# O Macaco e a Banana usando Prolog e OpenGL

Daniel Tsuneo Moreira Komido[1]
Diego, Diego Freire Aprigio[1]
Dyonatan Jéferson Celestino[1]
Jaqueline Corrêa Silva de Carvalho[2]
Marcos Alberto de Carvalho[3]

Curso de Ciência da Computação - Universidade José do Rosário Vellano (UNIFENAS)

Rod. MG 179, Km 0 Câmpus Universitário

## Abstract

Computer Graphics encompasses techniques that allow imaging from computer models of objects or events. This allows for process visualization and interactive analysis of the images. This paper presents the development of an application intended to demonstrate the integration of the technologies studied in the course of Computer Graphics and Artificial Intelligence. With the integration of these technologies was developed a software in C # that has rules and functions in Prolog and developed in OpenGL animations.

## Resumo

A Computação Gráfica engloba técnicas que permitem a geração de imagens a partir de modelos computacionais de objetos ou eventos reais. Isso permite a visualização de processos e a análise interativa das imagens. Este trabalho apresenta o desenvolvimento de um aplicativo com intenção de demonstrar a integração das tecnologias estudadas na disciplina de Computação Gráfica e Inteligência Artificial. Com a integração dessas tecnologias foi desenvolvido um software em C# que possui regras e funções em Prolog e animações desenvolvidas em OpenGL.

Palavras Chave: Computação Gráfica. Inteligência Artificial. OpenGL. Prolog.

1 Acadêmicos do 7º. Período do Curso de Ciência da Computação, UNIFENAS, Brasil(2016)

2 Mestre em Engenharia Elétrica pela Universidade Federal de Itajubá, Brasil(2000)

3 Mestrado em Ciência e Tecnologia da Computação pela Universidade Federal de Itajubá, Brasil(2010)

# 1 Introdução

O presente projeto apresenta o desenvolvimento de uma aplicação gráfica para verificar o funcionamento dos fatos e regras criadas em Prolog. Através da ferramenta, é possível posicionar o macaco e a mesa em locais diferentes. Existem regras desenvolvidas usando inteligência artificial onde o macaco se desloca até a mesa e a posiciona debaixo da banana, sobe na mesa e pega a banana.

Foram criados fatos no Prolog, contendo posições para o macaco e para a mesa dentro de uma sala, onde o macaco parte de uma das posições, busca a mesa em outra posição levando-a para o centro, para que possa subir e pegar a banana localizada nesta posição fixa. Foi desenvolvido um software em C# utilizando a biblioteca OpenGL para demonstrar em vídeo os movimentos executados durante o processo.

# 2 Revisão da literatura

## 2.1 OpenGL

Segundo [1], como uma API, OpenGL segue a convenção de chamadas da linguagem C, isto significa que programas escritos em C podem facilmente chamar funções desta API, tanto porque estas foram escritas em C, como porque é fornecido um conjunto de funções C intermediárias que chamam funções escritas em assembler ou outra linguagem.

Apesar de OpenGL ser uma biblioteca de programação "padrão", existem muitas implementações desta biblioteca, por exemplo, para Windows e para Linux. A implementação utilizada no ambiente Linux é a biblioteca Mesa. Também existem implementações para os compiladores Visual C#, Visual C++, Borland C++, Dev-C++, Delphi e Visual Basic. Para obter as bibliotecas e a documentação de cada implementação acesse http://www.opengl.org/.

No caso da implementação da Microsoft, o sistema operacional fornece os arquivos opengl32.dll e glu32.dll, necessários para execução de programas OpenGL. Além disso, são fornecidas com suas ferramentas de programação, como por exemplo com o Microsoft Visual C++, as bibliotecas opengl32.lib (OpenGL) e glu32.lib (GLU - biblioteca de utilitários OpenGL). Assim, para criar programas com ferramentas Microsoft que usem OpenGL, tal como o MS Visual C++ 6.0, é necessário adicionar estas duas bibliotecas à lista de bibliotecas importadas. Protótipos para todas as funções, tipos e macros OpenGL estão (por convenção) no header gl.h. Os protótipos da biblioteca de funções utilitárias estão em um arquivo diferente, glu.h.

## 2.2 C#

A sintaxe do C# é altamente expressiva, mas ela também é simples e fácil de aprender. A sintaxe do C# será instantaneamente reconhecida por qualquer pessoa familiarizada com C, C++ ou Java. Os desenvolvedores que sabem qualquer uma dessas linguagens são geralmente capazes de começar a trabalhar de forma produtiva com C# dentro de um tempo muito curto. A sintaxe do C# simplifica muitas das complexidades do C++ e fornece recursos poderosos, como tipos de valor nulo, enumerações, delegações, expressões lambda e acesso direto a memória, que não são encontrados no Java. O C# suporta métodos e tipos genéricos, que fornecem uma melhor segurança de tipo e desempenho, e iteradores, que permitem

implementadores de coleções de classes para definir comportamentos de iteração personalizados que são simples de usar pelo código cliente.

Como uma linguagem orientada à objetos, o C# suporta os conceitos de encapsulamento, herança e polimorfismo. Todas as variáveis e métodos, incluindo o método principal (Main), o ponto de execução de uma aplicação, são encapsuladas em definições de classes. Uma classe derivada pode herdar diretamente somente de uma classe pai, mas pode herdar de qualquer quantidade de interfaces. Métodos da classe derivada que substituem métodos virtuais de uma classe pai exigem a utilização da palavra-chave **override** como forma de evitar a redefinição acidental. Em C#, uma struct é como uma classe simplificada; é um tipo alocado em pilha que pode implementar interfaces mas não suporta herança.

Além desses princípios básicos orientados a objeto, o C# facilita o desenvolvimento de componentes de software por meio de vários constructos de linguagem inovadores, inclusive os seguintes:

- Assinaturas de métodos encapsulados, chamadas delegates, que permitem notificações de evento de tipo seguro.
- Propriedades, que servem como acessadores para variáveis de membro particular.
- Atributos, que fornecem metadados declarativos sobre tipos em tempo de execução.
- Comentários Embutidos da Documentação XML.
- LINQ (Consulta Integrada à Linguagem) que fornece recursos internos de consulta através de várias de fontes de dados.

Se você tiver que interagir com outros softwares Windows como objetos COM ou DLLs Win32 nativas, você pode fazer isso em C# através de um processo chamado "Interop". O Interop permite aos programas C# fazer quase tudo que uma aplicação nativa C++ pode fazer. O C# suporta ponteiros e o conceito de código inseguro para esses casos em que acesso direto à memória é absolutamente crítico.

O processo de compilação do C# é simples comparado com C++ e mais flexível que em Java. Não há arquivos de cabeçalho separados, e não há a necessidade de que métodos e tipos sejam declarados em uma ordem específica. Um arquivo de código em C# pode definir qualquer número de classes, estruturas, interfaces e eventos [2].

### 2.3 Prolog

SWI-Prolog é uma IDE da linguagem Prolog. Possui uma robusta implementação de multi-threading, tipos de dados estendidos, aritmética ilimitada e representação Unicode do texto a permitir a representação natural de documentos (por exemplo, XML, JSON, RDF) e ao conjunto de dados com outros paradigmas de programação. Sua interface abrangente de baixo nível para C é a base para interfaces de alto nível para C ++, Java (fornecido), C #, Python entre outras. SWI-Prolog é equipado com um extenso servidor web (HTTP) estrutura que pode ser usado tanto para prestação de serviços (REST) e aplicativos de usuário final baseado em HTML5 + CSS + JavaScript. Pengines (motores Prolog) permitem que os clientes possam executar consultas com um programa fornecido pelo cliente em um servidor remoto usando uma API genérica. Tais programas podem ser executados em uma *Sandbox*.

Equipado com interfaces ricas, Prolog é uma linguagem atraente para a realização de aplicações. Sua compilação incremental combinadas com estruturas de dados geralmente

locais e backtrackable (desfazer) permite a aplicação de patches do programa e manter a testá-lo sem reiniciar. O seu paradigma relacional se encaixa bem com dados tabulares (RDBMS), enquanto que a sua força recursiva se encaixa bem com dados da árvore e gráfico. O Prolog pode expressar naturalmente conjuntos de regras simples necessárias para implementar a lógica da aplicação.

SWI-Prolog oferece uma variedade de ambientes de desenvolvimento, a maioria dos que podem ser combinados à vontade. O sistema nativo fornece um editor escrito em Prolog que é semelhante aos Emacs. Ele fornece realce de semântica baseada na análise em tempo real do código pelo sistema Prolog em si. Ferramentas complementares incluem um depurador gráfico, profiler e cross-referencer. Existe um modo para GNU-Emacs e um plug-in Eclipse, ambos os quais podem ser combinados com as ferramentas gráficas nativas. Finalmente, SWISH fornece um ambiente baseado na web com base nos Pengines supracitados que podem tanto ser usados para fornecer acesso à área restrita ou acesso completo após a autenticação. ofertas SWISH edição de múltiplas fontes usando realce de semântica. SWISH pode gerenciar plugins para processamento de dados Prolog utilizando HTML5 + CSS + JavaScript. Isso pode ser usado para processar dados como tabelas, gráficos, tabelas.

SWI-Prolog fornece pacotes de distribuição e mecanismo de instalação. Um pacote é um diretório com algumas convenções organizacionais mínimos e um arquivo de controle que descreve a origem, versão, dependências e suporte de atualização automática. Packs pode ser instalado a partir de um arquivo, o repositório GIT ou URL usando pack_install / 1. Packs são usadas para compartilhar o código na comunidade. [3]

## 2.4 Computação Gráfica

### 2.4.1. Sistema de Coordenadas 3D

Existem dois sistemas de coordenadas 3D: mão esquerda ou mão direita. Em um sistema da mão direita, fazendo-se a mão girar em torno do eixo z, do eixo x positivo para o eixo y positivo, tem-se o polegar apontando na direção positiva do eixo z. O sistema de coordenadas do mundo é definido neste sistema. No sistema da mão esquerda o polegar aponta para o z negativo. Geralmente utiliza-se o sistema da mão direita em coordenadas de visualização.

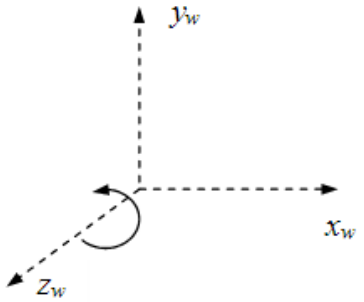


**Figura 1: Sistema da mão direita**

### 2.4.2 Transformações

Um dos mais poderosos recursos da Computação Gráfica é a facilidade com que determinadas alterações podem ser executadas em um objeto. Essa facilidade advém do fato que um modelo digital é um conjunto de números armazenados em estruturas de dados e

disponíveis no computador. É possível aplicar operações matemáticas a estes números. Tais operações são denominadas Transformações.

**Translação**: Pontos do Plano XY podem ser deslocados (trasladados) para novas posições através da adição de valores de translação às coordenadas desses pontos. Considerando os seguintes valores de translação: tx unidades, deslocadas paralelamente ao Eixo X, e ty unidades, deslocadas paralelamente ao Eixo Y, pode-se escrever:

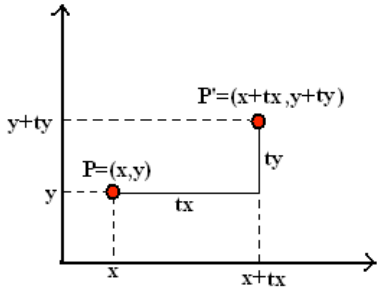


**Figura 2: Translação**

**Escala:** Pontos podem ser submetidos à Transformação de Escala segundo os valores sx e sy (fatores de escala) que significam compressão ou estiramento segundo as direções dos eixos X e Y respectivamente. Os novos pontos são obtidos pela Multiplicação desses valores, segundo as seguintes equações: $x' = x \cdot sx$ e $y' = y \cdot sy$. É importante observar que a escala é relativa à origem do sistema de coordenadas. As proporções do objeto mudam nas direções X e Y quando os fatores de escala são diferentes ($sx \neq sy$).

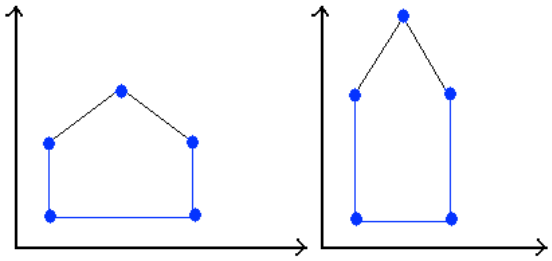

**Figura 3: Escala**

**Rotação:** Pontos podem sofrer rotação de um ângulo φ relativa à origem. Ângulos Positivos são medidos no sentido Anti-Horário, de X para Y e Ângulos Negativos, sentido Horário.

### 2.4.3 Cena 3D

O processo de visualização de uma cena 3D por computador é semelhante ao processo envolvido para se tirar uma fotografia. Deve-se posicionar a câmera no espaço e definir sua orientação. Como a maioria dos monitores (displays) existentes são bidimensionais, devem-se usar processos para converter objetos do espaço tridimensional para uma representação bidimensional. Este processo possui um termo conhecido como 3D pipeline. As etapas deste pipeline são mostradas a seguir, e incluem modelagem, visualização e conversão de diferentes tipos de coordenadas.

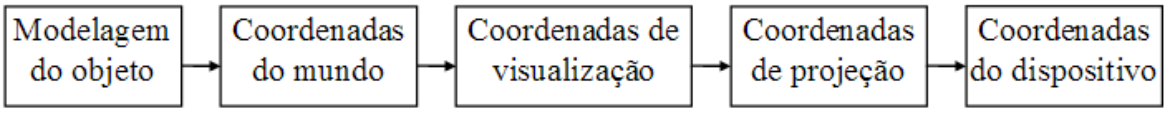


**Figura 4: Etapas de 3D para 2D**

Como primeiro passo da etapa tem-se a definição de como será composto o cenário a ser modelado. Após sua definição, têm-se pontos tridimensionais no sistema de coordenadas do mundo que o definem. Neste sistema de coordenadas é que estão localizados o observador e o plano de projeção da imagem. A próxima etapa do processo é fazer a conversão das coordenadas de visualização para o plano de projeção, que será mapeado para o monitor. É nesta etapa do processo onde é realizada a conversão do espaço tridimensional para o bidimensional (tela do computador). Ao mapear a imagem para a tela, são realizadas operações para recortar partes do cenário que não estão dentro da área da janela de visualização (viewport), para posterior identificação de superfícies visíveis e aplicação de processos de renderização.

Coordenadas de visualização o viewing system é um sistema de coordenadas usado para dar ao observador um maior controle e mobilidade da posição em que se deseja observar um cenário tridimensional. Estes sistemas permitem que o plano de projeção esteja em qualquer lugar do espaço, ou seja, permite que se veja o cenário sob qualquer ângulo. Um exemplo pode ser dado ao se observar um automóvel. Movendo-se ao seu redor, têm-se diferentes ângulos de visão. Abaixando-se se pode vê-lo por baixo, usando-se uma escada pode-se vê-lo de cima; e obtêm-se uma visão interna entrando-se dentro do mesmo. Coordenadas do mundo, coordenadas de visualização, coordenadas de projeção, coordenadas do dispositivo, modelagem do objeto, tudo depende da forma como definimos o ponto onde o observador se encontra, que é chamado de centro de projeção (view point ou view reference point) e, associado a ele, existe o plano de projeção onde será projetada a imagem. Além do view point, pode-se também definir uma orientação espacial do observador, ou seja, para onde ele esta olhando (ângulo de elevação) e qual a inclinação de sua cabeça (angulo de rotação). Os diferentes sistemas de visualização apresentados a seguir nos mostram mais características do que podemos simular em um computador.

A forma mais simples de se visualizar um cenário é localizar o observador sobre o eixo de projeção z, como mostrado figura 5. Para este tipo de sistema, não é necessário nenhum processamento na etapa de geração de coordenadas de visualização do pipeline de visualização, pois o view point, assim como o plano de projeção paralelo ao plano x w y w, já estão localizados sobre o eixo z. Deste modo, necessita-se usar apenas a projeção para fazer a visualização do cenário.

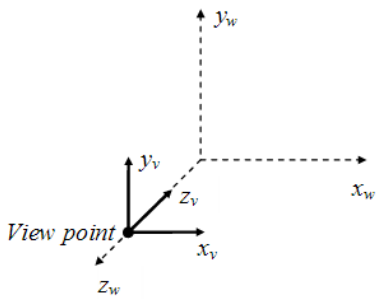


**Figura 5: Visualização da Cena 3D**

## 3 Implementação

Essa etapa pode ser dividida nas regras, componentes da cena e animação. Para o desenvolvimento foram utilizadas as seguintes bibliotecas: Tao.OpenGl, Tao.FreeGlut e SbsSW.SwiPlCs.

### 3.1 Regras

Inicialmente foi criada uma base de fatos em Prolog que determina a posição dos Objetos (macaco, mesa e banana) dentro do ambiente "sala", onde a sala possui as seguintes posições: canto_direito, canto_esquerdo, porta, janela e centro. Depois foram criadas as funções (caminhar, empurrar, move e consegue), com elas o macaco a partir de um ponto dado pela aplicação irá percorrer o trajeto afim de pegar a mesa, empurrá-la até o centro da sala e pegar a banana subindo na mesa.
Fatos criados no Prolog:

```
ponto(na_janela).
ponto(no_centro).
ponto(na_porta).
ponto(no_canto_e).
ponto(no_canto_d).
```

Regras utilizadas para que o macaco consiga pegar a banana:

```
empurrar(P1,P2):-ponto(P1),ponto(P2), P1\=P2.
caminhar(P1,P2):-ponto(P1),ponto(P2), P1\=P2.

move(
   estado(no_centro,acima_mesa,no_centro,nao_tem),
   pegar_banana,
   estado(no_centro,acima_mesa,no_centro,tem)).

move(
   estado(no_centro,no_chao,P,Banana),
   subir,
   estado(no_centro,acima_mesa,P,Banana)).

move(
   estado(P1,no_chao,P1,Banana),
   empurrar(P1,P2),
   estado(P2,no_chao,P2,Banana)).

move(
   estado(P1,no_chao,Mesa,Banana),
   caminhar(P1,P2),
   estado(P2,no_chao,Mesa,Banana)).

move(
   estado(P,acima_mesa,P,Banana),
   descer,
   estado(P,no_chao,P,Banana)).

consegue(estado(_,_,_,tem),[]).
consegue(Estado1,[Movimento|T]):-move(Estado1,Movimento,Estado2),
          consegue(Estado2,T).
```

Chamadas efetuadas no C# do prolog.

```
var consulta = new PlQuery("consegue(estado(PosMacaco,Situacao,PosMesa,nao_tem),L)");
                consulta.Variables["PosMacaco"].Unify(posmacaco);
                consulta.Variables["Situacao"].Unify("no_chao");
                consulta.Variables["PosMesa"].Unify(posmesa);
                consulta.NextSolution();
```

### 3.2 Componentes da cena

Todos os componentes foram definidos usando primitivas em 3D e comandos para criação de sólidos geométricos. Conforme Figura 6 o macaco esta posicionado no centro da sala (fato prolog no_centro), a mesa encontra se no canto direito da sala (fato prolog no_canto_d) e a banana está posicionada no alto, no centro da sala. Neste cenário a regra move(no_centro, no_chao,no_canto_d,nao_tem) retorna uma lista com os passos para que o macaco consiga caminhar até a mesa, empurrá-la até o centro, subir na mesa e pegar a banana.

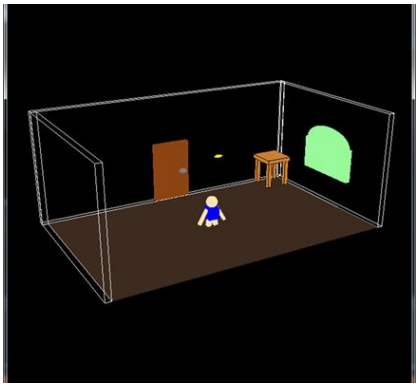

**Figura 6: Visualização da Cena 3D**

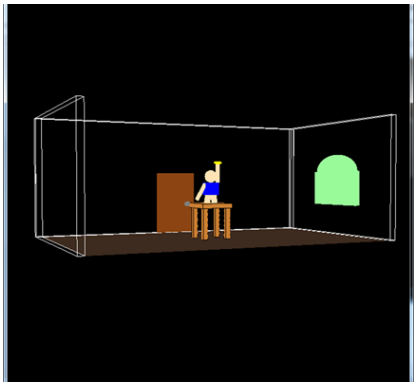

**Figura 7: Macaco em cima da mesa pegando a banana**

### 3.2 Animação

#### 3.2.1 Animação do macaco

Com a lista retornada pelo Prolog é possível verificar a posição do macaco e da mesa no cenário open GL, obtendo a posição da mesa o código C# abaixo faz com que o macaco caminhe até a mesa e então é chamado o método AnimacaoMesa().

```
static void AnimacaoMacaco(int value)
    {
      var valor = lista[i].Split('(');
      valor = valor[1].Split(',');
      int aux = valor[0].Equals("no_canto_e")? -1 : 1;
      switch (valor[1])
      {
        case "no_canto_d)":
          if (mx <= no_canto_d[0] || mz <= no_canto_d[2]-0.2f)
          {
            mx += mx <= no_canto_d[0]?0.001f:0;
            mz+= mz <= no_canto_d[2] - 0.2f ? 0.001f:0;
            Glut.glutPostRedisplay();
            Glut.glutTimerFunc(1, AnimacaoMacaco, 1);
          }
          else
          {
            i++;
            AnimacaoMesa(1);
          }
          break;
       ...
```

### 3.2.1 Animação da mesa

O método AnimacaoMesa do código abaixo é responsável por mover através do Open GL os objetos macaco e mesa até o centro (fato Prolog no_centro), estando já no centro é chamado o método AnimacaoPegaBanana que simplesmente faz o objeto macaco transladar até o topo da mesa e pegar a banana.

```
    static void AnimacaoMesa(int value)
    {
      if (lista[i].Equals("subir"))
      {
        i++;
        AnimacaoPegaBanana(1);
        return;
      }
      var a = lista[i].Split('(');
      a = a[1].Split(',');
      int aux = a[0].Equals("no_canto_e")  ? -1 : 1;
      switch (a[1])
      {
        case "no_centro)":
          if (cx * aux >= no_centro[0] || cz * aux >= no_centro[2])
          {
            cx -= cx * aux >= no_centro[0]? 0.001f*aux:0;
            cz -= cz * aux >= no_centro[0] ? 0.001f*aux : 0f;
            mx -= mx * aux >= no_centro[0]?0.001f *aux:0;
            mz -= mz * aux >= no_centro[2]?0.001f *aux:0;

            Glut.glutPostRedisplay();
            Glut.glutTimerFunc(1, AnimacaoMesa, 1);
          }
          else
          {
            i++;
            AnimacaoPegaBanana(1);
          }
          break;
      }
    }
```

## 4 Conclusão

Conclui-se que a integração das tecnologias C# e Prolog em conjunto com o OpenGl, mostrou-se extremamente eficaz no projeto que visava integra-las, para apresentar ao usuário uma interface. O prolog se demonstrou eficaz no auxílio das regras a serem aplicadas para a iteração dos objetos no cenário, o OpenGl foi utilizado através da biblioteca "TAO", que apresenta todas as funcionalidades do OpenGL necessárias para o desenvolvimento do projeto.

## 5 Referencias

[1] Harb Manssour, Isabel; **Introdução à OpenGL**;
< http://www.inf.pucrs.br/~manssour/OpenGL/Utilizacao.html > Acesso em: 19 de junho de 2016.
[2] **Introduction to the C# Language and the .NET Framework**
<https://msdn.microsoft.com/pt-br/library/z1zx9t92.aspx > Acesso em: 19 de junho de 2016
[3] **SWI-Prolog owl logoSWI-Prolog's features** < http://www.swi-rolog.org/features.html > Acesso em: 19 de junho de 2016
[4] Pozzer, Cesar Tadeu; **Computação Gráfica 3D**
< http://www-usr.inf.ufsm.br/~pozzer/disciplinas/cg_8_3d.pdf > Acesso em: 22 de junho de 2016.