A ideia é exercitar a leitura de objetos tridimensionais a partir de arquivos e sua posterior exibição na tela. Um dos formatos mais simples apra armazenar objetos 3D é o ".TRI" que descreve um objeto a partir de uma seqüência de triângulos. Estes triângulos são formado por 9 números reais que descrevem as coordenadas X,Y e Z de cada um de seus vértices.

---

## Descrição do formato TRI

Os arquivos TRI são armazenados em formato texto. A primeira linha do arquivo contém o número de triângulos que formam o objeto
As demais linhas descrevem os triângulos na forma

x1 y1 z1 x2 y2 z2 x3 y3 z3

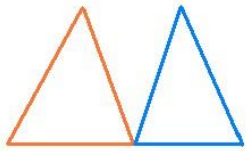
onde

x1 y1 z1 : primeiro vértice do triângulo  x2
y2 z2 : segundo vértice do triângulo  x3
y3 z3 : terceiro vértice do triângulo

Um exemplo com dois triângulos:

<table>
<tr>
<td>

```
2
0 0 10 10 0 10 5 10 10
10 0 10 20 0 10 15 10 10
```

</td>
<td>  </td>
</tr>
</table>

**OBSERVAÇÃO: Note que alguns arquivos de objetos têm um cabeçalho diferente e também a cor de cada triângulo, como no exemplo a seguir**

```
2
0 0 10 10 0 10 5 10 10 0x181818
10 0 10 20 0 10 15 10 10 0x181818
```

---

# Exercício

Copie o arquivo **Leitura-Basico3d.cpp** inclua-o em seu projeto de programas OpenGL.

Neste programa, você encontra uma sugestão de estrutura de dados para armazenar um objeto 3D.

Esta estrutura chama-se `Objeto 3D` e é composta de um vetor de triângulos (`TTriangle`). Cada um destes triângulos é formado por um vetor de 3 pontos (`TPoint`).

```
typedef struct  // Struct para armazenar um ponto
{
    float X,Y,Z;
    void Set(float x, float y, float z)
    {
         X = x;
         Y = y;
         Z = z;
    }
    void Imprime()
    {
        cout << "X: " << X << " Y: " << Y << " Z: " << Z;
    }
```

```
} TPoint;


typedef struct // Struct para armazenar um triângulo
{
    TPoint P1, P2, P3;
    void imprime()
    {
          cout << "P1 ";  P1.Imprime(); cout << endl;
          cout << "P2 ";  P2.Imprime(); cout << endl;
          cout << "P3 ";  P3.Imprime(); cout << endl;
    }
} TTriangle;

// Classe para armazenar um objeto 3D
class Objeto3D
{
    TTriangle *faces; // vetor de faces
    unsigned int nFaces; // Variavel que armazena o numero de faces do objeto
public:
    Objeto3D()
    {
           nFaces = 0;
          faces = NULL;
    }
    unsigned int getNFaces()
    {
        return nFaces;
    }
   void LeObjeto (char *Nome); // implementado fora da classe
  void ExibeObjeto(); // implementado fora da classe };
```

Complete o programa de forma que ele seja capaz de ler e exibir o arquivo **Vaca.tri**.

No programa fonte **Leitura-Basico.cpp** você encontra as indicações de onde alterar o programa. Procure pelos métodos

**`void Objeto3D::LeObjeto (char *Nome)`**

e

**`void Objeto3D::ExibeObjeto ()`**

Observe também na função main como é feita a carga de um objeto.

```
 // aloca memória para 5 objetos
 MundoVirtual = new Objeto3D[5];
// carrega o objeto 0
MundoVirtual[0].LeObjeto (Nome);
```

O resultado final de seu exercício deve ser semelhante a esta imagem:


Você pode obter outros arquivos .TRI nos arquivos:

**TRIs.zip**

**Avioes.zip**

Tente carregar alguns destes arquivos. Note que alguns deles têm pequenas diferenças no cabeçalho, que pode ser facilmente editado.
Monte um cenário que tenha uma:

- Uma rua;
- Duas instâncias da casa;
- Algumas árvores;
- Um carro;
- Um cachorro;
- Um avião no céu.

---

Novos Objetos
Você também pode converter objetos 3D de outros formatos para o TRI, a partir de programas como

FBX -> OBJ: http://www.greentoken.de/onlineconv/
OBJ -> TRI: http://3doc.i3dconverter.com/index.html

Para obter objetos 3D, acesse sites como
https://free3d.com/3d-models/obj
https://www.turbosquid.com/Search/3D-Models/free

---

## Iluminação

Para que a iluminação seja corretamente aplicada aos objetos, você precisará calcular o **vetor normal** de cada face e usá-lo antes de desenhar cada uma delas, conforme este exemplo:

```
glBegin(GL_TRIANGLES);
      glNormal3f(Nx, Ny, Nz); // comando
      que define a normal de uma face
      glVertex3f(.........);
      glVertex3f(.........);
      glVertex3f(.........);

glEnd();
```

Além disto, é preciso usar vetores unitários. Para calcular as normais das faces, você pode usar as rotinas abaixo.

```
// Rotina que faz um produto vetorial

void ProdVetorial(TPoint v1, TPoint v2, TPoint &vresult)
{

      vresult.X = v1.Y * v2.Z - (v1.Z * v2.Y);
      vresult.Y = v1.Z * v2.X - (v1.X * v2.Z);
      vresult.Z = v1.X * v2.Y - (v1.Y * v2.X);

}

// Esta rotina tem como funcao calcular um vetor unitario
void VetUnitario(TPoint &vet)
    {
        float modulo;
```

```
    modulo = sqrt (vet.X * vet.X + vet.Y * vet.Y + vet.Z * vet.Z);

    if (modulo == 0.0) return;

      vet.X /= modulo;
      vet.Y /= modulo;
      vet.Z /= modulo;
  }
```

---

## Posicionamento do Objeto no Cenário

Com o objetivo de posicionar o objeto mais facilmente no espaço, uma alternativa é, durante a leitura de um objeto, analisar as coordenadas dos vértices e determinar os limites em X, Y e Z, encontrando os maiores e menores valores em cada eixo. Com isto, cria-se um **envelope 3D** para o objeto.

A partir disto, tem-se duas alternativas:

- Após a leitura, mas ainda antes de iniciar o rendering, altere as coordenadas dos vértices do objeto de forma que o novo centro dele seja o (0,0,0). Isto vai facilitar o posicionamento do objeto durante o rendering
- A cada frame, utilize o centro do objeto para descolcar o objeto para o (0,0,0) e depois aplique um novo translate para colocar o objeto na posição desejada.

A figura a seguir exemplifica este deslocamento do centro do objeto.

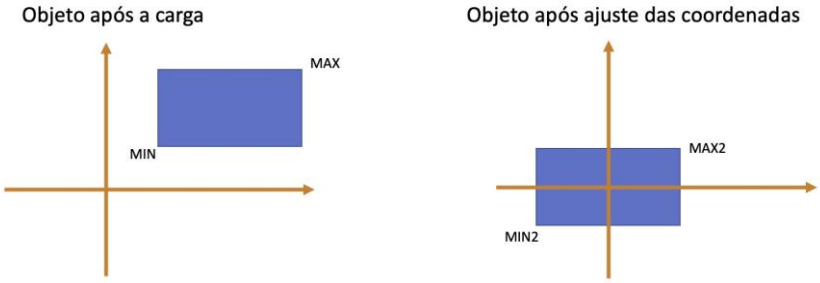



**Figura - Mudança do Centro do Objeto**

Para ajustar o tamanho do objeto de forma que este fique com um tamanho adequado ao cenário, pode-se usar as mesmas alternativasas anteriores:

- Após a leitura, mas ainda antes de iniciar o rendering, calcular uma escala com base no envelope do objeto e alterar as coordenadas de maneira que o objeto tenha, na sua maior dimensão o tamanho 1.0. Assim, no momento da rendezização é possível como saber qual o glScale a ser usado para posicionar o objeto;
- A cada frame, utilizar o envelope do objeto para redimensioná-lo com glScale e depois aplicar um novo scale para colocar o objeto na posição desejada.

Para o momento do rendering do objeto, fica também a sugestão de primeiro posicionar o objeto e spmente depois realizar a escala. Do contrário, o posicionamento se torma mais difícil pois o OpenGL muda o sistema de coordenadas durante a operação de escala.

---

**Exemplo pronto** (para funcionar deve ser colocado na mesma pasta onde está o arquivo Vaca.tri).