# IMPLEMENTAÇÃO DE DEFORMAÇÃO DE MALHAS NO HUMANÓIDE V-ART

Acadêmico:
Karlyson Schubert Vargas
Orientador:
Dalton Solano dos Reis

# Roteiro

- Introdução
- Objetivos
- Fundamentação Teórica
- Especificação
- Resultados
- Conclusões
- Extensões

# Introdução

- Humanóides
  - Articulações
- Animação
- V-Art
  - Necessidade do algoritmo

# Objetivos

- Definir um algoritmo de deformação de malhas
- Implementar o algoritmo selecionado no *framework* V-Art
- Associar pontos de controle da deformação de malha nas articulações

# Algoritmos

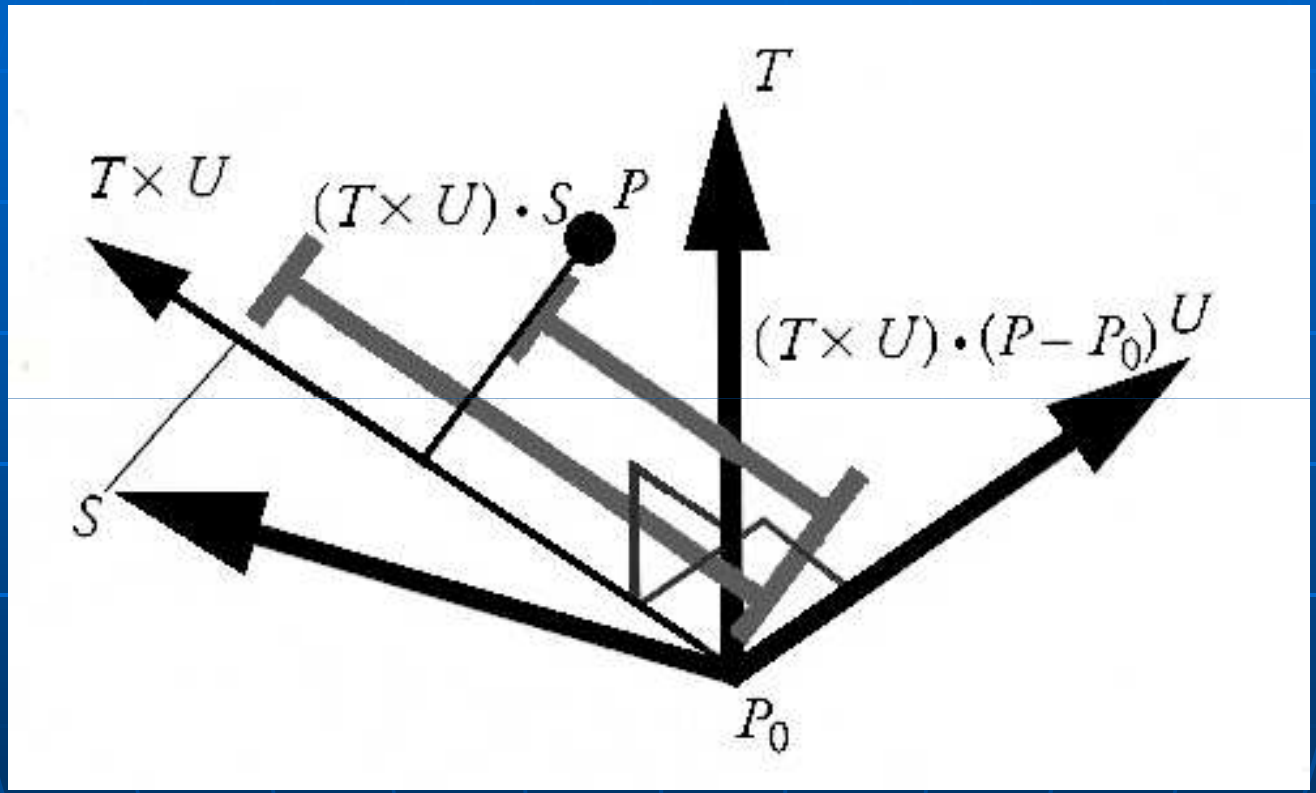
- Algoritmo de deformação de representações poligonais
- Algoritmo de deformação de representações paramétricas
  - Free Form Deformation (FFD)
  - Pontos representantes

# Algoritmos - FFD

| | |
|---|---|
| Vetor max = maior(Grade);<br>Vetor min = menor(Grade);<br>Vetor X0 = Vetor(min.x, min.y, min.z);<br>Vetor S = Vetor(max.x, min.y, min.z) – X0;<br>Vetor T = Vetor(min.x, max.y, min.z) – X0;<br>Vetor U = Vetor(max.x, min.y, max.z) – X0; | Criação vetores ortogonais |

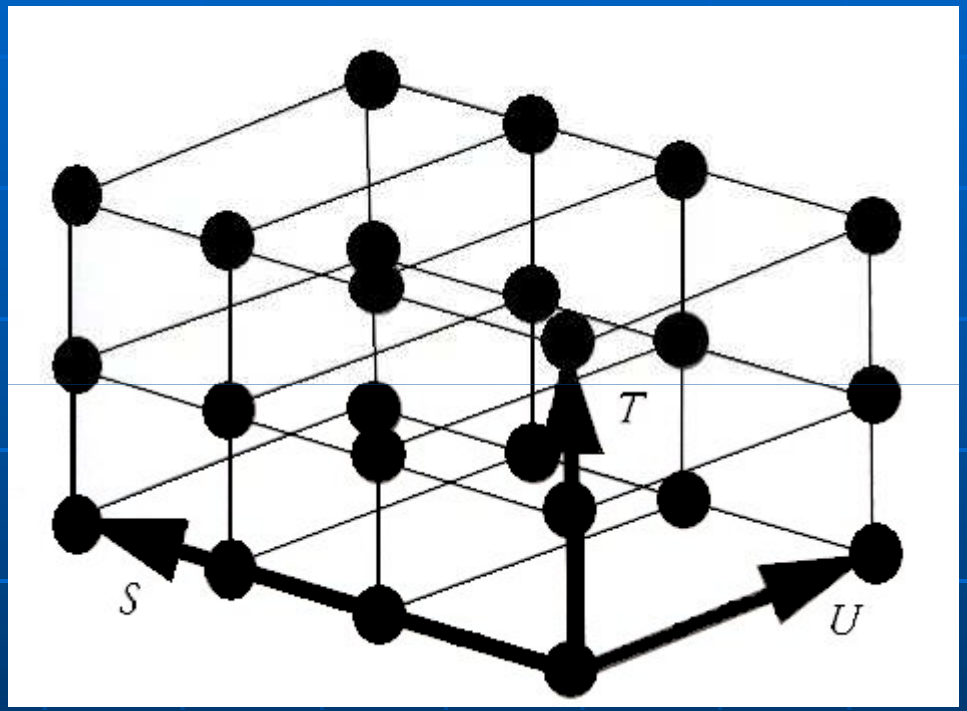
| | |
|---|---|
| s = (T x U) . (P – P0) / ((T x U) . S)<br>t = (U x S) . (P – P0) / ((T x U) . S)<br>u = (U x S) . (P – P0) / ((T x U) . S) | Interpoladores trilineares |

# Algoritmos - FFD




Ponto registrado no sistema de coordenadas

# Algoritmos - FFD




Pontos de controle nos eixos S, T e U

$$P_{ijk} = P0 + \frac{i}{l}.S + \frac{j}{m}.T + \frac{k}{n}.U$$

Localização dos pontos de controle

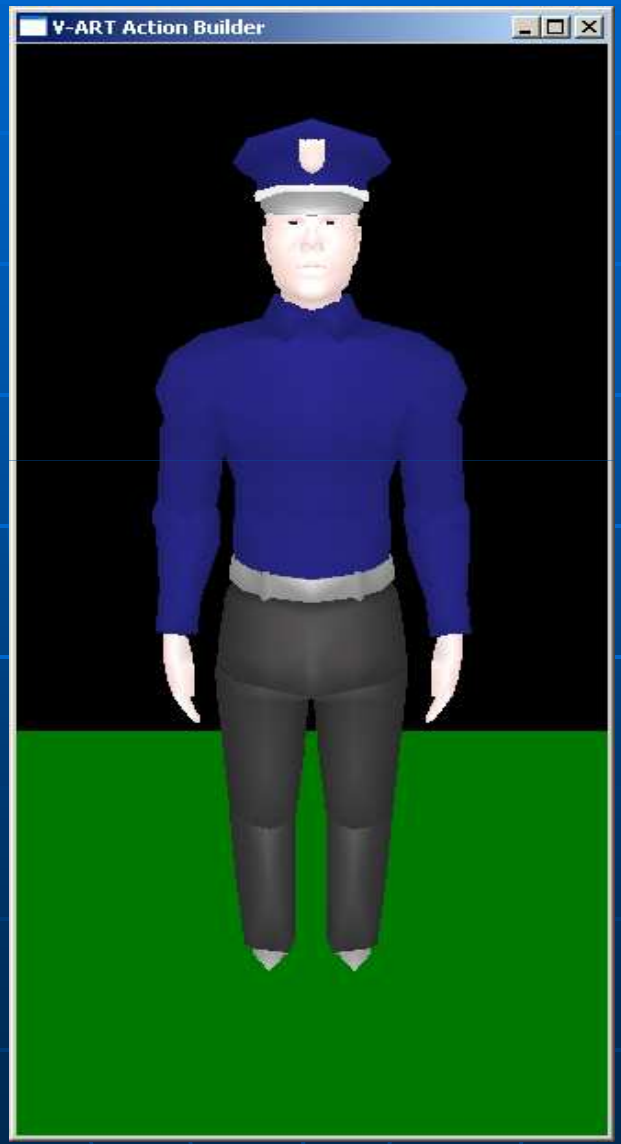
# Algoritmos - FFD

$$P(s,\ t,\ u) = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-i} s^{i}$$

$$\cdot \left( \sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^{j} \cdot \left( \sum_{k=0}^{n} \binom{n}{k} (1-u)^{n-k} u^{k} P_{ijk} \right) \right)$$

Interpolação de Bézier

# Humanóide

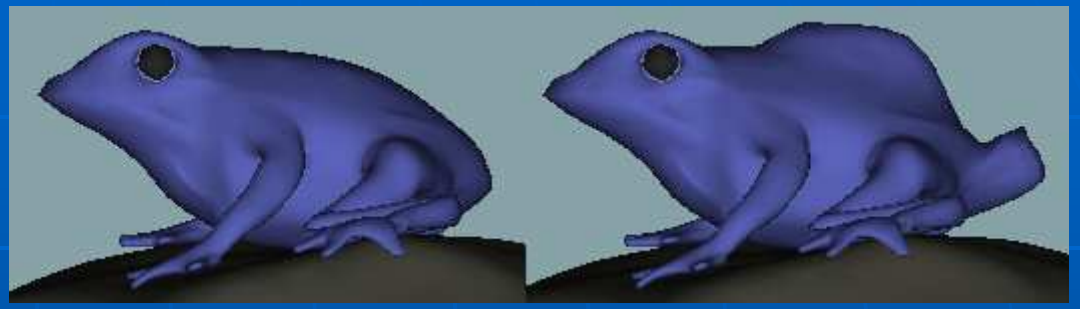



- Corpo Articulado
- Juntas
- Problemas

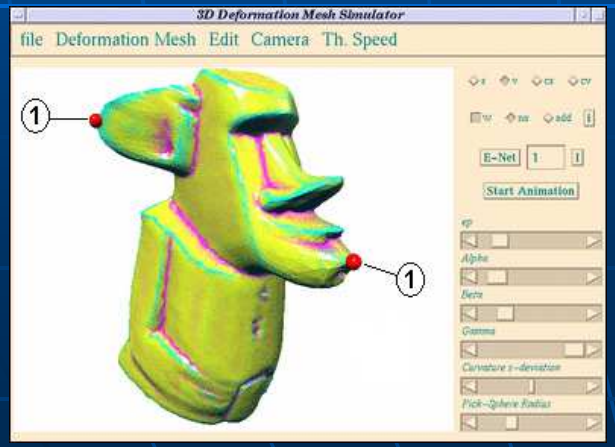
# V-Art

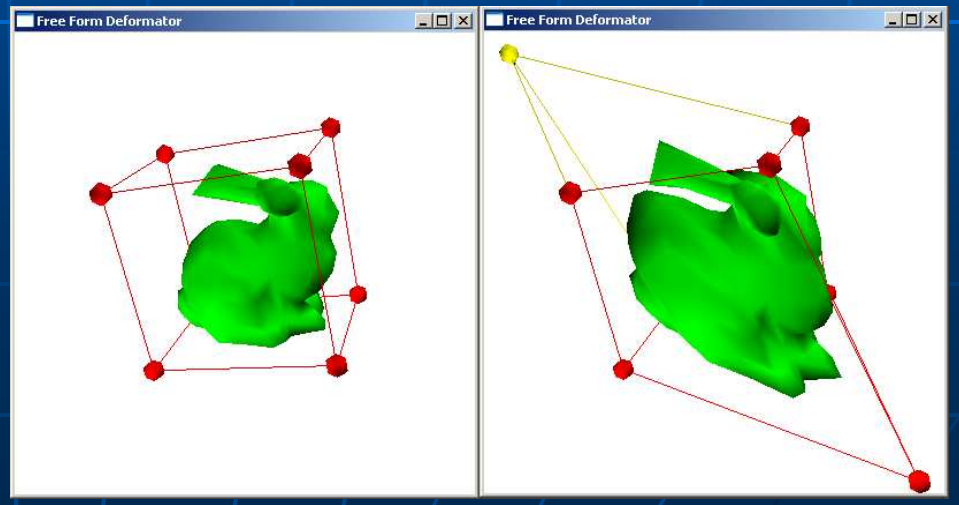
- O *framework* V-Art foi desenvolvido com o intuito de criar ambientes virtuais 3D, especialmente os que contenham humanóides
- Multiplataforma e independente de API gráfica
- Arquivos XML

# Trabalhos Correlatos



- Art of Illusion
- Free Form Deformation
- 3D Mesh Simulator

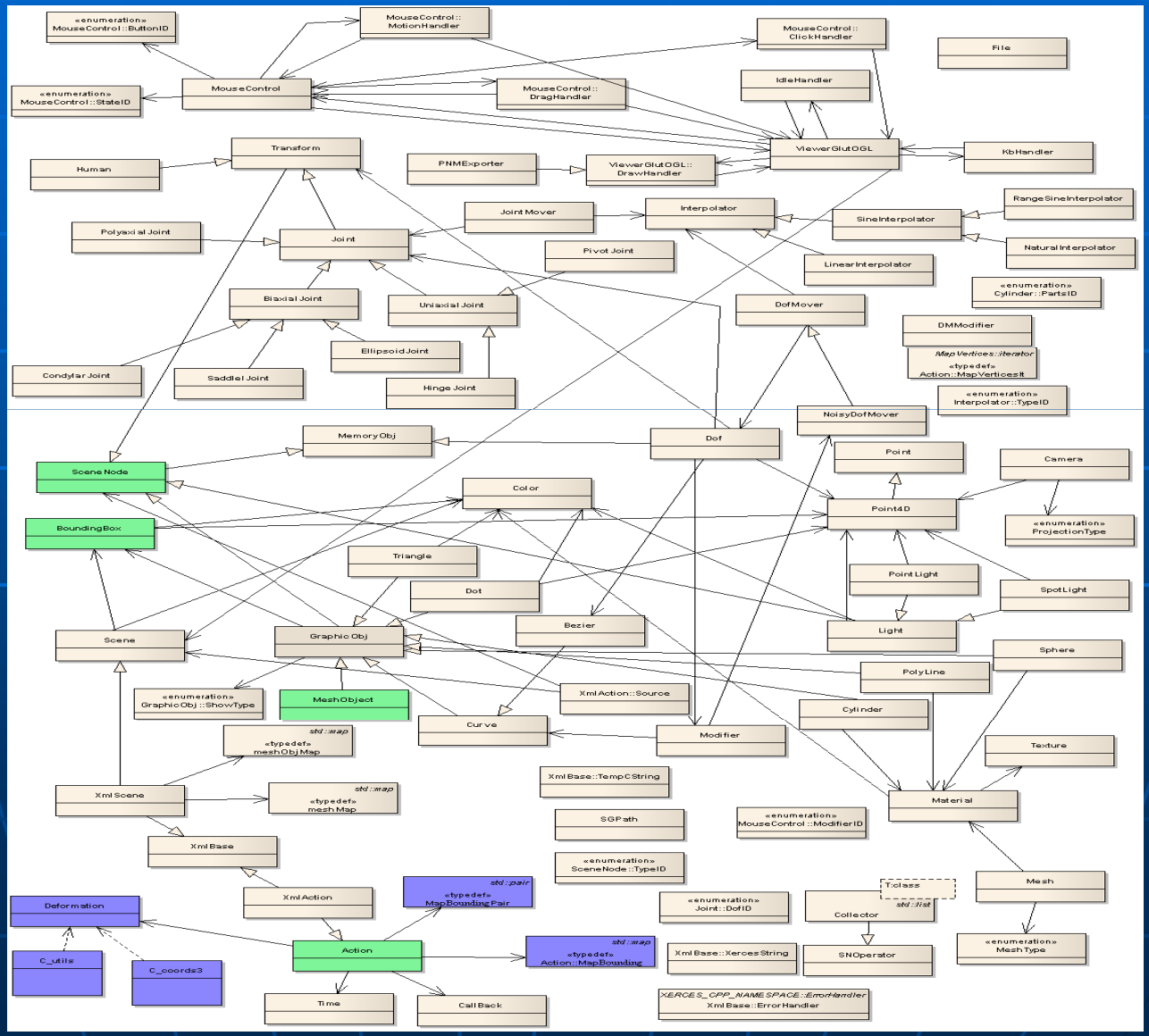


# Requisitos

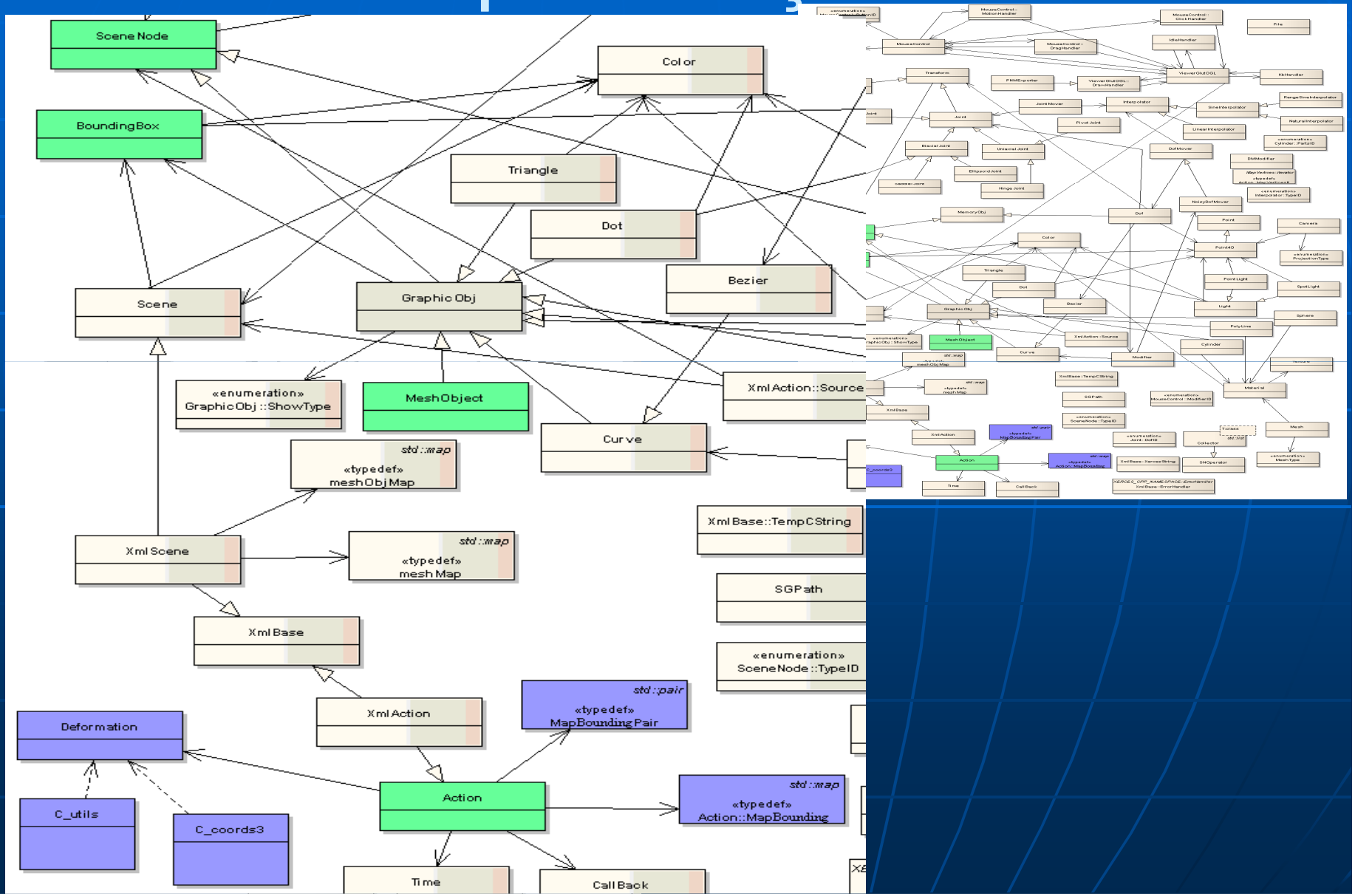
- Adicionar pontos de controle no humanóide
- Mostrar os pontos dos vetores de uma malha para que o usuário deforme manualmente a mesma
- Realizar a deformação da malha no humanóide V-Art
- Implementar rotinas que permitam que a malha seja deformada sem a interação do usuário após uma movimentação da articulação
- Implementar utilizando o ambiente Microsoft Visual C++ 2005

# Especificação




- SVN
- 69 Classes
- Pesquisadores

# Especificação



# Implementação

- Microsoft Visual Studio 2005
  - STL
- Blender
- DoxyGen
- Enterprise Architect

# Implementação

```
1   for(int w = 0; w < vertices.size() ; w += 3){
2       C_coords3 X;
3       X.x = vertices[w];
4       X.y = vertices[w+1];
5       X.z = vertices[w+2];
6       s = stu[w];
7       t = stu[w+1];
8       u = stu[w+2];
9       Xnew.setCoords(0,0,0);
10      for (int i = 0; i <= l; i++) {
11          Xnew2.setCoords(0,0,0);
12          for (int j = 0; j <= m; j++) {
13              Xnew3.setCoords(0,0,0);
14              for (int k = 0; k <= n; k++) {
15                  Xnew3 += C_utils::combination(n, k) * pow((1 - u), (n - k)) * pow(u, k) *
16                                  (X0+(i/(double)l)*S+(j/(double)m)*T+(k/(double)n)*U);
17              }
18              Xnew2 += C_utils::combination(m, j) *  pow((1 - t),(m - j)) *  pow(t,j) *
19                              Xnew3;
20          }
21          Xnew += C_utils::combination(l, i) *  pow((1 - s),(l - i)) *  pow(s,i) *   Xnew2;
22      }
23      verticesDef.push_back(Xnew.x);
24      verticesDef.push_back(Xnew.y);
25      verticesDef.push_back(Xnew.z);
26  }
27  return verticesDef;
```

Execução da interpolação de Bézier

# Implementação

```
3       X.x = vertices[w];
4       X.y = vertices[w+1];
5       X.z = vertices[w+2];
6       s = stu[w];
7       t = stu[w+1];
8       u = stu[w+2];
```

interpolação de Bézier

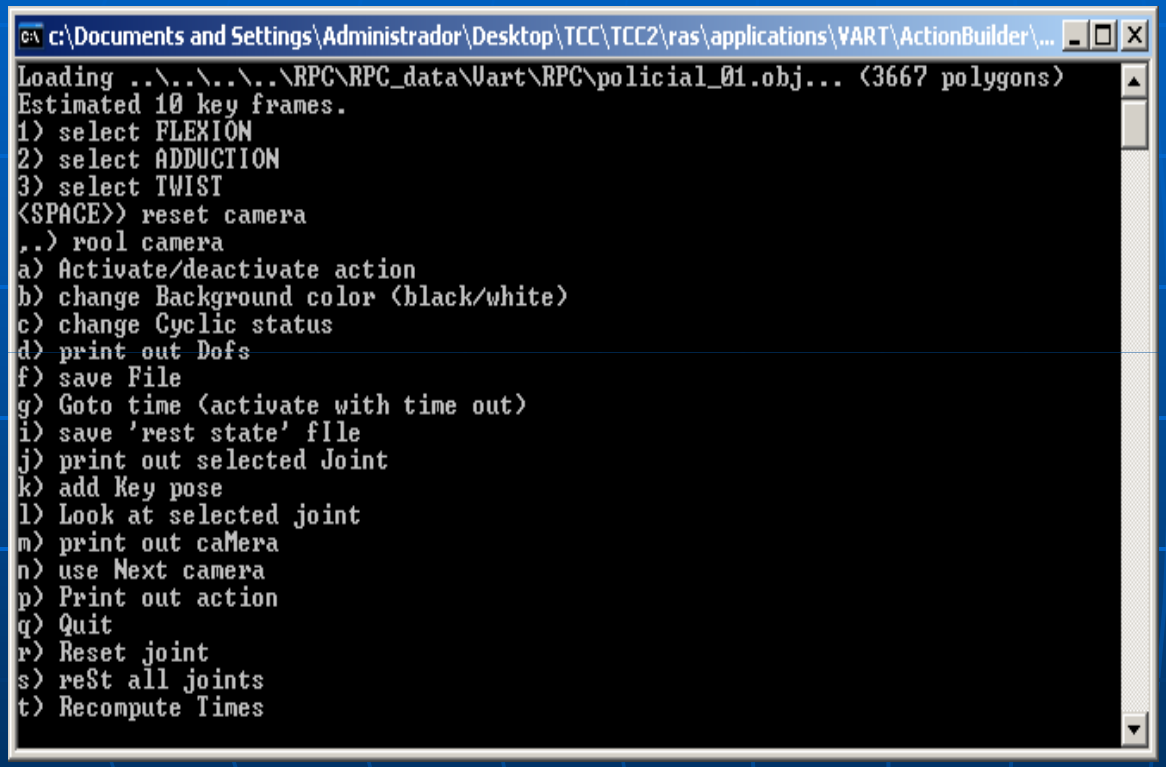
```
3    X.x = vertices[w];
4    X.y = vertices[w+1];
5    X.z = vertices[w+2];
6    s = stu[w];
7    t = stu[w+1];
8    u = stu[w+2];
```

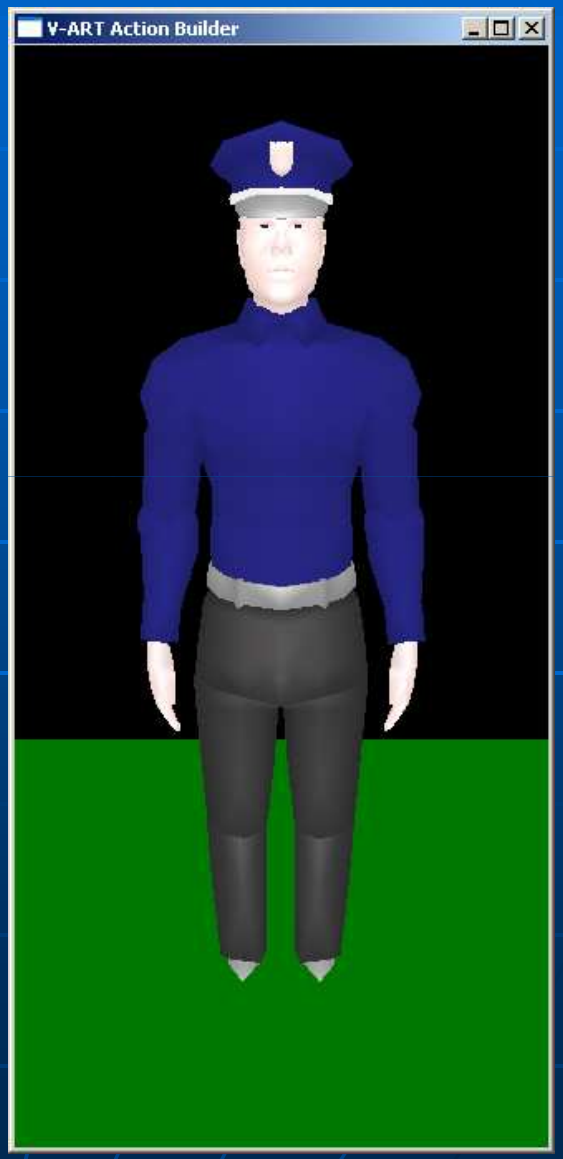
# Implementação

```
15  Xnew3 += C_utils::combination(n, k) * pow((1 - u), (n - k)) * pow(u, k) *

16  (X0+(i/(double)l)*S+(j/(double)m)*T+(k/(double)n)*U);

18  Xnew2 += C_utils::combination(m, j) *  pow((1 - t),(m - j)) *  pow(t,j) *
19               Xnew3;

21  Xnew += C_utils::combination(l, i) *  pow((1 - s),(l - i)) *  pow(s,i) *
             Xnew2;
```

```
15              Xnew3 += C_utils::combination(n, k) * pow((1 - u), (n - k)) * pow(u, k) *
16                              (X0+(i/(double)l)*S+(j/(double)m)*T+(k/(double)n)*U);
17          }
18          Xnew2 += C_utils::combination(m, j) *  pow((1 - t),(m - j)) *  pow(t,j) *
19                          Xnew3;
20      }
21      Xnew += C_utils::combination(l, i) *  pow((1 - s),(l - i)) *  pow(s,i) *   Xnew2;
```

# Operacionalidade



# Resultados

- Validação do algoritmo
- Sem interferência visual

- Trabalhos Correlatos
  - Deformação na animação (Art of Illusion)
  - Conjunto de malhas (Free Form Deformation)
  - Animação (3D Mesh Deformation)

# Conclusão

- Funcionamento do algoritmo
- Visualização
- Queda do FPS
- Modelo humanóide

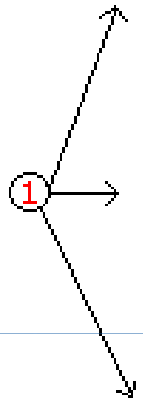
# Extensões

- Implementar outro algoritmo de deformação
- Adicionar a possibilidade de ocorrer a deformação manual da malha do humanóide
- Criar um novo humanóide onde as juntas fiquem visíveis, para realização de novos testes
- Criar uma maior quantidade de animações

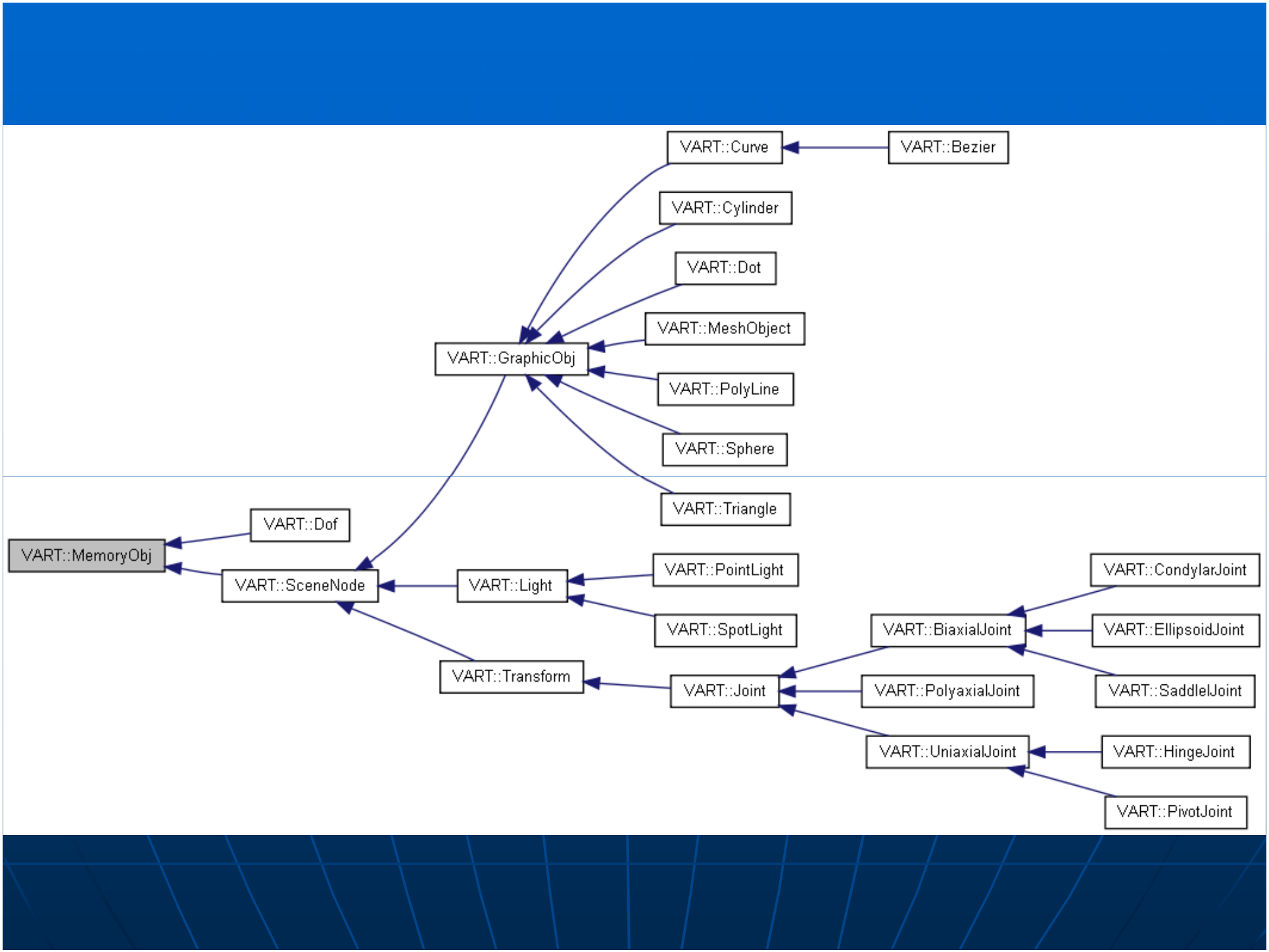
A provação vem, não só para testar o nosso valor, mas para aumentá-lo; o carvalho não é apenas testado, mas enrijecido pelas tempestades.

Lettie Cowman

```
<node>
    <joint description="R_thighJoint" type="polyaxial">
        <dof description="flexR_thighJoint">
            <position x="-0.799850" y="0.155381" z="-0.911524"/>
            <axis x="0.998605" y="0.033865" z="-0.040510"/>
            <range min="-1.570796" max="0.523599" rest="0.750000"/>
        </dof>
        <dof description="adductR_thighJoint">
            <position x="-0.799850" y="0.155381" z="-0.911524"/>
            <axis x="-0.033843" y="0.999426" z="0.001233"/>
            <range min="-0.349066" max="0.523599" rest="0.400000"/>
        </dof>
        <dof description="twistR_thighJoint">
            <position x="-0.799850" y="0.155381" z="-0.911524"/>
            <axis x="-0.040529" y="-0.000139" z="-0.999178"/>
            <range min="-0.698132" max="0.698132" rest="0.500000"/>
        </dof>
    </joint>
    <node>
        <meshobject filename="policial_01.obj"  description="R_thighMesh"
         type = "obj"/>
    </node>
</node>
```




```
<action action_name= "caminha" speed = "1.0000" cycle = "true">
    <joint_movement joint_name = "R_legJoint" duration = "1.6000">
        <interpolation type= "ease-in_ease-out"/>
        <dof_movement dofID = "FLEX" initialTime = "0.0000" finalTime = "0.3250"
                      finalPosition = "0.2270"/>
    </joint_movement>
    <joint_movement joint_name = "R_armJoint" duration = "1.6000">
        <interpolation type= "ease-in_ease-out"/>
        <dof_movement dofID = "ADDUCT" initialTime = "0.0000" finalTime = "0.5000"
                      finalPosition = "0.9100"/>
    </joint_movement>
    <joint_movement joint_name = "R_armJoint" duration = "1.6000">
        <interpolation type= "ease-in_ease-out"/>
        <dof_movement dofID = "TWIST" initialTime = "0.0000" finalTime = "0.5000"
                      finalPosition = "0.6700"/>
    </joint_movement>
</action>
```


VART::Curve
VART::Bezier
VART::Cylinder
VART::Dot
VART::MeshObject
VART::GraphicObj
VART::PolyLine
VART::Sphere
VART::Triangle
VART::Dof
VART::MemoryObj
VART::SceneNode
VART::Light
VART::PointLight
VART::SpotLight
VART::Transform
VART::CondylarJoint
VART::BiaxialJoint
VART::EllipsoidJoint
VART::Joint
VART::PolyaxialJoint
VART::SaddleJoint
VART::UniaxialJoint
VART::HingeJoint
VART::PivotJoint