Computação Gráfica
Projeto Prático: Fase 2

Licenciatura em Ciências da Computação

Grupo 3

Alef Keuffer (A91683), Alexandre Rodrigues Baldé (A70373),
Pedro Paulo Costa Pereira (A88062)

Notas sobre Implementação

Nesta fase, apenas foram solicitadas alterações na aplicação *Engine* e no nosso *parser* de ficheiros XML.

Como os ficheiros XML podem conter grupos hierárquicos, o grupo decidiu que a melhor forma seria criarmos constantes para diferenciar:

- os tipos de transformações (translação, rotação ou escala);
- os ficheiros-modelos que deverão ser lidos;
- e o começo e fim de cada grupo.

Assim, a cada vez que é detetado o início de um novo grupo, faz-se **glPushMatrix()**, que é compensado por um **glPopMatrix()** ao final desse grupo. Dessa forma tem-se a certeza de que os filhos herdam as transformações dos pais, mas os irmãos não.

Tudo o que é lido pelo *parser* é inserido num vetor de *floats* chamado **operations**, e as constantes declaradas são os "pontos de referência", para saber o que vem a seguir.

```
void
operations_push_transforms(tinyxml2::XMLElement
*transforms, std::vector<float> *operations) {
	tinyxml2::XMLElement *transform =
transforms->FirstChildElement();

	do {
	const char *transformValue =
transform->Value();

	if (!strcmp("translate", transformValue))
```

```
            operations->push_back(TRANSLATE);
      else if (!strcmp("rotate", transformValue))
            operations->push_back(ROTATE);
      else if (!strcmp("scale", transformValue))
            operations->push_back(SCALE);
      else {
            fprintf(stderr, "Unknown transform:
\"%s\"", transformValue);
            exit(1);
      }


operations_push_transform_attributes(transform,
operations);

      } while ((transform =
transform->NextSiblingElement()));
}
```

Por exemplo, quando o *parser* encontra um elemento "transform", ele itera pelos filhos desse elemento, comparando seus valores com os tipos de transformação. De seguida, ele insere no vetor **operations** a constante referente à transformação correta, seguida dos valores de seus atributos, como pode ser visto no excerto de código acima.

Dessa forma, a função **operations_render()** (que é chamada na **renderScene**) corre pelo vetor operations com um switch-case; quando encontra um valor correspondente à constante de translação, por exemplo, a função **glRotatef()** é chamada com os 4 seguintes elementos do vetor. A seguir, apresenta-se uma parte desta função.

```
{...}
for (i += 3; i < operations->size (); i++)
      {
      switch ((int) operations->at (i))
      {
      case ROTATE:
      glRotatef (operations->at (i + 1),
                 operations->at (i + 2),
                 operations->at (i + 3),
                 operations->at (i + 4));
            i += 4;
            continue;
      case TRANSLATE:
      glTranslatef (operations->at (i + 1),
                    operations->at (i + 2),
                    operations->at (i + 3));
            i += 3;
            continue;
      case SCALE:
      glScalef (operations->at (i + 1),
                 operations->at (i + 2),
                 operations->at (i + 3));
            i += 3;
            continue;
      case BEGIN_GROUP:glPushMatrix ();
            continue;
      case END_GROUP:glPopMatrix ();
            continue;
{...}
```

Model Scene – sistema solar

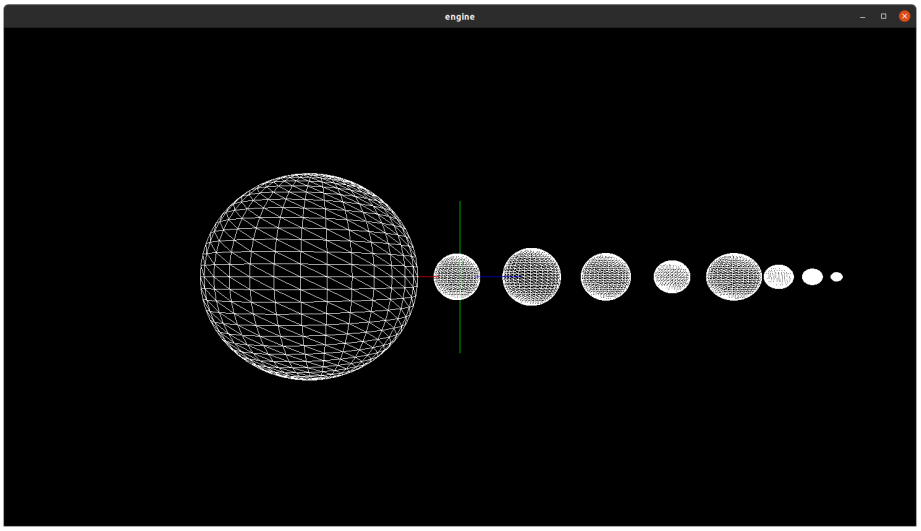
Para demonstrar as funcionalidades dessa melhoria à aplicação, foi solicitada como *model scene* a representação do sistema solar (estático), incluindo o sol, os planetas e as luas.

O grupo decidiu por fazer uma abordagem mais simples, apenas representando o sol e os planetas, com tamanhos não muito proporcionais aos tamanhos reais. Assim, foi criado o ficheiro "solar_system.xml", do qual apresenta-se um excerto a seguir:

```
<group>
      <models>
            <model file="sun.3d" />
      </models>
      <group>
            <transform>
                  <translate x="MERCURY" y="0" z="0"
/>
            </transform>
             <models>
                   <model file="mercury.3d" />
             </models>
      </group>
      <group>
            <transform>
                  <translate x="VENUS" y="0" z="0"
/>
            </transform>
            <models>
                  <model file="venus.3d" />
            </models>
```

```
    </group>
```

E, correndo o comando `./engine solar_system.xml`, obtém-se a *model scene* abaixo.

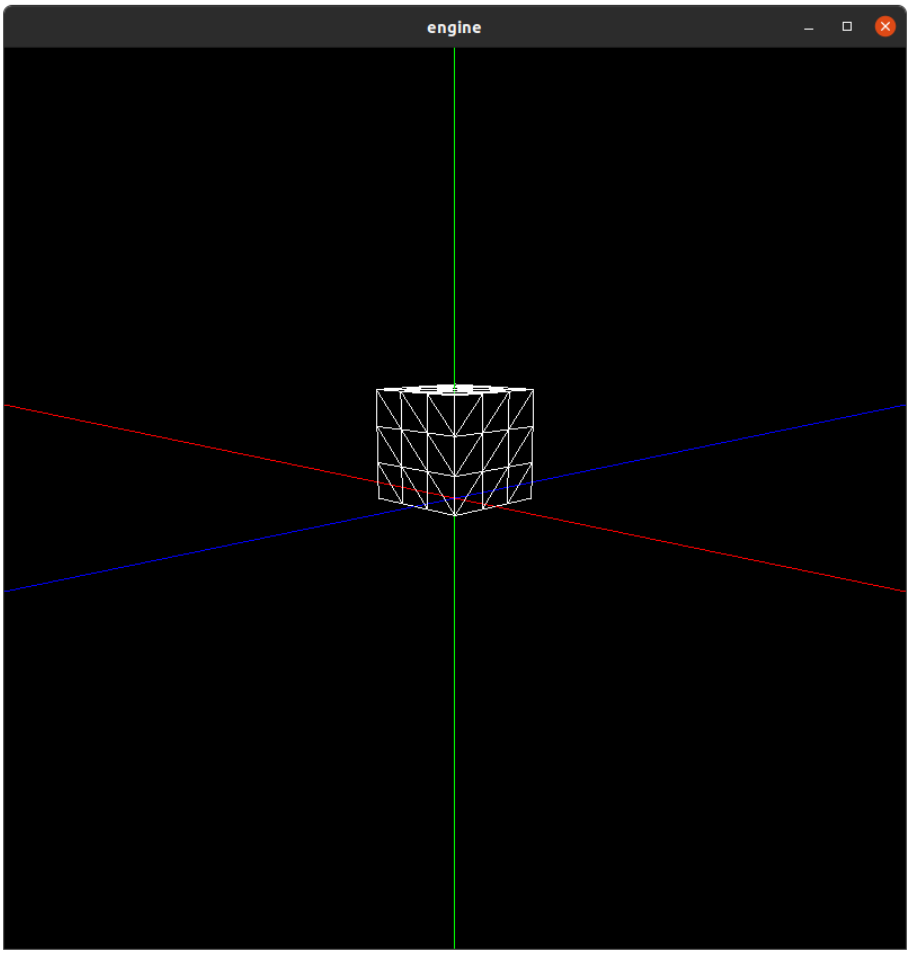


É importante dizer que o grupo percebe que tal representação do sistema solar não é a mais adequada e pretende apresentar uma melhoria de tal representação na próxima fase. Entretanto, considera-se que a *model scene* apresentada é suficiente para demonstrar que a implementação das funcionalidades exigidas está correta.

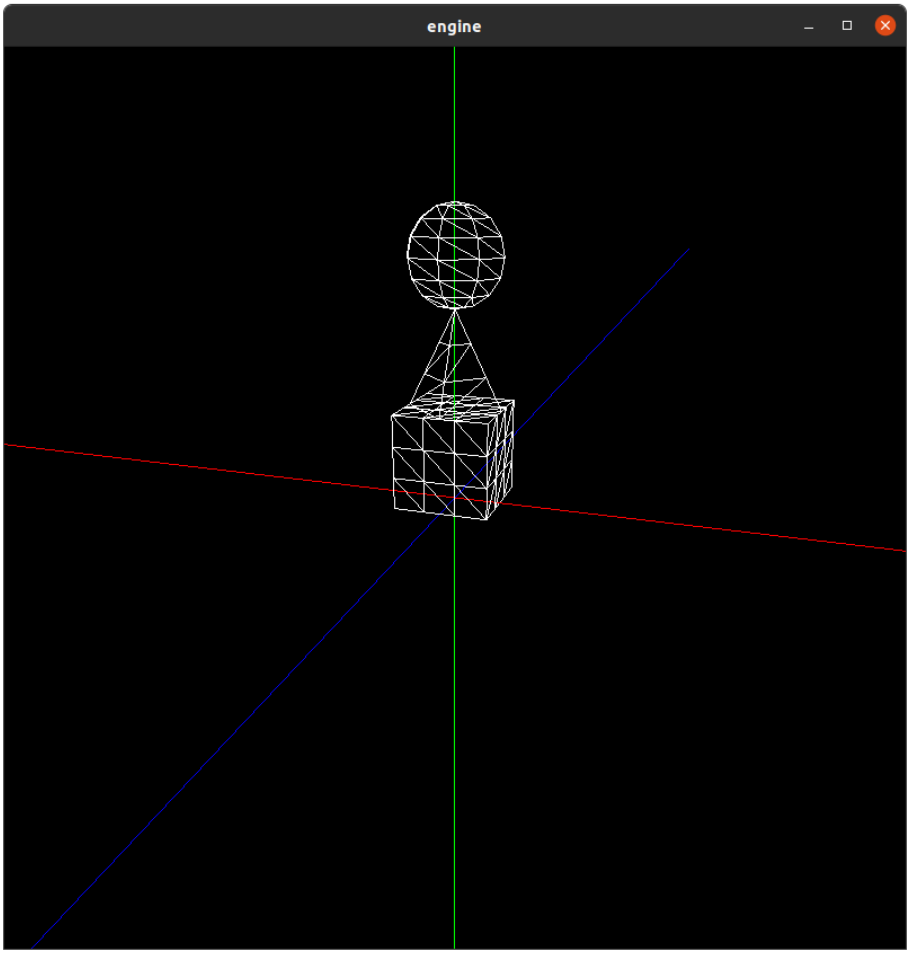
Exemplos de utilização

Apresentam-se agora os resultados de correr a aplicação produzida nos ficheiros de teste XML fornecidos, pela ordem que foram dados.

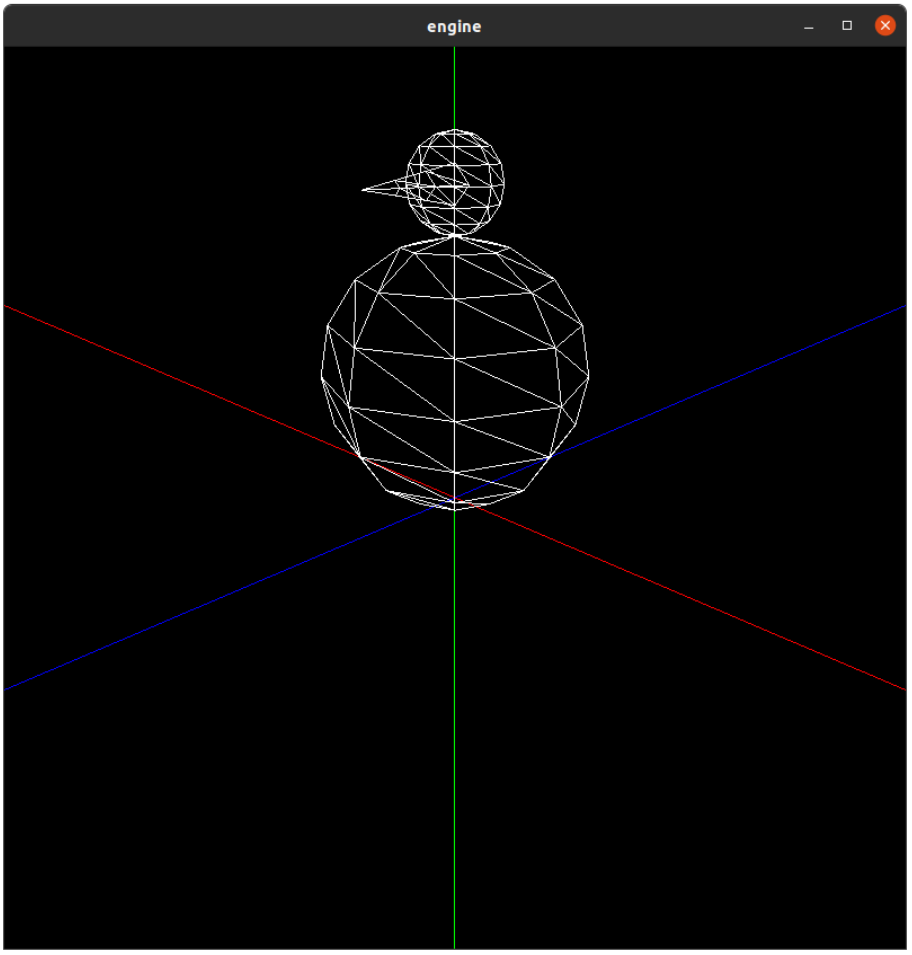
`test_files_phase_2/test_2_1.xml`



test_files_phase_2/test_2_2.xml



`test_files_phase_2/test_2_3.xml`



`test_files_phase_2/test_2_4.xml`