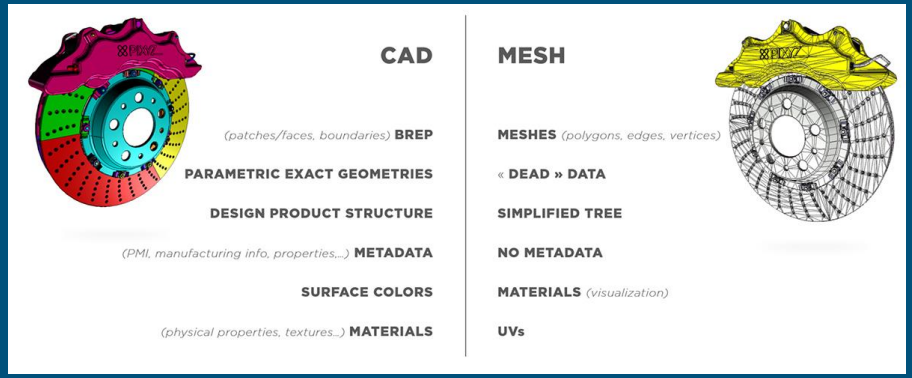
Modelagem Geométrica: Half Edge 3D e Curvas de Bézier

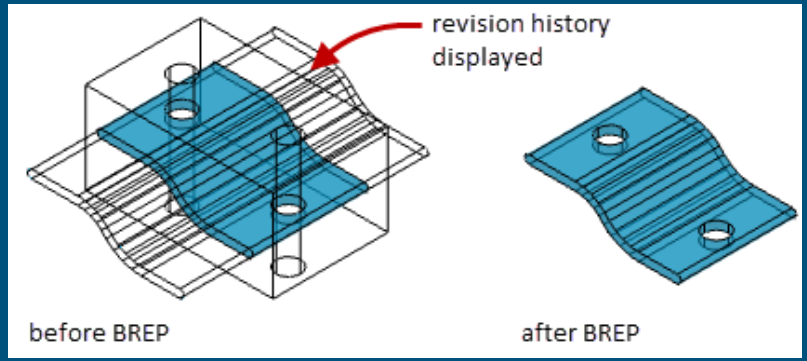
Matheus Santos Araújo

UFC

Half Edge - Motivação



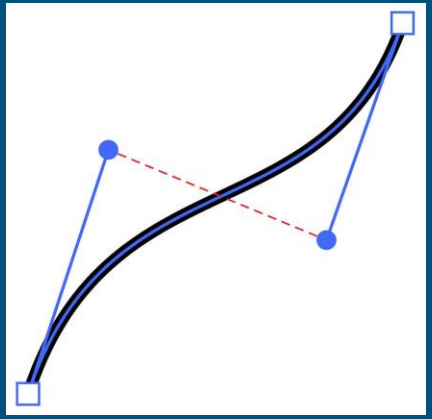
- Método para representar uma forma 3D definindo os limites de seu volume
- Sólido é representado como uma coleção de elementos que definem o limite entre pontos internos e externos
- Além das operações booleanas, o B-rep possui extrusão (ou varredura), chanfro , mesclagem, desenho, descascamento, ajuste...

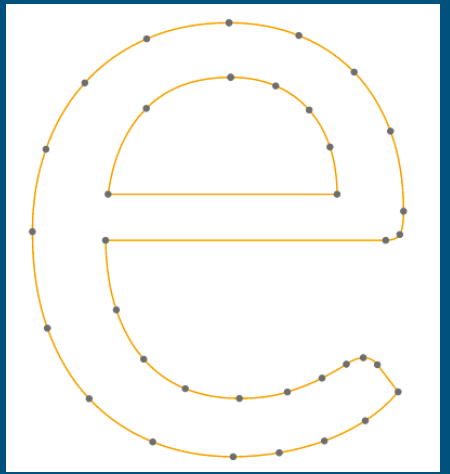


UFC

Bézier - Motivação

- Bézier é uma curva paramétrica usada em computação gráfica e campos relacionados
- Conjunto de "pontos de controle" define uma curva suave e contínua por meio de uma fórmula
- Curvas de Bézier também são usadas no domínio do tempo, particularmente em animação , design de interface do usuário...





UFC

Half Edge 3D

- Estratégia: criar estruturas de dados necessárias, implementar os operadores de Euler e modelar as primitivas
- Desenhar vértices, faces e arestas no openGL

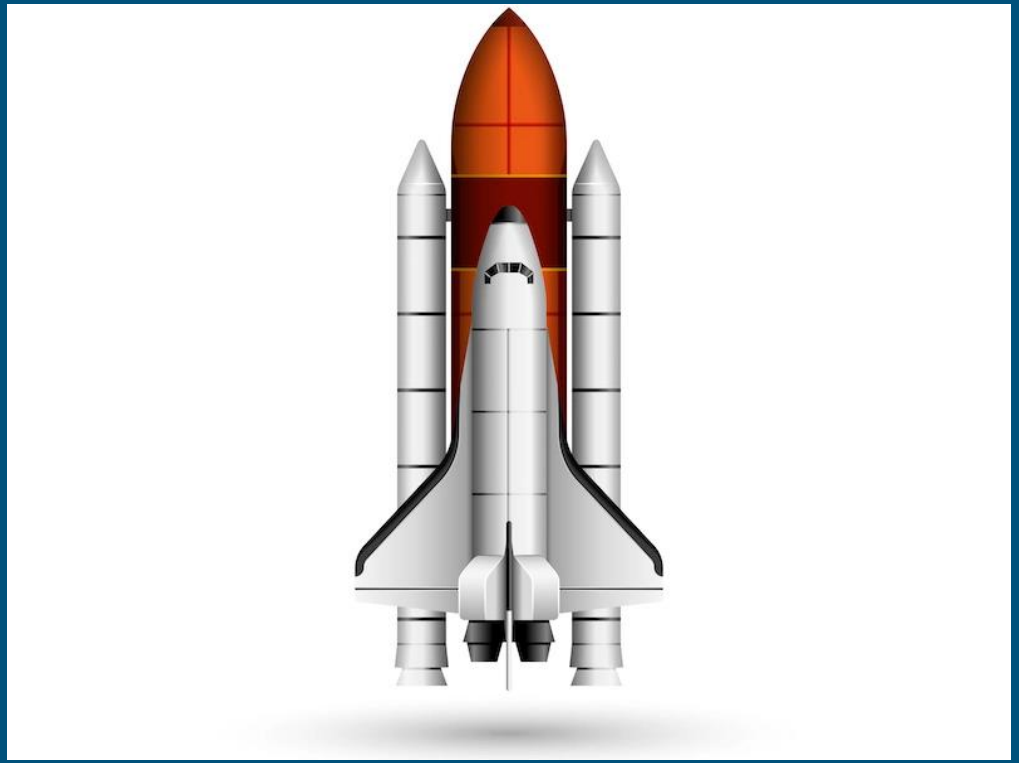
UFC

Bézier Curve

- Estratégia: criar uma estrutura de dados Bézier, implementar a interpolação e desenhar a curva usando openGL
- Além da curva interpolada desenhar os vértices e as retas entre pontos de controle

UFC

Tema - Simplificado



UFC

Implementação - Half Edge

- Vectorops: point, vec3, norm, unit, cross, dot, Point3D, Quaternion, sum, match, esc, equal, angle...

Data Structs:

- Objeto Vertex: x, y, z, id, getId, move setPosition
- Objeto Edge: id, Half Edge 1, Half Edge 2
- Objeto Half Edge: vertice origem, loop edge, Half Edge ant, Half Edge prox
- Objeto Loop: id, Vec3 normal, Half Edge he, Face face, setNormal
- Objeto Face: id, List loops, Loop outter, Mesh solid
- Objeto Mesh: numFaces, numVertices, numLoops, int numEdges, List vertice faces edges, tx, ty, tz, sx, sy, sz, Quat orientation, addVertex, addFace, addEdge

UFC

Implementação - Half Edge

Data Structs:

Objeto BREP - EulerOps:

- List meshes, int qtd, clear, paint, getFace, getHEd, getLoop, getVertex, mvfs, mev, mef, varr, rvarr

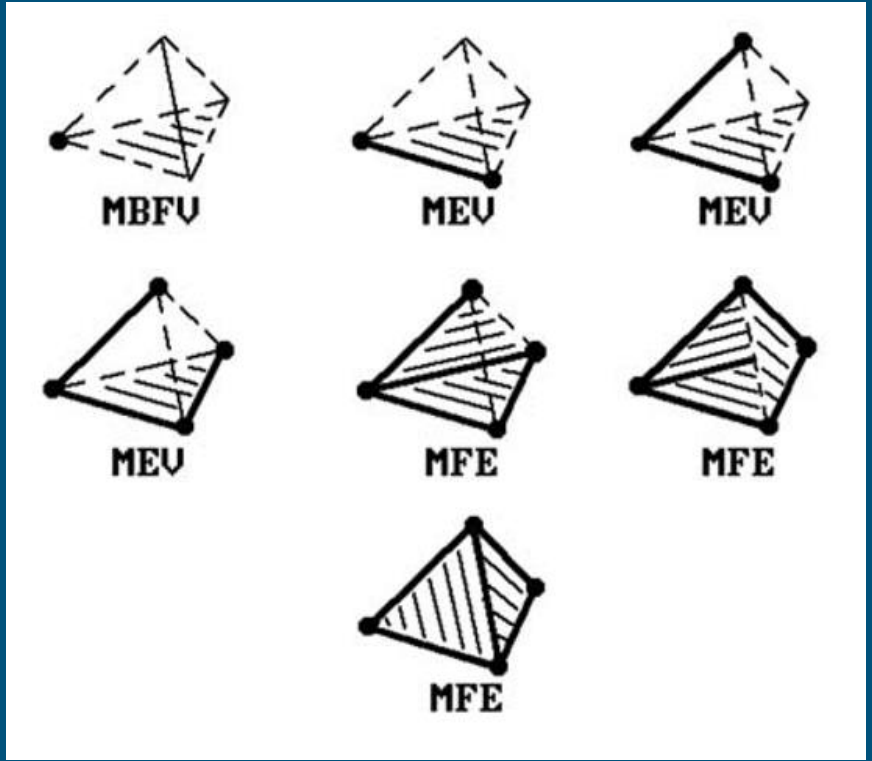
Objeto BREP - Primitive:

- Cube, Cylinder, Sphere, Tema

UFC

Implementação - Half Edge

EulerOps:



UFC

Implementação - Bézier

Data Structs:

- Objeto Bézier: List points, int order, getPoints, calc, paint

Implementação - EulerOps

- void mvfs(float x, float y, float z);

Inicializa objetos, 1 vértice e 1 aresta

- bool mev(HalfEdge* he1, HalfEdge* he2, int idSolid, int idFace, int idVertex1, float x, float y, float z);

Cria novo vértice e nova aresta

- void varr(int idSolid, int idFace, float dx, float dy, float dz);

Varre o sólido e o desloca (extrude) em uma proporção

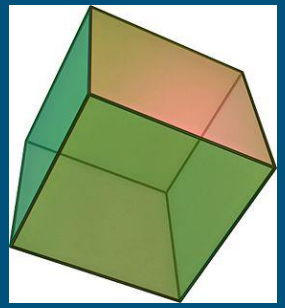
- void rvarr(int idSolid, int idFace, int disc, float angle);

Varre a sólido e o desloca (extrude) em uma proporção radial

UFC

Implementação - Cubo

- Entrada: a entrada do cubo consiste nos parâmetros de limites e tamanho



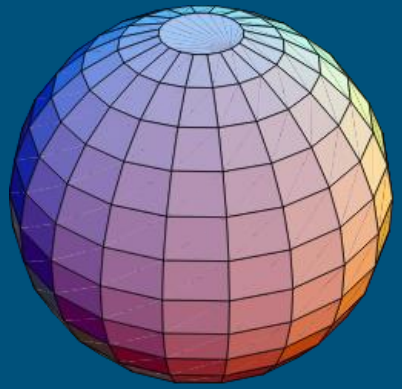
=

```
mvfs(minX, minY, minZ + size);
mev(NULL, NULL, qtd - 1, 0, 0, minX + size, minY, minZ + size);
mev(NULL, NULL, qtd - 1, 0, 1, minX + size, minY + size, minZ + size);
mev(NULL, NULL, qtd - 1, 0, 2, minX, minY + size, minZ + size);
mef(NULL, NULL, qtd - 1, 0, 3, 0);
varr(qtd - 1, 0, 0, 0, -size);
```

UFC

Implementação - Esfera

- Entrada: a entrada do cubo consiste nos parâmetros de limites e tamanho



=

```
float step = M_PI / disc;
mvfs(-1.0 * radius, 0, 0);
for (int i = 0, idNum = 0; i < disc; i++, idNum++) {
float angle = M_PI - step * (i + 1);
mev(NULL, NULL, qtd - 1, 0, idNum, radius * cos(angle),
radius * sin(angle), 0.0); }
rvarr(qtd - 1, 0, 2 * disc, 2 * M_PI);
```

UFC

Implementação - Bézier

$$Bézier(n,t) = \sum_{i=0}^{n} \underbrace{\binom{n}{i}}_{\text{binomial term}} \cdot \underbrace{(1-t)^{n-i} \cdot t^{i}}_{\text{polynomial term}}$$

- É basicamente apenas uma soma de "cada combinação de a e b ", substituindo progressivamente a 's por b 's após cada sinal de +
- Interpolação por coeficientes binomiais:

```
 std::vector<Point3D> points = points;
for (int i = 0; i <= order; i++) {
   double x = (factorial(order) / (factorial(i) * factorial(order - i))) * pow(u, i) * pow(1.0 - u, order - i);

   points[i].esc(x);
}

for (int x = 1; x <= order; x++) { points[x].sum(points[x - 1]); }

return points[order];
```

UFC

Implementação - Bézier

- Entrada: a entrada da curva consiste nos parâmetros de pontos desejados

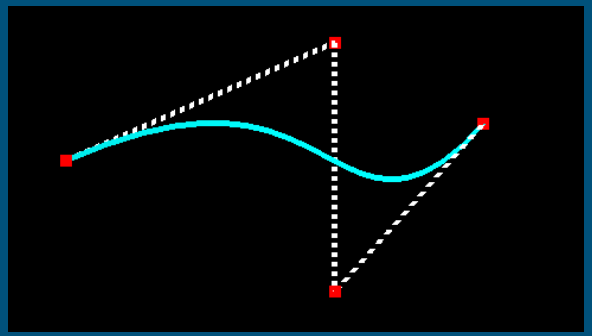
  - Pontos de Controle:

Point3D p1 = Point3D(5.0, 0.0, 2.0);
Point3D p4 = Point3D(-10, 0.0, 0.0);

  - Pontos Inseridos:

Bezier curve(Point3D {0, -5, 0 }, Point3D { 0, 5, 0 });

=



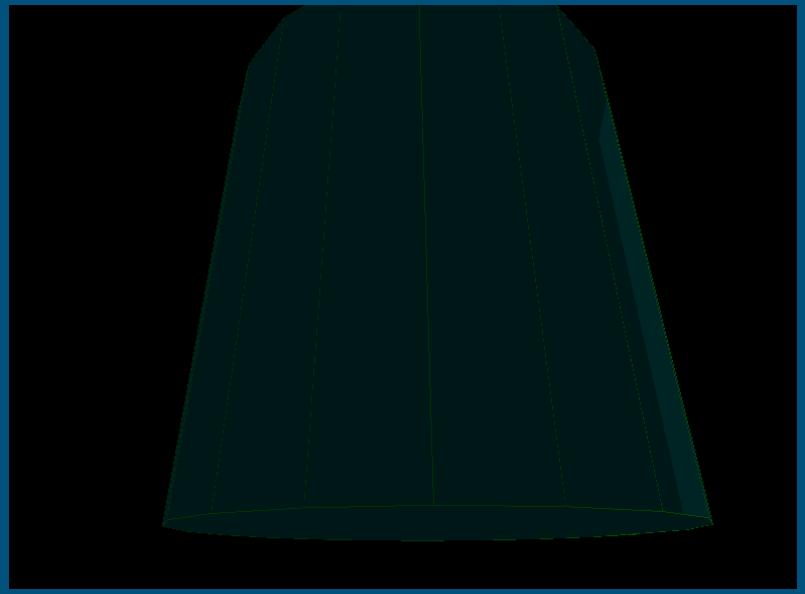
UFC

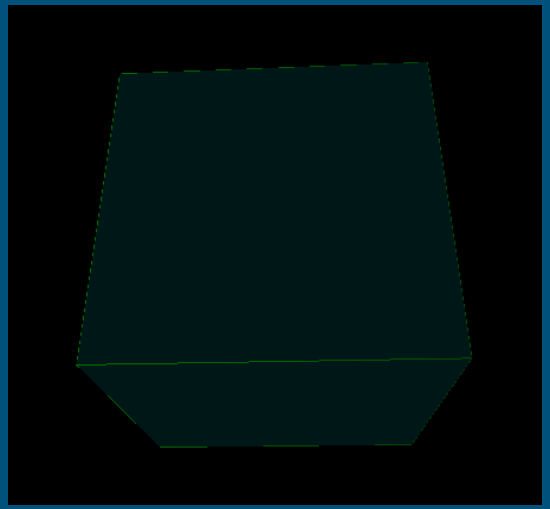
Execução

- OpenGL para visualização dos resultados
- Utilitário keyboard
- Tecla E: modela esfera
- Tecla B: modela cubo
- Tecla C: modela cilindro
- Tecla T: modela Tema
- Tecla I: translada para um lado
- Tecla K: translada pra outro lado
- Tecla U: aumenta escala
- Tecla D: diminui escala
- Tecla Y: modela curva de bézier

UFC

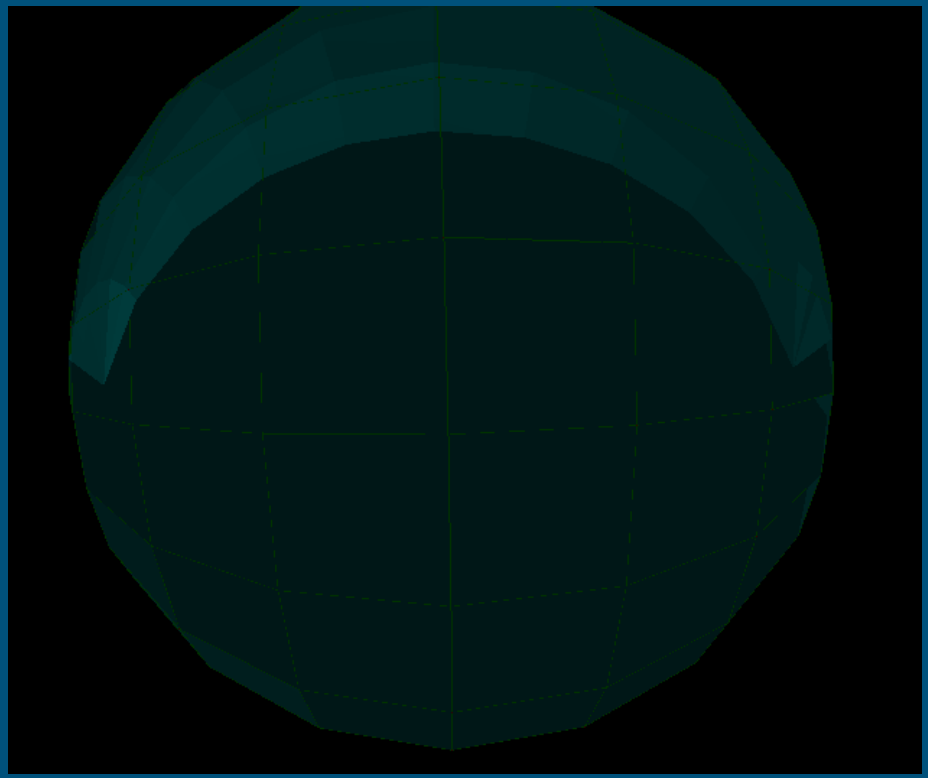
Modelagem de primitivas





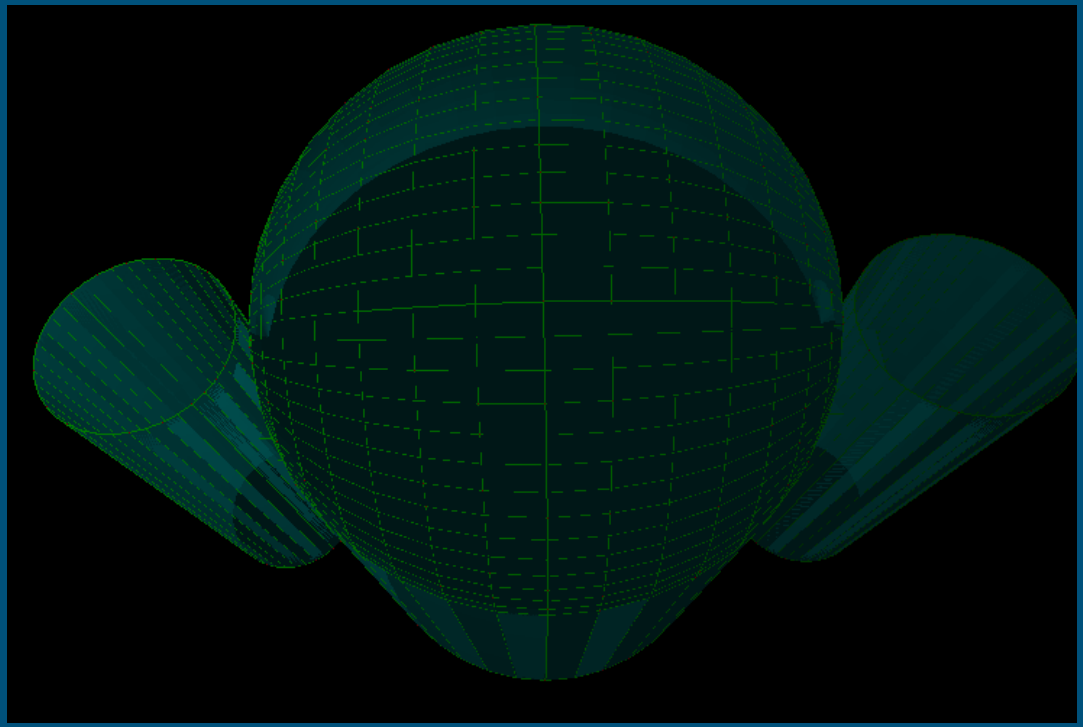
UFC

Modelagem de primitivas



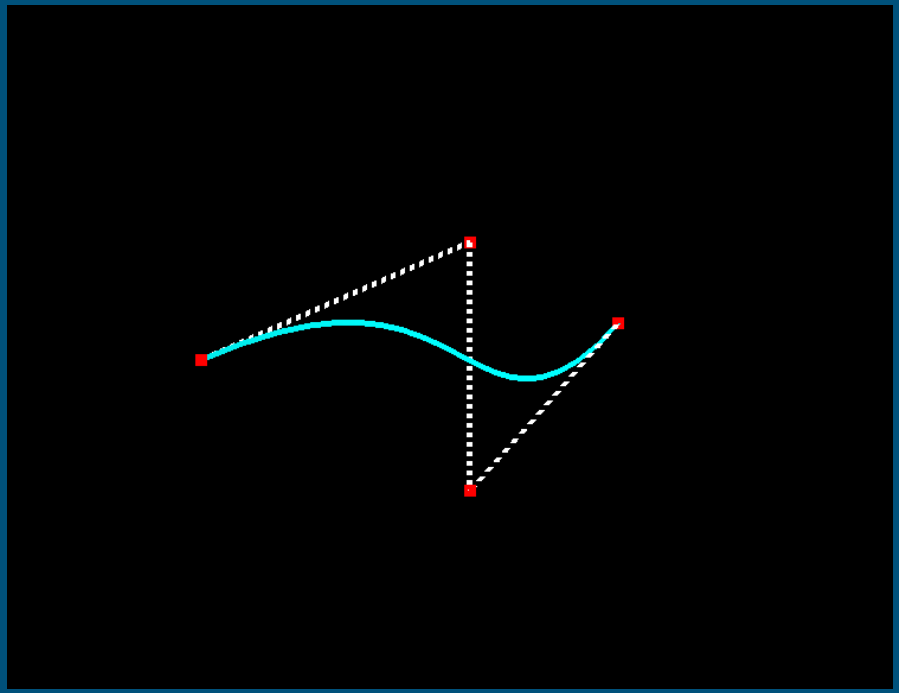
UFC

Modelagem do tema - simplificado



UFC

Modelagem de curvas de bézier



UFC

Conclusão

- Todas as estruturas de dados foram implementadas
- As Operações de Euler: mvfs, mev e mef foram implementadas além da operação extrusão e extrusão radial
- Algoritmo funciona para cubo e formas do tema: cubo, cilindro, esfera em um tempo de execução curto
- Curva de Bézier também funciona em tempo curto
- Não foi implementada a primitiva cone

UFC