# Computação Gráfica

Trabalho Prático - Fase 2

Universidade do Minho, Departamento de Informática
José Correia, Diogo Abreu, e Rodrigo Monteiro
{100610, 100646, 100706}@alunos.uminho.pt

Grupo 69

## 1. Introdução

Para esta segunda fase do projeto tem-se como objetivo dar seguimento ao trabalho realizado na primeira fase, expandindo as funcionalidades da Engine. Sendo o foco principal implementar a capacidade de aplicar transformações geométricas aos modelos de forma hierárquica, utilizando instruções em formato XML.

## 2. Leitura dos grupos XML

### 2.1. Classe `Transform`

```
class Transform {
public:
    enum TransformType { TRANSLATE = 1, SCALE = 2, ROTATE = 3 };

    Transform(float x, float y, float z, float angle, TransformType type);
    ~Transform();

    float x, y, z, angle;
    TransformType type;
};
```

### 2.2. Classe `Group`

```
class Group {
public:
    Group();
    ~Group();

    std::vector<std::string> models;
    std::vector<Transform*> transforms;
    std::vector<Group*> groups;

};
```

A classe `Group` tem um comportamento semelhante a uma *rose tree*, visto que cada nodo pode ter um conjunto variável de ramos. Assim, para além de um grupo conter um conjunto de *models* e de *transforms*, também contém um conjunto de outros grupos.

### 2.3. Parsing dos grupos

```
Group* Config::parse_groups(tinyxml2::XMLElement* xml_group) {
    if (xml_group) {
        Group* group = new Group();

        // Transforms
        XMLElement* tr = xml_group->FirstChildElement("transform");
        if (tr) {
            for (XMLElement* t = tr>FirstChildElement(); t;
                 t = t->NextSiblingElement()) {
                // ...
                group->transforms.push_back(new Transform(x, y, z, angle, type));
            }
        }

        // Models
        XMLElement* xml_models = xml_group->FirstChildElement("models");
        if (xml_models) {
            for (XMLElement* m = xml_models>FirstChildElement("model"); m;
                 m = m->NextSiblingElement()) {
                group->models.push_back((m->Attribute("file")));
            }
        }

        // Groups
        for (XMLElement* g = xml_group->FirstChildElement("group"); g;
             g = g->NextSiblingElement("group")) {
            Group* child = parse_groups(g);
            group->groups.push_back(child);
        }

        return group;
    }
    return nullptr;
}
```

Começando na `root` e depois de criada uma instância da classe `Group`, percorrem-se os elementos dentro de `transform` - do tipo *translate*, *scale* e *rotate* - e estes são adicionados ao grupo. Depois, percorrem-se os elementos dentro de `models`, e estes são também adicionados ao grupo. Por fim, por cada grupo dentro do grupo chama-se outra vez a função `parse_groups`. Desta forma todos os grupos são percorridos recursivamente até serem atingidos os casos de paragem, sendo formada uma estrutura semelhante a uma *rose tree*.

## 3. Renderização dos grupos

### 3.1. Código

```
void drawGroups(Group* group) {
  if (group) {
    glPushMatrix();

    for (Transform* t : group->transforms) {
      switch (t->type) {
        case Transform::TransformType::TRANSLATE:
          glTranslatef(t->x, t->y, t->z);
          break;
        case Transform::TransformType::SCALE:
          glScalef(t->x, t->y, t->z);
          break;
        case Transform::TransformType::ROTATE:
          glRotatef(t->angle, t->x, t->y, t->z);
          break;
        default:
          break;
      }
    }

    std::vector<Figure*> figures;
    for (const auto& model : group->models)
      figures.push_back(Figure::from_file(model));

    glBegin(GL_TRIANGLES);
    drawFigures(figures);
    glEnd();

    for (Group* g : group->groups)
      drawGroups(g);

    glPopMatrix();
  }
}
```

É mantida uma "máquina de estados" com o comportamento de uma *stack* de modo a evitar o cálculo e a multiplicação por matrizes inversas, quando se quer reverter transformações feitas. Para iniciar um novo estado com base no anterior (se exisitr), é efetuado um `glPushMatrix`. De seguida, podem ser efetuadas diversas transformações sobre esse estado através de multiplicação de matrizes. E, por fim, é possível voltar ao estado anterior, quando é efetuado um `glPopMatrix`.

Assim, um grupo para além das suas transformações, também possui as transformações dos grupos em que está contido, mas não possui as transformações de grupos vizinhos ou de grupos que contém.

## 3.2. Matrizes

$$\text{translate, } T = \begin{bmatrix} 1 & 0 & 0 & Vx \\ 0 & 1 & 0 & Vy \\ 0 & 0 & 1 & Vz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{scale, } S = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

$$\text{rotate, } Rx = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\text{sen}\alpha & 0 \\ 0 & \text{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Ry = \begin{bmatrix} \cos\beta & 0 & \text{sen}\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\text{sen}\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Rz = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\text{sen}\gamma & 0 \\ 0 & \text{sen}\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{rotate, } R = \begin{bmatrix} \cos\beta & 0 & \text{sen}\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\text{sen}\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\text{sen}\alpha & 0 \\ 0 & \text{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\text{sen}\gamma & 0 \\ 0 & \text{sen}\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

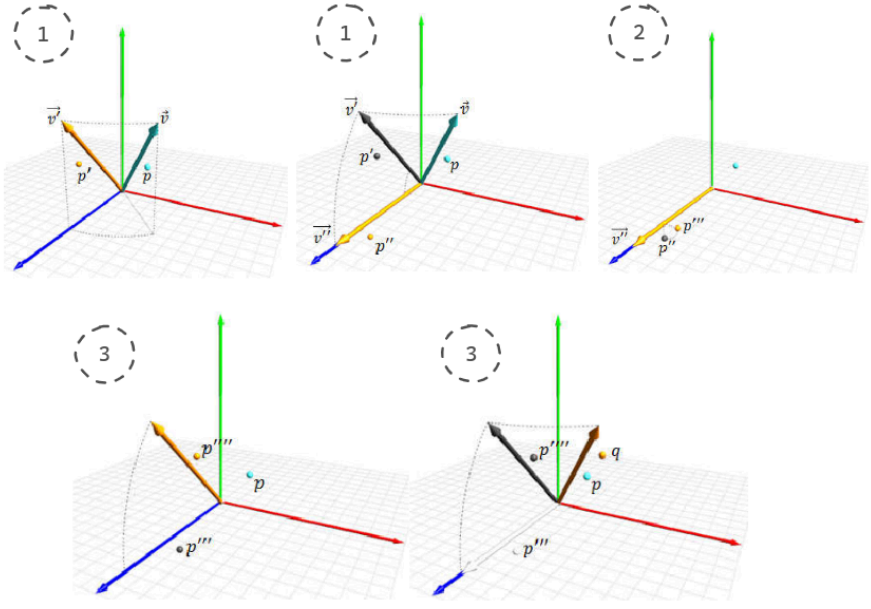
General 3D rotation

$$R = \begin{bmatrix} \cos\beta\cos\gamma & \text{sen}\alpha\text{sen}\beta\cos\gamma - \cos\alpha\text{sen}\gamma & \cos\alpha\text{sen}\beta\cos\gamma + \text{sen}\alpha\text{sen}\gamma & 0 \\ \cos\beta\text{sen}\gamma & \text{sen}\alpha\text{sen}\beta\text{sen}\gamma + \cos\alpha\cos\gamma & \cos\alpha\text{sen}\beta\text{sen}\gamma - \text{sen}\alpha\cos\gamma & 0 \\ -\text{sen}\beta & \text{sen}\alpha\cos\beta & \cos\alpha\cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

Rotation from axis (v) and angle (θ):

1. Rotate vector v so that it aligns with the Z (or X or Y) axis.
2. Rotate the point around the selected axis by θ.
3. Undo the rotations in step 1

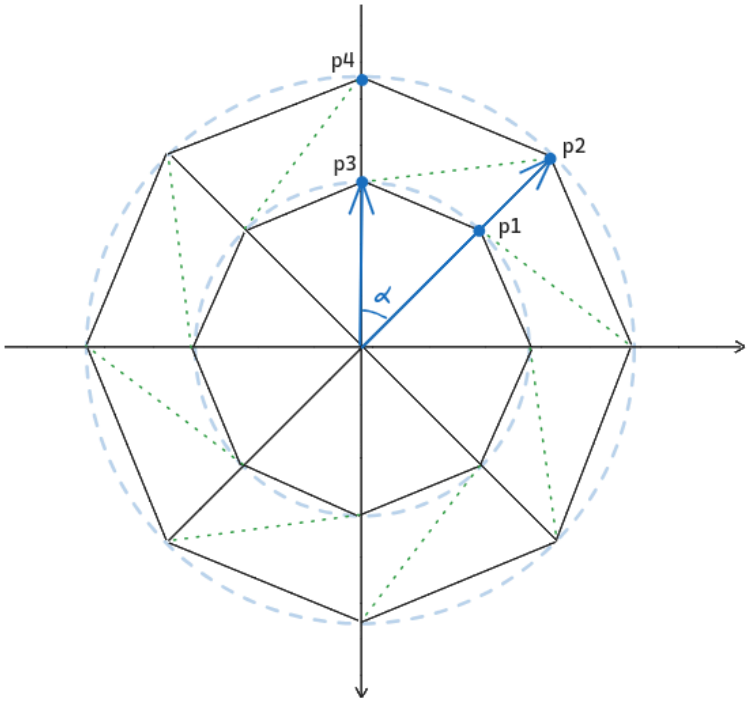


4. **Ring**

Para podermos complementar Saturno com o seu anel, decidimos acrescentar ao generator a primitiva geométrica *ring*, que permite desenhar um anel, tendo como variáveis, o raio interior, o raio exterior e o número de divisões do anel (*slices*).

O anel é construído slice a slice e os seus vértices são deslocados segundo o ângulo $\delta = \frac{2\pi}{\text{slices}}$. Os valores dos vértices são obtidos utilizando coordenadas esféricas.

Cada slice é constituída por 4 triângulos (dois para a face superior, dois para a face inferior), sendo uma slice formada da seguinte forma:

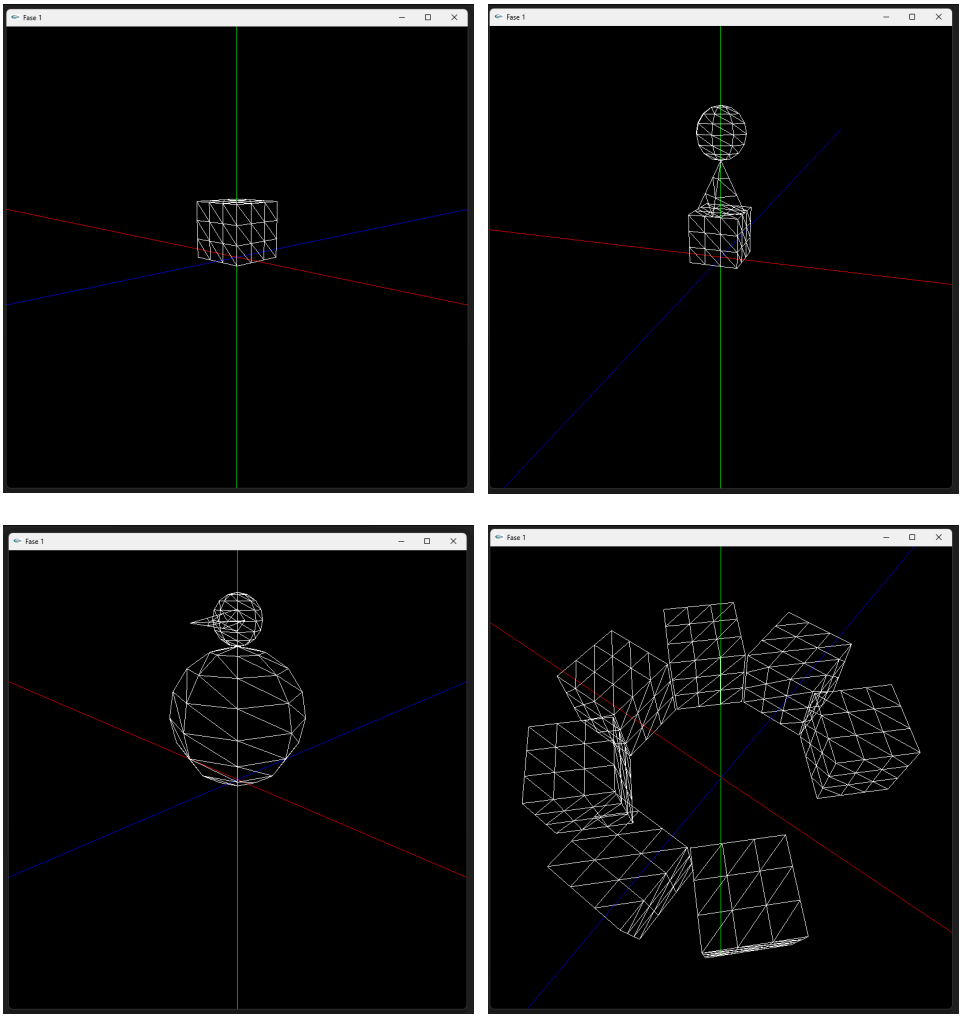



Os pontos p1, p2, p3 e p4 são adicionados ao *vector* de pontos do *ring* pela seguinte ordem: [p1, p2, p3], [p3, p2, p4], [p3, p2, p1] e [p2, p3, p4]. A cada iteração, o valor de $\alpha$ de cada um dos pontos é incrementado pelo valor de $\delta$. Este processo é repetido tantas vezes quanto o número de *slices* fornecido.

## 5. **Output**

### 5.1. **Testes**

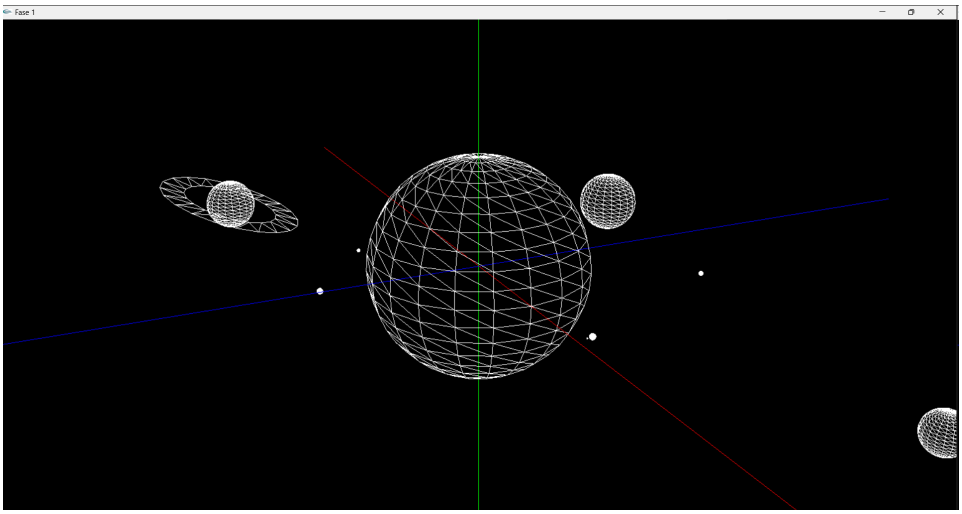
Com a execução dos testes xml, obtivemos todos os resultados pretendidos.



### 5.2. **Sistema Solar**

Na nossa adaptação do Sistema Solar, as proporções foram preservadas com a máxima fidelidade à realidade. Por questões de simplicidade, optamos por descrever apenas o Sol, os nove Planetas e a Lua.

Para tal, criamos um ficheiro xml com este sistema solar, e nomeamos o mesmo de **test_2_5.xml**.




## 6. Conclusão

Ao longo do desenvolvimento desta fase do projeto, foi nos possíveis consolidar os conhecimentos relativos às transformações geométricas, e em particular, das combinações de matrizes de transformação.

Quanto ao trabalho realizado, encontrámo-nos satisfeitos, dado que conseguimos concretizar todas as funcionalidades pretendidas.

## References

1. Wikipedia: Rotation Matrix, `https://en.wikipedia.org/wiki/Rotation_matrix`
2. Swiftless: OpenGL Popping and Pushing Matrices, `https://www.swiftless.com/tutorials/opengl/pop_and_push_matrices.html`