Trabalho de CG1

Carolina, Daniel, Heitor, Leonardo e Mariana

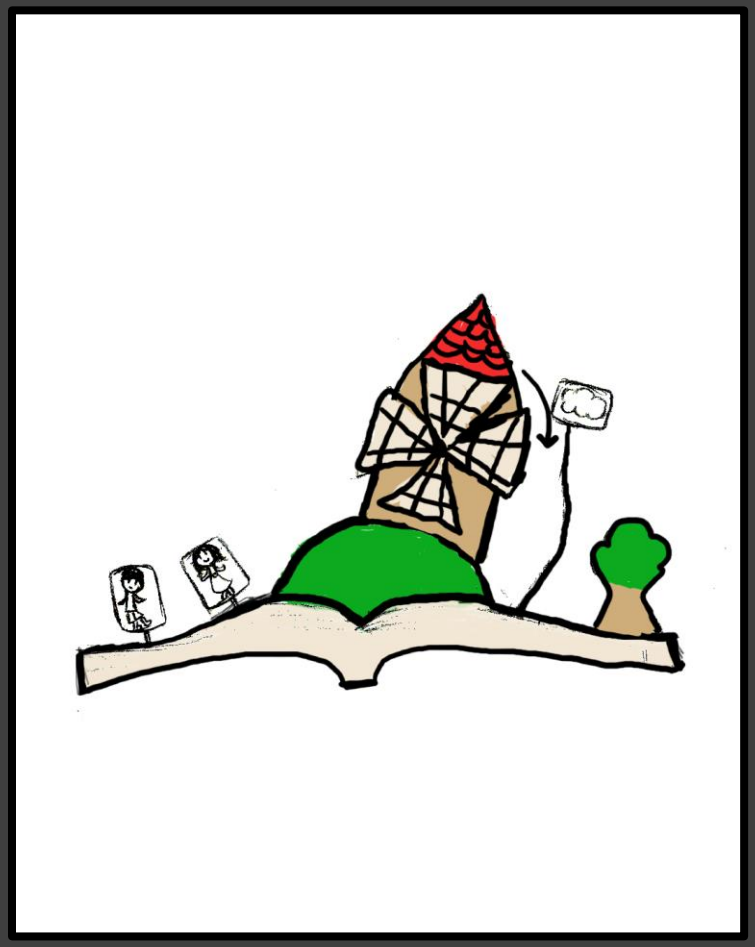
Como as coisas funcionam

- Criação do modelo
- Leitura
- Processamentos pré-renderização
- Matrizes de Transformação
- Funções do OpenGL

Criação do Modelo

Design

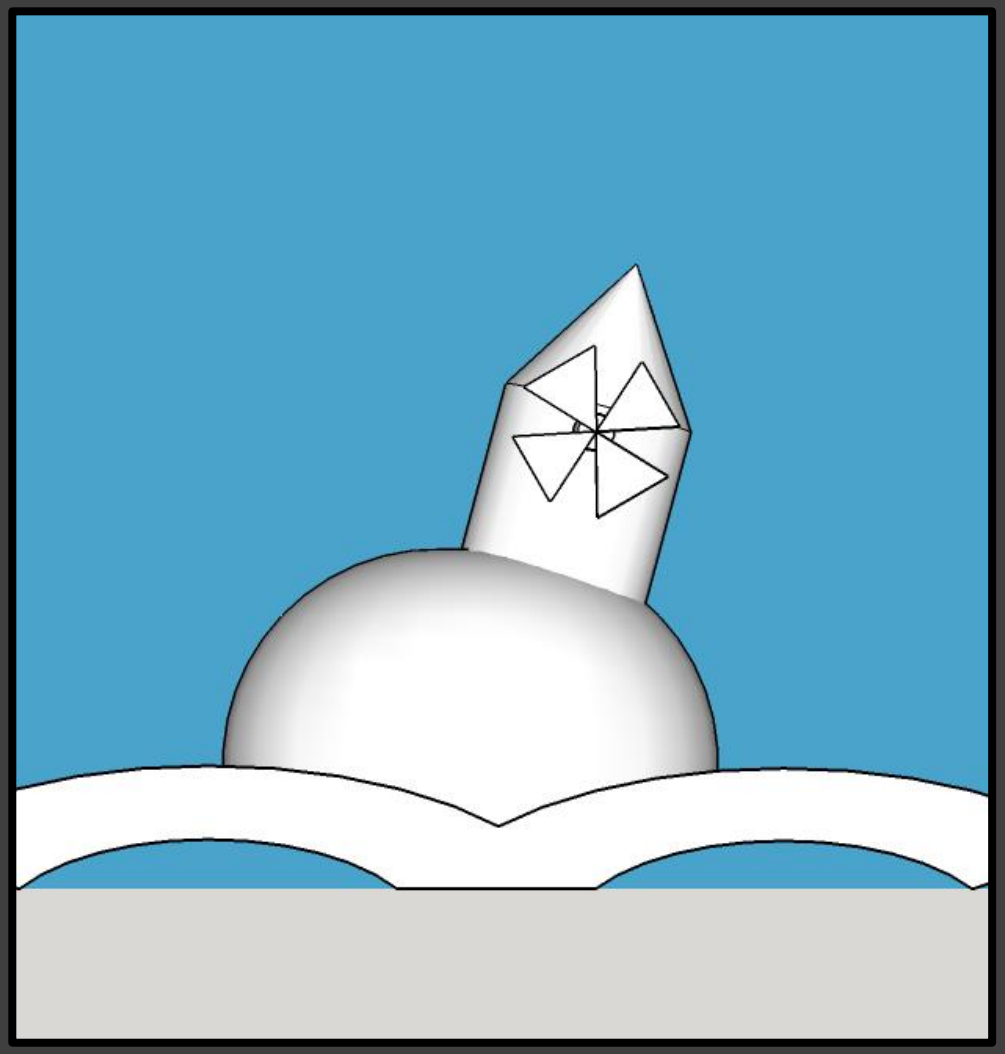
- Tema de fantasia
- Inspiração em livros de pop-up
- Mais artificial do que orgânico.



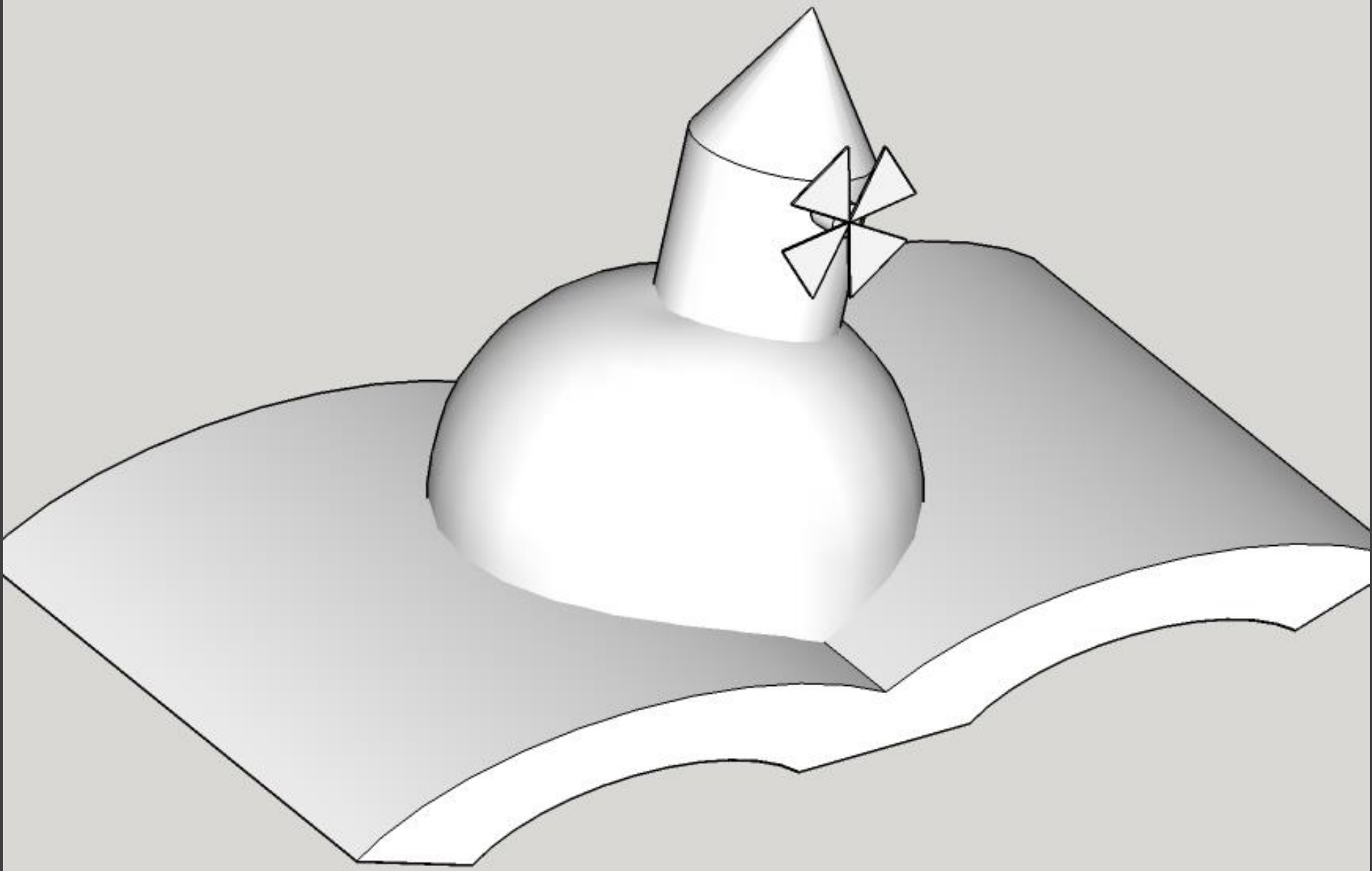
SketchUp Make

Sketchup era uma ferramenta do @Last Software (Depois comprada pela Google e atualmente pela Trimble).

Usado por arquitetos, engenharia mecânica e civil.



Modelo Atual



Leitura e Processamento

Arquivo .obj

```
# Comentário
g Nome Do Grupo

v 1193.55 106.86 1.59046e-014
# Representa um vértice (x, y, z) o w é opcional.
vt -2.53675e-013 -44.6388
# Representa coordenadas da textura (u, v) - Não usado
vn 0.249008 0.968501 -9.9527e-016
# Representa as coordenadas de uma normal - Não usado
v 1129 120.18 -792.172
vt 31.1879 -42.044
vn 0.160157 0.987092 -1.22081e-015
v 1129 120.18 1.59046e-014
vt -2.39435e-013 -42.044
f 1/1/1 2/2/2 3/3/2
# Representa um face com índices para v/vt/vn
```

Lendo um .obj

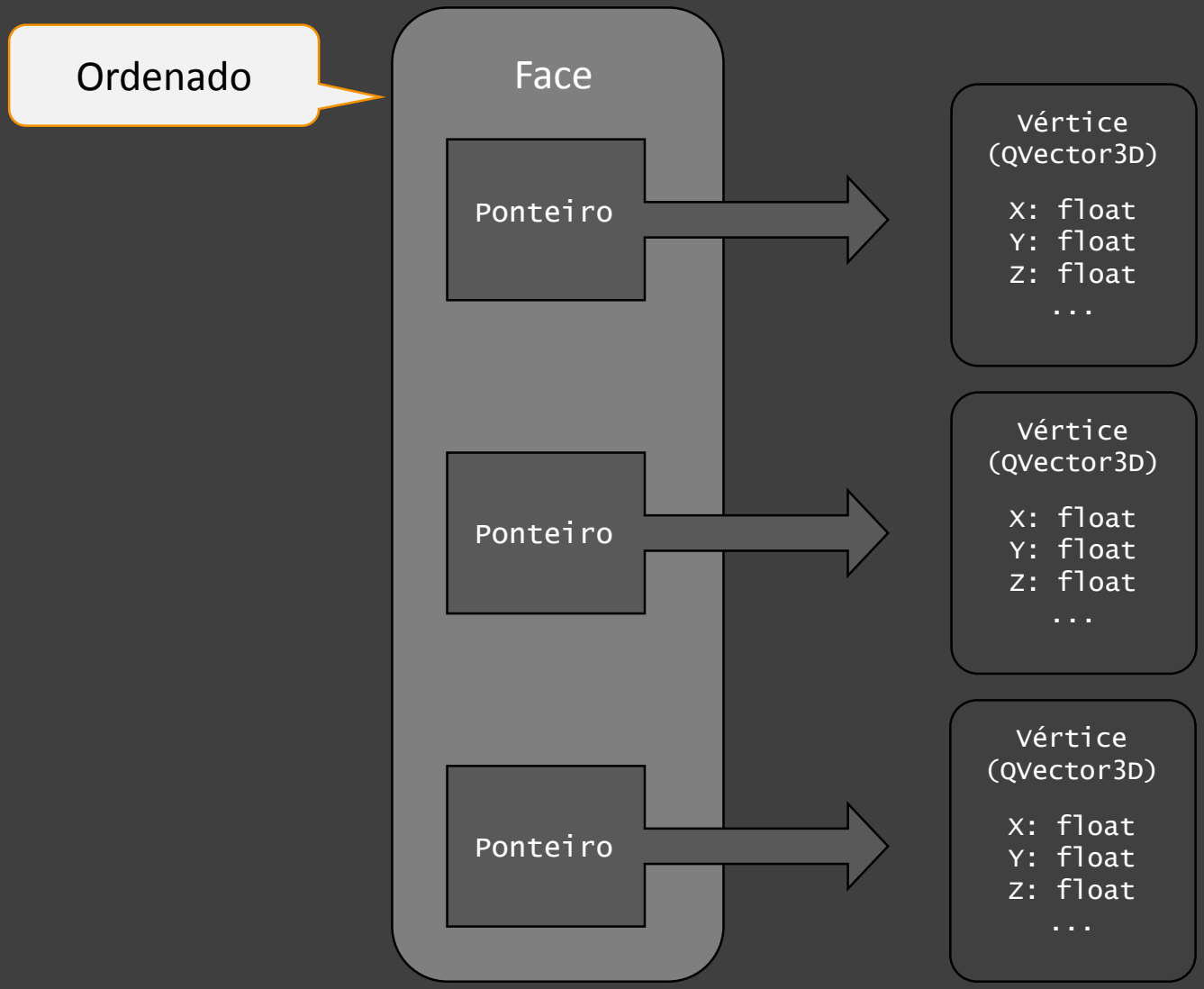
- Uma máquina de estados para cada “identificador”
- Enquanto não estamos no final do arquivo vamos ver o que fazer baseado na primeira letra do arquivo
- Mas primeiro temos que descrever como é um modelo no nossos sistema

Modelo

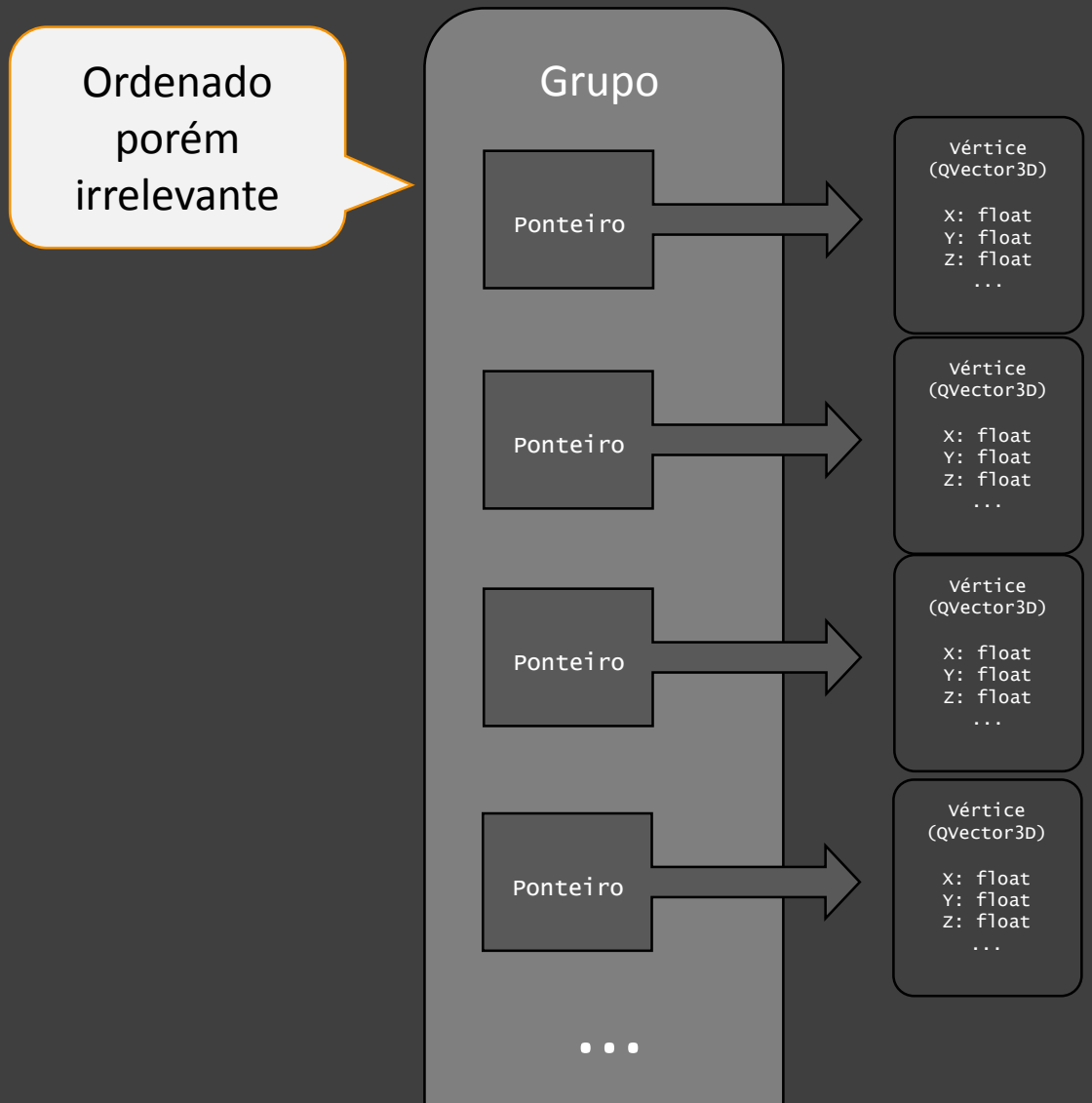
```
/** Uma face é simplesmente uma tripla de Vertices. */
typedef std::tuple<QVector3D*,QVector3D*,QVector3D*> Face;

/** Um vertice tem a mesma representação que um QVector3D. */
typedef QVector3D Vertice;

/* Um grupo de vértices é uma lista de vértices */
typedef QList<Vertice*> Grupo;

/**
 *  Esta classe representa um modelo.
 */
class Model {
private:
    QList<Vertice> vertices; // A lista de vertices normalzados do modelo.
    QList<Face> faces; // A lista de faces do modelo.
    QList<Grupo> grupos; // A lista de grupos
    Vertice pontoMedio; // Ponto médio do objeto, pode estar desatualizado
public:
    //Métodos
    void desenhar();
    void aplicarTransformacao(TransformMatrix m); // Aplica a transformação no modelo inteiro
    void aplicarTransformacao(TransformMatrix m,int i); // Aplica a transformação no i-ésimo grupo do modelo.
    // Get and Setters
    QList<Vertice> getVertices();
    QList<Face> getFaces();
    Vertice getPontoMedio(); // Retorna e atualiza o ponto médio
    Vertice getPontoMedio(int i); // Retorna o ponto médio do i-ésimo grupo
    // Construtores
    Model(QString pathname); // Construtor que recebe um arquivo OBJ.
    Model(QList<Vertice> vertices,QList<Face> faces); // Construtor que recebe uma lista de vertices e faces.
};
```

Vértices, Faces e Grupos

Vértices, Faces e Grupos



Vértices, Faces e Grupos



E agora?

- No .obj, as linhas com ‘v’ e ‘f’ seguem uma *expressão regular*.
- "([\S]*)([\s]*)([\S]*[\s]*)([\S]*[\s]*)([\S]*[\s]*)“
  - Identificador (‘f’ ou ‘v’)
  - Primeiro vértice/coordenada
  - Segundo vértice/coordenada
  - Terceiro vértice/coordenada
- No caso de faces temos que analisar os ‘/’ para só ler o índice do vértice do modelo, não de coordenadas de textura ou de normais.
- No fim, esses valores serão normalizados e também iremos calcular o ponto médio do modelo.

Arquivo .obj

```
# Comentário
g Nome Do Grupo

v 1193.55 106.86 1.59046e-014
vt -2.53675e-013 -44.6388
vn 0.249008 0.968501 -9.9527e-016
v 1129 120.18 -792.172
vt 31.1879 -42.044
vn 0.160157 0.987092 -1.22081e-015
v 1129 120.18 1.59046e-014
vt -2.39435e-013 -42.044
f 1/1/1 2/2/2 3/3/2
```

E agora?

- Quanto um ‘g’ é lido, empilhamos um novo grupo e todos os próximos vértices iram fazer parte do topo da pilha.
- Os grupos são úteis quando queremos transformar só alguns objetos.
  - Motivo principal para os grupos não saberem as faces que seus vértices pertencem.
  - É isso que faz o moinho girar.

Modelo com Grupos Explicitados