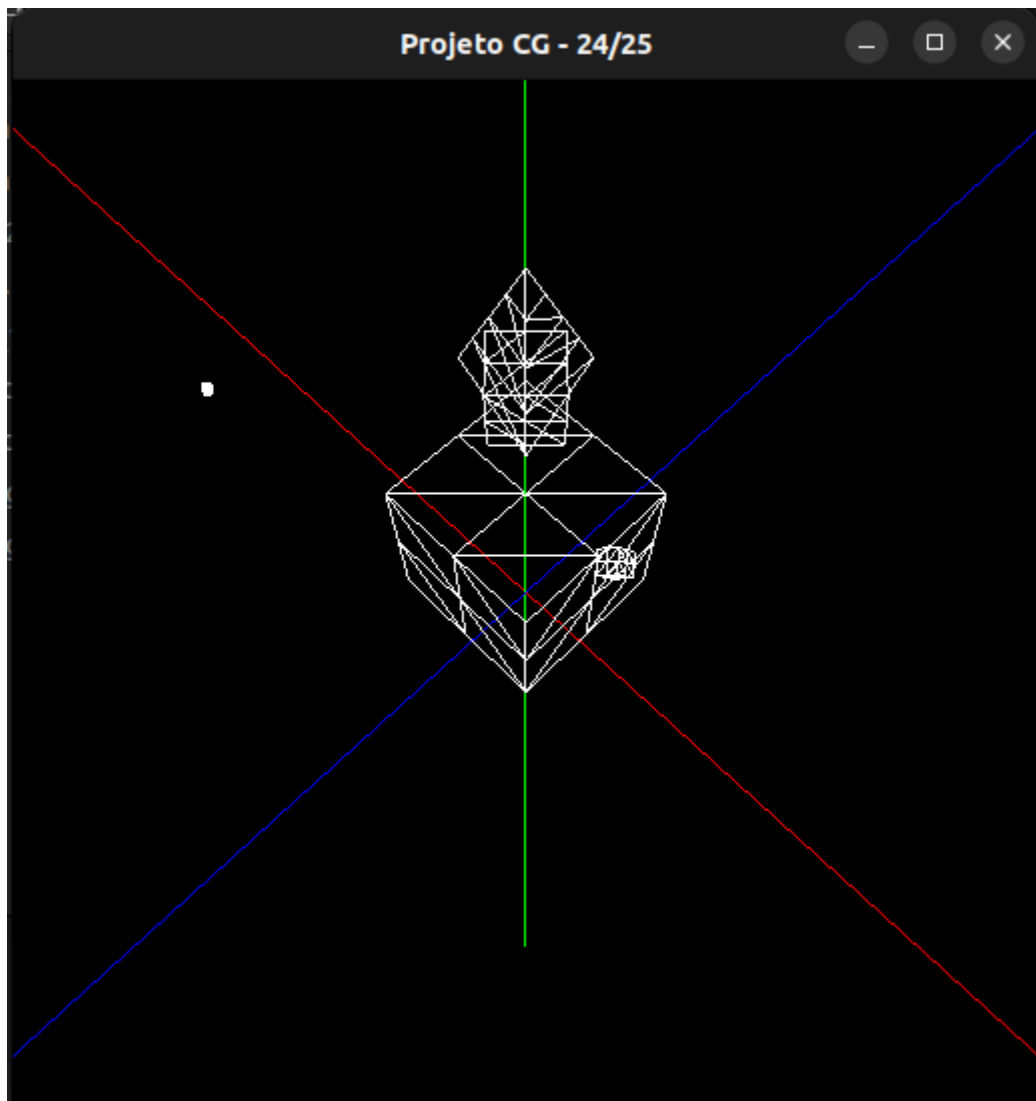


Figura 21. Cenário desenhado pela Engine descrito pelo ficheiro XML Demo da Fase 2

5.2.2. Ficheiro sistema_solar.xml

Neste sistema solar estão representados os planetas, o sol e a lua. Para fazer esta demo, o ficheiro XML definido segue a mesma estrutura dos ficheiros XML fornecidos. Cada planeta ou conjunto planeta e lua é definido como um **group** e contém as transformações a aplicar e o modelo a desenhar. Para posicionar os planetas, usamos a seguinte sequência de transformações: rotação, translação e escala. A rotação foi utilizada para definir a posição angular do planeta, a translação foi aplicada para posicionar ou afastar corretamente o planeta em relação ao sol, isto é, a distância entre eles, e a escala foi aplicada para ajustar o tamanho de cada planeta.

Além disso, o planeta Terra e a Lua encontram-se no mesmo grupo para que a Lua seja posicionada em relação ao planeta Terra. Na lua apenas é aplicada uma translação em relação ao planeta Terra e uma escala para diminuir o seu tamanho.

```
andre@msi: ~/Desktop/CG-Grupo02-Fase2/Demo - Fase2
andre@msi:~/Desktop/CG-Grupo02-Fase2/Demo - Fase2$ ./engine sistema_solar.xml
```

Figura 22. Execução da Engine do Sistema Solar da Fase 2

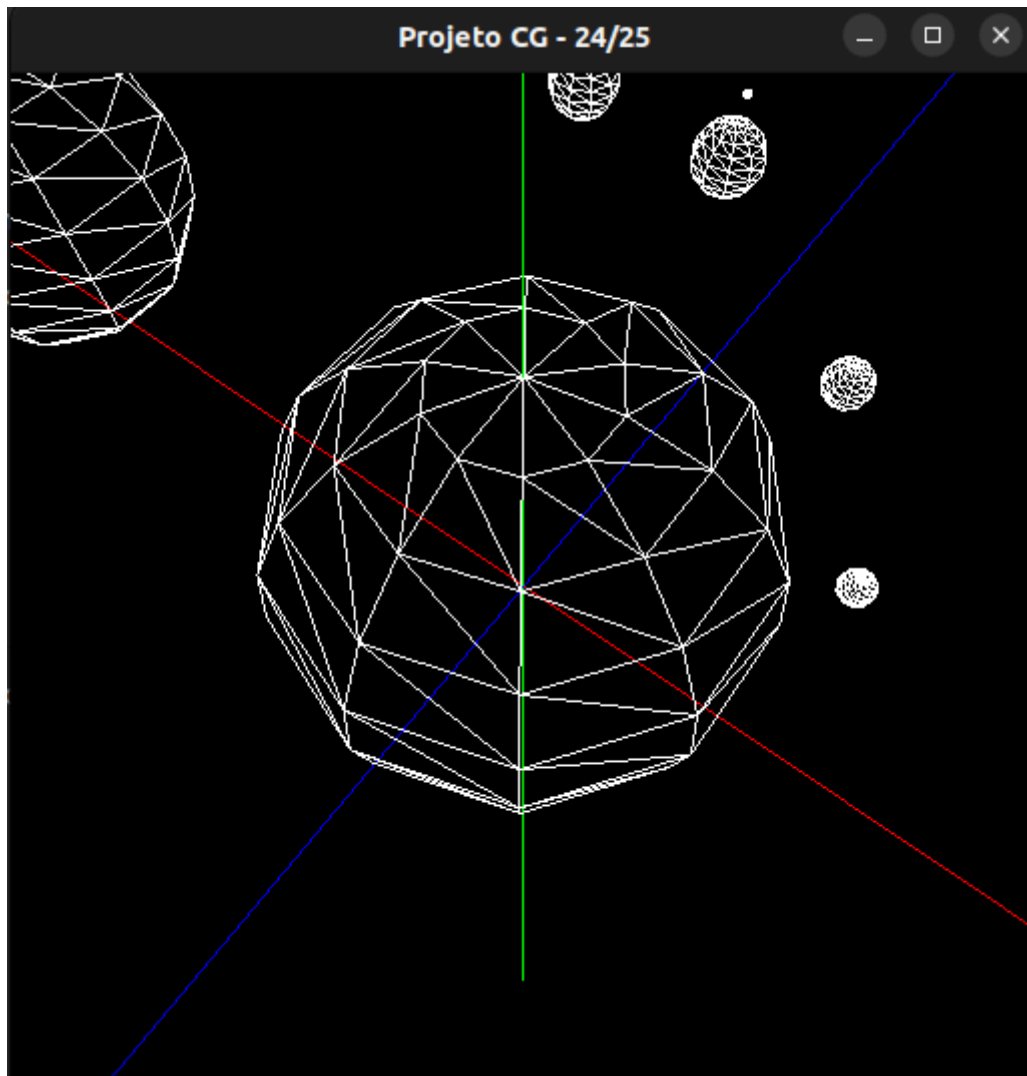


Figura 23. Cenário desenhado pela Engine descrito pelo ficheiro XML Sistema Solar da Fase 2

6. Conclusão

Com este projeto, conseguimos realizar as transformações geométricas em cenários hierárquicos, algo pedido nesta fase. Alteramos o parser de XML e a engine para que interprete o XML como uma RoseTree, em vez de uma lista de modelos, tal como tínhamos na primeira fase. Nessa RoseTree há informações sobre cada nodo descrito no ficheiro XML. Temos também um Iterador que vai iterar sobre essa árvore e indicar à

engine quais os procedimentos deve tomar para desenhar o cenário descrito no ficheiro XML.

Nesta fase conseguimos aprender como aplicar transformações, isto é, translações, rotações e escalas, e conseguimos aprender também como usar as funções de Pop e Push das matrizes de OpenGL e o quão úteis estas ferramentas são para compor um cenário.

Conseguimos aplicar técnicas de otimizações e melhorias de desempenho em OpenGL, fazendo com que as figuras repetidas só sejam carregadas a partir de um ficheiro .3d uma única vez. Além disso, permite à engine evitar transformações desnecessárias ou redundantes, fazendo com que apenas se realize Push e Pop das matrizes em OpenGL quando estas são necessárias. Isto evita ter que repetir transformações.

Futuramente queremos adicionar novos modelos mais complexos e uma câmara livre para enriquecer este trabalho.